Joseph L. Awange
Béla Paláncz
Robert H. Lewis
Lajos Völgyesi

Mathematical Geosciences

Hybrid Symbolic-Numeric Methods

Springer

Mathematical Geosciences

Joseph L. Awange · Béla Paláncz
Robert H. Lewis · Lajos Völgyesi

Mathematical Geosciences

Hybrid Symbolic-Numeric Methods

Springer

Joseph L. Awange
Spatial Sciences
Curtin University
Perth, WA
Australia

Béla Paláncz
Budapest University of Technology and Economics
Budapest
Hungary

Robert H. Lewis
Fordham University
New York, NY
USA

Lajos Völgyesi
Budapest University of Technology and Economics
Budapest
Hungary

ISBN 978-3-319-67370-7 ISBN 978-3-319-67371-4 (eBook)
https://doi.org/10.1007/978-3-319-67371-4

Library of Congress Control Number: 2017953801

© Springer International Publishing AG 2018
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

Hybrid symbolic-numeric computation (HSNC, for short) is a large and growing area at the boundary of mathematics and computer science, devoted to the study and implementation of methods that mix symbolic with numeric computation.

As the title suggests, this is a book about some of the methods and algorithms that benefit from a mix of symbolic and numeric computation. Three major areas of computation are covered herein. The first part discusses methods for computing all solutions to a system of polynomials. Purely symbolic methods, e.g., via Gröbner bases tend to suffer from algorithmic inefficiencies, and purely numeric methods such as Newton iterations have trouble finding all solutions to such systems. One class of hybrid methods blends numerics into the purely algebraic approach, e.g., computing numeric Gröbner bases or Dixon resultants (the latter being extremely efficient, e.g., for elimination of variables). Another mixes symbolic methods into more numerical approaches, e.g., finding initializations for numeric homotopy tracking to obtain all solutions.

The second part goes into the realm of "soft" optimization methods, including genetic methods, simulated annealing, and particle swarm optimization, among others. These are all popular and heavily used, especially in the context of global optimization. While often considered as "numeric" methods, they benefit from symbolic computation in several ways. One is that implementation is typically straightforward when one has access to a language that supports symbolic computation. Updates of state, e.g., to handle mutations and gene crossover, are easily coded. (Indeed, this sort of thing can be so deceptively simple. baked into the language so to speak, that one hardly realizes symbolic computation is happening.) Among many applications in this part there is, again, that of solving systems of equations. Also covered is mixed-integer programming (wherein some variables are discrete-valued and others continuous). This is a natural area for HSNC since it combines aspects of exact and numeric methods in the handling of both discrete and continuous variables.

The third part delves into data modeling. This begins with use of radial basis functions and proceeds to machine learning, e.g., via support vector machine (SVM) methods. Symbolic regression, a methodology that combines numerics with

evolutionary programming, is also introduced for the purpose of modeling data. Another area seeing recent interest is that of robust optimization and regression, wherein one seeks results that remain relatively stable with respect to perturbations in input or random parameters used in the optimization. Several hybrid methods are presented to address problems in this realm. Stochastic modeling is also discussed. This is yet another area in which hybrid methods are quite useful.

Symbolic computing languages have seen a recent trend toward ever more high level support for various mathematical abstractions. This appears for example in exact symbolic programming involving probability, geometry, tensors, engineering simulation, and many other areas. Under the hood is a considerable amount of HSNC (I write this as one who has been immersed at the R&D end of hybrid computation for two decades.) Naturally, such support makes it all the easier for one to extend hybrid methods; just consider how much less must be built from scratch to support, say, stochastic equation solving, when the language already supports symbolic probability and statistics computations. This book presents to the reader some of the major areas and methods that are being changed, by the authors and others, in furthering this interplay of symbolic and numeric computation. The term hybrid symbolic-numeric computation has been with us for over two decades now. I anticipate the day when it falls into disuse, not because the technology goes out of style, but rather that it is just an integral part of the plumbing of mathematical computation.

Urbana—Champaign
IL, USA
July 2017

Daniel Lichtblau
Ph.D., Mathematics UIUC 1991
Algebra, Applied Mathematics
Wolfram Research, Champaign

Preface

It will surprise no one to hear that digital computers have been used for numerical computations ever since their invention during World War II. Indeed, until around 1990, it was not widely understood that computers could do anything else. For many years, when students of mathematics, engineering, and the sciences used a computer, they wrote a program (typically in Fortran) to implement mathematical algorithms for solving equations in one variable, or systems of linear equations, or differential equations. The input was in so-called "float" numbers with 8–12 significant figures of accuracy. The output was the same type of data, and the program worked entirely with the same type of data. This is *numerical computing*.

By roughly 1990, computer algebra software had become available. Now it was possible to enter data like x^2+3x+2 and receive output like $(x+2)(x+1)$. The computer is doing algebra! More precisely, it is doing *symbolic computing*. The program that accomplishes such computing almost certainly uses no float numbers at all.

What is still not widely understood is that often it is productive to have algorithms that do both kinds of computation. We call these *hybrid symbolic-numeric* methods. Actually, such methods have been considered by some mathematicians and computer scientists since at least 1995 (ISSAC 1995 conference). In this book, the authors provide a much-needed introduction and reference for applied mathematicians, geoscientists, and other users of sophisticated mathematical software.

No mathematics beyond the undergraduate level is needed to read this book, nor does the reader need any pure mathematics background beyond a first course in linear algebra. All methods discussed here are illustrated with copious examples.

A brief list of topics covered:

- Systems of polynomial equations with resultants and Gröbner bases
- Simulated annealing
- Genetic algorithms
- Particle swarm optimization
- Integer programming
- Approximation with radial basis functions

- Support vector machines
- Symbolic regression
- Quantile regression
- Robust regression
- Stochastic modeling
- Parallel computations

Most of the methods discussed in the book will probably be implemented by the reader on a computer algebra system (CAS). The two most fully developed and widely used CAS are *Mathematica* and *Maple*. Some of the polynomial computations here are better done on the specialized system *Fermat*. Other systems worthy of mention are *Singular* and *SageMath*.

The second author is a regular user of *Mathematica*, who carried out the computations, therefore frequent mention is made of *Mathematica* commands. However, this book is not a reference manual for any system, and we have made an effort to keep the specialized commands to a minimum, and to use commands whose syntax makes them as self-explanatory as possible. More complete *Mathematica* programs to implement some of the examples are available online. Similarly, a program written in *Fermat* for the resultant method called Dixon-EDF is available online.

The authors:

July 2017

Joseph L. Awange
Perth, Australia

Béla Paláncz
Budapest, Hungary

Robert H. Lewis
New York, USA

Lajos Völgyesi
Budapest, Hungary

Acknowledgements

The authors wish to express their sincere thanks to *Dr. Daniel Lichtblau* for his helpful comments and for agreeing to write a foreword for our book. *R. Lewis* and *B. Paláncz* highly appreciate and thank *Prof. Jon Kirby* the Head of the Department of Spatial Sciences, Curtin University, Australia for his hospitality and his support of their visiting Curtin. *B. Paláncz* wishes also thank the TIGeR Foundation for the partial support of his staying in Perth. This work was funded partially by OTKA project No. 124286.

Contents

Introduction

Numeric and Symbolic Methods—What are they?

Basically, a *numeric* (*or numerical*) *method* is one that could be done with a simple handheld calculator, using basic arithmetic, square roots, some trigonometry functions, and a few other functions most people learn about in high school. Depending on the task, one may have to press the calculator buttons thousands (or even millions) of times, but theoretically, a person with a calculator and some paper could implement a numerical method. When finished, the paper would be full of arithmetic.

A *symbolic method* involves algebra. It is a method that if a person implemented, would involve algebraic or higher rational thought. A person implementing a symbolic method will rarely need to reach for a calculator. When finished, there may be some numbers, but the paper would be full of variables like *x, y, z*.

Students usually meet the topic of quadratic equations in junior high school. Suppose you wanted to solve the equation $x^2 + 3x - 2 = 0$. With a handheld calculator, one could simply do "intelligent guessing." Let us guess, say, x =1. Plug it in, get a positive result. OK, that is too big. Try $x = 0$; that is too small. Go back and forth; stop when satisfied with the accuracy. It does not take long to get x = 0.56155, which might well be considered accurate enough. Furthermore, it is easy to write a computer program to implement this idea. That is a numeric method.

But wait. There is another answer, which the numeric method missed, namely −3.56155. Even worse, if one were to continue this method on many problems, one would soon notice that some equations do not seem to have solutions, such as $x^2 - 2x + 4 = 0$. A great deal of effort could be expended in arithmetic until finally giving up and finding no solution.

The problem is cured by learning algebra and the symbolic method called the quadratic formula. Given $ax^2 + bx + c = 0$ the solution is $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. It is now immediately obvious why some problems have no solution: it happens precisely when $b^2 - 4ac < 0$.

In the previous example, $x^2 + 3x - 2 = 0$, we see that the two roots are exactly $(-3 \pm \sqrt{17})/2$. There is no approximation whatever. Should a decimal answer correct to, say, 16 digits be desired, that would be trivially obtained on any modern computer.

There is more. Not only does the symbolic method concisely represent all solutions, it invites the question, can we define a new kind of number in which the negative under the square root may be allowed? The symbolic solution has led to a new concept, that of complex numbers!

Symbolic methods may be hard to develop, and they may be difficult for a computer to implement, but they lead to insight.

Fortunately, we are not forced into a strict either/or dichotomy. There are symbolic-numeric methods, hybrids using the strengths of both ideas.

Numeric Solution

In order to further illustrate numeric, symbolic, and symbolic-numeric solutions, let us consider an algebraic system of polynomial equations. For such systems, there may be no solution, one solution, or many solutions. With numerical solutions, one commonly utilizes iterative techniques starting from an initially guessed value. Let us start with a two variable system of two equations $f(x, y) = 0$ and $g(x, y) = 0$,

$$f = (x-2)^2 + (y-3)^2,$$
$$g = \left(x - \tfrac{1}{2}\right)^2 + \left(y - \tfrac{3}{4}\right)^2 - 5.$$

This actual problem has two real solutions, see Fig. 1.

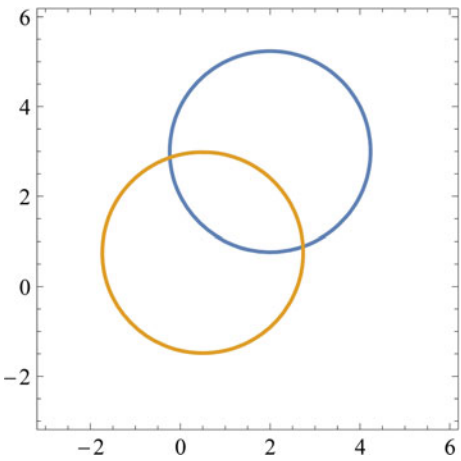


Fig. 1 Geometrical representation of a multivariate polynomial system

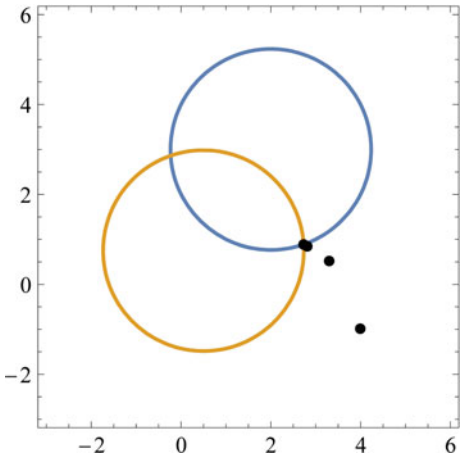


Fig. 2 Local solution with initial guess and iteration steps

A numeric solution starts with the initial value and proceeds step-by-step locally. Depending on the method, we expect to converge to one of the solutions in an efficient manner. Employing the initial value (4, −1) and a multivariate Newton's method, the solution after seven steps is (2.73186, 0.887092). Let us visualize the iteration steps, see Fig. 2.

However, if the initial guess is not proper, for example (0, 0), then, we may have a problem with the convergence since the Jacobian may become singular.

Symbolic Solution

Let us transform the original system into another one, which has the same solutions, but for which variables can be isolated and solved one-by-one. Employing Gröbner basis, we can reduce one of the equations to a univariate polynomial,

$$gry = 2113 - 3120y + 832y^2,$$

$$grxy = -65 + 16x + 24y.$$

First, solving the quadratic equation gry, we have

$$y = \tfrac{1}{104}\left(195 - 2\sqrt{2639}\right),$$

$$y = \tfrac{1}{104}\left(195 + 2\sqrt{2639}\right).$$

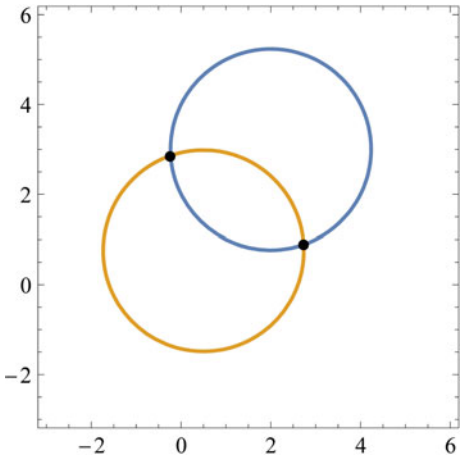


Fig. 3 Global solution—all solutions without initial guess and iteration

Then employing these roots of y, the corresponding values of x can be computed from the second polynomial of the Gröbner basis as

$$x = \tfrac{1}{104}\left(130 + 3\sqrt{2639}\right),$$

$$x = \tfrac{1}{104}\left(130 - 3\sqrt{2639}\right).$$

So, we have computed both solutions with neither guessing nor iteration. In addition, there is no round-off error. Let us visualize the two solutions, see Fig. 3:

Let us summarize the main features of the symbolic and numeric computations:

Numeric computations:

- usually require initial values and iterations. They are sensitive to round-off errors, provide only one local solution,
- can be employed for complex problems, and the computation times are short in general because the steps usually translate directly into computer machine language.

Symbolic computations:

- do not require initial values and iterations. They are not sensitive for round-off errors, and provide all solutions,
- often cannot be employed for complex problems, and the computation time is long in general because the steps usually require computer algebra system software.

Ideally, the best strategy is to divide the algorithm into symbolic and numeric parts in order to utilize the advantages of both techniques. Inevitably, numeric computations will always be used to a certain extent. For example, if polynomial *gry* above had been degree, say, five, then a numeric univariate root solver would have been necessary.

Hybrid (symbolic-numeric) Solution

Sometimes, we can precompute a part of a numerical algorithm in symbolic form. Here is a simple illustrative example.

Consider a third polynomial and add it to our system above:

$$h = \left(x + \tfrac{1}{2}\right)^2 + \left(y - \tfrac{7}{4}\right)^3 - 5.$$

In that case, there is no solution, since there is no common point of the three curves representing the three equations, see Fig. 4.

However, we can look for a solution of this overdetermined system in the minimal least squares sense by using the objective function

$$G = f^2 + g^2 + h^2,$$

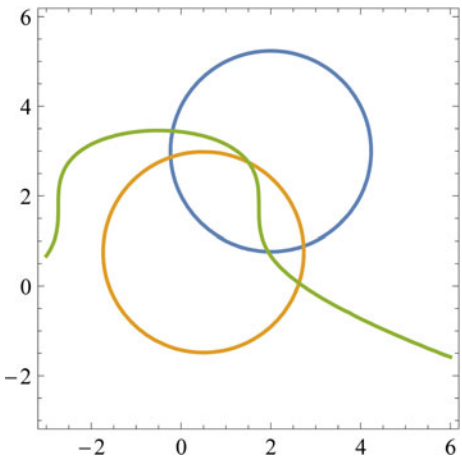


Fig. 4 Now, there is no solution of the overdetermined system

or

$$G = \left(-5+(-2+x)^2+(-3+y)^2\right)^2 + \left(-5+\left(\frac{1}{2}+x\right)^2+\left(-\frac{7}{4}+y\right)^3\right)^2 + \left(-5+\left(-\frac{1}{2}+x\right)^2+\left(-\frac{3}{4}+y\right)^2\right)^2$$

and minimizing it.

Employing Newton's method, we get

$x= 2.28181, y= 0.556578.$

The computation time for this was 0.00181778 s. The solution of the overdetermined system can be seen in Fig. 5.

Here, the gradient vector as well as the Hessian matrix is computed in numerical form in every iteration step. But we can compute the gradient in symbolic form:

$$\text{grad} = \left(\frac{1}{32}\left(2x(173+192(-2+x)x)+216xy-16(41+26x)y^2+64(1+2x)y^3+3(-809+740y)\right)-\frac{137829}{512}+\frac{555x}{8}+\frac{27x^2}{8}+\frac{60527y}{128}-41xy-13x^2y-\frac{6321y^2}{16}+6xy^2+6x^2y^2+\frac{767y^3}{4}-\frac{105y^4}{2}+6y^5\right).$$

Employing this symbolic form the computation time can be reduced. The running time can be further reduced if the Hessian matrix is also computed symbolically,

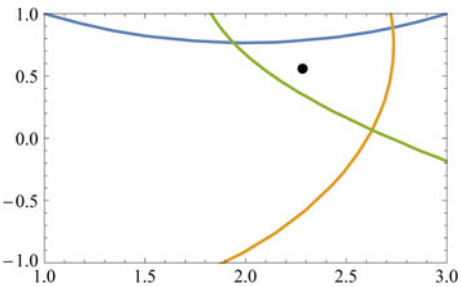


Fig. 5 The solution of the overdetermined system

$$H = \begin{bmatrix} \frac{173}{16} + 12x(-4+3x) + \frac{27y}{4} - 13y^2 + 4y^3 & \frac{555}{8} + y(-41+6y) + x\left(\frac{27}{4} + 2y(-13+6y)\right) \\ \frac{555}{8} + y(-41+6y) + x\left(\frac{27}{4} + 2y(-13+6y)\right) & \frac{60527}{128} - 41x - 13x^2 - \frac{6321y}{8} \\ & + 12xy + 12x^2y + \frac{2301y^2}{4} - 210y^3 + 30y^4 \end{bmatrix}.$$

Now, the computation time is less than half of the original one.

So using symbolic forms, the *computation time can be reduced* considerably. This so-called *hybrid computation* has an additional advantage too, namely the symbolic part of the algorithm does *not generate round-off errors.*

Another approach of applying the hybrid computation is to *merge* symbolic evaluation with numeric algorithm. This technique is illustrated using the following example.

Let us consider a linear, nonautonomous differential equation system of n variables in matrix form:

$$\frac{d}{dx}y(x) = A(x)y(x) + b(x),$$

where A is a matrix of $n \times n$ dimensions, $y(x)$ and $b(x)$ are vectors of n dimensions, and x is a scalar independent variable. In the case of a boundary value problem, the values of some dependent variables are not known at the beginning of the integration interval, at $x = x_a$, but they are given at the end of this interval, at $x = x_b$. The usually employed methods need subsequent integration of the system, because of their trial–error technique or they require solution of a large linear equation system, in the case of discretization methods. The technique is based on the symbolic evaluation of the well-known Runge–Kutta algorithm. This technique needs only one integration of the differential equation system and a solution of the linear equation system representing the boundary conditions at $x = x_b$.

The well-known fourth-order Runge–Kutta method, in our case, can be represented by the following formulas:

$$\begin{aligned} R1_i &= \mathrm{A}(x_i)y(x_i) + b(x_i), \\ R2_i &= \mathrm{A}\left(x_i + \tfrac{h}{2}\right)\left(y(x_i) + \tfrac{R1_i h}{2}\right) + b\left(x_i + \tfrac{h}{2}\right), \\ R3_i &= \mathrm{A}\left(x_i + \tfrac{h}{2}\right)\left(y(x_i) + \tfrac{R2_i h}{2}\right) + b\left(x_i + \tfrac{h}{2}\right), \\ R4_i &= \mathrm{A}(x_i + h)(y(x_i) + R3_i h) + b(x_i + h) \end{aligned}$$

and then the new value of $y(x)$ can be computed as:

$$y_{i+1} = y(x_i) + \frac{(R1_i + 2(R2_i + R3_i) + R4_i)h}{6}.$$

A symbolic system like *Mathematica*, is able to carry out this algorithm not only with numbers but also with symbols. It means that the unknown elements of $y_a = y(x_a)$ can be considered as unknown symbols. These symbols will appear in every evaluated y_i value, as well as in $y_b = y(x_b)$ too.

Let us consider a simple illustrative example. The differential equation is:
$\left(\frac{d^2}{dx^2}y(x)\right) - \left(1 - \frac{x}{5}\right)y(x) = x.$

The given boundary values are:

$$y(1) = 2$$

and

$$y(3) = -1$$

After introducing new variables, we get a first-order system,

$$y1(x) = y(x)$$

and

$$y2(x) = \frac{d}{dx}y(x)$$

the matrix form of the differential equation is:

$$\left[\frac{d}{dx}y1(x),\ \ \frac{d}{dx}y2(x)\right] = \begin{bmatrix} 0 & 1 \\ 1 - x/5 & 0 \end{bmatrix}[y1(x),\ \ y2(x)] + [0,\ \ x].$$

Employing *Mathematica*'s notation:

```
A[x_]:={{0,1},{1-1/5 x,0}};
b[x_]:={0,x};
x0=1;
y0={2.,s}
```

The unknown initial value is s. The order of the system $M = 2$. Let us consider the number of the integration steps as $N = 10$, so the step size is $h = 0.2$.

```
ysol=RKSymbolic[x0,y0,A,b,2,10,0.2];
```

The result is a list of list data structure containing the corresponding (x, y) pairs, where the y values depend on s.

```
ysol[[2]][[1]]
{{1,2.},{1.2,2.05533+0.200987 s},{1.4,2.22611+0.407722
s},
{1.6,2.52165+0.625515 s},
{1.8,2.95394+0.859296s}, {2.,3.53729+1.11368s},
```

```
{2.2,4.28801+1.39298 s},
{2.4,5.22402+1.70123 s},{2.6,6.36438+2.0421 s},
{2.8,7.72874+2.41888 s},{3.,9.33669+2.8343 s}}
```

Consequently, we have got a symbolic result using traditional numerical Runge–Kutta algorithm.

In order to compute the proper value of the unknown initial value, s, the boundary condition can be applied at $x = 3$. In our case $y1(3) = -1$.

```
eq=ysol[[1]][[1]]==-1
9.33669+2.8343 s==-1
```

Let us solve this equation numerically, and assign the solution to the symbol s:

```
sol=Solve[eq,s]
{{s -> -3.647}}
s=s/.sol
{-3.647}
s=s[[1]]
-3.647
```

Then, we get the numerical solution for the problem:

```
ysol[[2]][[1]]
{{1,2.},{1.2,1.32234},{1.4,0.739147},{1.6,0.240397},
{1.8,-0.179911}, {2.,-0.524285},{2.2,-0.792178},
{2.4,-0.980351},{2.6,-1.08317},{2.8,-1.09291},
{3.,-1.}}
```

The truncation error can be decreased by using smaller step size h, and the round-off error can be controlled by the employed number of digits.

Part I
Solution of Nonlinear Systems

Chapter 1
Solution of Algebraic Polynomial Systems

1.1 Zeros of Polynomial Systems

Let us consider the following polynomial

$$p = 2x + x^3y^2 + y^2.$$

The monomials are x^3y^2 with coefficient 1, and x^1y^0 with coefficient *2* and x^0y^2 with coefficient 1. The degree of such a monomial is defined as the sum of the exponents of the variables. For example, the second monomial x^3y^2, has degree 3 + 2 = 5. The *degree of the polynomial* is the maximum degree of its constituent monomials. In this case $\deg\,(p) = \max(1,\ 5,\ 2) = 5$.

Some polynomials contain *parameters* as well as variables. For example, the equation of a circle centered at the origin is $x^2 + y^2 - r^2 = 0$. Only x and y are actual variables; the r is a parameter.

Now consider a polynomial system like

$$g(x, y) = a_1 + a_2x + a_3xy + a_4y,$$

$$h(x, y) = b_1 + b_2x^2y + b_3xy^2.$$

The total degree of the system is defined to be

$$\deg\,(g)\deg\,(h) = 2 \times 3 = 6.$$

Notice that we do not count the parameters in this computation.

Define the *roots* or *zeros* of a polynomial system to be the set of pairs (r, s) such that $g(r, s) = 0$ and $h(r, s) = 0$.

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_1

Bézout's Theorem: Consider two polynomial equations in two unknowns: $g(x, y) = h(x, y) = 0$. If this system has only finitely many zeros $(x, y) \in \mathbb{C}^2$, then the number of zeros is at most deg (g) deg (h). Here deg (g) and deg (h) are the total degree of $g(x, y)$ and $h(x, y)$.

1.2 Resultant Methods

In this section we introduce two different symbolic methods: Sylvester and Dixon resultants see Dickenstein and Emiris (2005). These techniques eliminate variables and yield univariate polynomials, which then can be solved numerically.

1.2.1 Sylvester Resultant

Let us consider the following system (Fig. 1.1)

$$p = xy - 1,$$

$$g = x^2 + y^2 - 4.$$

Since linear systems of equations are well known, let's try to convert this into a useful system of linear equations. With x as the "real" variable and y as a "parameter," consider x^0, x^1, and x^2 as three independent symbols. The two equations in the original system give us two linear equations, and we generate a third by multiplying p by x. This yields

$$M(y)\begin{pmatrix} x^0 \\ x^1 \\ x^2 \end{pmatrix} = \bar{0},$$

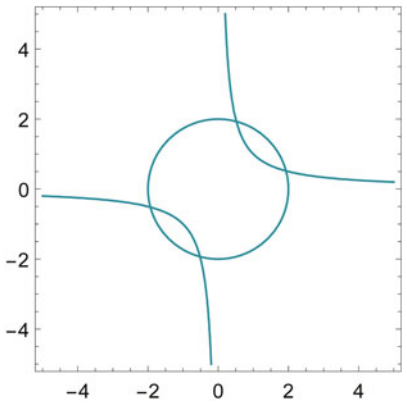


Fig. 1.1 Graphical interpretation of the real roots of the system

where $M(y)$ is

$$\begin{pmatrix} -1 & y & 0 \\ y^2-4 & 0 & 1 \\ 0 & -1 & y \end{pmatrix} \begin{pmatrix} x^0 \\ x^1 \\ x^2 \end{pmatrix} = \bar{0}.$$

Since x^0 is really 1, any solution to this homogeneous system must be nontrivial. Thus

$$\det(M(y)) = -1 + 4y^2 - y^4 = 0.$$

Solving this gives us y; we have *eliminated x.* This function is built into *Mathematica,*

```
Resultant[p,g,x]
1 - 4y^2 + y^4
```

For the other variable

```
Resultant[p,g,y]
1 - 4x^2 + x^4
```

The solutions of these two polynomials are the solutions of the system $\{p(x,y),\ g(x,y)\}$.

```
Roots[ - 1 + 4y^2 - y^4 = = 0,y]
y = √(2 - √3) || y = - √(2 - √3) || y = √(2 + √3) || y = - √(2 + √3)

Roots[1 - 4x^2 + x^4 = = 0,x]
x = √(2 - √3) || x = - √(2 - √3) || x = √(2 + √3) || x = - √(2 + √3)
```

The main handicap of the *Sylvester resultant* is that it can be directly employed only for systems of two polynomials.

1.2.2 Dixon Resultant

Let us introduce a new variable b and define the following polynomial,

$$\delta(x,y,b) = \frac{p(x,y)g(x,b) - p(x,b)g(x,y)}{y-b} = b - 4x + x^3 + y - bxy.$$

(One can show that the numerator above is always divisible by $y - b$). We call this polynomial the *Dixon polynomial.*

It is easy to see that plugging in any common root of $\{p(x,y),\ g(x,y)\}$ forces the Dixon polynomial to be 0, for *any* value of b. The Dixon polynomial can be written as,

$$\delta(x,y,b) = b^0\left(-4x+x^3+y\right)+b^1(1-xy).$$

Then the following homogeneous linear system should have solutions for every b

$$-4x+x^3+y=0,$$

$$1-xy=0$$

or, where x is considered as parameter

$$\begin{pmatrix} -4x+x^3 & 1 \\ 1 & -x \end{pmatrix}\begin{pmatrix} y^0 \\ y^1 \end{pmatrix}=0,$$

therefore

$$\det\left[\begin{pmatrix} -4x+x^3 & 1 \\ 1 & -x \end{pmatrix}\right] = -1+4x^2-x^4,$$

must be zero. The matrix

$$\begin{pmatrix} -4x+x^3 & 1 \\ 1 & -x \end{pmatrix}$$

is called the *Dixon matrix*, and its determinant is called as *Dixon resultant.*

[*Historical note*: the argument above, for two variables, was first used by Bezout.]

Let us employ *Mathematica*

```
≪Resultant'Dixon'
DixonPolynomial[{p,g},{y},{b}]
b - 4x + x^3 + y - bxy
DixonMatrix[{p,g},{y},{b}]//MatrixForm
( - 4x + x^3    1
      1       - x )
DixonResultant[{p,g},{y},{b}]
 - 1 + 4x^2 - x^4
```

Similarly, for the other variable, we get

```
DixonResultant[{p,g},{x},{a}]
 - 1 + 4y^2 - y^4
```

Here a and b are dummy formal variables (symbolic variables), without assigned values.

The Dixon resultant method can be generalized to polynomial systems of more than two polynomials. For example,

```
P = x + y + z;
G = x - 2y + z^3;
S = x^2 - 2y^3 + z;
```

To eliminate variables x and y, we introduce dummy variables X and Y, then

```
DixonResultant[{P,G,S},{x,y},{X,Y}]//Expand
324z + 144z^2 + 24z^3 + 144z^4 - 72z^5 + 36z^6 + 72z^7 - 24z^9
```

or

```
%/12//Expand
27z + 12z^2 + 2z^3 + 12z^4 - 6z^5 + 3z^6 + 6z^7 - 2z^9
```

(% refers to the last previous output.)

Remark 1 For other multivariate resultant methods such as Sturmfels' approach, see Awange and Paláncz (2016).

Remark 2 For three or more variables, the discussion above of the Dixon resultant has been simplified. Sometimes the Dixon matrix is not square, and sometimes when it is square the determinant is identically 0. Then the method would seem to fail. Kappur et al. (1994) showed how to proceed and define the Dixon resultant using maximal minors, see Exercises 1.6.4 Dixon KSY solution.

Remark 3 If the system contains parameters, then the resultant will contain those parameters.

Remark 4 If there are n equations, the Dixon resultant will eliminate $n - 1$ variables.

1.3 Gröbner Basis

This technique introduced by *Buchberger* and named after his Ph.D. supervisor *Gröbner,* is more general and mostly more efficient than the resultant methods, unless parameters are present. To have an idea how this method works, first, let us see how the *greatest common divisor of polynomials* can be defined.

1.3.1 Greatest Common Divisor of Polynomials

Greatest common divisor, GCD, is a familiar concept from arithmetic, as in GCD (12, 30) = 6. The same concept applies to polynomials, of any number of variables.

Let us consider two univariate polynomials $s(x)$ and $v(x)$ with the same variable x

```
s = 8 + 22x + 21x^2 + 8x^3 + x^4;
v = 6 + 11x + 6x^2 + x^3;
```

The greatest common divisor (GCD) of these polynomials

```
gcd = PolynomialGCD[s,v]
2 + 3x + x^2
```

Let us divide $s(x)$ with this GCD

```
{{Cs},Rs} = PolynomialReduce[s,gcd,x]
{{4 + 5x + x^2},0}
```

The remainder is zero and we can check

```
Csgcd
(2 + 3x + x^2)(4 + 5x + x^2)
Expand[%]
8 + 22x + 21x^2 + 8x^3 + x^4
```

Let us carry out these operations with $v(x)$, too

```
{{Cv},Rv} = PolynomialReduce[v,gcd,x]
{{3 + x},0}
```

and

```
Csgcd
(3 + x)(2 + 3x + x^2)
Expand[%]
6 + 11x + 6x^2 + x^3
```

This means that the original polynomials $s(x)$ and $v(x)$ can be expressed as the linear combination of the GCD, like.

$$s(x) = Cs(x)\ \gcd(x) + \ 0\ \gcd(x)$$

and

$$v(x) = \ 0\ \gcd(x) + Cv(x)\ \gcd(x)$$

or

$$\begin{pmatrix} s(x) \\ v(x) \end{pmatrix} = \begin{pmatrix} Cs(x) & 0 \\ 0 & Cv(x) \end{pmatrix} \begin{pmatrix} \gcd(x) \\ \gcd(x) \end{pmatrix}.$$

Since there is only one variable, the roots of $\gcd(x)$, the GCD of these two polynomials $\{s(x),\ v(x)\}$ are the roots of the polynomial system. This important fact is because any one-variable polynomial can be factored over $\mathbb{C}$ into linear pieces. The roots of $\gcd(x) = 2 + 3x + x^2 = 0$ are in Fig. 1.2.
However, we normally have polynomials of two variables (x, y)

```
p
 -1 + xy
g
 -4 + x^2 + y^2
```

In case of more than one variable, the greatest common divisor, though it exists, does not play the role it did in the previous paragraph. That role is filled by the *Gröbner basis* (Buchberger and Winkler 1998).

```
{G1,G2}=GroebnerBasis[{p,g},{x,y}]
{1-4y^2 + y^4,x-4y + y^3}
```

As in the case of univariate polynomial, for the two variables (x, y), the original system can be expressed as the linear combination of the polynomials of the Gröbner basis $\{G1(y),\ G2(x, y)\}$, where the coefficients are also polynomials.

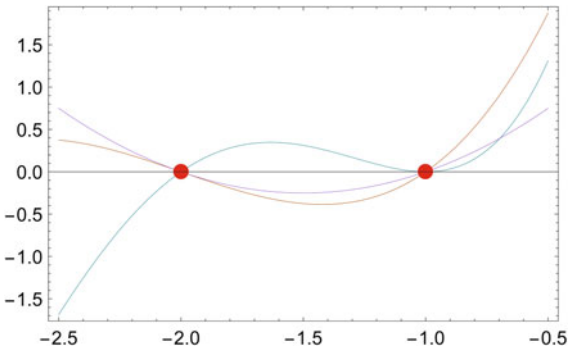


Fig. 1.2 Common roots of polynomials

The coefficients are the remainders.

```
{c1,r1} = PolynomialReduce[p,{G1,G2},{x,y}]
{{ - 1,y},0}
```

Then p(x, y) can be expressed as

```
{ - 1,y}.(G1
          G2)//Simplify
{ - 1 + x y}
```

and

```
{c2,r2} = PolynomialReduce[g,{G1,G2},{x,y}]
{{ - 4 + y^2,x + 4y - y^3},0}
```

then $g(x, y)$

```
{ - 4 + y^2,x + 4y - y^3}.(G1
                           G2)//Simplify
{ - 4 + x^2 + y^2}
```

In matrix form

$$\begin{pmatrix} p(x,y) \\ g(x,y) \end{pmatrix} = \begin{pmatrix} -1 & y \\ -4+y^2 & x+4y-y^3 \end{pmatrix} \begin{pmatrix} G1(y) \\ G2(x,y) \end{pmatrix}$$

or

$$\begin{pmatrix} p(x,y) \\ g(x,y) \end{pmatrix} = \begin{pmatrix} -1 \\ -4+y^2 \end{pmatrix} G1(y) + \begin{pmatrix} y \\ x+4y-y^3 \end{pmatrix} G2(x,y)$$

The roots of the system $\{p(x,y),\ g(x,y)\}$ are the same as the roots of the system $\{G1(y),\ G2(x,y)\}$. Note that this basis consists of special polynomials, since $G1$ (y) is a univariate polynomial!

Generally speaking, the original polynomial system $\{p(x,y),\ g(x,y)\}$ can be expressed as a linear combination of the basis polynomials $\{G1(x,y),\ G2(x,y)\}$.

There are many other basis polynomials too and the set of these basis polynomials is called the *ideal* of the original polynomial. However, the Gröbner basis is a special basis, since one of its polynomials is a univariate one. If the Gröbner basis is 1, the polynomials have no common divisor, consequently they have no common roots.

Remark The theory of Gröbner bases is much more extensive and sophisticated than we can go into here. Our focus is on using Gröbner bases to eliminate variables.

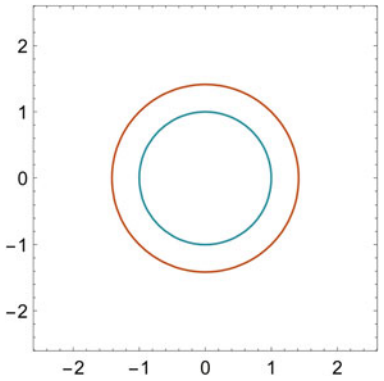


Fig. 1.3 No common roots of polynomials

Let us employ the built-in function for the system {P, S, G} considered in previous Sect. 1.2.2,

```
GroebnerBasis[{P,S,G},{x,y,z}]
{-27z-12z^2-2z^3-12z^4+6z^5-3z^6-6z^7+2z^9,3y+z-z^3,3x+2z+z^3}
```

where the first element of the Gröbner basis is the same provided by the Dixon resultant.

Now, let us compute the Gröbner basis of the following system

```
U=x^2+y^2==1
x^2+y^2-1
V=x^2+y^2==2
x^2+y^2-2
GroebnerBasis[{U,V},{x,y}]
{1}
```

There are no common roots, see Fig. 1.3, however the upper limit of the number of roots is $2 \times 2 = 4$.

1.3.2 Reduced Gröbner Basis

The *Mathematica* built in function can carry out the elimination process too, employing the so called reduced Gröbner Basis.

To get the univariate polynomial of x, we should eliminate y and z,

```
grbx=GroebnerBasis[{P,S,G},{x},{y,z}]
{-27x+18x^2-342x^3+306x^4-186x^5+229x^6-18x^7+12x^8+8x^9}
```

and then for the other variables,

```
grby = GroebnerBasis[{P,S,G},{y},{x,z}]
{-21y - 23y^2 - 30y^3 - 36y^4 - 9y^5 + 6y^6 - 12y^7 + 8y^8}
grbz = GroebnerBasis[{P,S,G},{z},{x,y}]
{-27z - 12z^2 - 2z^3 - 12z^4 + 6z^5 - 3z^6 - 6z^7 + 2z^9}
```

These algebraic methods are very impressive and useful, but they are limited by the size of the system. In general, systems with more than ten unknowns cannot be solved this way due to time and space (RAM) limitations.

1.3.3 Polynomials with Inexact Coefficients

Computing Gröbner bases with inexact coefficients is often desired in industrial applications, but the computation with floating-point numbers is quite unstable if performed naively (Sasaki 2014). The solution methods of the Gröbner basis are very sensitive to round off error, therefore sometimes in case of systems that are over-constrained or have roots with multiplicities, and are given with inexact coefficients, using hybrid symbolic-numeric methods are required (Szanto 2011).

Lichblau (2013) discussed computation of Gröbner bases using approximate arithmetic for coefficients and showed how certain considerations of tolerance, corresponding roughly to accuracy and precision from numeric computation, allow us to obtain good approximate solutions to problems that are overdetermined.

Let us consider the following polynomial system,

```
polys = -4 + x^2 - 1.49071xy + y^2, -8 + x^2 - 0.4xz + z^2,
 -4 + t^2 - 0.894427tx + x^2, -4 + y^2 - 1.49071yz + z^2,
 -8 + t^2 - 0.666667ty + y^2, -4 + t^2 - 0.894427tz + z^2;
```

If we try to find the Gröbner basis, we get the trivial answer `{1.}`, which means there is no relationship between the polynomials.

```
sol = GroebnerBasis[polys,{x,y,z,t}]
{1.}
```

Even employing rationalization of the coefficients will not solve the problem,

```
n = 10;
polysR = Map[Rationalize[#,10^-n]&,polys]
 -4 + x^2 - (149071xy)/100000 + y^2, -8 + x^2 - (2xz)/5 + z^2,
 -4 + t^2 - (216200tx)/241719 + x^2, -4 + y^2 - (149071yz)/100000 + z^2,
 -8 + t^2 - (666503ty)/999754 + y^2, -4 + t^2 - (216200tz)/241719 + z^2
```

```
solR = GroebnerBasis[polysR, {x, y, z, t}]
{1.}
```

However, applying an *approximate hybrid technique*,

```
solA = GroebnerBasis[polys, x, y, z, t, Tolerance → 10^(-3)]
```

yields

```
solt = NSolve[solA[[1]], t]
{{t → -1.00002 - 0.0044912 i}, {t → -1.00002 + 0.0044912 i},
 {t → 1.00002 - 0.0044912 i}, {t → 1.00002 + 0.0044912 i}}
```

Since we are interested in real solutions,

```
Map[Re[#[[2]]]&, Flatten[solt]]
{-1.00002, -1.00002, 1.00002, 1.00002}
```

Then the other variables are

```
solz = NSolve[solA[[2]] /. t → 1.00002, z]
{{z → 2.2361}}

soly = NSolve[solA[[3]] /. t → 1.00002, y]
{{y → 3.00005}}

solx = NSolve[solA[[4]] /. t → 1.00002, x]
{{x → 2.23606}}
```

Let us check our result via least squares technique employing global minimization. Our objective function is

```
G = Total[Map[#^2&, polys]]
(-4 + t^2 - 0.894427 tx + x^2)^2 + (-8 + t^2 - 0.666667 ty + y^2)^2
  + (-4 + x^2 - 1.49071 xy + y^2)^2 + (-4 + t^2 - 0.894427 tz + z^2)^2
  + (-8 + x^2 - 0.4 xz + z^2)^2 + (-4 + y^2 - 1.49071 yz + z^2)^2
```

and

```
NMinimize[G, {x, y, z, t}]
2.10012 × 10^-10, x → 2.23607, y → 3., z → 2.23607, t → 1.0023
```

1.4 Using Dixon-EDF for Symbolic Solution of Polynomial Systems

We have discussed the basic idea of a system of polynomial equations in the Introduction. Earlier in this chapter we introduced the ideas of resultants and Gröbner bases and did some examples. In this section we will show some much more difficult problems that reveal the great power of the Dixon resultant as extended with "Early Detection of Factors", or Dixon-EDF.

As before, we have in each case n equations in n variables $x_1, x_2, \ldots, x_n$ and some parameters. We assume that the system is neither over- nor underdetermined. Usually $3 < n < 15$, though we can work with more variables if the system is sparse enough and does not involve variables with high exponent. In most examples from actual applications, one rarely sees an exponent larger than 2.

Again, by "solve the system" we mean we have eliminated all but one of the variables. We are left with one equation in one variable and the parameters. If desired, numerical values for the parameters can then be substituted, and the variable obtained by one-variable numerical solvers.

The ideas in this section were developed by Lewis (2007, 2008).

1.4.1 Explanation of Dixon-EDF

The basic idea of the Dixon method is to construct a square matrix M whose determinant D is a multiple of the resultant. Usually M is not unique, it is obtained as a maximal minor, in a larger matrix we shall call M^+, and there are usually many maximal minors—any one of which will do. The entries in M are polynomials in parameters. The factors of D that are not the resultant are called the *spurious factors*, and their product is sometimes referred to as the *spurious factor*.

The naive way to proceed is to compute D, factor it, and separate the spurious factor from the actual resultant. But there are problems. On one the hand, the determinant may be so large as for it to be impractical or even impossible to compute; even though the resultant is relatively small, the spurious factor is huge. On the other hand, the determinant may be so large that factoring it is impractical.

Lewis developed three heuristic methods to overcome these problems (2008). The methods were discovered by experimentation and may apply to other resultant formulations, such as the Macaulay. The one that concerns here is called EDF.

The EDF method exploits the observed fact that D usually has many factors. In other words, we try to turn the existence of spurious factors to our advantage. By elementary row and column manipulations (Gaussian elimination) we discover probable factors of D and extract them from $M_0 = M$. This produces a smaller matrix M_1, still with polynomial entries, and a list of discovered numerators and denominators.

Here is very simple example.

$$M_0 = \begin{pmatrix} 9 & 2 \\ 4 & 4 \end{pmatrix} \qquad \text{numerators:} \qquad \text{denominators:}$$

Of course the determinant is trivial, but suppose we wish to keep the arithmetic very simple, and never work with numbers bigger than 9. We factor a 2 out of the second column, then a 2 from the second row. Thus:

$$M_0 = \begin{pmatrix} 9 & 1 \\ 2 & 1 \end{pmatrix} \qquad \text{numerators: } 2,\ 2 \qquad \text{denominators:}$$

We change the second row by subtracting 2/9 of the first:

$$M_0 = \begin{pmatrix} 9 & 1 \\ 0 & 7/9 \end{pmatrix} \qquad \text{numerators: } 2,\ 2 \qquad \text{denominators:}$$

We pull out the denominator 9 from the second row, and factor out 9 from the first column:

$$M_0 = \begin{pmatrix} 1 & 1 \\ 0 & 7 \end{pmatrix} \qquad \text{numerators: } 2,\ 2,\ 9 \qquad \text{denominators: } 9$$

We "clean up" by dividing out the common factor of 9 from the numerator and denominator lists; any 1 that occurs may be erased and the list compacted. Since the first column is canonically simple, we are finished with one step of the algorithm, and have produced a one-smaller M_1 for the next step.

$$M_1 = (7) \qquad \text{numerators: } 2,\ 2 \qquad \text{denominators: } 1$$

The algorithm terminates by pulling out the 7:

$$\text{numerators: } 2,\ 2,\ 7 \qquad \text{denominators: } 1$$

As expected (since the original matrix contained only integers) the denominator list is trivial. The product of all the entries in the numerator list is the determinant, but we never needed to deal with any number larger than 9.

The EPF method is implemented in the computer algebra system *Fermat* by Lewis (2009).

The Dixon resultant is a very attractive tool for solving systems of multivariate polynomial geodetic equations (see Paláncz et al. 2008). Comparing it to other multi-polynomial resultant like Sturmfels's method it has advantages of (i) the small size of the Dixon matrix, (ii) faster computational speed, (iii) being robust.

In the following sections we provide some examples where Dixon EDF method proved to be very effective.

1.4.2 Distance from a Point to a Standard Ellipsoid

Given an ellipsoid $x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0$ and a point (u, v, w), compute the point (x, y, z) on the ellipsoid closest to the point. We have three variables x, y, z. We derive equations using partial derivatives, so we must add two more variables to stand for $\partial z/\partial x$, $\partial z/\partial y$. There are six parameters a, b, c, u, v, w. The new variables representing $\partial z/\partial x$, $\partial z/\partial y$ are artifacts. We don't care about them. We want to know just x, y, z. One advantage of resultants is that you can't tell a Gröbner basis algorithm not to bother with some of the variables (Fig. 1.4).
This is a fairly easy problem. The resultant is degree 6 in x.

With Dixon: 0.038 s, 22 MB RAM with Magma's Gröbner basis, 1 s, 100 MB. (Similar results were obtained with Maple and Mathematica.)

But we can say more. The coefficient of x^6 is

$$b^2c^2 - 2abc^2 + a^2c^2 - 2ab^2c + 4a^2bc - 2a^3c + a^2b^2 - 2a^3b + a^4.$$

This factors into $(a - c)^2(a - b)^2$, so we learn that if $b = a$ or $c = a$ there is a simpler solution. In fact, if $c = a$ it drops to degree 4. As we pointed out in the Introduction, the symbolic method leads to insight!

1.4.3 Distance from a Point to Any 3D Conic

Here is the image for a general ellipsoid, but we could have any 3D conic (Fig. 1.5). Given

$$ax^2 + by^2 + cz^2 + d\,xy + e\,xz + f\,yz + gx + hy + iz + j = 0$$

and point (u, v, w), compute point (x, y, z) with shortest distance. We have again three variables x, y, z, but now 13 parameters $a, b, c, \ldots, u, v, w$. At least one artifact variable must be added.

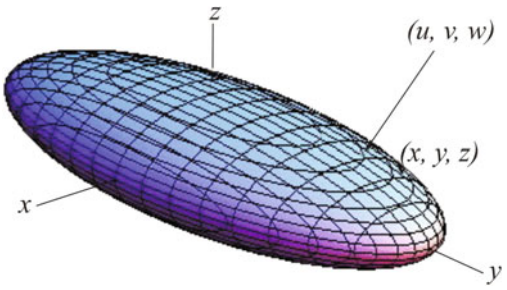


Fig. 1.4 Given u, v, w find x, y, z

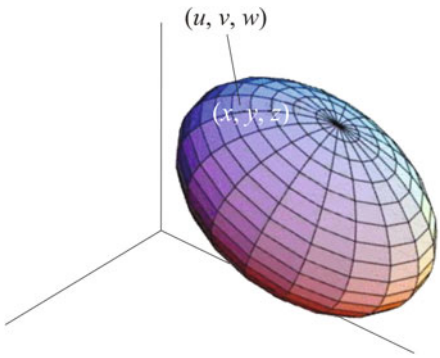


Fig. 1.5 Given u, v, w find x, y, z

This problem is much harder than the previous. With Dixon-EDF: 12 s, 270 MB RAM. The answer has 38,984 terms, degree 6. With Magma: killed after 24 h, 24 GB RAM. With Maple's FGb routine: Success after 5.8 h, 52 GB RAM.

The coefficient of x^6 has two factors, one is $af^2 - def + be^2 + cd^2 - 4abc$. If this were 0, the resultant simplifies.

1.4.4 Pose Estimation

Suppose we have a quadrilateral $ABCE$; it does not have to be planar. The distances between each pair of vertices are known. The object moves. We observe it from point P, noting the angles spanned by each pair of vertices. The classic four point pose problem is to deduce the distances X_1, X_2, X_3, X_4 (Fig. 1.6).

It is easy to derive six equations from the law of cosines:

$$X_1^2 + X_2^2 - X_1 X_2\, r - |AB|^2$$

$$X_1^2 + X_3^2 - X_1 X_3 q - |AC|^2$$

$$X_2^2 + X_3^2 - X_2 X_3\, p - |BC\,|^2$$

$$X_1^2 + X_4^2 - X_1 X_4\, s - |AE\,|^2$$

$$X_4^2 + X_3^2 - X_4 X_3\, t - |CE\,|^2$$

$$X_2^2 + X_4^2 - X_2 X_4\, u - |BE\,|^2$$

r, p, q, s, t, u, are cosines.

There are four variables X_1, X_2, X_3, X_4. The parameters are the lengths of AB, BC, CE, AE, AC, BE, and the six cosines.

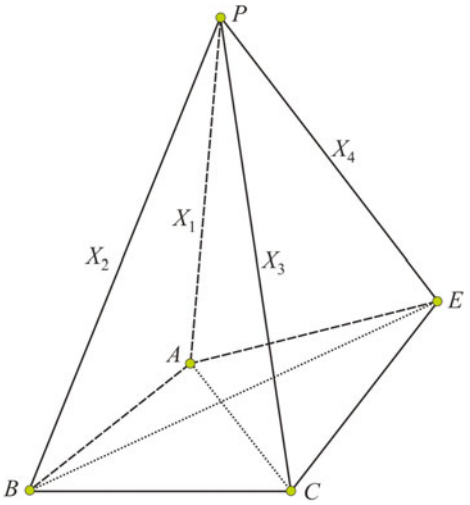


Fig. 1.6 Pose estimation problem

Using any four equations but including at least one diagonal *AC* or *BE* yields an easy system of equations, solvable by many means. Indeed, one could select, say, the first three equations and obtain a complete three variable system; see the exercises at the end of this chapter. But suppose the object could be flexible! Then we must use four equations from only the outside edges; diagonal distances might change. This turns out to be a much harder system to solve and is only doable with Dixon-EDF.

This problem is similar to resection; see Sect. 1.6.3.

1.4.5 How to Run Dixon-EDF

As far as we know, Dixon is implemented only in *Mathematica*, no other large multipurpose CAS. It is a package that must be downloaded and installed. It implements part of the KSY idea, but not EDF.
Dixon-EDF is implemented in *Fermat* as a series of procedures; see Lewis (2009).

1.5 Applications

1.5.1 Common Points of Geometrical Objects

It is well known that the visualization of curves and surfaces is easy and comfortable via parametric explicit equations of the geometrical objects. However, the implicit form of these equations is sometimes needed. For example one would like

to decide whether a point is on a curve or surface or not. Finding the common points of two or more geometrical objects is the generalization of this task. Converting explicit to implicit just means eliminating the parameter.

Application 1 Let us compute the implicit equation of a 2D circle.

The form of the explicit equation with the parameter is,

$$x = \cos(\alpha),$$

$$y = \sin(\alpha)$$

and in addition we know that

$$\sin^2(\alpha) + \cos^2(\alpha) = 1.$$

Solution
Therefore, we have the following system of equations with unknowns (x, y, α)

$$x - \cos(\alpha),$$

$$y - \sin(\alpha),$$

$$-1 + \sin^2(\alpha) + \cos^2(\alpha).$$

and we should eliminate the variable α. Let us compute the Gröbner basis for x and y eliminating α

```
GroebnerBasis[{x - cos[α], y - sin[α], sin[α]^2 + cos[α]^2 - 1},
{x, y}, {α, cos[α], sin[α]}]
{-1 + x^2 + y^2}
```

This elimination could easily be done with the Dixon resultant. Note that we really have four variables x, y, $\cos(\alpha)$, and $\sin(\alpha)$ and three equations. With three equations we can eliminate any two variables, so we choose the latter two.

Application 2 Now let us compute the common points of a cardioid and a circle.

The parametric equation of the cardioid is, see Fig. 1.7,

$$x = 2\,(1 + \cos(t))\,\cos(\mathrm{t}),$$

$$y = 2\,(1 + \cos(t))\,\sin(\mathrm{t}).$$

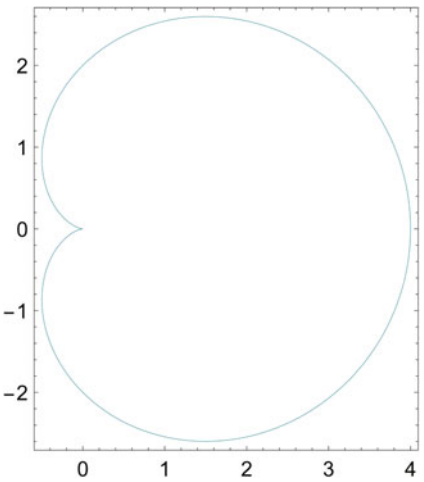


Fig. 1.7 A cardioid curve

Solution

As a first step, we compute the implicit form of the equation of the cardioid.

$$x - 2\ \cos(t) - 2\ \cos^2(t),$$

$$y - 2\ \sin(t) - 2\ \cos(t)\ \sin(\mathrm{t}),$$

$$1 + 2\ \cos^2(t) + \sin^2(t).$$

The Gröbner basis of the system is,

$$\left\{-4x^3 + x^4 - 4y^2 - 4xy^2 + 2x^2y^2 + y^4\right\}.$$

Now let us consider the following circle,

$$x^2 + y^2 - 2 = 0.$$

Then, the two geometrical objects together are as shown in Fig. 1.8.

The next step is the computation of the common points employing these implicit equations. Then the following system should be solved

```
g1 = - 4x^3 + x^4 - 4y^2 - 4xy^2 + 2x^2y^2 + y^4
- 4x^3 + x^4 - 4y^2 - 4xy^2 + 2x^2y^2 + y^4
g2 = x^2 + y^2 - 2
- 2 + x^2 + y^2
```

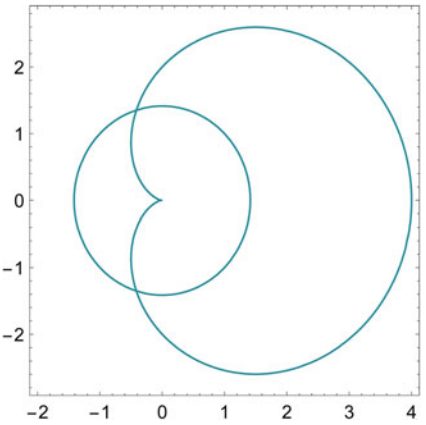


Fig. 1.8 The two geometrical objects

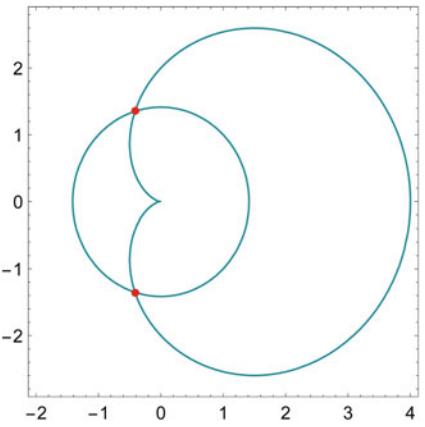


Fig. 1.9 The common points of the two geometrical objects

The reduced Gröbner basis for the x coordinate is given as

```
GroebnerBasis[{g1,g2},{x},{y}]
{-1-2x+x^2}
```

Similarly for the y coordinate

```
GroebnerBasis[{g1,g2},{y},{x}]
{-7+2y^2+y^4}
```

or with the built-in function `Solve`

```
solp={x,y}/.Solve[{g1==0,g2==0},{x,y}]//Simplify
{{1-√2,-√(-1+2√2)},{1-√2,√(-1+2√2)},
{1+√2,-i√(1+2√2)},{1+√2,i√(1+2√2)}}
```

There are only two real solutions! Let us visualize the common points, see Fig. 1.9.

To solve this problem with the Dixon resultant, just take the three equations defining the cardioid and the one defining the circle. First, eliminate the three variables *y*, *cos*(*t*), and *sin*(*t*). That yields the equation for *x*, `- 1 - 2x + x^2`. Then repeat, eliminating *y*, *cos(t),* and *sin(t)* to get the equation for *y*.

1.5.2 Nonlinear Heat Transfer

The nonlinear dimensionless equation of the steady state heat transfer in 1D is,

$$\frac{d}{dx}\left(\lambda(\theta)\frac{d\theta}{dx}\right) = 0.$$

The boundary conditions are,

$$\theta(0) = 0 \quad \text{and} \quad \theta(1) = 1.$$

The heat transfer coefficient depending on the temperature,

$$\lambda(\theta) = 1 + k\theta.$$

Let us approximate the temperature profile with the following polynomial,

$$\theta(x) = x + c_1\left(x^2 - x\right) + c_2\left(x^3 - x\right),$$

which satisfies the boundary conditions. Let us compute the c_i coefficients.

Solution
Substituting the temperature profile into the differential equation, we get

$$\begin{aligned} eq = {} & k + 2c_1 - 2kc_1 + 6kxc_1 + kc_1^2 - 6kxc_1^2 + 6kx^2c_1^2 - 2kc_2 + 6xc_2 + 12kx^2c_2 \\ & + 2kc_1c_2 - 6kxc_1c_2 - 12kx^2c_1c_2 + 20kx^3c_1c_2 + kc_2^2 - 12kx^2c_2^2 + 15kx^4c_2^2. \end{aligned}$$

Using the global integral method, the square of the integral should be minimized,

$$\begin{aligned} r = \int_0^1 eq^2 \mathrm{d}x = {} & k^2 + 4kc_1 + 2k^2c_1 + 4c_1^2 + 4kc_1^2 + 4k^2c_1^2 + \frac{1}{5}k^2c_1^4 + 6kc_2 + 4k^2c_2 \\ & + 12c_1c_2 + 20kc_1c_2 + 16k^2c_1c_2 + \frac{4}{5}k^2c_1^2c_2 + \frac{6}{5}k^2c_1^3c_2 + 12c_2^2 + 24kc_2^2 \\ & + \frac{84}{5}k^2c_2^2 + \frac{12}{5}k^2c_1c_2^2 + \frac{20}{7}k^2c_1^2c_2^2 + \frac{64}{35}k^2c_2^3 + \frac{111}{35}k^2c_1c_2^3 + \frac{48}{35}k^2c_2^4. \end{aligned}$$

Employing the necessary conditions of the minimum, differentiate the integral, we get an algebraic polynomial system for the unknown coefficients

```
eq1 = D[r, c1]
```

$$4k + 2k^2 + 8c_1 + 8kc_1 + 8k^2c_1 + \frac{4}{5}k^2c_1^3 + 12c_2 + 20kc_2 + 16k^2c_2 + \frac{8}{5}k^2c_1c_2 + \frac{18}{5}k^2c_1^2c_2 + \frac{12}{5}k^2c_2^2 + \frac{40}{7}k^2c_1c_2^2 + \frac{111}{35}k^2c_2^3$$

```
eq2 = D[r, c2]
```

$$6k + 4k^2 + 12c_1 + 20kc_1 + 16k^2c_1 + \frac{4}{5}k^2c_1^2 + \frac{6}{5}k^2c_1^3 + 24c_2 + 48kc_2 + \frac{168k^2c_2}{5} + \frac{24}{5}k^2c_1c_2 + \frac{40}{7}k^2c_1^2c_2 + \frac{192}{35}k^2c_2^2 + \frac{333}{35}k^2c_1c_2^2 + \frac{192}{35}k^2c_2^3$$

The Gröbner basis for c_1,

```
GroebnerBasis[{eq1,eq2},{c1},{c2}]
```

$$\{2222640000k + 25041744000k^2 + 106983636480k^3 + 216207482400k^4 + 217869466458k^5 + 105383544084k^6 + 21747027960k^7 + 982690800k^8 + 4445280000c_1 + 45638208000kc_1 + 172774344960k^2c_1 + 305473573440k^3c_1 + 294315313236k^4c_1 + 170466205476k^5c_1 + 78560129424k^6c_1 + 27948563160k^7c_1 + 4880962800k^8c_1 + 4834771200k^2c_1^2 + 14644375200k^3c_1^2 + 3743455968k^4c_1^2 - 11828581632k^5c_1^2 - 15676868844k^6c_1^2 - 6328977648k^7c_1^2 - 849050160k^8c_1^2 + 398664000k^2c_1^3 - 377496000k^3c_1^3 + 1763997312k^4c_1^3 + 2443968240k^5c_1^3 + 2017898358k^6c_1^3 + 441062580k^7c_1^3 + 61164425k^8c_{10}^3 + 52698240k^4c_1^4 + 370528368k^5c_1^4 + 23358168k^6c_1^4 + 197374240k^7c_1^4 + 18557000k^8c_1^4 - 4040400k^4c_1^5 - 27938400k^5c_1^5 - 12698784k^6c_1^5 - 6985752k^7c_1^5 + 4109544k^8c_1^5 - 1465920k^6c_1^6 - 716616k^7c_1^6 - 872028k^8c_1^6 + 55600k^6c_1^7 + 18640k^7c_1^7 + 58568k^8c_1^7 - 3120k^8c_1^8 + 100k^8c_1^9\}$$

and for c_2, we get similar polynomial.

```
GroebnerBasis[{eq1,eq2},{c2},{c1}]
{-709927680k^2 - 1419855360k^3 - 904619968k^4 - 194692288k^5
    + 4100908k^6 + 1419855360c2 + 4259566080kc2 + 5168334976k^2c2
    + 3237393152k^3c2 + 1556328200k^4c2 + 647559304k^5c2
    + 151166960k^6c2 - 92198400c2^2 - 276595200kc2^2 + 220266368k^2c2^2
    + 901524736k^3c2^2 + 983883152k^4c2^2 + 487021584k^5c2^2
    + 107131248k^6c2^2 + 16464000c2^3 + 49392000kc2^3 + 253787072k^2c2^3
    + 425254144k^3c2^3 + 384371484k^4c2^3 + 179976412k^5c2^3 + 43090110k^6c2^3
    - 6679680k^2c2^4 - 13359360k^3c2^4 - 604072k^4c2^4 + 6075608k^5c2^4
    + 5758480k^6c2^4 - 1117200k^2c2^5 - 2234400k^3c2^5 + 811440k^4c2^5
    + 1928640k^5c2^5 + 1604652k^6c2^5 + 312480k^4c2^6 + 312480k^5c2^6
    + 267470k^6c2^6 + 16800k^4c2^7 + 16800k^5c2^7 + 24612k^6c2^7 + 1560k^6c2^8
    + 75k^6c2^9}
```

From a practical point of view, it is more convenient to employ numerical Gröbner basis function, as in *Mathematica* using $k = 1$,

```
sol = NSolve[{eq1,eq2}/.k → 1,{c1, c2},Reals]//Flatten
{c1 → -0.6251338312334316, c2 → 0.19045444692157196}
```

Then the temperature profile is

```
T = θ/.sol
x - 0.625134 (-x + x^2) + 0.190454 (-x + x^3)
```

Figure 1.10 shows the dimensionless temperature profile for $k = 1$,

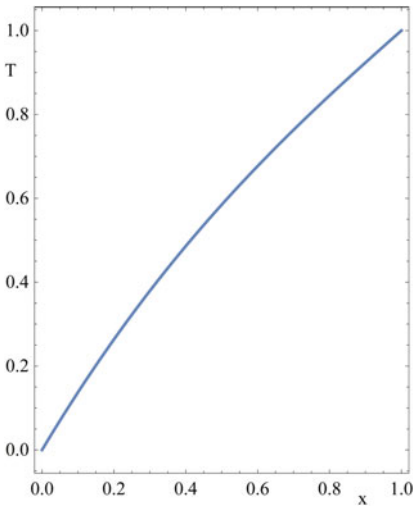


Fig. 1.10 The dimensionless temperature profile in case of $k = 1$

The general function for any $k = \kappa$ can be written as,

```
Ω[κ_] := θ/.(NSolve[{eq1,eq2}/.k
  → κ,{c1,c2},Reals]//Flatten)
```

Let us test this function for $k = 1$

```
Ω[1]
x - 0.625134(-x + x^2) + 0.190454(-x + x^3)
```

We utilized the common capability of the Computer Algebra System (CAS) type language providing symbolic computation as well as any size of digits in order to reduce round-off error.

One can realize that this example is a nice illustration of the hybrid computation, since our function is computed partly in numerical and partly in symbolic way.

1.5.3 Helmert Transformation

Let us consider a 2D Helmert transformation with parameters α and β,

$$\begin{pmatrix} X \\ Y \end{pmatrix} = s \begin{pmatrix} \cos(\Omega) & -\sin(\Omega) \\ \sin(\Omega) & \cos(\Omega) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

We have three control points in both systems, namely (Table 1.1).

Assuming that these values have errors in both systems (EIV model) let us consider the adjustments as Δx_i and $\Delta X_i \;\; i = 1, 2, 3$.

In order compute these adjustments the following minimization problem should be solved,

$$F = \sum_{i=1}^{3} \left(\Delta x_i^2 + \Delta X_i^2 \right)$$

with the constraints,

Table 1.1 Numerical data for the 2D Helmert transformation problem

i	x_i	y_i	X_i	Y_i
1	0.0	1.0	−2.1	1.1
2	1.0	0.0	1.0	2.0
3	1.0	1.0	−0.9	2.8

$$
\begin{aligned}
eq_1 &= \alpha(x_1 + \Delta x_1) - \beta y_1 - (X_1 + \Delta X_1),\\
eq_2 &= \beta(x_1 + \Delta x_1) + \alpha y_1 - Y_1,\\
eq_3 &= \alpha(x_2 + \Delta x_2) - \beta y_2 - (X_2 + \Delta X_2),\\
eq_4 &= \beta(x_2 + \Delta x_2) + \alpha y_2 - Y_2,\\
eq_5 &= \alpha(x_3 + \Delta x_3) - \beta y_3 - (X_3 + \Delta X_3),\\
eq_6 &= \beta(x_3 + \Delta x_3) + \alpha y_3 - Y_3.
\end{aligned}
$$

To transform this problem into a minimization without constraints, let us employ Lagrange-multipliers,

$$
\begin{aligned}
G = F + \sum_{i=1}^{6} \lambda_i\, eq_i &= \Delta x_1^2 + \Delta x_2^2 + \Delta x_3^2 + \Delta X_1^2 + \Delta X_2^2 + \Delta X_3^2\\
&+ (-X_1 - \beta y_1 + \alpha(x_1 + \Delta x_1) - \Delta X_1)\lambda_1 + (\alpha y_1 - Y_1 + \beta(x_1 + \Delta x_1))\lambda_2\\
&+ (-X_2 - \beta y_2 + \alpha(x_2 + \Delta x_2) - \Delta X_2)\lambda_3 + (\alpha y_2 - Y_2 + \beta(x_2 + \Delta x_2))\lambda_4\\
&+ (-X_3 - \beta y_3 + \alpha(x_3 + \Delta x_3) - \Delta X_3)\lambda_5 + (\alpha y_3 - Y_3 + \beta(x_3 + \Delta x_3))\lambda_6\ .
\end{aligned}
$$

Using the necessary condition, after differentiating the objective, we get the following algebraic polynomial system for the unknowns $\{\Delta x_1, \Delta X_1, \Delta x_2, \Delta X_2, \Delta x_3, \Delta X_3, \alpha, \beta, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6\}$

$$2\Delta x_1 + \alpha\lambda_1 + \beta\lambda_2,$$

$$2\Delta X_1 - \lambda_1,$$

$$2\Delta x_2 + \alpha\lambda_3 + \beta\lambda_4,$$

$$2\Delta X_2 - \lambda_3,$$

$$2\Delta x_3 + \alpha\lambda_5 + \beta\lambda_6,$$

$$2\Delta X_3 - \lambda_5,$$

$$x_1\lambda_1 + \Delta x_1\lambda_1 + y_1\lambda_2 + x_2\lambda_3 + \Delta x_2\lambda_3 + y_2\lambda_4 + x_3\lambda_5 + \Delta x_3\lambda_5 + y_3\lambda_6,$$

$$-y_1\lambda_1 + x_1\lambda_2 + \Delta x_1\lambda_2 - y_2\lambda_3 + x_2\lambda_4 + \Delta x_2\lambda_4 - y_3\lambda_5 + x_3\lambda_6 + \Delta x_3\lambda_6,$$

$$\alpha x_1 - X_1 - \beta y_1 + \alpha\Delta x_1 - \Delta X_1,$$

$$\beta x_1 + \alpha y_1 - Y_1 + \beta\Delta x_1,$$

$$\alpha x_2 - X_2 - \beta y_2 + \alpha\Delta x_2 - \Delta X_2,$$

$$\beta x_2 + \alpha y_2 - Y_2 + \beta \Delta x_2,$$

$$\alpha x_3 - X_3 - \beta y_3 + \alpha \Delta x_3 - \Delta X_3,$$

$$\beta x_3 + \alpha y_3 - Y_3 + \beta \Delta x_3.$$

Substituting the numerical values for $\{x_i, y_i\}$ and $\{X_i, Y_i\}$ from Table 1.1, we get 14 polynomials of the Gröbner basis of the problem. The rows of Table 1.2 show the exponents of the unknown variables in the different polynomials,
The first base is a degree six polynomial for λ_6,

$$\begin{aligned} &-763094995516248252876710834 0 \\ &+429598392278896826677930481 33\lambda_6 - 489181086373271123938589713 61\lambda_6^2 \\ &+104610954860700279913881577 80\lambda_6^3 + 104018749323715741160794050 00\lambda_6^4 \\ &-382929968026648328876789062 5\lambda_6^5 + 349089071788949996689453125\lambda_6^6 \end{aligned}$$

Which has two real solutions: $\{\lambda_6 \to -2.14502, \quad \lambda_6 \to 0.238268\}$. We consider the positive solution. (The reason will be given later.) Then the solutions can be obtained with successive elimination from the other bases,

$$\begin{aligned} &\{\Delta x_1 \to 0.021889, \quad \Delta X_1 \to 0.165307, \quad \Delta x_2 \to 0.021537, \quad \Delta X_2 \to 0.079912, \\ &\quad \Delta x_3 \to -0.109804, \quad \Delta X_3 \to -0.116768, \quad \alpha \to 1.057144, \quad \beta \to 1.957834\}. \end{aligned}$$

Table 1.2 Numerical data for the 2D Helmert transformation problem

Δx_1	ΔX_1	Δx_2	ΔX_2	Δx_3	ΔX_3	α	β	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
0	0	0	0	0	0	0	0	0	0	0	0	0	6
0	0	0	0	0	0	0	0	0	0	0	0	1	5
0	0	0	0	0	0	0	0	0	0	0	1	0	5
0	0	0	0	0	0	0	0	0	0	1	0	0	5
0	0	0	0	0	0	0	0	0	1	0	0	0	5
0	0	0	0	0	0	0	0	1	0	0	0	0	5
0	0	0	0	0	0	0	1	0	0	0	0	0	5
0	0	0	0	0	0	1	0	0	0	0	0	0	5
0	0	0	0	0	1	0	0	0	0	0	0	0	5
0	0	0	0	1	0	0	0	0	0	0	0	0	5
0	0	0	1	0	0	0	0	0	0	0	0	0	5
0	0	1	0	0	0	0	0	0	0	0	0	0	5
0	1	0	0	0	0	0	0	0	0	0	0	0	5
1	0	0	0	0	0	0	0	0	0	0	0	0	5

We could solve the problem via direct minimization, too. Employing global minimization method, we can get the same solution. The minimization problem has two local minimums and negative λ_6 refers to the other local minimum, which is not the global one.

1.6 Exercises

1.6.1 Solving a System with Different Techniques

Let us consider the following system,

$$f(x, y, z) = xyz - 1,$$

$$g(x, y, z) = x^2 + 2y^2 + 4z^2 - 7,$$

$$h(x, y, z) = 2x^2 + y^3 + 6z - 7.$$

We do not know approximate solutions, therefore we have no idea which initial values would be proper to start with in case of numerical (iterative) solutions.

Problem

(a) Estimate the number of common roots
(b) Find common roots via *Sylvester* resultant
(c) Find common roots via *Dixon* resultant
(d) Find the univariate polynomials for the unknowns (x, y, z) via *Gröbner basis*
(e) Compute the roots of these polynomials
(f) Carry out the computation with built-in function `NSolve`
(g) Employ high precision computation.

Solution

Considering the degree of the polynomials of the system, the total degree of the system is

$$d = 3 \times 2 \times 3 = 18,$$

Therefore the upper limit of the number of the common roots is 18.

Solution via *Sylvester resultant*

```
f = x y z - 1
-1 + x y z

g = x^2 + 2y^2 + 4z^2 - 7
-7 + x^2 + 2y^2 + 4z^2
```

```
h = 2x^2 + y^3 + 6z - 7
 - 7 + 2x^2 + y^3 + 6z
```

Eliminate z from $f(x, y, z)$ and $g(x, y, z)$

```
fg = Resultant[f,g,z]
4 - 7x^2y^2 + x^4y^2 + 2x^2y^2
```

Eliminate z from $g(x, y, z)$ and $h(x, y, z)$

```
gh = Resultant[g,h,z]
 - 56 - 76x^2 + 16x^4 + 72y^2 - 56y^3 + 16x^2y^3 + 4y^6
```

Eliminate y from $fg(x, y)$ and $gh(x, y)$

```
fggh = Resultant[fg,gh,y]
1048576 - 37748736x^2 + 1241776128x^4 - 5052301312x^6
 + 7358103552x^8 + 11476934656x^10 - 13391571968x^12
 - 6389601280x^14 + 5032881408x^16 - 314205696x^18
 + 369083136x^20 - 382296064x^22 + 92680960x^24
 - 3911168x^26 - 927488x^28 + 51200x^30 + 4096x^32
```

The real roots

```
solx = NSolve[fggh = = 0,x,Reals]//Simplify
{{x → - 1.754},{x → - 1.},{x → 1.},{x → 1.754}}
```

Solution Dixon resultant (*Mathematica*)

```
dr = DixonResultant[{f,g,h},{y},{Y,Z}]
   - 256(64x - 768x^2 + 3456x^3 - 4560x^4 - 696x^5 + 4032x^6 - 2534x^7
   + 5760x^8 + 1203x^9 - 2256x^10 + 1409x^11 + 192x^12
   - 531x^13 + 25x^15 + 4x^17)

solx = NSolve[dr[[2]] = = 0,x,Reals]//Simplify
{{x → - 1.754},{x → - 1.},{x → 0}}
```

Employing reduced Gröbner basis for x

```
grx = GroebnerBasis[{f,g,h},{x,y,z},{y,z}]
{64 - 768x + 3456x^2 - 4560x^3 - 696x^4 + 4032x^5
 - 2534x^6 + 5760x^7 + 1203x^8 - 2256x^9 + 1409x^10
 + 192x^11 - 531x^12 + 25x^14 + 4x^16}

solx = NSolve[grx = = 0,x,Reals]//Simplify
{{x → - 1.754},{x → - 1.}}
```

Employing built-in function `NSolve` for the system. The number of solutions

```
solT = Length[NSolve[{f,g,h},{x,y,z}]]
16
```

The real solutions

```
solR = NSolve[{f,g,h},{x,y,z},Reals]
{{x → - 1.754,y → - 1.24075,z → 0.45950},
  {x → - 1.,y → - 1.,z → 1.}}
```

Checking the solutions

```
Map[{f,g,h}/.#&,solR]
{{ - 1.40998 × 10^-14,1.62981 × 10^-13,3.61933 × 10^-14},
 {7.99361 × 10^-15,7.72715 × 10^-14,2.04281 × 10^-14}}
```

Employing higher (any) precision

```
solR30 = NSolve[{f,g,h},{x,y,z},Reals,WorkingPrecision → 30]


{{x → - 1.754004753619046218477869411 09,
 y → - 1.240746501350929911061218001 26,
 z → 0.459500697239036497067167646364},
{x → - 1.000000000000000000000000000 00,
 y → - 1.000000000000000000000000000 00,
 z → 1.000000000000000000000000000 00}}
```

and checking the solution via back substitution,

```
Map[{f,g,h}/.#&,solR30]
{{0. × 10^-30,0. × 10^-29,0. × 10^-29},{0. × 10^-30,0. × 10^-29,0. × 10^-29}}
```

1.6.2 Planar Ranging

Problem
Let us assume that the distances ($t1$, $t2$) of an unknown point (xu, yu) from two different locations ($x1$, $y1$) and ($x2$, $y2$) are known, to compute the unknown coordinates.

Solution
The distance equations for the two known points are,

```
eq1 = (x1 - xu)^2 + (y1 - yu)^2 - t1^2;
eq2 = (x2 - xu)^2 + (y2 - yu)^2 - t2^2;
```

Let the coordinate values of the two points be,

```
xm = {48177.62, 49600.15};
ym = {6531.28, 7185.19};
```

and the measured distances be,

```
tm = {611.023, 1529.482};
```

Applying the *Sylvester resultant*, we eliminate yu

```
Eqxu = Resultant[eq1, eq2, yu]
t1^4 - 2t1^2t2^2 + t2^4 - 2t1^2x1^2 + 2t2^2x1^2 + x1^4
   + 2t1^2x2^2 - 2t2^2x2^2 - 2x1^2x2^2 + x2^4 + 4t1^2x1 xu
   - 4t2^2x1 xu - 4x1^3xu - 4t1^2x2 xu + 4t2^2x2 xu
   + 4x1^2x2 xu + 4x1 x2^2xu - 4x2^3xu + 4x1^2xu^2
   - 8x1 x2 xu^2 + 4x2^2xu^2 - 2t1^2y1^2 - 2t2^2y1^2
   + 2x1^2y1^2 + 2x2^2y1^2 - 4x1 xu y1^2 - 4x2 xu y1^2
   + 4xu^2y1^2 + y1^4 + 4t1^2y1 y2 + 4t2^2y1 y2
   - 4x1^2y1 y2 - 4x2^2y1 y2 + 8x1 xu y1 y2
   + 8x2 xu y1 y2 - 8xu^2y1 y2 - 4y1^3y2 - 2t1^2y2^2
   - 2t2^2y2^2 + 2x1^2y2^2 + 2x2^2y2^2 - 4x1 xu y2^2
   - 4x2 xu y2^2 + 4xu^2y2^2 + 6y1^2y2^2 - 4y1 y2^3 + y2^4
```

and similarly

```
Eqyu = Resultant[eq1,eq2,xu]
t1^4 - 2t1^2t2^2 + t2^4 - 2t1^2x1^2 - 2t2^2x1^2 + x1^4 + 4t1^2x1 x2
    + 4t2^2x1 x2 - 4x1^3x2 - 2t1^2x2^2 - 2t2^2x2^2 + 6x1^2x2^2 - 4x1 x2^3
    + x2^4 - 2t1^2y1^2 + 2t2^2y1^2 + 2x1^2y1^2 - 4x1 x2 y1^2 + 2x2^2y1^2
    + y1^4 + 2t1^2y2^2 - 2t2^2y2^2 + 2x1^2y2^2 - 4x1 x2 y2^2 + 2x2^2y2^2
    - 2y1^2y2^2 + y2^4 + 4t1^2y1 yu - 4t2^2y1 yu - 4x1^2y1 yu + 8x1 x2 y1 yu
    - 4x2^2y1 yu - 4y1^3yu - 4t1^2y2 yu + 4t2^2y2 yu - 4x1^2y2 yu
    + 8x1 x2 y2 yu - 4x2^2y2 yu + 4y1^2y2 yu + 4y1 y2^2yu - 4y2^3yu
    + 4x1^2yu^2 - 8x1 x2 yu^2 + 4x2^2yu^2 + 4y1^2yu^2 - 8y1 y2 yu^2 + 4y2^2yu^2
```

then solving the two univariate polynomial equations separately,

```
NRoots[(Eqxu/.{x1 → xm[[1]],x2 → xm[[2]],y1 → ym[[1]],
y2 → ym[[2]],t1 → tm[[1]],t2 → tm[[2]]}) = = 0,xu]
xu = = 48071.6||xu = = 48565.3
```

```
NRoots[(Eqyu/.{x1 → xm[[1]],x2 → xm[[2]],y1 → ym[[1]],
y2 → ym[[2]],t1 → tm[[1]],t2 → tm[[2]]}) = = 0,yu]
yu = = 6058.98||yu = = 7133.03
```

1.6.3 3D Resection

In case of 3D resection problem, the distances between the points P_i are known $S_{i,j}$ as well as angles $\varphi_{i,j}$ and the length of the edges, S_i should be computed.

Problem

The means practically, we should solve the *Grunert equations,*

$$S_{1,2}^2 = S_1^2 + S_2^2 - 2S_1S_2\cos\left(\varphi_{1,2}\right),$$

$$S_{2,3}^2 = S_2^2 + S_3^2 - 2S_2S_3\cos\left(\varphi_{2,3}\right),$$

$$S_{3,1}^2 = S_3^2 + S_1^2 - 2S_3S_1\cos\left(\varphi_{3,1}\right),$$

where the unknowns are s_1, s_2, s_3, see Fig. 1.11.
In case of for a regular tetrahedron $\varphi_{i,j} = \pi/3$, and $2\cos(\pi/3) = 1$, and $S_{i,\,j} = d$. Introducing $x_i = S_i$, our system is,

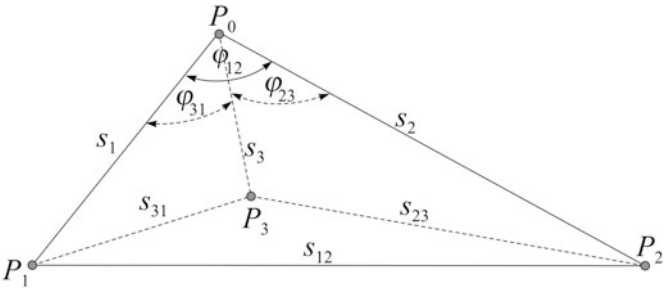


Fig. 1.11 The geometrical interpretation of the 3D resection problem

```
p1 = x1^2 - x1x2 + x2^2 - d = = 0;
p2 = x2^2 - x2x3 + x3^2 - d = = 0;
p3 = x3^2 - x1x3 + x1^2 - d = = 0;
```

Let us solve this problem in symbolic way employing Dixon resultant.

Solution

```
Clear[X1,X2,X3]
DixonResultant[{p1[[1]],p2[[1]],p3[[1]]},{x2,x3},{X2,X3}]
4 (d^3 x1^2 - 3d^2 x1^4 + 3dx1^6 - x1^8)
```

and solve the univariate polynomial equation,

```
Solve[% = = 0,x1]
{{x1 → 0},{x1 → 0},{x1 → - √d},{x1 → - √d},
 {x1 → - √d},{x1 → √d},{x1 → √d},{x1 → √d}}
```

Then we eliminate the variables ($x1$, $x3$)

```
DixonResultant[{p1[[1]],p2[[1]],p3[[1]]},{x1,x3},{X1,X3}]
 - 4 (d^3 x2^2 - 3d^2 x2^4 + 3dx2^6 - x2^8)
```

```
Solve[% = = 0,x2]
{{x2 → 0},{x2 → 0},{x2 → - √d},{x2 → - √d},
 {x2 → - √d},{x2 → √d},{x2 → √d},{x2 → √d}}
```

and then let us eliminate the variables ($x1$, $x2$)

```
DixonResultant[{p1[[1]],p2[[1]],p3[[1]]},{x1,x2},{X1,X2}]
```

```
 - 4 (d^3 x3^2 - 3d^2 x3^4 + 3dx3^6 - x3^8)
```

```
Solve[% == 0, x3]
{{x3 → 0}, {x3 → 0}, {x3 → -√d}, {x3 → -√d},
 {x3 → -√d}, {x3 → √d}, {x3 → √d}, {x3 → √d}}
```

Only positive solutions are accepted.

1.6.4 Pose Estimation

Now, we consider a similar but considerably more difficult problem discussed in Sect. 1.4.4. Suppose we have a quadrilateral ABCE; it does not have to be planar. The distances between each pair of vertices are known. The object moves. We observe it from point *P*, noting the angles spanned by each pair of vertices. The classic four point pose problem is to deduce the distances *X*1, *X*2, *X*3, *X*4.

```
eq1 = X1^2 + X2^2 - X1 X2 r - AB
 - AB + X1^2 - r X1 X2 + X2^2

eq2 = X1^2 + X3^2 - X1 X3 q - AC
 - AC + X1^2 - q X1 X3 + X3^2

eq3 = X2^2 + X3^2 - X3 X2 p - BC
 - BC + X2^2 - p X2 X3 + X3^2

eq4 = X1^2 + X4^2 - X1 X4 s - AE
 - AE + X1^2 - s X1 X4 + X4^2

eq5 = X4^2 + X3^2 - X4 X3 t - CE
 - CE + X3^2 - t X3 X4 + X4^2

eq6 = X2^2 + X4^2 - X4 X2 u - BE
 - BE + X2^2 - u X2 X4 + X4^2
```

There are four variables *X*1, *X*2, *X*3, *X*4. The parameters are AB, BC, CE, AE, AC, BE. An overdetermined system results from applying the law of cosines to each triangle having vertex *P*. Using four equations including the diagonals AC and BE gives an easy system of equations, solvable by many means.

For example in order to reduce the problem let us consider the equations eq3, eq5 and eq6 containing variables (*X*2, *X*3, *X*4). The *Gröbner basis* is,

```
AbsoluteTiming[sol = GroebnerBasis[{eq3, eq5, eq6}, {X2, X3, X4}];]
{2.48481, Null}
```

The number of the polynomials is

```
Length[sol]
31

Map[Exponent[#,{X2,X3,X4}]&,sol]//TableForm
0  0  8
0  1  7
0  1  6
0  1  7
0  1  7
0  1  5
0  1  7
0  1  7
0  1  7
0  1  7
0  1  7
0  1  7
0  1  7
0  1  4
0  1  6
0  1  6
0  1  4
0  1  7
0  1  7
0  2  2
1  1  3
1  1  3
1  1  3
1  1  5
1  1  2
1  1  3
1  1  4
1  1  3
1  1  3
1  1  4
2  0  2
```

therefore the problem is solvable, since considering the first polynomials we get an univariate polynomial of degree eight.

```
Sol[[1]]
```

$$
\begin{aligned}
&BC^4 - 4BC^3BE + 6BC^2BE^2 - 4BCBE^3 + BE^4 - 4BC^3CE \\
&\quad + 12BC^2BECE - 12BCBE^2CE + 4BE^3CE + 6BC^2CE^2 \\
&\quad - 12BCBECE^2 + 6BE^2CE^2 - 4BCCE^3 + 4BECE^3 + CE^4 \\
&\quad - 2BC^2BECEp^2 + 4BCBE^2CEp^2 - 2BE^3CEp^2 + 4BCBECE^2p^2 \\
&\quad - 4BE^2CE^2p^2 - 2BECE^3p^2 + BE^2CE^2p^4 + 8BC^3X4^2 - 24BC^2BEX4^2 \\
&\quad + 24BCBE^2X4^2 - 8BE^3X4^2 - 24BC^2CEX4^2 + 48BCBECEX4^2 \\
&\quad - 24BE^2CEX4^2 + 24BCCE^2X4^2 - 24BECE^2X4^2 - 8CE^3X4^2 \\
&\quad + 2BC^2BEp^2X4^2 - 4BCBE^2p^2X4^2 + 2BE^3p^2X4^2 \\
&\quad + 2BC^2CEp^2X4^2 - 16BCBECEp^2X4^2 + 14BE^2CEp^2X4^2 \\
&\quad - 4BCCE^2p^2X4^2 + 14BECE^2p^2X4^2 + 2CE^3p^2X4^2 \\
&\quad - 2BE^2CEp^4X4^2 - 2BECE^2p^4X4^2 - 2BC^3t^2X4^2 \\
&\quad + 6BC^2BEt^2X4^2 - 6BCBE^2t^2X4^2 + 2BE^3t^2X4^2 \\
&\quad + 4BC^2CEt^2X4^2 - 8BCBECEt^2X4^2 + 4BE^2CEt^2X4^2 \\
&\quad - 2BCCE^2t^2X4^2 + 2BECE^2t^2X4^2 - BC^2BEp^2t^2X4^2 \\
&\quad + 2BCBE^2p^2t^2X4^2 - BE^3p^2t^2X4^2 - BECE^2p^2t^2X4^2 \\
&\quad + BC^3ptuX4^2 - BC^2BEptuX4^2 - BCBE^2ptuX4^2 \\
&\quad + BE^3ptuX4^2 - BC^2CEptuX4^2 \\
&\quad + 6BCBECEptuX4^2 - 5BE^2CEptuX4^2 - BCCE^2ptuX4^2 \\
&\quad - 5BECE^2ptuX4^2 + CE^3ptuX4^2 + BCBECEp^3tuX4^2 \\
&\quad + BE^2CEp^3tuX4^2 + BECE^2p^3tuX4^2 - 2BC^3u^2X4^2 \\
&\quad + 4BC^2BEu^2X4^2 - 2BCBE^2u^2X4^2 + 6BC^2CEu^2X4^2 \\
&\quad - 8BCBECEu^2X4^2 + 2BE^2CEu^2X4^2 - 6BCCE^2u^2X4^2 \\
&\quad + 4BECE^2u^2X4^2 + 2CE^3u^2X4^2 - BC^2CEp^2u^2X4^2 \\
&\quad - BE^2CEp^2u^2X4^2 + 2BCCE^2p^2u^2X4^2 - CE^3p^2u^2X4^2 \\
&\quad + 24BC^2X4^4 - 48BCBEX4^4 + 24BE^2X4^4 - 48BCCEX4^4 \\
&\quad + 48BECEX4^4 + 24CE^2X4^4 \\
&\quad - 2BC^2p^2X4^4 + 12BCBEp^2X4^4 - 10BE^2p^2X4^4 + 12BCCEp^2X4^4 \\
&\quad - 28BECEp^2X4^4 - 10CE^2p^2X4^4 + BE^2p^4X4^4 + 4BECEp^4X4^4 \\
&\quad + CE^2p^4X4^4 - 10BC^2t^2X4^4 + 20BCBEt^2X4^4 - 10BE^2t^2X4^4 \\
&\quad + 12BCCEt^2X4^4 - 12BECEt^2X4^4 - 2CE^2t^2X4^4 + BC^2p^2t^2X4^4 \\
&\quad - 4BCBEp^2t^2X4^4 + 3BE^2p^2t^2X4^4 + 2BECEp^2t^2X4^4 \\
&\quad + CE^2p^2t^2X4^4 + BC^2t^4X4^4 - 2BCBEt^4X4^4 + BE^2t^4X4^4 \\
&\quad + 2BC^2ptuX4^4 - 4BCBEptuX4^4 + 2BE^2ptuX4^4 \\
&\quad - 4BCCEptuX4^4 + 20BECEptuX4^4
\end{aligned}
$$

```
+ 2CE^2ptuX4^4 - BCBEp^3tuX4^4 - BE^2p^3tuX4^4 - BCCEp^3tuX4^4
- 4BECEp^3tuX4^4 - CE^2p^3tuX4^4 - BC^2pt^3uX4^4
+ 2BCBEpt^3uX4^4 - BE^2pt^3uX4^4 - BCCEpt^3uX4^4
- BECEpt^3uX4^4 - 10BC^2u^2X4^4 + 12BCBEu^2X4^4
- 2BE^2u^2X4^4 + 20BCCEu^2X4^4 - 12BECEu^2X4^4
- 10CE^2u^2X4^4 + BC^2p^2u^2X4^4 + BE^2p^2u^2X4^4
- 4BCCEp^2u^2X4^4 + 2BECEp^2u^2X4^4 + 3CE^2p^2u^2X4^4
+ 3BC^2t^2u^2X4^4 - 4BCBEt^2u^2X4^4 + BE^2t^2u^2X4^4 - 4BCCEt^2u^2X4^4
+ CE^2t^2u^2X4^4 + BCBEp^2t^2u^2X4^4 + BCCEp^2t^2u^2X4^4
+ BECEp^2t^2u^2X4^4 - BC^2ptu^3X4^4 - BCBEptu^3X4^4
+ 2BCCEptu^3X4^4 - BECEptu^3X4^4 - CE^2ptu^3X4^4
+ BC^2u^4X4^4 - 2BCCEu^4X4^4 + CE^2u^4X4^4 + 32BCX4^6
- 32BEX4^6 - 32CEX4^6 - 8BCp^2X4^6 + 16BEp^2X4^6
+ 16CEp^2X4^6 - 2BEp^4X4^6 - 2CEp^4X4^6 - 16BCt^2X4^6
+ 16BEt^2X4^6 + 8CEt^2X4^6 + 2BCp^2t^2X4^6 - 4BEp^2t^2X4^6
- 2CEp^2t^2X4^6 + 2BCt^4X4^6 - 2BEt^4X4^6 + 4BCptuX4^6
- 12BEptuX4^6 - 12CEptuX4^6 + BCp^3tuX4^6 + 3BEp^3tuX4^6
+ 3CEp^3tuX4^6 - BCpt^3uX4^6 + 3BEpt^3uX4^6 + CEpt^3uX4^6
- 16BCu^2X4^6 + 8BEu^2X4^6 + 16CEu^2X4^6 + 2BCp^2u^2X4^6
- 2BEp^2u^2X4^6 - 4CEp^2u^2X4^6 + 8BCt^2u^2X4^6 - 2BEt^2u^2X4^6
- 2CEt^2u^2X4^6 - 2BCp^2t^2u^2X4^6 - BEp^2t^2u^2X4^6
- CEp^2t^2u^2X4^6 - BCt^4u^2X4^6 - BCptu^3X4^6 + BEptu^3X4^6
+ 3CEptu^3X4^6 + BCpt^3u^3X4^6 + 2BCu^4X4^6
- 2CEu^4X4^6 - BCt^2u^4X4^6 + 16X4^8
- 8p^2X4^8 + p^4X4^8 - 8t^2X4^8 + 2p^2t^2X4^8 + t^4X4^8 + 8ptuX4^8
- 2p^3tuX4^8 - 2pt^3uX4^8 - 8u^2X4^8 + 2p^2u^2X4^8
+ 2t^2u^2X4^8 + p^2t^2u^2X4^8 - 2ptu^3X4^8 + u^4X4^8
```

or using *Dixon KSY*

```
≪Resultant'Dixon'
Clear[x2,x3]

AbsoluteTiming[soldix=
 DixonMatrix[{eq3,eq5,eq6},{X2,X3},{x2,x3}]//Simplify;]
{0.021031,Null}
```

```
Dimensions[soldix]
{6,6}

MatrixRank[soldix]
5
```

The matrix is not full rank, therefore the maximal minor is used,

```
AbsoluteTiming[solX4 =
 Apply[Plus,Flatten[Minors[soldix,5]]]//Simplify;]
{1.00794,Null}
```

```
solX4
  p(tX4(tu^2X4^3(-BE+X4^2)(-CE+X4^2)-X4(BCt-BEt+CEpu
   +tX4^2-puX4^2)(-(-BC+BE+CE-2X4^2)^2+t^2X4^2(BC-BE+X4^2))
   -(CEp+(-p+tu)X4^2)(-BCt^2uX4^3+uX4(BC-CE+X4^2)
   (BC-BE-CE+2X4^2)))+uX4(X4(BE-X4^2)(BCt-BEt+CEpu
   +tX4^2-puX4^2)(-BCp+BEp+CEp-2pX4^2+pt^2X4^2+tuX4^2)
   (BC-BE-CE+2X4^2-ptuX4^2)(-BCt^2uX4^3+uX4(BC-CE+X4^2)
   (BC-BE-CE+2X4^2))+u^2X4^3(-(BC-CE+X4^2)(BEpt+BCu
   -CEu-ptX4^2+uX4^2)+BCt(BEp+(-p+tu)X4^2)))+(BC-BE
   -CE+2X4^2)(-(BE-X4^2)(-BCp+BEp+CEp-2pX4^2+pt^2X4^2+tuX4^2)
   (CEp+(-p+tu)X4^2)+(BC-BE-CE+2X4^2-ptuX4^2)
   (-(-BC+BE+CE-2X4^2)^2+t^2X4^2(BC-BE+X4^2))
   -uX4(X4(BC-BE-CE+2X4^2)(-BEpt-BCu+CEu+ptX4^2-uX4^2)
   +tX4(BC-BE+X4^2)(BEp+(-p+tu)X4^2)))+pX4(tuX4(-BE+X4^2)
   (-CE+X4^2)(-BC+BE+CE-2X4^2+ptuX4^2)-uX4(CEp+(-p+tu)X4^2)
   (-(BC-CE+X4^2)(BEpt+BCu-CEu-ptX4^2+uX4^2)+BCt(BEp
   +(-p+tu)X4^2))+(BCt-BEt+CEpu+tX4^2-puX4^2)
   (X4(BC-BE-CE+2X4^2)(-BEpt-BCu+CEu+ptX4^2-uX4^2)
   +tX4(BC-BE+X4^2)(BEp+(-p+tu)X4^2)))+p(CE-X4^2)
   (BE^3p-CE^2(p+tu-pu^2)X4^2-CE(p^3-2tu-p^2tu
   +p(-4+t^2+2u^2))X4^4+p(-4+p^2+t^2-ptu+u^2)X4^6
   +BE^2(-CEp(-2+p^2)+(-5p+p^3+tu)X4^2)
   +BC^2(BEp-(p+tu-pu^2)X4^2)+BE(CE^2p+CEp(-6+2p^2+t^2-ptu)X4^2
   +(8p-2p^3-pt^2-2tu+p^2tu)X4^4)
```

```
+ BC(-2BE^2p + 2CE(p + tu - pu^2)X4^2 + (p^2tu + t(-2 + t^2)u
  - p(4 + (-2 + t^2)u^2))X4^4 - BEp(2CE + (-6 + ptu)X4^2)))
  + ptX4(-X4(BE - X4^2)(CE^2pt + CE^2u + BECE(2 + p^2)u
  + BC^2(-pt + u) + BE^2(-pt + u) - 2CEptX4^2 - 4CEuX4^2
  - CEp^2uX4^2 + CEt^2uX4^2 + CEu^3X4^2 - BE(-2pt
  + 4u + p^2u)X4^2 + 4uX4^4 + p^2uX4^4 - t^2uX4^4
  - u^3X4^4 + BC(2BE(pt - u) - 2CEu +
(-u(-4 + u^2) + pt(-2 + u^2))X4^2)) + uX4(ptuX4^2(BE - X4^2)(-CE + X4^2)
+ (BC - BE - CE + 2X4^2)(-(-BC + BE + CE - 2X4^2)^2 + t^2X4^2(BC - BE + X4^2))
+ uX4(-BCt^2uX4^3 + uX4(BC - CE + X4^2)(BC - BE - CE + 2X4^2)))))
```

```
Exponent[solX4,{X2,X3,X4}]
{0,0,8}
```

So far we have used a set of equations that includes one of the diagonals in Fig. 1.6 (*BE* or *AC*). If the figure is known to be flexible, we must use equations that only refer to the outside edges. In this case, neither Gröbner Basis nor Dixon KSY will work in *Mathematica*.

In addition *Maple* and *Magma* both fail on this. Maple-FGb was killed after 25 h, exhausting 62 gig of RAM. *Magma* was killed after 40 h.

However, the resultant can be computed in less than one second by *Fermat* using the variation of EDF in which we run EDF to a certain point (in this case after the third row) and then use *Gentleman-Johnson* (1976) to compute the determinant of the remaining 9×9 matrix. This is a good example of the enormous flexibility of Dixon-EDF, in which the user is in control throughout the process and has many options. The resultant for X_4 has 24,068 terms.

References

Awange JL, Paláncz B (2016) Geospatial algebraic computations, theory and applications. Springer, Berlin

Buchberger B, Winkler F (1998) Groebner bases and applications. London mathematical society lecture note series 251. Cambridge University Press, Cambridge

Dickenstein A, Emiris IZ (eds) (2005) Solving polynomial equations, foundations, algorithms and applications (Algorithms and computation in mathematics), vol 14. Springer, Berlin

Kapur D, Saxena T, Yang L (1994) Algebraic and geometric reasoning using Dixon resultants. In: Proceeding of the International Symposium on Symbolic and Algebraic Computation. ACM Press, New York

Lewis RH (2008) Heuristics to accelerate the Dixon resultant. Math Comput Simul 77(4):400–407

Lichtblau D (2013) Approximate Gröbner bases, overdetermined polynomial systems and approximate GCDs. Hindawi Publishing Corporation, ISRN Computational Mathematics, vol 2013, Article ID 352806, pp 1–12, http://dx.doi.org/10.11155/2013/352806

Paláncz B, Zaletnyik P, Awange JL, Grafarend EW (2008) Dixon resultant's solution of systems of geodetic polynomial equations. J Geodesy 82(8):505–511

Sasaki T (2014) A practical method for floating-point Gröbner basis computation. In: Feng R, Lee W, Sato Y (eds) Computer mathematics. Springer, Berlin, pp 109–124

Szanto A (2011) Hybrid symbolic-numeric methods for the solution of polynomial systems: tutorial overview. In: Proceeding ISSAC '11 Proceedings of the 36th international symposium on symbolic and algebraic computation, San Jose, California, USA, June 08–11, 2011 ACM New York, NY, USA ©2011, pp 9–10

Chapter 2
Homotopy Solution of Nonlinear Systems

2.1 The Concept of Homotopy

The continuous deformation of an object to another object is known as *homotopy*. Let us consider a simple geometric example that defines the homotopy between a circle and a square. The parametric equations of the circle (Fig. 2.1) is given by,

$$x = R\cos(\alpha),$$

$$y = R\sin(\alpha).$$

while the parametric equations of the square are

$$x = f(\alpha)\,R\cos(\alpha),$$

$$y = f(\alpha)\,R\sin(\alpha),$$

where

$$f(\alpha) = \frac{1}{\max(|\sin(\alpha)|, |\cos(\alpha)|)}.$$

See Fig. 2.2.
Now, the homotopy function is given by

$$H(\alpha, \lambda) = \lambda \begin{pmatrix} R\cos(\alpha) \\ R\sin(\alpha) \end{pmatrix} + (1-\lambda)\begin{pmatrix} f(\alpha)\,R\cos(\alpha) \\ f(\alpha)\,R\sin(\alpha) \end{pmatrix}.$$

In geometrical terms, the homotopy H provides us a continuous, smooth deformation from a *square*—which is obtained for $\lambda = 0$ by $H(\alpha, 0)$—to a circle—which is obtained for $\lambda = 1$ by $H(\alpha, 1)$. One moment of the animation of the homotopy can be seen in Fig. 3.3 for $\lambda \in [0, 1]$. We call it linear homotopy because H is a linear function of the variable λ (Fig. 2.3).

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_2

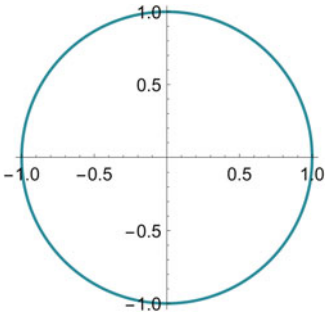


Fig. 2.1 A circle ($R = 1$)

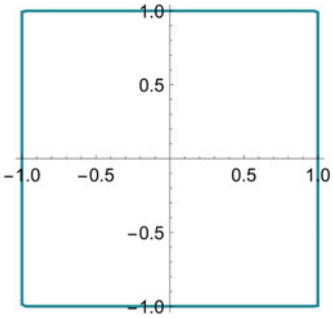


Fig. 2.2 A square

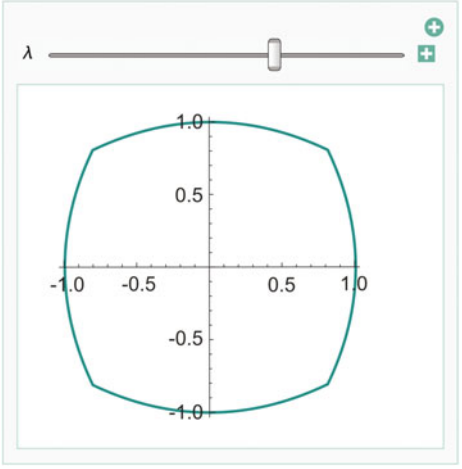


Fig. 2.3 Homotopy between a square and a circle

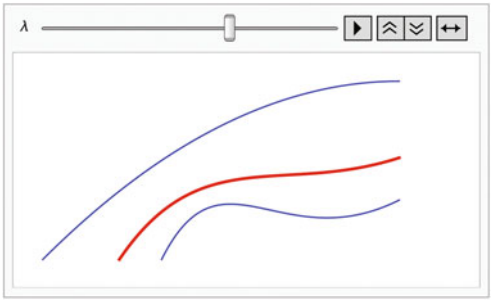


Fig. 2.4 Homotopy between two Bezier splines (i.e., between the two thin lines)

Lines are also geometric objects. Let us consider two Bezier splines with four control points each,

```
pts1 = {{0, -1}, {2, 1}, {4, 2}, {6, 2}};
pts2 = {{2, -1}, {3, 1}, {4, -1}, {6, 0}};
```

and a homotopy between them (see Fig. 2.4),

```
BezierCurve[(1 - λ)pts1 + λpts2]
```

2.2 Solving Nonlinear Equation via Homotopy

A homotopy continuation method deforms the known roots of the *start system* into the roots of the *target system*. Now, let us look at how homotopy can be used to solve a simple polynomial equation. Consider the polynomial equation of degree two,

$$q(x) = x^2 + 8x - 9 = 0.$$

By deleting the middle term, we can get a simpler equation, which can be solved easily by inspection,

$$p(x) = x^2 - 9 = 0.$$

This equation also has two roots and will be considered the start system for the target system. The linear homotopy can be defined as follows

$$H(x, \lambda) = (1 - \lambda)p(x) + \lambda q(x) = -9 + x^2 + 8x\lambda.$$

Let us plot the homotopy H for the polynomials p(x) and q(x) for different values of λ (Fig. 2.5).

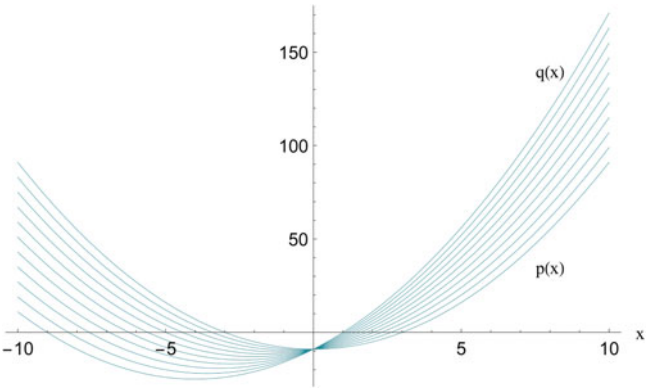


Fig. 2.5 Deformation of the function H from p(x) to q(x) as function of parameter λ

The homotopy continuation method deforms $p(x) = 0$, the known roots of the start system, into $q(x) = 0$, the roots of the target system. Let us solve the equation $H(x, \lambda) = 0$ for different values of λ. First let $\lambda_1 = 0.2$, and consider $x_0 = 3$, one of the solutions of $p(x) = 0$, as the initial guess value. Solving $H(x, \lambda_1) = 0$ using the Newton-Raphson method, we get

```
x0 = 3; λ1 = 0.2; x1 = x/. FindRoot [H(x, λ1) = = 0, (x, x0)]
2.30483
```

Now use this result as the guess value for the next solution step

```
λ2 = 0.4; x2 = x/. FindRoot [H(x, λ2) = = 0, (x, x1)]
1.8
```

and so on,

```
λ3 = 0.6; x3 = x/. FindRoot[H(x, λ3) = = 0, (x, x2)]
1.44187
λ4 = 0.8; x4 = x/. FindRoot[H(x, λ4) = = 0, (x, x3)]
1.18634
λ5 = 1; x5 = x/. FindRoot[H(x, λ5) = = 0, (x, x4)]
1.
```

Let us display the transition of a root of the polynomial $p(x)$ into a root of the polynomial $q(x)$, (Fig. 2.6).

The homotopy path is the function $x = x(\lambda)$, and x_i is the root of $H(x, \lambda_i)$. Figure 2.7 below shows the path of homotopy transition of the root of $p(x)$ into the root of $q(x)$.

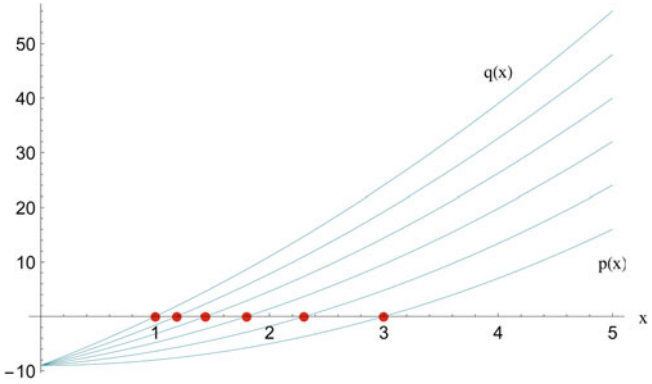


Fig. 2.6 The transition of the root from $x = 3$ to $x = 1$ during the deformation of the function H

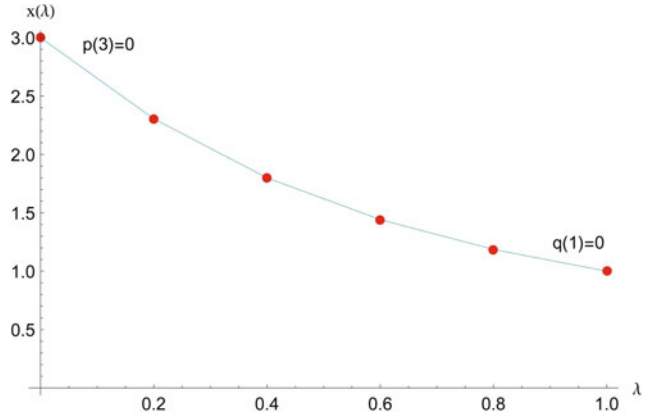


Fig. 2.7 The homotopy path is the function $x = x(\lambda)$, where in every point, at every lambda value $H \equiv 0$

2.3 Tracing Homotopy Path as Initial Value Problem

Comparing homotopy solution with the traditional Newton-Raphson solution, it is clear that if $\Delta\lambda$ is small enough, the convergence may be ensured in every step. Consequently one can consider the root tracing procedure with an infinitesimal step size as an initial value problem of an ordinary differential equation.
Since $H(x,\lambda) = 0$ for every $\lambda \in [0, 1]$, therefore

$$dH(x,\lambda) = \frac{\partial H}{\partial x}dx + \frac{\partial H}{\partial \lambda}d\lambda \equiv 0 \qquad \lambda \in [0,1].$$

Then the initial value problem is

$$H_x \frac{dx(\lambda)}{d\lambda} + H_\lambda = 0l$$

with

$$x(0) = x_0.$$

It goes without saying that for systems of equations, the Jacobian should be used.

Here H_x is the Jacobian of H with respect to x_i, $i = 1,\ldots, n$, in case of n nonlinear equations with n variables.

In our single variable case, the two partial derivatives of the homotopy function are

```
dHdλ = D[H[x, λ], λ]
8x
dHdx = D[H[x, λ], x]
2x(1 - λ) + (8 + 2x)λ
```

Then the right hand side of the differential equation to be solved is

$$\texttt{deqrhs} = -\frac{\texttt{dHd}\lambda}{\texttt{dHdx}}\texttt{/.x -> x[}\lambda\texttt{]}$$

$$-\frac{8x[1]}{2(1-1)x[1] + 1(8 + 2x[1])}$$

The differential equation,

$$\texttt{deq} = \texttt{D[x[}\lambda\texttt{], }\lambda\texttt{] == deqrhs}$$

$$x'[1] == -\frac{8x[1]}{2(1-1)x[1] + 1(8 + 2x[1])}$$

The initial value is

```
x0 = x0
3
```

The numerical solution of this initial value problem is

```
sol = NDSolve[{deq, x[0] == x0}, {x[λ]}, {λ, 0, 1}];
```

The resulting trajectory is the homotopy path (Fig. 2.8),
The value of the corresponding root of $q(x)$ is $x(\lambda)$ at $\lambda = 1$,

```
First[x[λ]/.sol/.λ → 1]
1.
```

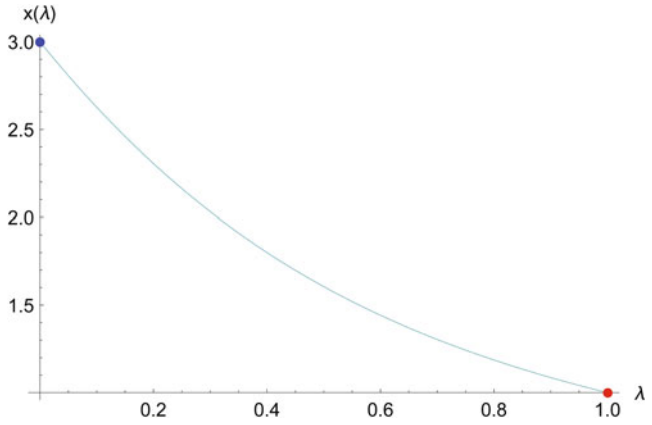


Fig. 2.8 The trajectory of the solution as the homotopy path

2.4 Types of Linear Homotopy

Here we consider the most popular five different types of the homotopy.

2.4.1 General Linear Homotopy

As we have seen, the start system can be constructed intuitively, reducing the original system (target system) to a more simple system (start system), which roots can be easily computed. In order to get all of the roots of the target system, the start system should have so many roots as many as the target system has.

The start system can be constructed in different ways however there are two typical techniques for generating the start systems, which are usually employed.

2.4.2 Fixed-Point Homotopy

The starting system can be considered as

$$p(x) = x - x_0,$$

where x_0 is a guess value for the root of the target system $q(x) = 0$.

In that case the homotopy function is

$$H(x, \lambda) = (1 - \lambda)(x - x_0) + \lambda q(x).$$

2.4.3 Newton Homotopy

Another construction for the start system, when we consider p(x) as

$$p(x) = q(x) - q(x_0),$$

in that case the homotopy function is

$$H(x, \lambda) = (1 - \lambda)(q(x) - q(x_0)) + \lambda q(x),$$

or

$$H(x, \lambda) = q(x) - (1 - \lambda)q(x_0).$$

2.4.4 Affine Homotopy

This type of homotopy suggested by Jalali and Seader (2000) requires the first derivative of the target function $q'(x)$, then

$$H(x, \lambda) = (1 - \lambda)q'(x_0)(x - x_0) + \lambda q(x).$$

Undoubtably, this can effectively employed when the derivative can given in analytical form. In case of polynomials it is the case. It goes without saying that for system of equations the Jacobian should be used.

2.4.5 Mixed Homotopy

Rahimian et al. (2011) concerned with the use of homotopy function,

$$H(x, \lambda) = (1 - \lambda)(q(x) - q(x_0) + (x - x_0)) + \lambda q(x)$$

to track the approximate solution. Here the start system is a linear combination of the fixed point and the Newton homotopy.

There are other methods, too to construct start system for linear homotopy. In a section, following later, we shall see how one can define start system for polynomial systems automatically.

It is known that local methods, like Newton-Raphson method, require initial guess of a root, which is close to the intended root for the particular application. Global methods, like homotopy continuation method can find solution from a start guess, which is far from the solution.

2.5 Regularization of the Homotopy Function

Sometimes even this form of homotopy method may fail, because of a singularity resulting in diverging paths.

In order to avoid singularities in the field of real numbers, we consider a modified complex homotopy function,

$$H(x, \lambda) = \gamma(1 - \lambda)(x - x_0) + \lambda q(x),$$

where γ is a complex number. For almost all choices of a complex constant γ, all solution paths defined by the homotopy above are regular, i.e., for all $\lambda \in [0, 1]$, the Jacobian matrix of H (x, λ) is regular and no diverging path occurs.

2.6 Start System in Case of Algebraic Polynomial Systems

How can we automatically find the proper start system, which will provide all of the solutions of the target system? This problem can be solved if the nonlinear system is a system of algebraic polynomial equations.

Let us consider the case where we are looking for the homotopy solution of $f(x) = 0$, where f(x) is a *polynomial system*, $f(x) : \mathbb{R}^n \to \mathbb{R}^n$. To get all of the solutions, one should find out a proper polynomial system, as a start system, $g(x) = 0$, where $g(x) : \mathbb{R}^n \to \mathbb{R}^n$ with known or easily computable solutions.

An appropriate start system can be generated in the following way.

Let $f_i(x_1, \ldots, x_n)$, $i = 1, \ldots, n$ be a system of n polynomials. We are interested in the common zeros of the system namely, $f = (f_1(x), \ldots, f_n(x)) = 0$.

Let d_j denote the degree of the jth polynomial—that is, the degree of the highest order monomial in the equation. Then the system will be a proper starting system

$$g_j(x) = \mathrm{e}^{i\phi_j}\left(x_j^{d_j} - \left(\mathrm{e}^{i\vartheta_j}\right)^{d_j}\right) = 0, \qquad j = 1, \ldots, n,$$

where ϕ_j and ϑ_j are random real numbers in the interval $[0, 2\pi]$. The equation above has the obvious particular solution $x_j = \mathrm{e}^{i\vartheta_j}$ and the complete set of the starting solutions for $j = 1, \ldots, n$ are given by

$$\mathrm{e}^{\left(i\vartheta_j + \frac{2\pi i k}{d_j}\right)}, \qquad k = 0, 1, \ldots, d_j - 1.$$

Bezout's theorem states that the number of isolated roots of such a system is bounded by the total degree of the system,

$$\prod_{i=1}^{n} d_i = d_1 \cdot d_2 \cdot \ldots \cdot d_n.$$

Example
Let us consider the following system,

$$f_1(x,y) = x^2 + y^2 - 1,$$
$$f_2(x,y) = x^3 + y^3 - 1.$$

The degrees of the polynomials are

```
d1 = 2; d2 = 3;
```

Indeed, this system has the following six roots ($d_1\ d_2$ = 2 * 3 = 6), as expected.

```
NSolve[{f1[x,y] = = 0, f2[x,y] = = 0}, {x,y}]
 {{x → - 1. + 0.707107 i, y → - 1. - 0.707107 i},
 {x → - 1. - 0.707107 i, y → - 1. + 0.707107 i}, {x → 1., y → 0.},
 {x → 1., y → 0.}, {x → 0., y → 1.}, {x → 0., y → 1.}}
Length[%]
6
```

Now, we compute the start system. We generate random real numbers in the interval [0, 2π] as it follows

```
ϕ1 = Random[Real, {0, 2π}]
4.46668
ϕ2 = Random[Real, {0, 2π}]
1.97211
θ1 = Random[Real, {0, 2π}]
2.71370
θ2 = Random[Real, {0, 2π}]
0.163568
```

Then the start system is,

```
g1[x_, y_] : = e^(i ϕ1) (x^d1 - (e^(i θ1))^d1)
g2[x_, y_] : = e^(i ϕ2) (x^d2 - (e^(i θ2))^d2)
g1[x,y]
(- 0.243245 - 0.969965 i) ((- 0.655628 + 0.755084 i) + x^2)
g2[x,y]
(- 0.390631 + 0.920548 i) ((- 0.882001 - 0.471247 i) + y^3)
```

Then the complete set of the starting solutions are,

```
Xi = Table[Exp[iθ1 + 2πik/d1],{k,0,d1 - 1}]
(-0.390631 + 0.920548 i)((-0.882001 - 0.471247 i) + y^3)
```

and

```
Yi = Table[Exp[i θ2 + 2πik/d2],{k,0,d2 - 1}]
{0.986653 + 0.16284 i, -0.63435 + 0.773046 i, -0.352303 - 0.935886 i}
```

We need all of the combination of the initial values $\{X_i, Y_j\}, \quad i = 1,2,\ j = 1,2$

```
X0 = Tuples[{Xi,Yi}]
 {{-0.909843 + 0.414953 i, 0.986653 + 0.16284 i},
 {-0.909843 + 0.414953 i, -0.63435 + 0.773046 i},
 {-0.909843 + 0.414953 i, -0.352303 - 0.935886 i},
 {0.909843 - 0.414953 i, 0.986653 + 0.16284 i},
 {0.909843 - 0.414953 i, -0.63435 + 0.773046 i},
 {0.909843 - 0.414953 i, -0.352303 - 0.935886 i}}
```

These values satisfy the start system.

So we have six initial values for solving our homotopy equations. These initial values will provide the start point of the six homotopy paths. The end points of these paths are the six solutions of the target system.

2.7 Homotopy Methods in *Mathematica*

The algorithm discussed above have been implemented in *Mathematica,*

```
≪Homotopy'LinearHomotopy'
```

There are the following functions

```
?LinearHomotopyFR
```

```
Computes the homotopy paths with direct path tracing.
Input parameters:
F  - list of functions of the target system,
G  - list of functions of the start system,
X  - list of variables,
X0 - list of initial values,
γ  - list of complex weights,
n  - number of subintervals,
λ  - dummy variable.
Output variables:
sol[[1]] - list of the solutions,
sol[[2]] - list of homotopy paths
```

```
?LinearHomotopyNDS02
```

```
Computes the homotopy paths with integration using numeric inverse.
Input parameters:
X  - list of variables,
F  - list of functions of the target system,
G  - list of functions of the start system,
X0 - list of initial values,
γ  - list of complex weights,
p  - standard precision -> 0; higher precision -> 1,
λ  - dummy variable.
Output variables:
sol[[1]] - list of the solutions,
sol[[2]] - list of homotopy paths
```

```
?Paths
```

```
Display homotopy paths.
Input parameters:
X   - list of variables,
sol - list of homotopy paths,
X0  - list of initial values,
 λ  - dummy variable.
```

```
?StartingSystem
```

```
Computes the start system for polynomial system.
Input parameters:
 F - list of functions of the target system,
 X - list of variables,
 d - list of orders of the equations.
Output parameters:
 G  - list of functions of the start system,
 X0 - solution of the start system, the initial values for homotopy
```

Now let us solve the system considered in the last example. The system is,

```
F = {f1[x,y],f2[x,y]}
{-1 + x^2 + y^2, -1 + x^3 + y^3}
```

The list of variables is

```
X = {x,y}
{x,y}
```

The degree of the polynomials

```
d = {d1,d2}
{2,3}
```

Let us generate the starting system

```
sol = StartingSystem[F,X,d];
```

Now our starting system is

```
G = sol[[1]]
 {(0.981373 - 0.192112 i)((0.971733 - 0.236081 i) + x^2),
 (-0.555004 - 0.831847 i)((-0.998626 - 0.0523973 i) + y^3)}
```

The initial values

```
X0 = sol[[2]]
 {{-0.118884 - 0.992908 i, 0.999847 + 0.0174729 i},
 {-0.118884 - 0 + 992908 i, -0.515056 + 0.857157 i},
 {-0.118884 - 0.992908 i, -0.484792 - 0.87463 i},
 {0.118884 + 0.992908 i, 0.999847 + 0.0174729 i},
 {0.118884 + 0.992908 i, -0.515056 + 0.857157 i},
 {0.118884 + 0.992908 i, -0.484792 - 0.87463 i}}
```

Let us compute the solution. Since the system of polynomials and the starting system are complex, we do not need the gamma "trick", therefore let,

```
γ = {1,1};
```

First we employ the direct path tracing

```
sol = LinearHomotopyFR[F,G,X,X0,γ,100,λ];
```

The roots are

```
sol1FR = Chop[sol[[1]],10^-8]
 {{0,1.},{-1.-0.707107 i,-1.+0.707107 i},
 {1.,0},{0,1.},{1.,0},{-1.+0.707107 i,-1.-0.707107 i}}
```

The trajectories of the solutions are

```
pFRS = Paths[X,sol[[2]],X0,λ]
```

Now, let us employ the method based on integration of the differential equation system, which employ numerical inverse (Fig. 2.9).

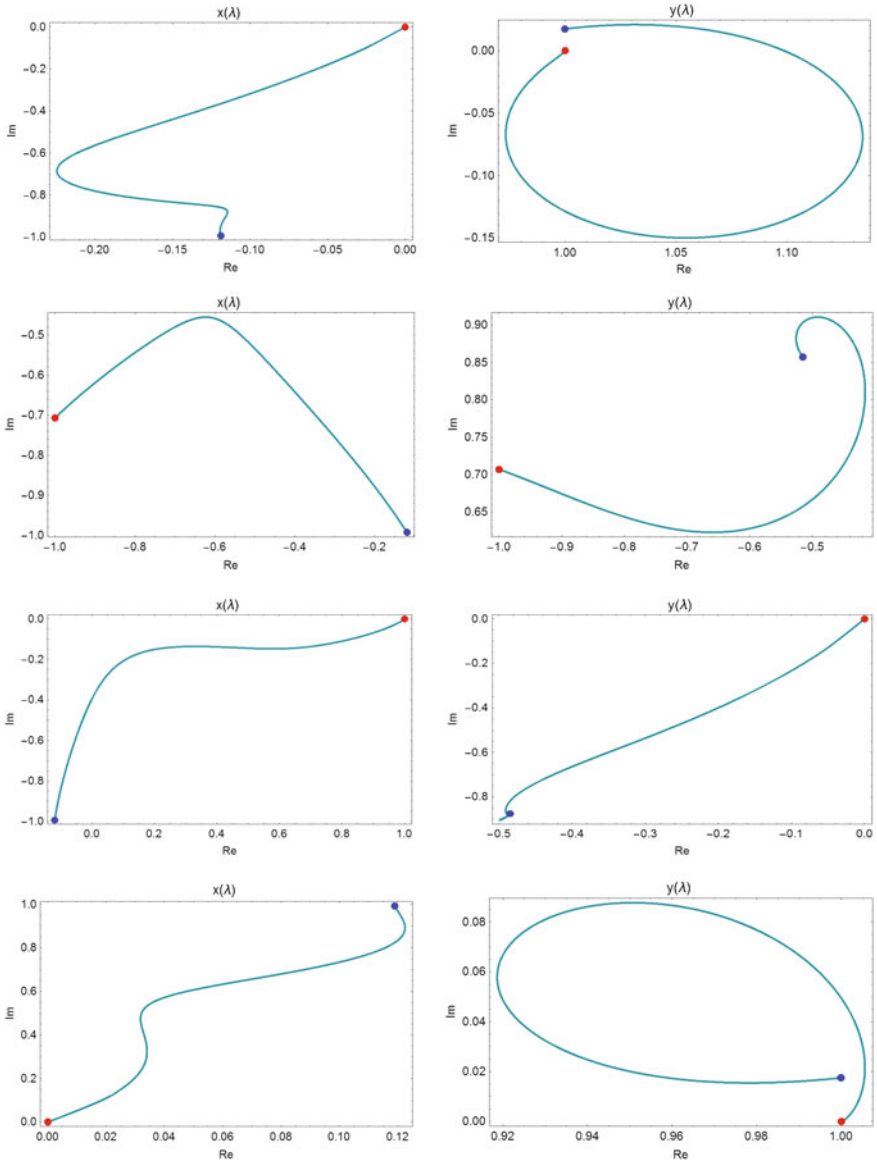


Fig. 2.9 The trajectories of the solutions

```
sol = LinearHomotopyNDS02[X, F, G, X0, γ, 1, λ][[1]]
 {{0, 1.}, {-1. - 0.707107 i - 1. + 0.707107 i},                .
 {1., 0}, {0, 1.}, {1., 0}, {-1. + 0.707107 i, -1. - 0.707107 i}}
```

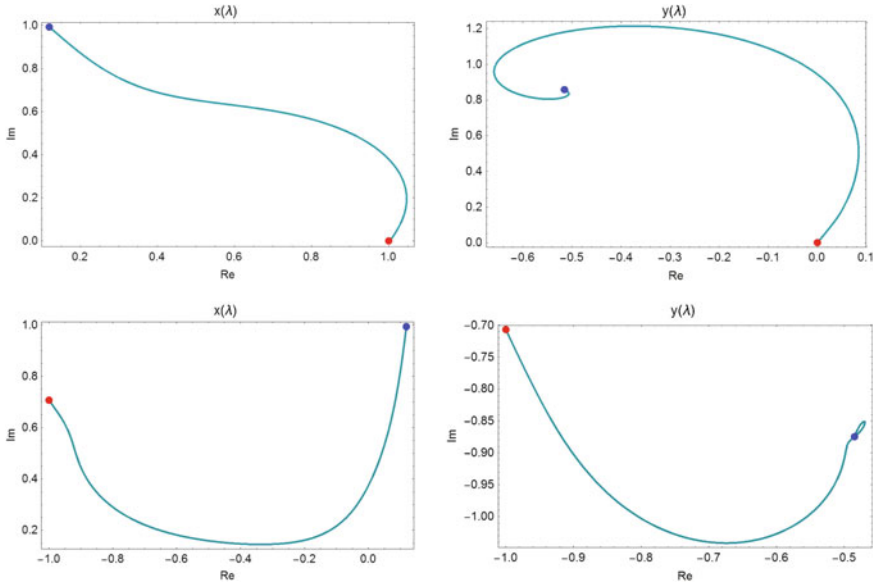


Fig. 2.9 (continued)

2.8 Parallel Computation

The tracing of the different paths are independent of each other, therefore parallel computation can be employed. Let us carry out the computation in non-parallel way

```
AbsoluteTiming[sol = LinearHomotopyFR[F,G,X,X0,γ,5000,λ];]
{14.9039,Null}
```

Without parallel execution, the processor performance is about 12%. The result is

```
sol1FR = Chop[sol[[1]],10^-8]
{{0,1.},{-1.-0.707107 i,-1.+0.707107 i},
{1.,0},{0,1.},{1.,0},{-1.+0.707107 i,-1.-0.707107 i}}
```

The preparation of the initial conditions for the parallel computation

```
X0P = Map[{#},X0]
{{{-0.118884 -0.992908 i, 0.999847 + 0.0174729 i}},
 {{-0.118884 -0.992908 i, -0.515056 + 0.857157 i}},
 {{-0.118884 -0.992908 i, -0.484792 -0.87463 i}},
 {{0.118884 + 0.992908 i, 0.999847 + 0.0174729 i}},
 {{0.118884 + 0.992908 i, -0.515056 + 0.857157 i}},
 {{0.118884 + 0.992908 i, -0.484792 -0.87463 i}}}
```

Our computer has four cores with two threads each,

```
LaunchKernels[2 $ProcessorCount]//Quiet;
"Updating from Wolfram Research server...".
```

We should distribute the information for the 8 threads

```
DistributeDefinitions[LinearHomotopyFR,F,G,X,γ,X0P];
```

Then the parallel computation can be carried out

```
AbsoluteTiming[sol =
 ParallelMap[LinearHomotopyFR[F,G,X,#,γ,5000,λ][[1]],X0P];]
{4.2528,Null}
Chop[sol,10^-8]
 {{{0,1.}},{{-1.-0.707107 i,-1.+0.707107 i}},{{1.,0}},
  {{0,1.}},{{1.,0}},{{-1.+0.707107 i,-1.-0.707107 i}}}
```

Now, the processor performance is about 100%. It means that using parallel evaluation the computation time can be reduced to the third of the non-parallel case, 4.25 < 14.9.

2.9 General Nonlinear System

The homotopy method is not restricted to algebraic systems. Now, we consider the following system of equations

```
eq1 = x^2 + y^2 - 1;
eq2 = Sin[x] - y;
```

The system cannot be solved via none of the symbolic methods discussed in Chap. 1, however local numerical solution requiring initial guess values is possible (Fig. 2.10)

```
solF = FindRoot[{eq1,eq2},{{x,-0.5},{y,-0.75}}]
{x → -0.739085,y → -0.673612}
```

Let us try to employ homotopy

```
F = {eq1,eq2}
{-1 + x^2 + y^2,-y + Sin[x]}
X = {x,y};
```

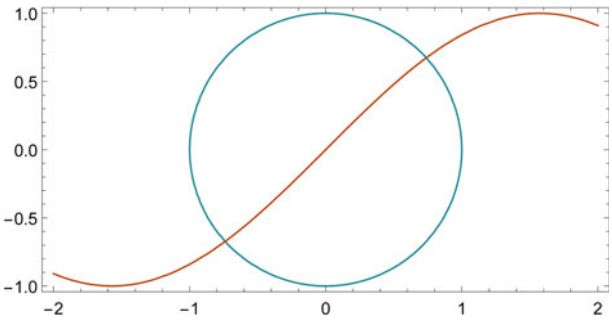


Fig. 2.10 Geometric representation of the non-polynomial system

Considering the starting system as

```
g1 = eq1
 - 1 + x^2 + y^2
```

Employing the linearized form of eq2 at $x = 0$

```
g2 = x - y
x - y
```

The solutions of this system are

```
X0 = {x, y} /. Solve[{g1 = = 0, g2 = = 0}, {x, y}]
{{ - 1/√2, - 1/√2}, {1/√2, 1/√2}}
```

The starting system

```
G = {g1, g2}
{ - 1 + x^2 + y^2, x - y}
```

Now, let us employ gamma trick

```
γ = {1 + i, i};
sol = LinearHomotopyFR[F, G, X, X0, γ, 100, λ];
sol[[1]]
{{ - 0.739085, - 0.673612}, {0.739085, 0.673612}}
```

Let us visualize these solutions, see Fig. 2.11.

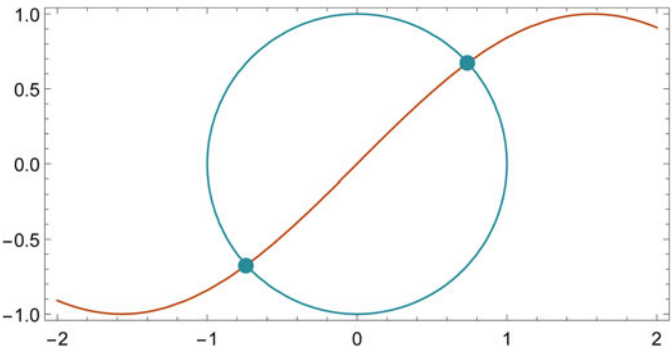


Fig. 2.11 Homotopy paths in case of non-polynomial system

2.10 Nonlinear Homotopy

The idea of this nonlinear homotopy function comes from the construction of the Bezier splines.

2.10.1 Quadratic Bezier Homotopy Function

Bezier curves are used to draw smooth curves along points on a path. In case of two points (P_0, P_1) the point Q_0 is running from P_0 to P_1 while the parameter λ is changing from 0 to 1, see Fig. 2.12

$$\vec{Q}_0 = (1 - \lambda)\vec{P}_0 + \lambda\vec{P}_1, \quad \lambda \in [0, 1].$$

In case of three points (P_0, P_1, P_2) the point Q_0 is ruining from P_0 to P_1, while point Q_1 is running from P_1 to P_2, see Fig. 2.13

$$\vec{Q}_1 = (1 - \lambda)\vec{P}_1 + \lambda\vec{P}_2, \quad \lambda \in [0, 1]$$

and point R_0 is running along a smooth path from P_0 to P_2 in a way,

$$\vec{R}_0 = (1 - \lambda)\vec{Q}_0 + \lambda\vec{Q}_1, \quad \lambda \in [0, 1]$$

or

$$\vec{R}_0 = (1 - \lambda)^2\vec{P}_0 + 2(1 - \lambda)\lambda\vec{P}_1 + \lambda^2\vec{P}_2, \quad \lambda \in [0, 1]$$

Considering analogy between this Bezier curve construction and the quadratic homotopy function Nor et al. (2013) suggested the following casting,

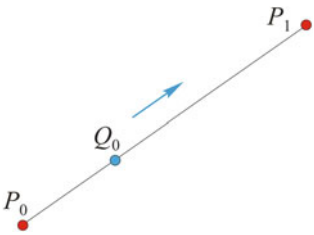


Fig. 2.12 Linear Bezier spline

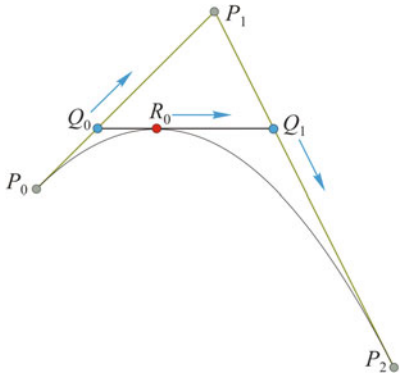


Fig. 2.13 Quadratic Bezier spline

$$P_0 \sim G(x)$$
$$P_1 \sim H_1(x, \lambda)$$
$$P_2 \sim F(x)$$

and

$$Q_0 \sim A(x, \lambda)$$
$$Q_1 \sim B(x, \lambda)$$
$$R_0 \sim H_2(x, \lambda)$$

where $H_1\ (x,\ \lambda)$ is the linear homotopy function,

$$H_1(x, \lambda) = (1 - \lambda)G(x) + \lambda F(x)$$

and $H_2\ (x,\ \lambda)$ is the quadratic Bezier homotopy function. Applying the analogy

$$A(x, \lambda) = (1 - \lambda)G(x) + \lambda H_1(x, \lambda)$$

$$B(x, \lambda) = (1 - \lambda)H_1(x, \lambda) + \lambda F(x)$$

then

$$H_2(x,\lambda)=(1-\lambda)^2G(x)+2\lambda(1-\lambda)H_1(x,\lambda)+\lambda^2F(x)=$$
$$(1-\lambda)^2G(x)+2\lambda(1-\lambda)((1-\lambda)G(\mathrm{x})\ +\ \lambda F(x))+\lambda^2F(x)$$

The coefficients of $H_2(x,\ \lambda)$ can be computed also as the coefficients of Bezier function. Let us employ the variables

$$H_0=G(x) \quad \text{for } \lambda=0$$

$$H_1=H_1(x,\lambda) \quad \text{for } \lambda\in(0,1)$$

$$H_2=F(x) \quad \text{for } \lambda=1$$

then

$$H_2(x,\lambda)=H_0B_0^2(\lambda)+H_1B_1^2(\lambda)+H_2B_2^2(\lambda)=\sum_{i=0}^{2}H_iB_i^2(\lambda),$$

where $B_i^n(\lambda)$ represents the ith Bernstein basis function of degree n at λ,

$$B_i^n(\lambda)=\binom{n}{i}(1-i)^{n-i},$$

where i = 0, 1, 2, …, n. In *Mathematica* has a built in function for these basis functions, however it is only a numerical one, `BernsteinBasis[n,i,λ]`.

For linear homotopy

$$\sum_{i=0}^{2}B_i^1(\lambda)=(1-\lambda)+\lambda=1,$$

```
Plot[ Σ_{i=0}^{2} BernsteinBasis[1,i,1],{λ,0,1}]
```

For quadratic homotopy (Fig. 2.14)

$$\sum_{i=0}^{2}B_i^2(\lambda)=(1-\lambda)^2+2\lambda(1-\lambda)+\lambda^2=1,$$

```
Plot[ Σ_{i=0}^{2} BernsteinBasis[2,i,1],{λ,0,1}]
```

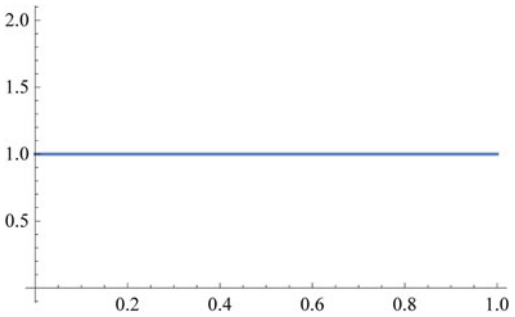


Fig. 2.14 Numerical verification of the sum of the first degree of Bernstein basis

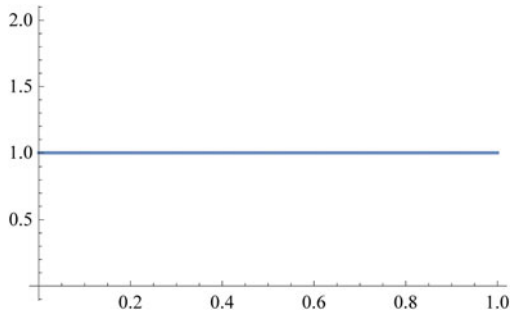


Fig. 2.15 Numerical verification of the sum of the second degree of Bernstein basis

It goes without saying that (Fig. 2.15)

$$H_2(x,0) = G(x)$$
$$H_2(x,1) = F(x)$$

Remark A recursive construction of the quadratic homotopy function can be given by the De Casteljau's algorithm. In our case see Fig. 2.16.

2.10.2 Implementation in Mathematica

The implementation of this quadratic homotopy function is easy only one line should be changed,

```
NonLinearHomotopyFR[F_,G_,X_,X0_,γ_,n_]:=
 Module[{H,X0L,λ0,i,β,m,R,RR,j,k,sol},
 Off[FindRoot::"lstol"];λ0=Table[i1/n,{i,0,n}];
 H=Flatten[(1-β)^2Thread[Gγ] +
  2β(1-β)((1-β)Thread[Gγ] + βF) + β^2F];
  m=Length[X];k=Length[X0];RR={};
```

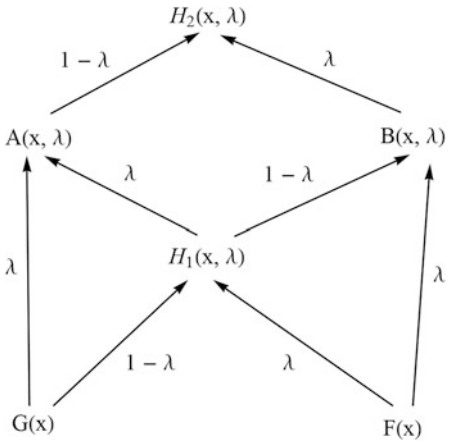


Fig. 2.16 The De Casteljau's algorithm

```
Do[
 X0L = {X0[[j]]};
 Do[AppendTo[X0L,
 Map[#[[2]], FindRoot[H/.β → λ0[[i + 1]],
  MapThread[{#1, #2}, {X, X0L[[i]]}]]]], {i, 1, n}];
R = {};

Do[AppendTo[R, Interpolation[
 MapThread[{#1, #2[[i]]}, {λ0, X0L}]]], {i, 1, m}];
AppendTo[RR, {Map[Chop[N[#[1]]], R], R}],
 {j, 1, k}];
sol = Transpose[RR];
{sol[[1]],
 Table[MapThread[#1[λ] - > #2[λ], {X, sol[[2, i]]}],
  {i, 1, Length[X0]}]}]
```

2.10.3 Comparing Linear and Quadratic Homotopy

Let us consider this simple polynomial system

```
Clear[f1, f2]
f1[x_, y_] := x^2 - 2x - y + 1/2
f2[x_, y_] := x^2 + 4y^2 - 4
```

First we solve it with numerical Gröbner basis

```
AbsoluteTiming[sol = NSolve[{f1[x,y], f2[x,y]}, {x,y}];]
{0.0195764, Null}
```

The solution

```
Sol
 {{x → 1.16077 - 0.654492i, y → - 0.902513 - 0.210444i},
 {x → 1.16077 + 0.654492i, y → - 0.902513 + 0.210444i},
 {x → - 0.222215, y → 0.993808}, {x → 1.90068, y → 0.311219}}
```

Let us substitute back the solutions into the system and compute the mean of the error norms

```
Mean[Map[Norm[#], {f1[x,y], f1[x,y]}/.sol]]
4.71401 × 10^-15
```

Now let us carry out the computation with linear homotopy. The system is

```
F = {f1[x,y], f2[x,y]}
{1/2 - 2x + x^2 - y, - 4 + x^2 + 4y^2}
```

The variables

```
X = {x,y}
{x,y}
```

The degree of the polynomials

```
d = {2,2}
{2,2}
```

We generate a start system

```
sol = StartingSystem[F,X,d];
G = sol[[1]]
(- 0.419162 - 0.907912 i) ((- 0.657419 + 0.753525 i) + x^2),
 (0.609013 - 0.79316 i) ((- 0.278254 + 0.960507 i) + y^2)
```

with its solutions

```
X0 = sol[[2]]
{{- 0.910335 + 0.413873 i, 0.799454 - 0.600727 i},
 {- 0.910335 + 0.413873 i, - 0.799454 + 0.600727 i},
 {0.910335 - 0.413873 i, 0.799454 - 0.600727 i},
 {0.910335 - 0.413873 i, - 0.799454 + 0.600727 i}}
```

Now, we do not need γ trick,

```
γ = {1, 1};
AbsoluteTiming[sol10L = LinearHomotopyFR[F, G, X, X0, γ, 5];]
{0.0210878, Null}
```

The solution

```
sol10L[[1]]
{{-0.222215, 0.993808}, {1.16077 + 0.654492i, -0.902513 + 0.210444i},
 {-0.222215, 0.993808}, {1.16077 - 0.654492i, -0.902513 - 0.210444i}}
```

Back substitution

```
error10L = Map[F/.{x → #[[1]], y → #[[2]]}, sol10L[[1]]]
{{0., 4.44089 × 10^-16},
 {2.22045 × 10^-16 - 4.44089 × 10^-16 i, 0. - 4.44089 × 10^-16 i},
 {0., 4.44089 × 10^-16}, {0. + 0. i, 0. + 2.22045 × 10^-16 i}}
```

The average of error norms

```
Mean[Map[Norm[#], error10L]]
4.44089 × 10^-16
```

Now we try quadratic homotopy

```
AbsoluteTiming[sol10NL = NonLinearHomotopyFR[F, G, X, X0, γ, 5];]
{0.0304908, Null}
```

The solution

```
sol10NL[[1]]
{{-0.222215, 0.993808}, {1.16077 + 0.654492 i, -0.902513 + 0.210444 i},
 {-0.222215, 0.993808}, {1.16077 - 0.654492 i, -0.902513 - 0.210444 i}}
```

Back substitution

```
error10NL = Map[F/.{x → #[[1]], y → #[[2]]}, sol10NL[[1]]]
0.0, -4.44089 × 10^-16, 0.0 + 0.0 i, -4.44089 × 10^-16 + 0.0 i,
 1.11022 × 10^-16, -4.44089 × 10^-16, 0.0 + 0.0 i, 0.0 + 2.22045 × 10^-16 i
```

The average of error norms

```
Mean[Map[Norm[#],error10NL]]
3.91995 × 10^-16
```

In order to improve the result of the linear homotopy we need smaller step size, which means more steps,

```
AbsoluteTiming[sol10L = LinearHomotopyFR[F,G,X,X0,γ,300];]
{0.53911,Null}
```

The solution

```
sol10L[[1]]
{{1.90068,0.311219},{1.16077 + 0.654492i, - 0.902513 + 0.210444i},
 { - 0.222215,0.993808},{1.16077 - 0.654492i, - 0.902513 - 0.210444i}}
```

Back substitution

```
error10L = Map[F/.{x → #[[1]],y → #[[2]]},sol10L[[1]]]
{{ - 4.44089 × 10^-16, - 7.21645 × 10^-16}, {0.0 + 0.0 i, 0.0 + 0.0 i,}
 {0.0, 4.44089 × 10^-16}, {0. + 0.0 i, 0. + 0.0 i}}
```

Now the average of error norms is smaller but the running time longer

```
Mean[Map[Norm[#],error10L]]
3.22858 × 10^-16
```

In this example the quadratic homotopy gives higher precision than the linear one or linear homotopy requires longer computation time to reach the same error limit. However, until now, only restricted computational experiments are at our disposal!

2.11 Applications

2.11.1 Nonlinear Heat Conduction

Let us consider again a 1D heat conduction problem

$$\frac{d}{dx}\left(\lambda(\theta)\frac{d\theta}{dx}\right) = 0.$$

The boundary conditions are

$$\theta(0)= 0 \quad \text{and} \quad \theta(1)= 1.$$

The heat conduction coefficient depends on the temperature

$$\lambda(\theta) = 1 + k\theta.$$

Substituting $\lambda(\theta)$ into the differential equation and expanding it, we get

```
D[(1 + k θ[x]) D[θ[x],x],x]//Expand
kθ'[x]^2 + θ''[x] + kθ[x]θ''[x]
```

Let us employ finite difference method. The equation for the *i*-th node

```
eq = k((θ_{i+1} - θ_i)/Δ)^2 + (1 + kθ_i)(θ_{i+1} - 2θ_i + θ_{i-1})/Δ^2;
```

Considering six nodes and numbering them with 1, …, 6, the boundary conditions are

```
θ_1 = 0; θ_6 = 1;
```

Let k = 70 and the length of the interval $\Delta x = 1/(6-1) = 1/5$

```
data = {k → 70, Δ → 1/5};
```

Then the equations for the internal nodes are

```
eqs = Table[eq,{i,2,5}]/.data//Expand
 {-50 θ_2 - 1750 θ_2^2 + 25 θ_3 - 1750 θ_2 θ_3 + 1750 θ_3^2,
  25 θ_2 - 50 θ_3 + 1750 θ_2 θ_3 - 1750 θ_3^2 + 25 θ_4 - 1750 θ_3 θ_4 + 1750 θ_4^2,
  25 θ_3 - 50 θ_4 + 1750 θ_3 θ_4 - 1750 θ_4^2 + 25 θ_5 - 1750 θ_4 θ_5 + 1750 θ_5^2,
  1775 + 25 θ_4 - 1800 θ_5 + 1750 θ_4 θ_5 - 1750 θ_5^2}
```

If we employ linear homotopy, then the equations are

```
F = eqs;
```

The variables

```
X = Table[[θ_i,{i,2,5}]
{θ_2,θ_3,θ_4,θ_5}
```

The degree of the equations

```
d = {2,2,2,2}
```

The starting system

```
sol = StartingSystem[F,X,d];
G = sol[[1]];
```

The initial values of the starting system

```
X0 = sol[[2]];
```

Since the starting system is complex

```
γ = {1,1,1,1};
```

Let us employ the direct path tracing method

```
sol = LinearHomotopyFR[F,G,X,X0,γ,100,λ];
```

Cutting imaginary tails,

```
sol1FR = Chop[sol[[1]],10^-8];
```

Considering the positive solutions

```
sol2 = Select[
 sol1FR,(#[[1]] > 0) ∧ (#[[2]] > 0) ∧ (#[[3]] > 0) ∧ (#[[4]] > 0)&]
{{0.348022, 0.5655, 0.734005, 0.875685}}
```

Let us visualize the temperature profile (Fig. 2.17)

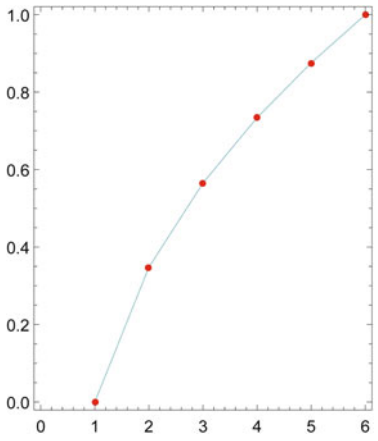


Fig. 2.17 Temperature profile

We can find a natural starting system in this case, namely using the linear heat conduction problem, $k = 0$. The equations of the starting system are,

```
eqsL = Table[eq, {i, 2, 5}] /. {k → 0, Δ → 1/5} // Expand
{-50 θ2 + 25 θ3, 25 θ2 - 50 θ3 + 25 θ4, 25 θ3 - 50 θ4 + 25 θ5, 25 + 25 θ4 - 50 θ5}
```

The initial conditions are the solutions of this linear system

```
solL = NSolve[eqsL, X] // Flatten
{θ2 → 0.2, θ3 → 0.4, θ4 → 0.6, θ5 → 0.8}
```

The degree of the original system is $d = 2^4 = 16$. Formally therefore, we need to create 16 initial values for the 16 paths,

```
X0 = Table[X /. solL, {i, 1, 16}];
sol = LinearHomotopyFR[F, G, X, X0, γ, 100, λ];.
```

The solutions are

```
{0.348022, 0.565500, 0.7340052, 0.875685}
```

The result using complex starting system was the same.

2.11.2 Local Coordinates via GNSS

Throughout history, position determination has been one of the most important tasks of mountaineers, pilots, sailor, civil engineers etc. In modern times, Global Navigation Satellite Systems (GPS) provide an ultimate method to accomplish this task. If one has a hand held GPS receiver, which measures the travel time of the signal transmitted from the satellites, the distance traveled by the signal from the satellites to the receiver can be computed by multiplying the measured time by the speed of light in vacuum. The distance of the receiver from the i-th GPS satellite, the pseudo-range observations, d_i is related to the unknown position of the receiver, X, Y, Z by

$$d_i = \sqrt{(a_i - X)^2 + (b_i - Y)^2 + (c_i - Z)^2} + \xi \quad i = 0, 1, 2, 3,$$

where $\{a_i, b_i, c_i\}$ are the coordinates of the satellite and ξ the adjustment of the satellite clock as an additional unknown variable. Let us determine the coordinates of a location on the Earth employing Global Positioning System (GNSS) $\{X, Y, Z\}$,

Now let us employ the variables: x_1, x_2, x_3 and x_4 representing the 4 unknowns $\{X, Y, Z, \xi\}$.

$$
\begin{aligned}
f1 &= x_1^2 + x_2^2 + x_3^2 - x_4^2 - 2x_1a_0 + a_0^2 - 2x_2b_0 + b_0^2 - 2x_3c_0 + c_0^2 + 2x_4d_0 - d_0^2,\\
f2 &= x_1^2 + x_2^2 + x_3^2 - x_4^2 - 2x_1a_1 + a_1^2 - 2x_2b_1 + b_1^2 - 2x_3c_1 + c_1^2 + 2x_4d_1 - d_1^2,\\
f3 &= x_1^2 + x_2^2 + x_3^2 - x_4^2 - 2x_1a_2 + a_2^2 - 2x_2b_2 + b_2^2 - 2x_3c_2 + c_2^2 + 2x_4d_2 - d_2^2,\\
f4 &= x_1^2 + x_2^2 + x_3^2 - x_4^2 - 2x_1a_3 + a_3^2 - 2x_2b_3 + b_3^2 - 2x_3c_3 + c_3^2 + 2x_4d_3 - d_3^2.
\end{aligned}
$$

We construct the starting system keeping only one of the unknown variables from each equation, in the following way:

$$
\begin{aligned}
g1 &= x_1^2 - 2x_1a_0 + a_0^2 + b_0^2 + c_0^2 - d_0^2,\\
g2 &= x_2^2 + a_1^2 - 2x_2b_1 + b_1^2 + c_1^2 - d_1^2,\\
g3 &= x_3^2 + a_2^2 + b_2^2 - 2x_3c_2 + c_2^2 - d_2^2,\\
g4 &= -x_4^2 + a_3^2 + b_3^2 + c_3^2 + 2x_4d_3 - d_3^2.
\end{aligned}
$$

Let us use the following numerical data, see Awange et al. (2010) page 180, Table 12.1. Then the system to be solved is,

```
F = {f1, f2, f3, f4}/.data
    {1.0305 × 10^14 - 2.9665 × 10^7 x1 + x1^2 + 4.0933 × 10^7 x2
     + x2^2 + 1.4857 × 10^7 x3 + x3^2 + 4.8621 × 10^7 x4 - x4^2, 1.9504 × 10^14
     + 3.16000 × 10^7 x1 + x1^2 + 2.6602 × 10^7 x2
     + x2^2 - 3.4268 × 10^7 x3 + x3^2 + 4.5829 × 10^7 x4 - x4^2, 2.8173 × 10^14
     - 396964.82 x1 + x1^2 + 2.3735 × 10^7 x2
     + x2^2 - 4.7434 × 10^7 x3 + x3^2 + 4.1258 × 10^7 x4 - x4^2, 1.6464 × 10^14
     + 2.4960 × 10^7 x1 + x1^2 + 4.6765 × 10^7 x2
     + x2^2 - 6556945.4 x3 + x3^2 + 4.6845 × 10^7 x4 - x4^2}
```

The starting system

```
G = {g1, g2, g3, g4}/.data
{1.03055 × 10^14 - 2.96646 × 10^7 x1 + x1^2, 1.95045 × 10^14
 + 2.66023 × 10^7 x2 + x2^2, 2.81726 × 10^14 - 4.74338 × 10^7 x3
 + x3^2, 1.64642 × 10^14 + 4.68448 × 10^7 x4 - x4^2}
```

The initial values for the starting system can be computed via solving the independent equations of the starting system

```
X = {x1, x2, x3, x4};
Transpose[Partition[Map[ #[[2]]&, Flatten[MapThread
[Nsolve[#1 = = 0, #2]&, {G, X}]]], 2]]{{4.01833 × 10^6,
 - 1.33011×10^7 - 4.25733 × 10^6 i, 6.96083*10^6, - 3.28436 × 10^6},
 {2.56463 × 10^7, - 1.33011 × 10^7 + 4.25733 × 10^6 i,
  4.0473 × 10^7, 5.01291 × 10^7}}
```

Now we use the γ trick

```
γ = i{1, 1, 1, 1};
```

Employing linear homotopy

```
sol = LinearHomotopyFR[F, G, X, X0, γ, 100, λ];
```

The solution,

```
{1.11159 × 10^6, - 4.34826 × 10^6, 4.52735 × 10^6, 100.001}
```

using more digits

```
NumberForm[%, 16]
{1.111590459962204 × 10^6, - 4.348258630909095 × 10^6,
 4.527351820245796 × 10^6, 100.000550673451}
```

Let us compute the solution via Gröbner basis,

```
NSolve[F, X]
{{x1 → - 2.89212 × 10^6, x2 → 7.56878 × 10^6,
  x3 → - 7.20951 × 10^6, x4 → 5.74799 × 10^7}, {x1 → 1.11159 × 10^6
  x2 →  - 4.34826 × 10^6, x3 → 4.52735 × 10^6, x4 → 100.001}}
```

using 16 digits,

```
{x1 → 1.111590459962204 × 10^6, x2 → - 4.348258630909095 × 10^6,
 x3 → 4.527351820245798 × 10^6, x4 → 100.0005506748347 }
```

we have the same result.

2.12 Exercises

2.12.1 GNSS Positioning N-Point Problem

Problem

In case of more than 4 satellites, $m > 4$, let us consider two distance representations. The algebraic one is,

$$f_i = (x_1 - a_i)^2 + (x_2 - b_i)^2 + (x_3 - c_i)^2 - (x_4 - d_i)^2,$$

and the geometrical one,

$$g_i = d_i - \sqrt{(x_1 - a_i)^2 + (x_2 - b_i)^2 + (x_3 - c_i)^2} - x_4,$$

The results will be not equivalent in least square sense, namely

$$\min_{x_1,x_2,x_3,x_4} \sum_{i=1}^{m} f_i \neq \min_{x_1,x_2,x_3,x_4} \sum_{i=1}^{m} g_i^2,$$

Let us consider six satellites with the following numerical values,

```
datan = {a0 → 14177553.47, a1 → 15097199.81, a2 → 23460342.33,
 a3 → -8206488.95, a4 → 1399988.07, a5 → 6995655.48,
 b0 → -18814768.09, b1 → -4636088.67, b2 → -9433518.58,
 b3 → -18217989.14, b4 → -17563734.90, b5 → -23537808.26,
 c0 → 12243866.38, c1 → 21326706.55, c2 → 8174941.25,
 c3 → 17605231.99, c4 → 19705591.18, c5 → -9927906.48,
 d0 → 21119278.32, d1 → 22527064.18, d2 → 23674159.88,
 d3 → 20951647.38, d4 → 20155401.42, d5 → 24222110.91};
```

Solution

The number of the equations,

```
m = 6;
```

The prototype form of the error of the i-th equation is,

```
en = di - √((x1 - ai)^2 + (x2 - bi)^2 + (x3 - ci)^2) - x4;
```

then

```
Map[D[en^2,#]&,x1,x2,x3,x4]
```

$$\left\{ -\frac{2(x1-a_i)\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)}{\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}},\right.$$

$$-\frac{2(x2-b_i)\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)}{\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}},$$

$$-\frac{2(x3-c_i)\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)}{\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}},$$

$$\left.-2\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)\right\}$$

in general form,

```
v=Map[∑_{i=0}^{m-1} #&,%]
```

$$\left\{ \sum_{i=0}^{-1+m} -\frac{2(x1-a_i)\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)}{\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}},\right.$$

$$\sum_{i=0}^{-1+m} -\frac{2(x2-b_i)\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)}{\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}},$$

$$\sum_{i=0}^{-1+m} -\frac{2(x3-c_i)\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)}{\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}},$$

$$\left.\sum_{i=0}^{-1+m} -2\left(-x4-\sqrt{(x1-a_i)^2+(x2-b_i)^2+(x3-c_i)^2}+d_i\right)\right\}$$

In our case $m = 6$, therefore the numeric form of the equations,

```
vn=v/. m→m/.datan//Expand;
```

For example the first equation`vn[[1]]`

$$
\begin{aligned}
&-1.05849\times10^{8}+12x1+\\
&6.801919\times10^{14}/\left(\sqrt{\left((-1.50972\times10^{7}+x1)^{2}+(463609.67+x2)^{2}+(-2.13267\times10^{7}+x3)^{2}\right)}\right)-\\
&(4.50541\times10^{7}x1)/\left(\sqrt{\left((-1.50972\times10^{7}+x1)^{2}+(463609.67+x2)^{2}+(-2.13267\times10^{7}+x3)^{2}\right)}\right)+\\
&5.64346\times10^{13}/\left(\sqrt{\left((-139999.07+x1)^{2}+(1.75637\times10^{7}+x2)^{2}+(-1.97056\times10^{7}+x3)^{2}\right)}\right)-\\
&(4.03108\times10^{7}x1)/\left(\sqrt{\left((-139999.07+x1)^{2}+(1.75637\times10^{7}+x2)^{2}+(-1.97056\times10^{7}+x3)^{2}\right)}\right)-\\
&3.43879\times10^{14}/\left(\sqrt{\left((820649.95+x1)^{2}+(1.8218\times10^{7}+x2)^{2}+(-1.76052\times10^{7}+x3)^{2}\right)}\right)-\\
&(4.19033\times10^{7}x1)/\left(\sqrt{\left((820649.95+x1)^{2}+(1.8218\times10^{7}+x2)^{2}+(-1.76052\times10^{7}+x3)^{2}\right)}\right)+\\
&5.98839\times10^{14}/\left(\sqrt{\left((-1.41776\times10^{7}+x1)^{2}+(1.88148\times10^{7}+x2)^{2}+(-1.22439\times10^{7}+x3)^{2}\right)}\right)-\\
&(4.22386\times10^{7}x1)/\left(\sqrt{\left((-1.41776\times10^{7}+x1)^{2}+(1.88148\times10^{7}+x2)^{2}+(-1.22439\times10^{7}+x3)^{2}\right)}\right)+\\
&1.11081\times10^{15}/\left(\sqrt{\left((-2.34603\times10^{7}+x1)^{2}+(9.43352\times10^{6}+x2)^{2}+(-8.17494\times10^{6}+x3)^{2}\right)}\right)-\\
&(4.73483\times10^{7}x1)/\left(\sqrt{\left((-2.34603\times10^{7}+x1)^{2}+(9.43352\times10^{6}+x2)^{2}+(-8.17494\times10^{6}+x3)^{2}\right)}\right)+\\
&3.38899\times10^{14}/\left(\sqrt{\left((-6.99566\times10^{6}+x1)^{2}+(2.35378\times10^{7}+x2)^{2}+(9.92791\times10^{6}+x3)^{2}\right)}\right)-\\
&(4.84442\times10^{7}x1)/\left(\sqrt{\left((-6.99566\times10^{6}+x1)^{2}+(2.35378\times10^{7}+x2)^{2}+(9.92791\times10^{6}+x3)^{2}\right)}\right)-\\
&(3.01944\times10^{7}x4)/\left(\sqrt{\left((-1.50972\times10^{7}+x1)^{2}+(4.63609\times10^{6}+x2)^{2}+(-2.13267\times10^{7}+x3)^{2}\right)}\right)+\\
&(2x1x4)/\left(\sqrt{\left((-1.50972\times10^{7}+x1)^{2}+(4.63609\times10^{6}+x2)^{2}+(-2.13267\times10^{7}+x3)^{2}\right)}\right)-\\
&(2.79998\times10^{6}x4)/\left(\sqrt{\left((-1.39999\times10^{6}+x1)^{2}+(1.75637\times10^{7}+x2)^{2}+(-1.97056\times10^{7}+x3)^{2}\right)}\right)+\\
&(2x1x4)/\left(\sqrt{\left((-1.39999\times10^{6}+x1)^{2}+(1.75637\times10^{7}+x2)^{2}+(-1.97056\times10^{7}+x3)^{2}\right)}\right)+\\
&(1.6413\times10^{7}x4)/\left(\sqrt{\left((8.20649\times10^{6}+x1)^{2}+(1.8218\times10^{7}+x2)^{2}+(-1.76052\times10^{7}+x3)^{2}\right)}\right)+\\
&(2x1x4)/\left(\sqrt{\left((8.20649\times10^{6}+x1)^{2}+(1.8218\times10^{7}+x2)^{2}+(-1.76052\times10^{7}+x3)^{2}\right)}\right)-\\
&(2.83551\times10^{7}x4)/\left(\sqrt{\left((-1.41776\times10^{7}+x1)^{2}+(1.88148\times10^{7}+x2)^{2}+(-1.22439\times10^{7}+x3)^{2}\right)}\right)+\\
&(2x1x4)/\left(\sqrt{\left((-1.41776\times10^{7}+x1)^{2}+(1.88148\times10^{7}+x2)^{2}+(-1.22439\times10^{7}+x3)^{2}\right)}\right)-\\
&(4.69207\times10^{7}x4)/\left(\sqrt{\left((-2.34603\times10^{7}+x1)^{2}+(9.43352\times10^{6}+x2)^{2}+(-8.17494\times10^{6}+x3)^{2}\right)}\right)+
\end{aligned}
$$

$$(2x1x4) / \left(\sqrt{\left((-2.34603\times10^{7}+x1)^{2} + (9.43352\times10^{6}+x2)^{2} + (-8.17494\times10^{6}+x3)^{2}\right)}\right) -$$
$$(1.39913\times10^{7}x4) / \left(\sqrt{\left((-6.99566\times10^{6}+x1)^{2} + (2.35378\times10^{7}+x2)^{2} + (9.92791\times10^{6}+x3)^{2}\right)}\right) +$$
$$(2x1x4) / \left(\sqrt{\left((-6.99566\times10^{6}+x1)^{2} + (2.35378\times10^{7}+x2)^{2} + (9.92791\times10^{6}+x3)^{2}\right)}\right)$$

We pick up a fairly wrong initial guess provided by the solution of a triplet subset,

```
X0 = {{596951.53, -4.8527796 × 10^6, 4.0887586 × 10^6, 3510.400237}}
{{596952.0, -4.85278 × 10^6, 4.08876 × 10^6, 3510.4}}
```

The variables,

```
V = {x1, x2, x3, x4};
```

The start system using fixed point homotopy,

```
gF = V - First[X0]
{-596952. + x1, 4.85278 × 10^6 + x2, -4.08876 × 10^6 + x3, -3510.4 + x4}
```

To avoid singularity of the homotopy function, let

```
γ = i{1, 1, 1, 1};
AbsoluteTiming[solH = LinearHomotopyFR[vn, gF, V, X0, γ, 10];]
{0.837211, Null}

solH[[1]]
{{596930., -4.84785 × 10^6, 4.08823 × 10^6, 15.5181}}

NumberForm[%, 15]
596929.65, -4.84785155 × 10^6, 4.0882268 × 10^6, 15.5180507
```

Displaying the homotopy paths,

```
paH = PathsV[V, solH[[2]], X0];
paH = Flatten[paH]
```

Now we try to use nonlinear homotopy, (Fig. 2.18)

```
AbsoluteTiming[solH = NonLinearHomotopyFR[vn, gF, V, X0, γ, 3];]
{0.0333693, Null}
```

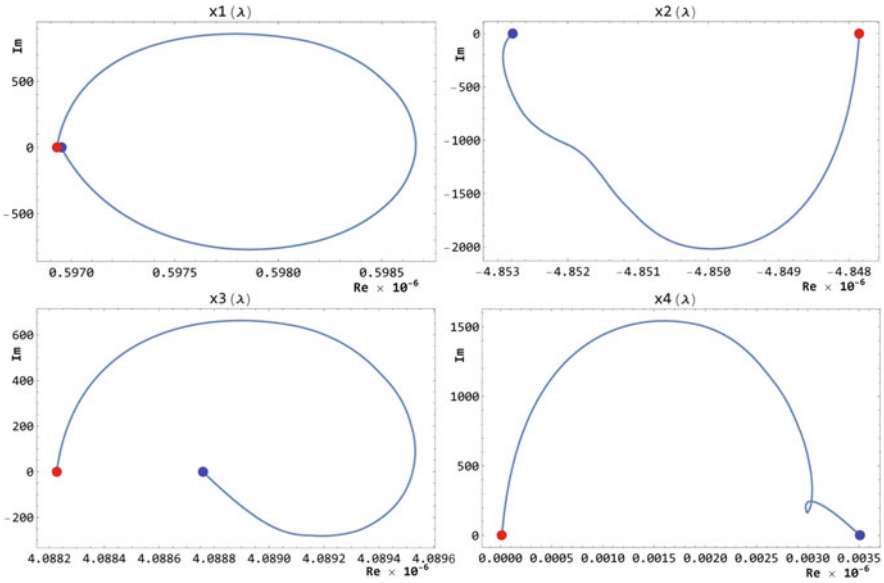


Fig. 2.18 The homotopy paths in case of linear homotopy

```
solH[[1]]
{{596930., -4.84785 × 10^6, 4.08823 × 10^6, 15.5181}}

NumberForm[%, 15]

596929.65, -4.84785155 × 10^6, 4.0882268 × 10^6, 15.5180507
```

Displaying the homotopy paths,

```
paH = PathsV[V, solH[[2]], X0];
paH = Flatten[paH]
```

Now the running time of the nonlinear homotopy is shorter (Fig. 2.19).

As demonstrated by these examples, the linear and nonlinear homotopy methods proved to be powerful solution tool in solving nonlinear geodetic problems, especially if it is difficult to find proper initial values to ensure convergence of local solution methods. These global numerical methods can be successful when symbolic computation based on Groebner basis or Dixon resultant fail because of the size and complexity of the problem. Since the computations along the different paths are independent, parallel computation can be successfully employed. In general, nonlinear homotopy gives more precise result or required less iteration steps and needs less computation time at a fixed precision than its linear counterpart.

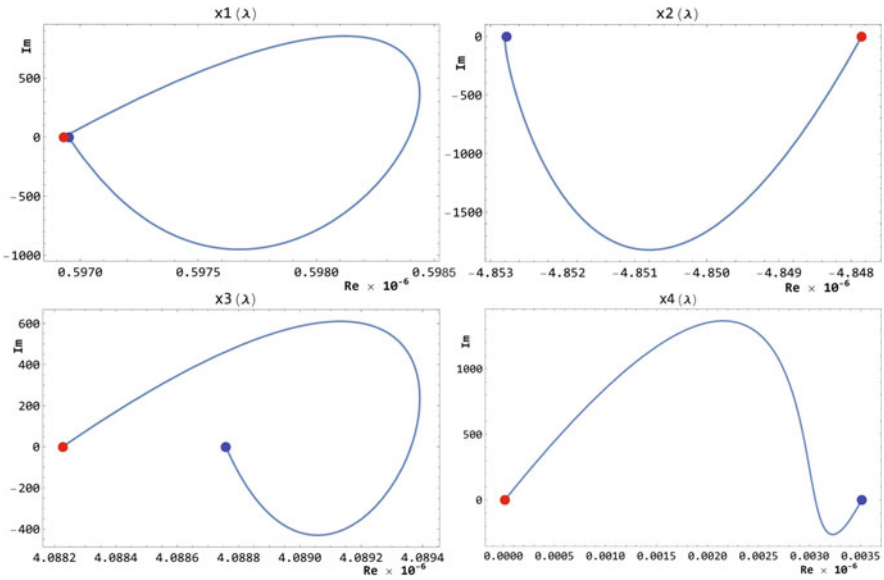


Fig. 2.19 The homotopy paths in case of nonlinear homotopy

References

Jalali F, Seader JD (2000) Use of homotopy-continuation method in stability analysis of multiphase reacting systems. Comput Chem Eng 24:1997–2000

Nor HM, Ismail AIM, Majid AA (2013) Quadratic Bezier homotopy function for solving system of polynomial equations. Matematika 29(2):159–171

Chapter 3
Overdetermined and Underdetermined Systems

3.1 Concept of the Over and Underdetermined Systems

As we know, overdetermined systems have no solution while underdetermined systems have infinitely many solutions. From an engineering point of view we can consider the solution of an overdetermined system to be the minimum of the sum of the squares of the errors, while in the case of an underdetermined system, we look for the solution having the minimal norm.

3.1.1 Overdetermined Systems

Let us consider the following system

$$\begin{aligned} f_1 &= x^2 + 2x - 1, \\ f_2 &= 5xy - 1 + x^3, \\ f_3 &= y^2 - 12 + 10\sin(x). \end{aligned}$$

Now, we have more equations than unknowns. These equations have pairwise solutions, ((f_1, f_2): solid line, (f_2, f_3): dashed line, (f_1, f_3): dotted line), but there is no solution for the three equations simultaneously, see Fig. 3.1.

We may consider the solution in the sense of the least squares, namely

$$\sum_{i=1}^{3} f_i^2(x, y) \to \min_{x,y}.$$

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_3

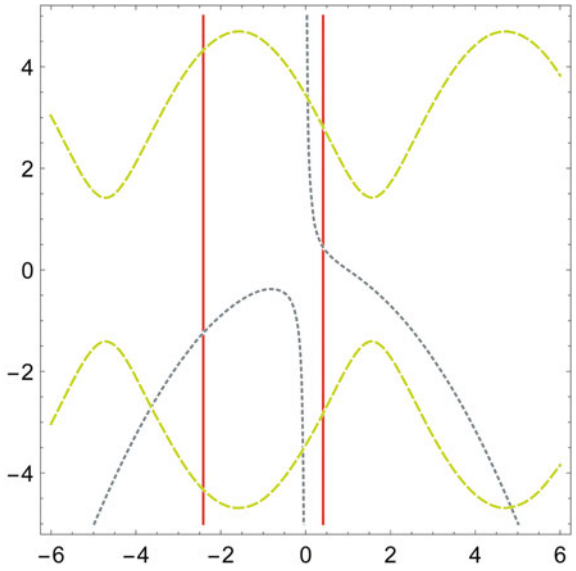


Fig. 3.1 Overdetermined system

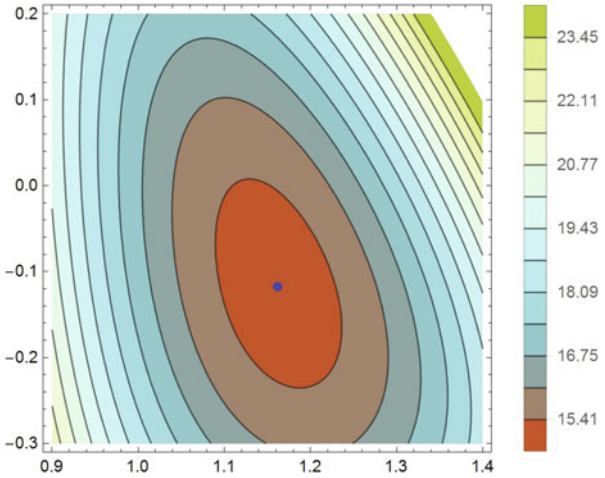


Fig. 3.2 Solution of the overdetermined system

In our case the minimization results in a single local minimum, see Fig. 3.2.

```
sol = NMinimize[f1^2 + f2^2 + f3^2, {x, y}]
{15.0617, {x → 1.16201, y → -0.117504}}
```

3.1.2 Underdetermined Systems

In this case, we have less equations than unknown variables, i.e.,

$$f_1 = x^2 + 2x + 2yz^2 - 5,$$
$$f_2 = 5xy - 1 + x^3 + xz.$$

Let us visualize the system for different z values, see Fig. 3.3.
It is clear that there are infinitely many solutions. We consider the solution, which has minimal norm,

$$\|\{x, y, z\}\| \to \min_{x,y,z}$$

under the constraint

$$f_i(x, y, z) = 0, \quad i = 1, 2, \ldots n.$$

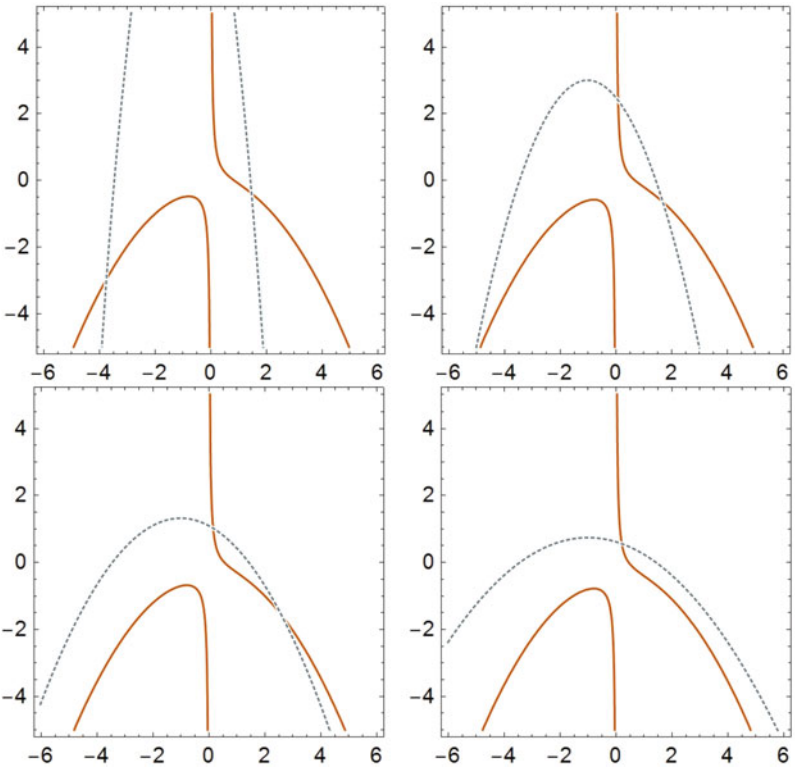


Fig. 3.3 The graphs of the solutions of the underdetermined system for different constant z values, z = (0.5, 1.0, 1.5, 2.0)

Namely, in our case the solution of this constrained minimization problem is

```
sol = NMinimize[{Norm[{x,y,z}], f1 = 0, f2 = 0}, {x,y,z},
  Method → "SimulatedAnnealing"]
{1.80538, {x → 0.288172, y → 0.975707, z → -1.49142}}
```

We employed here a stochastic global minimization technique, Simulated Annealing, which will be discussed later in the second part of the book (Fig. 3.4).

The solution of the over or underdetermined systems can be computed via minimization or constrained minimization, respectively. Here, we compute always the global minimum, therefore the solutions of such systems are unique.

3.2 Gauss–Jacobi Combinatorial Solution

This technique can be applied to overdetermined systems. Let us suppose that we have m equations with n unknowns, $m > n$. Then we can select $\binom{m}{n}$ number of determined subsystems size of $(n \times n)$. In case of linear systems, the sum of the weighted solutions of these subsystems will give exactly the solution of the overdetermined system.

Let us illustrate the algorithm with a small linear system,

$$\begin{aligned} a_1x + b_1y &= c_1, \\ a_2x + b_2y &= c_2, \\ a_3x + b_3y &= c_3. \end{aligned}$$

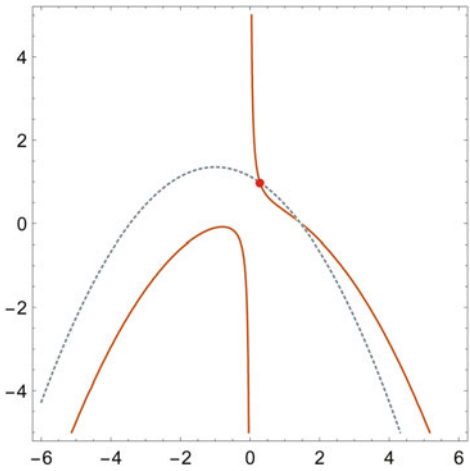


Fig. 3.4 Solution of the underdetermined system

The possible combinations of these equations are: {1, 2}, {1, 3} and {2, 3}. Let us suppose that the solutions of these subsystems are (x_{12}, y_{12}), (x_{13}, y_{13}), and (x_{23}, y_{23}). Then the solution of the overdetermined linear system is

$$x = \frac{\pi_{12}x_{12} + \pi_{13}x_{13} + \pi_{23}x_{23}}{\pi_{12} + \pi_{13} + \pi_{23}}$$

and

$$y = \frac{\pi_{12}y_{12} + \pi_{13}y_{13} + \pi_{23}y_{23}}{\pi_{12} + \pi_{13} + \pi_{23}},$$

where the $\pi_{i,j}$ weights can be computed from the square of the sub-determinants of the coefficient matrix of the overdetermined system as follows:

The transpose of the coefficient matrix is

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix}^T = \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix},$$

the squares of the different sub-determinants are,

$$\pi_{12} = \begin{Bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{Bmatrix}^2 = (a_1b_2 - a_2b_1)^2,$$

$$\pi_{13} = \begin{Bmatrix} a_1 & a_3 \\ b_1 & b_3 \end{Bmatrix}^2 = (a_1b_3 - a_3b_1)^2,$$

$$\pi_{23} = \begin{Bmatrix} a_2 & a_3 \\ b_2 & b_3 \end{Bmatrix}^2 = (a_2b_3 - a_3b_2)^2.$$

Let us see a numerical example.

```
eq1 = x + 2y = 3;
eq2 = 4x + 7y = 6;
eq3 = 5x + 8y = 9;
n = 2; m = 3;
```

The number of the combinations

```
mn = Binomial[m,n]
3
```

The actual combinations are

```
R = Subsets[Range[m],{n}]
{{1,2},{1,3},{2,3}}
```

The solutions of the determinant subsystems are,

```
sol12 = LinearSolve[(1 2; 4 7),(3; 6)];MatrixForm[sol12]
```

$$\begin{pmatrix} -9 \\ 6 \end{pmatrix}$$

```
sol13 = LinearSolve[(1 2; 5 8),(3; 9)];MatrixForm[sol13]
```

$$\begin{pmatrix} -3 \\ 3 \end{pmatrix}$$

```
sol23 = LinearSolve[(4 7; 5 8),(6; 9)];MatrixForm[sol23]
```

$$\begin{pmatrix} 5 \\ -2 \end{pmatrix}$$

Let us compute the weights. The transpose of the coefficient matrix is

```
Transpose[{{1 2},{4 7},{5 8}}]//MatrixForm
```

$$\begin{pmatrix} 1 & 4 & 5 \\ 2 & 7 & 8 \end{pmatrix}$$

then the weights are,

```
π12 = (1*7 - 4*2)^2
1
π13 = (1*8 - 5*2)^2
4
π23 = (4*8 - 5*7)^2
9
```

Therefore the solution of the overdetermined system can be computed as

```
x = (π12 sol12[[1]] + π13 sol13[[1]] +
  π23 sol23[[1]])/(π12 + π13 + π23)
```

$$\left\{\frac{12}{7}\right\}$$

```
y = (π12 sol12[[2]] + π13 sol13[[2]] +
  π23 sol23[[2]])/(π12 + π13 + π23){0}
```

Now, let us compare this result to the solution computed in sense of least squares. The system in matrix form is

$$\mathtt{A}=\begin{pmatrix}1 & 2\\ 4 & 7\\ 5 & 8\end{pmatrix};\mathtt{b}=\begin{pmatrix}3\\ 6\\ 9\end{pmatrix};$$

The rank of the coefficient matrix,

$$\mathtt{MatrixRank}\left[\begin{pmatrix}1 & 2\\ 4 & 7\\ 5 & 8\end{pmatrix}\right]$$

$$2$$

The rank of the coefficient matrix extended with right hand side vector (b)

$$\mathtt{MatrixRank}\left[\begin{pmatrix}1 & 2 & 3\\ 4 & 7 & 6\\ 5 & 8 & 9\end{pmatrix}\right]$$

$$3$$

These ranks are different; therefore there is no traditional solution. The solution can be computed in sense of least squares via pseudoinverse, $\mathbf{P}=\left(\mathbf{A}^{\mathbf{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathbf{T}}$,

```
P = Inverse[Transpose[A].A].Transpose[A];MatrixForm[P]
```

$$\begin{pmatrix}-\frac{23}{14} & -\frac{11}{7} & \frac{25}{14}\\ 1 & 1 & -1\end{pmatrix}$$

then the solution is

```
x = P.b;MatrixForm[x]
```

$$\begin{pmatrix}\frac{12}{7}\\ 0\end{pmatrix}$$

One can also use a built-in function of the pseudoinverse directly,

```
PseudoInverse[A]//MatrixForm
```

$$\begin{pmatrix}-\frac{23}{14} & -\frac{11}{7} & \frac{25}{14}\\ 1 & 1 & -1\end{pmatrix}$$

```
%.b//MatrixForm
```

$$\begin{pmatrix}\frac{12}{7}\\ 0\end{pmatrix}$$

3.3 Gauss–Jacobi Solution in Case of Nonlinear Systems

The Gauss–Jacobi method can be extended to nonlinear systems. Let us consider a simple example $n = 2$ and $m = 3$.

$$f_1(x, y) = 0,$$
$$f_2(x, y) = 0,$$
$$f_3(x, y) = 0.$$

In this case, the algorithm is the following:

Let us compute the solutions of the nonlinear subsystems via *resultant, Gröbner basis* or *homotopy* methods, namely $\eta_{12} = (x_{12}, y_{12})$, $\eta_{13} = (x_{13}, y_{13})$, and $\eta_{23} = (x_{23}, y_{23})$.

Compute the means of these solutions, $\eta = (x_s, y_s)$

$$\eta = \frac{1}{3}(\eta_{12} + \eta_{13} + \eta_{23}).$$

Let us linearize the overdetermined system at this point.
The linearized form of the j-th nonlinear equation at $\eta = (x_s, y_s)$, for $j = 1,.., 3$

$$f_j(\eta) + \frac{\partial f_j}{\partial x}(\eta)(x - x_s) + \frac{\partial f_j}{\partial y}(\eta)(y - y_s) = 0$$

or

$$\frac{\partial f_j}{\partial x}(\eta)x + \frac{\partial f_j}{\partial y}(\eta)y + \left(f_j(\eta) - \frac{\partial f_j}{\partial x}(\eta)x_s - \frac{\partial f_j}{\partial y}(\eta)y_s\right) = 0.$$

Let us recall the computation of the weight of the {1, 2} subsystem in linear case ($n = 2$ and $m = 3$)

$$\pi_{12} = \begin{Bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{Bmatrix}^2 = (a_{11}a_{22} - a_{21}a_{12})^2.$$

Similarly, the weight of the solution of the linearized subsystem {1, 2} is

$$\pi_{12}(\eta) = \begin{Bmatrix} \frac{\partial f_1}{\partial x}(\eta) & \frac{\partial f_1}{\partial y}(\eta) \\ \frac{\partial f_2}{\partial x}(\eta) & \frac{\partial f_2}{\partial y}(\eta) \end{Bmatrix}^2 = \left(\frac{\partial f_1}{\partial x}(\eta)\frac{\partial f_2}{\partial y}(\eta) - \frac{\partial f_2}{\partial x}(\eta)\frac{\partial f_1}{\partial y}(\eta)\right)^2.$$

Then computing the other two weights similarly, the approximate Gauss–Jacobi solution of the overdetermined nonlinear system is

$$x_{GJ} = \frac{\pi_{12}x_{12} + \pi_{13}x_{13} + \pi_{23}x_{23}}{\pi_{12} + \pi_{13} + \pi_{23}}$$

and

$$y_{GJ} = \frac{\pi_{12} y_{12} + \pi_{13} y_{13} + \pi_{23} y_{23}}{\pi_{12} + \pi_{13} + \pi_{23}}.$$

The precision of the solution can be increased by further linearization of the system at (x_{GJ}, y_{GJ}) and using a pseudoinverse solution.

Remarks Keep in mind that we did not need initial values!

It goes without saying that after computing the average of the solutions of the subsystems, one may employ standard iteration techniques; however there is no guarantee for convergence.

Let us illustrate the method via solving the following system,

$$\begin{aligned} f_1(x,y) &= x^2 - xy + y^2 + 2x, \\ f_2(x,y) &= x^2 - 2xy + y^2 + x - y - 2, \\ f_3(x,y) &= xy + y^2 - x + 2y - 3. \end{aligned}$$

We are looking for the real solution in least square sense

$$\sum_{i=1}^{3} \left(f_i^2(x,y)\right)^2 \to \min_{x,y}.$$

```
f1 = x^2 - xy + y^2 + 2x;
f2 = x^2 - 2xy + y^2 + x - y - 2;
f3 = xy + y^2 - x + 2y - 3;
```

The solutions of the subsystems {1, 2} is

```
sol12 = NSolve[{f1, f2}, {x, y}]
 {{x → - 2., y → 0.}, {x → - 2., y → 0.},
 {x → - 0.5 + 0.866025 i, y → - 1.5 + 0.866025 i},
 {x → - 0.5 - 0.866025 i, y → - 1.5 - 0.866025 i}}
```

The real solution,

```
xy12 = sol12[[1]]
{x → - 2., y → 0.}
```

The subsystem {1, 3} has two real solutions

```
sol13 = NSolve[{f1, f3}, {x, y}]
 {{x → - 2.43426, y → - 0.565741}, {x → - 0.5 + 0.866025 i, y → 1.},
    {x → - 0.5 - 0.866025 i, y → 1.}, {x → - 1.23241, y → - 1.76759}}
```

```
xy13a = sol13[[1]]
{x → -2.43426, y → -0.565741}
xy13b = sol13[[4]]
{x → -1.23241, y → -1.76759}
```

We choose the solution which gives the smaller residual, i.e.,

```
obj = f1^2 + f2^2 + f3^2
(-2 + x + x^2 - y - 2xy + y^2)^2 + (2x + x^2 - xy + y^2)^2 + (-3 - x + 2y + xy + y^2)^2
```

and

```
obj/.xy13a
0.14225
obj/.xy13b
1.38861
```

Therefore the first solution $\{x \to -2.43426, y \to -0.565741\}$ is chosen. Considering the subsystem {2, 3}, we get

```
sol23 = NSolve[{f2, f3}, {x, y}]
{{x → -2.5, y → -0.5}, {x → -1., y → 1.},
  {x → 2., y → 1.}, {x → -1., y → -2.}}
```

Now, we have four real solutions, the residuals are

```
Map[obj/.#&, sol23]
{0.0625, 1., 49., 1.}
```

Therefore we choose again the first solution since it has the smallest residual corresponding with the global minimum,

```
sol23a = sol23[[1]]
{x → -2.5, y → -0.5}
```

Let us compute the average of the solutions of the subsystems,

```
{xs, ys} = 1/3 Apply[Plus, Map[{x, y}/.#&, {xy12, xy13a, sol23a}]]
{-2.31142, -0.355247}
```

or

```
Mean[Map[{x, y}/.#&, {xy12, xy13a, sol23a}]]
{-2.31142, -0.355247}
```

Now we can compute the weights at $\{xs, ys\}$,

```
π12 = det[( D[f1,x]  D[f1,y] )]^2 /.{x → xs, y → ys}
                D[f2,x]  D[f2,y]
3.76967
π13 = det[( D[f1,x]  D[f1,y] )]^2 /.{x → xs, y → ys}
                D[f3,x]  D[f3,y]
20.1326
π23 = det[( D[f2,x]  D[f2,y] )]^2 /.{x → xs, y → ys}
                D[f3,x]  D[f3,y]
47.9295
```

Then the weighted solution is

```
{xy12, xy13, xy23} = Map[{x, y} /.#&, {xy12, xy13a, sol23a}]
{{-2., 0.}, {-2.43426, -0.565741}, {-2.5, -0.5}}
{xGJ, yGJ} = (π12xy12 + π13xy13 + π23xy23) / (π12 + π13 + π23)
{-2.45533, -0.492186}
```

Again, we did not need to guess an initial value, and no iteration was necessary! However, we may repeat (this is not a traditional numerical iteration) the algorithm.

Let us employ one step iteration, in order to improve the solution. The linear form of $f(x, y)$ at $(x, y) = (x0, y0)$ is

$$f(x,y) = f(\mathrm{x0},\mathrm{y0}) + \frac{\partial f(\mathrm{x0},\mathrm{y0})}{\partial x}(x - \mathrm{x0}) + \frac{\partial f(\mathrm{x0},\mathrm{y0})}{\partial y}(y - \mathrm{y0}).$$

Therefore the linearized system is

```
A = ( D[f1,x]  D[f1,y] ) /.{x → xGJ, y → yGJ}; MatrixForm[A]
    ( D[f2,x]  D[f2,y] )
    ( D[f3,x]  D[f3,y] )
( -2.41848   1.47096 )
(  -2.9263    2.9263 )
( -1.49219  -1.43971 )
b = ( D[f1,x]x + D[f1,y]y - f1 ) /.{x → xGJ, y → yGJ}; MatrixForm[b]
    ( D[f2,x]x + D[f2,y]y - f2 )
    ( D[f3,x]x + D[f3,y]y - f3 )
( 5.06243 )
( 5.85395 )
( 4.45073 )
```

Having A and b, the solution of the linearized system is

```
{xss,yss} = Flatten[LeastSquares[A,b]]
{ - 2.4646, - 0.500688}
```

Then the residual

```
obj/.{x → xss,y → yss}
0.0403161
```

Let us check the result via direct minimization of $f = f_1^2 + f_2^2 + f_3^2$.
There are two local minimums, see Fig. 3.5. We are looking for the global one, see Fig. 3.6.

```
NMinimize[obj,{x,y},Method → {RandomSearch,SearchPoints → 1000}]
{0.0403159,{x → - 2.46443,y → - 0.500547}}
```

This is really the global one, since the other minimum is

```
FindMinimum[obj,{x, - 1},{y,0.75}]
{0.631509,{x → - 1.09675,y → 0.835606}}
```

Comparing the global minimum to the Gaussian solution with one iteration step,

```
{xss,yss}
{ - 2.4646, - 0.500688}
```

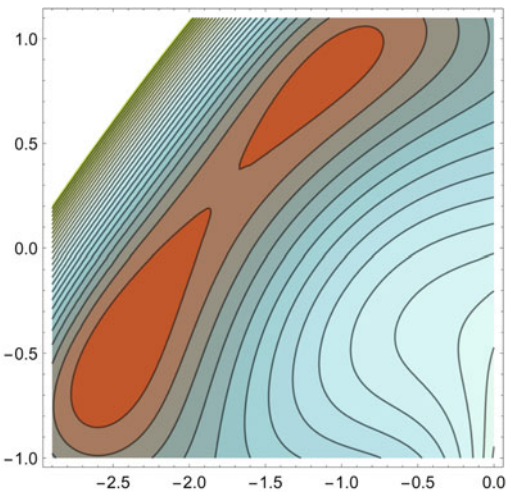


Fig. 3.5 The residual function of the nonlinear system

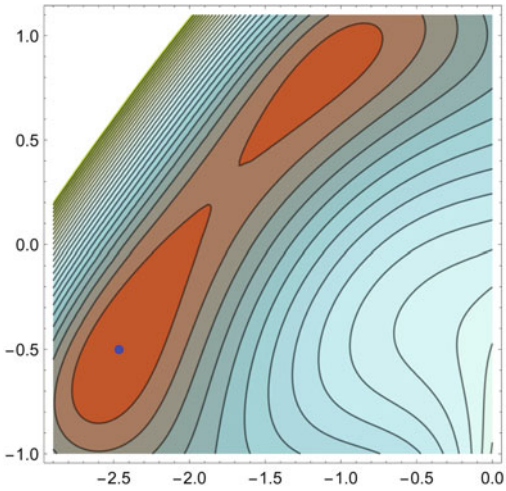


Fig. 3.6 The global minimum

Remark The main handicap of the Gauss–Jacobi method is that frequently, we have cases where $m \gg n$, which will result in too many subsystems (combinatorics runaway!).

The advantage is that the solutions of the subsystems can be computed in parallel.

3.4 Transforming Overdetermined System into a Determined System

Let us consider again our overdetermined system. We are looking for the solution in the least squares sense. Therefore, the solution should minimize the following objective function,

$$G(x,y) = f_1(x,y)^2 + f_2(x,y)^2 + f_3(x,y)^2.$$

The necessary condition of the optimum is

$$F_1(x,y) = \frac{\partial G}{\partial x}(x,y) = 0,$$

$$F_2(x,y) = \frac{\partial G}{\partial y}(x,y) = 0.$$

This is already a determined system.

```
F1 = D[obj,x]
2(1 + 2x - 2y)(-2 + x + x^2 - y - 2xy + y^2) +
  2(2 + 2x - y)(2x + x^2 - xy + y^2) + 2(-1 + y)(-3 - x + 2y + xy + y^2)
F2 = D[obj,y]
2(-1 - 2x + 2y)(-2 + x + x^2 - y - 2xy + y^2) +
  2(-x + 2y)(2x + x^2 - xy + y^2) + 2(2 + x + 2y)(-3 - x + 2y + xy + y^2)
```

Let us compute the real solutions of this algebraic system,

```
sol = NSolve[{F1,F2},{x,y},Reals]
{{x → -2.46443,y → -0.500547},
  {x → -1.76765,y → 0.282464},{x → -1.09675,y → 0.835606},
  {x → -1.8227,y → -1.28262},{x → -1.06936,y → -1.8973}}
```

The number of the real solutions is

```
Length[sol]
5
```

We choose the solution that provides the smallest residual

```
Map[obj/.#&,sol]
{0.0403159, 1.23314, 0.631509, 6.14509, 0.577472},
```

which is the first one,

```
sol[[1]]
{x → -2.46443,y → -0.500547}
```

3.5 Extended Newton–Raphson Method

The Newton–Raphson method solves a linear system by iteration. The coefficient matrix is the Jacobian of the linearized system, which is a square matrix with full rank when the nonlinear system is determined. Let us consider the nonlinear system $f(x)$, then, in the i-th iteration step, we have the linearization as

$$f(x_{i+1}) = f(x_i) + J(x_i)(x_{i+1} - x_i).$$

Then the linear system to be solved is $f(x_{i+1}) = 0$, which means we should compute x_{i+1} from the linear system,

$$J(x_i)\, x_{i+1} = J(x_i)x_i - f(x_i).$$

When the nonlinear system is over- or underdetermined, this linear system is over- or underdetermined, too. Therefore one can generalize or extend the Newton–Raphson method for over- or underdetermined systems by employing a pseudoinverse solution.

There is a *Mathematica* function to do that

```
≪Newton`NewtonExtended`
?NewtonExtended
```

```
Computes the solution of an overdetermined nonlinear system.
Input parameters:
 f   - list of functions of the system,
 x   - list of variables,
 x0  - list of the initial values,
 eps - error limit for the iteration, default value: 10^-12
 n   - maximum number of the iterations, default value: 100.
Output:
list of the iterative solutions
```

Let us employ this method to solve our overdetermined problem. Since this method is a local technique, we generate ten initial values in the region $[-2, 0] \times [-1, 1]$, randomly

```
iv = {RandomReal[{-2,0},10],RandomReal[{-1,1},10]}//Transpose
{{-0.211648,-0.843957},{-0.458807,-0.27978},
 {-0.907141,-0.160848},{-1.34021,-0.596869},
 {-0.806718,0.114513},{-1.60056,0.930948},{-0.825661,-0.738885},
 {-0.819798,-0.16371},{-0.893981,0.26915},{-1.53862,-0.121982}}
```

Then the solutions are,

```
soli = Map[Last[NewtonExtended[{f1,f2,f3},{x,y},#]]&,iv]
{{-1.06936,-1.8973},{-1.09675,0.835606},
 {-1.09675,0.835606},{-1.09675,0.835606},
 {-1.09675,0.835606},{-1.09675,0.835606},{-1.06936,-1.8973},
 {-1.09675,0.835606},{-1.09675,0.835606},{-2.46443,-0.500547}}
```

We choose the solution, which provide the smallest residual,

```
ob = Map[obj/.{x → #[[1]],y → #[[2]]}&,soli]
 {0.577472,0.631509,0.631509,0.631509,0.631509,
 0.631509,0.577472,0.631509,0.631509,0.0403159}
```

The position of this minimum in the list is

```
First[Position[ob,Min[ob]]//Flatten]
10.
```

Then the corresponding solution is

```
soli[[%]]
{-2.46443,-0.500547}
```

Alternatively, we can employ the average of the solutions of the subsystems (see Gauss–Jacobi solution) as an initial value,

```
{xs,ys}
{-2.31142,-0.355247}
```

namely,

```
soli=NewtonExtended[{f1,f2,f3},{x,y},{xs,ys}]//Last
{-2.46443,-0.500547}
```

Remark The solutions belong to different initial values and can be computed in parallel.

3.6 Solution of Underdetermined Systems

We have seen that this problem can be considered as a constrained minimization problem, which can be solved in different ways. Our system represents such a constraint. For example

```
f1=3x^2+2xy+2y^2+z+3u+xv-6;
f2=2x^2+x+y^2+10z+2u+yv-23;
f3=3x^2+xy+2y^2+2z+9u+zv-9;
f4=x^2+3y^2+2z+3u+uv-5;
```

and the objective function to be minimized is the norm of the solution,

```
G=x^2+y^2+z^2+u^2+v^2
```

3.6.1 Direct Minimization

We employ global constrained minimization. The solutions is

```
sol1 = NMinimize[{G, f1 == 0, f2 == 0, f3 == 0, f4 == 0}, {x, y, z, u, v}]
     {5.96532, {x → 0.975755, y → 0.0701982,
 z → 2.00317, u → 0.00669931, v → 0.997781}}
```

This is a solution satisfying the equations to a certain error,

```
{f1, f2, f3, f4}/.sol1[[2]]
{6.28357 × 10^-8, -1.39885 × 10^-7, -1.15531 × 10^-7, 4.31944 × 10^-8}
```

3.6.2 Method of Lagrange Multipliers

Introducing new variables λ_i, we can transform the constrained problem into an unconstrained one,

```
G = G + λ1f1 + λ2f2 + λ3f3 + λ4f4
u^2 + v^2 + x^2 + y^2 + z^2 + (-6 + 3u + vx + 3x^2 + 2xy + 2y^2 + z)λ1 +
 (-23 + 2u + x + 2x^2 + vy + y^2 + 10z)λ2 +
 (-9 + 9u + 3x^2 + xy + 2y^2 + 2z + vz)λ3 +
 (-5 + 3u + uv + x^2 + 3y^2 + 2z)λ4
```

The necessary conditions of the optimum is that every partial derivative is zero,

```
eqs = Map[D[G, #]&, {x, y, z, u, v, λ1, λ2, λ3, λ4}]
{2x + (v + 6x + 2y)λ1 + (1 + 4x)λ2 + (6x + y)λ3 +
 2xλ4, 2y + (2x + 4y)λ1 + (v + 2y)λ2 +
 (x + 4y)λ3 + 6yλ4, 2z + λ1 + 10λ2 + (2 + v)λ3 +
 2λ4, 2u + 3λ1 + 2λ2 + 9λ3 + (3 + v)λ4,
 2v + xλ1 + yλ2 + zλ3 + uλ4, -6 + 3u + vx +
 3x^2 + 2xy + 2y^2 + z, -23 + 2u + x + 2x^2 + vy + y^2 +
 10z, -9 + 9u + 3x^2 + xy + 2y^2 + 2z + vz, -5 +
 3u + uv + x^2 + 3y^2 + 2z}
```

This algebraic system can be solved via numerical Gröbner basis,

```
sol = NSolve[eqs, {x, y, z, u, v, λ1, λ2, λ3, λ4}];
```

Let us select the real solutions

```
sol2 =
 Select[sol,
  (Im[#[[1,2]]] == 0 ∧ Im[#[[2,2]]] == 0 ∧ Im[#[[3,2]]] == 0∧
   Im[#[[4,2]]] == 0 ∧ Im[#[[5,2]]] == 0)&]
{{x → 0.0532491, y → -2.15786, z → 3.14204, u → -2.1505, v → 4.09418,
λ1 → -5.62931, λ2 → -0.553225, λ3 → -0.395869, λ4 → 3.64499},
 {x → -0.237014, y → -0.137118, z → 2.07153, u → 1.03871, v → -2.28339,
λ1 → -0.607563, λ2 → -1.21588, λ3 → -0.100285, λ4 → 4.29743},
 {x → 0.975687, y → 0.0703823, z → 2.00318, u → 0.00670792,
v → 0.997813, λ1 → 0.416967, λ2 → -0.619598, λ3 → -1.18648,
λ4 → 2.66475}, {x → -1.38903, y → -0.0438894,
z → 2.11802, u → -0.308493, v → 0.796651, λ1 → -0.0577799,
λ2 → -0.630616, λ3 → -0.538566, λ4 → 1.81704}}
```

Finding the corresponding minimal norm can remove the extraneous solutions corresponding to local minimums. The lengths of the norm of the different real solutions are

```
ss = Map[Norm[{x,y,z,u,v}]/.#&,%]
{5.99321, 3.26482, 2.4424, 2.67342}
```

Then we choose the solution providing the smallest norm

```
sol2[[Position[ss,Min[ss]]//Flatten//First]]
{x → 0.975687, y → 0.0703823, z → 2.00318,
u → 0.00670792, v → 0.997813, λ1 → 0.416967,
λ2 → -0.619598, λ3 → -1.18648, λ4 → 2.66475}
```

This is a solution indeed since substituting back, we get

```
{f1,f2,f3,f4}/.%
{4.44089 × 10^-16, 0., 8.88178 × 10^-16, 8.88178 × 10^-16}
```

The solution resulting from the direct global minimization above was the same, see Sect. 3.6.1.

```
sol1
{5.96532, {x → 0.975755, y → 0.0701982, z → 2.00317,
  u → 0.00669931, v → 0.997781}}
```

Remark In case of non-algebraic system, we may employ the homotopy method discussed in the earlier chapter.

3.6.3 Method of Penalty Function

Let us introduce a parameter K as a weight of the norm of the constraints. Then our objective function is

```
F(K_) := u^2 + v^2 + x^2 + y^2 + z^2 + K(f1^2 + f2^2 + f3^2 + f4^2)
```

Employing an initial value for the parameter $K = 10$, we get

```
sol = NMinimize[F[10], {x, y, z, u, v}]
{5.75969, {x → 0.998177, y → 0.0639928, z → 1.9847,
 u → 0.037631, v → 0.79637}}
```

The constraints are not satisfied correctly,

```
{f1, f2, f3, f4}/.sol[[2]]
{0.0175328, -0.0317418, -0.0502123, 0.120914}
```

Let us increase the value of $K = 500$,

```
sol = NMinimize[F(500), {x, y, z, u, v},
 Method → {"RandomSearch", "SearchPoints" → 500}]
{5.96079, {x → 0.976156, y → 0.0702523, z → 2.0028,
  u → 0.00735433, v → 0.993576}}
```

This is a better, acceptable solution,

```
{f1, f2, f3, f4}/.sol[[2]]
{0.000415708, -0.000620085, -0.00118282, 0.0026602}
```

3.6.4 Extended Newton–Raphson

Let us guess five initial values randomly between $\{-1, 1\}$,

```
x0 = Table[1 - 2RandomReal[], {5}]
{0.89982, 0.676421, 0.376708, -0.842843, -0.867964}
```

Then the Extended Newton–Raphson can be employed again,

```
sol = NewtonExtended[{f1, f2, f3, f4}, {x, y, z, u, v}, x0]//Last
{1.01847, 0.673925, 2.03846, 1.02869, -4.43554}
Norm[sol]
5.13605
%^2
5.98607
```

Substituting the results back into the equation,

```
{f1,f2,f3,f4}/.MapThread[(#1->#2)&,{{x,y,z,u,v},sol}]
{1.77636×10^-15,3.552716×10^-15,-1.77636×10^-15,8.88178×10^-16}
```

Here parallel computation can be employed again.

3.7 Applications

3.7.1 *Geodetic Application—The Minimum Distance Problem*

The problem definition: Given a point in space $P(X, Y, Z)$ and an ellipsoid, find the closest ellipsoid point $p(x, y, z)$ to this P, see Fig. 3.7. We will assume the ellipsoid has a circular cross section on the xy-plane.

The distance to be minimized is

```
d=(X-x)^2+(Y-y)^2+(Z-z)^2;
```

The point p is on the surface of the ellipsoid, therefore the coordinates of point $p(x, y, z)$ should satisfy the following constraint,

$$\texttt{c} = \frac{\texttt{x}^2 + \texttt{y}^2}{\texttt{a}^2} + \frac{\texttt{z}^2}{\texttt{b}^2} - 1;$$

This optimization problem can be represented as an underdetermine problem, where the norm of the solution vector is the distance to be minimized. Let us solve this problem using the *Extended Newton–Raphson* method. Introducing new variables (α, β, γ)

```
denoting={x→X-α,y→Y-β,z→Z-γ};
```

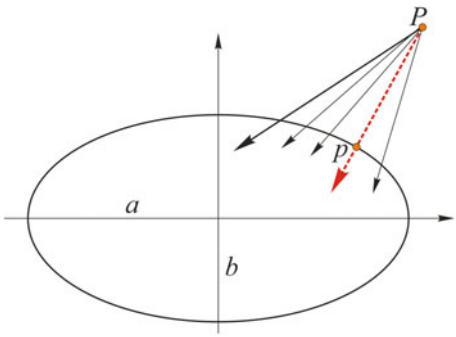


Fig. 3.7 The closest point problem

Then the constraint is,

```
eq = c/.denoting
(X - α)^2 + (Y - β)^2 / a^2 + (Z - γ)^2 / b^2 - 1
```

or

```
eq = eq a^2 b^2//Expand
 - a^2 b^2 + a^2 γ^2 + a^2 Z^2 - 2 a^2 γ Z + α^2 b^2 + b^2 β^2 + b^2 X^2 - 2 α b^2 X + b^2 Y^2 - 2 b^2 β Y
```

Using actual numerical data, let the values of the coordinates of *P* be

```
data = {X → 3770667.9989, Y → 446076.4896,
  Z → 5107686.2085, a → 6378136.602, b → 6356751.860};
eqn = eq/.data
4.04083 × 10^13 α^2 - 3.04733 × 10^20 α + 4.04083 × 10^13 β^2 -
 3.60504 × 10^19 β + 4.06806 × 10^13 γ^2 - 4.15568 × 10^20 γ + 2.33091 × 10^22
```

Normalizing the equation,

```
eqn = eqn/eqn[[1]]//Expand
1. - 0.0130735 α + 1.73358 × 10^-9 α^2 - 0.001546622 β +
 1.73358 × 10^-9 β^2 - 0.0178286 γ + 1.74527 × 10^-9 γ^2
```

Now, we have one equation with three unknowns (α, β, γ). So we have an underdetermined system of one equation with three unknowns. The norm of the solution vector (α, β, γ) should be minimized,

$$d = \alpha^2 + \beta^2 + \gamma^2$$

A reasonable initial guess can be (α, β, γ) = {0, 0, 0},

```
sol = Last[NewtonExtended({eqn}, {α, β, γ}, {0., 0., 0.})]
{26.6174, 3.14888, 36.2985}
```

Therefore, the coordinates of point *p*(x, y, z) are

```
sol = {X - α, Y - β, Z - γ}/.data/.
 {α → sol[[1]], β → sol[[2]], γ → sol[[3]]}
{3.77064 × 10^6, 446073., 5.10765 × 10^6}
```

Displayed in more digits as

```
NumberForm[%, 15]
{3.77064138151124 × 10^6, 446073.34071727, 5.10764991001691 × 10^6}
```

The solution cab now be visualized in Fig. 3.8.

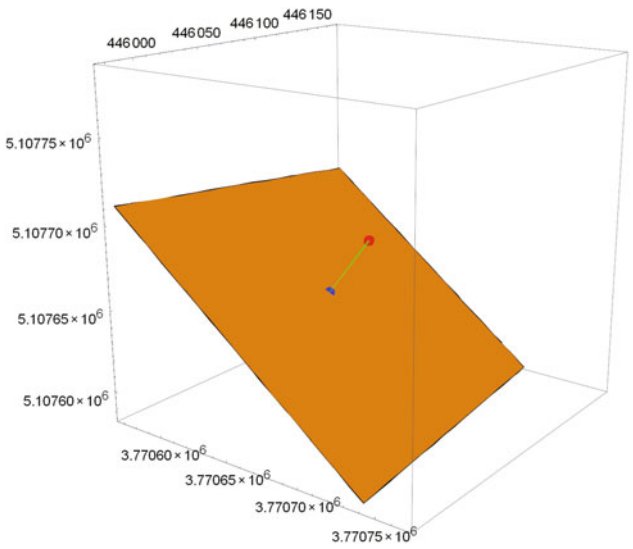


Fig. 3.8 Visualization of the solution of the minimum distance problem

3.7.2 Global Navigation Satellite System (GNSS) Application

Let us consider the satellite global positioning problem discussed and solved via homotopy method in the previous section. Recall that we need to determine the coordinates $\{X, Y, Z\}$ of a location on Earth employing the Global Navigation Satellite System (GNSS). The equations relating the satellite position and the unknown point to be positioned are given in terms of distances as

$$d_i = \sqrt{(a_i - X)^2 + (b_i - Y)^2 + (c_i - Z)^2} + \xi \quad i = 0, \ldots n > 3,$$

where $\{a_i, b_i, c_i\}$are the coordinates of the satellite and ξ the adjustment of the satellite clock as additional unknown variable.

We introduce new variables: x_1, x_2, x_3 and x_4 for the 4 unknowns$\{X, Y, Z, \xi,\}$. Since the number of the satellite is $n = 6 > 3$, we have an overdetermined system.

The numerical data are

```
datan = {a0 → 14177553.47, a1 → 15097199.81, a2 → 23460342.33,
  a3 → -8206488.95, a4 → 1399988.07, a5 → 6995655.48,
  b0 → -18814768.09, b1 → -4636088.67, b2 → -9433518.58,
  b3 → -18217989.14, b4 → -17563734.90, b5 → -23537808.26,
```

```
 c0 → 12243866.38, c1 → 21326706.55, c2 → 8174941.25,
 c3 → 17605231.99, c4 → 19705591.18, c5 → -9927906.48,
 d0 → 21119278.32, d1 → 22527064.18, d2 → 23674159.88,
 d3 → 20951647.38, d4 → 20155401.42, d5 → 24222110.91};
```

The number of satellites

```
m = 6;
```

The i-th equation is,

```
e = (x1 - ai)^2 + (x2 - bi)^2 + (x3 - ci)^2 - (x4 - di)^2;
```

Let us choose the first 4 equations as a determined subsystem,

```
eqs = Table[e, {i, 1, 4}] /. datan
 {(-1.50972 × 10^7 + x1)^2 + (4.63609 × 10^6 + x2)^2 +
 (-2.13267 × 10^7 + x3)^2 - (-2.25271 × 10^7 + x4)^2, (-2.34603 × 10^7 + x1)^2 +
 (9.43352 × 10^6 + x2)^2 + (-8.17494 × 10^6 + x3)^2 - (-2.36742 × 10^7 + x4)^2,
 (8.20649 × 10^6 + x1)^2 + (1.8218 × 10^7 + x2)^2 + (-1.76052 × 10^7 + x3)^2 -
 (-2.09516 × 10^7 + x4)^2, (-1.39999 × 10^6 + x1)^2 +
 (1.75637 × 10^7 + x2)^2 + (-1.97056 × 10^7 + x3)^2 - (-2.01554 × 10^7 + x4)^2}
```

and solve them via numerical Gröbner basis

```
solin = NSolve[eqs, {x1, x2, x3, x4}]
{{x1 → -2.68254 × 10^6, x2 → 1.16034 × 10^7,
  x3 → -9.36553 × 10^6, x4 → 6.1538 × 10^7},
  {x1 → 597005., x2 → -4.84797 × 10^6, x3 → 4.0883 × 10^6, x4 → 120.59}}
```

We shall use the second solution as an initial guess for the Extended Newton–Raphson method because it has a physical reality. The overdetermined system with the six equations is given as,

```
eqT = Table[e, {i, 0, 5}] /. datan
{(x1 - 1.41776 × 10^7)^2 + (x2 + 1.88148 × 10^7)^2 + (x3 - 1.22439 × 10^7)^2 -
 (x4 - 2.11193 × 10^7)^2, (x1 - 1.50972 × 10^7)^2 + (x2 + 4.63609 × 10^6)^2 +
 (x3 - 2.13267 × 10^7)^2 - (x4 - 2.25271 × 10^7)^2, (x1 - 2.34603 × 10^7)^2 +
 (x2 + 9.43352 × 10^6)^2 + (x3 - 8.17494 × 10^6)^2 - (x4 - 2.36742 × 10^7)^2,
 (x1 + 8.20649 × 10^6)^2 + (x2 + 1.8218 × 10^7)^2 + (x3 - 1.76052 × 10^7)^2 -
 (x4 - 2.09516 × 10^7)^2, (x1 - 1.39999 × 10^6)^2 + (x2 + 1.75637 × 10^7)^2 +
 (x3 - 1.97056 × 10^7)^2 - (x4 - 2.01554 × 10^7)^2, (x1 - 6.99566 × 10^6)^2 +
 (x2 + 2.35378 × 10^7)^2 + (x3 + 9.92791 × 10^6)^2 - (x4 - 2.42221 × 10^7)^2}
```

Let us employ Newton's method

```
sol = NewtonExtended
 [eqT,{x1,x2,x3,x4},{x1,x2,x3,x4}/.solin[[2]]]//Last
{596929., -4.84785 × 10^6, 4.08822 × 10^6, 13.4526}
NumberForm[%,12]
{596928.910449, -4.84784931442 × 10^6,
  4.08822444717 × 10^6,13.4525758581}
```

Now, let us check the results via direct minimization. The function to be minimized is

```
f = Apply[Plus,Table[e^2/.datan,i,0,m-1]]//Simplify
 ((-6.99566 × 10^6 + x1)^2 + (2.35378 × 10^7 + x2)^2 +
    (9.92791 × 10^6 + x3)^2 -
 1.(-2.42221 × 10^7 + x4)^2)^2 + ((-2.34603 × 10^7 + x1)^2 +
    (9.43352 × 10^6 + x2)^2 +
 (-8.17494 × 10^6 + x3)^2 - 1.(-2.36742 × 10^7 + x4)^2)^2 +
    ((-1.50972 × 10^7 + x1)^2 +
 (4.63609 × 10^6 + x2)^2 + (-2.13267 × 10^7 + x3)^2 -
    1.(-2.25271 × 10^7 + x4)^2)^2 +
 ((-1.41776 × 10^7 + x1)^2 + (1.88148 × 10^7 + x2)^2 +
    (-1.22439 × 10^7 + x3)^2 -
 1.(-2.11193 × 10^7 + x4)^2)^2 + ((8.20649 × 10^6 + x1)^2 +
    (1.8218 10^7 + x2)^2 +
 (-1.76052 × 10^7 + x3)^2 - 1.(-2.09516 × 10^7 + x4)^2)^2 +
    ((-1.39999 × 10^6 + x1)^2 +
 (1.75637 × 10^7 + x2)^2 + (-1.97056 × 10^7 + x3)^2 -
    1.(-2.01554 × 10^7 + x4)^2)^2
```

Then, the global minimization via the `NMinimize` becomes,

```
solN = NMinimize[f,{x1,x2,x3,x4}]//AbsoluteTiming
{0.0700622,{2.21338 × 10^18,
 {x1 → 596929.,x2 → -4.84785 × 10^6,x3 → 4.08822 × 10^6,x4 → 13.4526}}}
```

or

```
NumberForm[%,12]
{0.0700621592115, {2.21337940327 × 10^18,
 {x1 → 596928.910449, x2 → -4.84784931442 × 10^6,
 x3 → 4.08822444717 × 10^6, x4 → 13.4525758564 }}}
```

3.7.3 *Geometric Application*

The infrared depth camera of Microsoft Kinect XBOX (see, Fig. 3.9) can provides 2D RGB color image and the RGB data representing the depth—the distance of the object—in 11 bits (0.2048).

This low resolution causes discontinuities, which can be as much as 10 cm above 4 m object distance, and is about 2 cm when the object distance is less than 2.5 m. In our case, a spherical object having a radius $R = 0.152$ m was placed in the real world position $x = 0$, $y = 0$ and the object distance $z = 3$ m. Since the intensity threshold resolution is low, one may have quantized levels caused by round-off processes. Figure 3.10 represents the quantized depth data points simulated as synthetic data for illustration.

These points are those of the measured points of the sphere, but they have such layer type form because of the low depth resolution, see Fig. 3.11.

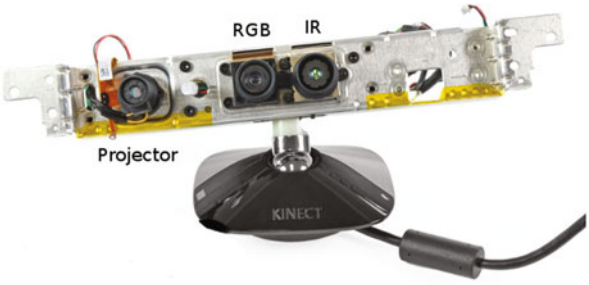


Fig. 3.9 Microsoft Kinect XBOX

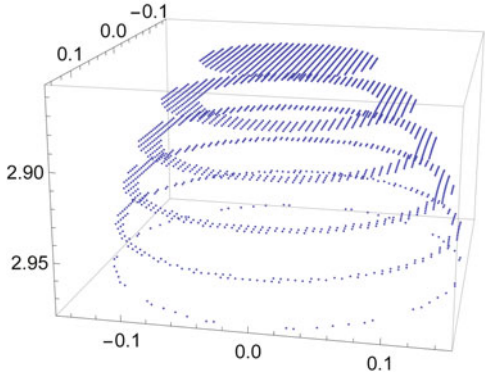


Fig. 3.10 Quantized depth data points

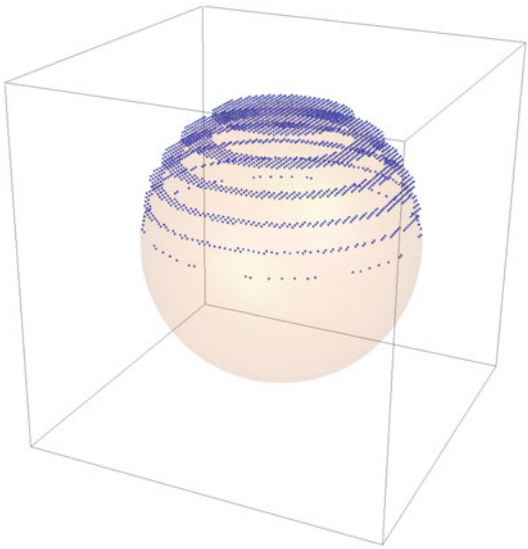


Fig. 3.11 The sphere with the measured points

Our task is to compute the position—coordinates of the center point (*a, b, c*)—and the radius of the sphere (*r*) from the measured data, the coordinates of the points.

In this case, we want to fit the given n points to a sphere such that the sum of the squared *algebraic distance* is minimized. The local error is defined as

$$\delta_i = (x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 - R^2,$$

where (x_i, y_i, z_i) are the coordinates of the measured points. First, let us suppose that the position as well as the radius of the sphere is unknown.

The parameters to be estimated are R, a, b and c. Let us introduce a new parameter,

$$d = a^2 + b^2 + c^2 - R^2.$$

Then δ_i can be expressed as

$$\delta_i = (-2x_i, -2y_i, -2z_i, 1) \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + x_i^2 + y_i^2 + z_i^2.$$

Indeed

```
{-2x_i, -2y_i, -2z_i, 1}.( a ) + x_i^2 + y_i^2 + z_i^2 =
                         ( b )
                         ( c )
                         (a^2 + b^2 + c^2 - R^2)
 {(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 - R^2}//Simplify
True
```

Consequently the parameter estimation problem is linear, namely one should solve a linear overdetermined system. In matrix form

$$Mp = h,$$

where

$$M = \begin{pmatrix} -2x_1 & -2y_1 & -2z_1 & 1 \\ \cdots & \cdots & \cdots & \cdots \\ -2x_i & -2y_i & -2z_i & 1 \\ \cdots & \cdots & \cdots & \cdots \\ -2x_n & -2y_n & -2z_n & 1 \end{pmatrix}, \quad p = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \quad \text{and} \quad h = \begin{pmatrix} -\left(x_1^2 + y_1^2 + z_1^2\right) \\ \cdots \\ -\left(x_i^2 + y_i^2 + z_i^2\right) \\ \cdots \\ -\left(x_n^2 + y_n^2 + z_n^2\right) \end{pmatrix}.$$

Then the solution in a least squares sense is

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = M^{-1} h,$$

where M^{-1} is the pseudoinverse of M.

The attractive feature of the algebraic parameter estimation is that its formulation results in a linear least squares problem, which has a non-iterative closed-form solution. Therefore the algebraic distance based fitting can be used as a quick way to calculate approximate parameter values without requiring an initial guess. First, we create the M matrix from data measured.

Then let us compute *a, b, c* and *d*

```
AbsoluteTiming[p = {a, b, c, d} = PseudoInverse[M].h]
{0.0386268, {1.54043 × 10^-15, -2.07993 × 10^-15, 2.99766, 8.96339}}
```

and the estimated radius of the sphere becomes

```
R = √(a^2 + b^2 + c^2 - d)
0.150345
```

Now, let us consider the *geometrical distance error*, which is the difference of the distance and not the square of the distance,

$$\Delta_i = \sqrt{(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2} - R \qquad i = 1,\ 2,\ \ldots n.$$

The equations can be easily generated. The prototype of the equation is,

```
G = Sqrt[(x - a)^2 + (y - b)^2 + (z - c)^2] - R;
```

Then our system is

```
eqs = Map[G/.{x → #[[1]], y → #[[2]], z → #[[3]]}&, dataQ];
```

and so we have 2241 nonlinear equations. For example, the first one is,

```
eqs[[1]]
Sqrt[(-0.14634 - a)^2 + (-0.0337707 - b)^2 + (2.97744 - c)^2] - R

Length[eqs]
2241
```

Now we employ Newton's method. The previous result employing algebraic distance can be used as an initial guess,

```
solNE = AbsoluteTiming[
 NewtonExtended[eqs, {a, b, c, R}, {0, 0, 2.99766, 0.150345}];]
{1.44983, Null}
```

The solution is

```
solNE//Last
{-1.71339 × 10^-19, 1.02352 × 10^-18, 3.0002, 0.152028}
```

The number of the iterations

```
Length[solNE]
7.
```

One may employ direct minimization, too. The objective function is

$$\rho = \sum_{i=1}^{n} \Delta_i^2 = \sum_{i=1}^{n} \left(\sqrt{(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2} - R \right)^2 .$$

Here we use local method to find the minimum

```
solLM = AbsoluteTiming[FindMinimum[Apply[Plus,
 Map[#^2&, eqs]], a, 0, b, 0, c, 2.99766, R, 0.150345]]
{0.605015, {0.0496067,
 {a → 5.93913 × 10^-12, b → -1.3497 × 10^-19, c → 3.0002, R → 0.152028}}}
```

Alternatively, we can employ global minimization, which does not require initial guess, however we need constrain of the region of R,

```
solGM = AbsoluteTiming[NMinimize[
 {Apply[Plus,Map[#^2&,eqs]],0 < R < 0.5},a,b,c,R]]
 {6.92959, {0.0496067,
 {a → 5.93913 × 10^-12, b → 2.72509 × 10^-11, c → 3.0002, R → 0.152028}}}
```

3.8 Exercises

3.8.1 Solution of Overdetermined System

Let us solve the following overdetermined system, transformation into determined system

$$f_1 = x^2 - xy + y^2 - 3,$$
$$f_2 = x^2 - 2xy + x + y^2 - y,$$
$$f_3 = xy - x + y^2 + 2y - 9.$$

via direct minimization, Gauss–Jacobi method and Extended Newton–Raphson

(a) The objective function for direct minimization is,

```
G = f1^2 + f2^2 + f3^2
(x + x^2 - y - 2xy + y^2)^2 + (-3 + x^2 - xy + y^2)^2 + (-9 - x + 2y + xy + y^2)^2
```

Let us employ global minimization,

```
solxy = NMinimize[G,{x,y}].
3.9443 × 10^-30,x → 1.,y → 2.
```

It means that there is a solution, which practically satisfies all of the three equations,

```
{f1,f2,f3}/.solxy[[2]]
{0.,8.88178 × 10^-16, -1.77636 × 10^-15}
```

(b) Now as second technique, we use Gauss–Jacobi combinatorial approach.

The solutions of the subsystems (1, 2) is

```
sol12 = NSolve[{f1,f2},{x,y}]
{{x → -2.,y → -1.},{x → 1.73205,y → 1.73205},
 {x → -1.73205,y → -1.73205},{x → 1.,y → 2.}}
```

We choose the solution, which gives the smaller residual. The residual is

```
G/.sol12
{36.,1.6077,22.3923,4.48862 × 10^-28}
```

This is the last one,

```
sol12op = sol12//Last
{x → 1.,y → 2.}
```

The subsystem (1, 3) has two real solutions

```
sol13 = NSolve[{f1,f3},{x,y},Reals]
{{x → 1.,y → 2.},{x → 1.3553,y → 1.95137}}
```

Similar way one can select the proper solution of the subsystem

```
G/.sol13
{2.68213 × 10^-29,0.0579703}
```

Therefore we pick up the first solution,

```
sol12op = sol13//First
{x → 1.,y → 2.}
```

Then for subsystem (2, 3) we get

```
sol23 = NSolve[{f2,f3},{x,y},Reals]
{{x → -3.,y → -2.},{x → -2.386,y → -2.386},
 {x → 1.886,y → 1.886},{x → 1.,y → 2.}}
```

Selection of the proper solution for subsystem (2, 3)

```
G/.sol23
{16.,7.25225,0.310248,2.88709 × 10^-26}
```

So we get the last solution again,

```
sol23op = sol23//Last
{x → 1.,y → 2.}
```

Now it is clear, that the solution of the overdetermined system is (1, 2).

(c) As last technique, let us employ the Extended Newton Method,

```
≪Newton`NewtonExtended`
```

Let us employ this method to solve our overdetermined problem. Since this method is a local technique, we generate ten initial values in the region $[-2,2] \times [-3,3]$, randomly

```
iv = {RandomReal[{-2,2},10],RandomReal[{-3,3},10]}//Transpose
{{-1.22005,0.629526},{-0.870999,0.544002},{0.662122,-2.56815},
 {-1.03441,0.169312},{1.86959,1.69326},{1.52015,-2.34583},
 {0.181326,1.03311},{-1.36694,-2.74561},{-0.331234,-1.9184},
 {-1.81949,-0.354369}}
```

Then the solutions are

```
soli = Map[Last[NewtonExtended[{f1,f2,f3},{x,y},#]]&,iv]
{{1.,2.},{1.,2.},{-2.10295,-2.35517},{1.,2.},{1.,2.},
 {-1.67642,-2.67781},{1.,2.},{-2.19274,-2.34771},
 {-2.29053,-2.23825},{-2.21021,-2.2809}}
```

One can use here parallel computation. We choose the solution, which provide the smallest residual,

```
ob = Map[G/.{x → #[[1]],y → #[[2]]}&,soli]
{0.,0.,5.39282,0.,0.,11.2655,0.,5.46,5.63858,5.42019}
```

There are more candidates, but they are the same. The position of this minimum in the list is

```
First[Position[ob,Min[ob]]//Flatten]
1
```

the corresponding solution is

```
soli[[%]]
{1.,2.}
```

3.8.2 Solution of Underdetermined System

Let us solve the following underdetermined system

$$f_1 = x^2 + y^2 + z - 10,$$
$$f_2 = x^2 + x + y^2 - 5$$

via direct minimization, penalty function method, method of Lagrange multipliers and Extended Newton–Raphson.

(a) Using direct minimization means, we select the minimum norm solution from the infinite solutions. Therefore our objective will be the norm of the solution and the constrains are the equations, namely

```
sol = NMinimize[{Norm[{x,y}], f1 == 0, f2 == 0}, {x,y}]
{1.79214, {x → 1.78823, y → 0.118427}}
```

(b) Let us introduce a parameter κ as a weight of the norm of the constrains, then our objective function is

```
F[κ_] := Norm[x,y] + κ(f1^2 + f2^2)
```

Employing an initial value for the parameter $\kappa = 10$

```
sol = NMinimize[F[10], {x,y}]
{1.79088, {x → 1.78603, y → .113296}}
```

The constrains are not satisfied correctly

```
{f1, f2}/.sol[[2]]
{0.00039999, - 0.0112166}
```

Let us increase the value of $\kappa = 500$

```
sol = NMinimize[F[500], {x,y}, Method → {"RandomSearch",
"SearchPoints" → 500}]
{1.79212, {x → 1.78818, y → 0.118325}}
```

This is a better, acceptable solution,

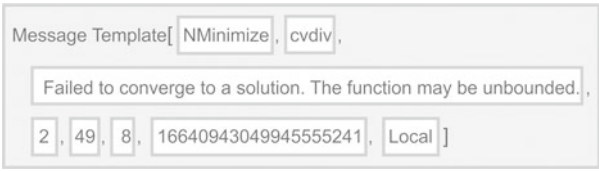
```
{f1, f2}/.sol[[2]]
{0.000415708, - 0.000620085, - 0.00118282, 0.0026602}
```

(c) Introducing new variables, we can transform the constrained problem into an unconstrained one. In addition to make the numerical solution easier, just use a little bit different objective for the norm,

```
G = x^2 + y^2 + λ1f1 + λ2f2
x^2 + y^2 + (- 3 + x^2 - xy + y^2)λ1 + (- 5 + x + x^2 + y^2)λ2
```

Now let us minimize the objective,

```
solxy = NMinimize[G, {x,y,λ1,λ2}, Method → {"RandomSearch",
"SearchPoints" → 500}]
```



$-7.529408327549746 \times 10^{622}, x \to 2.75378*10^{207}, y \to -2.75378*10^{207}, \lambda 1 \to -2.29129*10^{207}, \lambda 2 \to -1.52752*10^{207}$

The global optimization technique may fail. Let us try an algebraic way. The necessary conditions of the optimum is

```
eqs = Map[D[G, #] &, {x, y, λ1, λ2}]
{2x + (2x - y) λ1 + (1 + 2x) λ2, 2y + (-x + 2y) λ1 + 2y λ2,
 -3 + x^2 - xy + y^2, -5 + x + x^2 + y^2}
```

This algebraic system can be solved via numerical Gröbner basis,

```
sol = NSolve[eqs, {x, y, λ1, λ2}, Reals]
{{x → 0.673918, y → 1.96772, λ1 → -0.389762, λ2 → -0.676982},
 {x → 1.78823, y → 0.118427, λ1 → 0.0299103, λ2 → -0.804091}}
```

since the second solution has the smaller norm

```
Norm[{x, y}] /. sol
{2.07992, 1.79214}
```

we choose this one as a solution.

(d) The Extended Newton method can be employed as last technique. Let us try ten random initial conditions in the region [0, 2] × [0, 1]

```
iv = {RandomReal[{0, 2}, 10], RandomReal[{0, 1}, 10]} // Transpose
{{0.538531, 0.101471}, {1.04811, 0.903041}, {1.7324, 0.413993},
 {1.54789, 0.829299}, {0.565043, 0.591834}, {1.35753, 0.492477},
 {1.32906, 0.67617}, {0.0835006, 0.148338}, {1.93989, 0.0707861},
 {0.18167, 0.155675}}
```

Then the solutions are

```
soli = Map[Last[NewtonExtended[{f1, f2}, {x, y}, #]] &, iv]
{{1.78823, 0.118427}, {0.673918, 1.96772}, {1.78823, 0.118427},
 {1.78823, 0.118427}, {0.673918, 1.96772}, {1.78823, 0.118427},
 {1.78823, 0.118427}, {0.673918, 1.96772}, {1.78823, 0.118427},
 {0.673918, 1.96772}}
```

One can use here parallel computation. We choose the solution, which provide the smallest residual,

```
ob = Map[Norm[{x,y}]/.{x → #[[1]],y → #[[2]]}&,soli]
{1.79214,2.07992,1.79214,1.79214,2.07992,
 1.79214,1.79214,2.07992,1.79214,2.07992}
```

There are more candidates, but they are the same. The position of this minimum in the list is

```
First[Position[ob,Min[ob]]//Flatten]
1
```

the corresponding solution

```
soli[[%]]
{1.78823,0.118427}
```

Part II
Optimization of Systems

Chapter 4
Simulated Annealing

This is a stochastic optimization method based on a random search technique. It can find the global optimum efficiently even in a decision space of high dimension. The algorithm is based on a physical process. Let us suppose that at temperature T, the state of the molecules of a material in liquid phase can be represented by $s \in S$, where s is a stochastic variable and S is the set of all the possible states at temperature T. The *probability* of an s state can be computed from the normalized *Boltzmann* distribution,

$$p_T(s) = \frac{1}{n}\exp\left(\frac{-E(s)}{kT}\right)$$

where n is the normalization coefficient, E is the free energy belonging to the s state and k is the *Boltzmann constant*.

4.1 Metropolis Algorithm

This algorithm finds the equilibrium state s, belonging to *a given temperature*. Let us consider a state s_i and then perturb it to get another state s_{i+1}. This perturbed state will be accepted as a new state if its free energy is less than that of the original state, namely

$$\Delta E = E(s_{i+1}) - E(s_i) < 0.$$

However, even in case of $E(s_{i+1}) \geq E(s_i)$, the new state can be accepted with a small

$$\exp\left(-\frac{\Delta E}{kT}\right)$$

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_4

probability. This feature can ensure that the optimization process *will not get stuck at a local optimum*.

During the computation, the Boltzmann constant is considered as $k = 1$.

4.2 Realization of the Metropolis Algorithm

Let us consider the following example where someone thinks of a string consisting of 13 characters, let say "tobeornottobe", and the requirement is to find out this word. If we suggest a string as a possible solution, then we will receive information about how bad our trial is, namely how many wrong characters are in our suggestion.

Remark Without any strategy, we would need a maximum of 2.48115×10^{18} trials to hit the target. Considering the English alphabet, there are 26 characters (small letters). The probability that one character is in the right place is

```
1/26//N
0.0384615
```

Then the probability that all of the 13 characters are in the right place is

```
(1/26)^13//N
4.03038*10^-19
```

So the maximum number of the necessary trials is

```
1/%
2.48115*10^18
```

4.2.1 Representation of a State

The trial string can be represented by the ASCII code of the characters

```
s1=Table[RandomInteger[{97,122}],{13}]
{108,111,115,108,109,98,97,116,117,107,114,102,111}
```

in text form

```
FromCharacterCode[s1]
loslmbatukrfo
```

4.2.2 The Free Energy of a State

The free energy of a state can be represented by the number of the wrong characters in the trial string, namely the difference between the target string ("tobeornottobe") and the trial string

```
keyPhrase = ToCharacterCode ["tobeornottobe"]
{116,111,98,101,111,114,110,111,116,116,111,98,101}
```

The difference between the two strings can be associated with the free energy of the actual trial string

```
E[s_] := Count [keyPhrase - s, Except [0]]
E[s1]
12
```

4.2.3 Perturbation of a State

In order to perturb a state s, we should consider the string of s, character by character, and change them one by one with a probability μ. To do that, we define a function which produces true or false with a μ probability,

```
flip [x_] := If [RandomReal[] ≤ x, True, False].
```

Then we apply it to carry out the perturbation of s with μ probability. In case we get true for a certain character, then we change it with another character (ASCII code) randomly from the English alphabet.

```
Perturb [μ_, s_] := Map [If [flip [μ], RandomInteger [{97,122}], #]&, s];
```

Let us employ it for s_1 with probability $\mu = 0.3$

```
s2 = Perturb[0.3, s1]
{108,111,115,108,109,98,98,110,117,107,111,119,117}
```

In text form

```
FromCharacterCode [s2]
loslmbbnukowu
```

The free energy of this state (string) is

```
E[s2]
11
```

4.2.4 Accepting a New State

If the free energy of the perturbed state is less than the original one, then we accept it as a new state or reject it with a probability

$$1 - \exp\left(-\frac{\Delta E}{T}\right).$$

This means that the new state can be accepted with a probability

$$\exp\left(-\frac{\Delta E}{T}\right)$$

even with higher energy level.

```
Accept[ΔE_,T_]:=
 If[ΔE<0∨RandomReal[]<Exp[ΔE/T],True,False].
```

For example let $T = 0.5$

```
Accept[E[s2]-E[s1],0.5]
True
```

Remark If the energy level does not change then the new state will be accepted.

4.2.5 Implementation of the Algorithm

Collecting these functions, the function of the Metropolis algorithm is the following, see the flow chart Fig. 4.1.

```
Metropolis[s0_,μ_,T_,n_]:=Module[{i,s1,s2,S},
  i=0;
  S={};
  s1=s0;
 While[i≤n,
  S=AppendTo[S,s1];
  s2=Perturb[μ,s1];
  If[Accept[E[s2]-E[s1],T],s1=s2];
  i=i+1;];
  S];
```

The flow chart demonstrates the Metropolis algorithm.

The output is the series of strings of the states computed during the iteration steps.

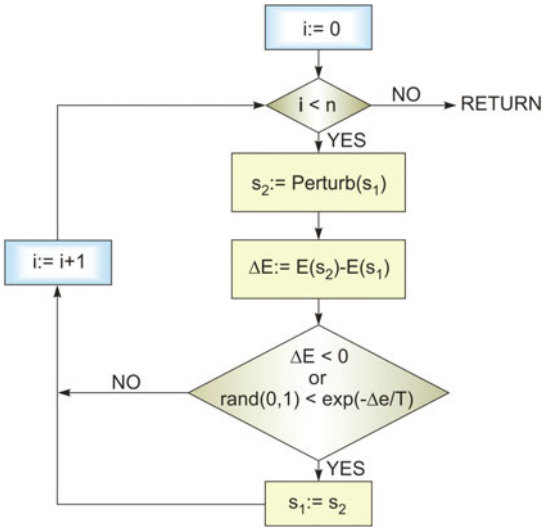


Fig. 4.1 The Metropolis algorithm

Let us test our function with temperature $T = 1$ and iteration number $n = 100$. The probability of the perturbation is $\mu = 0.3$.

First, we randomly generate an initial state,

```
s0 = Table [RandomInteger [{97,122}],{13}];
```

Then, employing the algorithm,

```
M = Map [E[#]&,Metropolis [s0,0.3,1.,100]];
```

The energy level of the series of the subsequent states can be seen in Fig. 4.2.

We can see a strong fluctuation of the energy level at this relatively high temperature. Let us repeat this numerical simulation at a lower temperature $T = 0.1$.

```
M = Map [E[#]&,Metropolis [s0,0.3,0.1,100]];
```

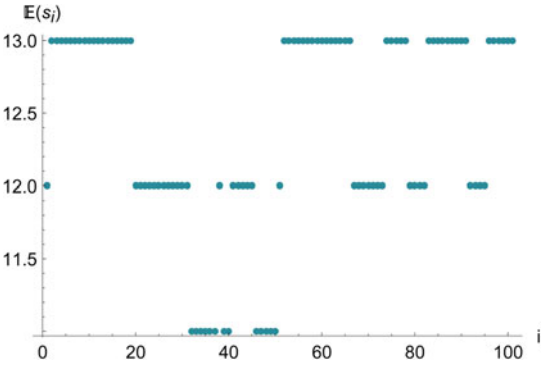


Fig. 4.2 The energy levels belonging to the different states computed in the iteration steps at $T = 1$

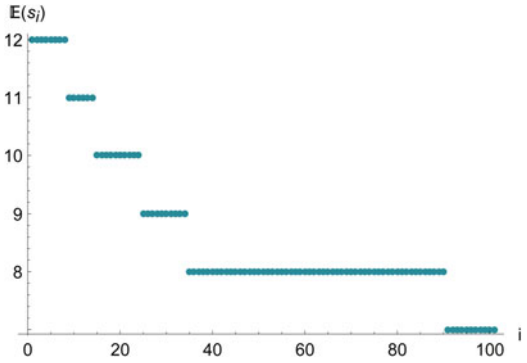


Fig. 4.3 The energy levels belonging to the different states computed in the iteration steps at $T = 0.1$

Figure 4.3 shows, that the energy level reaches a stable equilibrium at this low temperature.

4.3 Algorithm of the Simulated Annealing

In order to get the minimum of the energy level, the Metropolis algorithm will be executed at different decreasing temperature levels, step by step. The simplest "cooling" strategy can be,

$$T_{i+1} = \alpha T_i$$

where $0 < \alpha < 1$. The region where the best convergence can be achieved is $\alpha \in [0.8, 0.99]$. Figure 4.4 shows the flow chart of the algorithm.

where T_{min}, the lowest temperature is input as a simulation parameter.

4.4 Implementation of the Algorithm

Now, the function of the simulated annealing method can be easily defined as,

```
SimulatedAnnealing[s0_,T0_,Tf_,μ_,α_,n_]:=Module[{T,s,S,TS},
 T=T0;
 s=s0;
 S={s0};
 TS={T0};
 While[T>Tf,
  s=Last[Metropolis[s,μ,T,n]];
  S=AppendTo[S,s];
  T=αT;(*annealing schedule*)
  TS=AppendTo[TS,T];];
 {S,TS}]
```

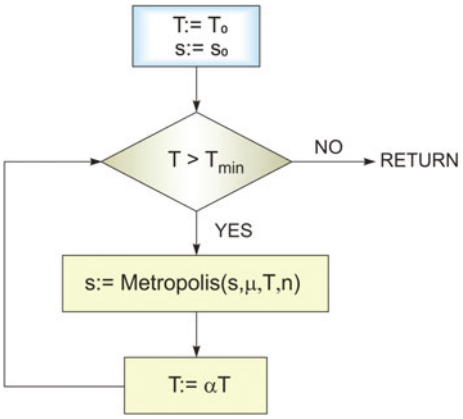


Fig. 4.4 The algorithm of the simulated annealing

The output is the actual cooling temperatures and the corresponding steady states of the energy levels. Let us carry out a simulation with the initial state

```
s0 = Table [RandomInteger [{97,122}],{13}];1
```

The initial temperature T_0 is,

$$T_0 = 0.5.$$

The termination temperature is,

$$T_{min} = 0.005.$$

The probability of the perturbation is,

$$\mu = 0.3.$$

The cooling factor,

$$\alpha = 0.92.$$

Let us carry out 200 iteration steps in the Metropolis algorithm at every temperature levels,

```
MS = SimulatedAnnealing [s0,0.5,0.005,0.3,0.92,200];
```

The cooling temperatures (Fig. 4.5).

The steady state energy levels at these temperatures are shown in Fig. 4.6.

Figure 4.6 shows that at the end of the iteration, the energy level is zero. This means that all of the characters are in the proper place in the string representing the terminal state,

```
FromCharacterCode [Last [MS [[1]]]]
tobeornottobe.
```

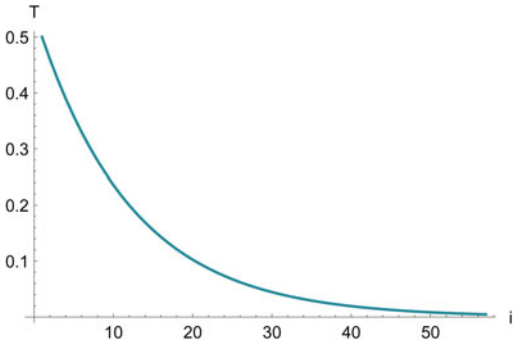


Fig. 4.5 Actual temperatures during the cooling process

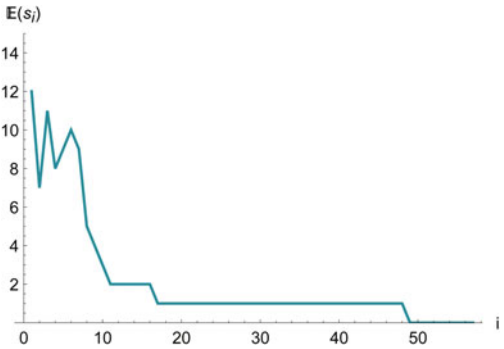


Fig. 4.6 The equilibrium energy levels

4.5 Application to Computing Minimum of a Real Function

Now, let us employ the method for finding global minimum of a real function using a simple example. Let us find the global minimum of a function $f(x)$ in the region $x \in [0, 50]$ where (Fig. 4.7)

$$f(x) = |2(x-24) + (x-24)\ \sin(x-24)|$$

```
f[x_]:=Abs[2(x-24) + (x-24) Sin[(x-24)]];
```

In this case one point of the region is considered as an ith state, x_i and the corresponding free energy is $f_i = f(x_i)$. The x_i can be represented in binary form as a string. Using more digits (characters) in the string, we have more precise representation of a real value (higher resolution). Let us consider 15 bits then the biggest number which can be represented is $s_{max} = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$, or

```
maxi = 2^15 - 1
32767.
```

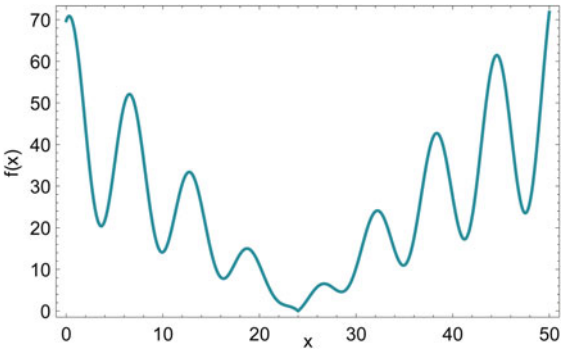


Fig. 4.7 A multimodal real function with a global minimum

The smallest number s_{min} = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

```
mini = 2^0 - 1
0
```

This means that the interval [0, 32767] should be projected into the interval [0, 50]. Let us consider the string s_i. The corresponding real value d_i can then be computed with the following function

```
Horner[u_,base_:2]:=Fold[base#1 + #2&,0,u]
```

For example, the binary form is

```
Subscript[s,i]={1,0,1,0,0,0,0,0,0,0,0,0,0,1,0};
```

and its decimal representation is,

```
di = Horner[si]
20482
```

Indeed

```
2^14 + 2^12 + 2^1
20482
```

and the scaled decimal value x_i is

$$x_i = \frac{50.}{32767} d_i$$

```
31.254
```

The energy level of s_i can be computed as

```
E[s_] := f[50./32767 Horner[s]]
```

In our case

```
E[s_i]
20.4951
```

It goes without saying, that the function of the perturbation should be changed, since now there are only two characters 0 or 1,

```
Perturb[μ_, s_] := Map[If[flip[μ], RandomInteger[{0,1}], #]&, s];
```

Then we are looking for the string of 15 characters, which provides the minimum of the $E(s)$, which is the same as the minimum of $f(x)$.

Let the initial state be

```
s0 = Table[RandomInteger[{0,1}], {15}]
{1,0,0,0,0,1,0,0,0,0,1,0,0,0,1}
x0 = 50./32767 Horner[s0]
25.808
```

Then using the parameters $T_{max} = 0.5$, $T_{min} = 0.005$, $\mu = 0.3$, $\alpha = 0.9$ and $n = 200$,

```
MS = SimulatedAnnealing[s0, 0.5, 0.005, 0.3, 0.9, 200];
```

The values of the free energy levels during the iteration process are (Fig. 4.8)

```
Map[E[#]&, MS[[1]]];
```

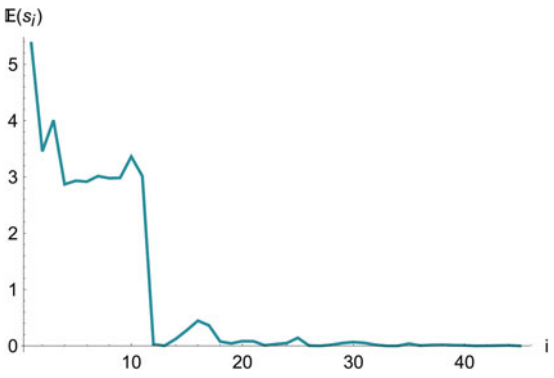


Fig. 4.8 The free energy levels as function of the number of the iterations

The convergence is quite fast, see Figs. 4.9 and 4.10.

The location of the global minimum is,

```
50.
-----  Horner[Last[MS[[1]]]]
32767
23.9998
```

The value of the global minimum

```
E[Last[MS[[1]]]]
0.000488237
```

The solution of the built-in function is

```
NMinimize[{f[x], 0 ≤ x ≤ 50}, x]
 {1.27898 × 10^-13, {x → 24.}}
```

Using more digits, we can get better and more precise solutions.

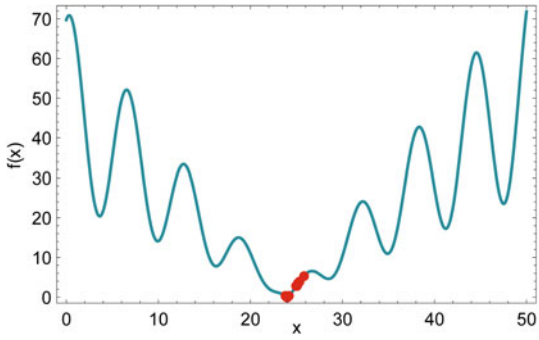


Fig. 4.9 Convergence of the iteration

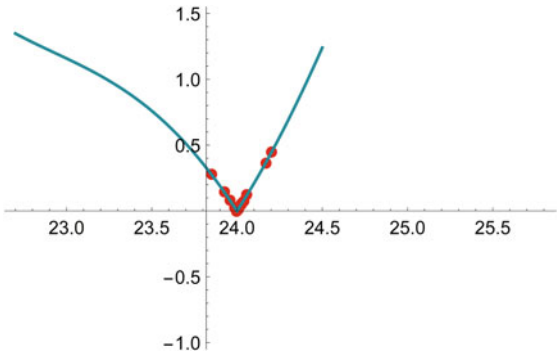


Fig. 4.10 Convergence in the neighborhood of the global minimum

4.6 Generalization of the Algorithm

The simulated annealing method is built into most computing systems like *Mathematica*, *Maple* and *Matlab*.

Now, let us employ *Mathematica* to solve the following optimization problem. The function to be maximized is

$$f(x,y) = g(x+11, y+9) + g(x-11, y-3) + g(x+6, y-9),$$

where

$$g(x,y) = \frac{50 \sin \sqrt{x^2+y^2}}{\sqrt{x^2+y^2}} - \sqrt{x^2+y^2}.$$

We are looking for the *global maximum* in the region $(x,y) \in [-20,20]^2$ employing simulated annealing method,

```
sol = Reap [NMaximize [{f[x,y], -20 ≤ x ≤ 20, -20 ≤ y ≤ 20}, {x,y},
 EvaluationMonitor: → Sow [{x,y}],
 Method → {"SimulatedAnnealing", "PerturbationScale" → 5}]];
```

The maximum,

```
sol [[1]]
{10.8432, {x → -6.01717, y → 9.06022}}
```

The number of the iteration is (Fig. 4.11)

```
ni = Length [hist]
317
```

Let us display the states belonging to the actual iteration steps, see Fig. 4.12.
Let us display every 5th point of the function values computed during the iteration, see Fig. 4.13.

Remarks Keep in mind that $\max(f(x)) = \min(-f(x))$.

4.7 Applications

4.7.1 A Packing Problem

Given n disks with different radii r_i. Let us arrange them in the smallest square area without overlapping each other! Let us consider 10 disks,

```
n = 10;
```

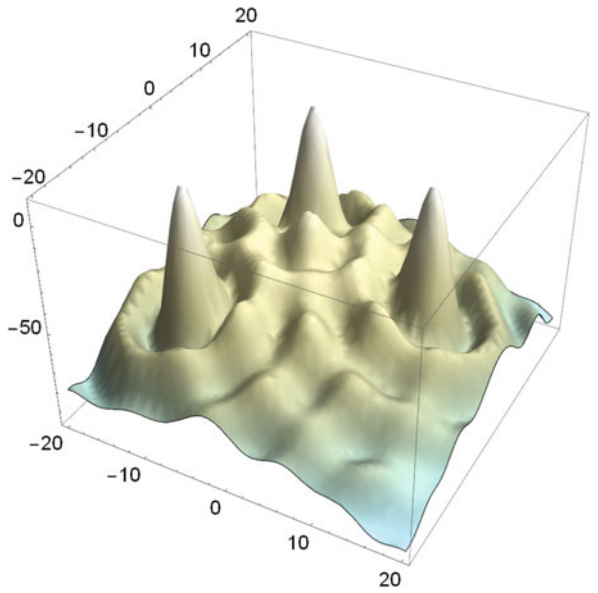


Fig. 4.11 A function of two variables with many local maximums

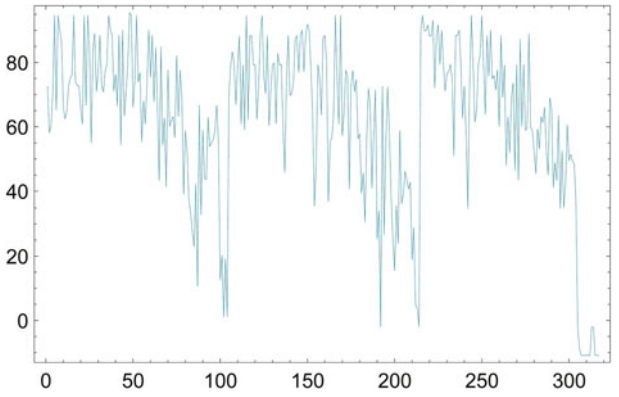


Fig. 4.12 The states vs. number of iterations

Their radii are chosen randomly between a lower and an upper limit

```
maxR = 0.2;
minR = 0.8;
Table[r_i = Random[Real,{minR,maxR}],{i,1,n}]
{0.466063,0.396305,0.706345,0.471148,
 0.5559,0.642377,0.25306,0.512608,0.62259,0.792981}
```

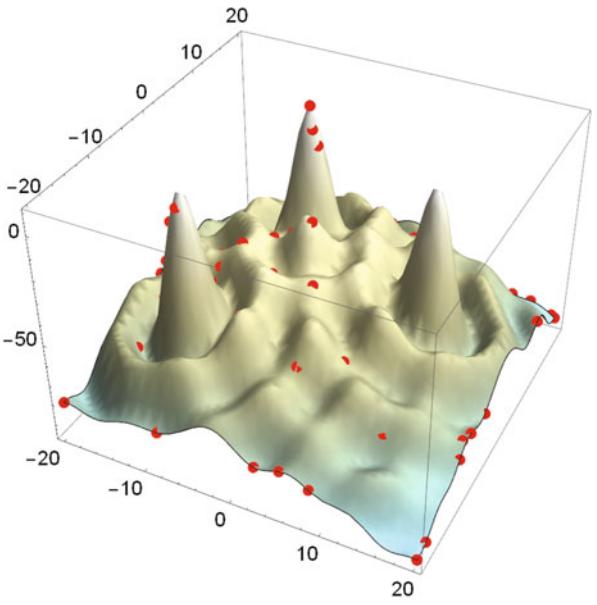


Fig. 4.13 The function values computed during the iteration process

We assume that the area of the square is $2w \times 2w$, and w has an upper limit $w \leq B$. Now let $B = 10$

```
B = 10;
```

The first group of the constrains hinders the overlapping of the disks,

```
constraints01 =
 Table[(xi - xj)^2 + (yi - yj)^2 ≥ ((ri + rj)^2), i, 1, n, j, i + 1, n] // Flatten;
```

The second group of the constrains ensures that $2w \times 2w \leq 2B \times 2B$

```
constraints02 = Table[
 {ri + xi - w ≤ 0, - ri + xi + w ≥ 0, ri + yi - w ≤ 0, - ri + yi + w ≥ 0,
 - B ≤ xi ≤ B, - B ≤ yi ≤ B}, {i, 1, n}]}, {i, 1, n}] // Flatten;
```

Combining these constrains with the upper limit of the positive radius,

```
constraints = Join[constraints01, constraints02, {0 < = w < = B}];
```

The unknown variables we are looking for are the center coordinates of the disks $\{x_i, y_i\}$, which minimize w,

```
vars = Join[{w}, Table[{xi, yi}, {i, 1, n}]] // Flatten;
```

Employing simulated annealing method,

```
soln = NMinimize [Join[{w}, constraints], vars,
  Method → {"SimulatedAnnealing", "PerturbationScale" → 6,
  "SearchPoints" → 100}] // Timing // Quiet
{37.0658, {1.84394, {w → 1.84394, x1 → 1.37788, y1 → - 0.392879,
  x2 → 0.513346, y2 → - 1.44764, x3 → 1.13759, y3 → 0.75464,
  x4 → 1.3728, y4 → - 1.33008, x5 → - 0.0064139, y5 → 1.28804,
  x6 → - 1.20157, y6 → 1.20157, x7 → - 0.135096, y7 → - 1.58773,
  x8 → - 0.644569, y8 → 0.189765, x9 → 0.290907, y9 → - 0.45332,
  x10 → - 1.05096, y10 → - 1.05096 }}}
```

Figure 4.14 shows the positions of the different disks.

4.7.2 The Traveling Salesman Problem

The problem is to visit each town once and minimize the length of the tour. We have to find an ordering of the visits that minimizes the total distance and visits all the v_i's once, where v_i's are the locations of the towns.

Let us generate 20 town-coordinates in 2D,

```
SeedRandom [2]; points = RandomReal [10, {20, 2}];
```

The coordinates of the town,

```
points
{{7.2224, 1.09449}, {4.70703, 5.35582}, {5.83178, 2.93942},
 {1.65154, 6.01258}, {7.54218, 7.71123}, {7.78574, 0.236104},
 {9.22757, 9.92454}, {3.50409, 0.450047}, {5.01359, 6.33756},
 {6.42208, 3.89875}, {6.64971, 8.43882}, {5.6904, 3.98212},
 {2.38652, 6.73513}, {4.19507, 5.87398}, {0.0833523, 9.42441},
 {7.71263, 1.47503}, {9.64774, 8.98747}, {3.32963, 2.04548},
 {8.39035, 2.50388}, {2.38638, 6.16616}}
```

Then we employ simulated annealing method

```
{length, tour} =
 FindShortestTour [points, Method → " SimulatedAnnealing"];
```

The total length of the distance of the travel is,

```
length
41.7378
```

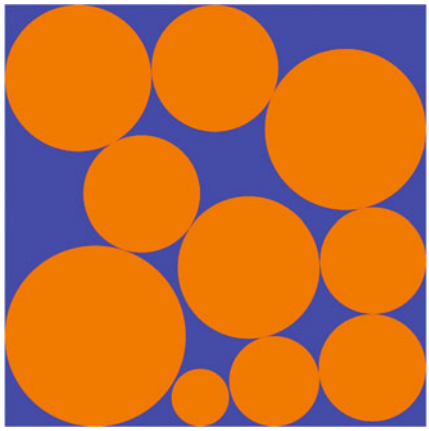


Fig. 4.14 Optimal placement of disks in a minimal square area

The order of the optimal visit is

```
tour
{9,5,17,7,11,15,13,20,4,18,8,6,1,16,19,3,10,12,2,14,9}
```

Figure 4.15 displays the route.
Let us solve the problem in space (3D). We are visiting planets in space,

```
SeedRandom[2];
With[{p=RandomReal[20,{9,3}]},
 Graphics3D[
  {Line[
  p[[Last[FindShortestTour[p,
   Method→"SimulatedAnnealing"]]]]],Sphere/@p},
  ImageSize→300]]//Quiet
```

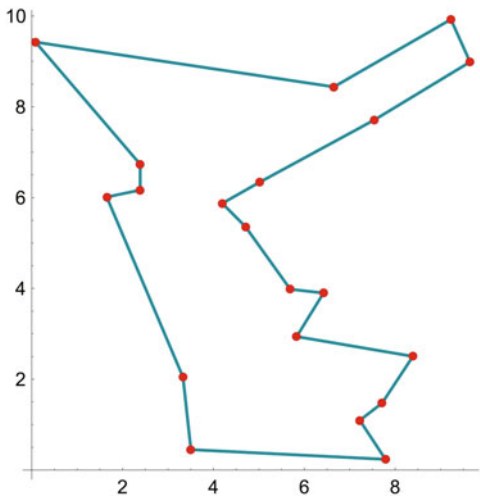


Fig. 4.15 Optimal route of the tour

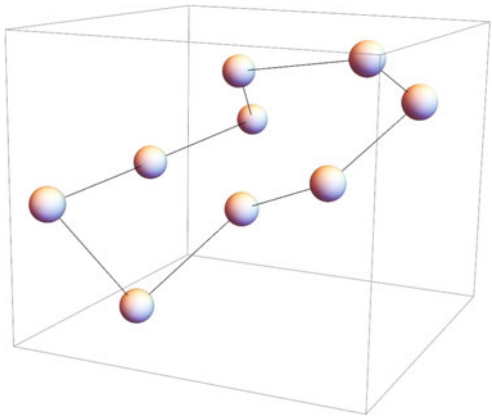


Fig. 4.16 Optimal route of the flight in space

Now, let us visit the capitals of countries of the world (Fig. 4.16).

Function to give a location on the globe (sphere),

```
SC[{lat_,lon_}]:=
 r{Cos[lon°]Cos[lat°],Sin[lon°]Cos[lat°],Sin[lat°]};
```

Locations of the towns

```
r=6378.7;
places=CountryData["Countries"];
centers=Map[CountryData[#,"CenterCoordinates"]&,places];
distfun[{lat1_,lon1_},{lat2_,lon2_}]:=
  VectorAngle[SC[{lat1,lon1}],SC[{lat2,lon2}]]r;
```

The countries to be visited are

```
places
```

{Afghanistan, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antigua and Barbuda, Argentina, Armenia, Aruba, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados, Belarus, Belgium, Belize, Benin, Bermuda, Bhutan, Bolivia, Bosnia and Herzegovina, Botswana, Brazil, British Virgin Islands, Brunei, Bulgaria, Burkina Faso, Burundi, Cambodia, Cameroon, Canada, Cape Verde, Cayman Islands, Central African Republic, Chad, Chile, China, Christmas Island, Cocos Keeling Islands, Colombia, Comoros, Cook Islands, Costa Rica, Croatia, Cuba, Curacao, Cyprus, Czech Republic, Democratic Republic of the Congo, Denmark, Djibouti, Dominica, Dominican Republic, East Timor, Ecuador, Egypt, El Salvador, Equatorial Guinea, Eritrea, Estonia, Ethiopia, Falkland Islands, Faroe Islands, Fiji, Finland, France, French Guiana, French Polynesia, Gabon, Gambia, Gaza Strip, Georgia, Germany, Ghana, Gibraltar,

Greece, Greenland, Grenada, Guadeloupe, Guam, Guatemala, Guernsey, Guinea, Guinea-Bissau, Guyana, Haiti, Honduras, Hong Kong, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland, Isle of Man, Israel, Italy, Ivory Coast, Jamaica, Japan, Jersey, Jordan, Kazakhstan, Kenya, Kiribati, Kosovo, Kuwait, Kyrgyzstan, Laos, Latvia, Lebanon, Lesotho, Liberia, Libya, Liechtenstein, Lithuania, Luxembourg, Macau, Macedonia, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta, Marshall Islands, Martinique, Mauritania, Mauritius, Mayotte, Mexico, Micronesia, Moldova, Monaco, Mongolia, Montenegro, Montserrat, Morocco, Mozambique, Myanmar, Namibia, Nauru, Nepal, Netherlands, New Caledonia, New Zealand, Nicaragua, Niger, Nigeria, Niue, Norfolk Island, Northern Mariana Islands, North Korea, Norway, Oman, Pakistan, Palau, Panama, Papua New Guinea, Paraguay, Peru, Philippines, Pitcairn Islands, Poland, Portugal, Puerto Rico, Qatar, Republic of the Congo, Réunion, Romania, Russia, Rwanda, Saint Helena, Ascension and Tristan da Cunha, Saint Kitts and Nevis, Saint Lucia, Saint Pierre and Miquelon, Saint Vincent and the Grenadines, Samoa, San Marino, São Tomé and Príncipe, Saudi Arabia, Senegal, Serbia, Seychelles, Sierra Leone, Singapore, Sint Maarten, Slovakia, Slovenia, Solomon Islands, Somalia, South Africa, South Korea, South Sudan, Spain, Sri Lanka, Sudan, Suriname, Svalbard, Swaziland, Sweden, Switzerland, Syria, Taiwan, Tajikistan, Tanzania, Thailand, Togo, Tokelau, Tonga, Trinidad and Tobago, Tunisia, Turkey, Turkmenistan, Turks and Caicos Islands, Tuvalu, Uganda, Ukraine, United Arab Emirates, United Kingdom, United States, United States Virgin Islands, Uruguay, Uzbekistan, Vanuatu, Vatican City, Venezuela, Vietnam, Wallis and Futuna Islands, West Bank, Western Sahara, Yemen, Zambia, Zimbabwe}
The number of the countries to be considered is

```
Length[%]
240
```

The traveling salesman problem is a famous example of an NP-complete problem. There is no known algorithm that is guaranteed to solve every-city problem in polynomial time. Brute force is completely impractical. The total number of possible tours when there are cities is. So, for instance, with 30 cities there are 4420880996869850977271808000000 possible tours. Suffice it to say that calculating the length of each and then identifying the shortest among them is well beyond the capacity of today's computers. Optimal solution is possible in practice up to 20 via dynamic programming. Although it might rise up to 50 if integer linear programming is utilized.

Other approximate methods are available for big problems, but they provide, only suboptimal solutions. Let us carry out the computation some different suboptimal methods.

Computing the route with Simulated Annealing method provide a quite good result, but takes time,

```
AbsoluteTiming[{dist,route}=FindShortestTour[centers,
 DistanceFunction→distfun,Method→"SimulatedAnnealing"];
 //Quiet]
{181.848,Null}
```

The total distance,

```
dist
191706
```

The 2-*opt* algorithm is much faster but provide somewhat longer distance

```
AbsoluteTiming[{dist,route}=FindShortestTour[centers,
 DistanceFunction→distfun,Method→"TwoOpt"];
 //Quiet]
{5.14926,Null}
```

The total distance,

```
dist
198191
```

The *greedy* algorithm is faster than the 2-*opt* but its solution is quite bad

```
AbsoluteTiming[{dist,route}=FindShortestTour[centers,
 DistanceFunction→distfun,Method→"Greedy"];
 //Quiet]
{1.0654,Null}
```

The total distance,

```
dist
235646
```

The integer linear programming can not be employed for so big problem, it takes enormous time, after one hour running we could not get any result.

```
AbsoluteTiming[{dist,route}=FindShortestTour[centers,
 DistanceFunction→distfun,Method→"IntegerLinearProgramming"];
 //Quiet]
$Aborted
```

The fastest computation and the shortest tour is provided by the *Mathematica* built-in algorithm

```
AbsoluteTiming[{dist,route}=FindShortestTour[centers,
 DistanceFunction→distfun,Method→"Automatic",
 PerformanceGoal→"Quality"];//Quiet]
{0.906354,Null}
```

The total distance,

```
dist
181071.
```

Figure 4.17 shows the visualization of the globe with the optimal route,

```
surfaceCenters=Map[SC,centers[[route]]];
GreatCircleArc[{lat1_,lon1_},{lat2_,lon2_}]:=
 Module[{u=SC[{lat1,lon1}],v=SC[{lat2,lon2}],a},
 a=VectorAngle[u,v];Table[Evaluate[RotationTransform[θ,
 {u,v}][u]],{θ,0,a,a/Ceiling[10a]}]]
tourLine=
 Apply[GreatCircleArc,Partition[centers[[route]],2,1],{1}];
Graphics3D[{Sphere[{0,0,0},0.99r],
 Map[Line[Map[SC,CountryData[#,"SchematicCoordinates"],
 {-2}]]&,places],{Red,Thick,Line[tourLine]},
 {Yellow,PointSize[Medium],
 Map[Tooltip[Point[SC[CountryData[#,"CenterCoordinates"]]],#]&,
 places]}},Boxed→False,SphericalRegion→True]
```

4.8 Exercise

Let us solve the following nonlinear parameter estimation problem. Our model is

$$f(x) = \alpha \sin(\beta x)$$

```
f[x_]:=2.5 Sin[1.5 x]
```

The parameters to be estimated are (α, β). We have noisy measurements $\{x_i, f_i\} + \epsilon$, where ϵ is a random noise in the region $x \in [-2, 8]$ with uniform distribution, see Fig. 4.18.

Fig. 4.17 Optimal route of the travel on the Earth surface

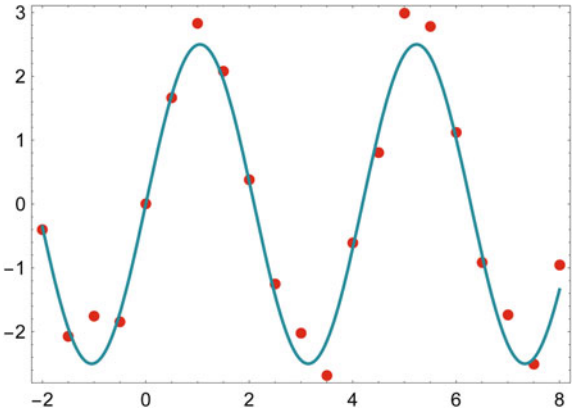


Fig. 4.18 Noisy measurements (points) for the parameter estimation problem. The continuous line represents the function with parameters $\alpha = 2.5$ and $\beta = 1.5$

```
SeedRandom[12];
XF = Table[{-2 + i0.5, Random[Real, {0.7, 1.3}]f[-2 + i0.5]},
 {i, 0, 20}];
p1 = ListPlot[XF, PlotStyle → {RGBColor[1, 0, 0], PointSize[0.02]},
 Frame → True, Axes → False, AspectRatio → 0.7,
 ImageSize → {300, 250}];
Show[{p1, Plot[f[x], {x, -2, 8}]}]
```

Our task is to compute the parameters α and β on the bases of the *noisy data*. To do that we employ the least squares method, namely we are going to minimize the following objective

$$G(\alpha, \beta) = \sum_{i=1}^{m} (f_i - \alpha \sin(\beta x_i))^2,$$

where m is the number of the measured data pairs.

```
m = Length [XF]
21
```

Then our objective function is

$$\texttt{G}(\{\alpha\_, \beta\_\}) := \sum_{\texttt{i}=1}^{\texttt{m}} (\texttt{XF}[[\texttt{i},2]] - \alpha \texttt{Sin}(\beta \texttt{XF}[[\texttt{i},1]]))^2$$

The contour plot of this function in Fig. 4.19 indicates three local minimums,

Computing the local minimums via a local technique (Newton's method), one gets a minimum which is "close" to the initial guess. These local minimums are, see Fig. 4.20. They are local minimums indeed, since

```
min1 = {α, β}/.FindMininmum[G[{α, β}], {α, 0.75}{β, 0.5}][[2]]
{0.755528, 0.537572}
```

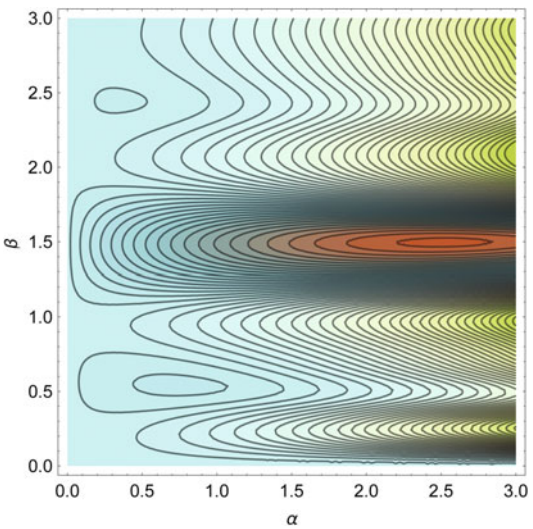


Fig. 4.19 The contour plot of the objective function in the region of [0, 3] × [0, 3]

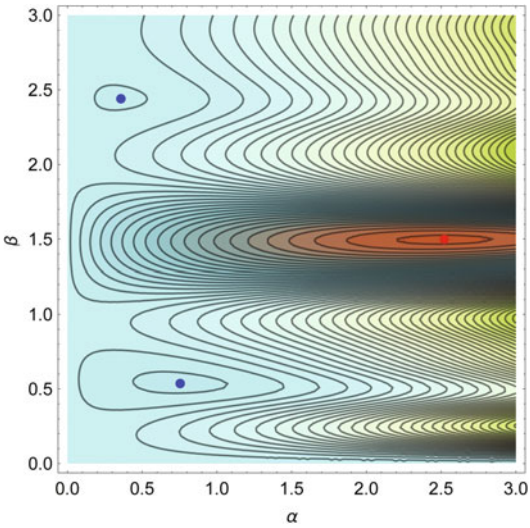


Fig. 4.20 The local minimums of the problem

```
G[min1]
63.6745

min2 = {α, β}/.FindMinimum[G[{α, β}], {{α, 0.5}, {β, 2.5}}][[2]]
{0.357603, 2.44236}

G[min2]
67.6705

min3 = {α, β}/.FindMinimum[G[{α, β}], {{α, 2.5}, {β, 1.5}}][[2]]
{2.52079, 1.49648}

G[min3]
2.33031
```

Let us employ global minimization via simulated annealing,

```
sol = NMinimize[G[{α, β}], {α, β}, Method → "SimulatedAnnealing"]
{2.33031, {α → 2.52079, β → 1.49648}}
```

which is the global minimum.

The approximation of the function $f(x)$ is $g(x)$, and Fig. 4.21 shows the approximation of the original function (not the measurement points!). It is

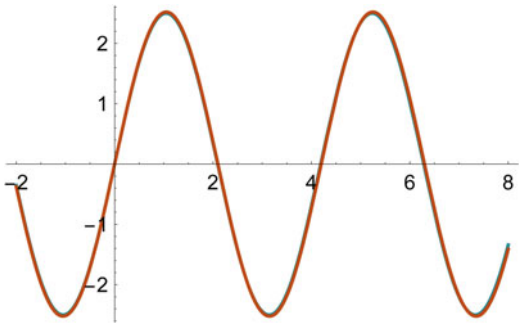


Fig. 4.21 The original $f(x)$ and the approximating function $g(x)$

important to keep in mind that the original function should be approximated on the basis of the noisy measurements and not noisy points.

```
g[x_]:=αSin[βx]/.sol[[2]]
```

Chapter 5
Genetic Algorithms

5.1 The Genetic Evolution Concept

This stochastic method is based on Darwin's principle of natural selection. It means the fittest individual organisms of a population will survive. An individual can be represented as a string of chromosomes (characters). The string will be the binary form of a value of a real x variable (an individual), and $f(x)$ will be the measure of the fitness of x. Therefore finding the fittest individual means maximizing the $f(x)$ function, see Freeman (1994).

5.2 Mutation of the Best Individual

First, a population of individuals will be generated with random chromosome strings. This will be the first generation. Then the fittest individual will be selected form this population. The chromosomes of this fittest individual will be mutated (changed) randomly in order to generate a new population as the second generation. This strategy leads to the fittest generation in general and probably its fittest individual will have higher fitness value than that of the best individual of the previous generation.

Let us consider again the "tobeornottobe" problem. The ASCII codes of the English alphabet will represent the chromosomes. First, we generate the population of the first generation with 200 individuals.

```
P1 = Table[RandomInteger[{97,122}],{i,1,200},{j,1,13}];
Short[P1,5]
{{119,113,117,102,100,99,110,100,110,106,117,106,115},
 {106,114,105,99,113,97,108,98,98,111,108,106,122}, ≪197≫,
 {106,114,114,109,109,112,106,121,113,113,118,97,100}}
```

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_5

We are looking for the string

```
tobe = "tobeornottobe";
```

Let us convert the text into a string of ASCII codes

```
keyPhrase = ToCharacterCode[tobe]
{116,111,98,101,111,114,110,111,116,116,111,98,101}
```

The string of the fittest individual will have the most characters (ASCII code) in the proper place

```
diff = Map[(keyPhrase - #)&,P1];
matches = Map[Count[#,0]&,diff];
fitness = Max[matches];
index = Position[matches,fitness];
```

in our case,

```
pbest = P1[[First[Flatten[index]]]]
{109,111,112,118,115,109,97,122,102,97,106,98,98}
```

In character form

```
FromCharacterCode[pbest]
mopvsmazfajbb
```

Its fitness is

```
fitness
2
```

This means that two characters are in the proper place.

Now we are going to create the second generation. Here is a function which gives us the value True with probability μ,

```
flip[μ_] := If[RandomReal[] ≤ μ,True,False]
```

This function will mutate a string changing its characters one by one with a probability *pmutate*,

```
Mutate[pmute_,letter_]:=
 If[flip[pmute],RandomInteger[{97,122}],letter];
```

Let us consider the best individual of the first generation,

```
pbest
{109,111,112,118,115,109,97,122,102,97,106,98,98}
```

and carry out the mutation of all its characters with probability 0.3,

```
Map[Mutate[0.3,#]&,pbest]
{115,111,112,121,98,120,97,97,102,97,106,121,98}
```

or in text form,

```
FromCharacterCode[%]
sopybxaafajyb
```

In order to get the population of the second generation, one may have to carry out this mutation 200 times,

```
P2 = Table[Map[Mutate[0.3,#]&,pbest],{200}];
```

Figure 5.1 shows the flow chart of the algorithm
The *Mathematica* function for this algorithm is the following,

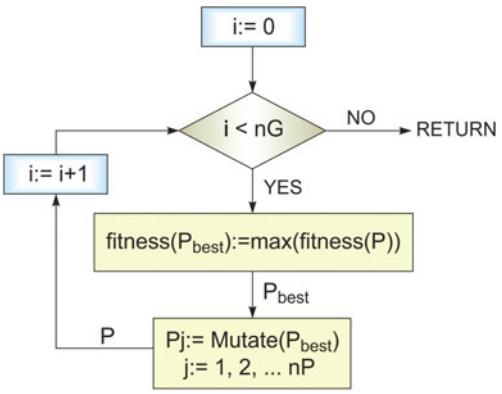


Fig. 5.1 The flow chart of the algorithm of the *Mutation of the Best* principle

```
Mutation[pmutate_,keyPhrase_,pop_,numGens_]:=
 Module[{i,P,pbest,diff,matches,index,fitness,n,S},
   n=Length[pop];
   P=pop;
   S={};
 For[i=1,i≤numGens,i++,
  diff=Map[(keyPhrase-#)&,P];
  matches=Map[Count[#,0]&,diff];
  fitness=Max[matches];
  index=Position[matches,fitness];
  pbest=P[[First[Flatten[index]]]];
  AppendTo[S,fitness];
  FromCharacterCode[pbest];
  If[fitness==13,Break[]];
  P=Table[Map[Mutate[pmutate,#]&,pbest],
   {n}];
 ];
 S];
```

The input of the `Mutation`:

pmutate—the probability of the chromosome change (mutation of the string of an individual)
keyPhrase—the string we are looking for, we need it to define the fitness function
pop—the population of the first generation
numGens—the number of the iterations (the maximum number of the generations)

In this example, if the fitness value of the best element of a generation is 13 then the iteration will stop.

The output: the string of the best individuals of the generations, (*S*).

Let us carry out a simulation,

```
AbsoluteTiming[G1=Mutation[0.1,keyPhrase,P1,200];]
{0.189405,Null}
G1
{2,3,4,5,6,7,8,9,10,10,11,12,12,12,13}
```

The result can be computed in finite steps!

5.3 Solving a Puzzle

Problem

Let us consider a puzzle in order to illustrate how one can formulate a problem to be solved by the Mutation of the Best method.

Given a 5 × 5 grid, see Fig. 5.2, and 9 positive integer numbers {1, 2, 3,…,9}. The 9 numbers should be arranged in the grid in such way that the sum shown in Fig. 5.2 should simultaneously add up to 25. The arrangement is as follows:

(a) the sum of the numbers in horizontal direction, $(x_1 + x_2 + x_9 + x_5 + x_6)$,
(b) the sum of the numbers in vertical direction, $(x_4 + x_3 + x_9 + x_8 + x_7)$,
(c) the sum of the numbers staying at the ends of the cross, $(x_1 + x_4 + x_6 + x_7)$

Solution:

One individual can be randomly generated without repetition of the numbers

```
s = RandomSample[{1,2,3,4,5,6,7,8,9},9]
{3,8,5,7,2,9,6,1,4}
```

The function measuring the fitness is

$$1/(1+|(x_1+x_2+x_5+x_6+x_9)-25|+|(x_3+x_4+x_7+x_8+x_9)-25|+ |(x_1+x_4+x_6+x_7)-25|)$$

```
fit[x_]:=
 1/(1 + Abs[x[[1]] + x[[2]] + x[[9]] + x[[5]] + x[[6]] - 25] +
 Abs[x[[4]] + x[[3]] + x[[9]] + x[[8]] + x[[7]] - 25] +
 Abs[x[[4]] + x[[1]] + x[[7]] + x[[6]] - 25])
fit[s]//N
0.25
```

The optimal value is 1 but in other cases, it is less than 1. A function which generates a population of n individuals is

·	·	x_4	·	·
·	·	x_3	·	·
x_1	x_2	x_9	x_5	x_6
·	·	x_8	·	·
·	·	x_7	·	·

Fig. 5.2 The grid of the puzzle

```
generation[n_]:=Table[RandomSample
 [{1,2,3,4,5,6,7,8,9},9],{n}]
```

A generation in case of $n = 10$

```
u=generation[10]
{{9,8,5,6,3,2,7,4,1},{6,5,9,7,1,2,8,4,3},
 {4,3,6,1,9,8,7,2,5},{2,5,7,6,4,8,9,3,1},
 {5,3,2,1,4,7,8,9,6},{8,3,1,9,4,7,5,6,2},
 {6,8,4,1,9,7,5,3,2},{6,8,4,9,3,7,2,5,1},
 {6,7,3,8,4,5,2,1,9},{5,2,1,4,8,3,6,9,7}}
```

The fitness value of the fittest individual is,

```
fu=Map[fit[#]&,u];
Max[fu]//N
0.166667
```

The string of the fittest individual itself is

```
parent=u[[First[Flatten[Position[fu,Max[fu]]]]]]
{9,8,5,6,3,2,7,4,1}
```

Now, we mutate this individual with probability μ,

```
flip[μ_]:=If[RandomReal[]≤μ,True,False]
```

After the mutation there should be 9 different numbers. We choose two positions randomly and then change their numbers,

```
Mutation[x_,pmutate_]:=Module[{x0,i1,i2,s},
 x0=x;
 If[flip[pmutate],i1=RandomInteger[{1,9}];
  i2=RandomInteger[{1,9}];
  s=x0[[i1]];
  x0[[i1]]=x0[[i2]];
  x0[[i2]]=s];
 x0];
```

The mutation of the best individual is

```
Mutation[parent,0.3]
{9,8,5,3,6,2,7,4,1}
fit[%]//N
0.0909091
```

The following function creates the subsequent generations. Its input is a generation *g* and the number of the generations *ngen*,

```
Gena[g_,ngen_]:=Module[{fi,newg,fitg,parent,maxfit,n,i},
 fi={};
 i=1;
 n=Length[g];
 newg=g;
 For[i=1,i<=ngen,i++,
  fitg=Map[fit[#]&,newg];
   maxfit=Max[fitg];
   AppendTo[fi,maxfit];
   parent=newg[[First[Flatten[Position[fitg,Max[fitg]]]]]];
If[maxfit==1,Print["Hit!"];Break[]];
     newg=Table[Mutation[parent,0.25],{n}];
   ];
 parent];
```

Let us carry out the simulation with a population of 1000 individuals. The initial generation is

```
fg=generation[1000];
```

The iteration for *ngen* = 100 results in

```
sc=Gena[fg,100]
Hit!
{9,4,7,2,1,6,8,3,5}
```

We have found a proper solution! Let us visualize it in matrix form

```
M=Table[0,{5},{5}];M[[3,1]]=sc[[1]];M[[3,2]]=sc[[2]];
M[[2,3]]=sc[[3]];M[[1,3]]=sc[[4]];M[[3,4]]=sc[[5]];
```

```
M[[3,5]] = sc[[6]]; M[[5,3]] = sc[[7]]; M[[4,3]] = sc[[8]];
M[[3,3]] = sc[[9]];
MatrixForm[M]
```

$$\begin{pmatrix} 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 9 & 4 & 5 & 1 & 6 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 \end{pmatrix}$$

It is clear that there are many different solutions:

$$S1 = \begin{pmatrix} 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 8 & 7 & 5 & 3 & 2 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 \end{pmatrix};$$

$$S2 = \begin{pmatrix} 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 2 & 3 & 5 & 7 & 8 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \end{pmatrix};$$

$$S3 = \begin{pmatrix} 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 3 & 8 & 5 & 2 & 7 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \end{pmatrix};$$

$$S4 = \begin{pmatrix} 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 8 & 2 & 5 & 3 & 7 \\ 0 & 0 & 9 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \end{pmatrix};$$

$$S5 = \begin{pmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 9 & 6 & 5 & 4 & 1 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \end{pmatrix};$$

5.4 Application to a Real Function

Let us compute the global maximum of the function $f(x)$, see Fig. (5.3) in the region of $[-40, 40]$,

$$f(x) = 1 + \frac{\cos(x)}{1 + 0.01x^2}$$

```
f[x_] := 1 + Cos[x]/(1 + 0.01x^2)
```

Now, we employ 10 bits binary representation for x_i as an individual and its fitness value $f_i = f(x_i)$.

In this case x_i represents an individual with fitness $f_i = f(x_i)$. The maximum number is $\{1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$, namely

```
maxi = 2^10 - 1
1023
```

The smallest one is $\{0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$ giving

```
mini = 2^0 - 1
0
```

The conversion can then be carried out using the function

```
Horner[u_,base_:2] := Fold[base#1 + #2&,0,u]
```

For example,

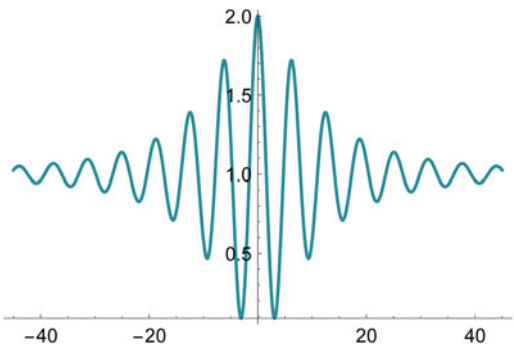


Fig. 5.3 The function $f(x)$

```
s_i = {1,0,1,0,0,0,0,0,0,0};
d_i = Horner[s_i]
640
```

Indeed,

```
2^9 + 2^7
640
```

Now, we should project this $d_i \in [0, 1023]$ interval on $x_i \in [-40, 40]$, then the corresponding value of 640 is

$$x_i = \frac{d_i 80}{1023} - 40.$$

```
10.0489
```

Considering these two steps together we get,

```
decodeBGA[chromosome_]:=
 Module[{phenotype,decimal},decimal = Horner[chromosome];
  phenotype = (decimal80)/1023 - 40.`;
  Return[phenotype]];
decodeBGA[s_i]
10.0489
```

The smallest positive value, which can be represented on 10 binary bits is,

```
decodeBGA[{1,0,0,0,0,0,0,0,0,0}]
0.0391007
```

The highest negative value is,

```
decodeBGA[{0,1,1,1,1,1,1,1,1,1}]
-0.0391007
```

So these are the best approaches for zero!

The corresponding fitness value is

```
f[%]
1.99922
```

This function will generate the initial generation,

```
initPop[psize_,csize_]:=RandomInteger[{0,1},{psize,csize}];
```

where

psize—the size of the population,
csize—individuals of the population in binary string.

Let us consider 50 individuals in the population,

```
InitialPop=initPop[15,10];
Short[InitialPop,5]
{{0,1,1,0,0,1,1,0,0,1},
 {1,0,0,1,0,0,1,1,0,1},{0,1,0,1,1,0,0,1,1,1},
 {0,0,0,0,1,0,0,0,0,0}, ≪8≫ , {0,1,0,0,1,0,1,1,1,1},
 {1,1,1,1,1,1,1,0,1,1},{1,1,1,1,0,0,0,1,0,0}}
```

The function for the mutation is,

```
mutateBit[pmute_,Bit_]:=If[flip[pmute],RandomInteger[],Bit];
```

The function for the Mutation of the Best algorithm is,

```
bgaM[pmutate_,popInitial_,fitFunction_,numGens_]:=
 Module[{i,newPop,index,pheno,f,fpheno,parent,fitness,m,
  n,Slide,p1,p2,x},
 newPop=popInitial;
 n=Length[newPop];
 f=fitFunction;
 Slide={};p1=Plot[f[x],{x,-45,45},PlotRange→{-0.5,2.1}];
 For[i=1,i<=numGens,i++,
  pheno=Map[decodeBGA,newPop];
  fpheno=f[pheno];
  p2=ListPlot[Transpose[{pheno,fpheno}],
   PlotStyle→{RGBColor[1,0,0],PointSize[0.015]}];
  AppendTo[Slide,Show[{p1,p2}]];
  fitness=Max[fpheno];
  index=Position[fpheno,fitness];
  parent=newPop[[First[Flatten[index]]]];
```

```
    Print["Generation",i,":",
     decodeBGA[parent],
     "Fitness=",fitness];
    newPop=Table[Map[mutateBit[pmutate,#]&,parent],
     {n}];];(*endofFor*)
   m=Max[Map[f[#]&,pheno]];
    {Select[pheno,f[#]==m&]//First,Slide}];
```

Let us employ it,

```
sol=bgaM[0.25,InitialPop,f,25];
Generation1:6.06061   Fitness=1.71332
Generation2:6.21701   Fitness=1.71966
Generation3:6.21701   Fitness=1.71966
Generation4:6.21701   Fitness=1.71966
Generation5:6.21701   Fitness=1.71966
Generation6:6.21701   Fitness=1.71966
Generation7:6.21701   Fitness=1.71966
Generation8:6.21701   Fitness=1.71966
Generation9:6.21701   Fitness=1.71966
Generation10:0.50831  Fitness=1.87132
Generation11:0.50831  Fitness=1.87132
Generation12:0.03911  Fitness=1.99922
Generation13:0.03911  Fitness=1.99922
Generation14:0.03911  Fitness=1.99922
Generation15:0.03911  Fitness=1.99922
Generation16:0.03911  Fitness=1.99922
Generation17:0.03911  Fitness=1.99922
Generation18:0.03911  Fitness=1.99922
Generation19:0.03911  Fitness=1.99922
Generation20:0.11730  Fitness=1.99299
Generation21:0.11730  Fitness=1.99299
Generation22:0.03911  Fitness=1.99922
Generation23:0.03911  Fitness=1.99922
Generation24:0.03911  Fitness=1.99922
Generation25:0.03911  Fitness=1.99922
```

Then, since no further improvement occurs

```
sol[[1]]
0.0391007
f[sol[[1]]]
1.99922
```

Figure 5.4 shows the positions of the individuals at the beginning and at the end of the process.
The solution with high precision is

```
Chop[NMaximize[f[x],x]]
{2.,{x→-8.18433×10^-9}}
```

To improve the method, we introduce the concept of sexuality in Sect. 5.5.

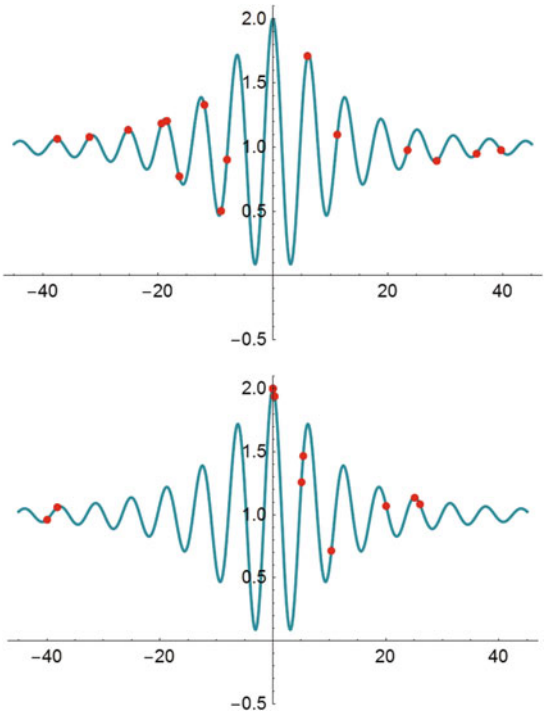


Fig. 5.4 Simulation of computing the global maximum in case of 10 bits representation at the beginning (top) and at the end of the process (bottom)

5.5 Employing Sexual Reproduction

The effectiveness of the natural selection can be improved when the principle of sexuality is introduced. Namely we employ mating of two individuals as parents and using a crossover method to interchange part of their chromosomes, resulting in two new individuals as children partly having the chromosomes of both parents.

5.5.1 Selection of Parents

The selection of the parents is carried out by the *roulette-wheel method*. In principle, we construct a roulette wheel on which each individual of the population is given a sector, whose size is proportional to the individual's fitness. Then we spin the wheel and whichever individual comes up becomes a parent. In this way the individual having higher fitness will become a parent with higher probability, but not deterministically.

Let us consider the example of a maximization function discussed in the previous section.

We generate a population of ten individuals,

```
pop = Table[RandomInteger[{0,1}],{i,10},{j,10}]
{{1,0,0,1,1,1,1,0,1,1},{0,0,1,1,1,1,0,1,1,0},
 {1,1,1,0,0,0,1,1,0,1},{1,0,1,0,0,0,0,1,0,1},
 {1,0,1,0,0,1,1,1,0,0},{1,1,1,0,0,1,1,1,0,0},
 {0,0,0,0,1,0,1,0,0,0},{1,1,0,0,0,1,0,1,1,1},
 {0,0,1,0,1,1,0,1,1,1},{1,1,0,1,1,0,1,1,1,1}}
```

Or decoding them as real numbers,

```
pheno = Map[decodeBGA,pop]
{9.65787, -20.7625,31.085,10.4399,12.2385,
 32.2581, -36.8719,21.8573, -25.6891,28.739}
```

Let us compute their fitness,

```
fitList = Map[f,pheno]
{0.496593,0.936831,1.0887,0.747582,1.37903,
 1.05838,1.04638,0.828461,1.11174,0.903452}
```

The cumulative fitness of the population is

```
fitListSum = FoldList[Plus, First[fitList], Rest[fitList]]
{0.496593, 1.43342, 2.52212, 3.2697,
 4.64873, 5.70711, 6.75349, 7.58195, 8.69369, 9.59714}
```

Then the total fitness of the population is

```
fitSum = Last[fitListSum]
9.59714
```

This function simulates the selection of the parents using roulette-wheel operation

```
selectOne[foldedFitnessList_, fitTotal_]:=
 Module[{randFitness, elem, index}, randFitness =
    RandomReal[]fitTotal;
  elem = Select[foldedFitnessList, #1 ≥ randFitness&, 1];
  index = Flatten[Position[foldedFitnessList, First[elem]]];
  Return[First[index]];];
```

The first parent is

```
parent1Index = selectOne[fitListSum, fitSum]
6
```

the second one is,

```
parent2Index = selectOne[fitListSum, fitSum]
3
```

The binary representation of the parents is,

```
parent1 = pop[[parent1Index]]
{1, 1, 1, 0, 0, 1, 1, 1, 0, 0}
parent2 = pop[[parent2Index]]
{1, 1, 1, 0, 0, 0, 1, 1, 0, 1}
```

While the fitness of the parents is,

```
fitList[[parent1Index]]
0.818018
```

and

```
fitList[[parent2Index]]
0.954068
```

They are not the fittest individuals in the population, since

```
Max[fitList]
1.37903
```

So as we mentioned, the roulette-wheel algorithm provides a bigger chance for individuals having high fitness (bigger size of selection) to be set as parents, but it will not select deterministically the two best individuals.

5.5.2 *Sexual Reproduction: Crossover and Mutation*

The crossover process will produce two new individuals as children via partly changing the chromosomes characteristics in the strings of the two parents, see Fig. 5.5. The steps of the algorithm are the following:

(1) We decide with a certain probability whether we create new individuals or just copy the parents' chromosomes, and the children become clones.
(2) In case of creating new individuals we carry out the crossover process as in Fig. 5.5.
(3) Mutation of the children

In Fig. 5.5, two parents are selected according to their fitness, and a crossover point illustrated by the vertical line is chosen by a uniform random selection. The children's chromosomes are formed by combining opposite parts of each parents' chromosomes and a crossover point illustrated by the vertical line is chosen by a uniform random selection. The children's chromosomes are formed by combining opposite parts of each parents' chromosomes.

The process described above can be evaluated using the following functions

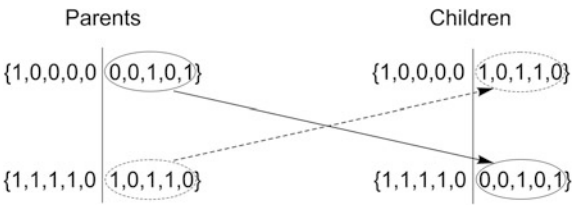


Fig. 5.5 The crossover operation

```
myXor[x_,y_]:=If[x==y,0,1];
mutateBGA[pmute_,allel_]:=If[flip[pmute],myXor[allel,1],allel];

crossOver[pcross_,pmutate_,parent1_,parent2_]:=
 Module[{child1,child2,crossAt,lchrom},lchrom=Length[parent1];
  If[flip[pcross],crossAt=RandomInteger[{1,lchrom-1}];
   child1=Join[Take[parent1,crossAt],Drop[parent2,crossAt]];
   child2=Join[Take[parent2,crossAt],Drop[parent1,crossAt]],
   child1=parent1;
   child2=parent2;];
  child1=(mutateBGA[pmutate,#1]&)/@child1;
  child2=(mutateBGA[pmutate,#1]&)/@child2;
  Return[{child1,child2}];];
```

where *pcross* is the probability of the crossover operation and *pmute* is the probability of the mutation.

Let us see an example for creating two children via crossover and the subsequent mutation,

```
pcross=0.75; pmute=0.005;
MatrixForm[children=crossOver[pcross,pmute,parent1,parent2]]
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

The real number is represented by

```
decodeList=Map[decodeBGA,children]
{32.3363,31.0068}
```

The corresponding function values (fitnesses) is

```
newfitness=Map[f,decodeList]
{1.05286,1.08644}
```

And the average is given by

```
Apply[Plus,newfitness]/2
1.06965
```

The average fitness of the parents is therefore

```
(fitList[[parent1Index]] + fitList[[parent2Index]])/2
1.07354
```

This means that the average fitness of the children is not necessarily higher than that of their parents. This technique is called the *generational replacement.*

5.6 The Basic Genetic Algorithm (BGA)

These functions can be collected in order to implement the basic genetic algorithm in *Mathematica* see Freeman (1994).

This function will print out the best individuals of the generation during the iteration process

```
displayBest[fitnessList_,number2Print_]:=
 Module[{i,sortedList},sortedList = Sort[fitnessList,Greater];
  For[i = 1,i <= number2Print,i + + ,
   Print["fitness = ",sortedList[[i]]];];(*endofFori*)];
 (*endofModule*)
```

where *number2Print* is for the number of the best individuals to be printed.

The main function is,

```
bga[pcross_,pmutate_,popInitial_,fitFunction_,numGens_,
 printNum_]:=
 Module[{i,newPop,parent1,parent2,diff,matches,
  oldPop,reproNum,index,fitList,fitListSum,fitSum,pheno,
   pIndex,pIndex2,f,children,m},
     oldPop = popInitial;(*initialize first population*)
  reproNum = Length[oldPop]/2;
   (*calculate number of reproductions*)
  f = fitFunction;(*assign the fitness function*)
  For[i = 1,i < = numGens,i + + ,(*perform numGens generations*)
   pheno = Map[decodeBGA,oldPop];(*decode the chromosomes*)
   fitList = f[pheno];(*determine the fitness of each phenotype*)
   Print[" "];(*print out the best individuals*)
   Print["Generation",i,"Best",printNum];
   Print[" "];
```

```
displayBest[fitList,printNum];
fitListSum=FoldList[Plus,First[fitList],Rest[fitList]];
fitSum=Last[fitListSum];(*find the total fitness*)
newPop=Flatten[Table[(*determine the new population*)
  pIndex1=selectOne[fitListSum,fitSum];
 (*select parent indices*)
 pIndex2=selectOne[fitListSum,fitSum];
 parent1=oldPop[[pIndex1]];(*identifyparents*)
 parent2=oldPop[[pIndex2]];
 children=crossOver[pcross,pmutate,parent1,parent2];
  (*crossover and mutate*)
 children,{reproNum}],1  (*add children to list;
  flatten to first level*)];(*end of Flatten[Table]*)
 oldPop=newPop;(*new becomes old for nextgen*)
];(*end of Fori*);
m=Max[Map[f[#1]&,pheno]];
Select[pheno,f[#]==m&]//First
];(*endofModule*)
```

where

pcross—the probability of the crossover,
pmutate—probability of the mutation of the children,
popInitial—the initial population,
fitFunction—function of fitness,
numGens—the maximum number of the generations,
printNum—the number of the best individuals of the actual generation to be printed

Pay attention to the fact that the function `decodeBGA` should be properly defined depending on the problem!

Now let us test this implementation to find the maximum of the function employed in the previous section, but this time with 12 bit representation. The initial population of 200 individuals is

```
initialPopulation=initPop[200,12];
```

Using 12 bits, then the maximum number $x_i \in [-40, 40]$ is

```
decodeBGA[chromosome_]:=
 Module[{phenotype,decimal},decimal=Horner[chromosome];
  phenotype=(decimal 80)/4095-40.`;
  Return[phenotype]];
```

Now, the approximation of the zero is

```
decodeBGA[{1,0,0,0,0,0,0,0,0,0,0,0}]
0.00976801
decodeBGA[{0,1,1,1,1,1,1,1,1,1,1,1}]
-0.00976801
f[%]
1.99995
```

Let us consider 10 generations and print out the best individual of the actual generation

```
bga[0.75,0.01,initialPopulation,f,10,1]
Generation 1  Best 1
fitness=1.99412
Generation 2  Best 1
fitness=1.99606
Generation 3  Best 1
fitness=1.99606
Generation 4  Best 1
fitness=1.99606
Generation 5  Best 1
fitness=1.99878
Generation 6  Best 1
fitness=1.99995
Generation 7  Best 1
fitness=1.99878
Generation 8  Best 1
fitness=1.99878
Generation 9  Best 1
fitness=1.99878
Generation 10  Best 1
fitness=1.99956
0.029304
```

Indeed

```
f[%]
1.99956
```

5.7 Applications

First let us solve the nonlinear parameter estimation problem discussed in the previous chapter.

5.7.1 Nonlinear Parameter Estimation

Our function is

$$f(x) = 2.5 \ \sin \ (1.5x).$$

We have noisy measurements in $x \in [-2, 8]$, see the points in Fig. 5.6.

```
f[x_]:=2.5Sin[1.5x]
XF=Table[{-2+i0.5,Random[Real,{0.7,1.3}]f[-2+i0.5]},
 {i,1,20}];
```

We are looking for the parameters α and β

$$f(x) = \alpha \ \sin \ (\beta \ \mathrm{x})$$

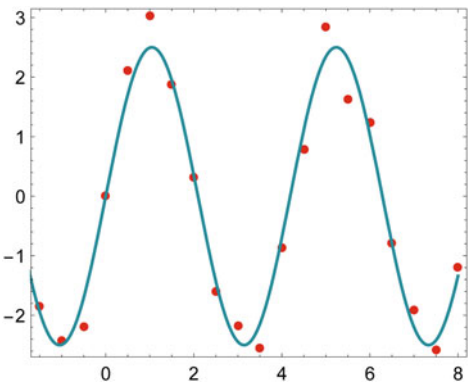


Fig. 5.6 The function and its noisy measured points

employing least squares method the function to be minimized is then (Fig. 5.7)

$$G(\alpha, \beta) = \sum_{i=1}^{n} (f_i - \alpha \sin(\beta x_i))^2$$

```
n = Length[XF];
G[{α_, β_}] := Sum[(XF[[i, 2]] - α Sin[βXF[[i, 1]]])^2, {i, 1, n}]
```

There are more local minimums, see Fig. 5.8

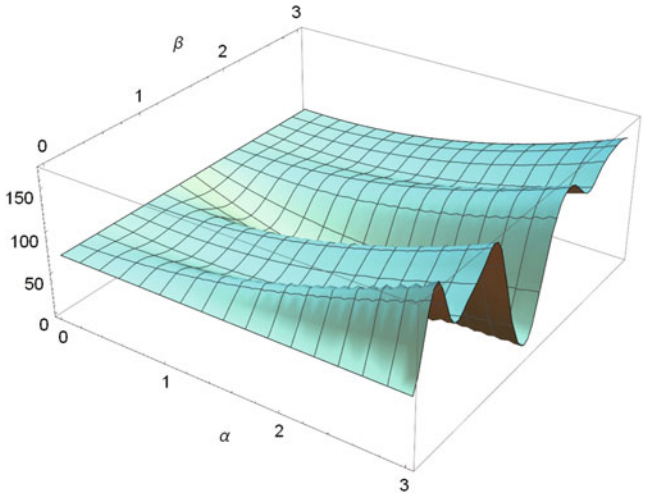


Fig. 5.7 The function to be minimized

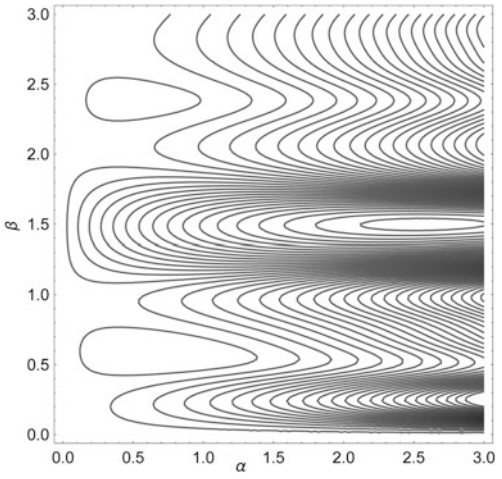


Fig. 5.8 More local minimums

Let us employ the built-in function (NMinimize) with a built-in evalutionary algorithm (Method →"DifferentialEvolution")

Remark Genetic algorithms is regarded as a subset of evalutionary algorithm.

```
sol = Reap[NMinimize[{G[{α, β}], 0 ≤ α ≤ 3, 0 ≤ β ≤ 3}, {α, β},
  EvaluationMonitor: → Sow[{α, β}],
  Method → 'DifferentialEvolution']];
```

The global minimum and its place,

```
sol[[1]]
{1.9324, {α → 2.57349, β → 1.49834}}
```

The number of iterations is, (Fig. 5.9)

```
{hist} = sol[[2]];
nh = Length[hist]
2029
```

In general, using the genetic algorithm, the neighborhood of the global minimum has been reached quickly, but to find a precise solution with high accuracy takes a long time with global method. Therefore often we combine the global and the local methods. The global method finds the neighborhood of the global minimum (processing), then the local method—which converges faster—finds the precise location of the minimum (post processing); see the next application.

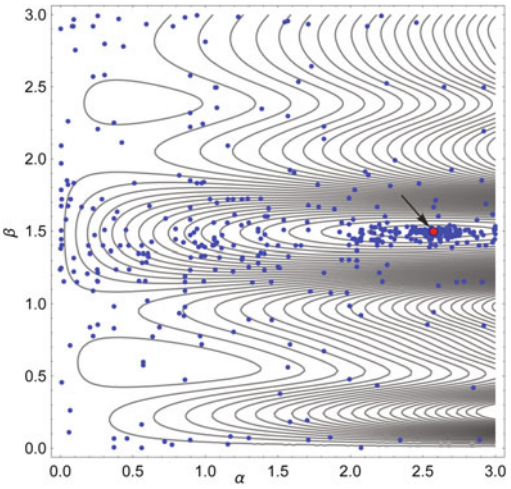


Fig. 5.9 The trial points of the genetic algorithm during the iteration process, and the global minimum (white point with arrow)

5.7.2 *Packing Spheres with Different Sizes*

Let us solve the problem discussed in the previous chapter but now in 3D! Consider five spheres with radius ranging between 0.5 and 0.8

```
n = 5; minR := 0.5; maxR := 0.8;
Table[r_i = Random[Real, {minR, maxR}], {i, 1, n}];
B = 1.5;
constraints01 =
 Table[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 ≥ (r_i + r_j)^2,
 {i, 1, n}, {j, i + 1, n}]//Flatten;
constraints02 =
 Table[{x_i + r_i - w ≤ 0, x_i - r_i + w ≥ 0, y_i + r_i - w ≤ 0, y_i - r_i + w ≥ 0,
 z_i + r_i - w ≤ 0, z_i - r_i + w ≥ 0, -B ≤ x_i ≤ B, -B ≤ y_i ≤ B, -B ≤ z_i ≤ B},
 {i, 1, n}]//Flatten;
constraints = Join[constraints01, constraints02, {0 ≤ w ≤ B}];
vars = Join[{w}, Table[{x_i, y_i, z_i}, {i, 1, n}]]//Flatten;
```

First, we try to solve the problem with the *simulated annealing* method,

$$\texttt{soln = NMinimize[Join[\{8w}^3 - \sum_{k=1}^{n} \frac{4r_k^3\pi}{3}\texttt{\}, constraints], vars,}$$

```
 Method → {"SimulatedAnnealing", "PerturbationScale" → 6,
 "SearchPoints" → 100}]//Timing
```

MessageTemplate[NMinimize , incst ,

NMinimize was unable to generate any initial points satisfying the inequality constraints $\{0.742592 - w - x_1 \le 0,\ 0.742592 - w + x_1 \le 0,\ 0.721401 - w - x_2 \le 0,\ 0.721401 - w + x_2 \le 0,$ ≪33≫, $1.18996 - (x_4 - x_5)^2 - (y_4 - y_5)^2 - (z_4 - z_5)^2 \le 0,\ 0.551895 - w - z_5 \le 0,\ 0.551895 - w + z_5 \le 0\}$. The initial region specified may not contain any feasible points. Changing the initial region or specifying explicit initial points may provide a better solution.

2 , 217 , 45 , 16816622134118316439 , Local]

Then, let us employ the *genetic algorithm with a post processing Newton's method* as a local method.

$$\texttt{soln = NMinimize[Join[\{8w}^3 - \sum_{k=1}^{n} \frac{4r_k^3\pi}{3}\texttt{\},constraints] ,vars,}$$

```
  Method → { "DifferentialEvolution", "ScalingFactor" → 0.9,
    "CrossProbability" → 0.1,
    "PostProcess" → {FindMinimum,Method → "QuasiNewton"}}]
    //Timing
```

The results show the computation time, the free volume, the size of the box, and the coordinates of the spheres.

$$\begin{aligned}
&\{15.3349,\{10.5258,\{w\to 1.26476,\\
&\quad x_1\to -0.714915, y_1\to -0.714915,\ z_1\to -0.714915,\\
&\quad x_2\to -0.481048, y_2\to 0.481048, z_2\to -0.17328,\\
&\quad x_3\to 0.679, y_3\to 0.692456, z_3\to 0.681176,\\
&\quad x_4\to -0.683168,\ y_4\to -0.683168,\ z_4\to 0.683168,\\
&\quad x_5\to 0.541146, y_5\to -0.541145, z_5\to 0.253675\}\}\}
\end{aligned}$$

The smallest results can be visualized in Fig. 5.10.

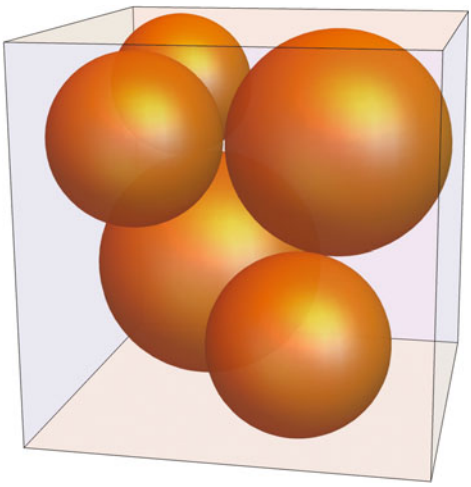


Fig. 5.10 The packed spheres with different radius

5.7.3 *Finding All the Real Solutions of a Non-algebraic System*

This is a fairly difficult problem, and in general there is no computer system that has a built-in function for that. Using brute force, however, *Mathematica* can solve this task. Let us consider the following problem see Wagon (2000),

```
f = -Cos[y] + 2yCos[y^2]Cos[2x];
g = -Sin[x] + 2Sin[y^2]Sin[2x];
```

The zero contours of the functions can be seen on Fig. 5.11. The crossing points represent the common zeros, namely the solutions of the system.

The first step is the *discretization* of the region (Fig. 5.12).

```
RR = DiscretizeRegion[Rectangle[{-3.5, -1.8}, {4, 4.2}],
  MaxCellMeasure → 0.08]
```

We compute the specifications of the subregions

```
S = MeshCoordinates[RR];
U = MeshCells[RR, 2];
V = Map[MeshRegion[S, #] &, U];
```

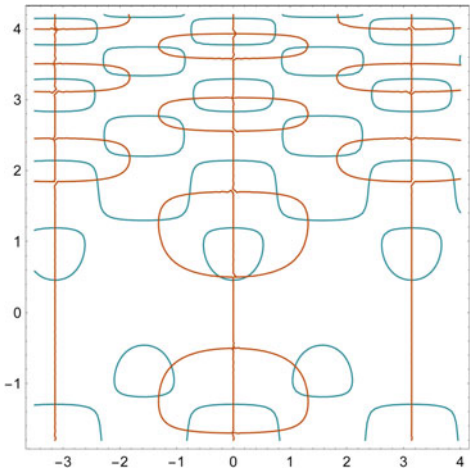


Fig. 5.11 The non-algebraic system has 67 real zeros in this region

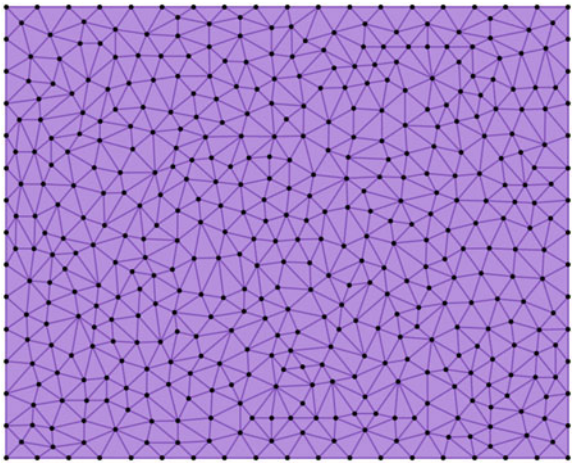


Fig. 5.12 The region is discretized

The number of the subregions

```
Length[V]
872
```

Let define the objective as

```
G = f^2 + g^2
(-Cos[y] + 2yCos[2x]Cos[y^2])^2 + (-Sin[x] + 2Sin[2x]Sin[y^2])^2
```

Over every sub-region we look for the global minimum of this objective using the Genetic Algorithm. To reduce the computation time, parallel evaluation is employed.

```
AbsoluteTiming[sol =
  ParallelMap[NMinimize[G,(x,y) ∈ #]&,V]//Chop;
```

We select the solutions where the objective is zero. It turns out that there are 67 real solutions (Fig. 5.13).

```
solv = Select[sol,(#[[1]] == 0)&];
solvv = Map[({x,y}/.#[[2]])&,solv];
Length[solvv]
67
```

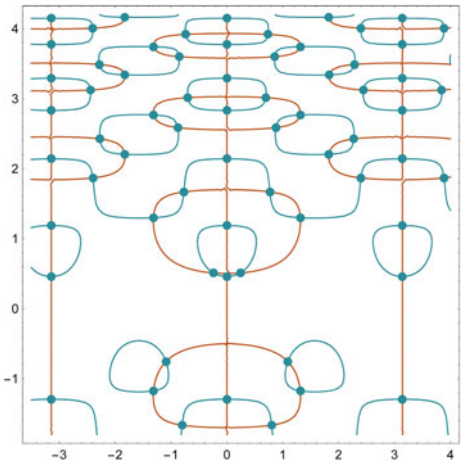


Fig. 5.13 The zeros of the system

5.8 Exercises

5.8.1 Foxhole Problem

Let us consider the *Shekels* foxhole problem. We want to find the global minimum of the following function (see Fig. 5.14).

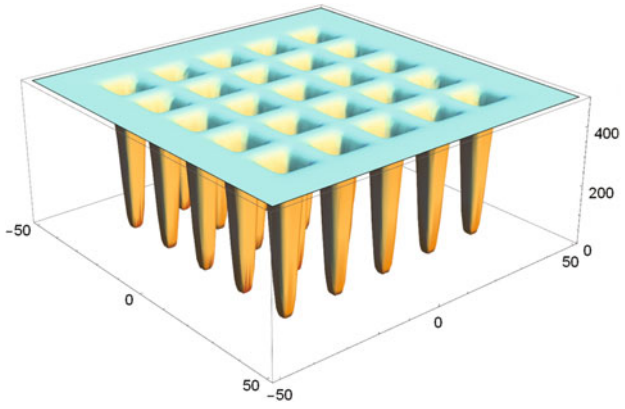


Fig. 5.14 The foxhole region

$$f = 1/\left(0.002 + \frac{1}{25 + (-32+x)^6 + (-32+y)^6} + \frac{1}{24 + (-16+x)^6 + (-32+y)^6} + \frac{1}{23 + x^6 + (-32+y)^6} + \frac{1}{22 + (16+x)^6 + (-32+y)^6} + \frac{1}{21 + (32+x)^6 + (-32+y)^6} + \frac{1}{20 + (-32+x)^6 + (-16+y)^6} + \frac{1}{19 + (-16+x)^6 + (-16+y)^6} + \frac{1}{18 + x^6 + (-16+y)^6} + \frac{1}{17 + (16+x)^6 + (-16+y)^6} + \frac{1}{16 + (32+x)^6 + (-16+y)^6} + \frac{1}{15 + (-32+x)^6 + y^6} + \frac{1}{14 + (-16+x)^6 + y^6} + \frac{1}{13 + x^6 + y^6} + \frac{1}{12 + (16+x)^6 + y^6} + \frac{1}{11 + (32+x)^6 + y^6} + \frac{1}{10 + (-32+x)^6 + (16+y)^6} + \frac{1}{9 + (-16+x)^6 + (16+y)^6} + \frac{1}{8 + x^6 + (16+y)^6} + \frac{1}{7 + (16+x)^6 + (16+y)^6} + \frac{1}{6 + (32+x)^6 + (16+y)^6} + \frac{1}{5 + (-32+x)^6 + (32+y)^6} + \frac{1}{4 + (-16+x)^6 + (32+y)^6} + \frac{1}{3 + x^6 + (32+y)^6} + \frac{1}{2 + (16+x)^6 + (32+y)^6} + \frac{1}{1 + (32+x)^6 + (32+y)^6}\right)$$

Let us employ a Quasi-Newton method as a post processing method,

```
solf =
 NMinimize[f,{{x,-50,50},{y,-50,50}},
  Method→{"DifferentialEvolution,"CrossProbability"→0.3,
  "PostProcess"→{FindMinimum,Method→"QuasiNewton"}}]//Timing
{0.187201,{0.998004,{x→-31.9786,y→-31.9783}}}
```

The global minimum value of the function value is, see Fig. 5.15

```
{{x,y,solf[[2,1]]}}/.solf[[2,2]]
{{-31.9786,-31.9783,0.998004}}
```

References

Freeman JA (1994) Simulating neural networks with *Mathematica*. Addison-Wesley, New York
Wagon S (2000) *Mathematica* in action, 2nd edn. Springer-TELOS, New York

Chapter 6
Particle Swarm Optimization

6.1 The Concept of Social Behavior of Groups of Animals

The idea of the Particle Swarm Optimization (PSO) was inspired by the social behavior of big groups of animals, e.g., flocking and schooling patterns of birds and fish as suggested by Russell and Kennedy (1995).

Imagine a flock of birds circling over an area where they can smell a hidden source of food, see Fig. 6.1. The one who is closest to the food chirps the loudest and the other birds swing around in its direction. If any of the other circling birds comes closer to the target than the first, it chirps louder and the others veer over toward it. This tightening pattern continues until one of the birds reaches the food. So the motion of the individuals of the group is influenced by the motion of their neighbors.

In the searching space, we define discrete points as individuals (particles), which are characterized by their *positional vectors* determining their *data* (*fitness*) *values* to be maximized (similar to fitness function) and their velocity vectors indicating how much the data value can change, and a *personal best value* indicating the closest the particle's data has ever come to the optimal value (target value), i.e., food in the swarming birds' example.

Let us consider a population of N individuals, with location vectors $\vec{p}_i$. The objective function to be maximized is F. The fitness of an individual is $F(\vec{p}_i)$. This value measures the attraction of the individual, namely the higher its fitness value, the more the followers in its neighborhood will follow its motion. The local velocity of the individual $\vec{v}_i$ will determine its new location after Δt time step.

The velocity value is calculated according to how far an individual's data is from the target. The further it is, the larger the velocity value. In the flocking birds example, the individuals furthest from the food would make an effort to keep up

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_6

Fig. 6.1 Swarming birds towards some source of food

with the others by flying faster toward the best bird (the individual having the highest fitness value in the population).

Each individual's personal best value only indicates the closest the particle's data has ever come to the target since the algorithm started.

The best bird value only changes when some other bird's personal best value comes closer to the target than best bird value. Through each iteration of the algorithm, the best bird value gradually moves closer and closer to the target until one of the birds reaches the target.

6.2 Basic Algorithm

The basic algorithm describes the motion of an individual of the population. Let us consider the actual position vector of an individual $\vec{x}_i(t)$, having the velocity vector $\vec{v}_i(t)$. Let us assume that the best ever position of the individual since the algorithm started is $\vec{p}_i(t)$. This means that at this position was the fitness value of the individual highest during its travel until the time point t.

The actual position of highest fitness value in the population is $\vec{g}_i(t)$.

The new velocity vector of the individual in the next time step $\vec{v}_i(t+1)$ can be computed as the linear combination of own velocity vector, $\vec{v}_i(t)$, the difference vector of his/her own best and actual position $\vec{p}_i(t) - \vec{x}_i(t)$ and the difference vector of his/her actual position and the highest fitness value position of the whole population $\vec{g}_i(t) - \vec{x}_i(t)$. This new velocity vector results in a new position $\vec{x}_i(t+1)$, see Fig. 6.2.

In Fig. 6.3 we can see the different terms of the velocity component. To make the algorithm stochastic here r_1 and r_2 are random numbers from a uniform distribution on [0, 1].

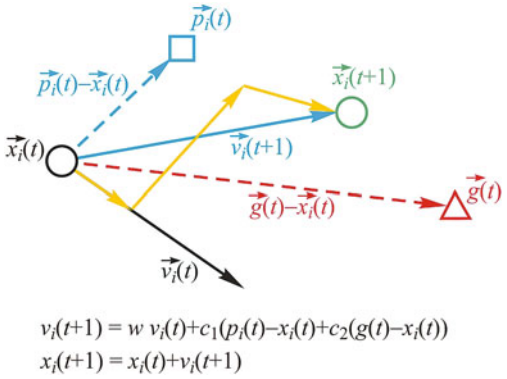


Fig. 6.2 Description of the motion of an individual

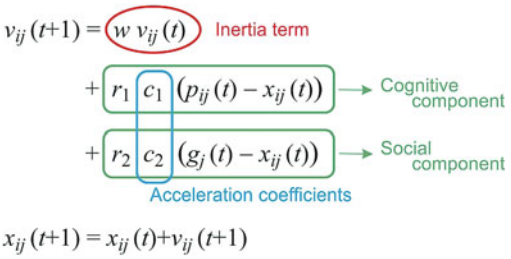


Fig. 6.3 Description of the j-th velocity component of the i-th individual

We can give the following somewhat simplified basic algorithm of the PSO:

(1) Generate the position and velocity vectors of each particle randomly

$$P = \left(\vec{p}_0, \ldots, \vec{p}_i, \ldots, \vec{p}_N\right) \quad \text{and} \quad V = \left(\vec{v}_0, \ldots, \vec{v}_i, \ldots, \vec{v}_N\right).$$

(2) Compute the fitness of the particle:

$$F(P) = \left(F\left(\vec{p}_0\right), \ldots, F\left(\vec{p}_i\right), \ldots, F\left(\vec{p}_N\right)\right).$$

(3) Store and continuously update the position of each particle where its fitness has the highest value

$$P = \left(\vec{p}_0^{best}, \ldots, \vec{p}_i^{best}, \ldots, \vec{p}_N^{best}\right).$$

(4) Store and continuously update the position of the best particle where its fitness has the highest value (global best):

$$\vec{p}_g^{\,best} = \max_{\vec{p} \in P} \left(F\left(\vec{p}\right) \right).$$

(5) Modify the velocity vectors of each particle considering its personal best and the global best

$$\vec{v}_i^{\,new} = \vec{v}_i + \varphi_1 \left(\vec{p}_i^{\,best} - \vec{p}_i \right) + \varphi_2 \left(\vec{p}_g^{\,best} - \vec{p}_i \right) \quad \text{for} \;\; 1 \le i \le N.$$

The parameters φ_1 and φ_2 are often called exploitation rates.

(6) Update the positions ($\Delta t = 1$):

$$\vec{p}_i^{\,new} = \vec{p}_i + \vec{v}_i^{\,new} \quad \text{for} \;\; 1 \le i \le N.$$

(7) Stop the algorithm if there is no improvement in the objective function during a certain number of iterations, or the number of iterations exceeds the limit.

(8) Go back to (2) and compute new fitness value with the new positions.

6.3 The Pseudo Code of the Algorithm

This is the program language-independent code of the algorithm presented by Jacob and Khemka (2004) in case of searching space dimension D:

$\vec{p}_i(t)$: position vector of the i-th particle at the time t
$\vec{p}_i^{\,best}$: best position vector of the i-th particle
$\vec{v}_i(t)$: velocity vector of the i-th particle at the time t

```
t := 0
For i = 1, …, N(number of the particles)
 / / Random generation of p_i (0) and v_i (0) for all i - th particles
 For d = 1, …, D
   p_i,d := Random [P_min,d, P_max,d]
   v_i,d := Random [V_min,d, V_max,d]
 Next d
Next i
t := 1
```

```
Loop
 For i = 1, …, N
  If F(p_i(t)) > F(p_i^best) then do / /computing the fitness value F(p_i^best)
   / /the best fitness vector of the i - th particle (to be stored)
   For d = 1, …, D
    p_{i,d}^best := p_{i,d}
   Next d
  End do
  g := i
  For k ∈ neighbours_g
   If F(p_k^best) > F(p_g^best) then g := k   / /g is the index of the fittest
    / /particle in the neighborhood of the global best particle
  Next k
  For d = 1, …, D (dimension of the searching space)
   / /updating the velocity vector of the i - the particle
   / /in the d - th dimension
   v_{i,d}(t) := v_{i,d}(t - 1) + φ_1(p_{i,d}^best - p_{i,d}(t - 1)) + φ_2(p_{g,d}^best - p_{i,d}(t - 1))
    v_{i,d} ∈ [V_{min,d}, V_{max,d}]
   / /updating the position vector of the i - the particle
   / /in the d - th dimension
   p_{i,d}(t) := p_{i,d}(t - 1) + v_{i,d}(t)
    p_{i,d} ∈ [P_{min,d}, P_{max,d}]
  Next d
  / / Now : p_i(t) = p_i(t - 1) + v_i(t)
 Next i
 t := t + 1
Until Stop
```

Let us see how to use the PSO method. First we consider a 1D problem.

6.4 Applications

First we demonstrate how this algorithm works in the case of 1D and 2D problems. Let us consider a 1D example.

6.4.1 1D Example

We are looking for the maximum of the function, see Fig. 6.5.

```
fitness[{x_}] = ( - 1.2 - cos(x^2)/(1 + 0.01x^2)) ^ -1;
F = fitness;
```

Initialization
The parameters to be initialized are the following,

```
u = 100; (*Number of Individuals*)
λ = {{ - 10,10}}; (*Location Ranges*)
μ = {{ - 0.1,0.1}}; (*Velocity ranges*)
Δ = 100; (*Number of Iterations*)
φ1 = 0.1; (*Exploitation rate*)
φ2 = 1.0; (*Exploitation rate*)
```

Running the algorithm
Running the method with *Mathematica,* recursion can be employed using Nest-List, which works as

```
NestList[G,x0,3]
{x0,G[x0],G[G[x0]],G[G[G[x0]]]}
```

Now, we use it for our function *initialPSOPopulation,* which carries out the initialization of the specifications of the individuals, and *psoStep*, which computes the new state after a discrete time step,

```
psoEvolution = NestList[psoStep[#, λ, μ, F, φ1, φ2]&,
initialPSOPopulation[u, λ, μ, F], Δ];
```

Interpretation of the result
The size of the output list is

```
Dimensions[psoEvolution]
{101,5,100}
```

The position vectors of the individuals in the first iteration (Fig. 6.4) are

```
Iteration001 = psoEvolution[[1]];Dimensions[Iteration001]
{5,100}
```

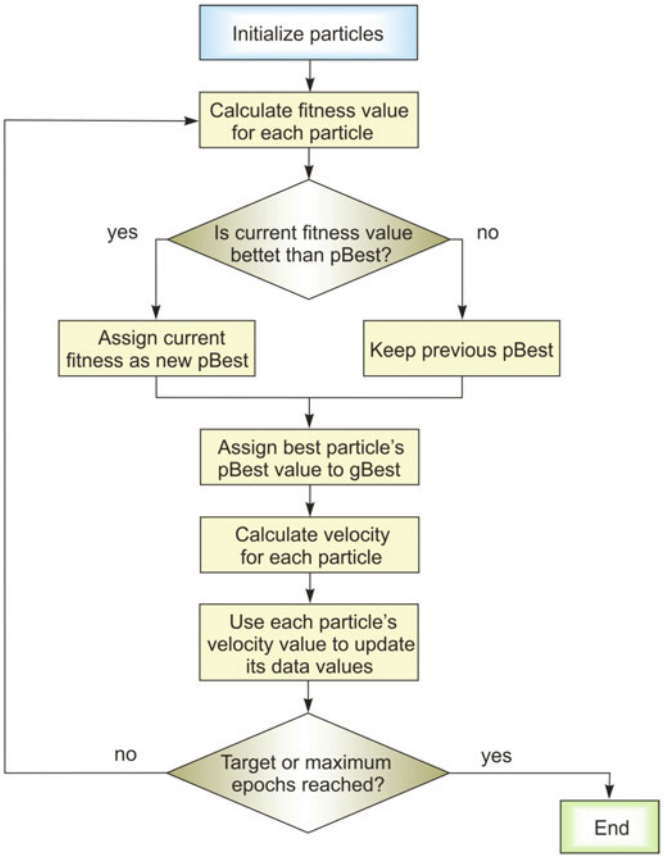


Fig. 6.4 Flow diagram illustrating the particle swarm optimization algorithm. (*Source* http://mnemstudio.org/particle-swarm-introduction.htm)

The x coordinates are

```
c1 = Iteration001[[1]]//Flatten; Short[c1,5]
{-3.91645, -2.64677, 5.85361, 3.65984,
 ≪92≫, -9.5419, 8.61003, -1.30677, -4.61456}
```

And the y coordinates, which should be minimized, namely $y(x) \equiv F(x)$ are

```
c2 = Iteration001[[3]]; Short[c2,5]
{-2.55443, -0.525978, -2.05388, -0.55667,
 ≪92≫, -1.4761, -0.728548, -0.938195, -1.76111}
```

Now we can display the positions of the individuals (Figs. 6.5 and 6.6).

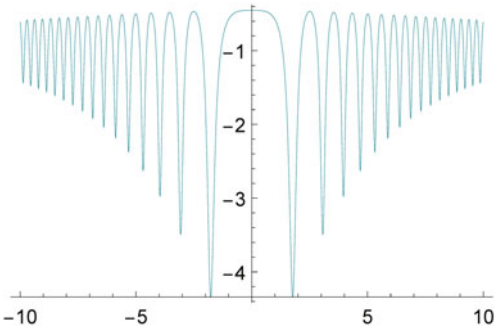


Fig. 6.5 The objective function

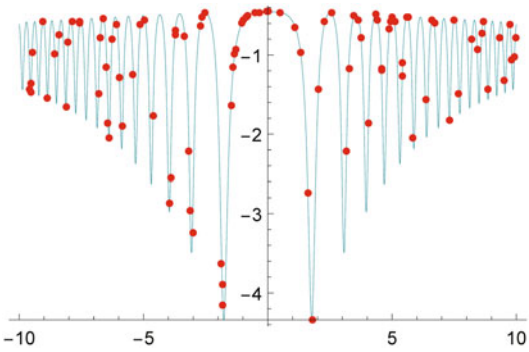


Fig. 6.6 The position vectors of the individuals in the first iteration

Figure 6.7 shows the position vectors of the individuals after 30, 60, and 100 iterations.
The position of the maximum value is

```
c1[[Position[c2,Max[c2]]//Flatten//First]]
0.000371768
```

And the value of the maximum is

```
fitness[{%}]
-0.454545
```

Alternatively using a built-in function we get

```
NMaximize[fitness[{x}],x]
-0.454545,x → 9.36564 × 10^-18
```

Remark The optimum neighborhood is very flat, see Fig. 6.8.

This means local methods working with gradient techniques can easily get stuck in the neighborhoods of the real maximum.

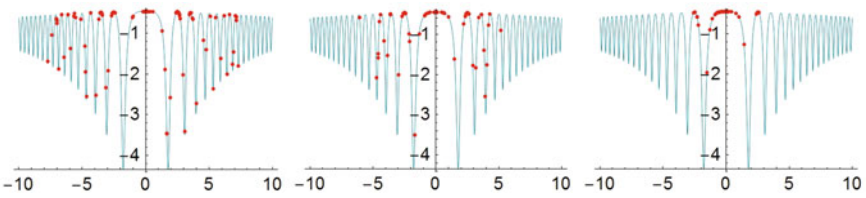
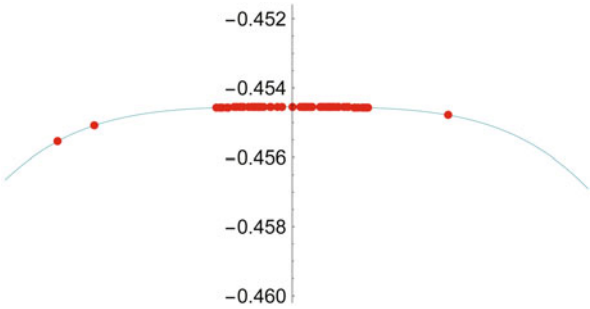


Fig. 6.7 The position vectors of the individuals after 30, 60 and 100 iterations

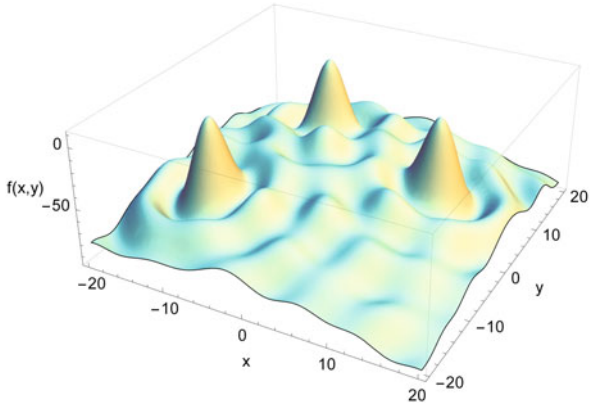
Fig. 6.8 The flocking of the individuals after 100 iteration



6.4.2 2D Example

Now we solve a standard 2D problem. The objective function to be maximized is (Fig. 6.9),

Fig. 6.9 The 2D objective function



$$f(x,y) = -\sqrt{(6+x)^2+(-9+y)^2} - \sqrt{(-11+x)^2+(-3+y)^2} - \sqrt{(11+x)^2+(9+y)^2} + \frac{50\sin\left[\sqrt{(6+x)^2+(-9+y)^2}\right]}{\sqrt{(6+x)^2+(-9+y)^2}} + \frac{50\sin\left[\sqrt{(-11+x)^2+(-3+y)^2}\right]}{\sqrt{(-11+x)^2+(-3+y)^2}} + \frac{50\sin\left[\sqrt{(11+x)^2+(9+y)^2}\right]}{\sqrt{(11+x)^2+(9+y)^2}}$$

The parameters to be initialized are

```
u = 100; (*Number of Individuals*)
λ = {{-20,20},{-20,20}}; (*Location Ranges*)
μ = {{-0.1,0.1},{-0.1,0.1}}; (*Velocity ranges*)
Δ = 300; (*Number of Iterations*)
φ1 = 0.2; (*Exploitation rate*)
φ2 = 2.0; (*Exploitation rate*)
```

Figures 6.10, 6.11 and 6.12 show the distribution of the individuals after different numbers of iterations,
The position of the maximum is

```
xyopt = c1[[Position[c2,Max[c2]]//Flatten//First]]
{-6.02194,9.04609}
```

and the value of the maximum is

```
f[%]
10.8406
```

Let us improve the global solution with a post-processing local Newton's method

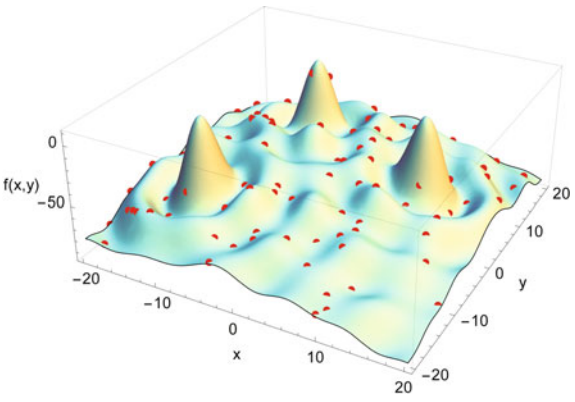


Fig. 6.10 The population distribution after the 1st iteration

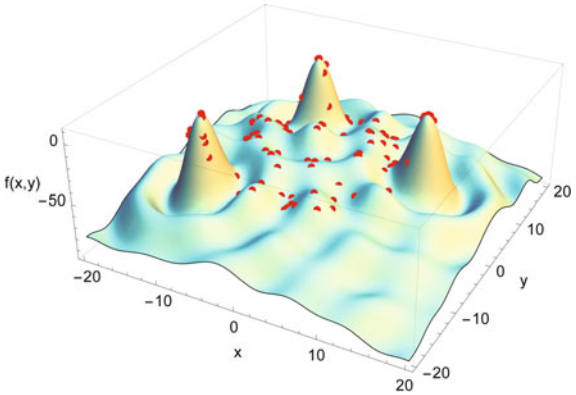


Fig. 6.11 The population distribution after the 100th iteration

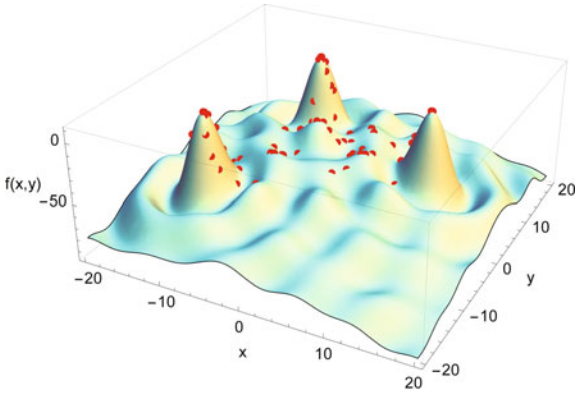


Fig. 6.12 The population distribution after the 200th iteration

```
FindMaximum[f[{x,y}],{x,xyopt[[1]]},{y,xyopt[[2]]}]//Quiet
{10.8432,{x → -6.01717,y → 9.06022}}

f[{x,y}/.%[[2]]]
10.8432
```

So we have some improvement since the value of the objective is greater. Checking the result via a built in global function,

```
NMaximize[{f[{x,y}],-20≤x≤20,-20≤y≤20},{x,y},
  Method → "SimulatedAnnealing"]
{10.8432,{x → -6.01717,y → 9.06022}}
```

We have the same value.

6.4.3 Solution of Nonlinear Non-algebraic System

Let us solve the following nonlinear system,

```
eq1 = x - y + Sin[2 x] - Cos[y];
eq2 = 4x - Exp[ - y] + 5Sin[6x - y] + 3Cos[3 y];
```

on the region of $\{x,y\} \in [-1,1] \times [-1,1]$. Figure 6.13 shows the zero contour plot of the equations. So we have one real solution in this region.

In order to get this solution, we are looking for the minimum of the objective function,

```
R = eq1^2 + eq2^2;
```

Figure 6.14 shows this error surface, $R(x,y)$.
There are two local minimums in the region, but we are looking for the solution that gives the smallest residual. So we are looking for the global minimum (Fig. 6.15).
Our PSO method will maximize the objective, therefore, one should use the objective with negative sign,

```
RR[{x, y_}] = - (eq1^2 + eq2^2);
```

The parameters are,

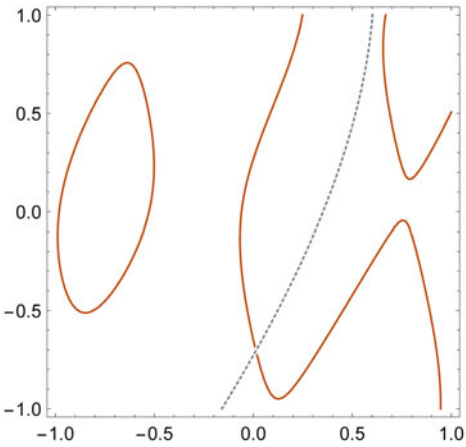


Fig. 6.13 The contour plot of eq1 = 0 and eq2 = 0

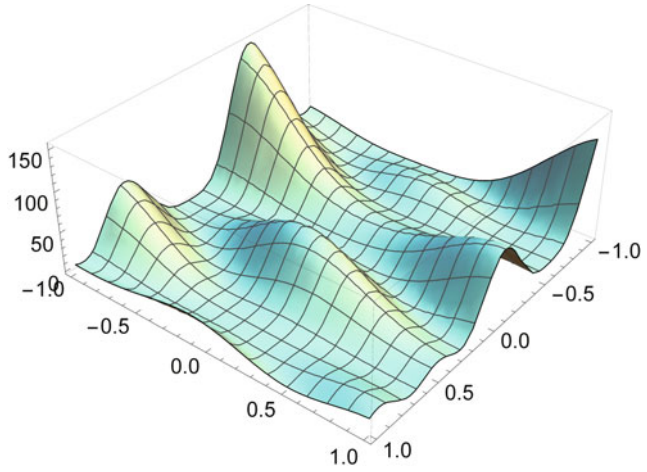


Fig. 6.14 Residual surface

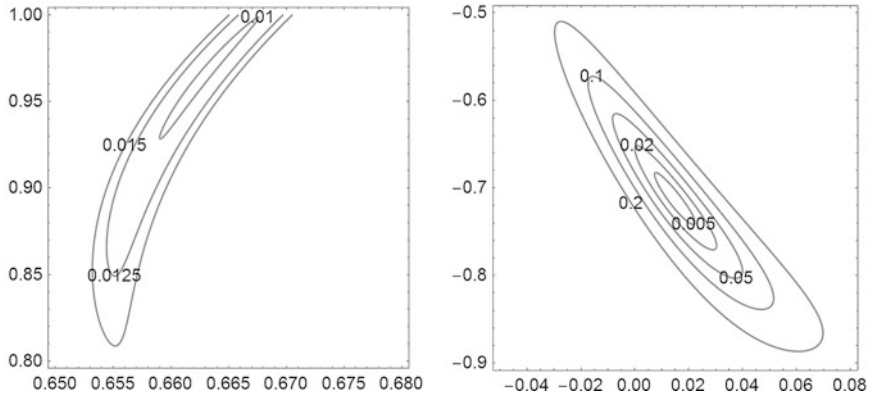


Fig. 6.15 The two local minimums of the residual surface

```
u = 300; (*Number of Individuals*)
λ = {{-1,1},{-1,1}}; (*Location Ranges*)
μ = {{-0.25,0.25},{-0.25,0.25}}; (*Velocity ranges*)
Δ = 300; (*Number of Iterations*)
φ1 = 0.2; (*Exploitation rate*)
φ2 = 2.0; (*Exploitation rate*)
```

The result, the coordinate of the location of the global minimum is

```
sol = c1[[Position[c2,Max[c2]]//Flatten//First]]
{0.00827066, - 0.6911}
```

and then the objective function is

```
R/.{x → sol[[1]],y → sol[[2]]}
0.0041124
```

Let us improve this solution by using it as an initial guess for Newton's method.

```
FindMinimum[R,{{x,sol[[1]]},{y,sol[[2]]}}]
{3.33314 × 10^-29,{x → 0.014759,y → - 0.712474}}
```

Now, to check these results, let's use the built-in function for global minimization,

```
sol = NMinimize[{R, - 1 ≤ x ≤ 1, - 1 ≤ y ≤ 1},{x,y}]
{0.00975625,{x → 0.662695,y → 0.963355}}
```

Let us improve this solution using a local method

```
FindMinimum[R,{{x,x/.sol[[2]]},{y,y/.sol[[2]]}}]
{0.00975625,{x → 0.662695,y → 0.963355}}
```

No improvement occurs since the built-in function has found the other local minimum. Now we try different types of global methods,

```
NMinimize[{R, - 1 ≤ x ≤ 1, - 1 ≤ y ≤ 1},{x,y},
  Method →"SimulatedAnnealing"]
{0.00975625,{x → 0.662695,y → 0.963355}}

NMinimize[{R, - 1 ≤ x ≤ 1, - 1 ≤ y ≤ 1},{x,y},
  Method →"DifferentialEvolution"]
{0.00975625,{x → 0.662695,y → 0.963355}}

NMinimize[{R, - 1 ≤ x ≤ 1, - 1 ≤ y ≤ 1},{x,y},
  Method →"NelderMead"]
{0.00975625,{x → 0.662695,y → 0.963355}}
```

All of them have failed. However *RandomSearch* method is successful,

```
NMinimize[{R, - 1 ≤ x ≤ 1, - 1 ≤ y ≤ 1},{x,y},
  Method →"RandomSearch"]
2.90974 × 10^-21,x → 0.014759,y → - 0.712474
```

Once again, if the proper global method can approach the neighborhood of the global minimum, then a local method can improve the result.

6.5 Exercise

Let us consider the following optimization problem,

```
f = xy^2 z
```

where *f* should be minimized under the following equality constraints

```
const1 = -6 + x^2 + y^2 + z^2
const2 = x + y - z
```

Let us employ the built-in function

```
NMinimize[{f, const1 == 0, const2 == 0}, {x, y, z}]
{-4., {x → -1., y → 2., z → 1.}}
```

Using different methods, one gets

```
NMinimize[{f, const1 == 0, const2 == 0}, {x, y, z},
  Method → "RandomSearch"]
4.89607 × 10^-59, {x → 1.73205, y → 4.03983 × 10^-30, z → 1.73205}

NMinimize[{f, const1 == 0, const2 == 0}, {x, y, z},
  Method → "DifferentialEvolution"]
{-4., {x → -1., y → 2., z → 1.}}

NMinimize[{f, const1 == 0, const2 == 0}, {x, y, z},
  Method → "SimulatedAnnealing"]
{-4., {x → -1., y → 2., z → 1.}}

NMinimize[{f, const1 == 0, const2 == 0}, {x, y, z},
  Method → "NelderMead"]
{-4., {x → -1., y → 2., z → 1.}}
```

Now, all of the methods except *RandomSearch* provide the proper solution. Note that different methods work differently on different problems! Let us employ algebraic minimization working on algebraic numbers. Using Lagrange method the constrained problem can be transformed into an unconstrained one,

```
Minimize[{f,const1 == 0,const2 == 0},{x,y,z}]
xy^2z + ( - 6 + x^2 + y^2 + z^2)λ1 + (x + y - z)λ2
```

The necessary conditions are

```
p1 = D[F,x]
y^2z + 2xλ1 + λ2

p2 = D[F,y]
2xyz + 2yλ1 + λ2

p3 = D[F,z]
xy^2 + 2zλ1 - λ2

p4 = D[F,λ1]
 - 6 + x^2 + y^2 + z^2

p5 = D[F,λ2]
x + y - z
```

Applying reduced Gröbner basis for x is

```
grx = GroebnerBasis[{p1,p2,p3,p4,p5},,y,z,λ1,λ2},{y,z,λ1,λ2}]
{27 - 90x^2 + 93x^4 - 34x^6 + 4x^8}
```

The roots of the polynomial of x are

```
solx = Solve[grx == 0,x]//Flatten
{x → - 1,x → 1,x → - √3,x → √3,x → - 1/2 √(3(3 - √5)),
 x → 1/2 √(3(3 - √5)),x → - 1/2 √(3(3 + √5)),x → 1/2 √(3(3 + √5))}
```

Similarly for y,

```
gry = GroebnerBasis[{p1,p2,p3,p4,p5},{x,y,z,λ1,λ2},,z,λ1,λ2}]
12y - 11y^3 + 2y^5
```

```
soly = Solve[gry == 0,y]//Flatten
{y → - 2,y → 0,y → 2,y → - √(3/2),y → √(3/2)}
```

and for z,

```
grz = GroebnerBasis[{p1,p2,p3,p4,p5},{x,y,z,λ1,λ2},{x,y,λ1,λ2}]
{27 - 90z^2 + 93z^4 - 34z^6 + 4z^8}
solz = Solve[grz == 0,z]//Flatten
{z → -1,z → 1,z → -√3,z → √3,z → -1/2 √(3(3 - √5)),
 z → 1/2 √(3(3 - √5)),z → -1/2 √(3(3 + √5)),z → 1/2 √(3(3 + √5))}
```

All of the combinations of the solutions

```
tp = Tuples[{solx,soly,solz}];Short[tp,5]
{{x → -1,y → -2,z → -1},{x → -1,y → -2,z → 1},
  {x → -1,y → -2,z → -√3},{x → -1,y → -2,z → √3},
  ≪313≫,{x → 1/2 √(3(3 + √5)),y → √(3/2),z → 1/2 √(3(3 - √5))},
  {x → 1/2 √(3(3 + √5)),y → √(3/2),z → -1/2 √(3(3 + √5))},
  {x → 1/2 √(3(3 + √5)),y → √(3/2),z → 1/2 √(3(3 + √5))}}
```

Some of them will not satisfy the constraint. To select solutions that satisfies them,

```
tpS = Select[tp,(Abs[const1/.#] + Abs[const2/.#]) == 0&]//Quiet
{{x → -1,y → 2,z → 1},{x → 1,y → -2,z → -1},
  {x → -√3,y → 0,z → -√3},{x → √3,y → 0,z → √3}}
```

Now we are looking for the solutions that provide the minimum, the lowest value for the objective

```
fv = Map[f/.#&,tpS]
{-4,-4,0,0}
```

Then

```
fvm = Min[fv]
-4
```

We have two such solutions,

```
posm = Position[fv,fvm]//Flatten
{1,2}

solopt = Map[tp[[#]]&,posm]
{{x → -1,y → -2,z → -1},{x → -1,y → -2,z → 1}}
```

Symbolic solution can provide all the solutions, without iteration and without round-off error!

Reference

Jacob C, Khemka N (2004) Particle swarm optimization: an exploration kit for evolutionary optimization. In: Mitic P, Jacob C, Carne J (eds) New ideas in symbolic computation: proceedings of the sixth international mathematica symposium (IMS'04), Banff, Alberta, Canada. Positive Corporation Ltd. Hampshire, UK. library.wolfram.com/infocenter/Conferences/6039

Chapter 7
Integer Programming

7.1 Integer Problem

Frequently there exist optimization problems where integer valued solutions are desired. Let us consider a simple production planning example where a furniture factory produces three products (bookcase, desks and chairs) using different combinations of three limited resources (machining, finishing, and labor). Resource requirements and unit products are listed in Table 7.1.

The furniture maker's problem is to utilize the available resources (capacity) to produce the best combination of bookcases, chairs and desks: one that achieves the largest possible total profit.

Let us employ linear programming to maximize the resources using the data in Table 7.1, i.e., net profit versus constraint (finish, labor and machine)

```
profit = 3x + y + 3z;
constraints = {2x + 2y + z ≤ 30 && x + 2y + 3z ≤ 25 && 2x + y + z ≤ 26 && x ≥ 0 && y ≥ 0 && z ≥ 0};
```

The constraints indicate that one should not be allowed to exceed the capacity limits and the number of the different products cannot be negative.

Looking for the solution on real numbers, we get,

```
sol = Maximize[{profit, constraints}, {x, y, z}]
```

$$\left\{\frac{231}{5}, \left\{x \to \frac{53}{5}, y \to 0, z \to \frac{24}{5}\right\}\right\}$$

or

```
sol//N
{46.2, {x → 10.6, y → 0., z → 4.8}}
```

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_7

Table 7.1 Resource requirements, capacity and profit in a furniture production

	Bookcases (x)	Chairs (y)	Desks (z)	Capacity
Finishing	2	2	1	30
Labor	1	2	3	25
Machining	2	1	1	26
Net profit	3	1	3	–

Which are *rounded* off to integer result,

```
solr = MapThread[#2 → Round[#1[[2]]] &, {sol[[2]], {x, y, z}}]
{x → 11, y → 0, z → 5}
```

but now the constraints are not satisfied,

```
constraints/.solr
False
```

An alternative way to get integer solution is to use *floor*, we get

```
solf = MapThread [#2 → Floor [#1[[2]]] &, {sol[[2]], {x, y, z}}]
{x → 10, y → 0, z → 4}
```

The constraints are okay,

```
constraints/.solf
True
```

and the profit is

```
profit/.solf
42
```

Is there a better integer solution in which profit is closer to 46.2? If we employ *integer programming* (IP), we get

```
Maximize[
 {profit, 2x + 2y + z <= = 30 && x + 2y + 3z ≤ 25 && 2x + y + z ≤ 26 && 
 x ≥ 0 && x ≥ 0 && y ≥ 0 && z ≥ 0 && Element[x | y | z, Integers]},
 {x, y, z}]
 {45, {x → 10, y → 0, z → 5}}
```

This example illustrates that one needs a special method to achieve the best integer result. Neither rounding nor flooring the real results will ensure the best profit.

7.2 Discrete Value Problems

A special situation is when the solution can be only a discrete value (which is not necessarily an integer) belonging to a given set. Let us consider the following objective function,

$$p = 0.2\,x_1 + 0.3\,x_2 + 0.23\,x_3;$$

The constraint is

$$c_0 = x_1 + x_2 + x_3 \leq 15.5;$$

The possible candidates for the solution are in the following sets (not interval),

$$x_1 \in \{1.2, 2.8\};$$
$$x_2 \in \{1.4, 3.8\};$$
$$x_3 \in \{2.5, 3.8, 4.2, 5.8\};$$

Every discrete programming problem can be transformed into a binary (0/1) *programming problem!*

Let us introduce the following binary variables $\delta_{i,j}$

$$x_1 = 1.2\,\delta_{1,1} + 2.8\,\delta_{1,2};$$
$$x_2 = 1.4\,\delta_{2,1} + 3.8\,\delta_{2,2};$$
$$x_3 = 2.5\,\delta_{3,1} + 3.8\,\delta_{3,2} + 4.2\,\delta_{3,3} + 5.8\,\delta_{3,4};$$

Only one value from the discrete domains can be assigned to each x_i, therefore we have constraints

$$c_1 = \delta_{1,1} + \delta_{1,2} == 1;$$
$$c_2 = \delta_{2,1} + \delta_{2,2} == 1;$$
$$c_3 = \delta_{3,1} + \delta_{3,2} + \delta_{3,3} + \delta_{3,4} == 1;$$

Now, substituting the new expressions of x_1, x_2 and x_3 into the objective p, the new objective function becomes

$$P = p // \text{Expand}$$
$$0.24\,\delta_{1,1} + 0.56\,\delta_{1,2} + 0.42\,\delta_{2,1} + 1.14\,\delta_{2,2} + 0.575\,\delta_{3,1} + 0.874\,\delta_{3,2} + 0.966\,\delta_{3,3} + 1.334\,\delta_{3,4}$$

and similarly the new constraints are

$$C_0 = c_0 // \text{Expand}$$
$$1.2\,\delta_{1,1} + 2.8\,\delta_{1,2} + 1.4\,\delta_{2,1} + 3.8\,\delta_{2,2} + 2.5\,\delta_{3,1} + 3.8\,\delta_{3,2} + 4.2\,\delta_{3,3} + 5.8\,\delta_{3,4} \leq 15.5.$$

The new variables are $\delta_{i,j} \in \{0, 1\}$,

```
var = {δ1,1, δ1,2, δ2,1, δ2,2, δ3,1, δ3,2, δ3,3, δ3,4};
```

There are two constraints for the variables. They should be integers,

```
c4 = Element [var, Integers]
(δ1,1|δ1,2|δ2,1|δ2,2|δ3,1|δ3,2|δ3,3|δ3,4) ∈ Integers
```

and their values should be 0 or 1,

```
c5 = Apply [And, Map [0 ≤ # ≤ 1 &, var]]
```

Since c_4 should be true, therefore c_5 is a proper constraint

```
0 ≤ δ1,1 ≤ 1 && 0 ≤ δ1,2 ≤ 1 && 0 ≤ δ2,1 ≤ 1 && 0 ≤ δ2,2 ≤ 1 &&
  0 ≤ δ3,1 ≤ 1 && 0 ≤ δ3,2 ≤ 1 && 0 ≤ δ3,3 ≤ 1 && 0 ≤ δ3,4 ≤ 1
```

Then all of the constraints are

```
constraints = Apply [And, {c0, c1, c2, c3, c4, c5}]
1.2 δ1,1 + 2.8 δ1,2 + 1.4 δ2,1 + 3.8 δ2,2 +
  2.5 δ3,1 + 3.8 δ3,2 + 4.2 δ3,3 + 5.8 δ3,4 ≤ 15.5 &&
  δ1,1 + δ1,2 = 1 && δ2,1 + δ2,2 = 1 && δ3,1 + δ3,2 + δ3,3 + δ3,4 = 1 &&
  ( δ1,1|δ1,2|δ2,1|δ2,1|δ3,1|δ3,2|δ3,3|δ3,4 ) ∈ Integers &&
  0 ≤ δ1,1 ≤ 1 && 0 ≤ δ1,2 ≤ 1 && 0 ≤ δ2,1 ≤ 1 && 0 ≤ δ2,2 ≤ 1 &&
  0 ≤ δ3,1 ≤ 1 && 0 ≤ δ3,2 ≤ 1 && 0 ≤ δ3,3 ≤ 1 && 0 ≤ δ3,4 ≤ 1
```

Therefore the solution is

```
sol = Maximize [Rationalize [{P, constraints}], var]
{1517/500, {δ1,1 → 0, δ1,2 → 1, δ2,1 → 0,
 δ2,2 → 1, δ3,1 → 0, δ3,2 → 0, δ3,3 → 0, δ3,4 → 1}}
```

which means

```
x1/.sol [[2]]
2.8

x2/.sol [[2]]
3.8

x3/.sol[[2]]
5.8
```

7.3 Simple Logical Conditions

Let us consider a periodic investment problem. There are 3 time periods and 4 different investment possibilities. The maximum budgets that are at our disposal in the different time periods are 18, 16 and 19 m$, respectively. The different investments need the following sums of money in the different time periods,

I. investment: 6, 9, 3
II. investment: 8, 0, 11
III. investment: 0, 5, 7
IV. investment: 4, 5, 6.

The profits of the different investments at the end of the third period are: 16, 22, 12, 8. Which investment can be and should be made in order to maximize the total profit?

Let $x_i = 1$ if the i-th investment takes place, and $x_i = 0$ otherwise. Then the following 0/1 programming problem can be stated:

```
profit = 16x1 + 22x2 + 12x3 + 8x4
16x1 + 22x2 + 12x3 + 8x4
```

The constraints for the budget in the different time periods are

```
constraints1 = {6x1 + 8x2 + 0x3 + 4x4 ≤ 18, 9x1 + 0x2 + 5x3 +
  5x4 ≤ 16, 3x1 + 11x2 + 7x3 + 6x4 ≤ 19};
```

In addition there are further constraints:

(1) No more than two investments can take place at the same time,

```
x1 + x2 + x3 + x4 ≤ 2,
```

(2) If the second investment takes place then the fourth one should be also realized,

```
x2 - x4 ≤ 0, since if x2 = 1 then x4 = 1, or
 if x2 = 0 then x4 = 0, or if x4 = 1 and x2 = 0
```

(3) If the first one takes place, then the third one should not be realized

```
x1 + x3 ≤ 1, since if x1 = 1 then x3 = 0, or if x1 = 0 then
  x3 = 0, or x3 = 1 and x1 = 0
```

Let us summarize these additional constrains as,

```
constraints2 = {x1 + x2 + x3 + x4 ≤ 2, x2 - x4 ≤ 0, x1 + x3 ≤ 1};
vars = Table [xi, {i, 1, 4}]
{x1, x2, x3, x4}
```

The variables are between 0 and 1,

```
constraints3 = Map [0 <= # <= 1 &, vars]
{0 ≤ x1 ≤ 1, 0 ≤ x2 ≤ 1, 0 ≤ x3 ≤ 1, 0 ≤ x4 ≤ 1}
```

We put these constrains together with the restriction that variables should be integers,

```
constraints = Apply [And, Join [constraints1, constraints2,
 constraints3, {Element [vars, Integers}]]
6x1 + 8x2 + 4x4 ≤ 18 && 9x1 + 5x3 + 5x4 ≤ 16 && 3x1 + 11x2
 + 7x3 + 6x4 ≤ 19 &&
 x1 + x2 + x3 + x4 ≤ 2 && x2 - x4 ≤ 0 && x1 + x3
 ≤ 1 && 0 ≤ x1 ≤ 1 &&
 0 ≤ x2 ≤ 1 && 0 ≤ x3 ≤ 1 && 0 ≤ x4
 ≤ 1 && (x1|x2|x3|x4) ∈ Integers
```

Now let us carry out the maximization,

```
sol = Maximize [{profit, constraints}, vars]
{30, {x1 → 0, x2 → 1, x3 → 0, x4 → 1}}
```

In a more *general* way, the following problem can be handled via introducing binary variables. Let us consider the constraints as:

$$\sum_{j=1}^{n} a_j x_j = b.$$

There are k different possible values for b: b_1,..., b_k

$$\sum_{j=1}^{n} a_j x_j = b_i \quad (i = 1, \ldots, k),$$

but only one of them can be true. This can be expressed as,

$$\sum_{j=1}^{n} a_j x_j = \sum_{i=1}^{k} b_i y_i$$

and

$$\sum_{i=1}^{k} y_i = 1$$

where

$$y_i = 0 \text{ or } 1, \quad (i = 1, \ldots, k).$$

7.4 Some Typical Problems of Binary Programming

7.4.1 Knapsack Problem

We have n objects, and the j-th object has a weight a_j which is a positive integer and the value of the object is p_j. We should select the most valuable subset of the objects, having a total weight less than b.

The total value of the objects

$$\max \sum_{j=1}^{n} p_j x_j$$

should be maximized to satisfy the constraint,

$$\sum_{j=1}^{n} a_j x_j \leq b,$$

where x_j is equal to 1 if the object is selected and 0 otherwise,

$$x_j \in \{0, 1\}.$$

Example Given a ferry which can transport 5t load as maximum. There are three vehicles to be transported: a car weighting 2t, a van 4t, and a motorcycle 1t.

The fares are 300, 400, and 100 $ respectively. How can we get the highest profit?

Our objective function to be minimized is

```
profit = 300 x + 400 y + 100 z;
```

Under the constraint imposed by the capacity of the ferry

```
constraints =
 {2 x + 4 y + 1 z ≤ 5 && 0 ≤ x ≤ 1 && 0 ≤ y ≤ 1 && 0 ≤ z ≤ 1 &&
 Element [x|y|z, Integers]};
```

Then

```
Maximize [{profit, constraints}, {x, y, z}]
{500, {x → 0, y → 1, z → 1}}
```

This therefore means that the ferry will bring van and the motorcycle and his profit will be $ 500.

7.4.2 Nonlinear Knapsack Problem

The nonlinear form of the profit function is,

$$\max \sum_{j=1}^{n} f_{\mathrm{j}}(x_j).$$

Assuming that the constrains are

$$\sum_{j=1}^{n} g_j(x_j) \leq b,$$

and

$$0 \leq x_j \leq d_j,$$

where x_j is integer.

Example Now we have a nonlinear objective as

```
profit = x1^3 + 3x2 + 0.25x3^4
x1^3 + 3x2 + 0.25x3^4

vars = Table[xi, {i, 1, 4}]
{x1, x2, x3, x4}

constraints = Join[{3x1^2 + 8x2 + x3^3 + x4 == 10}, Map[0 ≤ # ≤ 5 &, vars],
 Element[vars, Integers]]
{3x1^2 + 8x2 + x3^3 + x4 == 10, 0 ≤ x1 ≤ 5, 0 ≤ x2 ≤ 5,
 0 ≤ x3 ≤ 5, 0 ≤ x4 ≤ 5, (x1|x2|x3|x4) ∈ Integers}
```

Then

```
sol = Maximize[Rationalize[{profit, constraints}], vars]
{4, {x1 → 0, x2 → 0, x3 → 2, x4 → 2}}
```

7.4.3 Set-Covering Problem

Let us consider a deployment problem of fire-stations in a town that has the following districts (Fig. 7.1).

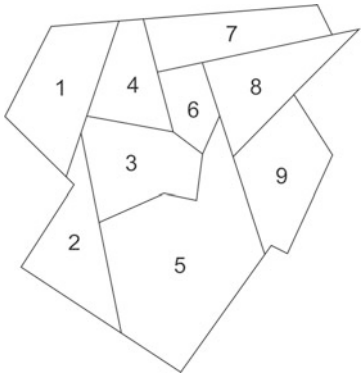


Fig. 7.1 District boundaries

It is desired to erect the fewest fire stations that can handle the fires of all districts assuming that a fire station can operate in the neighboring district, too. This is a so called *set covering problem.*

Let us consider the neighborhood matrix **A,** which can be defined as follows $A_{i,j} = 1$ if the i-th district and the j-th district has common border or $i = j$
$A_{i,j} = 0$ otherwise. In our case $\boldsymbol{A}$ is

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix};$$

Now the decision variable x_j can be defined as follows: $x_j = 1$ if in the j-th district a fire station will be built, and $x_j = 0$ otherwise.

The variables are

```
vars = Table[x_i, {i, 1, 9}]
{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9}
```

The objective function to be minimized is,

```
objective = Apply[Plus, vars]
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9
```

In every district or at least in its neighborhood, there should be a fire station,

```
constraints1 = Map [(# ≥ 1) &, A.vars];
constraints1 // TableForm
x1 + x2 + x3 + x4 ≥ 1
x1 + x2 + x3 + x5 ≥ 1
x1 + x2 + x3 + x4 + x5 + x6 ≥ 1
x1 + x3 + x4 + x6 + x7 ≥ 1
x2 + x3 + x5 + x6 + x8 + x9 ≥ 1
x3 + x4 + x5 + x6 + x7 + x8 ≥ 1
x4 + x6 + x7 + x8 ≥ 1
x5 + x6 + x7 + x8 + x9 ≥ 1
x5 + x8 + x9 ≥ 1
```

The usual constraints for the variables are:

```
constraints2 = Map[0 ≤ # ≤ 1 &, vars]
{0 ≤ x1 ≤ 1, 0 ≤ x2 ≤ 1, 0 ≤ x3 ≤ 1, 0 ≤ x4 ≤ 1,
  0 ≤ x5 ≤ 1, 0 ≤ x6 ≤ 1, 0 ≤ x7 ≤ 1, 0 ≤ x8 ≤ 1, 0 ≤ x9 ≤ 1}
```

Let us join the constraints,

```
constraints = Join [constraints1, constraints2,
 {Element [vars, Integers]}]
{x1 + x2 + x3 + x4 ≥ 1, x1 + x2 + x3 + x5 ≥ 1, x1 + x2 + x3 + x4 + x5 + x6 ≥ 1,
x1 + x3 + x4 + x6 + x7 ≥ 1, x2 + x3 + x5 + x6 + x8 + x9 ≥ 1,
x3 + x4 + x5 + x6 + x7 + x8 ≥ 1, x4 + x6 + x7 + x8 ≥ 1,
x5 + x6 + x7 + x8 + x9 ≥ 1, x5 + x8 + x9 ≥ 1, 0 ≤ x1 ≤ 1, 0 ≤ x2 ≤ 1,
0 ≤ x3 ≤ 1, 0 ≤ x4 ≤ 1, 0 ≤ x5 ≤ 1, 0 ≤ x6 ≤ 1, 0 ≤ x7 ≤ 1, 0 ≤ x8 ≤ 1,
0 ≤ x9 ≤ 1, (x1|x2|x3|x4|x5|x6|x7|x8|x9) ∈ Integers}


Minimize [{objective, constraints}, vars]
{2, {x1 → 0, x2 → 0, x3 → 1, x4 → 0, x5 → 0, x6 → 0, x7 → 0, x8 → 1, x9 → 0}}
```

The 5th and 6th districts are lucky since they have even two neighboring districts with fire stations, see Fig. 7.2.

7.5 Solution Methods

7.5.1 Binary Countdown Method

This is a brute force method, which can be useful and effective in case of small sized problems.

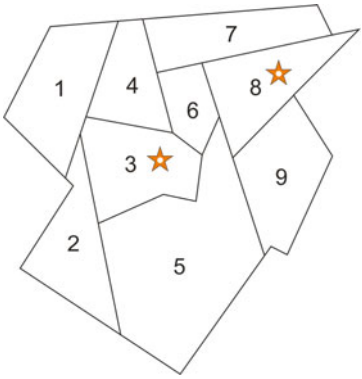


Fig. 7.2 Solution of the fire-station allocation problem

Let us consider the following 0/1 programming problem

```
profit = -8x1 - 2x2 - 4x3 - 7x4 - 5x5 + 10
constraints = Apply [And, { -3x1 - 3x2 + x3 + 2x4 + 3x5 ≤ -2,
 -5x1 - 3x2 - 2x3 - x4 + x5 ≤ -4}]
 -3x1 - 3x2 + x3 + 2x4 + 3x5 ≤ -2 && -5x1 - 3x2 - 2x3 - x4 - x5 ≤ -4
```

We generate all the possible combinations

```
total = Tuples [{0,1},5]
{{0,0,0,0,0},{0,0,0,0,1},{0,0,0,1,0},{0,0,0,1,1},
 {0,0,1,0,0},{0,0,1,0,1},{0,0,1,1,0},{0,0,1,1,1},
 {0,1,0,0,0},{0,1,0,0,1},{0,1,0,1,0},{0,1,0,1,1},
 {0,1,1,0,0},{0,1,1,0,1},{0,1,1,1,0},{0,1,1,1,1},
 {1,0,0,0,0},{1,0,0,0,1},{1,0,0,1,0},{1,0,0,1,1},
 {1,0,1,0,0},{1,0,1,0,1},{1,0,1,1,0},{1,0,1,1,1},
 {1,1,0,0,0},{1,1,0,0,1},{1,1,0,1,0},{1,1,0,1,1},
 {1,1,1,0,0},{1,1,1,0,1},{1,1,1,1,0},{1,1,1,1,1}}
Length [%]
32
```

Then select those combinations, which satisfy the constraints

```
candid = Select [total, constraints/.{x1 → # [[1]],
 x2 → # [[2]], x3 → # [[3]], x4 → # [[4]], x5 → # [[5]]} &]
{{0,1,1,0,0},{1,0,0,0,0},{1,0,1,0,0},
 {1,1,0,0,0},{1,1,0,0,1},{1,1,0,1,0},
 {1,1,1,0,0},{1,1,1,0,1},{1,1,1,1,0}}
```

Finally, choose the eligible solution which maximizes the objective

```
sols = Map [profit/.{x1 → # [[1]], x2 → # [[2]],
  x3 → # [[3]], x4 → # [[4]], x5 → # [[5]]} &, candid]
{4, 2, -2, 0, -5, -7, -4, -9, -11}

candid [[First [Position [sols, Max [sols]]]]]
{{0, 1, 1, 0, 0}}
```

Employing built in function

```
constraints =
  Apply [And, {constraints, 0 ≤ x1 ≤ 1, 0 ≤ x2 ≤ 1, 0 ≤ x3 ≤ 1,
  0 ≤ x4 ≤ 1, 0 ≤ x5 ≤ 1, Element [{x1, x2, x3, x4, x5}, Integers]}]
-3x1 - 3x2 + x3 + 2x4 + 3x5 ≤ -2 && -5x1 - 3x2 - 2x3 - x4 + x5 ≤ -4 &&
  0 ≤ x1 ≤ 1 && 0 ≤ x2 ≤ 1 && 0 ≤ x3 ≤ 1 && 0 ≤ x4 ≤ 1 && 0 ≤ x5 <= 1 &&
  (x1|x2|x3|x4|x5) ∈ Integers

Maximize [{profit, constraints}, {x1, x2, x3, x4, x5}]
{4, {x1 → 0, x2 → 1, x3 → 1, x4 → 0, x5 → 0}}
```

The result is the same.

7.5.2 Branch and Bound Method

Let us see another technique to solve the task. In this case the value of the original problem can be reduced to the series of solutions of real optimization problems. We shall demonstrate the algorithm with the following example. Consider the following simple integer programming problem.

The objective function is

```
p = 8x1 + 5x2;
```

The constraints are

```
cI = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 &&
  Element [x1|x2, Integers]};
```

The variables

```
var = {x1, x2};
```

Now we are looking for the solution on the real numbers

```
cR = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0};
```

Step 1.

```
Maximize[{p,cR},var]
```

$$\left\{\frac{165}{4}, \left\{x_1 \to \frac{15}{4}, x_2 \to \frac{9}{4}\right\}\right\}$$

In this case, the objective is 165/4 = 41.25, which should be smaller in integer case. Considering that $x_1 = 3.75$ let us investigate the cases $x_1 \leq 3$ and $x_1 \geq 4$ (branching)

Step 2.

case 2/a:

```
cRL = {x1 + x2 ≤ 6 && 9 x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 && x1 ≤ 3};
Maximize[{p,cRL},var]
{39,{x1 → 3,x2 → 3}}
```

Since both solutions are integer this is a candidate bounding and the bound is 39

case 2/b:

```
cRU = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 && x1 ≥ 4};
Maximize[{p,cRU},var]
```

$$\left\{41, \left\{x_1 \to 4, x_2 \to \frac{9}{5}\right\}\right\}$$

Since this a non-integer solution with bound: 41 > 39, → branching → step 3 (see Fig. 7.3).

Step 3.

branching: let us consider the cases $x_2 \leq 1$ and $x_2 \geq 2$ since $x_2 = 9/4 = 1.8$.

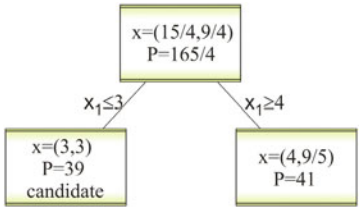


Fig. 7.3 First branch and bound (candidate)

case 3/a:

```
cRL = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 && x2 ≤ 1};
Maximize[{p,cRL},var]
```

$$\left\{\frac{365}{9}, \left\{x_1 \to \frac{40}{9}, x_2 \to 1\right\}\right\}$$

Since we have solution and the objective: 365/9 = 40.5556 > 39, but the solution is not integer → branching → step 4, see Fig. 7.4.

case 3/b:

```
cRU = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 && x2 ≥ 2};
Maximize[{p,cRU},var]
```

$$\left\{\frac{165}{4}, \left\{x_1 \to \frac{15}{4}, x_2 \to \frac{9}{4}\right\}\right\}$$

This was the original real solution → infeasible solution → dead end (see Fig. 7.4)

```
165/4//N
41.25
```

Step 4.

branching: consider the cases $x_1 \leq 4$ and $x_1 \geq 5$

case 4/a:

```
cRL = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 && x1 ≤ 4};
Maximize[{p,cRL},var]
```

$$\left\{\frac{165}{4}, \left\{x_1 \to \frac{15}{4}, x_2 \to \frac{9}{4}\right\}\right\}$$

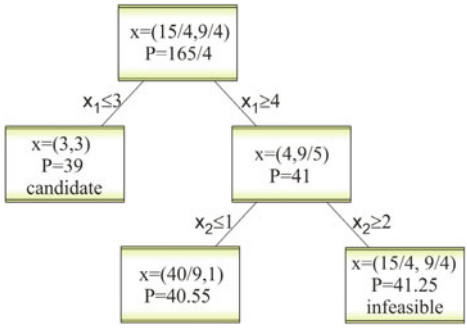


Fig. 7.4 Second branch and bound (infeasible)

```
165/4//N
41.25
```

Neither of them is integer → infeasible solution → dead end

case 4/b:

```
cRU = {x1 + x2 ≤ 6 && 9x1 + 5x2 ≤ 45 && x1 ≥ 0 && x2 ≥ 0 && x2 ≥ 5};
Maximize[{p,cRU},var]
{40,{x1 → 5,x2 → 0}}
```

Both solutions are integer → candidate with bound: 40 → dead end.

The candidate solution is the solution belonging to the maximum bound $\{x_1 \to 5, x_2 \to 0\}$

Indeed, using built-in function,

```
Maximize[{p,cI},var]
{40,{x1 → 5,x2 → 0}}
```

The procedure can be represented by a binary tree, see Fig. 7.5.

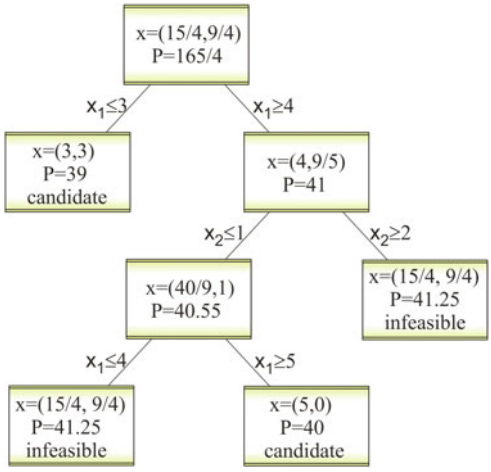


Fig. 7.5 Last branch and bound (candidate)

7.6 Mixed–Integer Programming

Sometimes the solution of a programming problem can be partly real and partly integer. If our computer system is not prepared for such problems, we can employ the following approximation.

Let us consider the following problem

```
3x1 + 5x2 → max
x1 + 2x2 ≤ 9.75
3x1 + 4x2 ≤ 21.75
x1, x2 ≥ 0
x2 ∈ integer
```

Let us introduce a new integer variable ξ_1, for the real variable x_1

```
ξ1 = 100x1
```

Then our constraints are

```
constraints = Rationalize[
 Apply [And, {10^-2 ξ1 + 2x2 ≤ 9.75, 3 × 10^-2 ξ1 + 4x2 ≤ 21.75,
 0 ≤ ξ1, 0 ≤ x2, Element [{ξ1, x2}, Integers]}]]
```

$$2x_2 + \frac{\xi_1}{100} \le \frac{39}{4} \;\&\&\; 4x_2 + \frac{3\xi_1}{100} \le \frac{87}{4} \;\&\&\; 0 \le \xi_1 \;\&\&\; 0 \le x_2 \;\&\&\; (\xi_1 | x_2) \in \text{Integers}$$

And the objective function is

```
objective = 3 × 10^-2 ξ1 + 5x2;
```

Let us get the solution of this integer programming problem

```
sol = Maximize [{objective, constraints}, {ξ1, x2}] // N
{25.25, {ξ1 → 175., x2 → 4.}}
```

Then we convert ξ_1 integer value back to x_1 real one.

```
x1 = ξ1 10^-2 /. sol [[2]]
1.75
```

7.7 Applications

7.7.1 Integer Least Squares

A typical problem is the computation of the integer least-squares estimates of the GNSS cycle ambiguities, see Teunissen (1995) i.e., the integer least-squares problem.

Suppose we know the floating point vector Z and the covariance matrix of the errors Q. We seek the integer vector z, which minimize the following objective,

$$(z - Z)^T Q^{-1} (z - Z) \to \min_{z},$$

So we are looking for an integer vector Z, which is closest to Z, but in the meantime, we consider the measure error of every element of the vector Z. The variables are the coordinates of the unknown integer vector.

```
var = Table[z_i, {i, 1, 3}]
```

$$\{z_{123}\}$$

```
Z = {{2.5602}, {-2.3256}, {-4.9076}}; Q = {{4.0722, -1.5606, 0.1293}, {-1.5606, 1.0943, 0.5632}, {0.1293, 0.5632, 1.6399}};
```

Let us rationalize Z and Q

```
ZR = Rationalize[Z]]; MatrixForm[ZR]
```

$$\begin{pmatrix} \frac{12801}{5000} \\ -\frac{2907}{1250} \\ -\frac{12269}{2500} \end{pmatrix}$$

```
QR = Rationalize[Q]]; MatrixForm[QR]
```

$$\begin{pmatrix} \frac{20361}{5000} & -\frac{7803}{5000} & \frac{1293}{10000} \\ -\frac{7803}{5000} & \frac{10943}{10000} & \frac{352}{625} \\ \frac{1293}{10000} & \frac{352}{625} & \frac{16399}{10000} \end{pmatrix}$$

then the objective function in our case is

```
objective = ((z_1  z_2  z_3) - Transpose[ZR] // Flatten).
 Inverse[QR].({{z_1}, {z_2}, {z_3}} - ZR) // Simplify // First
```

$$\frac{1}{4441351063357500\,(369337082550000002_1^2}$$
$$30000z_1(-444839980877 + 438674950000z_2 - 170070485000z_3) +$$
$$3(26434540496430823 + 5551068575000000z_2^2 + 11210835046450000z_3 +$$
$$16839467500000002_3^2 - 20000z_2(291064806319 + 207937385000z_3)))$$

The constraints can be defined as

```
constraints = Apply[And, {-10 < z1 < 10, -10 < z2 < 10, -10 < z3 < 10,
  Element[{z1, z2, z3}, Integers]}]
-10 < z1 < 10 && -10 < z2 < 10 && -10 < z3 < 10 && (z1|z2|z3) ∈ Integers
```

So this is a quadratic integer programming problem. The solution is,

```
sol = Minimize[{objective, constraints}, {z1, z2, z3}]//N
{0.15277, {z1 → 2., z2 → -2., z3 → -5.}}
```

However, employing a floating point solution and simple rounding of it, we would get

```
NMinimize[{objective, Apply[And,
  {-10 < z1 < 10, -10 < z2 < 10, -10 < z3 < 10}]}, {z1, z2, z3}]
{-3.60251 × 10^-15, {z1 → 2.5602, z2 → -2.3256, z3 → -4.9076}}
```

and blindly rounding it, we would get

```
objective/.{z1 → 3., z2 → -2., z3 → -5.}
1.12359
```

It is integer, but farther from Z.

7.7.2 Optimal Number of Oil Wells

There are five oil-bearing traps, see Shi (2014). The oil revenues of a single well in these different areas are 540, 375 240, 135 and 60, respectively. How many wells should be drilled in the different traps (x_i, i = 1, 2, 3, 4, 5) to maximize the total revenue?

```
profit = 540x1 + 375x2 + 240x3 + 135x4 + 60x5
540x1 + 375x2 + 240x3 + 135x4 + 60x5
```

We have the following constraints,

(a) 31 wells to be drilled in total

```
x1 + x2 + x3 + x4 + x5 = 31
```

(b) Minimum one well should be drilled in any trap,

```
1 ≤ xi, i = 1, 2, 3, 4, 5
```

(c) The following relations are among the number of wells drilled in the different traps,

```
x1 = 2x2 - 1,
x2 = 2x3 - 1,
x3 = x4 - 2x5
```

Consequently the constraints are,

```
constraints = Apply[And,
 {x1 + x2 + x3 + x4 + x5 = = 31 1 = = 2x2 - 1, x2 = = 2x3 - 1,
 x3 = = x4 - 2x5, 1 ≤ x1, 1 ≤ x2, 1 ≤ x3, 1 ≤ x4, 1 ≤ x5,
 Element[{x1, x2, x3, x4, x5}]}]
x1 + x2 + x3 + x4 + x5 = = 31 && x1 = = - 1 + 2x2 && x2 = = - 1 + 2x3 && 
 x3 = = x4 - 2x5 && 1 ≤ x1 && 1 ≤ x2 && 1 ≤ x3 && 1 ≤ x4 && 1 ≤ x5 && 
 (x1|x2|x3|x4|x5) ∈ Integers
```

The results is

```
Maximize[{profit, constraints}, {x1, x2, x3, x4, x5}]
{11475, {x1 → 13, x2 → 7, x3 → 4, x4 → 6, x5 → 1}}
```

If we drill the wells 13, 7, 4, 6 and 1, in different traps, respectively, then our total revenue will be 11475.

7.8 Exercises

7.8.1 Study of Mixed Integer Programming

We have seen that such problems can be handled by introducing new variables of the floating point variables like,

$$\xi = 10^n x,$$

where $x \in R$ (real) and $\xi \in Z$ (integer). The question is, which is the best value for the exponent n? Let us investigate this problem in the following mixed integer quadratic 0/1 programming problem!

Considering the objective to be maximized is,

```
f = y1 + y2 + y3 + 5x1^2
```

The constraints are

```
g1 = 3x1 - y1 - y2 ≤ 0
g2 = - x1 - 0.1y2 + 0.25y3 ≤ 0
g3 = 0.2 - x1 ≤ 0
```

The integer variables y_i, $i = 1, 2, 3$ can be 0 or 1,

```
vars = {y1, y2, y3};
constraints1 = Map [0 ≤ # ≤ 1 &, vars]
{0 ≤ y1 ≤ 1, 0 ≤ y2 ≤ 1, 0 ≤ y3 ≤ 1}
```

Since x_1 is a floating point variable, first we introduce a new integer variable ξ as,

$$\xi = 10x_1.$$

Then we have further constraints,

```
constraints2 =
  {3ξ10^-1 - y1 - y2 ≤ 0, - ξ10^-1 - 0.1y2 + 0.25y3 ≤ 0, 0.2 - ξ10^-1 ≤ 0} //
  Rationalize
```

$$\left\{\frac{3\xi}{10} - y_1 - y_2 \le 0,\ -\frac{\xi}{10} - \frac{y_2}{10} + \frac{y_3}{4} \le 0,\ \frac{1}{5} - \frac{\xi}{10} \le 0\right\}$$

Combining all the constraints, we get

```
constraints = Apply [And,
 Join [constraints1, constraints2, {Element [{y1, y2, y3, ξ},
        Integers]}]]
```

$$0 \le y_1 \le 1 \ \&\&\ 0 \le y_2 \le 1 \ \&\&\ 0 \le y_3 \le 1 \ \&\&\ \frac{3x}{10} - y_1 - y_2 \le 0 \ \&\&\ -\frac{x}{10} - \frac{y_2}{10} + \frac{y_3}{4} \le 0 \ \&\&\ \frac{1}{5} - \frac{x}{10} \le 0 \ \&\&\ (y_1|y_2|y_3|\xi) \in \text{Integers}$$

```
objective = y1 + y2 + y3 + 5(ξ10^-1)^2
ξ^2/20 + y1 + y2 + y3
Maximize [{objective, constraints}, {y1, y2, y3, ξ}]
```

$$\left\{\frac{24}{5}, \{y_1 \to 1, y_2 \to 1, y_3 \to 1, \xi \to 6\}\right\}$$

Increasing $n, n = 2, 3, \ldots, \infty$ we get more precise results for x1, see Table 7.2.

Table 7.2 shows how the result can change, i.e., x1 = 0.6, with the increasing exponent.

Table 7.2 The precission $x1$ depending of n

n	Objective	x_1
1	24/5 = 4.8	0.6
2	2589/500 = 5.178	0.66
…	…	…
∞	47/9 = 5.22…2	2/3 = 0.66…6

7.8.2 Mixed Integer Least Square

The mixed integer least square problem in case of a linear model can be formulated as follows,

$$(y - Ax - Bz)^T Q^{-1}(y - Ax - Bz) \to \min_{x,z},$$

where y, A, B and Q are known real vector and matrices, and x is a real and z is an integer unknown vector,

$$x \in R \text{ and } z \in Z.$$

Let us consider the actual values of the input arrays as

$$\mathtt{y} = \begin{pmatrix} 2.5602 \\ -2.3256 \end{pmatrix}; \ \mathtt{Q} = \begin{pmatrix} 4.0722 & -1.5606 \\ -1.5606 & 1.0945 \end{pmatrix};$$

$$\mathtt{A} = \begin{pmatrix} 1.2 & 0.97 \\ 0.1 & 1.99 \end{pmatrix}; \ \mathtt{B} = \begin{pmatrix} 2.1 & 0.12 \\ 3.12 & 0.5 \end{pmatrix};$$

Now, two new integer variables for elimination of the real unknown will be introduced as,

$\xi_1 = 100\mathtt{x}_1, \ \xi_2 = 100\mathtt{x}_2$

Then rationalizing the input,

```
{yR,QR,AR,BR} = Rationalize[{y,Q,A,B}]
```

$$\left\{\left\{\left\{\frac{12801}{5000}\right\}, \left\{-\frac{2907}{1250}\right\}\right\}, \left\{\left\{\frac{20361}{5000}, -\frac{7803}{5000}\right\}, \left\{-\frac{7803}{5000}, \frac{2189}{2000}\right\}\right\}, \left\{\left\{\frac{6}{5}, \frac{97}{100}\right\}, \left\{\frac{1}{10}, \frac{199}{100}\right\}\right\}, \left\{\left\{\frac{21}{10}, \frac{3}{25}\right\}, \left\{\frac{78}{25}, \frac{1}{2}\right\}\right\}\right\}$$

the objective function,

```
objective =
 ((yR - AR.{ξ1 10^-2, ξ2 10^-2} - BR.{z1, z2})//Flatten).Inverse[QR].
 (( 12801/5000 ) - AR.( ξ1 10^-2 ) - BR.( z1 ))//Simplify//First
    -2907/1250     ξ2 10^-2        z2
      1
2021550540000
(10614556745028 + 64917271080000 z1^2 + 1221082800000 z2^2 + 30802339440 ξ1 +
199134600 ξ1^2 + 233946969684 ξ2 + 1192491360 ξ1 ξ2 + 2318098663 ξ2^2 +
2400 z2 (2363910996 + 10969210 ξ1 + 44240009 ξ2) +
6000 z1 (6272889330 + 2950454880 z2 + 33997596 ξ1 + 129192397 ξ2))
```

In order to be able to define the constraints properly it is useful to solve the floating point problem first,

```
NMinimize[objective, {z1, z2, ξ1, ξ2}]
{-1.23667 × 10^-13, {z1 → 2.25206, z2 → -17.966, ξ1 → 14.477, ξ2 → -19.2707}}
```

Then the constraints can be written concerning the result of floating point problem, namely,

```
constraints =
 Apply[And, {0 < z1 < 5, -20 < z2 < 0, 0 < ξ1 < 20, -25 < ξ2 < 0,
 Element[{z1, z2, ξ1, ξ2}]}]
0 < z1 < 5 && -20 < z2 < 0 && 0 < ξ1 < 20 && -25 < ξ2 < 0 && (z1 | z2 | ξ1 | ξ2) ∈ Integers
```

This is a quadratic integer programming problem; its solution is,

```
Minimize[objective, constraints, {z1, z2, ξ1, ξ2}]
{16527800527/2021550540000, {z1 → 2, z2 → -17, ξ1 → 19, ξ2 → -1}}

%//N
{0.0081758, {z1 → 2., z2 → -17., ξ1 → 19., ξ2 → -1.}}
```

Therefore $x_1 = 0.19$ and $x_2 = -0.01$.

References

Teunissen PJG (1995) The least-squares ambiguity decorrelation adjustment: a method for fast GPS integer ambiguity estimation. J Geodesy 70:65–82

Shi G (2014) Data mining and knowledge discovery for geoscientests. Elsevier, Amsterdam

Chapter 8
Multiobjective Optimization

8.1 Concept of Multiobjective Problem

8.1.1 Problem Definition

It frequently happens that we have two or more competing objectives in case of an optimization problem. As the independent variables change, the value of one of the objectives increases on the one hand, while the value of the other one decreases on the other hand. In this case, we get a set of optimal solutions instead of only a single one. In general, the user should then select his/her favorable solution from this set.

Let us start with a simple example. During the processing of a certain type of product, some harmful material is released. The aim is two-fold: increase the profit by increasing the amount of the product, but decrease the contamination by decreasing the amount of the product. These two objectives are competing. In our case, we have two different products (x_1 and x_2). Suppose the profit objective happens to be

$$z_1 = 140x_1 + 160x_2 \rightarrow \max$$

This should be maximized, while the amount of cost caused by the release of the harmful material should be minimized, let's say

$$z_2 = 500x_1 + 200x_2 \rightarrow \min$$

Since the problem is linear, some restrictions should be introduced. Suppose they are

$$\begin{gathered} 2x_1 + 4x_2 \leq 28 \\ 5x_1 + 5x_2 \leq 50 \\ 0 \leq x_1 \leq 8 \\ 0 \leq x_2 \leq 6. \end{gathered}$$

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_8

8.1.2 *Interpretation of the Solution*

The feasible solutions can be visualized on Fig. 8.1. In a linear programming problem like this, we know the answer is always one of the vertices.
Table 8.1 contains the feasible solutions and the corresponding values of the objectives.
In the last column, "yes" means that at least one of the objectives z_1 or z_2 in that row has a good value relative to other rows. The feasible solutions B and C are so called "dominated" or "inefficient" solutions, since there is at least one other feasible solution that is better than these, in the sense that at least one of its objective values is better and the other is not worse, see Table 8.1.
Feasible solutions, which are not dominated solutions are called as "non-inferior" or "efficient" solutions, and all of them are candidates for being an optimal solution, see Fig. 8.2.

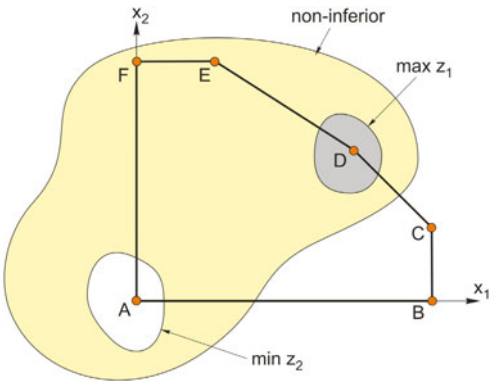


Fig. 8.1 Feasible solutions in the (x_1, x_2)—parameter space

Table 8.1 Feasible solutions and their objective values

Feasible solution	x_1	x_2	z_1	z_2	Suitability
A	0	0	0	0	Yes
B	8	0	1120	4000	D and E better
C	8	2	1440	4400	D better
D	6	4	1480	3800	Yes
E	2	6	1240	2200	Yes
F	0	6	960	1200	Yes

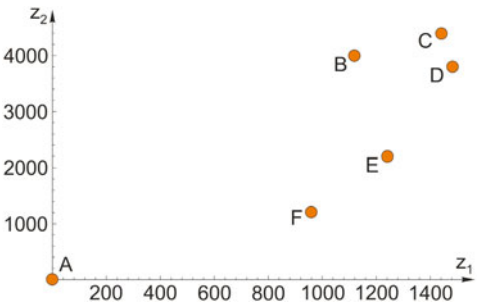


Fig. 8.2 Objectives (z_1, z_2)—function space

8.2 Pareto Optimum

The (x_1, x_2) values of the feasible, efficient solutions (A, F, E, D) are a *Pareto set*, while the corresponding objective values (z_1, z_2) form a *Pareto front*, see Fig. 8.3. The region of feasible independent variables is called as *parameter* or *decision* space, and the region of the corresponding function values is called as *function* or *criterion* space.

Conclusions

(a) In case of multiple objectives, there are optimal solutions that are not considered to be solutions of the problem. We call these feasible solutions *inferiority* or *inefficient* solutions.
(b) There are other feasible solutions that can be considered candidates for the solution of the problem. We call these *non-inferiority* or *efficient* solutions.

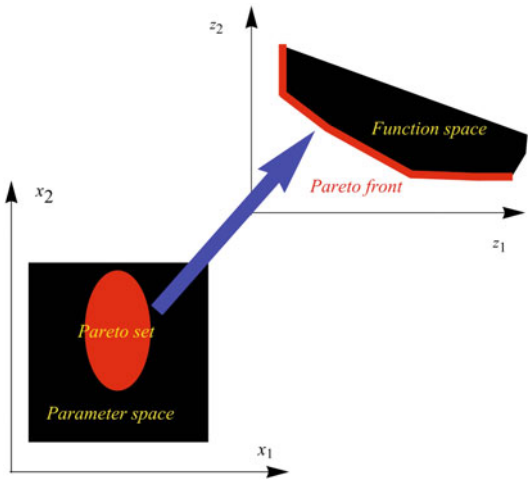


Fig. 8.3 Pareto front (z_1, z_2) and Pareto set

8.2.1 Nonlinear Problems

Let us consider two nonlinear objectives J_1 (x, y) and J_2 (x, y), where

$$J_1 = 3(1-x)^2 e^{-x^2-(1+y)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - 3e^{-(x+2)^2-y^2} + 0.5(2x+y)$$

$$J_2 = 3(1+y)^2 e^{-y^2-(1+(-x))^2} - 10\left(-\frac{y}{5} + y^3 - (-x)^5\right)e^{-y^2-(-x)^2} - 3e^{-(-y+2)^2-(-x)^2}$$

see Fig. 8.4.

The problem is to find the maximum of this multi-objective system. As a first naive approach let us compute the maximum of each of the objectives, and as a compromise, consider the maximum of the sum of the objectives in the parameter space, $(x, y) \in [-3, 3]^2$, see Fig. 8.5.

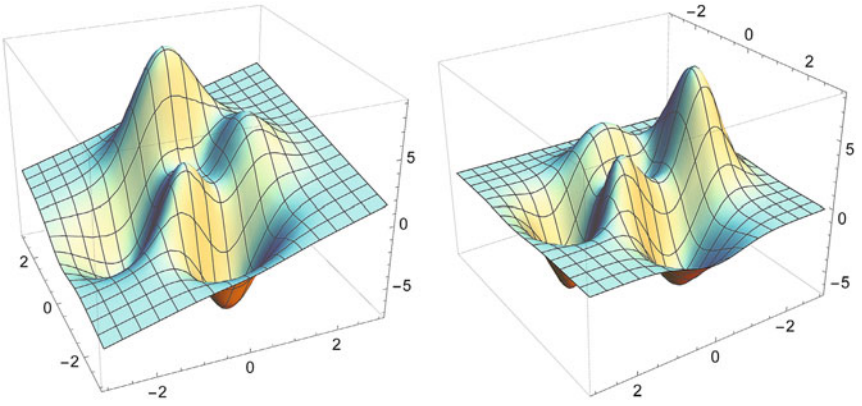


Fig. 8.4 $J_1(x, y)$ and $J_2(x, y)$ nonlinear functions

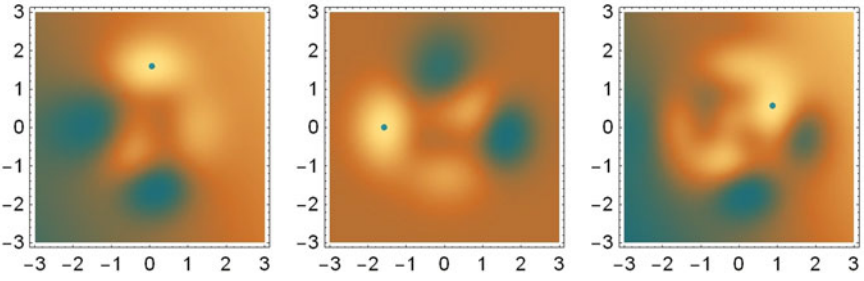


Fig. 8.5 Location of the global maximums of $J_1(x, y)$, $J_2(x, y)$ and their sum $J(x, y)$

In order to display the feasible function space, let us generate 10^4 random points in the parameter space and compute the corresponding points of the function space. Figure 8.6 shows the parameter space and the corresponding function space with the three optimal points.

8.2.2 *Pareto-Front and Pareto-Set*

We call an optimal solution in the decision space *efficient* if by changing the position of this solution a bit, in order to increase the value of one of the optimal solutions, leads to a decrease of the value of the other objective. The corresponding points in the criterion space are called Pareto front; see the upper boundary points of the function set in Fig. 8.7 on the right hand side. However not all of these boundary points belong to an efficient solution, e.g., fixing the value of J_2 at $J_2 = 3$, there are boundary points where the value of J_1 is less than at other boundary points belonging to this fixed J_2 value. Only the boundary point where we have the maximum J_1 value is the element of the Pareto front and has a corresponding efficient solution in the decision space. This efficient solution is dominating the other "unlucky" solutions. The feasible solutions in the decision space are called the Pareto set and the corresponding dominating points in the criterion space are called the Pareto front.

We can see that the Pareto front in the case of the linear problem was convex, but in this nonlinear problem there is a non-convex Pareto front. See the right hand side of Fig. 8.7.

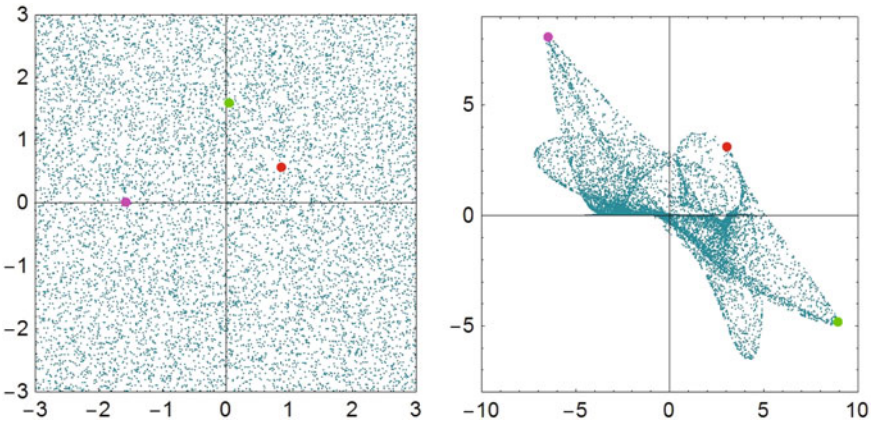


Fig. 8.6 The decision space and the criterion spaces with the three optimum points

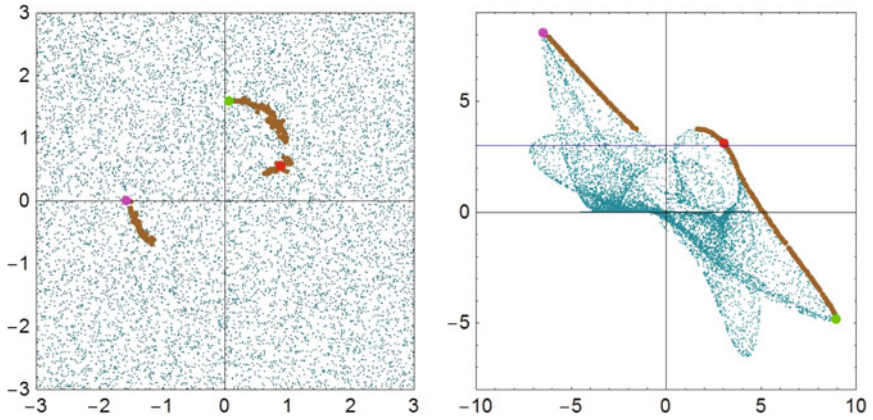


Fig. 8.7 The decision space and the criterion spaces with the three optimum points, as well as the Pareto-set and the Pareto-front

8.3 Computation of Pareto Optimum

8.3.1 Pareto Filter

How can we filter out the Pareto solutions from the feasible solutions? The Pareto filter is a very simple method based on brute force to do that.

Let us consider an optimization problem of an airplane design. We have here four objectives

$$J = \begin{pmatrix} J_1(\text{flying distance}, [\text{km}]) \rightarrow \max \\ J_2(\text{specific cost}, [\$/\text{km}]) \rightarrow \min \\ J_3(\text{number of passangers}, [\text{person}]) \rightarrow \max \\ J_4(\text{flying velocity}, [\text{km/h}]) \rightarrow \max \end{pmatrix}$$

Let us consider two objective vectors, the *i*th and the *j*th. Let $i = 1$ and $j = 2$. Suppose their numerical values are,

$$J_1 = \begin{pmatrix} 7587 \\ 321 \\ 112 \\ 950 \end{pmatrix}, \quad J_2 = \begin{pmatrix} 6695 \\ 211 \\ 345 \\ 820 \end{pmatrix},$$

Let us compare all the pairs,

$$J_1 = \begin{pmatrix} 7587 \\ 321 \\ 112 \\ 950 \end{pmatrix} \begin{pmatrix} > \\ > \\ < \\ > \end{pmatrix} \begin{pmatrix} 6695 \\ 211 \\ 345 \\ 820 \end{pmatrix} = J_2$$

Scoring the relations of the corresponding elements

$$P_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = P_2$$
$$\sum = 2 \text{ vs. } \sum = 2.$$

None of the solution dominates the other one!
Here is another example. Suppose

$$J_1 = \begin{pmatrix} 7587 \\ 321 \\ 112 \\ 950 \end{pmatrix}, \quad J_2 = \begin{pmatrix} 6695 \\ 211 \\ 345 \\ 820 \end{pmatrix}.$$

The objectives are

$$J_1 = \begin{pmatrix} 7587 \\ 321 \\ 112 \\ 950 \end{pmatrix} \begin{pmatrix} > \\ < \\ > \\ > \end{pmatrix} \begin{pmatrix} 6777 \\ 355 \\ 90 \\ 901 \end{pmatrix} = J_2.$$

Scoring

$$P_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = P_2$$
$$\sum = 4 \text{ vs. } \sum = 0.$$

The second solution is dominated by the first one.

If a solution is dominated then the sum of its scores is 0. To represent the result of this analysis, the so called *dominance matrix* can be employed. This matrix has elements 0 or 1.

$$M = \begin{pmatrix} . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & 1 & . & 0 & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \end{pmatrix}$$

if $M_{i,j} = 1$ then the i-th solution dominates the j-th one, if $M_{i,k} = 0$ then the i-th solution does not dominate the j-th one.

The sum of the i-th row shows that how many solutions are dominated by the i-th solution. The sum of the j-th column shows how many solutions dominate the j-th solution. Consequently the solutions having zero sum columns are not dominated by any other solution, therefore they are non-inferior, efficient solutions and they belong to the Pareto front.

In the following sections some techniques are introduced to solve multi-objective problems.

Remark It is efficient to organize the computation for the columns independently. It has two advantages,

- if there is a zero in the column, then there is no need to continue the investigation for the further elements of the column, since the solution belonging to this column is a dominated solution,
- the column-wise computation can be evaluated in parallel.

8.3.2 Reducing the Problem to the Case of a Single Objective

This technique tries to solve the multi-objective problem by reducing the number of the objectives to a single one. Let us consider the following two nonlinear objectives, which both to be minimized, see Fig. 8.8.

```
f1[x_] := (2x + 2)^2 - 40
f2[x_] := (3x - 1)^2 + 5
```

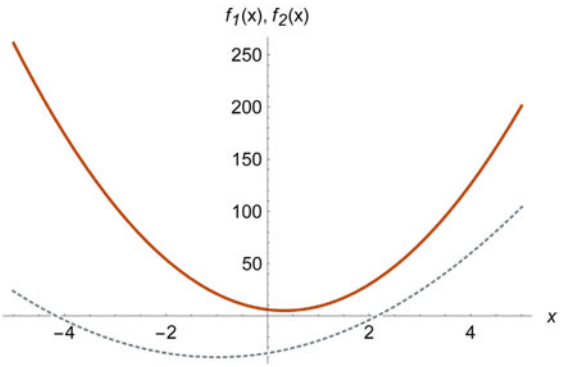


Fig. 8.8 The two objective functions

The minimum of each objective,

```
x1min = NMinimize[f1[x], x]
{-40., {x → -1.}}
x2min = NMinimize[f2[x], x]
{5., {x → 0.333333}}
```

Therefore the parameter space is $x \in \left\{-1, \frac{1}{3}\right\}$.

Let us compute the value of the second objective at the minimum location of the first objective,

```
f2[x]/.x1min[[2]]
21.
```

and vice versa,

```
f1[x]/.x2min[[2]]
-32.8889
```

Table 8.2 shows the results.
Therefore the decision space is $x \in \{-1,\ 1/3\}$ and the criterion space is $\{f_1, f_2\} \in \{-40,\ -32.8889\} \times \{5, 21\}$.

Then the *feasible solutions* can be computed as the solution of the following optimization problem,

Table 8.2 Parameter space and the function space

Optimum	x	f_1	f_2
f_1	−1	−40	21
f_2	1/3	−32.8889	5

$$f_1(x) \to \min_x$$

under the restriction,

$$f_2 = \xi, \quad \text{where} \quad 5 \le \xi \le 21.$$

Let us consider $(f_2)_i$ to be the discrete values of $f_2(x_i)$, where

```
xi = Range[5, 21, 0.5];
```

Then the corresponding minimum locations of $f_1(x)$ are

```
ParetoSet = Map[NMinimize[{f1[x], f2[x] == #}, x][[2]]&, xi]//Flatten;
```

The corresponding function values of $f_1(x)$,

```
ParetoFront = Map[{f1[x], f2[x]}/.#&, ParetoSet];
```

Let us display the Pareto front, see Fig. 8.9.
It can be seen that there are dominated solutions among the feasible solutions. We can filter them out by the Pareto filter. Let us construct the dominance matrix. The size of the matrix is $n \times n$, where

```
n = Length[xi]
33
```

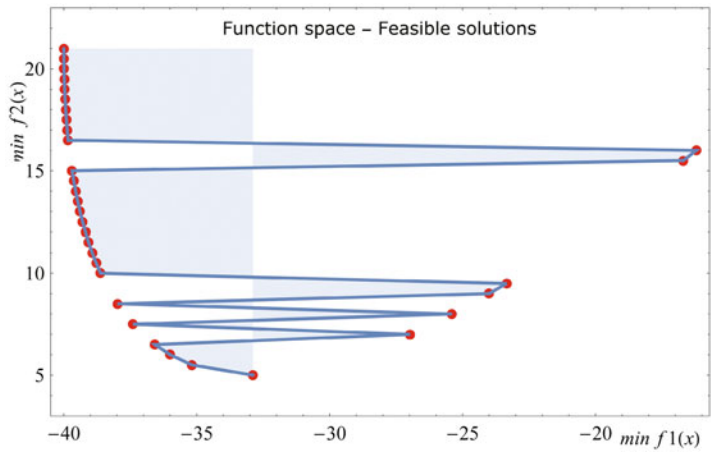


Fig. 8.9 Feasible solutions

The matrix itself,

```
M = Table[If[And[ParetoFront[[i,1]] < ParetoFront[[j,1]],
 ParetoFront[[i,2]] < ParetoFront[[j,2]]],1,0],{i,1,n},{j,1,n}];
MatrixForm[M]
```

$$\begin{pmatrix}
0000101011000000000000110000000000\\
0000101011000000000000110000000000\\
0000101011000000000000110000000000\\
0000101011000000000000110000000000\\
0000001011000000000000110000000000\\
0000001011000000000000110000000000\\
0000000011000000000000110000000000\\
0000000011000000000000110000000000\\
0000000001000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000110000000000\\
0000000000000000000000010000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000\\
0000000000000000000000000000000000
\end{pmatrix}$$

The sum of the different columns,

```
s = Table[Total[Transpose[M][[i]]],{i,1,n}]
{0,0,0,0,4,0,6,0,8,9,0,0,0,0,0,0,0,
 0,0,0,0,21,22,0,0,0,0,0,0,0,0,0,0}
```

The dominated feasible solutions are,

```
Map[Position[s,#]&,Select[s,#!=0&]]//Flatten
{5,7,9,10,22,23}
```

The corresponding points in the function space are

```
NonPareto = Map[ParetoFront[[#]]&,%]
{{-26.9717,7.},{-25.3972,8.},{-24.,9.},
 {-23.3464,9.5},{-16.7009,15.5},{-16.2076,16.}}
```

Getting rid of them, we have the feasible solutions belonging to the Pareto front, see Fig. 8.10.

```
ParetoFront = Complement[ParetoFront,NonPareto];
```

This method can be especially efficient if the objective function has more than one variable, since then one should look for the minimum in a lower dimensional region than the original one.

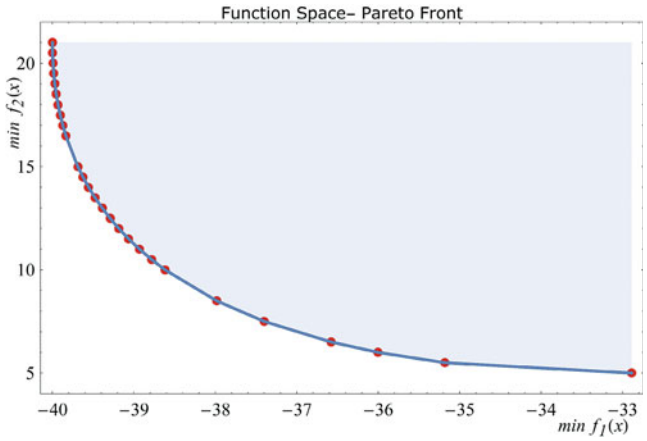


Fig. 8.10 Pareto front

8.3.3 Weighted Objective Functions

Similarly to the method introduced above, we can employ the weighted sum of the different objectives as a single objective,

$$F(x) = \lambda f_1(x) + (1 - \lambda) f_2(x) \rightarrow \min_x$$

where $0 \leq \lambda \leq 1$,

```
F[x_, λ_] := λ f1[x] + (1 - λ) f2[x]
```

Let us consider discrete values for λ

```
λi = Range[0, 1, 0.01];
```

The locations of the minimums,

```
ParetoSet = Flatten[Map[NMinimize[F[x, #], x][[2]][[1, 2]] &, λi];
```

Figure 8.11 shows the Pareto front.

```
ParetoFront = Map[f1[#], f2[#]} &, ParetoSet];
```

Now, the parameter space has one dimension $x \in \{-1,\ 1/3\}$. See Fig. 8.12.
This is an easy method; however it works only if the Pareto front is convex.

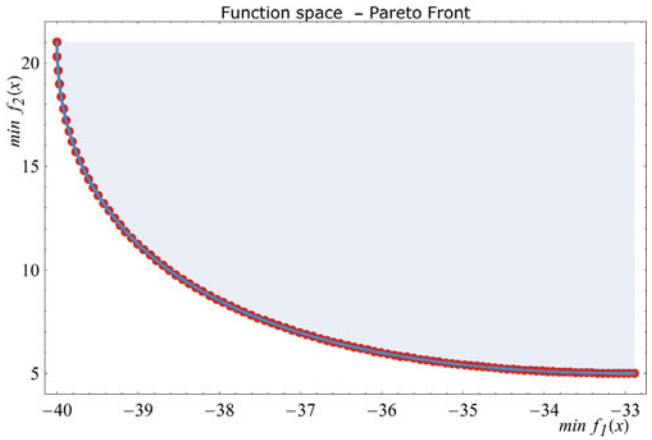


Fig. 8.11 Pareto front

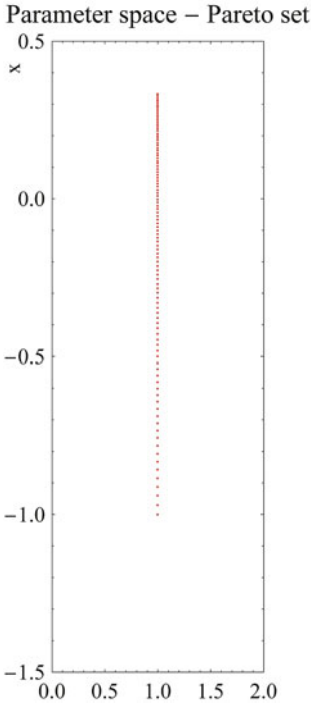


Fig. 8.12 Pareto front

8.3.4 Ideal Point in the Function Space

In Table 8.2 we could see that if the objectives were independent then their minimums would be $f_{1\min} = -40$ and $f_{2\min} = 5$.

This is the "ideal point" of the problem, see Fig. 8.13.

8.3.5 Pareto Balanced Optimum

The *Pareto balanced optimum* is the point of the Pareto front, which is closest to the ideal point according to some kind of norm. Let us find the Pareto balanced optimum considering Euclidean distance. The distance of the points of the Pareto front are

```
ideal = { - 40, 5};
distance = Map[Norm[ideal - #] &, ParetoFront];
```

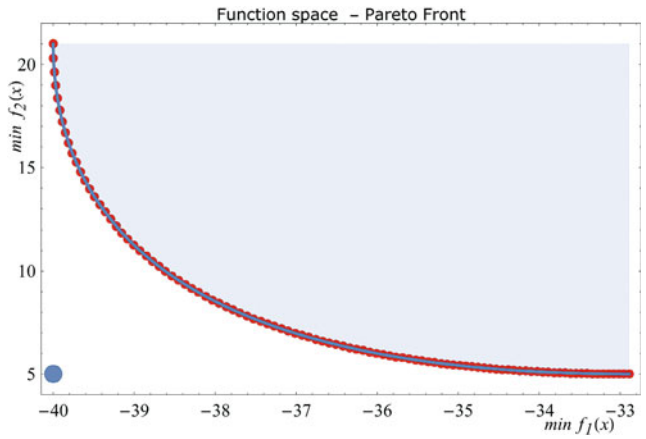


Fig. 8.13 The ideal point in the function space

Now the point which is closest to the ideal point

```
ParetoBalanced=
 ParetoFront[[Position[distance,Min[distance]]//
 Flatten]]//Flatten
{-37.3996,7.5}
```

Then the Pareto balanced point can be seen on Fig. 8.14.
Sometimes the L_1 norm can be also used

```
distance=Map[Norm[ideal-#,1]&,ParetoFront];
ParetoBalanced=
 ParetoFront[[Position[distance,Min[distance]]//
 Flatten]]//Flatten
{-36.5769,6.5}
```

This point is belonging to the parameter $\lambda = 0.5$,

```
ParetoSet=Flatten[Map[NMinimize[F[x,#],x][[2]][[1,2]]&,{0.5}]];
ParetoFront=Map[f1[#],f2[#]}&,ParetoSet]
{{-36.5917,6.51479}}
```

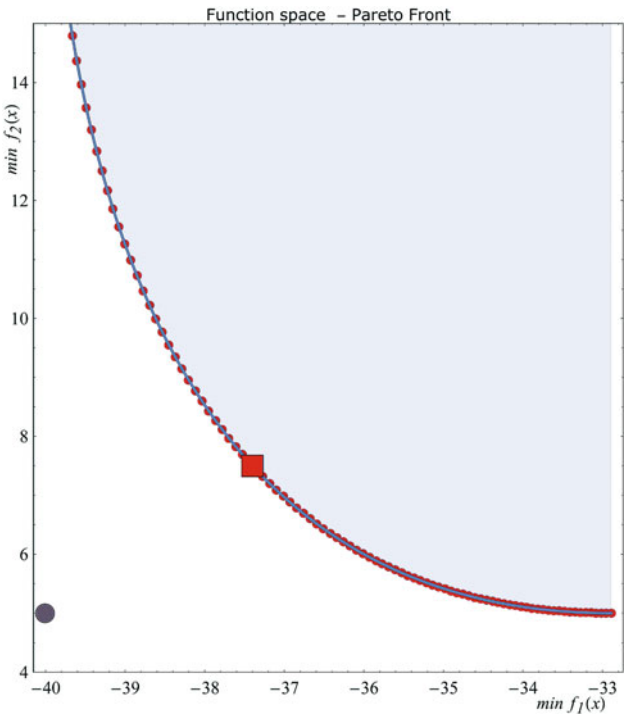


Fig. 8.14 Pareto balanced optimum (square) and ideal point (gray dot)

8.3.6 Non-convex Pareto-Front

We mentioned in Sect. 8.3.3 that the method of the weighted objective functions is simple, but can be employed only in case of convex Pareto front. Let us see what happens if we use this method even when the Pareto front is non-convex and how the problem can be cured.

Let us consider the following functions, see Fig. 8.15

$$f_1(x) = \begin{pmatrix} \text{if } x < 1 \text{ then } & -x \\ \text{if } x \leq 3 \text{ then } & x - 2 \\ \text{if } x \leq 4 \text{ then } & -x \\ \text{else } & x - 4 \end{pmatrix}$$

and

$$f_2(x) = (x - 5)^2.$$

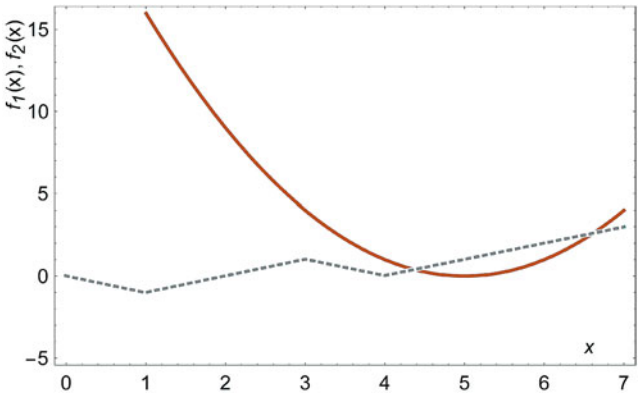


Fig. 8.15 Multi-objective problem leads to non-convex Pareto front

```
f1[x_]:=Which[x≤1,-x,x≤3,x-2,x≤4,4-x,True,x-4]
f2[x_]:=(x-5)^2
```

So the weighted objectives are

```
F[x_,λ_]:=λf1[x] + (1-λ)f2[x]
```

The discrete values of the parameter λ are

```
λi=Range[0,1,0.01];
```

Then the Pareto set

```
ParetoSet=Flatten[Map[NMinimize[F[x,#],x][[2]][[1,2]]&,λi];
```

Let us visualize the corresponding Pareto front, see Fig. 8.16.
The Pareto set consists of the two disjoint sets, see Fig. 8.17.
Now, the solutions represented by the upper branch in the function space are dominated in the interval of [0, 0.8]. So we should eliminate them. Let us employ the Pareto filter to eliminate dominated solutions; see Fig. 8.18.
The corresponding decision space can be seen on Fig. 8.19.

8.4 Employing Genetic Algorithms

In case of a nonlinear, nonconvex problem, a genetic algorithm can be very effective for finding multi-objective optimum, see Gruna (2009).

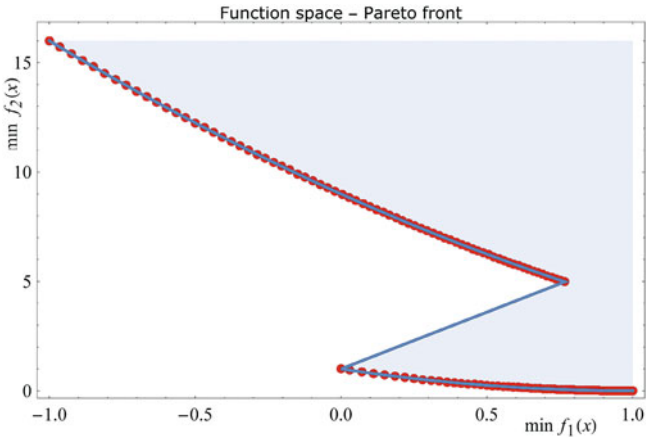


Fig. 8.16 Non-convex Pareto front

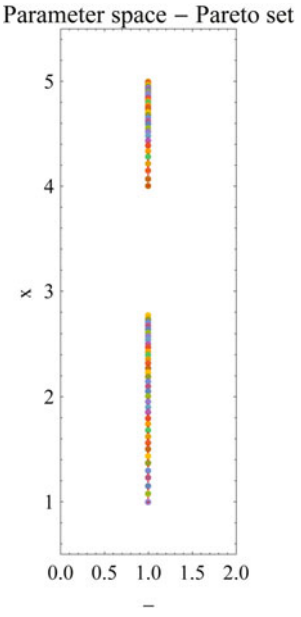


Fig. 8.17 Using only weighted objectives the Pareto-front without filtering is disjunct

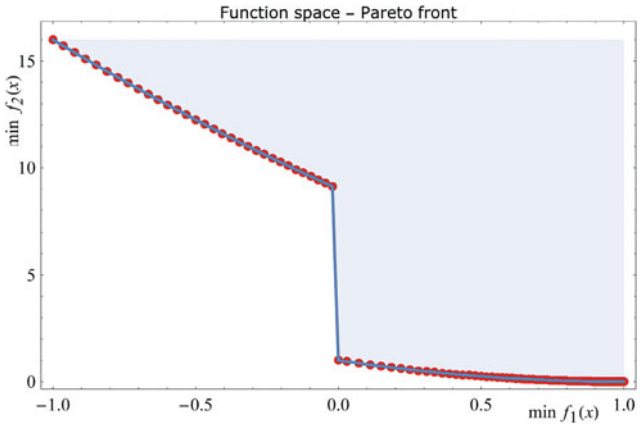


Fig. 8.18 The correct Pareto-front after filtering

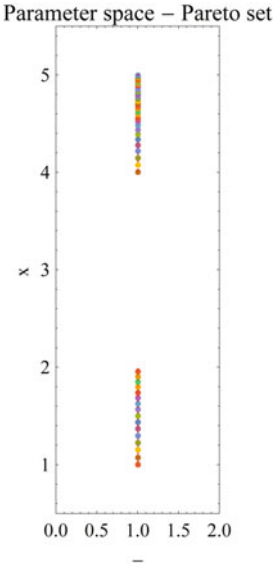


Fig. 8.19 The correct Pareto-set after filtering

```
≪" Pareto'MultiOptimum""
?MultiOptimum
```

```
MultiOptimum[xmin,xmax,F1,F2,isize,psize,mProb,rProb,numgen,
    F1min,F1max,F2min,F2max]
function computes Pareto front (minF2 vs.minF1) and Pareto
    solutions ( x- minimum values).

Input variables:
  xmin - smallest x value  ( min(xi)),
  xmax - biggest x value   ( max(xi)),
  F1, F2 - the competing objectives (Fi[x_]:= fi( x[[1]],
    x[[2]],...,x[[i]],..)),
  isize, psize - individual and population size,
  mProb, rProb - mutation and recombination probability,
  numgen - number of generations,
  F1min, F1max, F2min, F2max - PlotRange → {{F1min,F1max},
    {F2min,F2max}} in Pareto plot

Output variable:
  {out1,out2,out3} - out1: plot of Pareto front and dominating
    solutions if they exist in the last generation,
      out2: coordinates of Pareto front {minF1, minF2},
      out3: Pareto solutions in the variables space, {x[[1,j]],
        x[[2,j]],...,x[[i,j]],..}
```

Let us employ this function for the nonconvex problem discussed in the previous section. The variables of the parameter space should be represented as a vector $x = (x_1)$. Then the two objectives are

```
f1[x_]:=Which[x[[1]]≤1,-x[[1]],x[[1]]≤3,x[[1]]-2,
  x[[1]]≤4,4-x[[1]],True,x[[1]]-4]
```

and

```
f2[x_]:=(x[[1]]-5)^2;
```

Now, let us employ the genetic optimization, (Fig. 8.20)

```
R=MultiOptimum[0,6,f1,f2,100,100,0.02,0.8,10,-1.,1.5,0,16];
R[[1]]
```

As further example, let us consider a two variable problem. We are looking for the minimum of this multi-objective problem

$$f_1(x,y) = x \to \min_{x,y}$$

$$f_2(x,y) = (1+10y)\left(1 - \left(\frac{x}{1+10y}\right)^2 - \frac{x}{1+10y}\sin(8\pi x)\right) \to \min_{x,y}$$

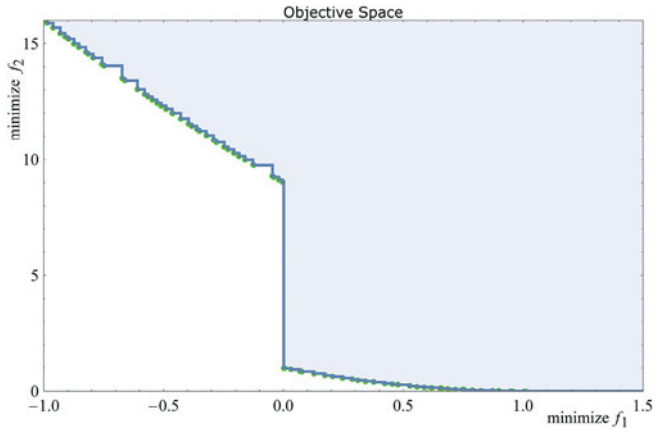


Fig. 8.20 The correct Pareto-front with genetic algorithm

in the $0 \leq x, y \leq 1$ region. The functions with vector variables

```
f1[x_]:=x[[1]]
f2[x_]:=(1 + 10x[[2]])(1 - (x[[1]]/(1 + 10x[[2]]))^2 -
  x[[1]]/(1 + 10x[[2]])Sin[2\[Pi]4x[[1]]])
```

First we compute two generations (two iterations), see Pareto-front in Fig. 8.21,

```
R=MultiOptimum[0,1,
  f1,f2,100,100,0.02,0.8,2,-0.1,1.1,-0.3,11.3];
R[[1]]
```

After 10 generations, we get the Pareto-front in Fig. 8.22.
After 100 generations, the front fully develops as in Fig. 8.23.

```
R=MultiOptimum[0,1,f1,f2,100,100,0.02,0.8,100,-0.1,1.1,-1,2];
R[[1]]
```

The number of points of the Pareto set is

```
R[[3]]//Length
100
```

Figure 8.24 shows the corresponding Pareto set of the problem.

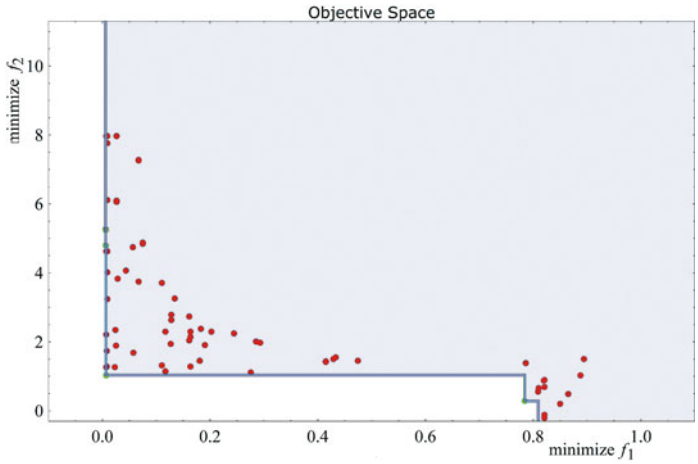


Fig. 8.21 The Pareto-front after two iterations

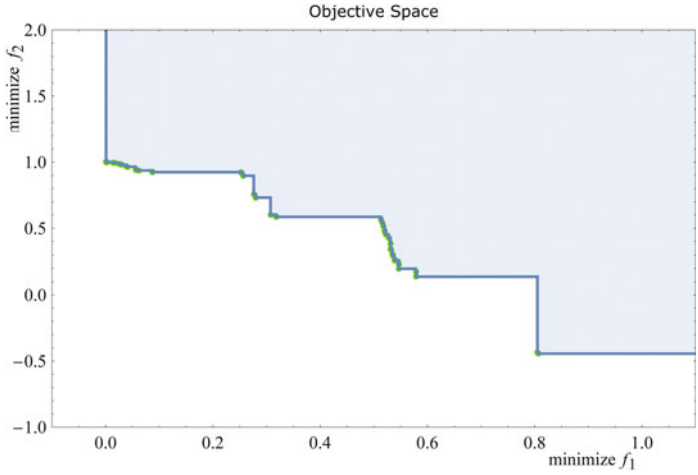


Fig. 8.22 The Pareto-front after 10 iterations

This means that the Pareto set consists of 4 disjunct sets of values of x, where in ideal case $y = 0$ for all solutions. The three points having value of 10^{-6} represent numerical error.

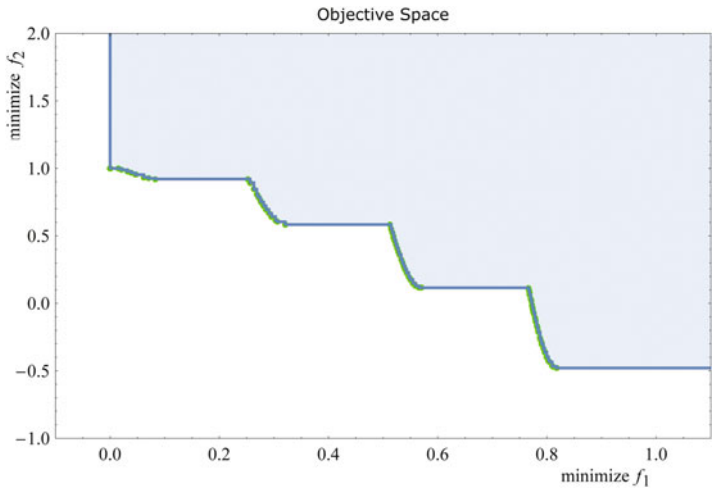


Fig. 8.23 The Pareto-front after 100 iterations

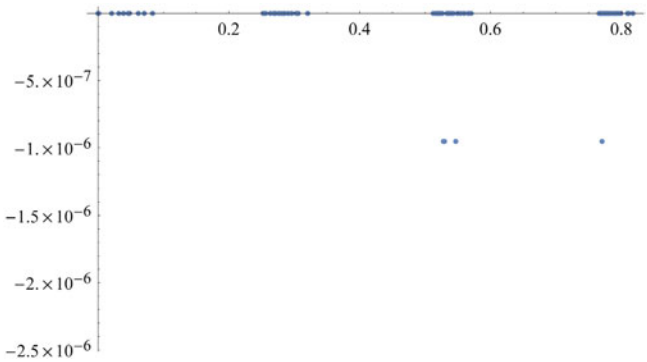


Fig. 8.24 The Pareto set after 100 iterations

8.5 Application

8.5.1 Nonlinear Gauss-Helmert Model

In order to compare the Pareto optimality method with other techniques, let us consider the Gauss-Helmert nonlinear model of weighted 2D similarity transformation,

$$\begin{pmatrix} X \\ Y \end{pmatrix} = F(x, y) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} \beta & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \gamma \\ \delta \end{pmatrix}$$

To determine the 4 unknown parameters α, β, γ, δ from the corresponding measured data pairs $\{(X_i, Y_i), (x_i, y_i)\}$ having the weights $\{(WX_i, WY_i), (wx_i, wy_i)\}$, the following multi-objective problem can be considered in the least squares sense employing the L_2 norm. Let us define the two competing objectives.

(a) The residual of the transformation $(x, y) \rightarrow (X, Y)$

$$f_1(\alpha, \beta, \gamma, \delta) = \sum_{i=1}^{n} WX_i(X_i - \beta(\cos(\alpha)x_i - \sin(\alpha)y_i) + \gamma)^2 + WY_i(Y_i - \beta(\sin(\alpha)x_i + \cos(\alpha)y_i) + \delta)^2$$

Considering the inverse form of the transformation,

$$\begin{pmatrix} x \\ y \end{pmatrix} = F^{-1}(X, Y) = \begin{pmatrix} \frac{1}{\beta} & 0 \\ 0 & \frac{1}{\beta} \end{pmatrix} \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} X - \gamma \\ Y - \delta \end{pmatrix}$$

(b) The residual of the transformation $(X, Y) \rightarrow (x, y)$

$$f_2(\alpha, \beta, \gamma, \delta) = \sum_{i=1}^{n} wx_i \left(x_i - \frac{\cos(\alpha)(X_i - \gamma) + \sin(\alpha)(Y_i - \delta)}{\beta} \right)^2 + wy_i \left(y_i - \frac{\cos(\alpha)(Y_i - \delta) - \sin(\alpha)(X_i - \gamma)}{\beta} \right)^2.$$

The multi-objective problem is to find the minimum of the two objective problems in the variable space—in our case they are α, β, γ and δ—in the sense of Pareto optimality. Then select a single optimum from the set of the Pareto optimums. The input data can be found in Tables 8.3 and 8.4.
The weights are in Table 8.4.
The first step is to compute the coordinates of the ideal point in the function space.

The individual minimum of the objectives were computed via global minimization by *Mathematica* and the maximum value of the objectives can be computed by simply substituting the minimum of the counterpart objective. The result can be found in Table 8.5.

Table 8.3 Corresponding coordinates of the two systems

$X\ [m] \times 10^{-6}$	$Y\ [m]$	$x\ [m] \times 10^{-6}$	$y\ [m]$
4.5401342780	382379.89640	4.5401240940	382385.99800
4.5399373890	382629.78720	4.5399272250	382635.86910
4.5399797390	381951.47850	4.5399695670	381957.57050
4.5403264610	381895.00890	4.5403162940	381901.09320
4.5392163870	382184.43520	4.5392062110	382190.52780

Table 8.4 The weights of the measured coordinates of the two systems

WX	*WY*	*wx*	*wy*
10.0000	14.2857	5.88240	12.5000
0.89290	1.42860	0.90090	1.72410
7.14290	10.0000	7.69230	16.6667
2.22220	3.22590	4.16670	6.66670
7.69230	11.1111	8.33330	16.6667

Table 8.5 The individual minimum and maximum of the objectives

.	f_1	f_2
α [rad]	$-5.8607956729643\times10^{-6}$	$-2.44615778001273\times10^{-6}$
β	0.9999970185952	0.9999937296126
γ [m]	21.4719793439680	37.7064925325464
δ [m]	21.6518647388158	7.4080798108086
f_{min} [m^2]	0.002113068954524	0.00225632756460
f_{max} [m^2]	0.00241943399890	0.00261007624997

To compute the Pareto set, the method of the weighted objective functions is used. Therefore the dimensionless single objective has to be defined for the multi-objective problem,

$$F(\alpha,\beta,\gamma,\delta,\eta) = \eta G_1(\alpha,\beta,\gamma,\delta) + (1-\eta)G_2(\alpha,\beta,\gamma,\delta)$$

where

$$G_i(x) = \frac{f_i(x) - f_{imin}}{f_{imax} - f_{imin}}, \quad 0 \le G_i(x) \le 1, \quad i = 1,2$$

The Perato set is the set of the four-tuples $\{\alpha, \beta, \gamma, \delta\}$, the minimums of $\{G_1, G_2\}$ for $\eta \in [0, 1]$.

The minimization has been carried out for 100 discrete, equidistant $\eta_i \in [0, 1]$ points, using *Mathematica* via parallel computation running on Intel Nehalem processor with 8 threads providing more than 7 times speed-up, see Fig. 8.25.
The transformation parameters $(\alpha, \beta, \gamma, \delta)$ as function of η can be seen on Fig. 8.26.
The Pareto front consists of the minimum values of $\{G_1, G_2\}$ with ideal point $\{0, 0\}$ can be seen in Fig. 8.27.
All of the points of the Pareto—front are optimal. In order to select a single optimum solution, one may choose the very point of the front which is the closest to the ideal point in L_1 norm. This optimum belongs to the parameter value $\eta = 0.5$. Figure 8.28 shows the Pareto balanced solution as well as the Neitzel's TLS (Total Least Square) solution see Neitzel (2010) with the ideal point in the function space.

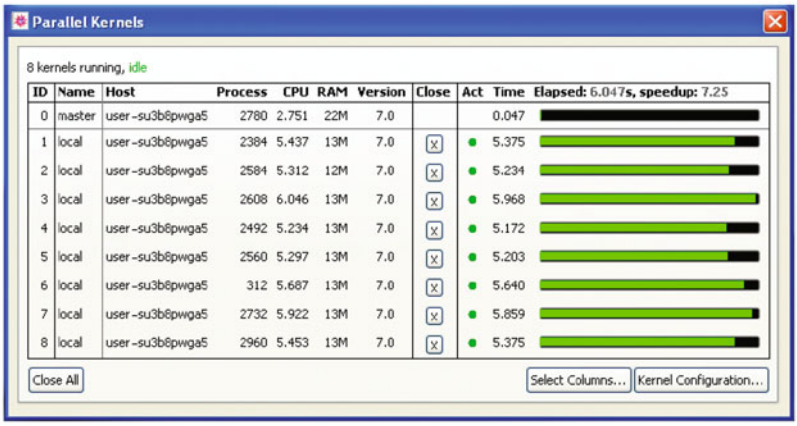


Fig. 8.25 The status of the kernels at the computation of the Pareto front using parallel evaluation, where the service time of the different threads (Time), the total elapsed time of the computation (Elapsed) as well as the speed-up achieved by parallel evaluation (speedup) can be seen

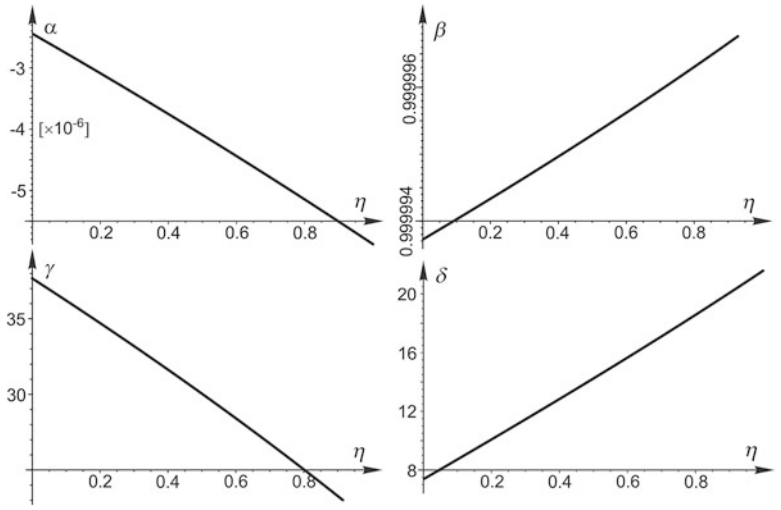


Fig. 8.26 The minimums belonging to the Pareto set as a function the parameter η

It can be seen, that although the TLS solution is close to the Pareto front, it is not a Pareto optimal solution and clearly farther from the ideal point than the Pareto balanced solution.

However, the TLS solution gives smaller error for the first objective (G_1) than the Pareto balanced solution. One can find a region on the Pareto front, where the corresponding Pareto solutions in the Pareto set provide smaller errors for both

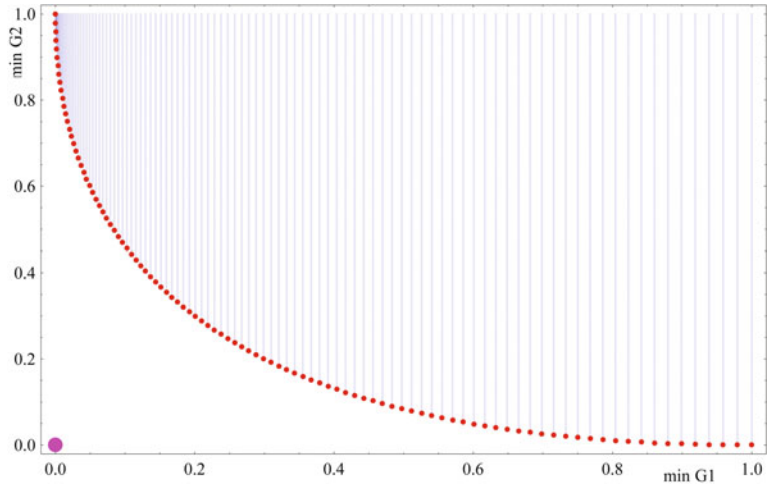


Fig. 8.27 The dimensionless Pareto-front with the ideal point (larger dot)

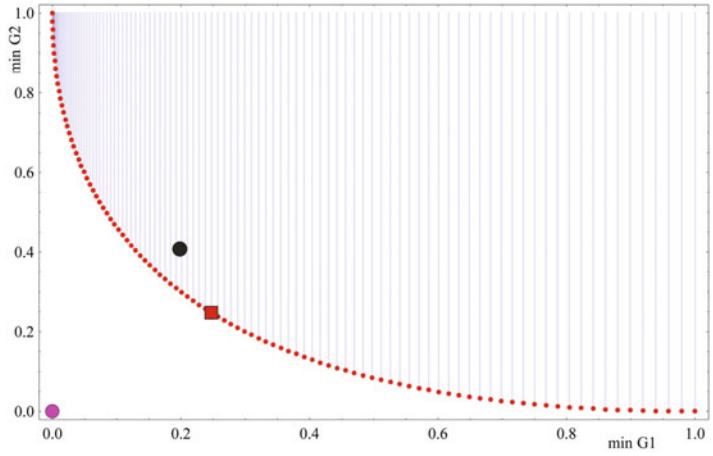


Fig. 8.28 Pareto front with the TLS solution (black dot) and Pareto balanced solution (square)

objectives (G_1 and G_2) than the TLS's one. This region belongs to the parameters $\eta \in (0.552846, 0.644113)$, see Fig. 8.29.

The selected Pareto optimal solution is roughly in the middle of this region, $\eta = 0.6$. The Pareto optimal solutions belong to this parameter region in the Pareto set, all giving smaller values for G_1 and G_2 than the TLS solution.

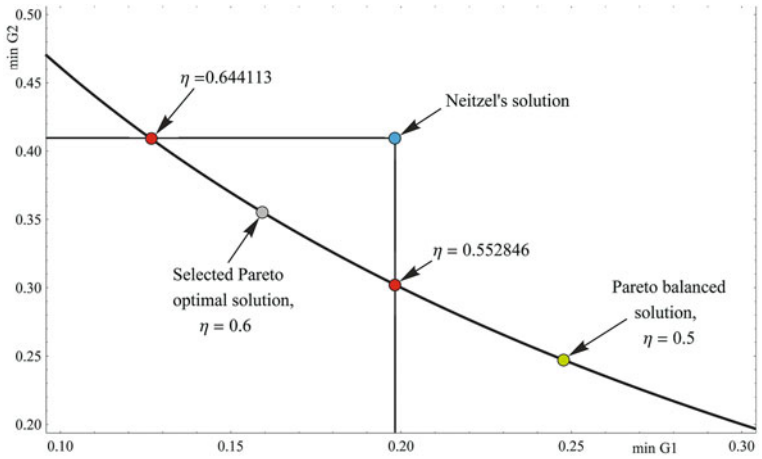


Fig. 8.29 The superiority region of the Pareto optimality $\eta \in (0.552846, 0.644113)$

8.6 Exercise

The relation between the level of rainfall (h) and the water level in the soil (L) was measured at Cincinnati. They were the following values, see Handeberg (2004) (Fig. 8.30)

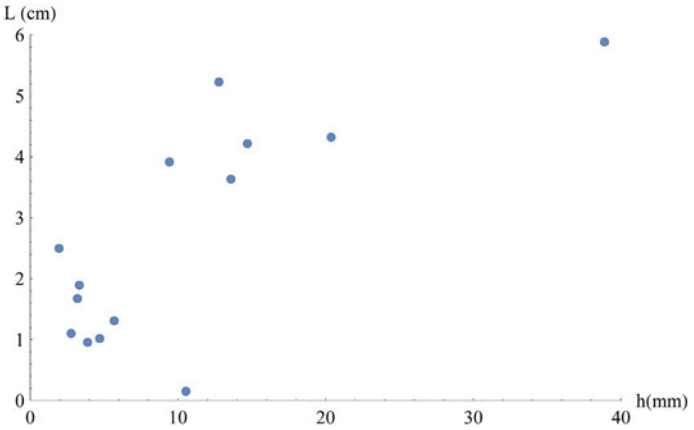


Fig. 8.30 Measured relation between level of rainfall (h) and that of the water in the soil

```
data = {{1.94, 2.5}, {3.33, 1.89}, {3.22, 1.67},
  {5.67, 1.31}, {4.72, 1.02}, {3.89, 0.96}, {2.78, 1.1},
  {10.56, 0.15}, {9.44, 3.92}, {12.78, 5.23}, {14.72, 4.22},
  {13.61, 3.63}, {20.39, 4.32}, {38.89, 5.89}};
```

Let us assume a linear model,

$$L = \beta + \alpha h$$

Assuming that the error of the measurement of the rainfall level can be neglected comparing to that of the water in soil, then employing the standard, ordinary least squares method, the parameters of the model (α, β) can be easily computed.

```
{x, y} = Transpose[data];
n = Length[data];
```

The function to be minimized is

```
f1[α_, β_] := Σ_{i=1}^{n} (y[[i]] - β - αx[[i]])^2
f1min = NMinimize[f1[α, β], {α, β}]
{18.3111, {a → 0.13763, b → 1.26602}}
```

Let us display the line (Fig. 8.31.)
Now, let us suppose the reversed situation, namely the error of the measurement of the water level in soil can be neglected comparing to that of the rainfall. The objective to be minimized is

$$f_2[\alpha\_, \beta\_] := \sum_{i=1}^{n} \left(x[[i]] - \frac{y[[i]] - \beta}{\alpha}\right)^2$$

therefore

```
f2min = NMinimize[f2[α, β], {α, β}]
{549.866, {α → 0.24196, β → 0.178453}}
```

Let us visualize the result, see dashed line (Fig. 8.32).
Now, let us suppose that the errors of the measurements are comparable. This is the so called Error—In—all Variables (EIV) model.

The problem can be solved using total least squares technique (TLS). Then we employ adjustments for both measured variables (Δh_i and ΔL_i). Then the objective function is

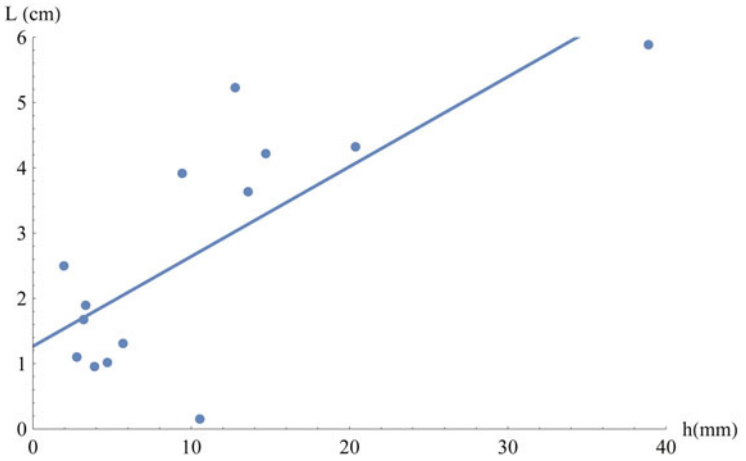


Fig. 8.31 Linear model fitting via least square method

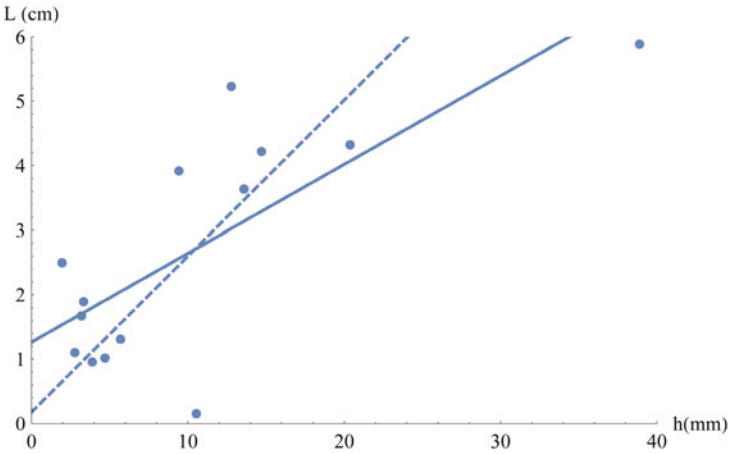


Fig. 8.32 Linear model fitting via least square method in reversed situation

$$f_3 = \sum_{i=1}^{n} (\Delta h_i^2 + \Delta L_i^2);$$

with the restriction

$$g_3 = y[[i]] - \Delta L_i = = \alpha(x[[i]] - \Delta h_i) + \beta,\ i = 1..n$$

The restriction is linear, therefore we can transform the problem into a minimization without restriction, namely

$$f_3 = \sum_{i=1}^{n} (\Delta h_i^2 + (y[[i]] - \alpha(x[[i]] - \Delta h_i) - \beta)^2);$$

The variables are

```
vars = Join[{α, β}, Table[{Δh_i}, {i, 1, n}]] // Flatten
{α, β, Δh_1, Δh_2, Δh_3, Δh_4, Δh_5, Δh_6, Δh_7, Δh_8, Δh_9, Δh_10, Δh_11, Δh_12, Δh_13, Δh_14}
```

Then

```
f_3min = NMinimize[f_3, vars]
{17.9659, {α → 0.139597, β → 1.24552, Δh_1 → -0.134692, Δh_2 → -0.0245957,
  Δh_3 → 0.0034259, Δh_4 → 0.0995512, Δh_5 → 0.121101, Δh_6 → 0.113452,
  Δh_7 → 0.0730646, Δh_8 → 0.351859, Δh_9 → -0.185769, Δh_10 → -0.301302,
  Δh_11 → -0.125922, Δh_12 → -0.0663514, Δh_13 → -0.031234, Δh_14 → 0.107411}}
```

The result is very close to the result of the ordinary least square, see Fig. 8.33 dotted line.

Now we employ Pareto optimization. Our two competing objectives are $f_1(\alpha, \beta)$ and $f_2(\alpha, \beta)$. Since the dimensions of the two objectives are different, we should normalize the functions. Table 8.6 shows the minimum and maximum of the two objectives. The values resulted from the computations of the two ordinary least squares.

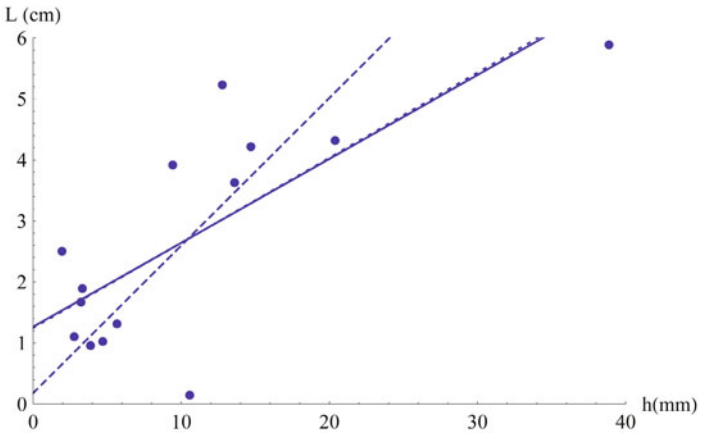


Fig. 8.33 Linear model fitting via Total Least Square (dotted line)

Table 8.6 The minimum and maximum of the two objectives

Optimum	α	β	f_1	f_2
f_1	0.138	1.266	18.31	961.54
f_2	0.242	0.179	32.20	549.87

```
f1[0.242,0.179]
32.2024

f2[0.138,1.266]
961.535
```

Then the normalized objectives are,

$$\texttt{f1}[\alpha\_,\beta\_] := \frac{\texttt{f1}[\alpha,\beta] - 18.31}{32.2 - 18.31}$$

$$\texttt{f2}[\alpha\_,\beta\_] := \frac{\texttt{f2}[\alpha,\beta] - 549.87}{961.54 - 549.87}$$

In vector form

```
F1[X_] := f1[X[[1]],X[[2]]]
F2[X_] := f2[X[[1]],X[[2]]]
```

Let us employ the genetic algorithm to compute the Pareto-front, (Fig. 8.34)

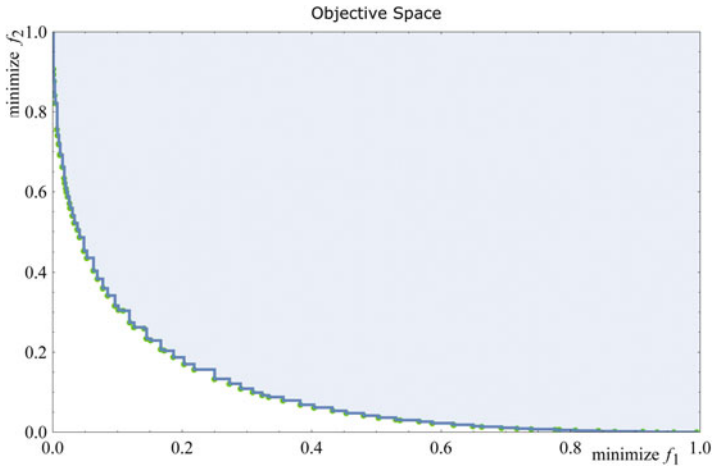


Fig. 8.34 The Pareto front of the linear model fitting problem

```
R = MultiOptimum[0.,2,F1,F2,100,100,0.02,0.8,100,0.,1.,0.,1.];
R[[1]]
```

The ideal point

```
ideal = {0,0};
```

To have a single solution, we compute the Pareto balanced solution (Fig. 8.35),

```
ParetoFront = R[[2]];
ParetoSet = R[[3]];
ParetoFront[[1]];
{0.00111459,100}
```

```
distance = Map[Norm[ideal - #]&,ParetoFront];
ParetoBalanced = ParetoFront
 [[Position[distance,Min[distance]]//Flatten]]//Flatten
{0.186577,0.187133,0.186577,0.187133}
```

```
opt = Union[%]
{0.186577,0.187133}
```

Then the parameters

```
solαβ = FindRoot[{F1[{α,β}] - opt[[1]],
  F2[{α,β}] - opt[[2]]},{{α,1.5},{β,0.75}}]//Flatten
{α → 0.182594,β → 0.8268}
```

The distance of the different solutions from the ideal point:
Ordinary least squares

```
OLSy = Norm[{F1[{0.138,1.266}],F2[{0.138,1.266}]}](*Green*)
0.999987
```

```
OLSx = Norm[{F1[{0.242,0.179}],F2[{0.242,0.179}]}](*Blue*)
1.00017
```

Total least square

```
TLS = Norm[{F1[{0.140,1.246}],F2[{0.140,1.246}]}](*Magenta*)
0.934608
```

Pareto balanced solution

```
PBO = Norm[{F1[{0.181, 0.850}], F2[{0.181, 0.850}]}](*Red*)
```

0.266064

Although their distances are different, the results of all methods are on the Pareto front! (Fig. 8.36).

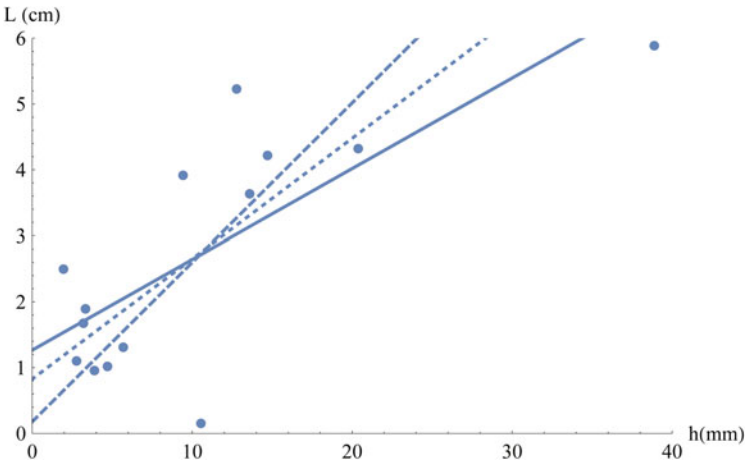


Fig. 8.35 The Pareto balanced solution (dotted line)

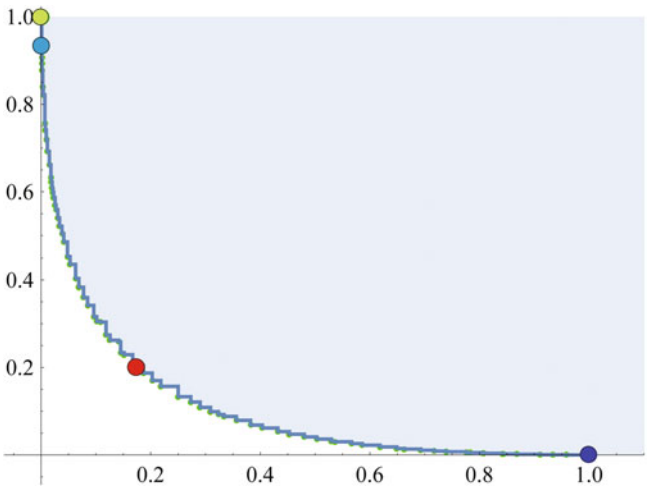


Fig. 8.36 All solutions are on the Pareto front

References

Gruna R (2009) Evolutionary multiobjective optimization. Wolfram Demonstrations Project, http://demonstrations.wolfram.com/EvolutionaryMultiobjectiveOptimization/

Haneberg WC (2004) Computational geosciences with mathematica. Springer, Berlin, Heidelberg

Neitzel F (2010) Generalization of total least-squares on example of unweighted and weighted 2D similarity transformation. J Geodesy 84(12):751–762

Part III
Approximation of Functions and Data

Chapter 9
Approximation with Radial Bases Functions

9.1 Basic Idea of RBF Interpolation

The radial basis method (RBF) is one of the kernel methods, which can be employed for interpolation as well as approximation of data or functions given by values. This is especially true when we do not have a regular grid, but scattered data in many dimensions. The RBF method employs a linear combination of so called *radial basis functions* to carry out a basically local approximation. This method is also applied in machine learning, for example as an activation function of artificial neural networks (ANN).

The wide spread successful application of RBF is based on the theoretical fact that the radial basis functions, like algebraic polynomials and sigmoid activation functions of ANN, are so called *universal approximators*, see Micchell et al. (2006).

For interpolation, the radial bases functions (RBF) method employs a linear combination of basis functions $\varphi(x)$,

$$f(x) = \sum_{i=1}^{n} c_i \varphi(\|x - x_i\|),$$

where the coefficients c_i can be computed from the known data pairs $f(x_i) = f_i$ via the solution of the following linear system,

$$Mc = \begin{pmatrix} \varphi(\|x_1 - x_1\|) & \varphi(\|x_1 - x_2\|) & \dots & \varphi(\|x_1 - x_n\|) \\ \varphi(\|x_2 - x_1\|) & \varphi(\|x_2 - x_2\|) & \dots & \varphi(\|x_2 - x_n\|) \\ \dots & \dots & \dots & \dots \\ \varphi(\|x_n - x_1\|) & \varphi(\|x_n - x_2\|) & \dots & \varphi(\|x_n - x_n\|) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_1 \\ \dots \\ f_n \end{pmatrix} = f.$$

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_9

Let us consider a 1D problem. Suppose the function is,

$$f(x) = x \sin(x).$$

We generate $n = 10$ points in the interval $x_i \in [1/2, 3]$, $i = 1, 2, \ldots, n$, see Fig. 9.1.

```
data = Table[{i 0.3, i 0.3 Sin[i 0.3]}, {i, 1, 10}]
{{0.3, 0.0886561}, {0.6, 0.338785}, {0.9, 0.704994},
 {1.2, 1.11845}, {1.5, 1.49624}, {1.8, 1.75293},
 {2.1, 1.81274}, {2.4, 1.62111}, {2.7, 1.15393}, {3., 0.42336}}
```

To interpolate the function we employ a thin-plate spline as a basis function, which is a type of the radial basis function family,

$$\varphi(r) = r^2 \log(r),$$

where

$$r = \|x - x_i\|.$$

It means this r, is always $r \geq 0$.

Remark Instead of $r^2 \log(r)$ you use $r^2 \log(\varepsilon + r)$ for suitable ε (1/100, say) then you avoid the singularity at zero and still get reasonable behavior. It makes for a cleaner formula, for instance, and is perhaps a bit faster for large computations.

The symbol r is used often in the following examples. Figure 9.2 shows two basis functions with different locations.

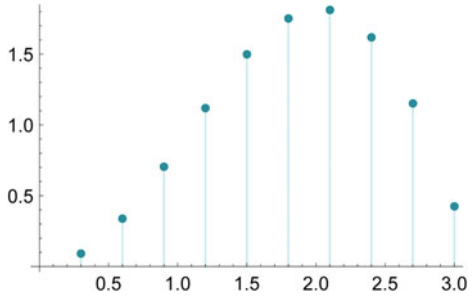


Fig. 9.1 The generated discrete points

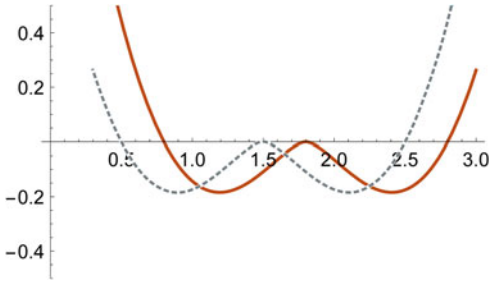


Fig. 9.2 Two basis functions

Separating the x_i and y_i coordinates into two arrays,

```
np = Length[data] ; datax = Transpose[data][[1]];
datay = Transpose[data][[2]]
```

the matrix of the linear system is

```
M = Table[First[TPSpline[datax[[i]],
  datax[[j]]]],{i,1,np},{j,1,np}];
```

This matrix has a relatively high condition number,

```
LinearAlgebra`MatrixConditionNumber[M]
505.204
```

Therefore, to compute the RBF coefficients, we use the pseudoinverse,

```
c = PseudoInverse[M].datay
```

The coefficients can be seen in Fig. 9.3.The interpolation function will be computed as

```
f = First[c.Map[TPSpline[x,#]&,datax]]
1.94971 If[x ≠ 0.3,Abs[x - 0.3]^2 Log[Abs[x - 0.3]],0] -
  2.26345 If[x ≠ 0.6,Abs[x - 0.6]^2 Log[Abs[x - 0.6]],0] -
  0.412961 If[x ≠ 0.9,Abs[x - 0.9]^2 Log[Abs[x - 0.9]],0] -
  0.419176 If[x ≠ 1.2,Abs[x - 1.2]^2 Log[Abs[x - 1.2]],0] -
  0.282941 If[x ≠ 1.5,Abs[x - 1.5]^2 Log[Abs[x - 1.5]],0] -
  0.22277 If[x ≠ 1.8,Abs[x - 1.8]^2 Log[Abs[x - 1.8]],0] -
  0.224389 If[x ≠ 2.1,Abs[x - 2.1]^2 Log[Abs[x - 2.1]],0] -
  0.178729 If[x ≠ 2.4,Abs[x - 2.4]^2 Log[Abs[x - 2.4]],0] -
  1.23521 If[x ≠ 2.7,Abs[x - 2.7]^2 Log[Abs[x - 2.7]],0] +
  1.00129 If[x ≠ 3.,Abs[x - 3.]^2 Log[Abs[x - 3.]],0]
```

Figure 9.4 shows the relative error of the RBF interpolation function.

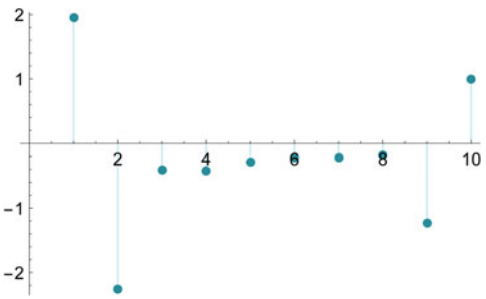


Fig. 9.3 The c_i coefficients

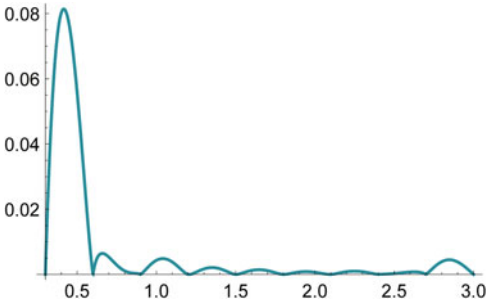


Fig. 9.4 The approximation of the interpolation function

In geosciences, frequently we have so called "scattered data interpolation problems", where the measurements do not lie on a uniform or regular grid. Let us consider a 2D problem, where the interpolation points are the discrete values of the following function (Fig. 9.5).

$$f(x, y) = 16x(1 - x)y(1 - y)$$

The interpolation should be carried out on the region $(x, y) \in [0, 1]^2$. We employ $n = 124$ Halton points, see Mangano (2010), see Fig. 9.6,
Now the following basis function is used,

$$\varphi(r) = r^{2\beta} \log r,$$

or

$$\varphi(x, x_i) = \|x - x_i\|^{2\beta} \log\|x - x_i\|.$$

Here we employ $\beta = 1$ and $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$. Figure 9.7 shows a basis function sitting on the origin,

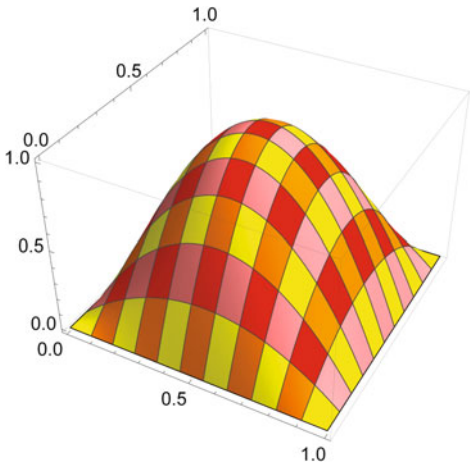


Fig. 9.5 The function to be approximated

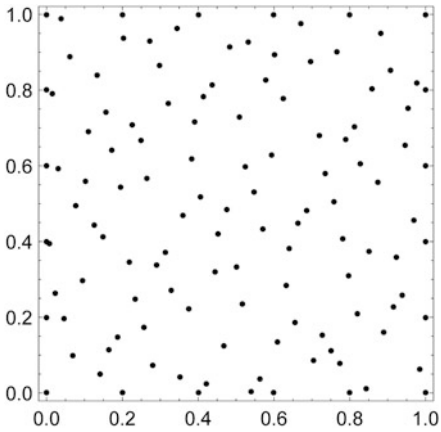


Fig. 9.6 The Halton-points as interpolation points

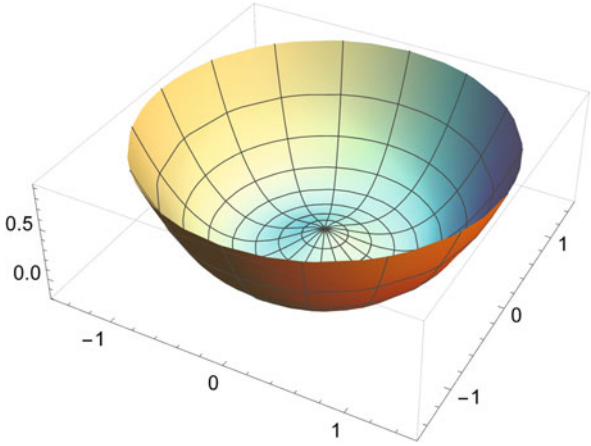


Fig. 9.7 The basis function of the 2D problem

The condition number of the coefficient matrix of 124×124 is very high, therefore our interpolation problem is ill-posed and the pseudoinverse should be used. Figure 9.8 shows the approximation relative errors. As usual, the error is high on the boundary region since the points in this region have neighbors on only one side.

9.2 Positive Definite RBF Function

The weakness of the RBF method is the ill-conditioned coefficient matrix, which leads to an ill-posed interpolation problem. To avoid this problem, we introduce the definition of a *positive definite* function. A real valued continuous function $\varphi \in R^S \rightarrow$ C is called positive definite if

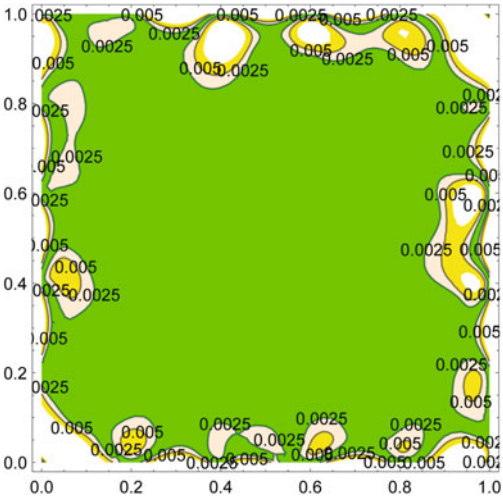


Fig. 9.8 The error of the RBF interpolation in case 2D scattered data

$$\sum_{j=1}^{N}\sum_{k=1}^{N} c_j c_k \varphi\left(x_j - x_k\right) \geq 0,$$

for any pairwise different points $x_1, \ldots, x_N \in R^s$ and $c = (c_1, \ldots, c_N)^T \in C^N$. The function φ is called *strictly positive definite* if this quadratic form is zero only for $c \equiv 0$.

In this case, the coefficient matrix will be positive definite, non singular. According to *Bochner's Theorem Fasshauer* (2007) the RBF is positive definite if its Fourier transform is positive. For example, the Gaussian type RBF

$$\varphi(r) = \mathrm{e}^{-(\epsilon r)^2}$$

for $\epsilon > 0$ is a positive definite function (Fig. 9.9), since

```
φ = e^-(ϵr)^2
e^-r^2ϵ^2
F = FourierTransform[φ, r, ω]
e^-ω^2/(4ϵ^2)/(√2 √ϵ^2)
```

In the special case of $\varepsilon = 1/\sqrt{2}$, the basis function and its Fourier transform are the same

$$F/.\varepsilon \to \frac{1}{\sqrt{2}}$$
$$\mathrm{e}^{-\frac{\omega^2}{2}}$$
$$\varphi/.\varepsilon \to \frac{1}{\sqrt{2}}$$
$$\mathrm{e}^{-\frac{t^2}{2}}$$

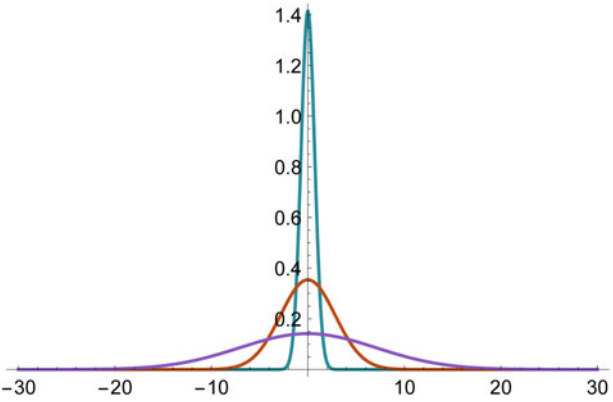


Fig. 9.9 The Fourier transform of a Gaussian RBF in case of different ε values (0.5, 2 and 5)

9.3 Compactly Supported Functions

As we mentioned above, an unwelcome situation appears when the linear systems are solved for large N, since for most radial basis functions the matrix is full and ill-conditioned. An exception is provided by the radial basis functions of compact support.

Compactly supported radial basis functions have been invented for the purpose of getting finite-element type approximations. They give rise to sparse interpolation matrices and can be used to numerically solve partial differential equations, called mesh-less methods. Under suitable conditions on degree and dimension N, they give rise to positive definite interpolation matrices A that are banded, therefore sparse, and then of course also regular.

A compact (closed and bounded) subset $S \subset R$ is a compact support of a function $\varphi(x)$, if $\varphi(x) \neq 0$ when $x \in S$ but it is zero outside of this compact set. If this set S is not bounded then the support is called a global support. Let us consider the following function, see Fig. 9.10.

```
φ = Piecewise[{{1 - x^2, -1 ≤ x < 1}}]
{ 1 - x^2   -1 ≤ x < 1
  0         True
```

The interval $[-1, 1]$ is obviously the compact support of this function.

For the function

$$\phi = \frac{x}{1+x^2}$$

defined on the domain $x \geq 0$, the interval $x \geq 0$ is the global support (Fig. 9.11) by elementary algebra.

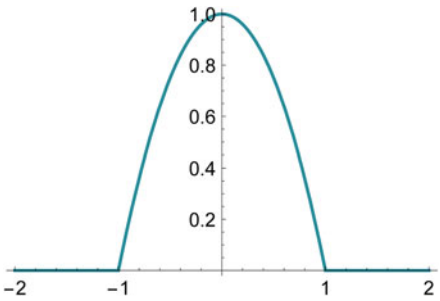


Fig. 9.10 A function with interval of [−1, 1] as compact support

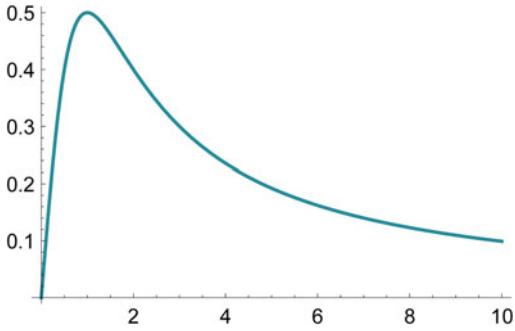


Fig. 9.11 Function with global support

(The interval [0, ∞] is not bounded, therefore it is a global support.)

For another example, the quadratic *Matérn* RBF function Fasshauer (2007), see Fig. 9.12,

```
φ = e^-r (3 + 3r + r^2)
e^-r (3 + 3r + r^2)
```

whose domain is $r \geq 0$, has global support ($r \geq 0$). This follows from elementary properties of the exponential function.

The *Wendland* $\varphi_{3,2}$ RBF basis function Wendland (2005) has a compact support ([0, 1]) (Fig. 9.13).

```
φ = (1 - r)^4 (1 + 4r);
```

Since, it is well known that the domain of this function is [0, 1].

The basis functions with compact support are positive definite functions and their corresponding coefficient matrices are sparse matrices, therefore the corresponding equation system can be solved iteratively.

Remark If we decrease the distance between the basis points (separation distance), the approximation will be better (smaller error), but the condition number of the coefficient matrix will increase, trade-off problem causing high round-off errors.

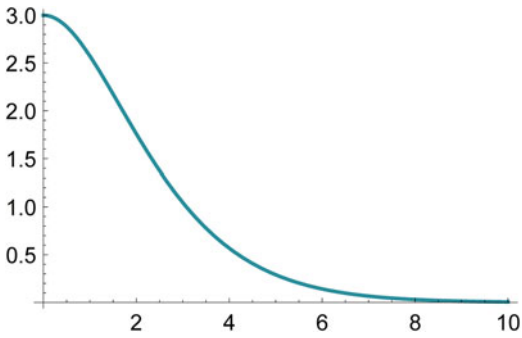


Fig. 9.12 *Matérn* RBF with global support

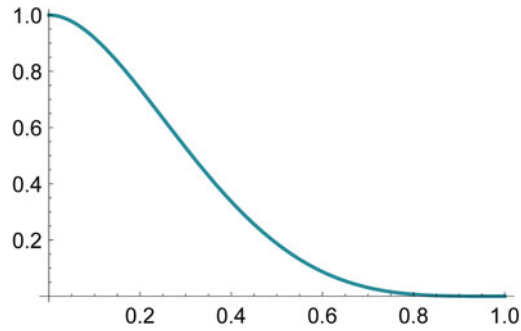


Fig. 9.13 *Wendland* $\varphi_{3,2}$ RBF with global support

9.4 Some Positive Definite RBF Function

In this section we list some well-known positive definite RBF functions. The domain is always restricted to $r \geq 0$.

9.4.1 Laguerre-Gauss Function

This is a generalization of the Gauss function. For example (Fig. 9.14),

$$\varphi(r) = \mathrm{e}^{-r^2}\left(3 - 3r^2 + \frac{1}{2}r^4\right)$$

```
LG = Function[{x,xi},(3 - 3Norm[x - xi,2]^2 + (1/2)Norm[x - xi,2]^4)
  exp[ - Norm[x - xi,2]^2]];
```

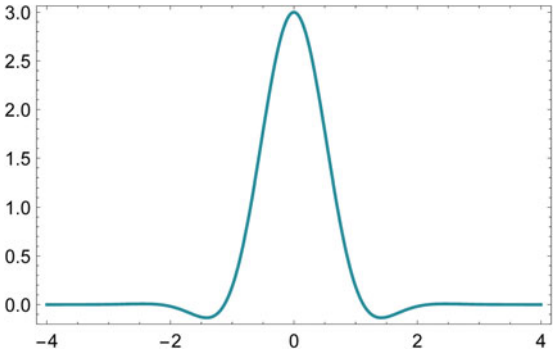


Fig. 9.14 Laguerre-Gauss function with compact support

In two dimensions,

```
z[r_, φ_] := LG[{r Cos[φ], r Sin[φ]}, {0, 0}]
```

Let us visualize the function, see Fig. 9.15.

```
ParametricPlot3D[{r Cos[φ], r Sin[φ], z[r, φ]}, {r, -4, 4},
 {φ, 0., 2 π}, BoxRatios → {1, 1, 0.5}]
```

9.4.2 Generalized Multi-quadratic RBF

The RBF basis function,

$$\varphi(r) = \left(1 + r^2\right)^{\beta}$$

for example with $\beta = -1/2$, we call inverse multi-quadratic RBF, see Fig. 9.16.

```
IMQ = Function[{x, xi}, (1 + Norm[x - xi, 2]^2)^(-1/2)];
```

In two dimensions, see Fig. 9.17.

```
z[r_, φ_] := IMQ[{r Cos[φ], r Sin[φ]}, {0, 0}]
```

```
ParametricPlot3D[{r Cos[φ], r Sin[φ], z[r, φ]}, {r, -4, 4},
 {φ, 0., 2π}, BoxRatios → {1, 1, 0.5}]
```

In case of $\beta = ½$, we get the *Hardy-type multi-quadratic* RBF, see Fig. 9.18.

$$\varphi(r) = \left(1 + r^2\right)^{\beta}$$

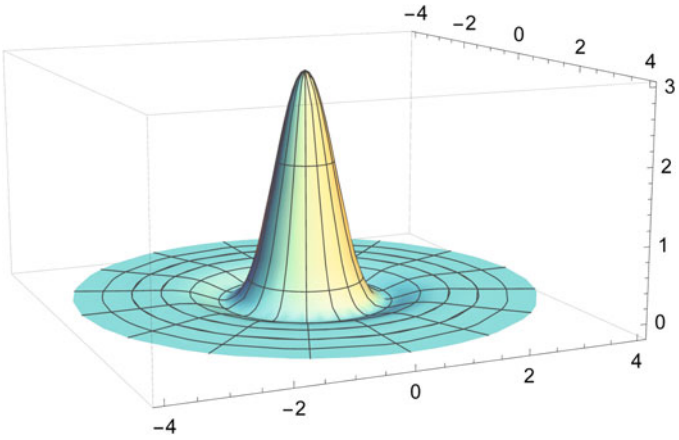


Fig. 9.15 Laguerre-Gauss function with compact support in two dimensions

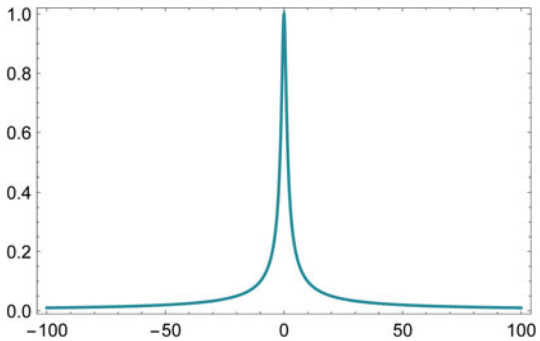


Fig. 9.16 Generalized multi-quadratic RBF

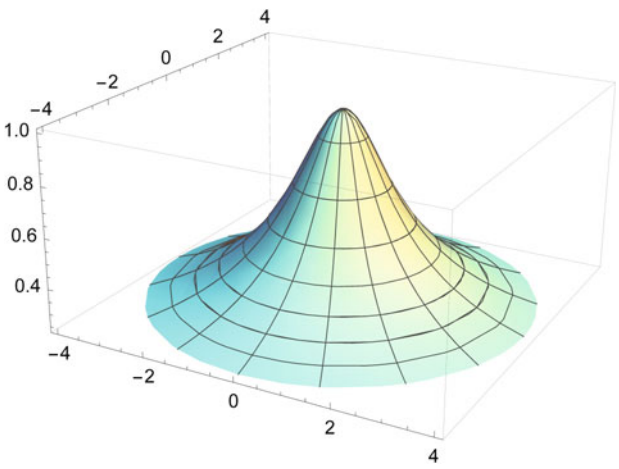


Fig. 9.17 Generalized multi-quadratic RBF in two dimensions

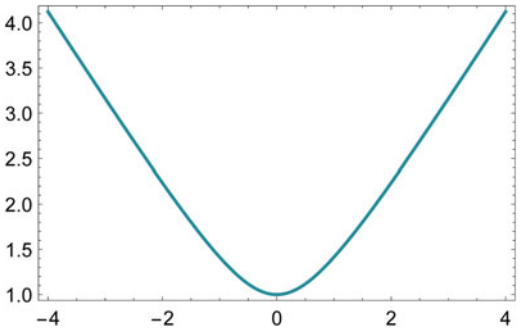


Fig. 9.18 Hardy-type multi-quadratic RBF

```
HIMQ = Function[{x, xi}, (1 + Norm[x - xi, 2]^2)^(1/2)];
```

In two dimensions (Fig. 9.19),

```
z[r_, φ_] := HIMQ[{r Cos[φ], r Sin[φ]}, {0, 0}]

ParametricPlot3D[{r Cos[φ], r Sin[φ], z[r, φ]}, {r, -4, 4},
 {φ, 0., 2π}, BoxRatios → {1, 1, 0.5}]
```

9.4.3 Wendland Function

The Wendland function is a polynomial-type RBF, see Fig. 9.20.

$$\varphi(r) = (1-r)^6\left(35r^2 + 18r + 3\right)$$

```
WLD = Function[x, xi, (1 - Norm[x - xi, 2])^6
 (35Norm[x - xi, 2]^2 + 18Norm[x - xi, 2] + 3)];
```

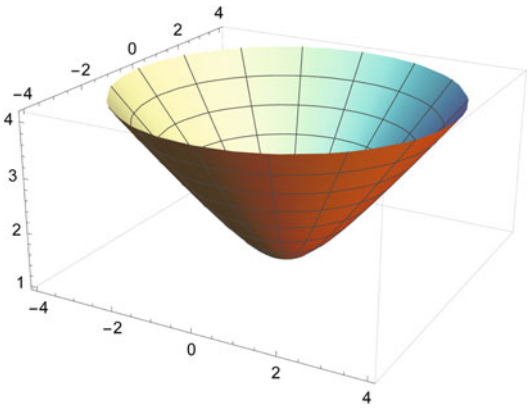


Fig. 9.19 Hardy-type multi-quadratic RBF in two dimension

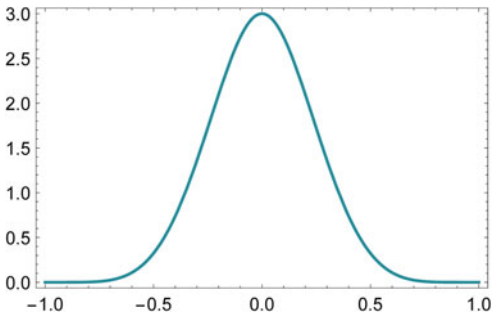


Fig. 9.20 Wendland function

In two dimensions, see Fig. 9.21.

```
z[r_, φ_] := VLD[{r Cos[φ], r Sin[φ]}, {0, 0}]

ParametricPlot3D[{r Cos[φ], r Sin[φ], z[r, φ]}, {r, -1.0, 1.0},
 {φ, 0., 2π}, BoxRatios → {1, 1, 0.5}]
```

9.4.4 Buchmann-Type RBF

This basis function is a mixed type, see Fig. 9.22.

$$\varphi(r) = 2r^4 \log(r) - \frac{7}{2}r^4 + \frac{16}{3}r^3 - 2r^2 + \frac{1}{6}, \quad 0 \le r \le 1$$

```
BH = Function[x, xi, If[x! = xi, 2Norm[x - xi, 2]^4 Log[Norm[x - xi, 2]] -
 7/2 Norm[x - xi, 2]^4 + 16/3 Norm[x - xi, 2]^3 -
 2 Norm[x - xi, 2]^2 + 1/6, 0]];
```

In two dimensions, see Fig. 9.23.

```
z[r_, φ_] := BH[{r Cos[φ], r Sin[φ]}, {0, 0}]

ParametricPlot3D[{r Cos[φ], r Sin[φ], z[r, φ]}, {r, -1.0, 1.0},
 {φ, 0., 2π}, BoxRatios → {1, 1, 0.5}]
```

9.5 Generic Derivatives of RBF Functions

This topic is important for solving partial differential equations.

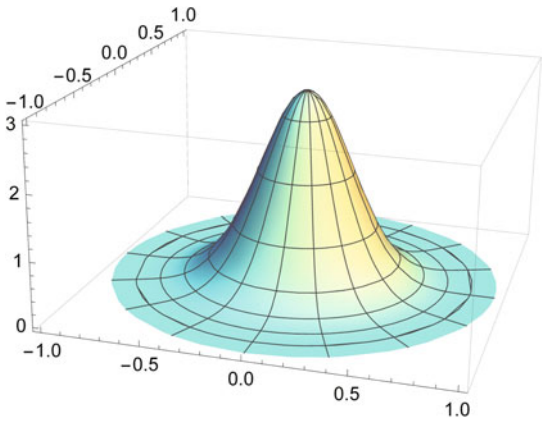


Fig. 9.21 Wendland function in two dimensions

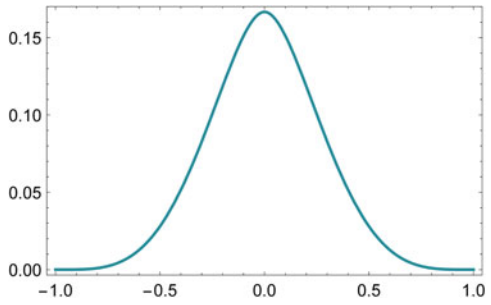


Fig. 9.22 Buchmann function

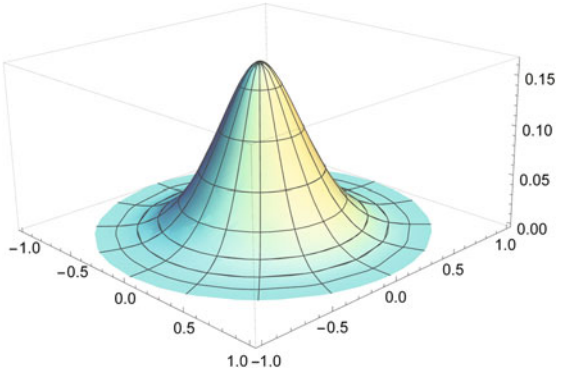


Fig. 9.23 Buchmann function in two dimensions

The formulas for all first and second-order derivatives of RBF of two variables are the followings:

If the basis function is given by

$$\varphi(r) = \varphi(\|u\|) = \varphi\left(\sqrt{x^2+y^2}\right),$$

then the first order partial derivatives is

$$\frac{\partial\varphi}{\partial x} = \frac{d\varphi}{dr}\frac{\partial r}{\partial x} = \frac{d\varphi}{dr}\frac{x}{\sqrt{x^2+y^2}} = \frac{x}{r}\frac{d\varphi}{dr},$$

similarly

$$\frac{\partial\varphi}{\partial y} = \frac{y}{r}\frac{d\varphi}{dr}.$$

The second order derivatives,

$$\frac{\partial^2\varphi}{\partial x^2} = \frac{x^2}{r^2}\frac{d^2\varphi}{dr^2} + \frac{y^2}{r^3}\frac{d\varphi}{dr},$$

similarly

$$\frac{\partial^2\varphi}{\partial y^2} = \frac{y^2}{r^2}\frac{d^2\varphi}{dr^2} + \frac{x^2}{r^3}\frac{d\varphi}{dr},$$

as well as

$$\frac{\partial^2\varphi}{\partial x\partial y} = \frac{xy}{r^2}\frac{d^2\varphi}{dr^2} - \frac{xy}{r^3}\frac{d\varphi}{dr}.$$

The Laplacian is

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\varphi(r) = \frac{d^2\varphi}{dr^2} + \frac{1}{r}\frac{d\varphi}{dr}.$$

The generic fourth-order biharmonic (or double Laplacian) is,

$$\left(\frac{\partial^4}{\partial x^4} + 2\frac{\partial^4}{\partial x^2\partial y^2} + \frac{\partial^4}{\partial y^4}\right)\varphi(r) = \frac{d^4\varphi}{dr^4} + \frac{2}{r}\frac{d^3\varphi}{dr^3} - \frac{1}{r^2}\frac{d^2\varphi}{dr^2} + \frac{1}{r^3}\frac{d\varphi}{dr}.$$

9.6 Least Squares Approximation with RBF

Up to now we have looked only at interpolation. However, many times it makes more sense to approximate the given data by a least squares fit. This is especially true if the data are contaminated with noise, or if there are so many data points that efficiency considerations force us to approximate from a space spanned by fewer basis functions than data points. We will use n_R as the number of the basis functions. In case of interpolation, this equals the number of data points. Here the user chooses n_R in the beginning. It should be less than the number of data points (otherwise it is the same as interpolation).

If $x \in R^m$, so x is a vector of m dimensions, then the number of the unknowns is $n_R + m\,n_R = (1+m)n_R$.

We speak about regression if the system is overdetermined $n > (1+m)n_R$, where n is the number of the data.

Let us consider n "noisy" data points, but n_R RBF basis functions ($n_R < n$), then we have an overdetermined system for the coefficients c_i, $i = 1, 2, \ldots, n_R$

$$f_j = \sum_{i=1}^{n_R} c_i \varphi(\|x - x_i\|), \quad j = 1, 2, \ldots, n > n_R,$$

For example, let us consider the following function, see Fig. 9.24.

$$f(\mathrm{x},\,\mathrm{y}) = 0.3\left(1.35 + e^{x}\sin\left(13(x-0.6)^2\right)e^{-y}\sin(7y)\right),$$

To approximate this function on the interval of [0, 1] × [0, 1] with $n = 100$ "noisy" points

```
f[{x_,y_}]:=0.3(1.35 + e^x sin[13(x-0.6)^2]e^-y sin[7y]);
```

We add a white noise $N(0, \sigma)$, with $\sigma = 0.05$ to the functions values,

```
fn[{x_,y_}]:=f[{x,y}] + Random[NormalDistribution[0.,0.05]]
```

Then the data triples $\{\{x_i, y_i, f(x_i, y_i)\}$ are (Fig. 9.25).

```
dataxyfn=Flatten[Table[N[{i,j,
 fn[{i,j}]}],{i,0,1,1/9},{j,0,1,1/9}],1];
Dimensions[dataxyfn]
{100,3}
```

In our case $m = 2$, therefore the maximum number of the basis is $n_R \sim 33$. Let us employ 6 thin plate spline basic functions,

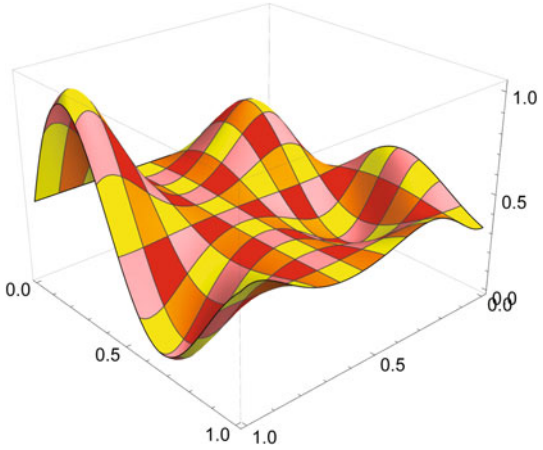


Fig. 9.24 The function to be approximated

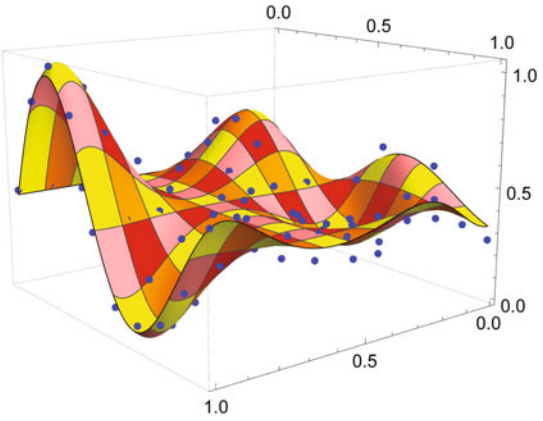


Fig. 9.25 The function with the "noisy" data

```
xym = Map[{#[[1]], #[[2]]}&, dataxyfn]; fm = Map[#[[3]]&, dataxyfn];
```

We organize the basis function in an RBF neural network,

```
<< NeuralNetworks'
rbf = InitializeRBFNet[xym, fm, 10,
 Neuron → Function[x, If[x ≠ 0, x^2 * Log[-x], 0]]]//Quiet
RBFNet[w1, λ, w2}, χ},
  Neuron → Function[x, If[x ≠ 0, x^2 Log[-x], 0]],
  FixedParameters → None, AccumulatedIterations → 0,
  CreationDate → 2016, 11, 12, 18, 27, 10.9401869,
  OutputNonlinearity → None, NumberOfInputs → 2]
```

Let us train the network on the noisy data, see Fig. 9.26.

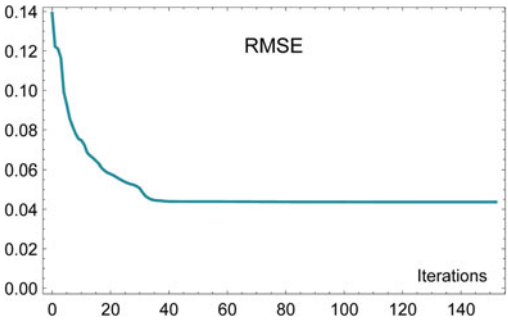


Fig. 9.26 The error of the network during the training process

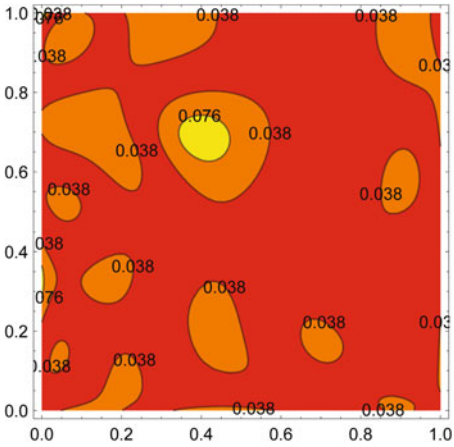


Fig. 9.27 The error of the approximation

```
{rbf2,fitrecord}=NeuralFit[rbf,xym,fm,500]//Quiet;
```

The approximation function is

```
Short[rbf2[{u,v}],20]
{-20.3697 + <<16>> +
  86084.4 If[-4.01953(-0.709155 + u)^2 - 4.01953(-0.0616565 + v)^2 != 0,
  (-4.01953(-0.709155 + u)^2 - 4.01953(-0.0616565 + v)^2)^2
  Log[-(-4.01953(-0.709155 + u)^2 - 4.01953(-0.0616565 + v)^2)],0] -
  86079.9 If[-4.01961(-0.709153 + u)^2 - 4.01961(-0.0616512 + v)^2 != 0,
  (-4.01961(-0.709153 + u)^2 - 4.01961(-0.0616512 + v)^2)^2
  Log[-(-4.01961(-0.709153 + u)^2 - 4.01961(-0.0616512 + v)^2)],0]}
```

The error of the approximation can be seen in Fig. 9.27.

As an alternative solution let us employ Hardy-type multi-quadratic RBF basis functions,

```
rbf = InitializeRBFNet[xym, fm, 10, Neuron → Function[x, √(1 + x^2)]]
RBFNet[w1, λ, w2, χ, Neuron → Function[x, √(1 + x^2)],
 FixedParameters → None, AccumulatedIterations → 0,
 CreationDate → 2016, 11, 12, 18, 27, 30.2686209,
 OutputNonlinearity → None, NumberOfInputs → 2]
```

Training the network (Fig. 9.28),

```
{rbf2, fitrecord} = NeuralFit[rbf, xym, fm, 500]//Quiet;
```

The approximation function is

```
Short[rbf2[{u, v}], 20]
 {30.4649 + 74.3045u -
  86.2706 Sqrt[1 + (-1.48345(-0.23777 + u)^2 -
    1.48345(-1.01835 + v)^2)^2] +
  247.353 Sqrt[1 + (-1.10768(-0.191987 + u)^2 -
    1.10768(-0.696926 + v)^2)^2] -
  8979.46 Sqrt[1 + (-1.60279(-0.0245169 + u)^2 -
    1.60279(-0.47947 + v)^2)^2] +
  9063.27 Sqrt[1 + (-1.60339(-0.0285279 + u)^2 -
    1.60339(-0.475971 + v)^2)^2] -
  7.20718 Sqrt[1 + (-7.58294(-0.459186 + u)^2 -
    7.58294(-0.430761 + v)^2)^2] +
  0.754657 Sqrt[1 + (-16.7615(-0.912564 + u)^2 -
    16.7615(-0.262636 + v)^2)^2] -
  1517.98 Sqrt[1 + (-1.34759(-0.437715 + u)^2 -
    1.34759(-0.24286 + v)^2)^2] +
  1228.5 Sqrt[1 + (-1.42576(-0.487658 + u)^2 -
    1.42576(-0.225803 + v)^2)^2] -
  4.40349*10^6 Sqrt[1 + (-2.85803(-0.272968 + u)^2 -
    2.85803(-0.058178 + v)^2)^2] +
  4.40355*10^6 Sqrt[1 + (-2.85801(-0.272968 + u)^2 -
    2.85801(-0.0581755 + v)^2)^2] - 148.471v}
```

The error of the approximation can be seen in Fig. 9.29.
The error of the approximation is very similar, but the formula is not so complex as was the case of the thin plate spline basic functions.

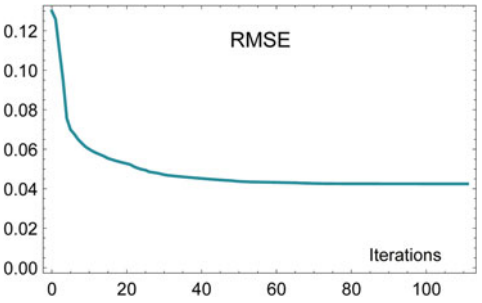


Fig. 9.28 The error of the network during the training process

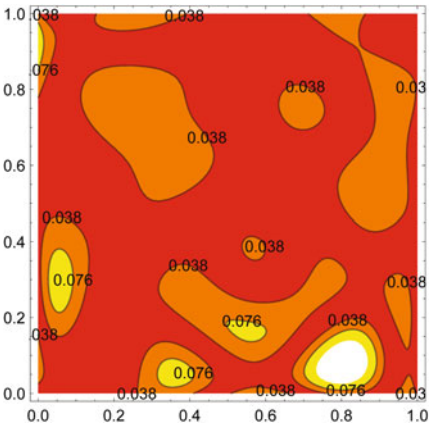


Fig. 9.29 The error of the approximation

9.7 Applications

9.7.1 Image Compression

Image compression methods try to reduce the necessary data of an image without loosing the basic information for the proper image reconstruction. Let us consider a well-known image, see Fig. 9.30.

```
MM = ImageData[image]
Image[MM]
```

The intensity values of the image pixels are between zero and one, see Fig. 9.31. The size of the image is 180×180 pixel giving 32,400 pixels. In order to a create "noisy" image, the non-white pixels will be randomly whitened,

```
MMR = MM;
Do[MMR[[RandomInteger[{1,180}],RandomInteger[{1,180}]]] =
  1,{i,1,10000}];
```

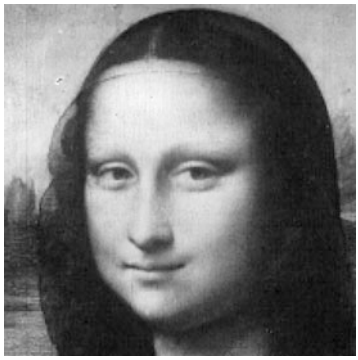


Fig. 9.30 The original image

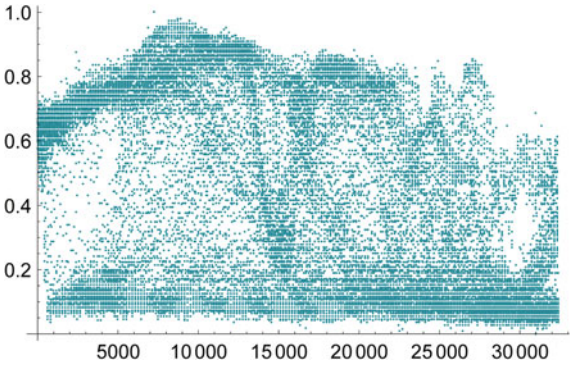


Fig. 9.31 The intensity value of the image pixels

Figure 9.32 shows the image after 10^4 times carried out whitening,

```
Image[MMR]
```

Let us carry out further whitening randomly, see Fig. 9.33.

```
MMR = MM;
Do[MMR[[RandomInteger[{1,180}],RandomInteger[{1,180}]]] =
  1,{i,1,80000}];
Image[MMR]
```

Now, the coordinates of the non black pixels will be normalized, see Fig. 9.34.

```
dataz = {};dataxy = {};
Do[If[MMR[[i,j]] ≠ 1,AppendTo[dataxy,{i/180.,j/180.}];
  AppendTo[dataz,MMR[[i,j]]]],{i,1,180},{j,1,180}]
```

These pixels will be the basic points and their intensity represents the function value to be approximated. Their number is

Fig. 9.32 The noisy image after 8×10^4 random whitening

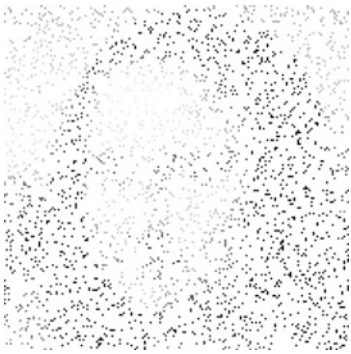


Fig. 9.33 Strongly blurred image after 80,000 additional random whitening

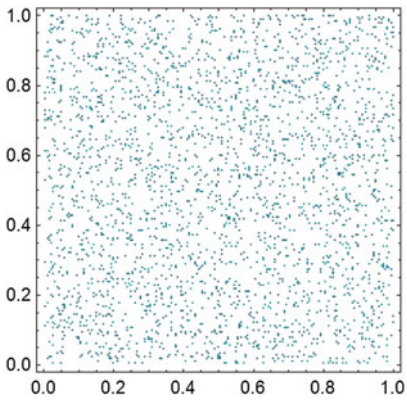


Fig. 9.34 The non-black pixels with normalized coordinates

```
nj = Length[dataxy]
2771
```

which is only

```
Length[dataxy]/(nm) 100.
8.55247
```

percent of the number of the original 32,400 pixels. Let us employ a second order TPS approximation

```
TPSpline2 =
 Function[x,xi,If[x ≠ xi,Norm[x - xi,2]^4 Log[Norm[x - xi,2]],0]]
Function[x,xi,If[x ≠ xi,Norm[x - xi,2]^4 Log[Norm[x - xi,2]],0]]
```

We can employ parallel computation for computing the elements of the coefficient matrix,

```
LaunchKernels[8]
{KernelObject[1,local],KernelObject[2,local],
  KernelObject[3,local],
  KernelObject[4,local],KernelObject[5,local],
  KernelObject[6,local],
  KernelObject[7,local],KernelObject[8,local]}
```

```
DistributeDefinitions[TPSpline2]
{TPSpline2}
```

```
M = ParallelTable[TPSpline2[dataxy[[i]],
 dataxy[[j]]],{i,1,nj},{j,1,nj}];
```

Using pseudoinverse,

```
PM = PseudoInverse[M];
```

we get the 2771 coefficients, see Fig. 9.35.

```
c = PM.dataz;
ListPlot[c,Filling → Axis]
```

With these, one can define the reconstruction (approximation) function as,

```
g[x_,y_] := c.Map[TPSpline2[{x,y},#]&,dataxy]
```

Now, generating 10^4 pixel values in parallel computation, we get the reconstructed image, see Fig. 9.36.

```
DistributeDefinitions[g]
{g,c,TPSpline}
RImage = ParallelTable[g[x,y],{x,0,1,0.01},{y,0,1,0.01}];
Image[RImage]
```

It goes without saying that the reconstructed image is less attractive than the original one, but in case of the original image one should store 32,400 intensity values plus their (x, y) coordinates, while in case of the reconstructed one, we need to store only the 2771 coefficients.

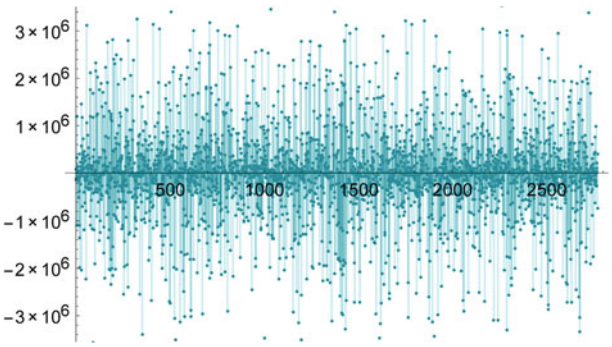


Fig. 9.35 The coefficients of the image reduction problem

Fig. 9.36 The reconstructed image

9.7.2 RBF Collocation Solution of Partial Differential Equation

To avoid finite element meshing problems, we often try meshless methods.

Let us consider a linear elliptic PDE problem,

$$\mathcal{L}\,u(x,y) = f(x,y), \quad x,y \in \Omega,$$

where $\mathcal{L}$ is a linear operator. The boundary condition is,

$$u(x,y) = g(x,y) \quad x,y \in \Gamma.$$

The number of the collocation points are $N = N_\Omega + N_\Gamma$. We are looking for the solution in a form

$$u(x,y) = \sum_{\varphi=1}^{\mathrm{N}_\Omega} c_j \varphi\big(\big\|\{x,y\} - \{x_j,y_j\}\big\|\big) + \sum_{\varphi=\mathrm{N}_\Omega+1}^{\mathrm{N}} c_j \varphi\big(\big\|\{x,y\} - \{x_j,y_j\}\big\|\big).$$

The first term is representing the solution inside the region Ω and the second one is for the boundary. In the region we carry out the substitution for every j index, we get

$$\mathcal{L}u(x,y) = \sum_{\varphi=1}^{\mathrm{N}_\Omega} c_j \mathcal{L}\varphi\big(\big\|\{x,y\} - \{x_j,y_j\}\big\|\big).$$

In matrix form,

$$\mathcal{L}\,\mathbf{u} = \mathbf{A}_\mathcal{L}\mathbf{c},$$

where

$$(\mathbf{A}_\mathcal{L})_{i,j} = \mathcal{L}\varphi\big(\big\|\{x,y\} - \{x_j,y_j\}\big\|\big)_{\{x=x_i,y=y_i\}}, \quad i,j = 1,\ldots,N_\Omega.$$

For the boundary

$$u(x_i,y_i) = g(x_i,y_i) \quad x_i,y_i \in \Gamma$$

we can develop our equation as follows,

$$u(x,y) = \sum_{\varphi=\mathrm{N}_\Omega+1}^{\mathrm{N}} c_j \varphi\big(\big\|\{x,y\} - \{x_j,y_j\}\big\|\big) \quad \rightarrow \mathbf{u} = \mathbf{A}\mathbf{c},$$

where

$$A_{i,j} = \varphi\left(\left\|\{x_i, y_i\} - \{x_j, y_j\}\right\|\right), \quad i,j = N_\Omega + 1, \ldots, N,$$

in matrix form

$$\mathbf{Ac} = \mathbf{g}.$$

Collecting our equations for the unknown coefficients, we get

$$\begin{pmatrix} \mathbf{A}_\mathcal{L} \\ \mathbf{A} \end{pmatrix} \mathbf{c} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}.$$

where $f_i = f(x_i, y_i),\ i \in [1, \ldots, N_\Omega]$ and $g_i = g(x_i, y_i),\ i \in [N_\Omega + 1, \ldots, N]$.

Here, we consider the following problem:
In the region

$$\Delta u(x, y) = -\frac{5\pi^2}{4} sin(\pi x) cos\left(\frac{\pi y}{2}\right), \quad x, y \in \Omega,$$

on the boundary

$$\Delta u(x, y) = sin(\pi x) cos\left(\frac{\pi y}{2}\right), \quad x, y \in \Gamma,$$

where

$$\Omega = [0, 1]^2 - [0.5, 1]^2$$

and Γ is the boundary of the region Ω, see Fig. 9.37.
The solution will be approximated via the following RBF functions,

$$u(x, y) \approx \sum_{i=1}^{ni} c_i \varphi(\{x, y\}, \{x_i, y_i\}) + \sum_{i=ni+1}^{ni+nb} c_i \varphi(\{x, y\}, \{x_i, y_i\}).$$

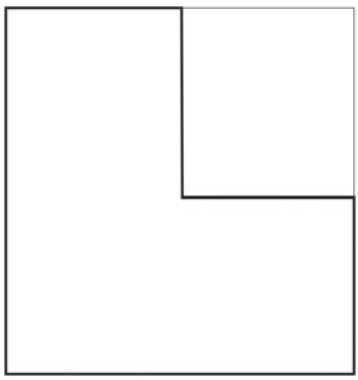


Fig. 9.37 The region of problem

The first term is for the region and the second one is for the boundary. Now we use a thin plate spline of second order

$$\varphi(\{x,y\},\{x_i,y_i\}) = \|\{x,y\},\{x_i,y_i\}\|^4 \log\|\{x,y\},\{x_i,y_i\}\| \quad \text{if}\{x,y\} \neq \{x_i,y_i\}$$

and

$$\varphi(\{x,y\},\{x_i,y_i\}) = 0, \quad \text{otherwise.}$$

The Laplace operator in case of a $\varphi(r)$ basis function can be expressed as (see Sect. 9.5).

$$\Delta u(x,y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2},$$
$$\Delta u(x,y)\varphi(r) = \frac{d^2}{dr^2}\varphi(r) + \frac{1}{r}\frac{d}{dr}\varphi(r).$$

For second order TPS, it can be written as

$$\varphi(r) = r^4 \log(r),$$
$$\tfrac{d}{dr}\varphi(r) = r^3(4\log(r)+1),$$
$$\tfrac{d^2}{dr^2}\varphi(r) = r^2(12\log(r)+7).$$

Then

$$\Delta u(x,y)\varphi(r) = \frac{d^2}{dr^2}\varphi(r) + \frac{1}{r}\frac{d}{dr}\varphi(r) = r^2(12\log(r)+7) + \frac{1}{r}r^3(4\log(r)+1)$$

and the Laplace operator is,

$$\Delta u(x,y)\varphi(r) = r^2(16\log(r)+8).$$

The implementation of the problem in *Mathematica* is easy. First we generate Halton points as collocation points inside the region,

```
corput[n_,b_]:=
 IntegerDigits[n,b].(b^Range[-Floor[log[b,n]+1],-1]);
SetAttributes[corput,Listable]

halton[n_,s_]:=corput[n,Prime[Range[s]]]
```

Then we define the Laplace function as

```
Δφ=Function[{x,xi},
 If[x≠xi,Norm[x-xi,2]^2(16Log[Norm[x-xi,2]]+8),0]]
Function[{x,xi},If[x≠xi,
  Norm[x-xi,2]^2(16 Log[Norm[x-xi,2]]+8),0]]
```

Let us visualize this function in 2D, see Fig. 9.38.

```
z[r_, φ_] := Δφ[x, {0, 0}]/.x → {r cos[φ], r sin[φ]}
```

The basis function is,

```
TPSpline2 =
 Function[x, xi, If[x! = xi, Norm[x - xi, 2]^4 Log[Norm[x - xi, 2]], 0]]
Function[{x, xi}, If[x ≠ xi, Norm[x - xi, 2]^4 Log[Norm[x - xi, 2]], 0]]
```

Let the number of the collocation points in the region of [0, 1] × [0, 0.5] be `ni1`, while in the region of [0, 0.5] × [0, 0.5] is `ni2`,

```
ni2 = 50; ni1 = 100;
dataxy2 =
 Map[{#[[1]] 0.5, #[[2]] 0.5 + 0.5} &, Table[halton[n, 2], {n, ni2}]];
dataxy1 = Map[{#[[1]], 0.5 #[[2]]} &, Table[halton[n, 2], {n, ni1}]];
```

see Fig. 9.39.
The boundary points are

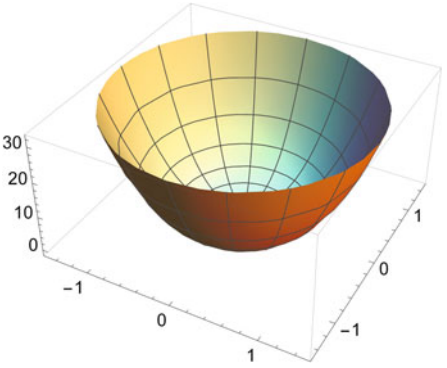


Fig. 9.38 The second order Laplace operator in our case

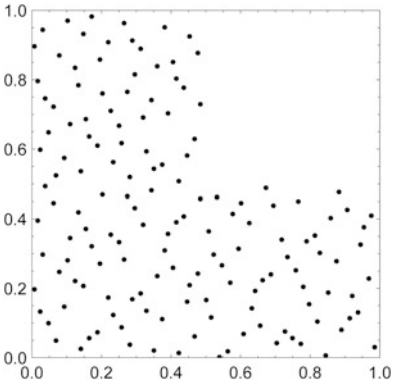


Fig. 9.39 The Halton points as collocation points in the region

```
dataA = Map[{#, 0}&, Range[0, 1, 0.05]];
dataB = Map[{1, #}&, Range[0.1, 0.5, 0.05]]
dataC = Map[{#, 0.5}&, Range[0.5, 0.95, 0.05]]
dataD = Map[{0.5, #}&, Range[0.55, 0.95, 0.05]]
dataE = Map[{#, 1}&, Range[0, 0.5, 0.05]]
dataF = Map[{0, #}&, Range[0.1, 0.95, 0.05]]
dataBoundary = Join[dataA, dataB, dataC, dataD,
  dataE, dataF]//Union;
```

We have 150 points inside the region and 78 points on the boundary, see Fig. 9.40.

```
nb = Length[dataBoundary]
78
```

Let us put these points together

```
dataxy12 = Join[dataxy1, dataxy2];
ni = ni1 + ni2; nj = ni + nb;
dataTotal = Join[dataxy12, dataBoundary];
```

The coefficient matrix is

$$\mathbf{MF} = \begin{pmatrix} \mathbf{A}_{\mathcal{L}} \\ \mathbf{A} \end{pmatrix}.$$

```
MF = Table[If[i ≤ ni, Δφ[dataTotal[[i]], dataTotal[[j]]],
  TPSpline2[dataTotal[[i]], dataTotal[[j]]]], {i, 1, nj}, {j, 1, nj}];
```

Its size is

```
Dimensions[MF]
{228, 228}
```

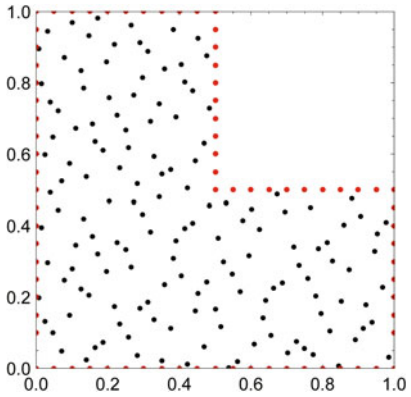


Fig. 9.40 The collocation points in the region and on the boundary

The right side vector can be generated as $\begin{pmatrix} f \\ g \end{pmatrix}$

```
f[x_]:= - 5π^2/4 sin[πx[[1]]]cos[π/2 x[[2]]]
g[x_]:=Sin[πx[[1]]]Cos[π/2 x[[2]]]
fg=
  Table[If[i≤ni,f[dataTotal[[i]]],g[dataTotal[[i]]]],
   {i,1,nj}];
Dimensions[fg]
{228}
```

The linear equation system for the coefficients can be written as

$$\mathbf{MFc} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}.$$

The solution of this linear system gives the coefficient vector, **c**, whose elements can be seen in Fig. 9.41.

```
cc=PseudoInverse[MF].fg;
```

Therefore the approximation function is

```
g[{x_,y_}]:=cc.Map[TPSpline2[{x,y},#]&,dataTotal]
```

We can compare our RBF approximation with the exact solution, which is

$$u(x, y) = \sin(\pi x)\cos\left(\pi \frac{y}{2}\right).$$

```
G[{x_,y_}]:=Sin[πx]Cos[πy/2]
```

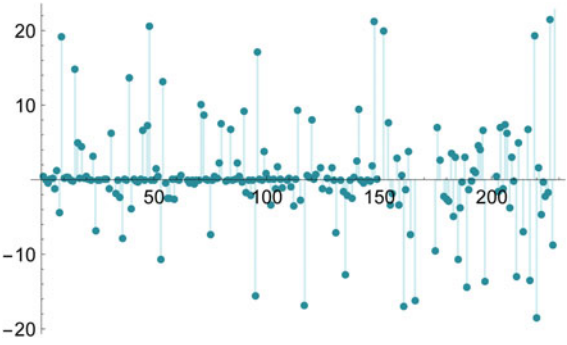


Fig. 9.41 The coefficients of our problem

The exact values in the collocation points are

```
dataz = Map[G[#]&, dataTotal]//N;
dataxyz = MapThread[Flatten[{#1, #2}]&, {dataTotal, dataz}];
```

Figure 9.42 shows the exact values in the collocation points with the surface of the approximation function.

The error function can be expressed as (Fig. 9.43).

```
Δ[{x_, y_}] := Abs[g[{x, y}] - G[{x, y}]]
```

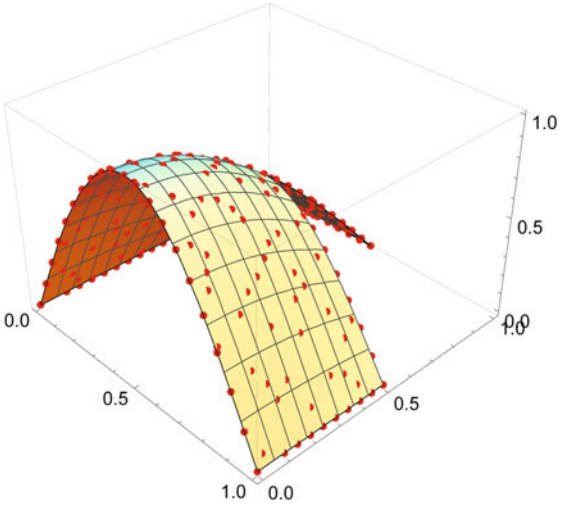


Fig. 9.42 The exact collocation values and the surface of the RBF approximation solution

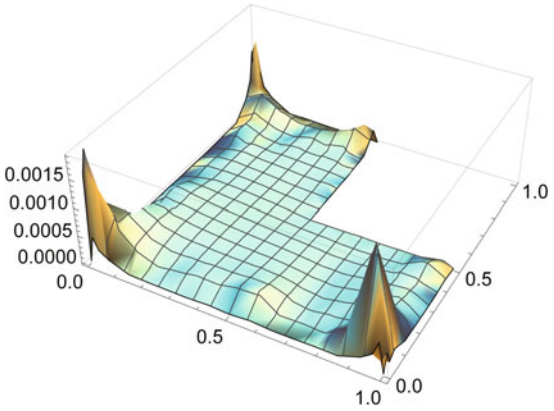


Fig. 9.43 The error function of the RBF approximation solution

As expected, the approximation error is highest in the corners of the region, but even here the error is less than 0.002.

9.8 Exercise

9.8.1 Nonlinear Heat Transfer

Let us consider again our nonlinear steady state heat transfer problem in 1D,

$$\frac{d}{dx}\left(\lambda(\theta)\frac{d\theta}{dx}\right) = 0.$$

The boundary conditions are,

$$\theta(0) = 0 \quad \text{and} \quad \theta(1) = 1.$$

The heat transfer coefficient depending on the temperature is

$$\lambda(\theta) = 1 + k\,\theta.$$

Now we should employ RBF collocation with the Hardy-type multi-quadratic RBF.

Solution The differential equation with $k = 1$ is

```
D[(1 + k θ[x])D[θ[x],x],x]/.k → 1
θ'[x]^2 + (1 + θ[x])θ''[x]
```

The RBF function can be written as

```
Hardy = Function[{x,xi}, √(1 + Abs[x - xi]^2)]
Function[{x,xi}, √(1 + Abs[x - xi]^2)]
```

Its first derivative is

```
D[√(1 + r^2), r]//Simplify
r/√(1 + r^2)
```

Then substituting our RBF into it, we get the first derivative

```
dθ = Function[{x,xi}, Abs[x - xi](1 + Abs[x - xi]^2)^-1/2]
Function[{x,xi}, Abs[x - xi] 1/√(1 + Abs[x - xi]^2)]
```

The second derivate can similarly be obtained since

```
D[√(1 + r^2), r, r]//Simplify
```

$$\frac{1}{(1 + r^2)^{3/2}}$$

```
ddθ = Function[{x, xi}, (1 + Abs[x - xi]^2)^-3/2]
Function[{x, xi}, (1 + Abs[x - xi]^2)^-3/2]
```

Using these derivatives, our differential equation can be written as

```
d = Function[{x, xi}, (Abs[x - xi] (1 + Abs[x - xi]^2)^-1/2)^2 +
      (1 + √(1 + Abs[x - xi]^2)) (1 + Abs[x - xi]^2)^-3/2]
```

Let us chose 4 collocation points

```
dataTotal = {1/3, 2/3, 0, 1}
```

$$\left\{\frac{1}{3}, \frac{2}{3}, 0, 1\right\}$$

Then the coefficient matrix is

```
MF = Table[If[i ≤ 2, d[dataTotal[[i]], dataTotal[[j]]],
  Hardy[dataTotal[[i]], dataTotal[[j]]]], {i, 1, 4}, {j, 1, 4}]//N;
MF//MatrixForm
```

$$\begin{pmatrix} 2. & 1.85381 & 1.85381 & 1.57603 \\ 1.85381 & 2. & 1.57603 & 1.85381 \\ 1.05409 & 1.20185 & 1. & 1.41421 \\ 1.20185 & 1.05409 & 1.41421 & 1. \end{pmatrix}$$

The boundary conditions in the collocation points are

```
fg = {0, 0, 0, 1}
{0, 0, 0, 1}
```

The solution of the linear system for the unknown coefficients of the Hardy-Function is

```
cc = PseudoInverse[MF].fg//N
{-8.21461, 6.02496, 4.97703, -2.51671}
```

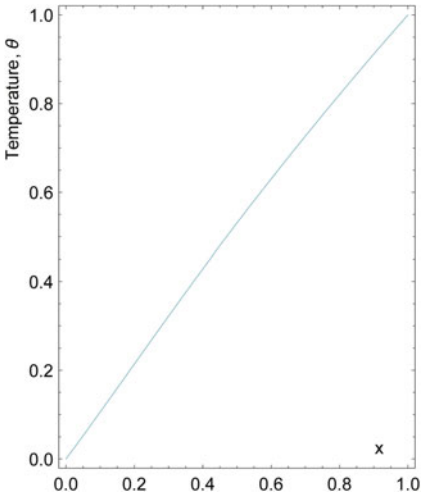


Fig. 9.44 The temperature profile employing the Hardy-type multi-quadratic RBF

Employing these coefficient values, the approximation function for the temperature distribution in analytical form can be written as

```
cc.Map[Hardy[x,#]&,dataTotal]
```

$$-2.51671\sqrt{1+\text{Abs}[-1+x]^2}+6.02496\sqrt{1+\text{Abs}\left[-\tfrac{2}{3}+x\right]^2}-$$
$$8.21461\sqrt{1+\text{Abs}\left[-\tfrac{1}{3}+x\right]^2}+4.97703\sqrt{1+\text{Abs}[x]^2}$$

The compact form of this function is

```
g[x_]:=cc.Map[Hardy[x,#]&,dataTotal]
```

Figure 9.44 shows the temperature profile.

References

Fasshauer GE (2007) Meshfree approximation methods with MATLAB. Interdisciplinary mathematical sciences, vol 6. World Scientific, New Jersey

Mangano S (2010) Mathematica cookbook. O'REILLY, Beijing, Cambridge, Farnham, Köln

Micchelli CA, Xu Y, Zhang H (2006) Universal kernels. Journal of Machine Learning Research 7:2651–2667

Wendland H (2005) Scattered data approximation, Cambridge monographs on applied and computational mathematics. Cambridge University Press, New York

Chapter 10
Support Vector Machines (SVM)

10.1 Concept of Machine Learning

In statistical learning theory (regression, classification, etc.) there are many regression models, such as algebraic polynomials, Fourier series, radial basis functions (RBF), neural networks, fuzzy logic models, and time series. In all of these the most important feature is the model's ability to generalize to other, future data.

The Learning machine is the model of the real System. The learning machine estimates the output of the system (y) (see Fig. 10.1) by adjusting the parameters (w) during its training phase. After training, in the generalization or validation phase, the output from the machine is expected to be a good estimate of the system's true response, y. At that point the learning machine will rarely try to interpolate training data pairs, but would rather seek an approximating function that can generalize well. In other words, during the training phase, the data (x) are from a training set (H) but a good approximation is expected on a target set (Z) where $Z \supset H$. There is a trade-off between the approximation error on H and the generalization error on Z–H. So we are looking for a *type of approximation function* f_B that has good performance on Z. In practice we can find only an f that has less than optimal behavior. Seeking the best function that can generalize is achieved through *structural risk minimization*. There are different techniques of constructing good estimators:

(a) *cross validation* divides the data into two parts; a training set and a validation set. The validation set is used *after* the training phase.

  (a.1) The cross validation can be used for comparing several learning machines, then choosing the model which provides the best trade-off between the error on the training and validation set.
  (a.2) Another technique is when the validation set is *monitored* during the training phase. When the error starts to increase on the validation set, the training will be stopped (early stopping).

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_10

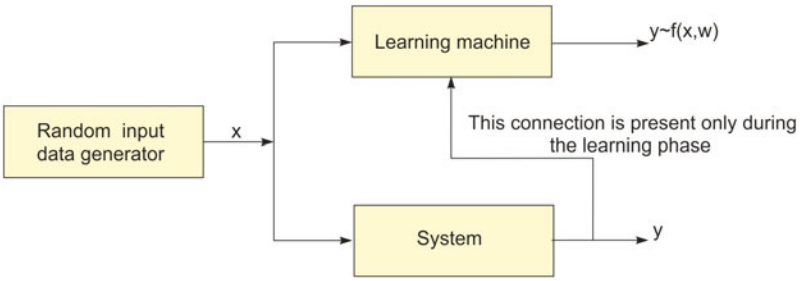


Fig. 10.1 Operation of a learning machine in case of supervised learning

(b) *regularization*, when the error function to be minimized on H is extended with an additional term, which is responsible for the smoothness of the approximation function. For example this term can be the square of the gradient of the approximating function.

Remark In general, we use the training set and validation set during developing our model. So the data of the validation set are also involved in the training phase. After that test data can be used as a last check for our model generalization ability.

The *support vector machine* provides a further method to construct an optimal estimator via structural risk minimization.

10.2 Optimal Hyperplane Classifier

The Support Vector Machine (SVM) is a very effective so called "kernel method" for classification as well as regression, even in cases of strongly nonlinear problems. First, we start with classification. We shall show how we can extend a linear maximal margin classifier for use as a nonlinear classifier employing kernel techniques. The SVM technique can be especially competitive when the vectors of the elements to be classified are very long.

10.2.1 Linear Separability

In many problems we wish to separate clusters of data with a line, plane, or hyper-plane. This is called *separable linear classification.* The plane is called a *decision plane.* Obviously, there are almost always infinitely many such planes, but some are better than others. We would like to select a decision plane that is "in the

middle", where the distances of closest points of the two different clusters are the maximum; see Fig. 10.2.
Let us consider a hyperplane as a decision plane. Here is a decision line separating the two clusters

$$y = wx + b.$$

For the boundary elements of the cluster dark blue squares (x_1), let

$$wx_1 + b = 1$$

and for the boundary elements of the cluster light yellow circle (x_2), let

$$wx_2 + b = -1.$$

Then

$$\mathrm{w}\ (x_1 - x_2) = 2.$$

The distance of the boundary points is

$$x_1 - x_2 = \frac{2}{w}.$$

Consequently, w is minimized in order to decrease the structural risk of the linear separation. In general, the scalar product

$$\frac{1}{2}\langle w, w\rangle$$

should be minimized. Figure 10.3 shows two solutions for the clustering problem.

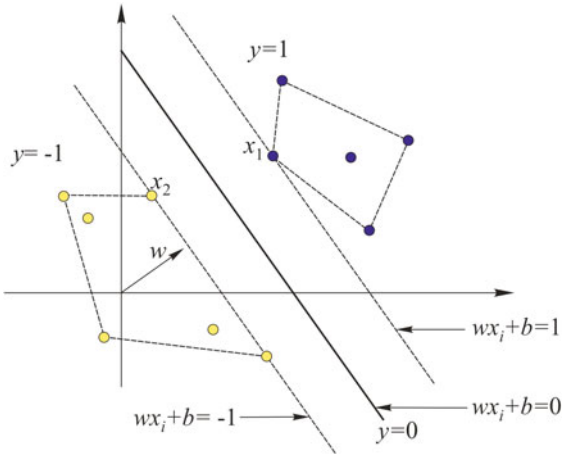


Fig. 10.2 A separable classification problem

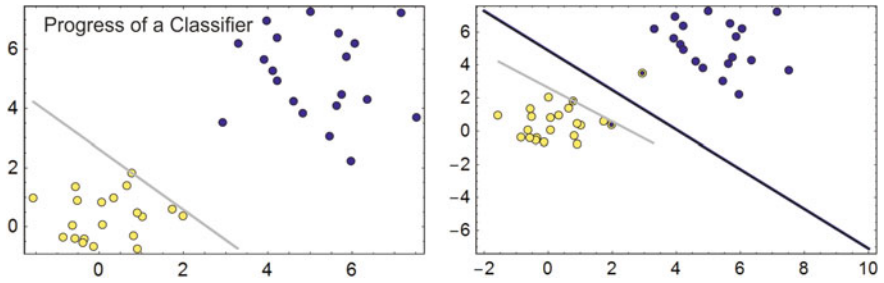


Fig. 10.3 Two different solutions for classification problem. Left: perceptron neural network, Right: support vector machine

The solution on the left-hand side was computed by a neural network's "perceptron". It provides higher structural risk than the solution on the right-hand side computed via an SVM linear classifier, because it just barely works. Should the data have small errors and the points move slightly, that decision line will no longer be correct.

Since for the light yellow circles,

$$\mathrm{w}x_i + b \leq -1,$$

and for the dark blue squares,

$$\mathrm{w}x_i + b \geq 1,$$

let $y_i = -1$ be valid for the points belonging to the gray cluster and $y_i = 1$ for the cluster of the black points. Then, the constraints can be written in general form as

$$y_i(wx_i + b) \geq 1, \quad i = 1, 2, \ldots, n, \quad x_i \in S.$$

We are looking for the weight vector in the following form

$$w = \sum_{i=1}^{n} \alpha_i x_i, \quad \text{where} \quad x_i \in S.$$

Therefore, the decision function can be written as

$$f(x) = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle + b\right).$$

where $f(x_i) = 1$ or -1 depending on which is the proper class for the element x_i.

10.2.2 Computation of the Optimal Parameters

The optimal α_i and b parameters can be computed via minimization of the following objective function

$$L(\alpha_i, b) = \frac{1}{2}w^2 + \sum_{i=1}^{n} \alpha_i(y_i(wx_i + b) - 1) \rightarrow \min .$$

This is the Lagrange form of the constrained optimization problem. In practice the optimal parameters will be computed from its dual form, see Berthold and Hand (2003)

$$G(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \left(\langle x_i, x_j \rangle + \frac{1}{c}\delta_{i,j} \right) \rightarrow \max$$

under the constraints,

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, n.$$

Let us suppose that the optimal solutions are α_i, $i = 1,\ldots, n$. Then our decision function is,

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle + b \right) = \sum_{i=1}^{n} y_i \alpha_i \langle x_i, x \rangle + b.$$

In the primary problem w is the sum of the α_i x_i. Here, c is a constant parameter given by the user. The elements for $\alpha_i \neq 0$, i = 1, 2,. . ., m are true and called *support vectors*. Compare this phrase with the support of a function as in the previous chapter.

The value of the parameter b should be considered, that for all i index where $\alpha_i \neq 0$, will be true

$$y_j f(x_j) = 1 - \frac{\alpha_j}{c}, \quad j = 1, 2, \ldots n.$$

Considering

$$f(x_j) = \sum_{i=1}^{n} y_i \alpha_i \langle x_i, x_j \rangle + b, \quad j = 1, 2, \ldots n$$

we get

$$b = \frac{1 - \frac{\alpha_j}{c}}{y_j} - \sum_{i=1}^{n} y_i \alpha_i \langle x_i, x_j \rangle, \quad j = 1, 2, \ldots n.$$

The reliability of the solution can be tested via controlling that for all j one gets the same b value.

10.2.3 Dual Optimization Problem

Since the dual form of the optimization problem in case of the determination of the parameters of the SVM approach plays a very important role, let us recall this technique.

First, we consider a linear problem,

$$\min \langle c^T, x \rangle, \quad Ax \geq b, \quad x \geq 0,$$

then its dual form is

$$\max \langle b^T, y \rangle, \quad A^T y \leq c, \quad y \geq 0.$$

In a nonlinear case the situation is more complicated. Let us consider the primary problem as

$$f(x) \to \min,$$

$$g_i(x) \leq 0, \quad i = 1, 2, \ldots, m,$$

$$h_i(x) = 0, \quad i = 1, 2, \ldots, p.$$

The Lagrange form of the problem is,

$$\mathcal{L}(x, \alpha, \beta) = f(x) + \sum_{i=1}^{m} \alpha_i g_i(x) + \sum_{i=1}^{p} \beta_i h_i(x),$$

where the variable x *is* called the *primary* variable while the variables α and β are called as *dual* variables.

Then, we consider the following primary problem

$$\min_x \left(\max_{a,b} \mathcal{L}(x, a, \beta) \right) = \min_x W_P(x)$$

where

$$W_P(x) = \max_{\alpha,\beta} \mathcal{L}(x, \alpha, \beta).$$

The dual form of this problem is

$$\max_{\alpha,\beta} \left(\min_{x} \mathcal{L}(x, \alpha, \beta) \right) = \max_{\alpha,\beta} W_P(\alpha, \beta),$$

where

$$W_P(\alpha, \beta) = \min_{x} \mathcal{L}(x, \alpha, \beta).$$

Let us consider a numerical illustration. Consider a simple problem as

$$x_1^2 + x_2^2 \rightarrow \min,$$

$$4 - 2x_1 - x_2 \leq 0.$$

The primary direct solution is

```
Minimize[{x1^2 + x2^2, 4 - 2x1 - x2 ≤ 0}, {x1, x2}]
```

$$\left\{\frac{16}{5}, \left\{x_1 \rightarrow \frac{8}{5}, x_2 \rightarrow \frac{4}{5}\right\}\right\}$$

The Lagrange form of the problem is

$$\mathcal{L}(x, \alpha) = x_1^2 + x_2^2 + \alpha(4 - 2x_1 - x_2).$$

Then, the dual form to be maximized is

$$W_D(\alpha) = \min_{x} \mathcal{L}(x, \alpha).$$

Considering the necessary conditions

$$\frac{\partial \mathcal{L}(x, \alpha)}{\partial x_1} = 2x_1 - 2\alpha = 0 \rightarrow x_1 = \alpha,$$

$$\frac{\partial \mathcal{L}(x, \alpha)}{\partial x_2} = 2x_2 - \alpha = 0 \rightarrow x_2 = \frac{\alpha}{2},$$

then, for the dual form, we get

$$W_D(\alpha) = \min_x \mathcal{L}(x, \alpha) = \mathcal{L}(x, \alpha)\big|_{x_1=\alpha, x_2=\frac{\alpha}{2}} = 4\alpha - \frac{5\alpha^2}{4},$$

hence the dual problem is

$$\max_\alpha \left(4\alpha - \frac{5\alpha^2}{4}\right), \quad \alpha \geq 0,$$

whose solution is

```
Maximize[{4α - 5α^2/4, α^3 0}, α]
```

$$\left\{\frac{16}{5}, \left\{\alpha \to \frac{8}{5}\right\}\right\}$$

Consequently

$$x_1 = \alpha = \frac{8}{5} \quad \text{and} \quad x_2 = \frac{\alpha}{2} = \frac{4}{5}.$$

10.3 Nonlinear Separability

It may not be possible to separate clusters with linear hyper-planes, so we have the concept of nonlinear separable problems. However, we can transform such problems into a higher dimensional space, where they can be considered as a linear separable problem. To illustrate such a situation, we consider here the XOR problem.

The input—output logical table can be seen in Table 10.1.
The geometrical interpretation of the XOR problem can be seen in Fig. 10.4. It is clear that one linear line cannot separate the points of the class of true ($y = 1$) from the points of class of false ($y = 0$).
However, introducing a new input variable,

$$x_3 = x_1 x_2.$$

Table 10.1 Input (x_1, x_2)—output (y) logical table of the XOR problem

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

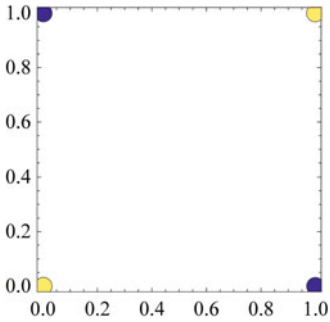


Fig. 10.4 Geometrical representation of the XOR problem

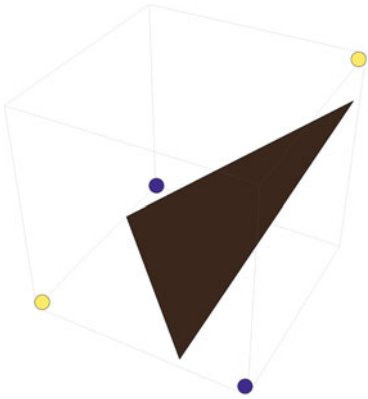


Fig. 10.5 Geometrical representation of the XOR problem with three inputs

Table 10.2 Input (x_1, x_2, x_3)—output (y) logical table of the XOR problem

x_1	x_2	x_3	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The input—output logical table is,
the XOR problem can be transformed into a linearly separable problem in 3D; see Fig. 10.5. A feasible decision plane can be defined with the following three points P_1(0.7, 0., 0.), P_2(0., 0.7, 0.) and P_3(1., 1., 0.8) (Table 10.2).

10.4 Feature Spaces and Kernels

The objective function of the dual form of the linear separable problem contains the scalar product (linear kernel) of the coordinates of the points to be classified, $\langle x_i, x_j \rangle$. A nonlinear separable problem can be transformed into a linear one via changing this linear kernel into a nonlinear one, i.e., $\langle x_i, x_j \rangle \rightarrow \langle \Phi(x_i), \Phi(x_j) \rangle$, see Fig. 10.6.

This transformation maps the data into some other dot space, called the feature space F. Recalling the dual form $G(\alpha)$ in Sect. 10.2.2, this only requires the evaluation of dot products, $K(x_i, x_j)$.

$$G(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \left(K(x_i, x_j) + \frac{1}{c} \delta_{i,j} \right) \rightarrow \max.$$

Clearly, if F is a high dimensional feature space, the dot product on the right hand side will be very expensive to compute. In some cases, however there is a simple *kernel* that can be evaluated efficiently. For instance, the polynomial kernel

$$K(\mathrm{u}, \mathrm{v}) = \langle u, v \rangle^d,$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly d dimensions of R^n. For $d = 2$ and $u, v \in R^2$, for example, we have

$$\langle u, v \rangle^2 = \left(\left\langle \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \right\rangle \right)^2 = \langle \Phi(u), \Phi(v) \rangle = \left\langle \begin{pmatrix} u_1^2 \\ \sqrt{2} u_1 u_2 \\ u_2^2 \end{pmatrix}, \begin{pmatrix} v_1^2 \\ \sqrt{2} v_1 v_2 \\ v_2^2 \end{pmatrix} \right\rangle.$$

defining $\Phi(x) = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2 \right)$.

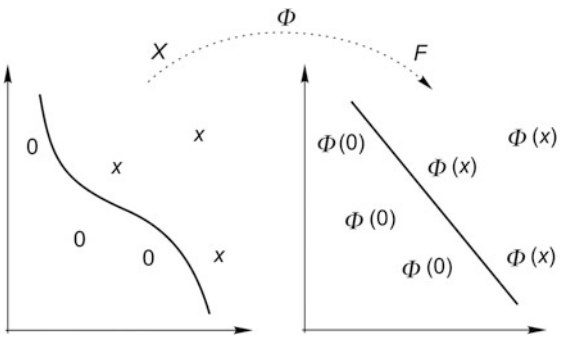


Fig. 10.6 The idea of nonlinear SVM

More generally, we can prove that for every kernel that gives rise to a positive matrix (kernel matrix) $M_{i,j} = K(x_i, x_j)$, we can construct a map such that $K(u, v) = \langle \Phi(u), \Phi(v) \rangle$ holds.

10.5 Application of the Algorithm

10.5.1 Computation Step by Step

The dual optimization problem can be solved conveniently using *Mathematica*. In this section, the steps of the implementation of the support vector machine classification (SVMC) algorithm are shown by solving XOR problem. The truth table of XOR, using bipolar values for the output, see Table 10.3.
The input and output data lists are

```
xym = {{0,0},{0,1},{1,0},{1,1}};
zm = {-1,1,1,-1};
```

Let us employ Gaussian kernel with β gain

```
β = 10.;
K[u_,v_] := Exp[-β(u-v).(u-v)]
```

The number of the data pairs in the training set, m is

```
m = Length[zm]
4
```

Create the objective function $W(\alpha)$ to be maximized, with regularization or penalty parameter, $c = 5$

```
c = 5.;
```

Table 10.3 Logical table of the XOR problem

x_1	x_2	y
0	0	−1
0	1	1
1	0	1
1	1	−1

First, we prepare a matrix *M,* which is an extended form of the kernel matrix,

```
M = (Table[N[K[xym[[i]], xym[[j]]]], {i, 1, m}, {j, 1, m}] +
 (1/c) IdentityMatrix[m]);
```

then the objective function can be expressed as,

$$\texttt{W} = \sum_{\texttt{i}=1}^{\texttt{m}} \alpha_{\texttt{i}} - \frac{1}{2}\sum_{\texttt{i}=1}^{\texttt{m}}\sum_{\texttt{j}=1}^{\texttt{m}} (\texttt{zm[[i]]zm[[j]]}\alpha_{\texttt{i}}\alpha_{\texttt{j}}\texttt{M[[i,j]]});$$

The constrains for the unknown variables are

$$\texttt{g} = \texttt{Apply}\left[\texttt{And}, \texttt{Join}\left[\texttt{Table}[\alpha_{\texttt{i}} \geq 0, \{\texttt{i}, 1, \texttt{m}\}], \left\{\sum_{\texttt{i}=1}^{\texttt{m}} \texttt{zm[[i]]}\alpha_{\texttt{i}} == 0\right\}\right]\right].$$

$$\alpha_1 \geq 0 \&\& \alpha_2 \geq 0 \&\& \alpha_3 \geq 0 \&\& \alpha_4 \geq 0 \&\& -\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 == 0$$

The solution of this constrained optimization problem will results in the unknown variables α_i. However, this is a quadratic optimization problem for α, we use global method with local post processing,

```
sol = NMaximize[{W, g}, vars, Method → {"DifferentialEvolution",
"CrossProbability" → 0.3, "PostProcess" → Automatic}]
{1.66679, {α1 → 0.833396, α2 → 0.833396, α3 → 0.833396, α4 → 0.833396}}
```

The consistency of this solution can be checked by computing values of parameter *b* for every data point. Theoretically, these values should be the same for any data point, however, in general, this is only approximately true.

$$\texttt{bdata} = \texttt{Table}\left[\left(\left(1 - \frac{\alpha_{\texttt{j}}}{\texttt{c}}\right)/\texttt{zm[[j]]} - \sum_{\texttt{i}=1}^{\texttt{m}} \texttt{zm[[i]]}\alpha_{\texttt{i}}\texttt{K[xym[[i]], xym[[j]]]}\right) /.\texttt{sol[[2]]}, \{\texttt{j}, 1, \texttt{m}\}\right]$$

$$\{-2.3063 \times 10^{-10}, -4.92456 \times 10^{-10}, 1.07811 \times 10^{-9}, -3.55027 \times 10^{-10}\}$$

The value of *b* can be chosen as the average of these values

```
b = Apply[Plus, bdata]/m
-1.38439 × 10^-17
```

Then the classifier function is,

$$\texttt{f[w\_]:=}\left(\left(\sum_{\texttt{i=1}}^{\texttt{m}}\texttt{zm[[i]]}\alpha_{\texttt{i}}\texttt{K[w,xym[[i]]]}\right)+\texttt{b}\right)\texttt{/.sol[[2]]}$$

In analytic form

```
f[{x,y}]//Chop
-0.833396e^-10.((-1+x)^2+(-1+y)^2) + 0.8333962e^-10.(x^2+(-1+y)^2) +
  0.833396e^-10.((-1+x)^2+y^2) - 0.833396e^-10.(x^2+y^2)
```

Let us display the contour lines of the continuous classification function, see Fig. 10.7.
For the discrete classifier, the decision rule using signum function is, see Fig. 10.8.
Let us check the classifier by substituting the input pairs. We get

```
Map[Sign[f[#]]&,xym]
{-1,1,1,-1}
```

The optimal separation boundary lines $x = 1/2$ and $y = 1/2$ crossing at $\{0.5, 0.5\}$, since

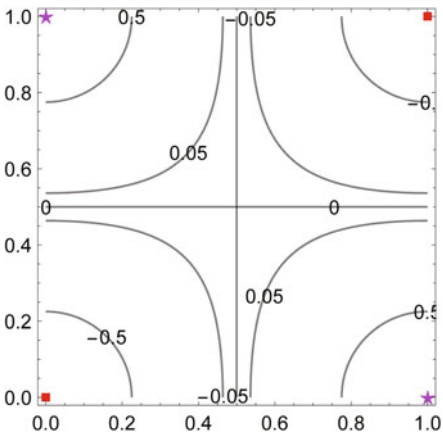


Fig. 10.7 The contour lines of the continuous classification function $f(x, y)$ for XOR problem

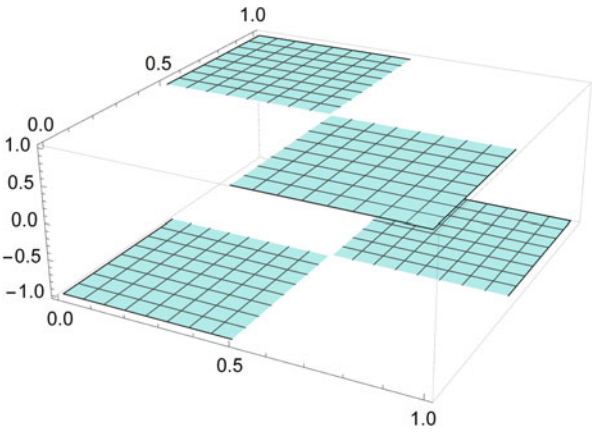


Fig. 10.8 The decision rule, sign($f(x, y)$) for XOR problem

```
Sign[f[{0.75,0.25}]]
```

1

```
Sign[f[{0.75,0.75}]]
```

-1

10.5.2 Implementation of the Algorithm

The SVM classification algorithm (SVMC) can be implemented in the following *Mathematica* function,

```
SupportVectorClassifier[xm_,ym_,K_,c_]:=Module[
 {m,n,M,i,j,W,g,vars,sol,bdata,b},
 m=Length[ym];n=Length[xm[[1]]];
 M=Table[K[xm[[i]],xm[[j]]],{i,1,m},{j,1,m}]+
   1/c IdentityMatrix[m];
 W=Sum[α[i],{i,1,m}]-1/2 Sum[Sum[(ym[[i]]ym[[j]]α[i]α[j]M[[i,j]]),{i,1,m}],{j,1,m}];
```

```
g = Apply[And, Join[Table[α_i^3 0, {i, 1, m}], {Sum_{i=1}^{m} ym[[i]] α_i 0}]];
vars = Table[α_i, {i, 1, m}];
sol = FindMaximum[{W, g}, vars, Method"InteriorPoint"][[2]];
bdata = Table[((1 - α_j/c)/ym[[j]] - Sum_{i=1}^{m} ym[[i]] α_i K[xm[[i]], xm[[j]]])/.
  sol[[2]], {j, 1, m}];
b = Apply[Plus, bdata]/m;
{(Sum_{i=1}^{m} ym[[i]] α_i K[Table[x_j, {j, 1, n}], xm[[i]]] + b)]
/.sol, vars/.sol};
```

The output of this function is the analytical form of the SVMC function.

Let us employ this function for a linear problem, see Fig. 10.9.
We use linear kernel,

```
K[u_, v_] := u.v
```

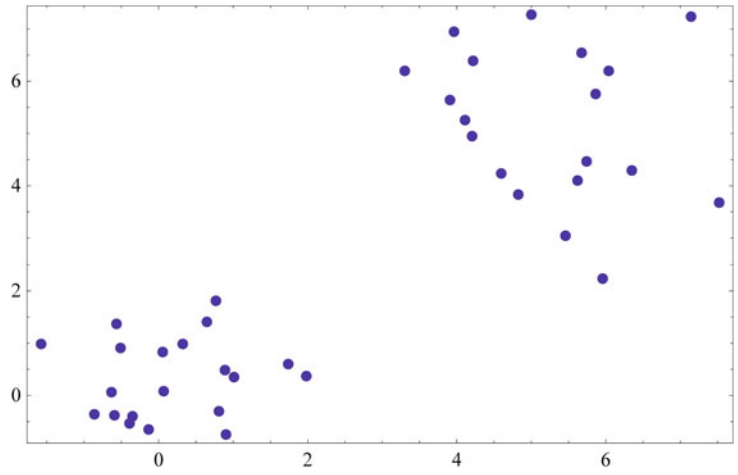


Fig. 10.9 A linear separable problem

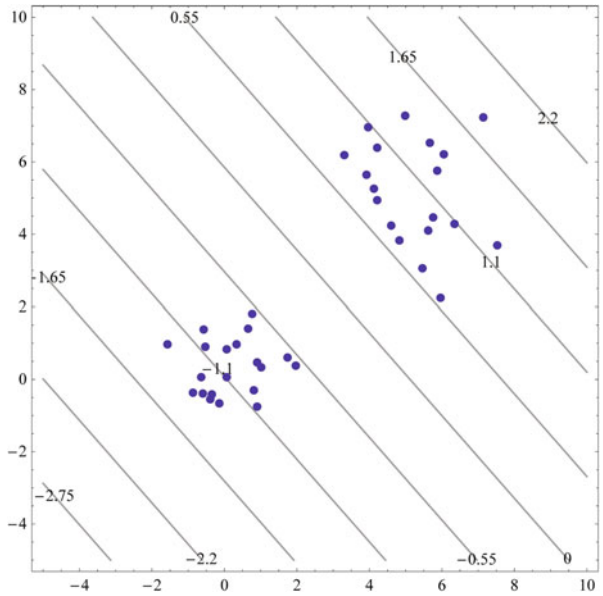


Fig. 10.10 A linear separable problem

Let the regularization parameter

```
c = 0.025;
```

Then employing our function, we get

```
F = SupportVectorClassifier[xn,yn,K,c];
```

Figure 10.10 shows the optimal separation line, the contour line with zero value (0).

10.6 Two Nonlinear Test Problems

10.6.1 Learning a Chess Board

Let us consider a 2×2 size chess board in Fig. 10.11. The training points are generated by uniformly distributed random numbers from the interval $[-1, 1] \times [-1, 1]$, see Fig. 10.11.

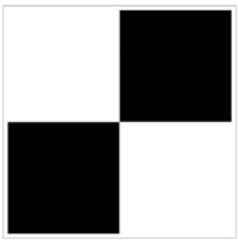


Fig. 10.11 The 2 × 2 chess board problem

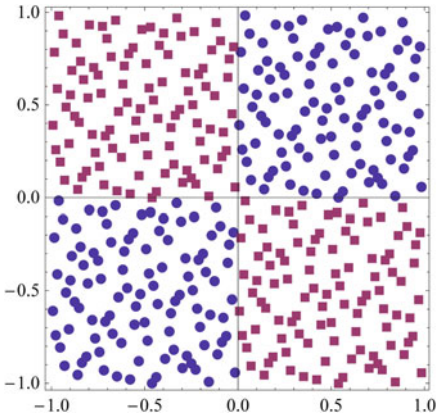


Fig. 10.12 The generated random points of the chess board problem

We create the training set using 100 Halton-points as random sample data points, see Fig. 10.12.
Let us employ the same Gaussian kernel, but now with gain $\beta = 20$.

```
K[u_,v_]:=Exp[-β(u-v).(u-v)]
β=20;
```

using parameter $c = 100$,

```
c=100;
```

Then the solution,

```
F=SupportVectorClassifier[xn,yn,K,c];
```

The contour lines can be seen on Fig. 10.13.

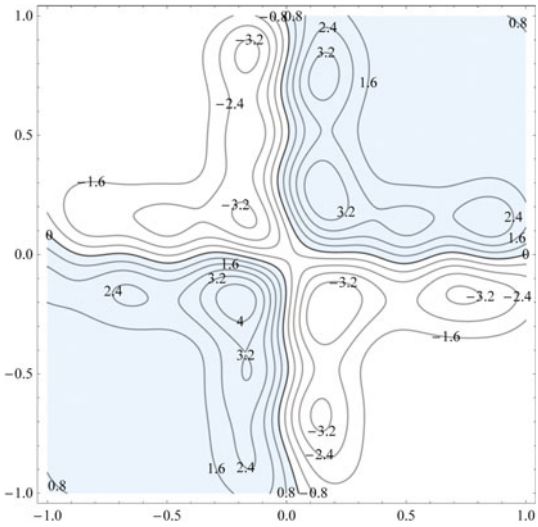


Fig. 10.13 The contour lines of the solution function of the chess board problem

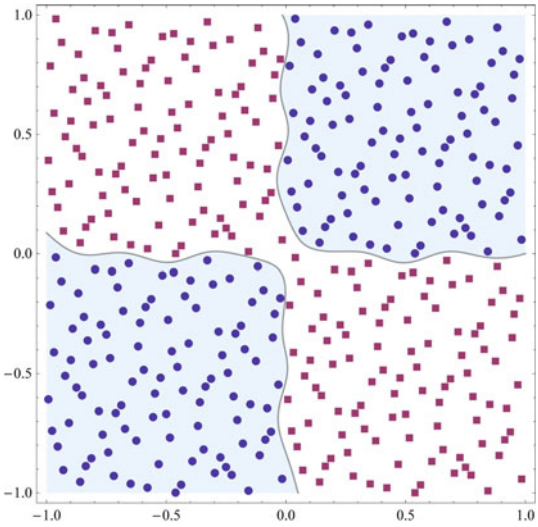


Fig. 10.14 The boundary lines of the clusters of the chess board problem

The zero contour lines represent the boundary of the clusters, see Fig. 10.14. One can get better results by increasing the number of points.

10.6.2 Two Intertwined Spirals

The *two intertwined spirals* is a challenging classification benchmark adopted from the field of neural networks.

Parametric equations of the spirals are

```
x1[t_] := 2 Cos[t] e^(t/10)
y1[t_] := 1.5 Sin[t] e^(t/10)
x2[t_] := 2.7 Cos[t] e^(t/10)
y2[t_] := 2.025 Sin[t] e^(t/10)
```

Generating 26–26 discrete points for each spiral,

$$\texttt{s1} = \texttt{Table}\left[\{\texttt{x1[t]},\texttt{y1[t]}\},\left\{\texttt{t},\pi,3.5\,\pi,\frac{2.5\,\pi}{25}\right\}\right];$$

$$\texttt{s2} = \texttt{Table}\left[\{\texttt{x2[t]},\texttt{y2[t]}\},\left\{\texttt{t},-\frac{\pi}{2},2.5\pi,\frac{3\,\pi}{25}\right\}\right];$$

and displaying these points, one gets the data points for the two classes, see Fig. 10.15.

```
S1 = ListPlot[s1, PlotStyle → {RGBColor[1,0,0], PointSize[0.02]},
  AspectRatio → 1];
S2 = ListPlot[s2, PlotStyle → [0,0,1], PointSize[0.015]},
  AspectRatio → 1];
pspiral = Show[{S1,S2}]
```

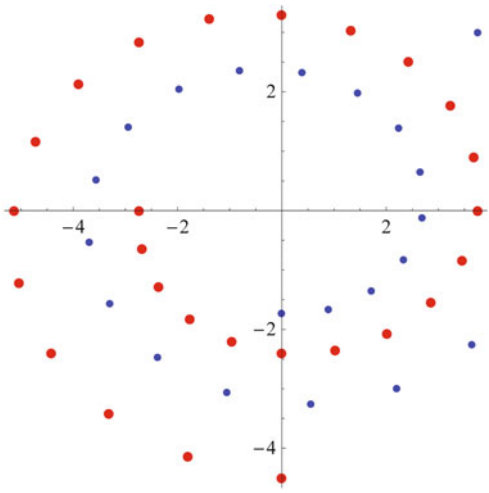


Fig. 10.15 Two intertwined spirals represented by 26 points each cluster

Creating the teaching set, putting these points into one list

```
xym=Join[s1,s2];
```

Generating the labels of the samples,

```
zm=Join[Table[1.,{26}],Table[-1.,{26}]];
```

Applying a wavelet kernel with parameter $a = 1.8$, in case of dimension $n = 2$

```
n=2;a=1.8;
K[u_,v_]:=∏_{i=1}^{n}(Cos[1.75 (u[[i]]-v[[i]])/a] Exp[-(u[[i]]-v[[i]])^2/(2a^2)])
```

and with parameter $c = 100$

```
c=100;
```

the solution is

```
F=SupportVectorClassifier[xym,zm,K,c];
```

Figure 10.16 shows the result of the classification,

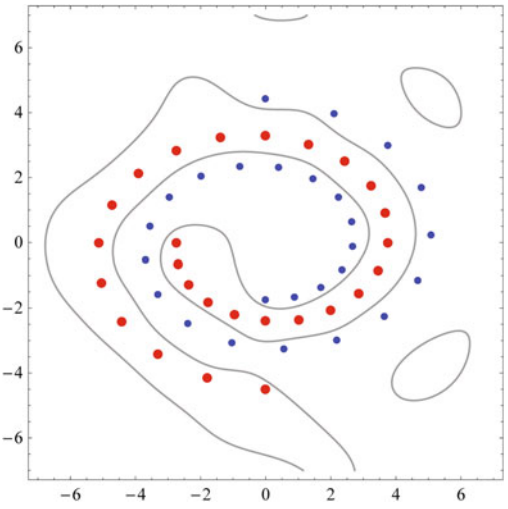


Fig. 10.16 Classification results of SVMC with the nonlinear decision boundary

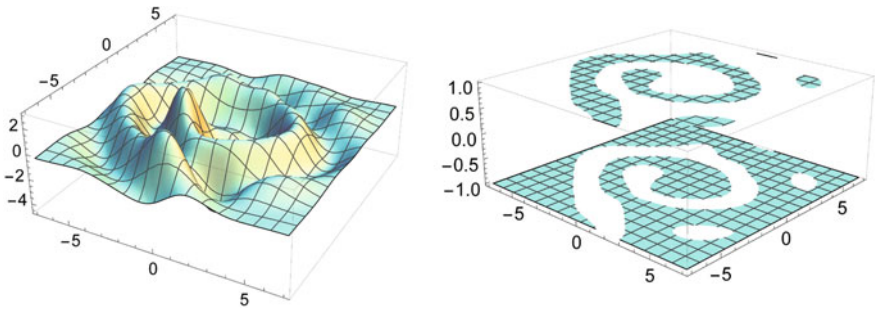


Fig. 10.17 Classification function and the corresponding decision rule using Signum function

The continuous classification function and the decision rule can be displayed in 3D, too (Fig. 10.17).

10.7 Concept of SVM Regression

10.7.1 ε-Insensitive Loss Function

The problem of regression is that of finding a function that approximates a mapping from an input domain to the real numbers based on a training sample. We refer to the difference between the hypothesis output and its training value as the residual of the output, an indication of the accuracy of the fit at this point. We must decide how to measure the importance of this accuracy, as small residuals may be inevitable while we wish to avoid large ones. The loss function determines this measure. Each choice of loss function will result in a different overall strategy for performing regression. For example, least square regression uses the sum of the squares of the residuals.

Although several different approaches are possible, we will provide an analysis for generalization of regression by introducing a threshold test accuracy ε, beyond which we consider a mistake to have been made. We therefore aim to provide a bound on the probability that a randomly drawn test point will have accuracy less than ε.

The linear ε-insensitive loss function $L^{\varepsilon}(x, y, f)$ is defined by

$$L^{\varepsilon}(x,y,f) = (|y - f(x)|)_{\varepsilon} = \max(0, |y - f(x)| - \varepsilon),$$

where f is a real-valued function on a domain X, $x \in X$ and $y \in \mathbb{R}$, see Cristianini and Taylor (2000). Similarly the quadratic ε-insensitive loss is given by

$$L^{\varepsilon}(x,y,f) = (|y - f(x)|)_{\varepsilon}^2.$$

This loss function determines how much a deviation from the true $f(x)$ is penalized; for deviations less than ε, no penalty is incurred. Here is what the loss function looks like, see Fig. 10.18.

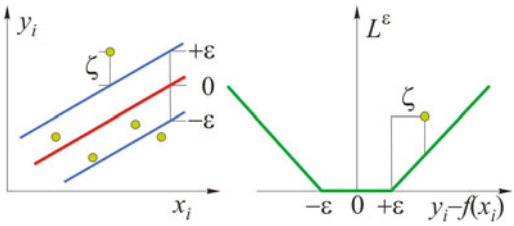


Fig. 10.18 ϵ-insensitivity loss function

10.7.2 Concept of the Support Vector Machine Regression (SVMR)

Let us suppose that there are data pairs, $\{(x_1, y_1), \ldots, (x_k, y_k) \subset X \times \mathbb{R}$, where $x \in X \equiv \mathbb{R}^n$, namely the input are n dimensional vectors, while the output are scalar values. Vapnik (1995) introduced the definition of the ε-SV regression: we are looking for the f(x) function, which has a maximum ε actual deviation from a y_i value, namely $|y_i - f(x_i)| \leq \varepsilon$, for every i-index and at the same time it is as flat as possible. This technique tries to minimize the error, but at the same time to avoids over learning. First, let us consider a linear function

$$f(x) = \langle w, x \rangle + b,$$

where $w \in X$ and $b \in \mathbb{R}$. The gradient of this function should be minimized to avoid over learning

$$\frac{1}{2} w^2 \rightarrow \min .$$

Assuming in a tube with radius of the function is $f(x_i) = 0$,

$$y_i - \langle w, x_i \rangle - b \leq \varepsilon,$$

$$b + \langle w, x_i \rangle - y_i \leq \varepsilon,$$

the error ζ (see Fig. 10.18) also should be minimized. Therefore our objective is

$$c \sum_{i=1}^{n} \zeta_i + \frac{1}{2} w^2 \rightarrow \min$$

where c is a regularization constant.

Since

$$y_i = \zeta_i + \varepsilon + f(x_i),$$

similarly

$$y_i + \zeta_i + \varepsilon = f(x_i).$$

Therefore

$$y_i - f(x_i) = \zeta_i + \varepsilon,$$

and

$$\zeta_i + \varepsilon = f(x_i) - y_i,$$

or

$$|y_i - f(x_i)| = \zeta_i + \varepsilon.$$

Considering the ε-insensitivity function,

$$L^{\in}(x, y, f) = \max(0, |y - f(x)| - \varepsilon).$$

The objective to be minimized is

$$G(\alpha) = c \sum_{i=1}^{n} L^{\varepsilon}\left(x_j, y_j, f\left(x_j\right)\right) + \frac{1}{2} w^2 \rightarrow \min.$$

As we did in case of SVMC, the weight vector will be represented as

$$w = \sum_{i=1}^{n} \alpha_i x_i.$$

Introducing a nonlinear kernel, our function can be generalized for nonlinear case, too

$$f\left(x_j\right) = \sum_{i=1}^{n} \alpha_i \langle x_j, x_i \rangle + b \rightarrow f\left(x_j\right) = \sum_{i=1}^{n} \alpha_i K\left(x_j, x_i\right) + b.$$

The dual form of the minimization problem, which should be maximized is

$$W(\alpha) = \sum_{i=1}^{n} y_i \alpha_i - \varepsilon \sum_{i=1}^{n} |\alpha_i| - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j \left(K\left(x_i, x_j\right) + \frac{1}{c} \delta_{ij} \right) \rightarrow \max,$$

$$\sum_{i=1}^{n} \alpha_i = 0 \quad \text{and} \quad -c < \alpha_i \leq c \quad i = 1, 2, \ldots n,$$

Using the α_i as the solution, the approximating function is,

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x) + b.$$

Similarly to the classification problem, we call the input vectors x_i support vectors if the corresponding $\alpha_i \neq 0$.

The parameter b should be chosen such that for $\alpha_i \neq 0$, we have

$$f(x_i) = y_i - \varepsilon - \frac{\alpha_i}{c}, \quad i = 1, 2, \ldots n.$$

Considering that,

$$f(x_j) = \sum_{i=1}^{n} \alpha_i K(x_i, x_j) + b,$$

then substituting and expressing b, we get

$$b = y_{\mathrm{j}} - \varepsilon - \frac{\alpha_j}{c} - \sum_{i=1}^{n} \alpha_i K(x_i, x_j), \quad j = 1, 2, \ldots n.$$

In practice, one may compute b as a mean,

$$b = \frac{1}{n} \sum_{j=1}^{n} \left(y_{\mathrm{j}} - \varepsilon - \frac{\alpha_j}{c} - \sum_{i=1}^{n} \alpha_i K(x_i, x_j) \right).$$

10.7.3 The Algorithm of the SVMR

The algorithm of the support vector regression will be illustrated via the following simple example. Let us consider the following data pairs, see Fig. 10.19.

```
xn = {1., 3., 4., 5.6, 7.8, 10.2, 11., 11.5, 12.7};
yn = {-1.6, -1.8, -1., 1.2, 2.2, 6.8, 10., 10., 10.};
```

First, we employ a linear kernel,

```
K[u_, v_] := u.v
```

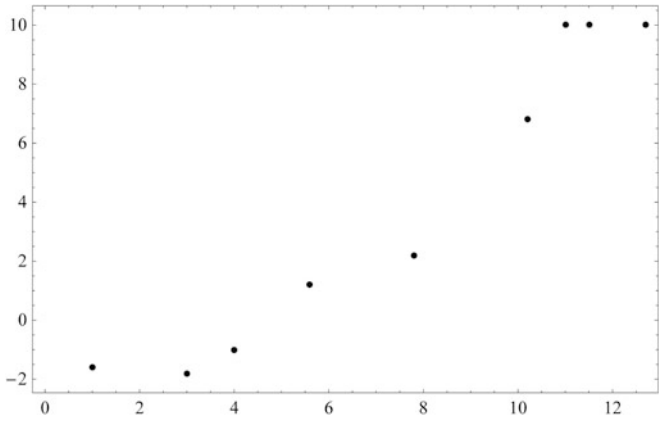


Fig. 10.19 Data pairs of the regression problem

The parameters are

```
ε = 0.25; c = 0.1;
```

The input and output vectors can be given as

```
xm = Partition[xn,1];
ym = yn;
```

The number of the measurements

```
m = Length[xm]
9
```

The kernel matrix is

```
M = Table[K[xm[[i]], xm[[j]]],{i,1,m},{j,1,m}] +
  1/cIdentityMatrix[m];
```

Then the objective function to be maximized is

$$\mathtt{W} = \sum_{\mathtt{i}=1}^{\mathtt{m}} \alpha_{\mathtt{i}} \mathtt{ym[[i]]} - \varepsilon \sum_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{Abs}[\alpha_{\mathtt{i}}] - \frac{1}{2} \sum_{\mathtt{i}=1}^{\mathtt{m}} \sum_{\mathtt{j}=1}^{\mathtt{m}} (\alpha_{\mathtt{i}} \alpha_{\mathtt{j}} \mathtt{M[[i,j]]});$$

and the constrains

$$g = \text{Apply}\left[\text{And}, \text{Join}\left[\text{Table}[-c < \alpha_i \le c, \{i,1,m\}], \left\{\left(\sum_{i=1}^{m} \alpha_i == 0\right)\right\}\right]\right];$$

The variables of the optimization problem

$$\text{vars} = \text{Table}[\alpha_i], \{i,1,m\}];$$

Let us compute the optimal α's using global maximization method

$$\text{sol} = \text{NMaximize}[\{W,g\}, \text{vars}, \text{Method} \to \text{DifferentialEvolution}][[2]];$$

The parameter b can be computed as a mean,

$$b = \frac{1}{m}\text{Apply}\left[\text{Plus}, \text{Table}\left[\left(\text{ym}[[j]] - \sum_{i=1}^{m} \alpha_i K[\text{xm}[[i]], \text{xm}[[j]]] - \varepsilon - \frac{\alpha_j}{c}\right)/.\text{sol}, \{j,1,m\}\right]\right]$$

Then the analytical form of the regression line is

$$F = \left(\sum_{i=1}^{m} \alpha_i K[\text{xm}[[i]], \text{Table}[x_j, \{j,1,n\}]] + b\right)/.\text{sol}$$

$$-4.05732 + 1.04889\, x_1$$

Figure 10.20 shows the linear SVMR approximation of the regression problem. The SVMR algorithm can be summarize and implemented as the following *Mathematica* function,

```
SupportVectorRegression[{xm_,ym_},K_,ε_,c_]:=Module[
 {m,n,M,i,j,W,g,vars,sol,b},
 m=Length[ym];n=Length[xm[[1]]];
 M=Table[K[xm[[i]],xm[[j]]],{i,1,m},{j,1,m}] +
   1/c IdentityMatrix[m];
```

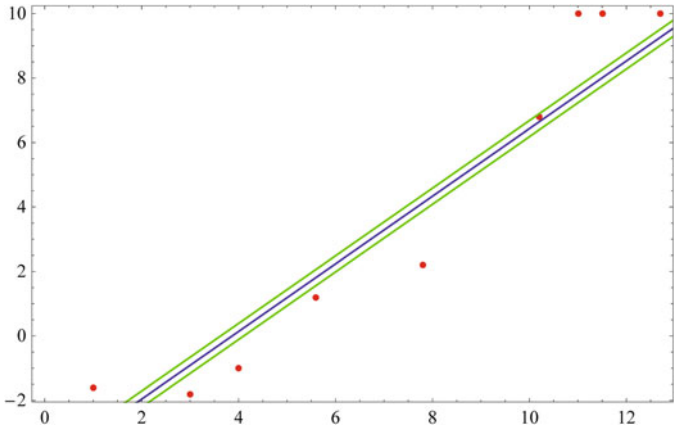


Fig. 10.20 The linear SVMR approximation of the regression problem

$$W = \sum_{i=1}^{m} \alpha_i ym[[i]] - \varepsilon \sum_{i=1}^{m} Abs[\alpha_i] - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} (\alpha_i \alpha_j M[[i,j]]); n$$

$$g = Apply\left[And, Join\left[Table[-c < \alpha_i \leq c, \{i,1,m\}], \left\{\left(\sum_{i=1}^{m} \alpha_i == 0\right)\right\}\right]\right];$$

$$vars = Table[\alpha_i], \{i,1,m\}];$$

$$sol = NMaximize[\{W,g\}, vars, Method \rightarrow DifferentialEvolution][[2]];$$

$$b = 1/m\ Apply[Plus,$$

$$Table\left[\left(ym[[j]] - \sum_{i=1}^{m} \alpha_i K[xm[[i]], xm[[j]]] - \varepsilon - \frac{\alpha_j}{c}\right)/.sol, \{j,1,m\}\right]\Bigg];$$

$$\left\{\left(\sum_{i=1}^{m} \alpha_i K[xm[[i]], Table[x_j, \{j,1,n\}]] + b\right)/.sol, vars/.sol\right\}\Bigg];$$

where the kernel function K, and the parameters ε and c should be given by the user! The data points are $\{xm, ym\}$. The main advantage of this *Mathematica* modul is that it provides the solution function in analytic form.

10.8 Employing Different Kernels

Now we employ nonlinear kernels for the same data set considered above.

10.8.1 Gaussian Kernel

Employing the Gaussian kernel

```
K[u_,v_]:=Exp[-β Norm[(u-v)]];
```

with the parameters,

```
ε=0.3; c=200; β=0.05;
```

we get the following approximation function,

```
xmp=Partition[xm,1];
F=SupportVectorRegression[{xmp,ym},K,ε,c];
F[[1]]
```

$$3.58965 - 1.908315e^{-0.05\text{Abs}[1.-x_1]} - 8.70777e^{-0.05Abs[3.-x_1]} - 2.322437e^{-0.05Abs[4.-x_1]} + 2.13623e^{-0.05Abs[5.6-x_1]} - 11.61337e^{-0.05Abs[7.8-x_1]} - 10.8503e^{-0.05Abs[10.2-x_1]} + 27.6728e^{-0.05Abs[11.-x_1]} + 2.63874e^{-0.05Abs[11.5-x_1]} + 2.95439e^{-0.05Abs[12.7-x_1]}$$

Figure 10.21 shows the result,

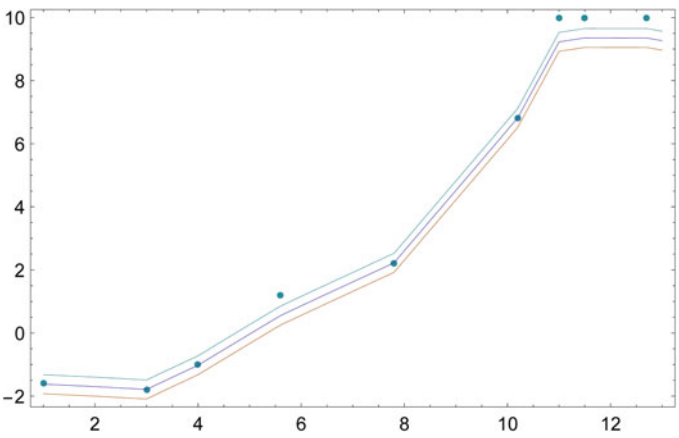


Fig. 10.21 SVM regression with Gaussian kernel

10.8.2 Polynomial Kernel

The general form of the polynomial kernel is

$$K(\mathrm{u},\ \mathrm{v}) = (c + \langle u, v\rangle)^d,$$

where $d > 0$, is the order of the kernel. In our case let $d = 3$,

```
d = 3;
```

and the kernel

```
K[u_, v_] := (a + u.v)^3
```

with the parameters

```
F = SupportVectorRegression[{xm, ym}, K, ε, c];
F[[1]]
-1.09704 - 45.3558 (0.01 + 1.0x1)^3 -
63.9509 (0.01 + 3.0x1)^3 + 144.114 (0.01 + 4.0x1)^3 +
88.8897 (0.01 + 5.6x1)^3 -
199.513 (0.01 + 7.8x1)^3 - 50.0423 (0.01 + 10.2x1)^3 +
123.831 (0.01 + 11.0x1)^3 +
83.8837 (0.01 + 11.5x1)^3 - 81.856 (0.01 + 12.7x1)^3
```

Figure 10.22 shows the third order SVMR function.

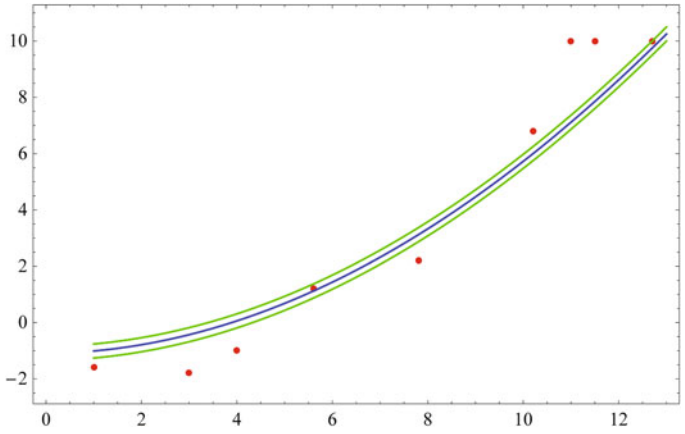


Fig. 10.22 SVM regression with polynomial kernel of order three

10.8.3 Wavelet Kernel

Let us employ the wavelet kernel, which is more flexible

$$K[u\_,v\_] := \prod_{i=1}^{n}\left(\text{Cos}\left[1.75\frac{u[[i]]-v[[i]]}{a}\right]\text{Exp}\left[-\frac{(u[[i]]-v[[i]])^2}{2a^2}\right]\right)$$

With the wavelet parameters

```
n = 1; a = 4.;
```

The parameter of the SVMR are (Fig. 10.23).

```
ε = 0.3; c = 200.;
```

Then the analytical solution is

```
F = SupportVectorRegression[{xmp,ym},K,ε,c];
F[[1]]
2.53089 - 3.07833e^-0.03125(1.-x1)^2 cos[0.4375(1.-x1)] -
 9.08084 × 10^-9 e^-0.03125(3.-x1)^2 cos[0.4375(3.-x1)] -
 3.79962e^-0.03125(4.-x1)^2 cos[0.4375(4.-x1)] +
```

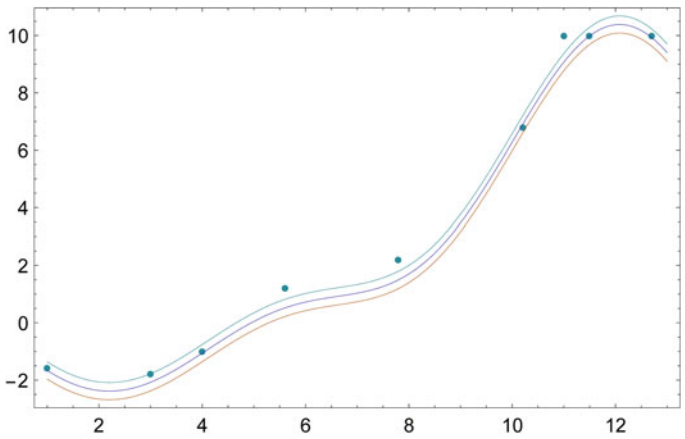


Fig. 10.23 SVM regression with Wavelet kernel, ε = 0.3

```
0.429218e^-0.03125(5.6-x1)^2 cos[0.4375(5.6-x1)] +
4.78458e^-0.03125(7.8-x1)^2 cos[0.4375(7.8-x1)] -
33.4234e^-0.03125(10.2-x1)^2 cos[0.4375(10.2-x1)] +
2.53089-3.07833e^-0.03125(1.-x1)^2 cos[0.4375(1.-x1)] -
9.08084×10^-9 e^-0.03125(3.-x1)^2 cos[0.4375(3.-x1)] -
3.79962e^-0.03125(4.-x1)^2 cos[0.4375(4.-x1)] +
```

Now let us consider a 2D function approximation ($n = 2$) with a wavelet kernel. The function to be approximated employing 5×5 data points is, see Fig. 10.24,

```
z[{x_,y_}]:=(x^2-y^2)Sin[0.5x]
```

The selected data triplets are $(x_i, y_i, f(x_i, y_i))$,

```
data=Flatten[Table[N[{i-10,j-10,z[{i-10,j-10}]}],
  {i,0,20,4},{j,0,20,4}],1];
dataxy=Transpose[Take[Transpose[data],{1,2}]];
dataz=Transpose[Take[Transpose[data],{3,3}]]//Flatten;
```

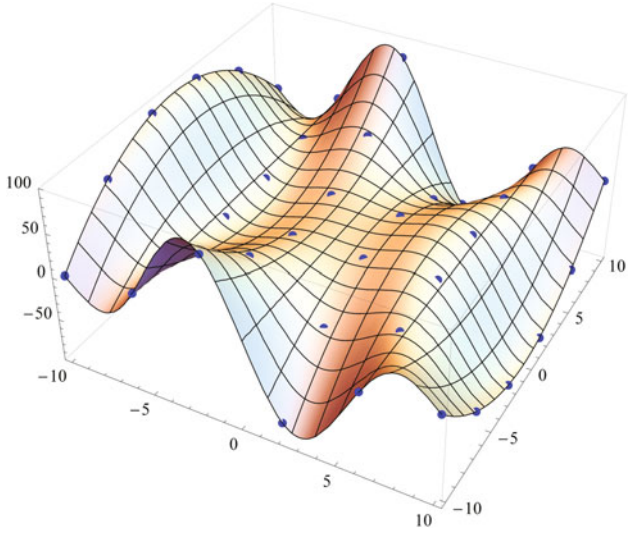


Fig. 10.24 The 2D test function with 25 data points

The wavelet and the SVMR parameters are

```
n = 2; a = 4.0; ε = 0.15; c = 200;
```

The kernel itself is

$$\texttt{K[u\_,v\_]} := \prod_{i=1}^{n}\left(\texttt{Cos}\left[1.75\,\frac{\texttt{u[[i]]}-\texttt{v[[i]]}}{\texttt{a}}\right]\texttt{Exp}\left[-\frac{(\texttt{u[[i]]}-\texttt{v[[i]]})^2}{2\texttt{a}^2}\right]\right)$$

Then the function in short form can be written as

```
F = SupportVectorRegression[{dataxy,dataz},K,ε,c];
Short[F[[1]],10]
```

$$-0.15 + 37.7231e^{-0.03125(-10.-x_1)^2-0.03125(-10.-x_2)^2}\,\texttt{Cos}[0.4375(-10.-x_1)]\texttt{Cos}[0.4375(-10.-x_2)] + 10.8829 \ll 3 \gg + \ll 52 \gg$$

Figure 10.25 shows the SVMR function with the data points,
Comparing the two figures, the approximation is quite reasonable.

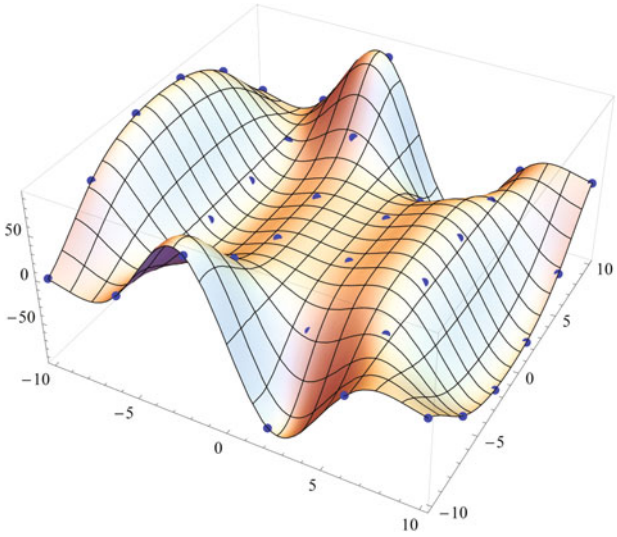


Fig. 10.25 The approximating surface of the SVMR function with the data points

10.8.4 Universal Fourier Kernel

This kernel is valid on the region of $[0, 2\,\pi]$ or in our case $x, y \in [0, 2\,\pi]^2$,

$$\texttt{K[u\_,v\_]} := \prod_{\texttt{i}=1}^{\texttt{n}} \frac{1 - \texttt{q}^2}{2\,(1 - 2\,\texttt{q}\,\texttt{Cos[u[[i]] - v[[i]]]} + \texttt{q}^2)}$$

Let us employ the following parameters

```
q = 0.25; n = 2; ε = 0.0025; c = 200;
```

Our task is to approximate the following function

$$\texttt{z[\{x\_,y\_\}]} := \frac{\texttt{Sin}\left[\sqrt{\texttt{x}^2 + \texttt{y}^2}\,\right]}{\sqrt{\texttt{x}^2 + \texttt{y}^2}}$$

in the region of $[-5, 5] \times [-5, 5]$. In order to employ the kernel, the function should be rescaled,

$$\texttt{z[\{x\_,y\_\}]} := \frac{\texttt{Sin}\left[\sqrt{(\frac{5\texttt{x}}{\pi} - 5)^2 + (\frac{5\texttt{y}}{\pi} - 5)^2}\,\right]}{\sqrt{(\frac{5\texttt{x}}{\pi} - 5)^2 + (\frac{5\texttt{y}}{\pi} - 5)^2}}$$

Our test function seems to be a simple one, however it is not. Therefore, we employ 10×10 data pairs, see Fig. 10.26.

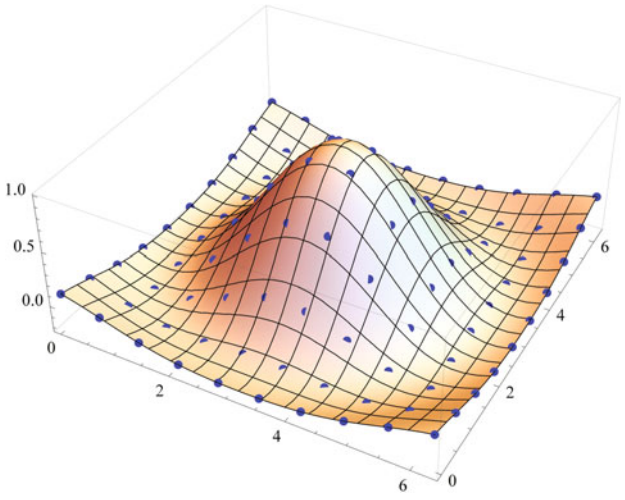


Fig. 10.26 The surface of the test function with the data points

```
data = Flatten[Table[N[{u,v,z[{u,v}]}],{u,0,2π,(2π)/9},
  {v,0,2π,(2π)/9}],1];
dataxy = Transpose[Take[Transpose[data],{1,2}]];
dataz = Transpose[Take[Transpose[data],{3,3}]]//Flatten;
```

Employing the Fourier kernel, the SVMR approximation function in short form is

```
F = SupportVectorRegression[{dataxy,dataz},K,ε,c];
Short[F[[1]],15]
```

$$0.0253295 - \frac{0.105729}{(1.0625 - 0.5\cos[0.0 - x_1])(1.0625 - 0.5\cos[0.0 - x_2])} + \frac{0.151706}{(1.0625 - 0.5\cos[0.698132 - x_1])(1.0625 - 0.5\cos[0.0 - x_2])} + \ll 150 \gg + \frac{0.14956}{(1.0625 - 0.5\cos[5.58505 - x_1])(1.0625 - 0.5\cos[6.28319 - x_2])} - \frac{0.107952}{(1.0625 - 0.5\cos[6.28319 - x_1])(1.0625 - 0.5\cos[6.28319 - x_2])}$$

Figure 10.27 shows the approximating function with the data points.
The comparison of the two figures indicates a fairly good approximation.

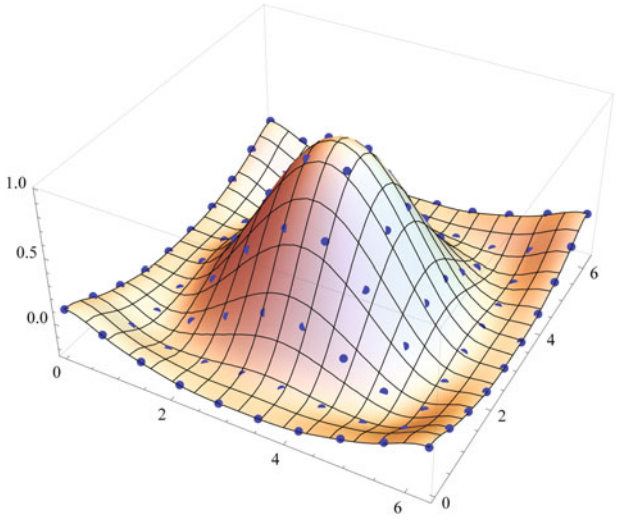


Fig. 10.27 The surface of approximating SVMR function with the data points

10.9 Applications

10.9.1 Image Classification

Let us classify images into two different categories, snowman and dice, see Figs. 10.28 and 10.29.
The feature extraction of these pictures represented by matrices of 128 × 128 pixels was carried out by a wavelet transform using the second order Daubechies filter, then employing averaging technique for seven non-overlapping bands of the spectrum. Consequently the dimension of these feature vectors is seven, $n = 7$.

The feature vectors of snowmen,

```
SnowMan = Import["G:\\Snowman.dat"];
Dimensions[SnowMan]
{10,7}
```

and those of the dices,

```
DiCes = Import["G:\\Dice.dat"];
Dimensions [DiCes]
{10,7}
```

From the 10–10 pictures, the first 7–7 ones are considered as elements of the training set, and the last 3–3 represent the testing set. Then the elements of the training set as input,

```
xym = Join[Take[SnowMan,{1,7}],Take[DiCes,{1,7}]];
Dimensions[xym]
{14,7}
```

Fig. 10.28 Representatives of snowman category

Fig. 10.29 Representatives of dice category

The out value for the snowman class is +1 and for the dice class is −1.

```
zm = {1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1};
```

First, wavelet kernel is employed with parameters

```
n = 7; a = 2.;
```

the kernel,

$$K[u\_,v\_] := \prod_{i=1}^{n}\left(\text{Cos}\left[1.75\,\frac{u[[i]] - v[[i]]}{a}\right]\text{Exp}\left[-\frac{(u[[i]] - v[[i]])^2}{2a^2}\right]\right)$$

and the penalty constant is

```
c = 150;
```

Then the SVMC function is in short form,

```
F = SupportVectorClassifier[xym,zm,K,c];
Short[F[[1]],25]
```

$$-0.0893017 + 0.00002233654\,e^{-0.125(-17.7665+x_1)^2-0.125(-15.3532+x_2)^2-0.125(\ll 1\gg)^2-0.125\ll 1\gg-0.125\ll 1\gg^2 - 0.125(-7.32602+\ll 1\gg)^2-0.125(-5.1615+x_7)^2}$$

$$\cos[0.875(-17.7665+x_1)]\cos[0.875(-15.3532+x_2)]\ll 1\gg$$
$$\cos[0.875(-\ll 19\gg+\ll 1\gg)]\cos[0.875(-9.49168+x_5)]$$
$$\cos[0.875(-7.32602+x_6)]\cos[0.875(-5.1615+x_7)]+\ll 20\gg$$

Let us define a function for the picture classification as

```
PictureClassifier[u_]:=Sign[F[[1]]]/.{x1 → u[[1]],x2 → u[[2]],
 x3 → u[[3]],x4 → u[[4]],x5 → u[[5]],x6 → u[[6]],x7 → u[[7]]}
```

and do employ it for the pictures of the testing set,

```
Test = Join[Take[SnowMan,{8,10}],Take[DiCes,{8,10}]];
Map[PictureClassifier[#]&,Test]
{1,1,1,-1,-1,-1}
```

The result is correct.

In addition, we try another kernel, *Kernel with Moderate Decreasing* (KMOD), which especially effective in case of classification problem,

```
K[u_,v_]:=Exp[γ/(Norm[u-v]^2 + σ^2)]-1
```

Now the parameters are

```
γ=0.5; σ=3;
```

Then the SVMC function is,

```
F=SupportVectorClassifier[xym,zm,K,c];
Short[F[[1]],10]
```

```
0.0191626 + 13.1271(-1 + e^(0.5/(9 + Abs[≪1≫]^2 + ≪1≫^2 + ≪2≫ + ≪1≫ + Abs[≪1≫]^2 + Abs[-≪18≫ + x7]^2))) +
39.3118(-1 + e^(0.5/(9 + ≪6≫ + ≪1≫^2))) - 14.9837(-1 + ≪1≫) +
≪12≫ + 17.038(-1 + e^(0.5/≪1≫)) -
2.30283 × 10^-8 (-1 + e^(0.5/(9 + ≪1≫^2 + ≪4≫ + ≪1≫^2 + Abs[≪1≫]^2))) - 37.6952
(-1 + e^(0.5/(9 + Abs[-≪18≫ + x1]^2 + Abs=≪1≫≪1≫]^2 + ≪1≫≪1≫≪1≫ + ≪1≫^2 + Abs[≪1≫]^2 + Abs[-4.32022 + x7]^2)))
```

which works perfectly on the testing set, too

```
Map[PictureClassifier[#]&,Test]
{1,1,1,-1,-1,-1}
```

10.9.2 Maximum Flooding Level

Let us forecast the maximum flooding level of the river Danube (z) at Budapest, Hungary, on the bases of the earlier statistical values, namely the level of the Danube at Budapest just before the flooding wave (x) and the average of the rainfall levels measured at the meteorological stations in the water collection region of the Danube in Austria (y), see Table 10.4. Let us employ the following kernel

Table 10.4 Data for the flooding level problem

x[mm]	y[cm]	z[cm]
58	405	590
52	450	660
133	350	780
179	285	770
98	330	710
72	400	640
72	550	670
43	480	520
62	450	660
67	610	690
64	380	500
33	460	460
57	425	610
62	560	710
54	420	620
48	620	660
86	390	620
74	350	590
95	570	740
44	710	730
77	580	720
46	700	640
123	560	805
62	430	673

```
K[u_,v_]:=Exp[-β Norm[(u-v)]];
```

with parameters

```
ε=10.; c=500; β=0.05;
```

Then the approximating function is

```
F=SupportVectorClassifier[xymT,zmT,K,ε,c];
```

$$658.525 + 90.5948\, e^{-0.05\sqrt{Abs[179-x_1]^2+Abs[285-x_2]^2}} -$$
$$106.158\, e^{-0.05\sqrt{Abs[133-x_1]^2+Abs[350-x_2]^2}} -$$
$$155.401\, e^{-0.05\sqrt{Abs[64-x_1]^2+Abs[380-x_2]^2}} +$$

$$
\begin{aligned}
&20.4979\, e^{-0.05\sqrt{\text{Abs}[72-x_1]^2+\text{Abs}[400-x_2]^2}} - \\
&27.0761\, e^{-0.05\sqrt{\text{Abs}[54-x_1]^2+\text{Abs}[420-x_2]^2}} + \\
&26.483\, e^{-0.05\sqrt{\text{Abs}[62-x_1]^2+\text{Abs}[430-x_2]^2}} + \\
&66.2043\, e^{-0.05\sqrt{\text{Abs}[52-x_1]^2+\text{Abs}[450-x_2]^2}} - \\
&189.417\, e^{-0.05\sqrt{\text{Abs}[33-x_1]^2+\text{Abs}[460-x_2]^2}} - \\
&89.7719\, e^{-0.05\sqrt{\text{Abs}[43-x_1]^2+\text{Abs}[480-x_2]^2}} - \\
&15.9142\, e^{-0.05\sqrt{\text{Abs}[72-x_1]^2+\text{Abs}[550-x_2]^2}} + \\
&30.5367\, e^{-0.05\sqrt{\text{Abs}[62-x_1]^2+\text{Abs}[560-x_2]^2}} + \\
&119.777\, e^{-0.05\sqrt{\text{Abs}[123-x_1]^2+\text{Abs}[560-x_2]^2}} + \\
&33.7208\, e^{-0.05\sqrt{\text{Abs}[95-x_1]^2+\text{Abs}[570-x_2]^2}} + \\
&6.24451\, e^{-0.05\sqrt{\text{Abs}[67-x_1]^2+\text{Abs}[610-x_2]^2}} - \\
&2.18785\, e^{-0.05\sqrt{\text{Abs}[48-x_1]^2+\text{Abs}[620-x_2]^2}} - \\
&76.3213\, e^{-0.05\sqrt{\text{Abs}[46-x_1]^2+\text{Abs}[700-x_2]^2}} + \\
&98.1576\, e^{-0.05\sqrt{\text{Abs}[44-x_1]^2+\text{Abs}[710-x_2]^2}}
\end{aligned}
$$

Figure 10.30 shows the SVMR function with the measured data points. The error of the approximation is quite reasonable; see Fig. 10.31 which also shows the region of the validity of the approximation.

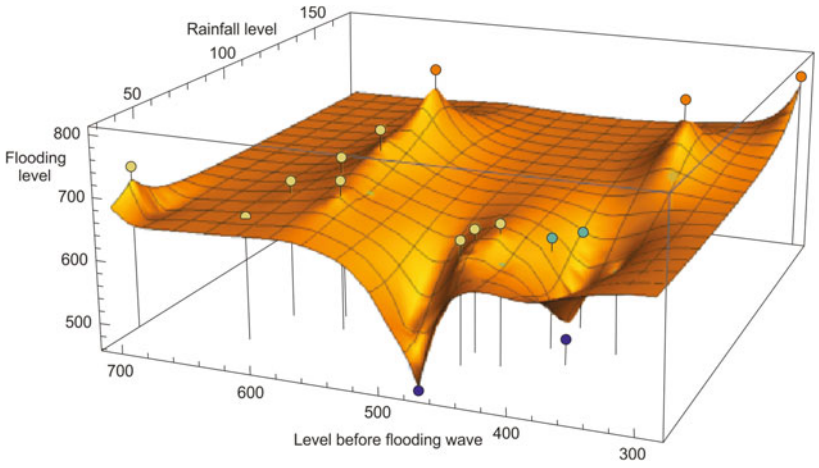


Fig. 10.30 The SVMR approximation function and the measured values

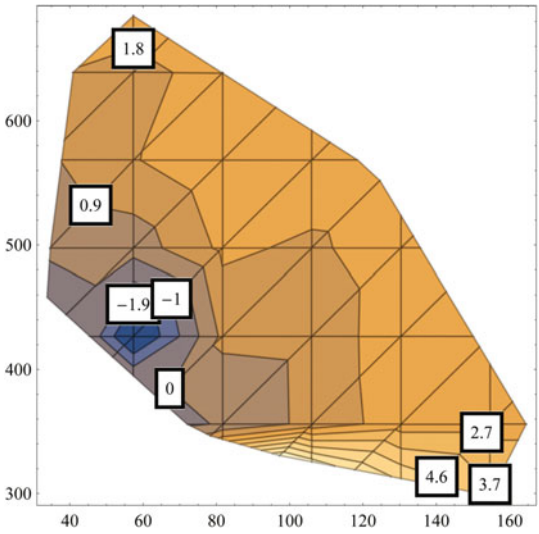


Fig. 10.31 The error in % of the SVMR approximation function in the region of the input, rainfall level versus level of the Danube before the flood wave

10.10 Exercise

10.10.1 Noise Filtration

Let us generate synthetic noisy data of the *sinc(x)* function on the interval [−10, 10]. The noise has a normal distribution $N(0, 0.1)$, see Fig. 10.32.

```
SeedRandom[2];
data = Table[{x, Sinc[x] + 2Random[NormalDistribution[0, 0.1]]},
{x, -10, 10, 0.25}];
```

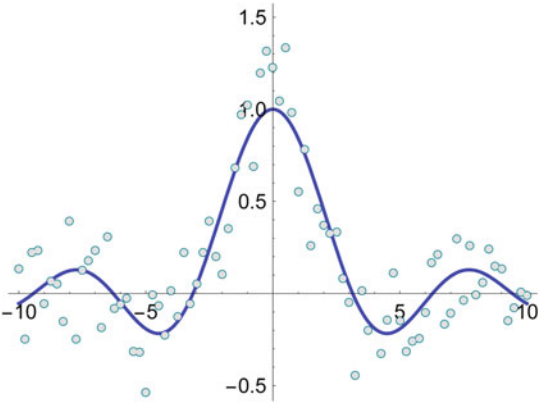


Fig. 10.32 Noisy data of the sinc(x) function and the function itself

Our task is to represent the *sinc(x)* function employing the noisy data points. Let us employ wavelet kernel with the parameters

```
n = 1; a = 4.;
K[u_, v_] := Product_{i=1}^{n} (Cos[1.75 (u[[i]] - v[[i]])/a] Exp[-(u[[i]] - v[[i]])^2/(2 a^2)])
```

The parameters for the SVMR approximation are

```
ε = 0.02; c = 200.;
```

Then the solution is

```
F = SupportVectorRegression[, ym}, K, ε, c]
0.124549 + 18.4605 e^(-0.03125 (-10. - x1)^2) cos[0.4375 (-10.0 - x1)] -
 51.5907 e^(-0.03125 (-9.75 - x1)^2) cos[0.4375 (-9.75 - x1)] +
 32.0491 e^(-0.03125 (-9.5 - x1)^2) cos[0.4375 (-9.5 - x1)] + ≪107≫ +
 9.87568 e^(-0.03125 (8.75 - x1)^2) cos[0.4375 (8.75 - x1)] +
 11.7166 e^(-0.03125 (9. - x1)^2) cos[0.4375 (9. - x1)] -
 31.539 e^(-0.03125 (9.25 - x1)^2) cos[0.4375 (9.25 - x1)] -
 11.2079 e^(-0.03125 (9.5 - x1)^2) cos[0.4375 (9.5 - x1)] +
 4.40217 e^(-0.03125 (9.75 - x1)^2) cos[0.4375 (9.75 - x1)] +
 7.84421 e^(-0.03125 (10. - x1)^2) cos[0.4375 (10. - x1)]
```

Figure 10.33 shows the SVMR approximation with the original *sinc(x)* function.

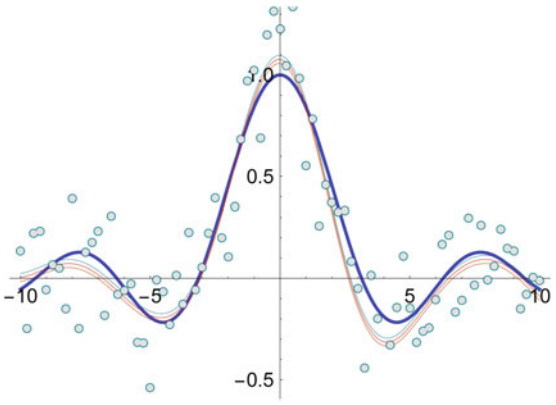


Fig. 10.33 SVM regression with wavelet kernel, ε = 0.02

References

Berthold M, Hand DJ (2003) Intelligent data analysis, an introduction. Springer, Heidelberg, Berlin

Cristianini N, Taylor JS (2000) An introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, Cambridge

Vapnik V (1995) The nature of statistical learning theory. Springer, Berlin, Heidelberg

Chapter 11
Symbolic Regression

11.1 Concept of Symbolic Regression

The symbolic regression SR can be considered as a broad generalization of the class of Generalized Linear Models (GLM), which is a linear combination of basic functions β_i, $i = 1, 2, \ldots, n$ with a dependent variable y, and an independent variable vector $\mathbf{x}$.

$$y(\mathbf{x}) = c_0 + \sum_{i=1}^{n} c_i \beta_i(\mathbf{x}) + \varepsilon,$$

where c_i are the coefficients and ε is the error term.

SR will search for a set of basic functions (building blocks) and coefficients (weights) in order to minimize the error ε when given y and $\mathbf{x}$. The standard basic functions of the coordinates of $\mathbf{x}$ are: constant, addition, subtraction, multiplication, division, sine, cosine tangent, exponential, power, square root, etc. To select the optimal set of basic functions, Koza (1992) suggested using genetic programming (GP). GP is a biologically inspired machine learning method that evolves computer programs to perform a task. In order to carry out genetic programming, the individuals (competing functions) should be represented by a binary tree. In standard GP, the leaves of the binary tree are called terminal nodes represented by variables and constants, while the other nodes, the so called non-terminal nodes are represented by functions. Let us see a simple example. Suppose for a certain i

$$\beta_i(\mathbf{x}) = x_1 x_2 + \frac{1}{2} x_3.$$

Its binary tree representation can be seen in Fig. 11.1.

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_11

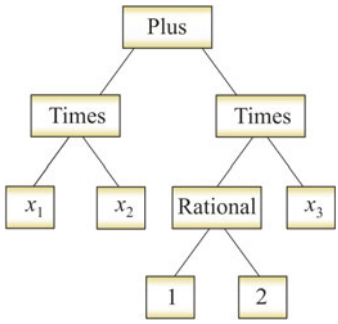


Fig. 11.1 The binary tree representation of a basic function β_i

In this example, there are three variables (x_1, x_2, x_3), two constants (1, 2), and three elementary functions (*plus*, *times*, *rational*). The binary tree of $y(\mathbf{x})$ can be built up from such trees as sub-trees. Mathematically,

$$y(\mathbf{x}) = c_0 + c_1\, tree_1 + c_2\, tree_2 + \cdots.$$

GP randomly generates a population of individuals $y_k(\mathbf{x})$, $k = 1, 2, \ldots, n$ represented by tree structures to find the best performing trees.

There are two important features of the function represented by a binary tree: complexity and fitness. We define complexity as the number of nodes in a binary tree needed to represent the function. The fitness qualifies how good a model $(y = y(\mathbf{x}))$ is. Basically, there are two types of measures used in SR; the root mean squared error (RMSE) and the R-square. The latter returns the square of the Pearson product moment correlation coefficient (R) describing the correlation between the predicted values and the target values. Then the goodness of the model, the fitness function, can be defined as,

$$f = \frac{1}{1 + \mathrm{RMSE}} \quad \text{or} \quad f = \mathrm{R}^2,$$

where of methods to solve the problem $0 \leq f \leq 1$.

GP tries to minimize this error to improve the fitness of the population consisting of individuals (competing functions) from generation to generation by mutation and cross-over procedure. Mutation is an eligible random change in the structure of the binary tree, which is applied to a randomly chosen sub-tree in the individual. This sub-tree is removed from the individual and replaced by a new randomly created sub-tree. This operation leads to a slightly (or even substantially) different basic function. Let us consider the binary tree on Fig. 11.2a, where the sub-tree of y^2 is replaced by $y + x^2$. Then the mutated binary tree can be seen in Fig. 11.2b.

The operation "cross-over" representing sexuality can accelerate the improvement of the fitness of a function more effectively than mutation alone can do. It is a random combination of two different basic functions (parents), based on their fitness, in order to create a new generation of functions, more fit than the original functions. To carry out cross-over, crossing points (non-terminal nodes) in the trees

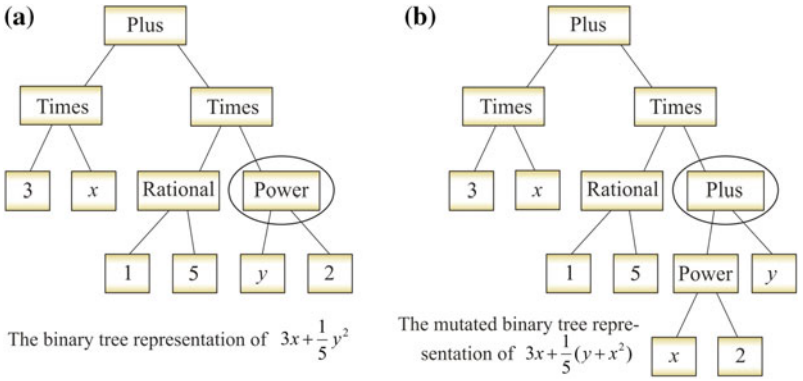


Fig. 11.2 Binary tree representations of the mutation, y^2 in (**a**) is replaced by $y + x^2$ in (**b**)

of both parents should be randomly selected, as can be seen in Fig. 11.3. Then, subtrees belonging to these nodes will be exchanged creating offspring. Let us consider the parents before the cross-over. The first parent $(x - y)/3$, with its crossing point (x) is shown in Fig. 11.3a, the second parent $3x + y^2/5$ with its crossing point (see the sub-tree of y^2) is presented in Fig. 11.3b.

The children produced by the cross-over are given on Fig. 11.4. The first child $(y^2 - y)/3$ is shown in Fig. 11.4a while the second child $(16/5)x$ is shown in Fig. 11.4b.

The generalization of GP was invented by Cramer (1985) and further developed by Koza (1992). GP is a class of evolutionary algorithms working on executable tree structures (parse trees). Koza (1992) showed that GP is capable of doing symbolic regression (or function identification) by generating mathematical expressions approximating a given sample set very closely or in some cases even perfectly. Therefore, GP finds the entire approximation model and its (numerical) parameters simultaneously. An important goal in symbolic regression is to get a

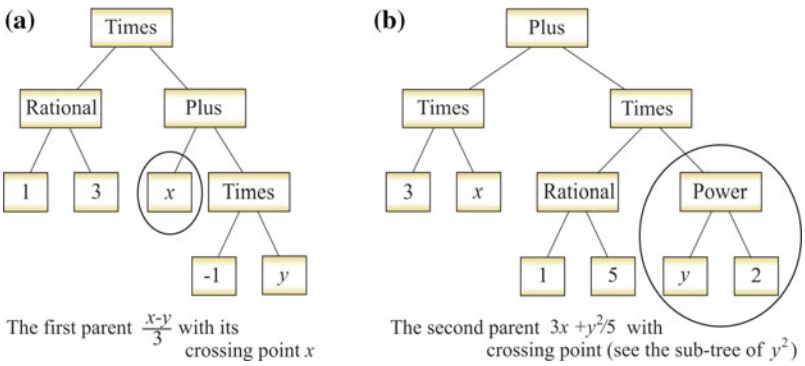


Fig. 11.3 The parents before the crossover

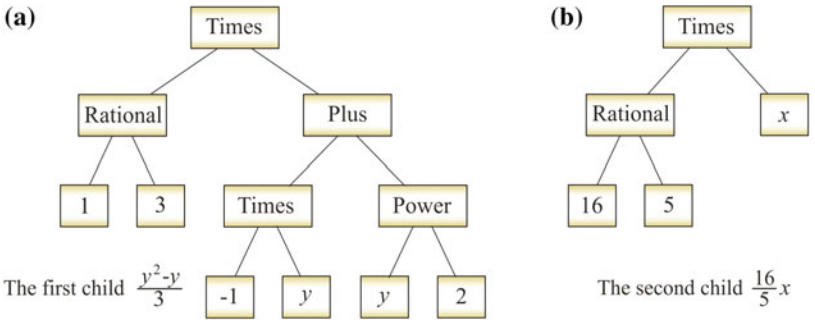


Fig. 11.4 The children produced by the crossover

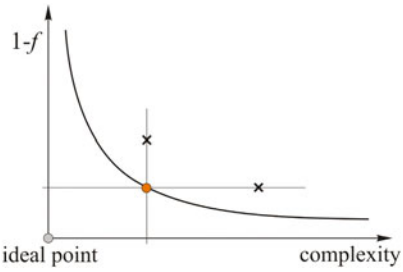
solution, which is numerically robust and does not require high levels of complexity to give accurate output values for given input parameters. Small mean errors may lead to wrong assumptions about the real quality of the expressions found. To be on the safe side, a worst case absolute error should be determined. Sometimes, alternating the worst case absolute error and RMSE or R^2 as the target to be minimized during GP process is the best strategy.

Complexity and fitness are conflicting features leading to a multi-objective problem. A useful expression is both predictive and parsimonious. Some expressions may be more accurate but over-fit the data, whereas others may be more parsimonious but oversimplify. The prediction error versus complexity or 1-fitness versus complexity of the Pareto front represent the optimal solutions as they vary over expression complexity and maximum prediction error. As Fig. 11.5 shows, functions representing the Pareto front have the following features:

- In case of fixed complexity, there is no such solution (function), which could provide less error than the Pareto solution,
- Conversely, in case of fixed error, there is no such solution (function), which would have smaller complexity than the Pareto solution.

The Pareto front tends to contain a cliff where predictive ability jumps rapidly at some minimum complexity. Predictive ability then improves only marginally with more complex expressions. Since the Pareto front provides the set of optimal

Fig. 11.5 The Pareto front



solutions, the user should decide which one is preferable. However, one may select blindly the very solution on the Pareto front, which is closest to the ideal point (zero error, zero complexity).

To carry out an SR computation requires considerable computational power. Fortunately, the algorithm is suited perfectly for parallel computation. There are many software implementations of this method, both commercial and non-commercial. Commercial symbolic regression packages, such as *DTREG* (http://www.dtreg.com/) are designed for predictive modeling and forecasting. Besides symbolic regression, it can also perform other data-mining tasks (e.g., in neural networks, support vectors machines, etc.). *DataModeler* is another well documented commercial symbolic regression package available for *Mathematica* (http://www.evolved-analytics.com). Non-commercial open source symbolic regression packages such as *Eureqa* are available for free download (http://ccsl.mae.cornell.edu/eureqa) as well as *GPLab* and *GPTIPS* toolboxes for *Matlab* (http://gplab.sourceforge.net/) and (http://gptips.sourceforge.net), respectively. In this study, *DataModeler* is used, which has parallel implementation in *Mathematica*. We should mention that in the meantime *DataRobot* has integrated the *Nutonian Inc.* and in this way their product *Eureqa*, too, see, https://www.datarobot.com/.

11.2 Problem of Kepler

The third law of *Kepler* states: "The square of the orbital period of a planet is directly proportional to the cube of the semi-major axis of its orbit (average distance from the Sun)."

$$P^2 \propto a^3,$$

where P is the orbital period of the planet and a is the semi-major axis of the orbit.

For example, suppose planet A is 4 times as far from the Sun as planet B. Then planet A must traverse 4 times the distance of planet B each orbit, and moreover it turns out that planet A travels at half the speed of planet B, in order to maintain equilibrium with the reduced gravitational centripetal force due to being 4 times further from the Sun. In total it takes $4 \times 2 = 8$ times as long for planet A to travel an orbit, in agreement with the law ($8^2 = 4^3$).

The third law currently receives additional attention as it can be used to estimate the distance from an exoplanet to its central star, and help to decide if this distance is inside the habitable zone of that star.

The exact relation, which is the same for both elliptical and circular orbits, is given by the equation.

This third law used to be known as the harmonic law, because Kepler enunciated it in a laborious attempt to determine what he viewed as the "music of the spheres" according to precise laws, and express it in terms of musical notation. His result was

based on the *Rudolphine* table containing the observations of *Tycho Brache* 1605, see Table 11.1.

Where a is give in units of Earth's semi-major axis.

Let us assume that Kepler could have employed one of the function approximation techniques like polynomial regression, artificial neural networks, support vector machine, thin plate spline. Could he find this simple relation with these sophisticated methods?

Let us try different type of methods to solve the problem. The data pairs are,

```
samplePts =
 SetPrecision[{0.39,0.72,1.,1.52,5.2,9.54,19.19,30.06,39.53},10];
observedResponse = SetPrecision
 [{0.24,0.61,1.,1.88,11.86,29.46,84.01,164.79,248.54},10];
data = Transpose[{samplePts, observedResponse}];
```

11.2.1 Polynomial Regression

We compute a third order algebraic polynomial, (Fig. 11.6).

```
P = Fit[data,{1,a,a^2,a^3},a]
-0.903031 + 1.777511a + 0.15698459a^2 - 0.001072739a^3
```

11.2.2 Neural Network

Let us employ an RBF network with a single neuron,

Table 11.1 Normalized observation planetary data

Planet	Period P [yr]	Semimajor axis a
Mercury	0.24	0.39
Venus	0.61	0.72
Earth	1.00	1.00
Mars	1.88	1.52
Jupiter	11.86	5.20
Saturn	29.46	9.54
Uranus	84.01	19.19
Neptune	164.79	3.06
Pluto	284.54	39.53

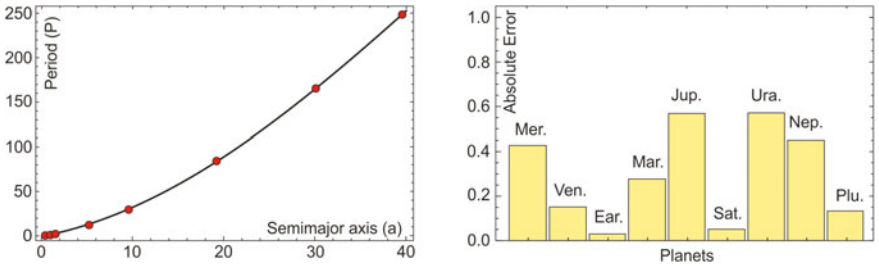


Fig. 11.6 Approximation via algebraic polynomial regression

```
<< NeuralNetworks'
rbf = InitializeRBFNet[samplePts, observedResponse, 1]
RBFNet[{{w1, λ, w2}, χ}, {Neuron → Exp, FixedParameters → None,
 AccumulatedIterations → 0, CreationDate →
  {2017, 1, 14, 17, 9, 9.892977}, OutputNonlinearity →
None, NumberOfInputs → 1}]{rbf2, fitrecord} = NeuralFit[rbf,
samplePts, observedResponse, 100];
```

Figure 11.7 shows the error of the Network during the iteration.
The neural network has the following form

$$\{P\} = rbf2[\{a\}]$$

$$P = 50302.7 - 163.097\,a - 68962.5\,e^{-0.00000851191(192.52+a)^2}$$

Figure 11.8 shows the error of the different planet.

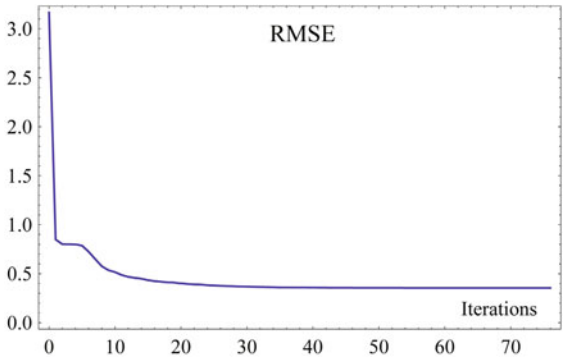


Fig. 11.7 Training process of RBF neural network with a single neuron

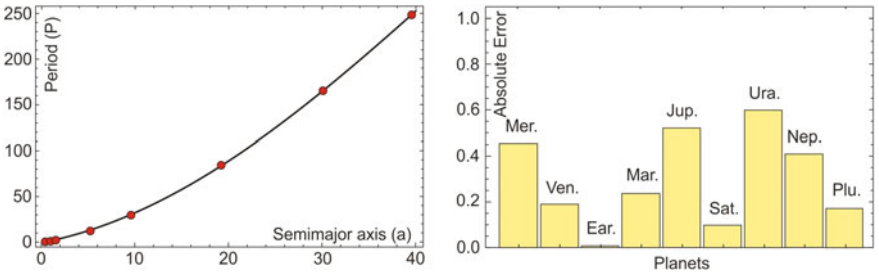


Fig. 11.8 Approximation via RBF neural network with a single neuron

11.2.3 Support Vector Machine Regression

Our kernel function is a wavelet kernel within the following parameters,

```
n = 1; b = 25.;
```

$$\texttt{K[u\_,v\_]:=}\prod_{\texttt{i=1}}^{\texttt{n}}\left(\texttt{Cos}\left[1.75\,\frac{\texttt{u[[i]]}-\texttt{v[[i]]}}{\texttt{b}}\right]\texttt{Exp}\left[-\frac{(\texttt{u[[i]]}-\texttt{v[[i]]})^2}{2\texttt{b}^2}\right]\right)$$

Parameter values for the SVMR are,

```
ε = 0.025; c = 500.; tp = Partition[samplePts, 1];
sol = SupportVectorRegression[{tp, observedResponse}, K, ε, c];
```

The analytic form of SVMR is

```
P = sol[[1]]/.x1 → a
126.361 - 61.1375 e^-0.0008(0.39-a)^2 cos[0.07(0.39 - a)]
 - 60.4577 e^-0.0008(0.72-a)^2 cos[0.07(0.72 - a)]
 - 47.5422 e^-0.0008(1.0-a)^2 cos[0.07(1.0 - a)]
 - 6.7781 e^-0.0008(1.52-a)^2 cos[0.07(1.52 - a)]
 + 113.053 e^-0.0008(5.2-a)^2 cos[0.07(5.2 - a)]
 - 167.045 e^-0.0008(9.54-a)^2 cos[0.07(9.54 - a)]
 + 317.417 e^-0.0008(19.19-a)^2 cos[0.07(19.19 - a)]
 - 446.980 e^-0.0008(3.06-a)^2 cos[0.07(3.06 - a)]
 + 359.47 e^-0.0008(39.53-a)^2 cos[0.07(39.53 - a)]
```

Figure 11.9 shows the SVMR approach and its error in case of the different planets.

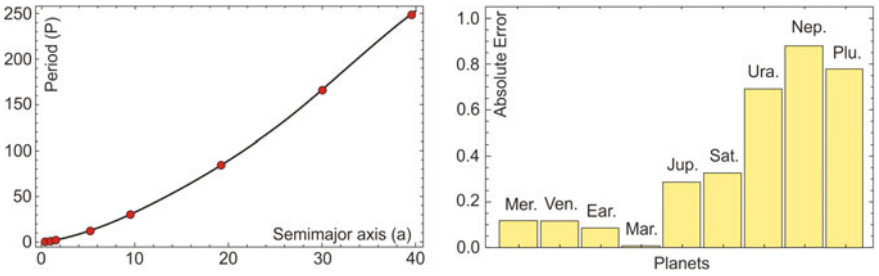


Fig. 11.9 Approximation via SVMR employing wavelet kernel

11.2.4 RBF Interpolation

Now our interpolation function is $r^2 \log(r)$ which is the *Thin-Plate Spline* interpolation function

```
Needs["Imtek`Interpolation`"]
polyharmonic =
  Function[x,xi,If[x ≠ xi,Norm[x - xi,2]^2 Log[Norm[x - xi,2]],0]]
Function[x,xi,If[x ≠ xi,Norm[x - xi,2]^2 Log[Norm[x - xi,2]],0]]

interpolationFunction =
imsUnstructuredInterpolation[data,polyharmonic];
```

The Thin-Plate Spline (TPS) function is

```
P = interpolationFunction[a]
-31.9484529
  + 0.82232083 If[a ≠ 0.39,Norm[a - 0.39,2]^2 Log[Norm[a - 0.39,2]],0]
  - 0.59163242 If[a ≠ 0.72,Norm[a - 0.72,2]^2 Log[Norm[a - 0.72,2]],0]
  - 0.006 If[a ≠ 1.0,Norm[a - 1.0,2]^2 Log[Norm[a - 1.0,2]],0]
  - 0.149 If[a ≠ 1.52,Norm[a - 1.52,2]^2 Log[Norm[a - 1.52,2]],0]
  - 0.04578 If[a ≠ 5.2,Norm[a - 5.2,2]^2 Log[Norm[a - 5.2,2]],0]
  - 0.0178870114 If[a ≠ 9.54,Norm[a - 9.54,2]^2 Log[Norm[a - 9.54,2]],0]
  - 0.0114 If[a ≠ 19.19,Norm[a - 19.19,2]^2 Log[Norm[a - 19.19,2]],0]
  + 0.02 If[a ≠ 30.06,Norm[a - 30.06,2]^2 Log[Norm[a - 30.06,2]],0]
 0.0194 If[a ≠ 39.53,Norm[a - 39.53,2]^2 Log[Norm[a - 39.53,2]],0]
```

The approximation can be seen in Fig. 11.10.

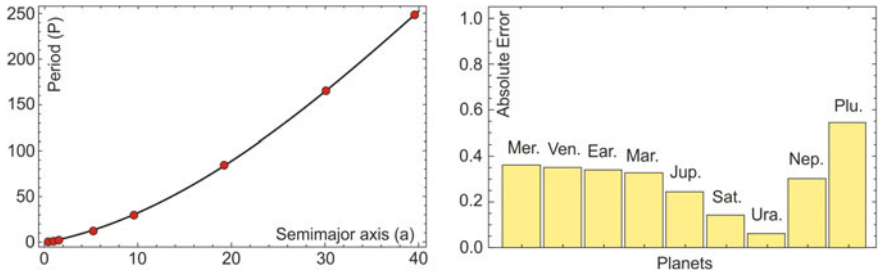


Fig. 11.10 Interpolation via TPS

There are different techniques with different models, but all of them provide a reasonably good approximation; however the complexity of the formulas are quite different. Can one provide a simpler model without loosing accuracy?

11.2.5 Random Models

The simplest form of symbolic regression is when the models are generated just randomly.

```
≪DataModeler'
```

Now we generate models representing algebraic expressions randomly. Let us consider 60 models as population size.

```
randomModels = UpdateModelQuality[RandomModels[DataVariables →
 {a}, PopulationSize → 60], samplePts, observedResponse];
ModelSelectionReport[FitModels@randomModels, SelectionStrategy →
 AllModels]
```

The result can be seen in Table 11.2.

The best model (31) is $P = \sqrt{a^3}$, which has a very small error, $1 - R^2 = 4.284 \times 10^{-8}$ and low complexity: 29. The random model generation technique is very fast, but no guarantee for the proper result, see later.

11.2.6 Symbolic Regression

The most effective technique is using multi-objective optimization. The competing objectives are the complexity and accuracy. Employing symbolic regression and generating 537 models, we get nearly the same result, see Table 11.3, model (4).

Table 11.2 Randomly generated models

Page 1	Page 2	Page 3	Page 4
Model selection report			
	Complexity	$1 - R^2$	Function
31	29	4.284×10^{-8}	$a^{3/2}$
32	29	0.797	$-(2.42 \times 10^{-3}/a^{3/2})$
33	29	0.824	$1/(-8.29 + 3a)$
34	31	1.682×10^{-4}	$(4.12 \times 10^{-2})\ (10 + 1/a + a + 6.24\ a^2)$
35	31	0.013	$9.88 + (1/7 - a)^2$
36	31	0.029	$-\sqrt{a} + (6.88 \times 10^{-2})\ a^6$
37	34	0.703	$(1/a^4)^{1/4}$
38	36	0.250	$164.42\ a^6\ (67.28 + a)^2$
39	41	0.287	$(18.50 + 4\ a)^9$
40	41	0.749	$a^{1/6}/(0.17 - a)$
41	42	0.705	$(-5.47 + 0.10\sqrt{a})/a$
42	44	0.053	$167^{1/3}\ a^2\sqrt{a^{3/2}}$
43	49	0.226	$-(7.37 + a - 677.16\ a^3)^2$
44	50	0.927	$(1.05\ (-514.82 + 3/a))/a^{14/3}$
45	53	0.113	$(93.07 + a^{1/3} - a + a^2)^2$

```
Timing@ParetoFrontLogPlot[
 quickModels1 = SymbolicRegression[
   samplePts, observedResponse, DataVariables → {a},
  AlignModel → True,
 RobustModels → True, TimeConstraint → 150, FitnessPrecision → 10
 ]
]
```

The Pareto front of the solution can be seen in Fig. 11.11.

```
ModelSelectionTable@quickModels1
```

The generated functions belonging to the Pareto front are summarized in Table 11.3.

Figure 11.12 shows the comparison of the errors of the two models.

Here, the chosen model is the 4th, which has a similar accuracy to the best random model. However, $\sqrt{a^3}$ represents lower complexity than $a^{3/2}$.

In Table 11.4 we can see the statistics of relative errors of the different models in percents.

It is inevitable that statistically the best model is provided by the symbolic regression even if its mean error is higher than that of the Kepler solution, but the latter one is simple in practice.

Table 11.3 Symbolic regression models

Page 1	Page 2		
Model selection table			
	Complexity	$1 - R^2$	Function
1	11	0.022	$-12.55 + 6.12\ a$
2	15	0.012	$7.17 + 0.16\ a^2$
3	19	1.948×10^{-4}	$-0.98 + 1.21\ a^{1.45}$
4	20	4.284×10^{-8}	$-(5.59 \times 10^{-3}) + 1.00\sqrt{a^3}$
5	28	1.927×10^{-8}	$8.85 \times 10^{-3} - (6.36 \times 10^{-3})\, a + 1.00\sqrt{a^3}$
6	32	1.214×10^{-8}	$2.69 \times 10^{-3} + (2.85 \times 10^{-6})\, a^3 + 1.00\sqrt{a^3}$
7	36	1.185×10^{-8}	$3.92 \times 10^{-3} + (5.36 \times 10^{-5})\, a^{2.32} + 1.00\sqrt{a^3}$
8	39	9.800×10^{-9}	$-(4.86 \times 10^{-3}) + .99\sqrt{a^3} + (5.96 \times 10^{-4})\, a\,(17 + a)$
9	40	7.278×10^{-9}	$-(4.03 \times 10^{-2}) + (7.08 \times 10^{-2})\sqrt{a^3} - (3.01 \times 10^{-2})\, a + 1.00\sqrt{a^3}$
10	44	6.624×10^{-9}	$0.14 - (2.00 \times 10^{-2})\, a + 1.00\sqrt{a^3} - 0.62/(4 + a)$
11	49	6.351×10^{-9}	$7.14 \times 10^{-2} - (1.56 \times 10^{-2})\, a + 1.00\sqrt{a^3} - 0.33/(4.69 + a^2)$
12	50	5.328×10^{-9}	$71.32 + 0.52\, a + 0.98\,\sqrt{a^3} - 19.50\,(49 + a)^{1/3}$
13	54	4.474×10^{-9}	$546.45 + 0.58\, a + 0.97\sqrt{a^3} - 451.40\,(49.93 + a)^{.0489}$
14	85	1.688×10^{-9}	$-0.54 + 1.00\ a^{3/2} - (6.35 \times 10^{-6})\ a^3 - 34.42/(-629 - (a^3)^{3/4})$
15	124	1.305×10^{-9}	$-0.59 + 1.00\,\mathrm{a}^{3/2} - (3.99 \times 10^{-5})\sqrt{\left(1.14 + a^{\sqrt{3}}\right)^3} - 372.19/(-629 - (\mathrm{a}^3)^{3/4})$

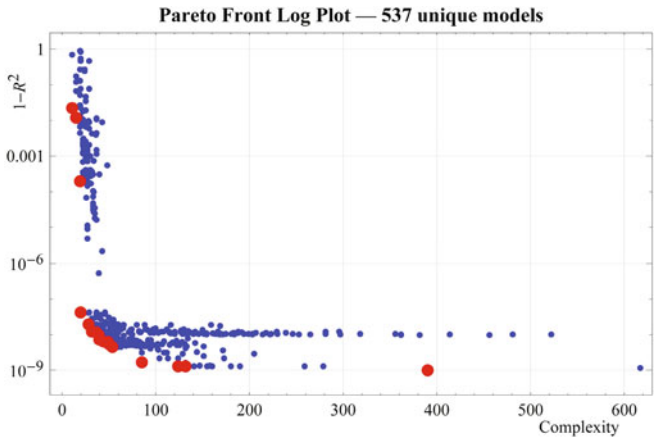


Fig. 11.11 The Pareto front of the Kepler problem solutions

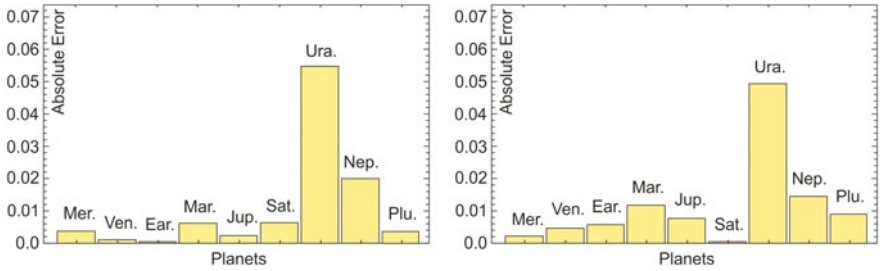


Fig. 11.12 The absolute errors of the randomly generated (to left) and the symbolic regression model (to right)

Table 11.4 Statistics of the relative error (%) of the different approximation methods

Method	Mean error	Max error	Standard deviation
Polynomial regression	25.14	177.49	59.96
Neural network	26.80	191.34	64.22
Support vector machine	8.89	47.47	16.14
Thin plate spline	29.10	149.87	49.59
Kepler solution	0.23	1.48	0.51
Symbolic regression	0.32	0.84	0.37

11.3 Applications

11.3.1 Correcting Gravimetric Geoid Using GPS Ellipsoidal Heights

The accuracy of the gravimetric geoid can be significantly improved using GPS (ellipsoidal) height measurements. The new, adjusted geoid can be constructed as a gravimetric surface plus the so called corrector surface. The difference between the gravimetric and the GPS/leveling geoid is $\Delta N(\varphi, \lambda) = N_{GPS} - N_{grav}$, where ΔN is the discrepancies between the GPS-derived ellipsoidal height and the height from the gravimetric geoid. The variables φ latitude and λ longitude are the ellipsoidal coordinates.

In practice, the various wavelength errors in the gravity solution are usually approximated by different kinds of functions in order to fit the geoid to a set of GPS/leveling points through an integrated least squares adjustment. Several models can be used ranging from simple linear regression to a more complicated seven parameter similarity transformation model. However, as the achievable accuracy of GPS and geoid heights improve, the use of such simplified models may not be sufficient. The problem is further complicated by the fact that selecting the proper model type depends on data distribution, density and quality, which varies for each case.

In order to quantify the effectiveness of the SR method, only global methods are considered here. We use three different parametric models and ANN with one hidden layer and sigmoid activation function, as well as with radial basis activation function (RBF).

11.3.1.1 Dataset for the Numerical Computations

For modeling the corrector surface, 302 GPS ellipsoidal height data points of the Hungarian geoid available from Kenyeres and Virág (1998) were used. As Fig. 11.13 shows, for the parameter estimations, 194 data points for training, and 108 data points for the validation were considered. The results for the different models are presented as follows.

11.3.1.2 Parametric Models

The family of these models based on the general 7-parameter similarity datum transformation, with the simplified classic 4-parameter model of Heiskanen and Moritz (1967, p. 213) is given by

$$\Delta N_4(\varphi, \lambda) = p_1 + p_2 \cos\varphi \cos\lambda + p_3 \cos\varphi \sin\lambda + p_4 \sin\varphi.$$

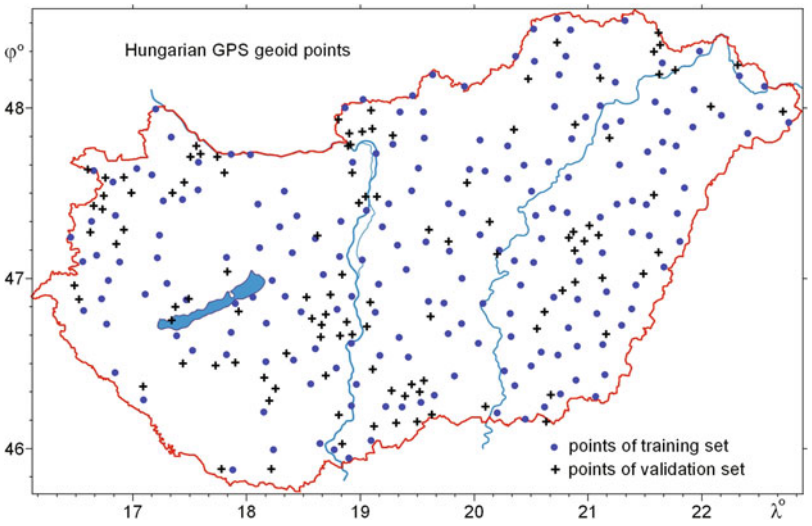


Fig. 11.13 Hungarian GPS geoidal height data: 194 training sets (*marked by circles*) and the 108 validation

An extended version of this model is given with the inclusion of a fifth parameter (Duquenne et al. 1995) as follows,

$$\Delta N_5(\varphi,\lambda) = p_1 + p_2 \cos\varphi\cos\lambda + p_3 \cos\varphi\sin\lambda + p_4 \sin\varphi + p_5 \sin^2\varphi$$

and a more complicated form of the differential similarity transformation model developed by Kotsakis et al. (2001) is given as,

$$\begin{aligned}\Delta N_7(\kappa,\lambda) = {} & p_1 \cos\varphi\cos\lambda + p_2 \cos\varphi\sin\lambda + p_3 \sin\varphi + p_4\, W \sin\varphi\cos\varphi\sin\lambda \\ & + p_5\, W \sin\varphi\cos\varphi\cos\lambda + p_6\, W \sin^2\varphi + p_7\, W(1 - f^2\sin^2\varphi),\end{aligned}$$

where $W = 1/\sqrt{1 - e^2\sin^2\varphi}$, $e^2 = 0.006694379990$ is the numerical eccentricity, and $f = 1/298.257223563$ is the flattening of the reference (WGS84) ellipsoid. The parametric models can be easily achieved via linear parameter estimation

$$\begin{aligned}\Delta N_4(\varphi,\lambda) = {} & 0.065666 - 0.0465406\cos\varphi\cos\lambda - 0.00943184\cos\varphi\sin\lambda \\ & - 0.0185872\sin\varphi,\end{aligned}$$

$$\begin{aligned}\Delta N_5(\kappa,\lambda) = {} & 0.0514108 - 0.0508839\cos\varphi\cos\lambda - 0.00115442\cos\varphi\sin\lambda \\ & - 0.0195404\sin\varphi + 0.0247211\sin^2\varphi,\end{aligned}$$

$$\begin{aligned}\Delta N_7(\varphi,\lambda) = &-0.0530829\cos\varphi\cos\lambda - 0,00618307\cos\kappa\sin\lambda \\ &- 0.0206765\sin\varphi + 0.0241447W\sin\varphi\cos\varphi\sin\lambda \\ &+ 0.0161115\ W\sin\varphi\cos\varphi\cos\lambda + 0.0240435\ W\sin^2\varphi \\ &+ 0.0518829\ W(1 - 0.0000112413\sin^2\varphi),\end{aligned}$$

where $W = 1/\sqrt{1 - 0.00669438\sin^2\varphi}$.

11.3.1.3 Artificial Neural Network (ANN) Models

Two types of ANN models have been studied: ANN with sigmoidal activation function and ANN with radial bases function (RBF). Both of them have universal approximation properties like polynomials, (e.g., Kecman 2001). Therefore, ANN with such activation functions can be a good candidate for approximating the corrector surface. The general structure of an ANN network with sigmoidal activation function in one hidden layer and with a linear tail is,

$$\Delta N(\varphi,\lambda) = \sum_{i=1}^{n} \frac{p_{1,i}}{1+\exp\left(p_{2,i}+p_{3,i}\varphi+p_{4,i}\lambda\right)} + p_5\kappa + p_6\lambda + p_7$$

where n is the number of the neurons in the hidden layer (Fig. 11.14). There are 3 input nodes for φ, λ and for the bias 1. The linear combinations of these inputs are transformed in the hidden layer nodes according to the sigmoid activation function. Then the linear combination of these transformed signals with the linear tail having weights p_5, p_6 and p_7 represent the output of the network. There is no further signal transformation in the output node. Similarly, the ANN with RBF (Fig. 11.15) is

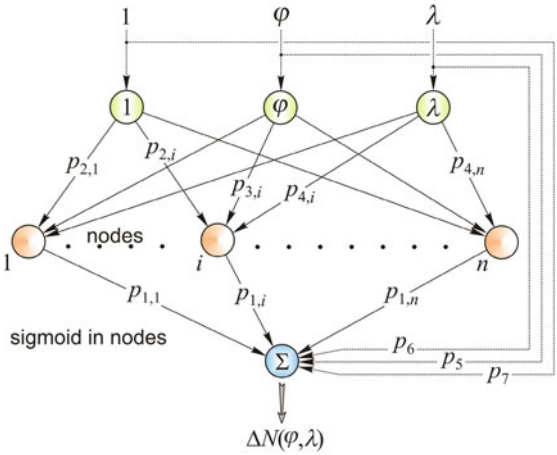


Fig. 11.14 ANN with sigmoid activation function

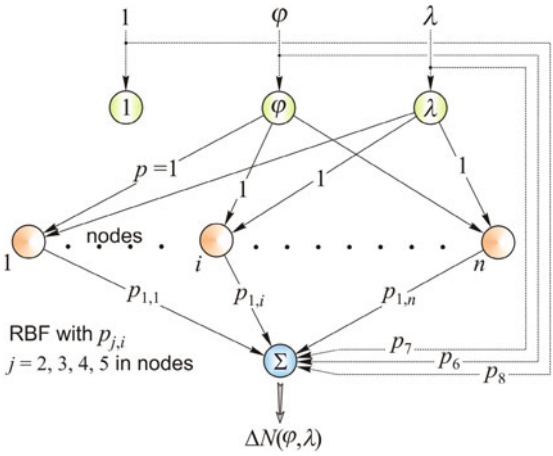


Fig. 11.15 ANN with RBF activation function

$$\Delta N(\varphi,\lambda)=\sum_{i=1}^{n} p_{1,i}\exp\left[p_{2,i}(p_{3,i}+\varphi)^2+p_{4,i}(p_{5,i}+\lambda)^2\right]+p_6\varphi+p_7\lambda+p_8.$$

The structure of the RBF network is quite similar to the network with sigmoid activation function. The main differences are that the weights between the input and hidden layer nodes are 1, and the activation functions in the hidden layer nodes are RBF functions. The RBF neural network has 4 nonlinear parameters ($p_{2,i}$, $p_{3,i}$, $p_{4,i}$, $p_{5,i}$) while the feedforward network has only 3, namely $p_{2,i}$, $p_{3,i}$ and $p_{4,i}$. Here 25 ANN models for both activation functions were created with $n = 1, 2, \ldots, 25$ nodes in the hidden layer, respectively.

Figure 11.16 represents the result for networks with sigmoid activation function. It can be clearly seen that the 23rd model with 23 nodes—represented by the last point of the Pareto Front (circle points)—is the best, since the last two models with 24 and 25 nodes (triangular points) have higher complexity and greater error.

In order to compare the results of the different regression techniques, RMSE is used as an error measure, which is usual in engineering society. However, to illustrate the Pareto front of a model family belonging to a given technique, the fitness measure R^2 is used since its range is restricted to the interval [0, 1]. Therefore in the figures, the model error is characterized by $1 - R^2$. Figure 11.17 shows the Pareto Front of the 25 ANN models with RBF activation functions.

Figure 11.17 shows the Pareto Front of the 25 ANN models with RBF activation functions. Now, the best is the last but one model, since the last model has higher complexity as well as greater error than the 24th model. As the figure shows, this model has higher complexity and somewhat greater error than the best ANN model with sigmoid activation function. Both types of the ANN models were evaluated via the Neural Network application package of *Mathematica.* We should mention here that the latest version *Mathematica* 11.1.1 has already built-in functions for ANN.

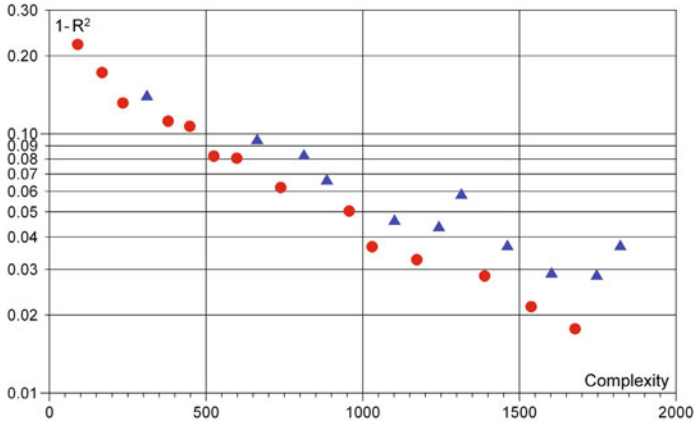


Fig. 11.16 The Pareto front (*circle points*) in case of the ANN models with sigmoidal activation function. The *triangular points* indicate optimal, but not Pareto optimal solutions

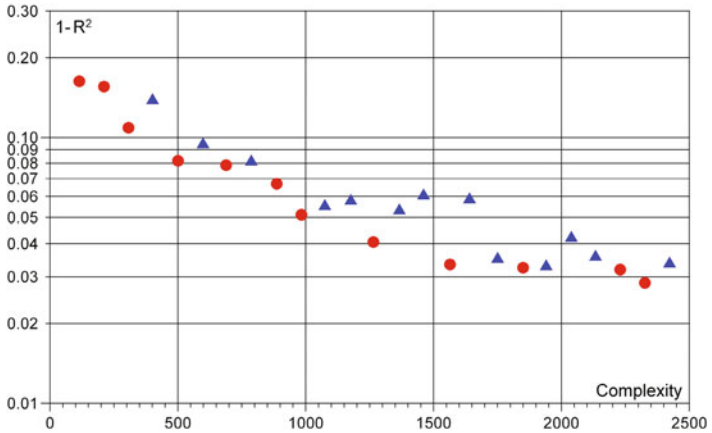


Fig. 11.17 The Pareto front (*circle points*) in case of the ANN models with RBF activation function. The *triangular points* indicate optimal, but not Pareto optimal solutions

11.3.1.4 Application of Symbolic Regression

There are many parameters to be adjusted properly in order to make the symbolic regression algorithm work successfully. But, probably, the two very important ones are the type of the functions in the initial population and the type of the functions employed as basic functions in the regression process. Functions included in the basic function set are: plus, times, divide and subtract as basic arithmetic; square, sqrt, inverse as Extended Mathematics; power, exp, log as Power Mathematics and sigmoid, radial basis (RBF), see *DataModeler*.

During this study, the initial population was selected in three different ways. The functions in the initial population can be selected randomly; this is the default, automatic option. Another more intelligent way is to apply Keijzer-expansion, (Keijzer 2003). The name of the method originates from Maarten Keijzer who suggested randomly synthesizing a selection of models, selecting the one having the best fit to the targeted response, removing the model contribution from the response, and iteratively repeating the process on the residual until the desired accuracy is achieved. The resulting model may provide a good starting point for SR. The third method applied here simply employs the ANN models as initial population for SR. The reason for this choice is that SR may improve the robustness as well as the generalization ability—the quality of the fitting on the validation set-of the ANN models. Different models have been applied with the following measured data. As Fig. 11.18 shows, employing SR with randomly generated initial population also results in a function with high complexity.

Applying Keijzer expansion to create the initial population, we obtained the following models (Fig. 11.19). As already mentioned, Keijzer expansion provides a good initial population for symbolic regression. In our case, all of the Keijzer models—all models, not only models which are the elements of the Pareto Front—have relatively low complexity (Fig. 11.19). The last 5 models of the Pareto Front of the 200 Keijzer models can be seen in Table 11.5.

It can be seen that this model has considerable lower complexity than the ANN ones however their errors are rather high. The statistics of the last 5 models of the Pareto front of the Keijzer models can be seen in Table 11.5.

As Fig. 11.20 shows, employing these models as initial population, the result of the SR can be improved considerably.

Comparing this result with those of ANN models (Figs. 11.16 and 11.17), it can be seen that the models generated using Keijzer models as initial population have relative higher errors but considerably lower complexities than those of the ANN

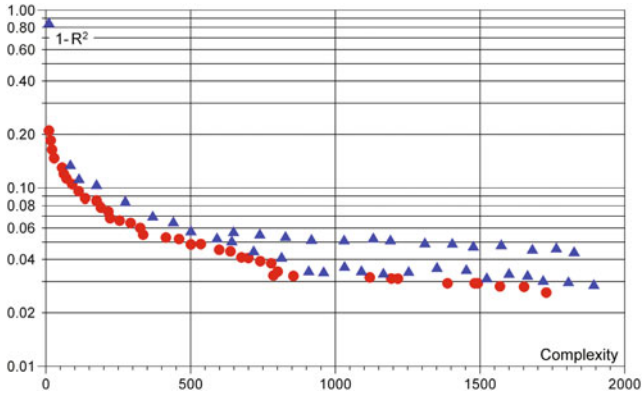


Fig. 11.18 The Pareto front (*circle points*) in case of the SR model with random initial population. The *triangular points* indicate optimal, but not Pareto optimal solutions

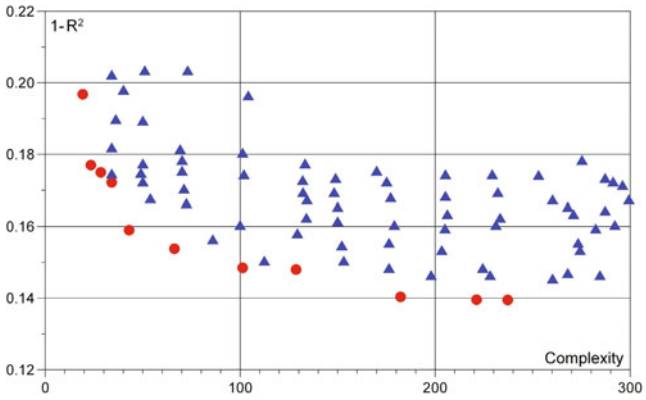


Fig. 11.19 The Pareto front (*circle points*) of the Keijzer models. The *triangular points* indicate optimal, but not Pareto optimal solutions

models. In Table 11.6, the statistics of the models investigated in study are summarized.

Here, η is the ratio of the RMSE of the validation and that of the training set. The generalization ability of the model reliable is ideal if η is close to 1. The parametric models are linear models and they cannot be considerably improved even with increasing the number of their parameters. The ANN models are nonlinear models and provide much better results than the parametric ones; however, their complexities are very high and the teaching process requires a large number of iterations leading to considerable time in both cases. In addition, the value of $\eta > 1$ indicates overlearning on the training set.

The SR model with randomly generated initial population (automatic) is quite good and $\eta < 1$ indicate under-learning. Unfortunately, the required running time is about 10 times larger than that of the ANN models. This model also has high complexity. However, employing Keijzer expansion, its complexity can be reduced considerably without significantly worsening the model quality. SR models with initial population with ANN models provide also quite good results. Considering these results, one may select the SR with ANN models as initial population since they provide quite good results; however the complexity of the resulting models are considerably higher. Therefore, on the one hand, one may select moderate complexity but on the other hand, if the model quality is the most important feature, then the SR model with initial population of the sigmoid ANN model is the best choice. The visualization of the corrector surface can be seen in Fig. 11.21. It is not surprising that one needs so high complexity function to approximate it.

Table 11.5 The last five models of the Keijzer expansion

No.	Complexity	$1 - R^2$	Functions
1	101	0.0149	$0.513 - \frac{6.738\times10^{-6}\lambda^4}{\varphi^2} - 5.412 \times 10^{-7}\left(-6.547 + \sqrt{\lambda}\right)\lambda^3\varphi - \frac{4.996\times10^{-4}}{5.961-\lambda+2\varphi}$
2	128	0.148	$0.513 - \frac{6.738 \times 10^{-6}\lambda^4}{\varphi^2} - 5.412 \times 10^{-7}\left(-6.547 + \sqrt{\lambda}\right)\lambda^3\varphi - 4.541 \times 10^{-27}\lambda^8\varphi^8 - \frac{4.996 \times 10^{-4}}{5.961 - \lambda + 2\varphi}$
3	182	0.140	$1.236 - 0.01\left(5 + 2\lambda - \frac{37.575\ (-0.589 + \lambda)}{\varphi}\right)^{-1} - \frac{0.024}{-3.314 + \lambda - 2\varphi} - 0.002\left(20 - \frac{2.484}{\lambda} - \varphi\right)^2 - 2.044 \times 10^{-9}\lambda^4(-0.197 + 2\lambda + \varphi)$
4	221	0.140	$1.236 - 0.01\left(5 + 2\lambda - \frac{37.575\ (-0.589 + \lambda)}{\varphi}\right)^{-1} - \frac{0.024}{-3.314 + \lambda - 2\varphi} - 0.002 \times \left(20 - \frac{2.484}{\lambda} - \varphi\right)^2 - 2.044 \times 10^{-9}\lambda^4(-0.197 + 2\lambda + \varphi) - \frac{22.174}{0.125 - \lambda + \lambda^2 + \varphi}$
5	237	0.140	$1.236 - 0.01\left(5 + 2\lambda - \frac{37.575\ (-0.589 + \lambda)}{\varphi}\right)^{-1} - \frac{0.024}{-3.314 + \lambda - 2\varphi} - 0.002\left(20 - \frac{2.484}{\lambda} - \varphi\right)^2 - 2.044 \times 10^{-9}\lambda^4(-0.197 + 2\lambda + \varphi) - \frac{22.174}{0.125 - \lambda + \lambda^2 + \varphi} + \frac{0.002}{-16.291 + \varphi}$

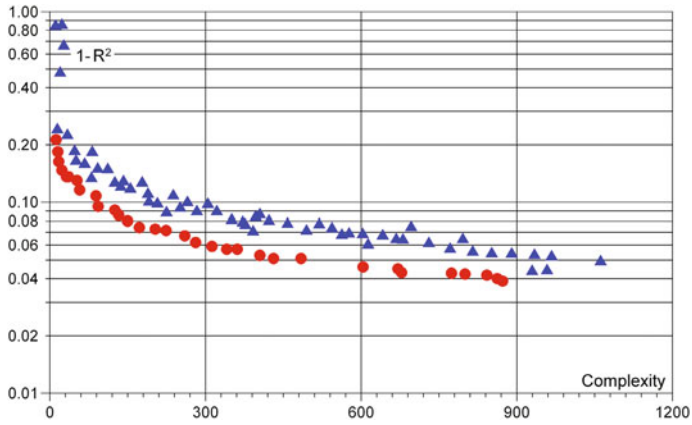


Fig. 11.20 The Pareto front (*circle points*) in case of the SR models based on the Keijzer models as initial population. The *triangular points* indicate optimal, but not Pareto optimal solutions

Table 11.6 Statistics of errors of different corrector models at the points of the validation set

Model	Standard Dev. (cm)	Min (cm)	Max (cm)	RMSE (cm)	$\eta = \frac{RMSE_V}{RMSE_T}$	Complexity
4-Parameter	8.23	−27.79	14.78	6.17	1.01	–
5-Parameter	8.03	−27.18	14.42	8.06	0.99	–
7-Parameter	8.06	−25.87	14.26	8.06	1.00	–
ANN Sigmoid	1.87	−6.01	4.21	1.90	1.64	1675
ANN RBF	2.03	−6.98	4.75	2.08	1.42	2323
SR Automatic	1.43	−3.09	3.52	1.40	0.45	1727
SR Keijzer	1.76	−4.66	4.20	1.73	0.52	873
SR Sigmoid	1.23	−3.61	3.14	1.65	1.34	2329
SR RBF	1.62	−3.98	3.25	1.62	0.45	1382

11.3.2 Geometric Transformation

Transformation of coordinates is important in computer vision, photogrammetry as well as in geodesy. In this example we consider some standard 2D transformations as similarity, affine and projective transformations between the coordinates of the fiducial marks on the comparator plate and that of the corresponding points on the reseau plate, see Gosh (2005). The 16 observed (x, y) and calibrated (X, Y) coordinates are given in Table 11.7.

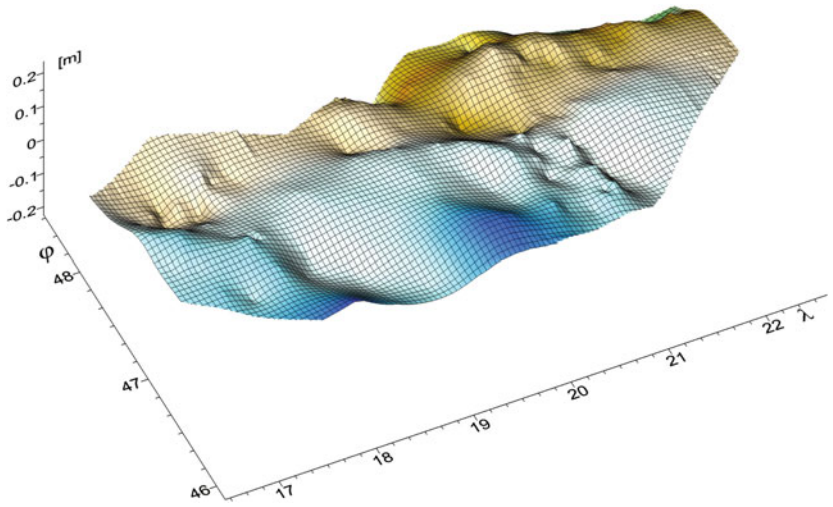


Fig. 11.21 The corrector surface computed using the SR model employing Keijzer expansions

Table 11.7 The coordinates of the corresponding observation points on the comparator and reseau planes

Points	x (mm)	y (mm)	y (mm)	y (mm)
1	−113.767	−107.4	−110	−110
2	−43.717	−108.204	−40	−110
3	36.361	−109.132	40	−110
4	106.408	−109.923	110	−110
5	107.189	−39.874	110	−40
6	37.137	−39.07	40	−40
7	−42.919	−38.158	−40	−40
8	−102.968	−37.446	−100	−40
9	−112.052	42.714	−110	40
10	−42.005	41.903	−40	40
11	38.051	4.985	40	40
12	108.089	4.189	110	40
13	108.884	11.221	110	110
14	38.846	111.029	40	110
15	−41.208	111.961	−40	110
16	−111.249	112.759	−110	110

11.3.2.1 Similarity Transformation

The advantage of this model is that it is linear in the coordinates as well as in the parameters. Consequently an iterative solution is not required and the inverse transformation is easy. This transformation can be parametrized in the following form,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ -b & a \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix}.$$

For each observed i-th point the following pair of observations equation can be written for the residuals $(r_{x_i},\ r_{y_i})$,

$$\begin{pmatrix} X_i & Y_i & 1 & 0 \\ Y_i & -X_i & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \begin{pmatrix} -x_i \\ -y_i \end{pmatrix} = \begin{pmatrix} r_{x_i} \\ r_{y_i} \end{pmatrix}.$$

A minimum of two fiducial marks or reseau crosses are required for a unique solution. Having more observation points, linear least square can be directly applied. In our example the result can be seen in Table 11.8.

11.3.2.2 Affine Transformation

This is also a linear model although it needs six parameters, therefore at least three fiducial marks or reseau crosses are required for a unique solution. Iterative solution is not required and the inverse transformation is easy. This model can be parametrized in the following form

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ d & e \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} c \\ f \end{pmatrix}.$$

For each observed ith point the following pair of observations equation can be written for the residuals $(r_{x_i},\ r_{y_i})$,

$$\begin{pmatrix} X_i & Y_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & X_i & Y_i & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} + \begin{pmatrix} -x_i \\ -y_i \end{pmatrix} = \begin{pmatrix} r_{x_i} \\ r_{y_i} \end{pmatrix}.$$

Having more than 3 observation points, linear least square can be directly applied. In our example the result can be seen in Table 11.9.

Table 11.8 The estimated parameters of the similarity transformation model

Parameter	Value
a	0.999162
b	−0.011416
c	2.4471
d	−1.3878

Table 11.9 The estimated parameters of the affine transformation model

Parameter	Value
a	0.999167
b	−0.011339
c	2.4470
d	0.011494
e	0.999157
f	−1.3877

11.3.2.3 Projective Transformation

The projective transformation can be expressed by the following two equations,

$$x = \frac{a_1 X + a_2 Y + a_3}{c_1 X + c_2 Y + 1}$$

and

$$y = \frac{b_1 X + b_2 Y + a_3}{c_1 X + c_2 Y + 1}.$$

These equations are a special case of the co-linearity condition for mapping of 2D points from one plane onto another. There are 8 unknown parameters, so if only four fiducial marks are available, the solution is not unique. This is perhaps the main reason why it is not frequently used. However, it is relevant for systems that have been retro-fitted with a reseau plate, such as the Hasselblad camera used to acquire the assignment imagery.

The least square solution is nonlinear due to the rational nature of the functions. However, an approximate linear solution can be implemented if both sides of the equations are multiplied by the denominator and partial derivatives with respect to the observables (as in the combined adjustment model) are ignored.

The inverse of the transformation can be also computed by solving the equation system in symbolic form for X and Y, we obtain

$$X = \frac{a_2(-y + a_3) + (x - a_3)b_2 + (-x + y)a_3 c_2}{-x b_2 c_1 + a_2(-b_1 + y c_1) + x b_1 c_2 + a_1(b_2 - y c_2)}$$

and

$$Y = \frac{a_1(y - a_3) + (-x + a_3)b_1 + (x - y)a_3 c_1}{-x b_2 c_1 + a_2(-b_1 + y c_1) + x b_1 c_2 + a_1(b_2 - y c_2)}.$$

Having more than four observation points allows application of nonlinear least squares. In our example the result can be seen in Table 11.10.

Table 11.10 The estimated parameters of the projective transformation model

Parameter	Value
a_1	0.999165
a_2	−0.01134
a_3	2.445675
b_1	0.011494
b_2	0.999156
b_3	−1.393161
c_1	0
c_2	−0.000001

11.3.2.4 Symbolic Regression

By definition, neither the model nor the parameters of the model are known. However in order to have a chance for computing the inverse of the transformation trigonometric functions are excluded from the function-set. The Pareto front for the $x = x\ (X,\ Y)$ relation can be seen in Fig. 11.22.

Table 11.11 shows the Model selection report. We selected the best linear model, the third one, with complexity 30 and with error $1 - R^2 = 3.029 \times 10^{-9}$.

Remark In Table 11.11 x_1 and x_2 stand for X and Y.

So our model is

$$x = -2.43296 + 1.0007X + 0.0113571\,Y - 9.51358 \times 10^{-7} X\,Y.$$

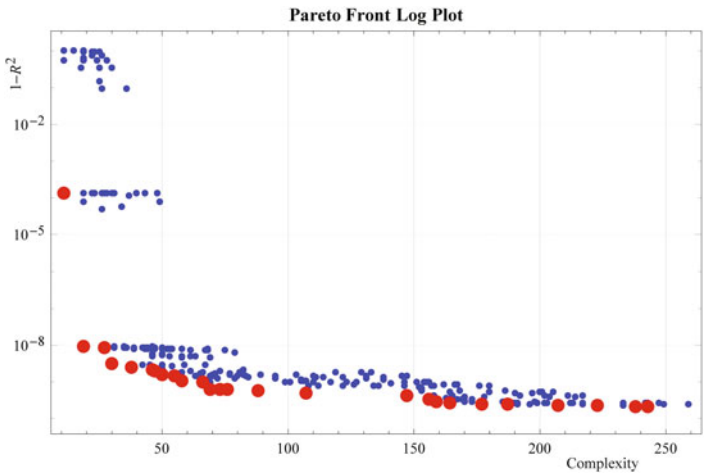


Fig. 11.22 The Pareto front in case of $x = x\ (X,\ Y)$

Table 11.11 Model selection report for $x = x\ (X, Y)$

Page 1	Page 2		
Model selection report			
	Complexity	$1 - R^2$	Function
1	19	9.312×10^{-9}	$-2.43 + 1.00\ x_1 + (1.14 \times 10^{-2})\ x_2$
2	27	8.794×10^{-9}	$-2.43 + 0.13/x_1 + 1.00\ x_1 + (1.14 \times 10^{-2})\ x_2$
3	30	3.029×10^{-9}	$-2.43 + 1.00\ x_1 + (1.14 \times 10^{-2})\ x_2 - (9.51 \times 10^{-7})\ x_1\ x_2$
4	38	2.540×10^{-9}	$-2.43 + 0.13/x_1 + 1.00\ x_1 + (1.14 \times 10^{-2})\ x_2 - (9.49 \times 10^{-7})\ x_1\ x_2$
5	46	2.165×10^{-9}	$-2.43 + 0.12/x_1 + 1.00\ x_1 + 0.11/x_2 + (1.13 \times 10^{-2})\ x_2 - (9.50 \times 10^{-7})\ x_1\ x_2$
6	47	1.934×10^{-9}	$-2.43 + 1.00\ x_1 - 1.01/(x_2)^2 + (1.14 \times 10^{-2})\ x_2 - (9.53 \times 10^{-7})\ x_1\ x_2$
7	50	1.547×10^{-9}	$-2.44 + 1.00\ x_1 + 0.11/x_2 + (1.13 \times 10^{-2})\ x_2 - (9.54 \times 10^{-7})\ x_1\ x_2 + (5.17 \times 10^{-7})(x_2)^2$
8	55	1.460×10^{-9}	$-2.43 + 0.12/x_1 + 1.00\ x_1 - 9.94/(x_2)^2 + (1.14 \times 1\ 0^{-2})\ x_2 - (9.51 \times 10^{-7})\ x_1\ x_2$
9	58	1.085×10^{-9}	$-2.44 + 0.12/x_1 + 1.00\ x_1 + 0.11/x_2 + (1.13 \times 10^{-2})\ x_2 - (9.52 \times 10^{-7})\ x_1\ x_2 + (5.13 \times 10^{-7})\ (x_2)^2$

Similarly for the relation $y = y\ (X, Y)$ we get

$$y = 1.42065 - 0.0115116X + 1.00071\,Y - 1.78795 \times 10^{-7} X\,Y - 5.60054 \times 10^{-7} Y^2.$$

The Fig. 11.23 illustrates how this nonlinear transformation warps (deforms) the original (X, Y) plane into the (x, y) plane.

In Table 11.12 the performance of the different transformation methods can be seen.

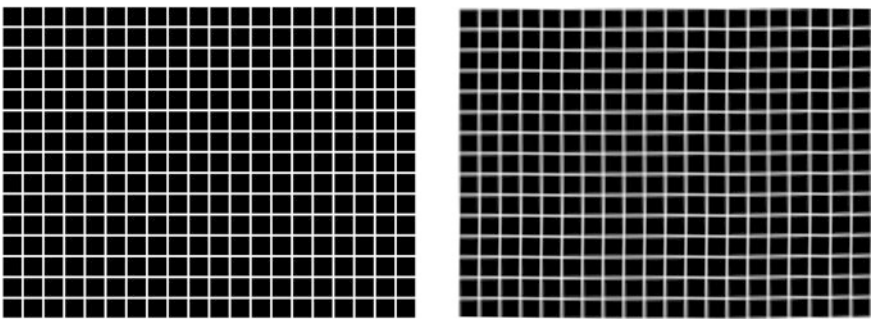


Fig. 11.23 The original plane and below the plane after warping via the nonlinear transformation

Table 11.12 Some statistical values of the different transformation models

Transformation model	Max of absolute errors (mm)	Standard deviation of the absolute errors (mm)	RMSE of the residual errors (mm)
Similarity	0.0268663	0.00882640	0.0106
Affine	0.0180314	0.00518175	0.0089
Projective	0.0128281	0.00342605	0.0066
Symbolic regression	0.0078720	0.00197883	0.0028
Similarity	0.0268663	0.00882640	0.0106
Affine	0.0180314	0.00518175	0.0089

The SR model is better than the other models, and *its inverse transformation can be computed* via *Groebner basis*.

Our model is $(x, y) = F(X, Y)$, now we should compute F^{-1} , where $(X, Y) = F^{-1}(x, y)$. Let us consider $F(X, Y)$ in general form

$$F_1 = a_1 + b_1X + c_1Y + d_1XY,$$
$$F_2 = a_2 + b_2X + c_2Y + d_2XY + e_2Y^2.$$

Assuming that the coordinates (x, y) are given, we have to solve the following multivariate polynomial system

$$F_1(X, Y) - x = 0,$$
$$F_2(X, Y) - y = 0.$$

Let us compute the Groebner basis for X, we get

$$\begin{aligned} gbX(X, x, y) = &-a_1^2e_2 - c_1c_2x - e_2x^2 - b_2c_1^2X + b_1c_1c_2X - c_2d_1xX \\ &- c_1d_2xX + 2b_1e_2xX - 2b_2c_1d_1X^2 + b_1c_2d_1X^2 + b_1c_1d_2X^2 - b_1^2e_2X^2 \\ &- d_1d_2xX^2 - b_2d_1^2X^3 + b_1d_1d_2X^3 - a_2(c_1 + d_1X)^2 \\ &+ a_1(2e_2(x - b_1X) + c_1(c_2 + d_2X) + d_1X(c_2 + d_2X)) + c_1^2y + 2c_1d_1Xy + d_1^2X^2y \end{aligned}$$

which is a polynomial for X of third order

$$\delta(x, y)X^3 + \gamma(x, y)X^2 + \beta(x, y)X + \alpha(x, y) = 0.$$

So to get X for given (x, y) we have to find the roots of this polynomial.

11.4 Exercise

11.4.1 Bremerton Data

Figures 11.24 and 11.25 represent the digital elevation model of Bremerton West, Washington. We should like to approximate this surface via symbolic regression model,

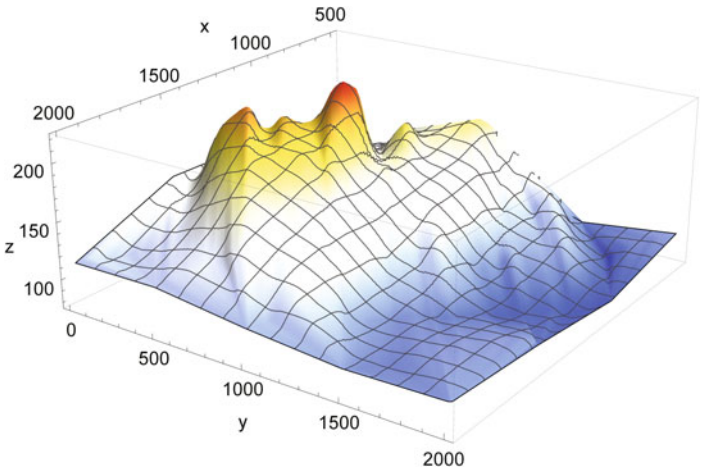


Fig. 11.24 The digital elevation model of the Bremerton West, Washington

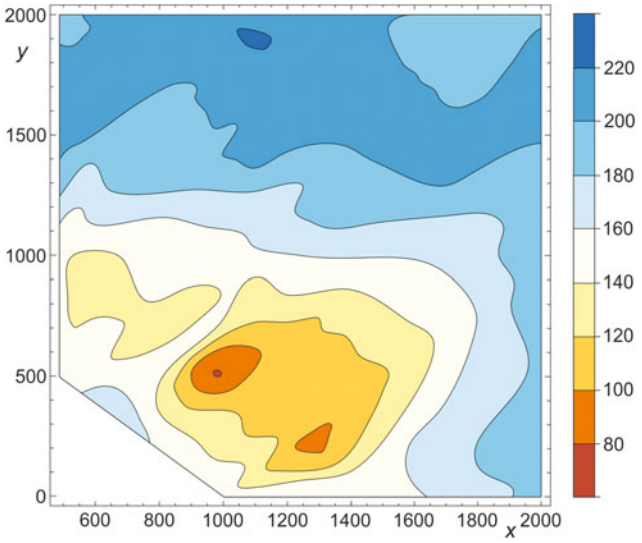


Fig. 11.25 Contour plot of the digital elevation model of the Bremerton West, Washington

```
dataFew = Import[″D:\\bremertonM.dat″];

≪DataModeler′
```

The data pairs of the digital elevation model,

```
samplePts = Map[{#[[1]],#[[2]]}&,dataFew];(*{x,y}*)
observedResponse = Map[#[[3]]&,dataFew];(*{z}*)
```

First let us try the random model generation technique.

11.4.1.1 Random Models

Now the population size is 200 models.

```
randomModels = UpdateModelQuality[
 RandomModels[DataVariables → {x,y},
 FunctionPatterns → {1,BuildFunctionPatterns[″PowerMath″,″RBF″,
 ″Tanh″,″Sinusoids″,″SigmoidDM″]},PopulationSize → 200],
 samplePts,observedResponse];
```

The result can be seen in Table 11.13

```
ModelSelectionReport[FitModels@randomModels,
 SelectionStrategy → AllModels]
```

The best model is the number 50 with 57 complexity and having relatively high error ($1- R = 0.443$).

Table 11.13 The result of the random model generation

Page 1	Page 2	Page 3	Page 4
Model selection report			
	Complexity	$1 - R^2$	Function
46	48	0.988	$\text{Log}[\text{Log}[x/8]]^2$
47	55	0.999	$\text{Cos}[9169.39\ x^5\ \text{Tan}[y]^2]$
48	55	1.000	$1/(-9 + (\text{Cos}[x] - \text{Sin}[x])^x)$
49	56	1.000	$\text{Tan}[12 - 2.17/(-3.81 - x) + x]$
50	57	0.443	$\text{Sqrt}[-4 + 1/\text{Log}[7]^{10} + x] - y$
51	60	0.377	$-1.05 + \text{Sin}[x] + \sqrt{y} + y^{0.98} + \text{Tan}\,[\text{Cos}[y]]$

11.4.1.2 Keijzer Models

The problem seems to be quite difficult; therefore first we employ a Keijzer model population of 302 models, which can be an initial population for the traditional symbolic regression. Figure 11.26 shows the Pareto front of the Keijzer models.

```
ParetoFrontPlot[
 keijzerModels = KeijzerExpansion[samplePts, observedResponse,
   DataVariables → {x, y}, TimeConstraint → 300,
   FunctionPatterns → {1, BuildFunctionPatterns["PowerMath"]},
   KeijzerPopulationSize → 300, MaximumKeijzerModelComplexity → 0.3,
   MemoryLimit → 5"GB"]
]
```

```
ModelSelectionReport[keijzerModels, QualityBox
  → {All, 0.15}]
```

The Table 11.14 shows statistics of the Keijzer models.

11.4.1.3 Symbolic Regression

Employing these Keijzer models as initial model population, 451 further models were generated, see Fig. 11.27

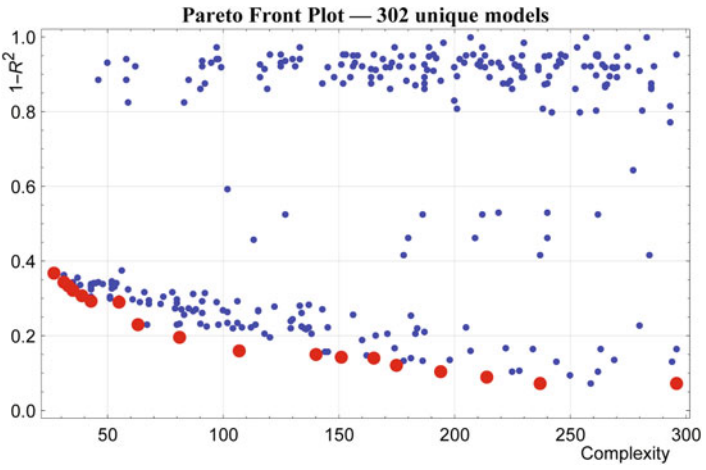


Fig. 11.26 The Pareto front of the generated Keijzer models

Table 11.14 The generated Keijzer models

Model selection report			
	Complexity	$1 - R^2$	Function
1	140	0.149	$-303.65 + 44.02x^{1/3} - 0.15x + 3.15y^{1/3} - (4.05 \times 10^{-2})y + (28.08y)/x + (2.76 \times 10^{-5})xy - (1.29 \times 10^{-4})y^2 + (4.31 \times 10^{-8})y^3 + 88.72\,(3.96 + y)^{1/9} + 471.85/(7.51 + y)$
2	151	0.143	$194.22 - 19634.99/x - 1.18\sqrt{x} + (1.42 \times 10^{-2})x - (6.24 \times 10^{-9})x^3 - 1.01y^{1/3} + 6.11\sqrt{y} - (4.18 \times 10^{-2})y + (1.74y)/x + (1.28 \times 10^{-5})xy - (9.68 \times 10^{-5})y^2 + (6.49 \times 10^{-12})y^4 + (1.49 \times 10^{-8})(23.10 + y)^3$
3	165	0.140	$165.30 - 4394.05/x + 0.052\sqrt{x} - (3.62 \times 10^{-2})x - (7.21 \times 10^{-9})x^3 + 4.79\sqrt{y} - (4.65 \times 10^{-2})y - (1.38 \times 10^{-4})y^2 + (4.7610^{-8})y^3 + (15.65\,(13.11 + x + y))/x + (6.37 \times 10^{-11})x^{5/2}(17.98 + x + 2y)$
4	175	0.120	$136.76 + 0.14x - (4.30 \times 10^{-5}) \times x^2 + 18.69y^{1/3} - 0.54 \times y + 0.11 \times y^{4/3} - (6.95 \times 10^{-4}) \times y^2 + (1.35 \times 10^{-7}) \times y^3 - 1.09\,(xy)^{1/3} - 3.34\sqrt{-5.65 + x + y} + (1.03 \times 10^{-9})\,(\sqrt{x} + x + y)^3$
5	194	0.104	$-313.81 + 56.52x^{1/3} - 0.18x - (5.52 \times 10^{-9})(x - y)^3 + 7.78y^{1/3} - 0.79\sqrt{y} - 0.14y + (1.052y)/x + (7.26 \times 10^{-6})xy + (2.28 \times 10^{-9})xy^2 - (2.70 \times 10^{-8})y^3 + (1.36 \times 10^{-11})y^4 + 15.32\,(1.94 + y)^{1/3} + 1024.05/(12 + 2y)$
6	214	0.090	$-7.82 + 16.87x^{1/3} + 0.14x - (5.96 \times 10^{-5})x^2 + 18.69y^{1/3} - .52\sqrt{y} - 0.54y + (1.71y)/x + 0.11y^{4/3} - (6.95 \times 10^{-4})y^2 + (1.35 \times 10^{-7})y^3 - 1.09\,(xy)^{1/3} - 3.34\sqrt{-5.65 + x + y} + (1.03 \times 10^{-9})\,(\sqrt{x} + x + y)^3$

(continued)

Table 11.14 (continued)

Model selection report			
	Complexity	$1 - R^2$	Function
7	237	0.072	$140.48 - 8679.84/x + (1.03 \times 10^{-2})x - (4.59 \times 10^{-6})x^2 - (1.21 \times 10^{-8})x^3$ $+ (1.66 \times 10^{-2})\sqrt{x^2} + 7.18\sqrt{y} - 0.21y + (1939078.70y)/x^3$ $+ (3.48 \times 10^{-6})xy + (5.31 \times 10^{-9})x^2y - (3.12 \times 10^{-6})y^2 + (8.01 \times 10^{-9})y^3$ $+ 821.90/(-5.38 + y)^2 - ((7.32 \times 10^{-6})y^3)/(3 + x + y) - 43233.23/\left(x^{1/3} + x + y\right)$
8	296	0.071	$-1074.43 + 3465.78/x^{1/3} - 0.31x - (6.73 \times 10^{-5})x^2 + 3.77\left(x^2\right)^{1/3}$ $- (2.84 \times 10^{-9})(x - y)^3 - 11.32y^{1/3} + 15.35\sqrt{y} - 4.45y^{2/3} - (6.33 \times 10^{-2})y$ $- (4.29 \times 10^{-5})xy + (2.66 \times 10^{-10})x^2y^{3/2} - (2.42 \times 10^{-5})y^2 + (1.72 \times 10^{-11})x^2y^2$ $+ (2.29 \times 10^{-8})y^3 - (5.39 \times 10^{-3})(16.00 + y)^{3/2} + 27.17\sqrt{1 + x + y}$

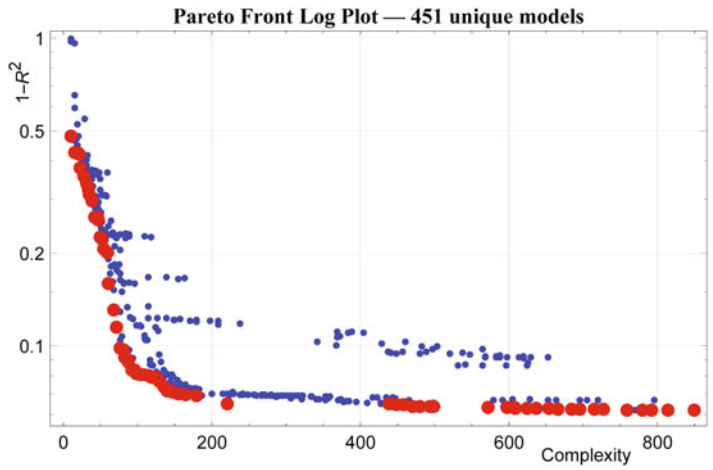


Fig. 11.27 The Pareto front and models via symbolic regression

```
ParetoFrontLogPlot[
 quickModels1 = SymbolicRegression[
 samplePts, observedResponse, DataVariables → {x, y},
 TimeConstraint → 10000, FunctionPatterns →
  {1, BuildFunctionPatterns["PowerMath", "Sinusoids",
"SigmoidDM", "RBF", "Tanh"]},
 InitialPopulation → keijzerModels, MemoryLimit → 20"GB"
 ]
]
```

```
ModelSelectionReport[quickModels1, QualityBox
  → {All, 0.0635}]
```

The Table 11.15 shows the statistic of the models.

The Pareto front consists of 69 models,

```
Length[ParetoFront[quickModels1]]
69
```

We select the model having number 9, see Tables 11.5 and 11.16

```
ModelSelectionReport[ParetoFront[quickModels1][[9]]]
```

Table 11.15 The statistics of the symbolic models

Page 1	Page 2		
Model selection table			
	Complexity	$1 - R^2$	Function
1	493	0.063	$425.89 - 87.87\ \mathrm{Sin}[4527.20 + \mathrm{Sin}[(4514.99 + y)^{1/3}]^4] - 37.87\ \mathrm{Sin}[\mathrm{Sin}[(20.53 + x + y)^{1/3}]^3]^3 - 6194.88\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[(64.53 + x + y)^{1/3}]]+ \mathrm{Sin}[x^{1/3}]]]]]]]] + \mathrm{Sin}[(x + 3\ y)^{1/9}] + (4527.20 + y)^{1/4})^{-1.34}$
2	499	0.063	$418.07 + 94.42\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4514.99 + y)^{1/3}]^4] - 37.72\ \mathrm{Sin}[\mathrm{Sin}[(20.53 + x + y)^{1/3}]^3]^3 - 6228.04\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[(64.53 + x + y)^{1/3}]]+\mathrm{Sin}[x^{1/3}]]]]]]] + \mathrm{Sin}[(x + 3\ y)^{1/9}] + (4527.20 + y)^{1/4})^{-1.34}$
3	572	0.063	$429.71 - 116.02\ \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(x + y)^{1/3}]]^3]^3 + 90.23\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 6379.36\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4+\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(2.083 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-1.34}$
4	597	0.063	$450.00 + 91.34\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 39.53\ \mathrm{Sin}[\mathrm{Sin}[(20.53 + x + y)^{1/3}]^3]^3 - 6759.17\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-1.34}$
5	607	0.063	$1203.22 + 90.92\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 38.06\ \mathrm{Sin}[\mathrm{Sin}[(x^{1/3} + x + y)^{1/3}]^3]^3 - 2906.45\ -0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
6	609	0.063	$1190.92 - 121.16\ \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(7 + x + y)^{1/3}]]^3]^3 + 92.49\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 2883.90\ -0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]]+\mathrm{Sin}[x^{1/3}]]]]]]]]]] + \mathrm{Sin}[(x + 3\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
7	625	0.063	$1243.53 + 91.58\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 38.97\ \mathrm{Sin}[\mathrm{Sin}[(16.53 + x + y)^{1/3}]^3]^3 - 3012.35\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
8	637	0.063	$1223.44 - 118.31\ \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(7 + x + y)^{1/3}]]^3]^3 + 91.26\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 2960.64\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
9	654	0.062	$1279.84 + 92.16\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 39.10\ \mathrm{Sin}[\mathrm{Sin}[(16.53 + x + y)^{1/3}]^3]^3 - 3108.27\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + \mathrm{x} + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
10	666	0.062	$1259.10 - 118.68\ \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(7 + x + y)^{1/3}]]^3]^3 + 91.83\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 3054.86\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
11	684	0.062	$1315.91 + 92.79\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 39.79\ \mathrm{Sin}[\mathrm{Sin}[(20.53 + x + y)^{1/3}]^3]^3 - 3203.28\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$
12	696	0.062	$1292.07 - 118.14\ \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(4 + x + y)^{1/3}]]^3]^3 + 92.34\ \mathrm{Sin}[1/4 + \mathrm{Sin}[(4527.20 + y)^{1/3}]^4] - 3142.20\ (-0.19 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[4 + \mathrm{Sin}[\mathrm{Sin}[\mathrm{Sin}[(20.83 + x + y)^{1/3}]]] + \mathrm{Sin}[x^{1/3}]]]]]]]]]]]+\mathrm{Sin}[(x + 4\ y)^{1/9}] + (4514.99 + y)^{1/4})^{-0.44}$

(continued)

Table 11.15 (continued)

Page 1	Page 2		
Model selection table			
	Complexity	$1 - R^2$	Function
13	715	0.062	1348.79 + 93.32 Sin[1/4 + Sin[(4527.20 + y)$^{1/3}$]^4] −39.95 Sin[Sin[(20.53 + x + y)$^{1/3}$]^3]^3−3290.18 (−0.19 + Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[4 + Sin[Sin[Sin[(20.83 + x + y)$^{1/3}$]]] + Sin[$x^{1/3}$]]]]]]]]]]]]]]+Sin[(x + 4 y)$^{1/9}$] + (4514.99 + y)$^{1/4}$)$^{-0.44}$
14	727	0.062	1325.14 −119.54 Sin[Sin[Sin[(7 + x + y)$^{1/3}$]]^3]^3+92.90 Sin[1/4 + Sin[(4527.20 + y)$^{1/3}$]^4] −3229.41 (−0.19 + Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[4 + Sin[Sin[Sin[(20.83 + x + y)$^{1/3}$]]] + Sin[$x^{1/3}$)]]]]]]]]]]]]]]+Sin[(x + 4 y)$^{1/9}$] + (4514.99 + y)$^{1/4}$)$^{-0.44}$
15	759	0.062	1355.85 −120.00 Sin[Sin[Sin[(7 + x + y)$^{1/3}$]]^3]^3+93.41 Sin[1/4 + Sin[(4527.20 + y)$^{1/3}$]^4] −3310.62(−0.19 + Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[4 + Sin[Sin[Sin[(20.83 + x + y)$^{1/3}$]]] + Sin[$x^{1/3}$]]]]]]]]]]]]]]+Sin[(x + 4 y)$^{1/9}$] + (4514.99 + y)$^{1/4}$)$^{-0.44}$

Table 11.16 The selected model

Model selection report			
	Complexity	$1 - R^2$	Function
1	654	0.062	1279.84 + 92.16 Sin[1/4 + Sin[(4527.20 + y)$^{1/3}$]^4] −39.10 Sin[Sin[(16.53 + x + y)$^{1/3}$]^3]^3−3108.27 (−0.19 + Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[4 + Sin[Sin[Sin[(20.83 + x + y)$^{1/3}$]]] + Sin[$x^{1/3}$]]]]]]]]]]]]+Sin[(x + 4 y)$^{1/9}$] + (4514.99 + y)$^{1/4}$)$^{-.44}$

Our model in analytic form is

```
ModelPhenotype[ParetoFront[quickModels1]][[59]]
1279.84 + 92.1596 Sin[1/4 + Sin[(4527.2 + y)(1/3)]^4]
 - 39.1007 Sin[Sin[(16.5283 + x + y)^(1/3)]^3]^3
 - 3108.27/(-.188723 + (4514.99 + y)^(1/4) + Sin[(x + 4y)^(1/9)]
 + Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin[Sin
   [Sin[4 + Sin[x^(1/3)]
 + Sin[Sin[Sin[(2.8302 + x + y)^(1/3)]]]]]]]]]]]]]]]]^0.438417
```

The histogram of the relative model error in percent, see Fig. 11.28.

The histogram indicates that the ∼90% of the data points have less than 10% model error.

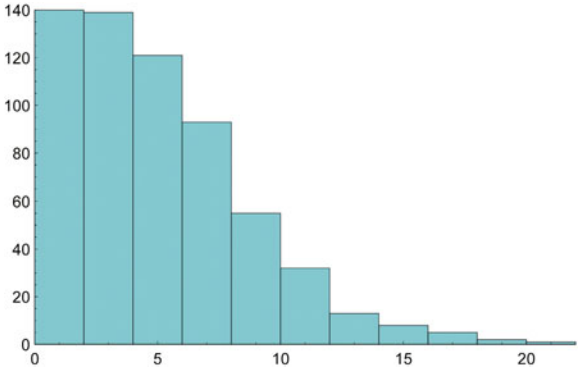


Fig. 11.28 The histogram of the model error of the selected model in %

Remark One may choose a different model having somewhat higher error but considerably lower complexity, than using this model structure, can carry out a parameter estimation, see Fig. 11.27.

References

Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: Grefenstette J.J (ed) Proceedings of the 1st international conference on genetic algorithm and their applications, Erlbaum, pp 183–187

Duquenne H, Jiang Z, Lemarie C (1995) Geoid determination and levelling by GPS: some experiments on the test network. In: IAG symposia gravity and geoid, 113. Springer pp 559–568

Ghosh (2005) Fundamentals of computational photogrammetry. Concept Publishing Company, New Delhi, pp 40–45

Heiskanen W, Moritz H (1967) Physical Geodesy. W H Freeman and Co., San Francisco

Kecman V (2001) Learning and soft computing: support vector machines, neural networks, and fuzzy logic models (complex adaptive systems). The MIT Press, Cambridge, MA, USA

Keijzer M (2003) Regression with interval arithmetic and linear scaling. In genetic programming, 6th european conference, EuroGP 2003. Springer 2610:70–82

Kenyeres A, Virág G (1998) Testing recent geoid models with GPS/levelling in Hungary. Rep. Finnish Geodetic Inst. Masala 98(4):217–223

Kotsakis C, Fotopulos G, Sideris M.G (2001) Optimal fitting of gravimetric geoid undulations to GPS/levelling data using an extended similarity transformation model. In: The 27th annual meeting of the Canadian Geophysical Union, Ottawa, Canada

Koza JR (1992) Genetic programming, On the programming of computers by means of natural selection. MIT Press, Cambridge (Massachusetts) USA

Chapter 12
Quantile Regression

12.1 Problems with the Ordinary Least Squares

In order to introduce this type of regression method, we shall consider some real world examples, where the application of traditional regression leads to misleading results. Then the definition of the quantile will be given and illustrated. With this definition, the method of quantile regression can be constructed.

12.1.1 Correlation Height and Age

Let us suppose that we want to find the correlation between height and age in a human population see Logan and Petscher (2013). It is clear that this correlation is different for teenagers than for adults; see Fig. 12.1.
The correlation between height and age is stronger in the case of teenagers than in that of adults. If we use Ordinary Least Square (OLS) we force the same correlation for the total population!

12.1.2 Engel's Problem

Engel were interested in the relationship between income and expenditures on food for a sample of working class Belgian households in 1857. The collected data are in Fig. 12.2.
Let us employ linear regression using least squares with the Euclidean metric, Ordinary Least Square (OLS). The regression line is

```
147.475 + 00.485178x
```

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_12

Fig. 12.1 Correlation between height and age in the human population

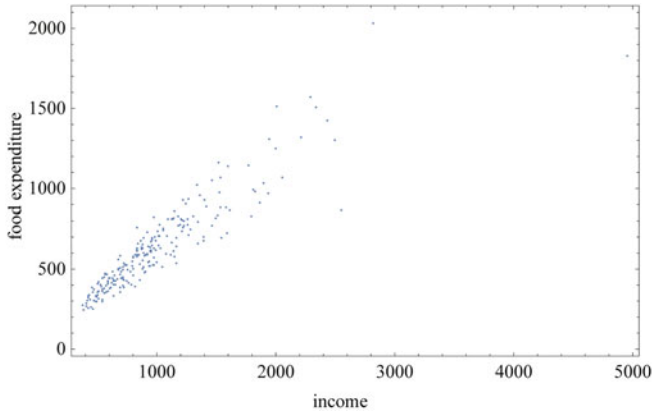


Fig. 12.2 Engel's household data

Let us visualize the fitted function with the data, see Fig. 12.3.
In order to qualify this prediction, we also visualize the *prediction error of the food expenditure*. It can be seen that this error increases with the income as well as the food expenditure. This phenomenon is called *heteroscedasticity*, see Fig. 12.4.
From studying these figures the following remarks can be made:

(1) Food expenditure increases with income,
(2) The dispersion of food expenditure increases with income (food expenditure): heteroscedasticity.
(3) The least squares estimates fit low income observations quite poorly, the OLS line passes over most low income households; see Fig. 12.3.

This latest remarks can be more clearly illustrated if we compare the predicted and the observed food expenditures; see Fig. 12.5. In case of low region of food expenditure, there is overestimation, while in high region mainly underestimation occurs.

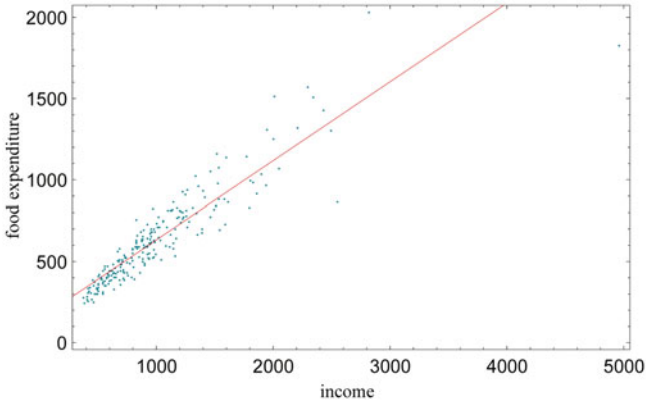


Fig. 12.3 Linear prediction via OLS

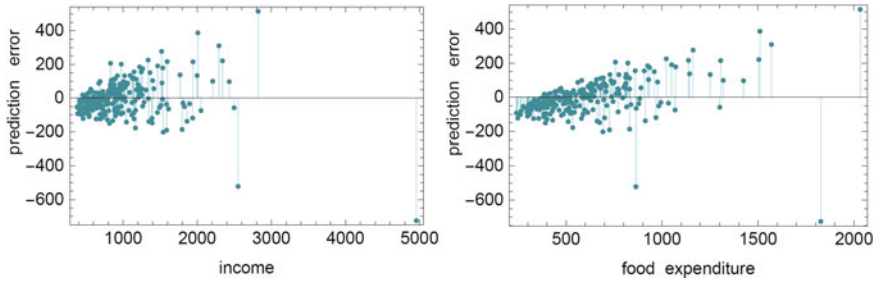


Fig. 12.4 The error of OLS as function of the income as well as that of the food expenditure

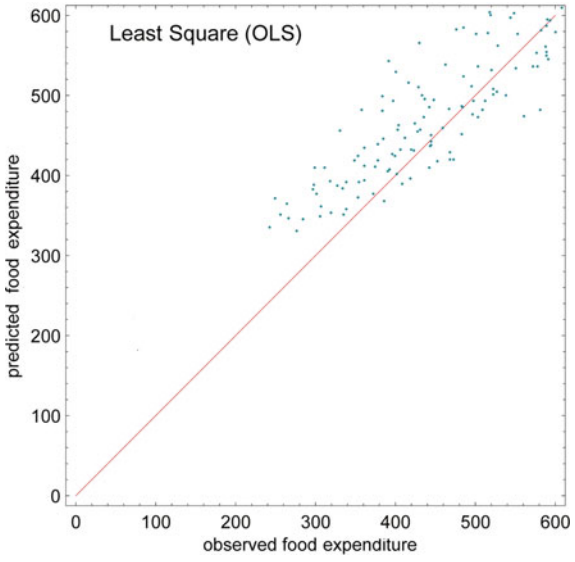


Fig. 12.5 Comparison of the predicted and observed values of the food expenditure

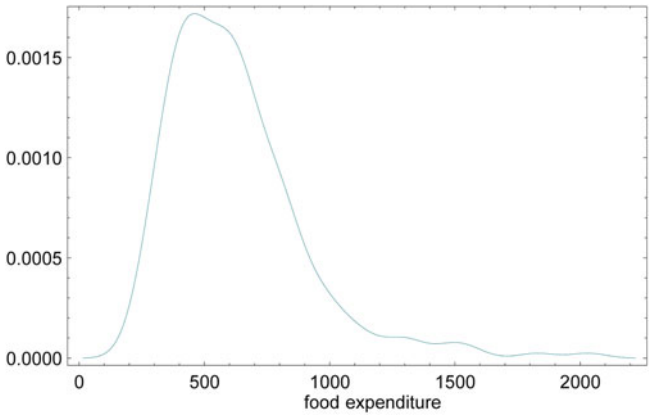


Fig. 12.6 PDF of the food expenditure is skewed to the left

Considering the Probability Density Function (PDF) of the food expenditure we also can see that the data is skewed to the left; see Fig. 12.6. Since OLS represents the correlation of the mean values, it indicates that this correlation is not representative for the households spending less or more money for food.
We should like to analyze the correlation between food expenditure and income in the region of the distribution at *lower* and at *higher probability* density *instead of the mean value region*! We are looking for the answer for the practical question: what is the effect of the income increase on the food expenditure in case of households having low income or high income, respectively? The technique which can give the answers is the *quantile regression*.

12.2 Concept of Quantile

12.2.1 Quantile as a Generalization of Median

We shall see that the extension of the definition of the *mean* and *median* value can lead to the definition of the *quantile*. Let us consider 10 000 random uniformly distributed integer numbers from the interval [−10, 10] (Fig. 12.7)

```
y = RandomReal[{-10,10},10000];Histogram[y,ImageSize → 350]
```

The *mean value* (μ_{av}) of these numbers is

$$\mu_{av} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

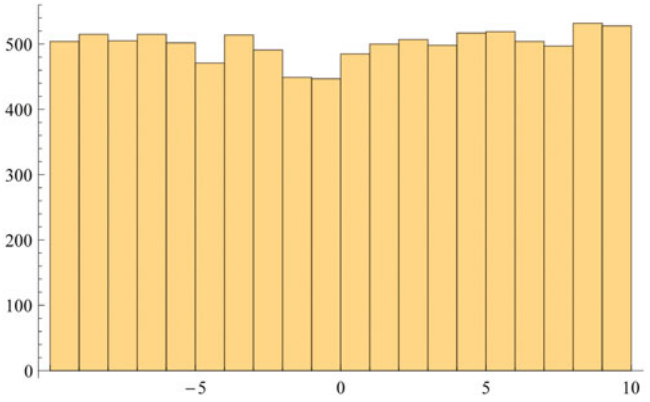


Fig. 12.7 Histogram of uniformly distributed integers from [−10, 10]

```
Mean[y]
0.0642386
```

This mean value also can be defined as the minimum of the following objective function

$$G(\mu) = \sum_{i=1}^{n} (\mu - y_i)^2.$$

namely

$$\mu_{av} = \min_{\mu}(G(\mu)).$$

```
G = Total[Map[(# - μav)^2 &, y]];
NMinimize[G, μav][[2]]
{μav → 0.0642386}
```

Similarly the *median* (μ_m), which is the "middle" value, separating the higher half of an *ordered data* sample, a population, or a probability distribution, from the lower half.

```
m = Median[y]
0.198665
```

The number of the elements being greater or equal to the median is equal with the number of the elements that are smaller than the median. Indeed, the number of the greater elements is

```
Select[y, # ≥ m&]//Length
5000
```

and that of the smaller elements is

```
Select[y, # < m&]//Length
5000
```

The median also can be computed as the solution of an optimization problem. The sum these absolute values should be minimized,

$$G(\mu) = \sum_{i=1}^{n} |(\mu - y_i)|.$$

namely

$$\mu_m = \min_{\mu}(G(\mu)).$$

```
G = Total[Map[Abs[(# - μm)]&, y]];
qm = NMinimize[G, μm][[2]]
{μm → 0.1989649}
```

This measure has a symmetrical characteristic; see the Cumulative Distribution Density Function (CDF) in Fig. 12.8.We have seen that in the data set, there are as many values bigger than the median as there are smaller. Now, we *generalize* this idea! The quantile is a generalization of the median Koenker (2005).

The quantile p of a population or data set y is a member of the data set μ_q, $(\mu_q = Q(y, p))$ so that

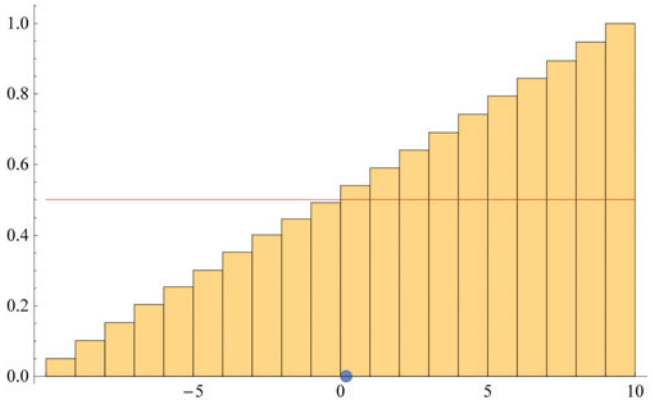


Fig. 12.8 The CDF of uniformly distributed integers with the median value (the ball)

$$\frac{\text{Length}(\{y_i : y_i > \mu_q\})}{\text{Length}(y)} = 1 - p$$

and

$$\frac{\text{Length}(\{y_i : y_i \leq \mu_q\})}{\text{Length}(y)} = p.$$

If $p = 0.5$ we get $\mu_q = \mu_m$. For example

```
p = 0.5;
Quantile[y, 0.5]
0.197367
```

So we see that the median of a set is its quantile at 0.5, namely $\mu_m = Q(0.5)$.

Now, one may ask what is the value separating the higher 80% of an *ordered data* sample, from the lower 20%? See Fig. 12.9,

```
p = 0.2;
μq = Quantile[y, p]
-6.0761
```

Indeed

```
nG = Select[y, # ≤ μq &] // Length
2000
```

```
nS = Select[y, # ≤ μq &] // Length
8000
```

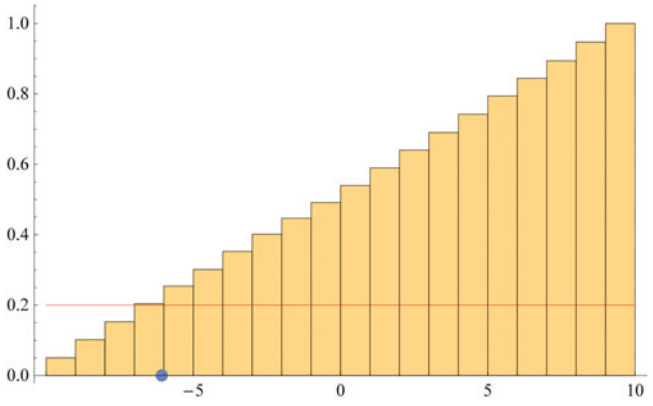


Fig. 12.9 The CDF of uniformly distributed integers with the quantile value μ_q (the ball)

The task to find the $q = Q(y, p)$ can be defined as an optimization problem, too. The objective function is

$$G(y,p,\mu) = \sum_{i\in\{i:y_i<\mu\}} (1-p)|(\mu - y_i)| + \sum_{i\in\{i:y_i\geq\mu\}} p|(\mu - y_i)|$$

and

$$\min_{\mu} G(y,p,\mu) \rightarrow \mu_q.$$

```
G = Total[Map[If[# < μq, (1 - p) Abs[# - μq], p Abs[# - μq]] &, y]];
NMinimize[G, μq][2]
{μq → - 6.07417}.
```

12.2.2 Quantile for Probability Distributions

Instead of sets of discrete values, now we consider continuous *distributions*. First, consider the *uniform distribution*.

```
U = UniformDistribution[{ - 10, 10}];
```

The probability density function is (Fig. 12.10) .
Now let us compute the value of $Q(U, 0.2)$

```
Quantile[U, 0.2]
 - 6
```

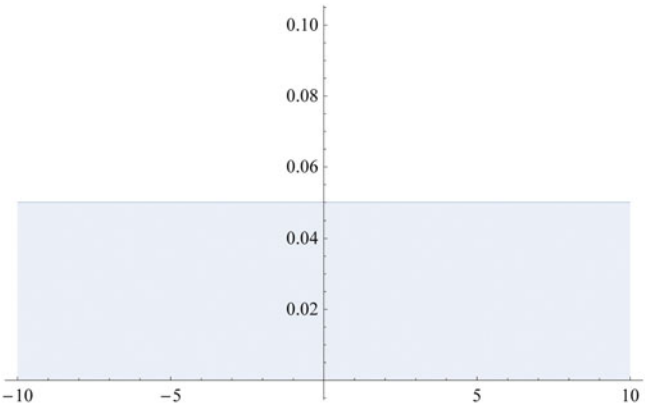


Fig. 12.10 The PDF of continuous uniform distribution over [−10, 10]

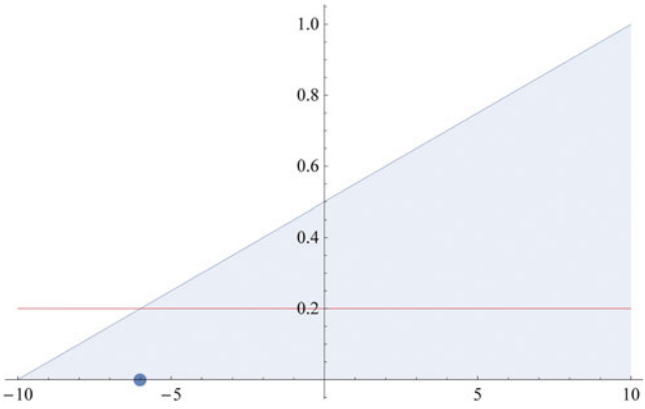


Fig. 12.11 The CDF of continuous uniform distribution over [−10, 10] with the quantile value (gray ball)

The cumulative density function is (Fig. 12.11)Indeed

```
NSolve[CDF[U,x] == p,x]//Quiet//Flatten
{x → -6.}
```

Which means, that the quantile p of the distribution D, $\mu_q = Q(D, p)$ satisfies the following relation

$$\text{CDF}\ (\mu_q) = p.$$

Namely the value of the CDF of the distribution D at μ_q is equal to p.

Now, let us see a *normal distribution* $N(\mu, \sigma) = N(1, 5)$

```
D = NormalDistribution[1,5];
```

The probability density function is (Fig. 12.12).

Calculate quantile for $p = 0.2$, namely $Q(N\ (1, 5), 0.2) = ?$

```
μq = Quantile[D,0.2]//N
-3.20811
```

Indeed, solving the equation

$$\text{CDF}\ (\mu_q) - p = 0.$$

```
μq = .//Quiet
NSolve[CDF[D,μq] == p,μq]//Quiet//Flatten
{μq → -3.20811}
```

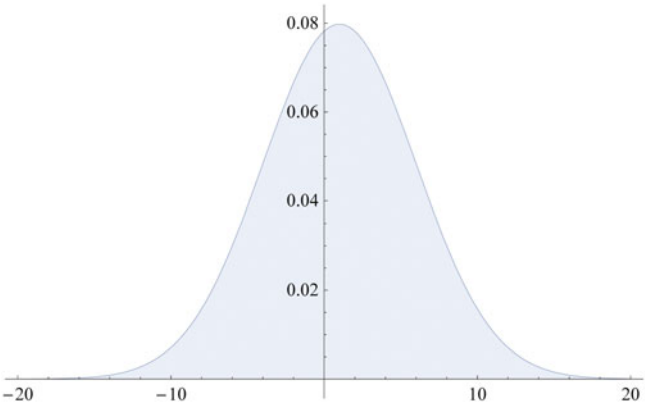


Fig. 12.12 The PDF of $N(1.5)$

Let us visualize the result in Fig. 12.13,
Now let us compute the quantiles for all of the values of $p \in [0, 1]$.

This curve is not symmetric, since the mean value is not zero, but 1. This function on Fig. 12.14 is the inverse of the function of the CDF on Fig. 12.13, see Davino et al. (2014).

12.3 Linear Quantile Regression

Let us consider a linear regression model, namely

$$y = \beta_0 + \beta_1 x.$$

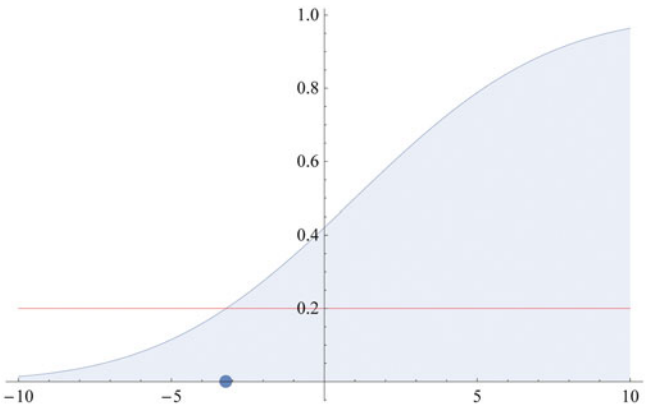


Fig. 12.13 The CDF of continuous $N(1, 5)$ distribution quantile value (the ball)

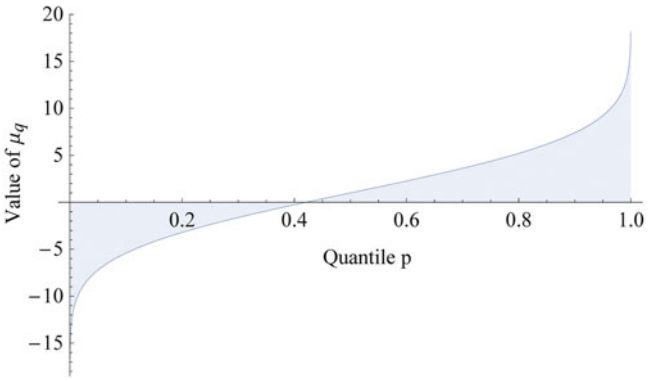


Fig. 12.14 The μ_q values of $N(1, 5)$ for $p \in [0, 1]$

12.3.1 Ordinary Least Square (OLS)

The objective function is

$$G(\beta_0, \beta_1,) = \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i - y_i)^2$$

where the data pairs of the observation are $\{x_i, y_i\}$, $i = 1, 2, \ldots, n$.

12.3.2 Median Regression (MR)

The objective function is

$$G(\beta_0, \beta_1,) = \sum_{i=1}^{n} |(\beta_0 + \beta_1 x_i - y_i)|$$

where the data pairs of the observation are $\{x_i, y_i\}$, $i = 1, 2, \ldots, n$.

Now let us compute the model parameter for *MR,*

```
G = Total[Map[Abs[β0 + β1] #[[1]] - #[[2]]]&, eng]];
NMinimize[G, {β0 + β1}][[2]]
{β0 → 81.4825, β1 → 0.56018}
```

Figure 12.15 shows some improvement of the quality of the estimation in case of the MR approach. We can also see this improvement in case of MR, when we consider the predicted food expenditure versus observed food expenditure in both cases; see Fig. 12.16.

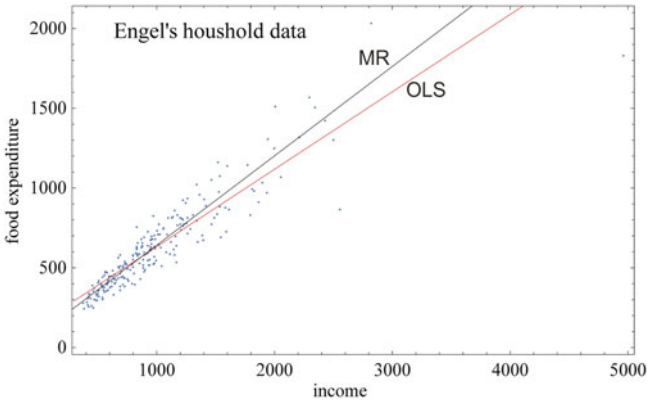


Fig. 12.15 The regression lines of the OLS and the MR approach

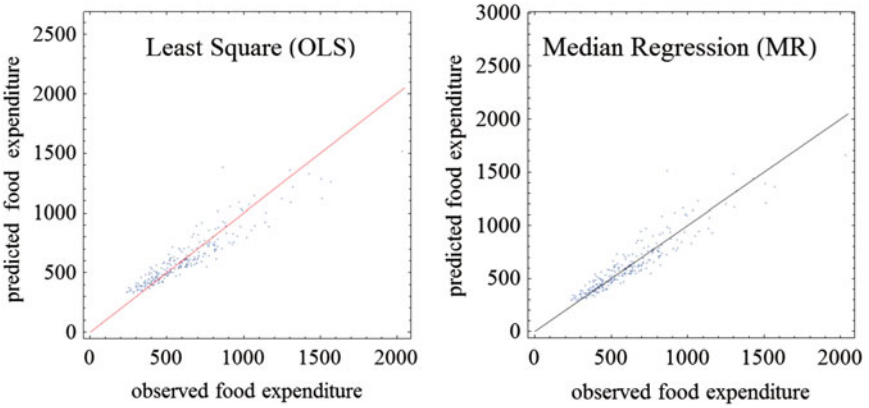


Fig. 12.16 The predicted versus observed food expenditure in case of the two different approach

The performance of the MR is better than that of the OLS in the low food expenditure region.

12.3.3 Quantile Regression (QR)

We can generalize the median regression similarly to that of the generalization of the median as quantile! The objective function is

$$G(x, y, p, \beta_0, \beta_1) = \sum_{i \in \{i: y_i < \beta_0 + \beta_1 y_i\}} (1-p)|(\beta_0 + \beta_1 x_i - y_i)| + \sum_{i \in \{i: y_i \geq \beta_0 + \beta_1 x\}} p|(\beta_0 + \beta_1 x_i - y_i)|$$

and

$$\min_{\beta_0, \beta_1}(G(x, y, p, \beta_0, \beta_1) \rightarrow \beta_0, \beta_1.$$

Let us compute the regression line for the region of data representing high food expenditure, $p = 0.95$

```
p = 0.95;
```

The result is

```
{β0 → 64.1042, β1 → 0.709068}
```

What does this regression line ($p = 0.95$) represent? Let us consider the distribution of the output (food expenditure) with the value of the food expenditure belonging to $p = 0.95$, (Fig. 12.17)

```
{X, Y} = Transpose[eng];
Quantile[Y, p]
1143.42
```

In Fig. 12.18 we see that the regression line for $p = 0.5$ represents the relation between income and food expenditure when the output data (food expenditure)

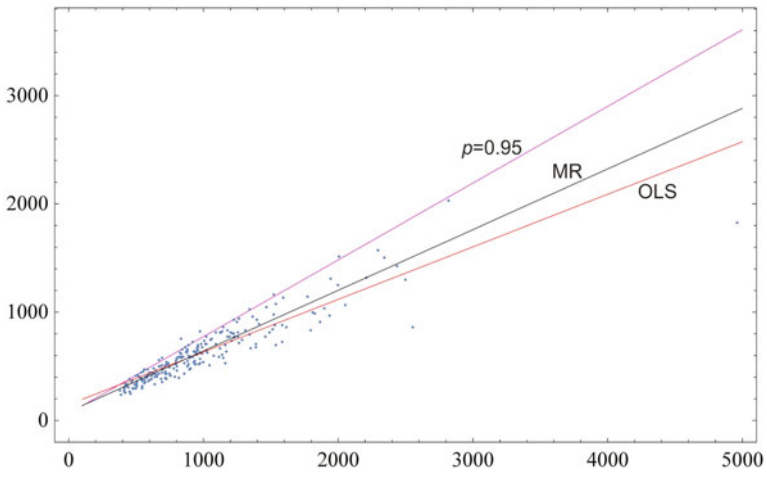


Fig. 12.17 The quantile regression line for $p = 0.95$ with the OLS and MR approach

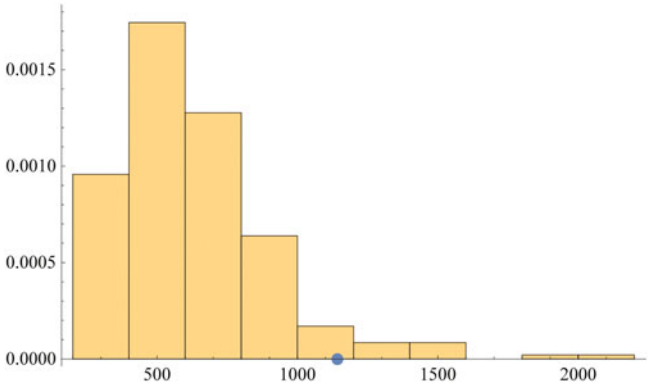


Fig. 12.18 The histogram of the output value (the food expenditure) with the quantile value 1143.42 belonging to p = 0.95

greater than or equal to 1143.42 are weighted with 0.95, and the output data less than 1143.2 are weighted with 0.05. This means that the food expenditure data greater than 1143.42 are strongly over-weighted. Therefore, this regression line essentially represents the relation between input and output at high output values!

Now let us consider a low quantile value $p = 0.05$ (Fig. 12.19).

```
p = 0.05;
Quantile[Y,p]
301
```

Now the quantile regression line represents the relation between income and food expenditure when the output data greater than or equal to 301 are weighted with 0.05, while output data less than 301 are weighted with 0.95. Thus the food expenditure data greater than 301 are strongly under-weighted. Therefore this

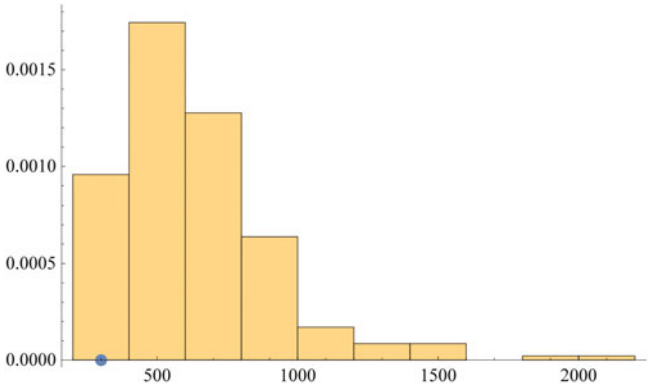


Fig. 12.19 The histogram of the output value (the food expenditure) with the quantile value 301 belonging to p = 0.05

regression line essentially represents the relation between input and output at low output values.

We can use a built-in function to compute the quantile regression for a set of p values

```
qs = {0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95};
```

The result is

```
124.88 + 0.343361 x
110.142 + 0.401766 x
95.4835 + 0.474103 x
81.4822 + 0.560181 x
62.3966 + 0.644014 x
67.3509 + 0.686299 x
64.104 + 0.709069 x
```

We also apply `Fit` to the data and the model functions in order to compare the regression quantiles with the least-squares regression fit:

```
147.475 + 0.485178 x
```

Here is a plot that combines the computed regression quantiles and the least squares fit:

For example, the line for quantile $p = 0.5$ just above the green line (OLS) represents the regression where all data are equally weighted. So this line represents the median regression (MR).

Similarly to Fig. 12.14 we can compute the slope of the regression lines belonging to the different quantiles.

This figure shows the change of the slope with the quantile value.

Figure 12.20 also can be understood as a contour plot representing a 3D graph, with food expenditure and income on the y and x axis, respectively. The third dimension arises from the *conditional probability density* of the output values, i.e., the values of the food expenditure, variable y (Fig. 12.21).

Let us consider the following figures, (Fig. 12.22).

We compute the values of the output values, y_i with the different regression lines at $x = x_0$, $y_i = y_{pi}(x_0)$. Then according to the definition of the quantile (Figs. 12.9, 12.11 and 12.13), the corresponding values $\{y_i, p_i\}$ will provide the points of CDF of the conditional probability of y, namely $y|x_0$.

Let us see an example in case of income $x_0 = 1000$.

```
x0 = 1000;
```

The function values of the regression lines at different quantiles (see Fig. 12.20)

```
yi = qrFuncs/.x → x0;
```

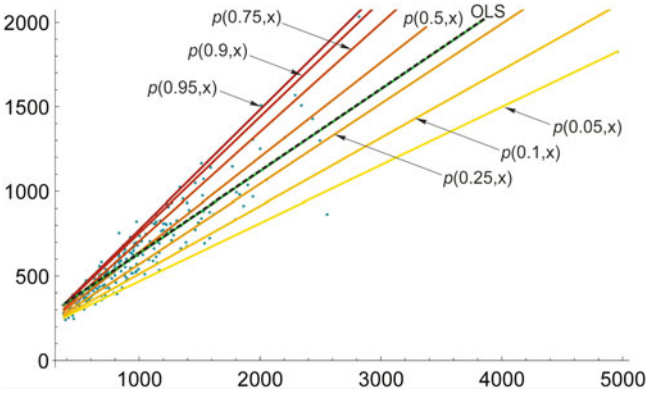
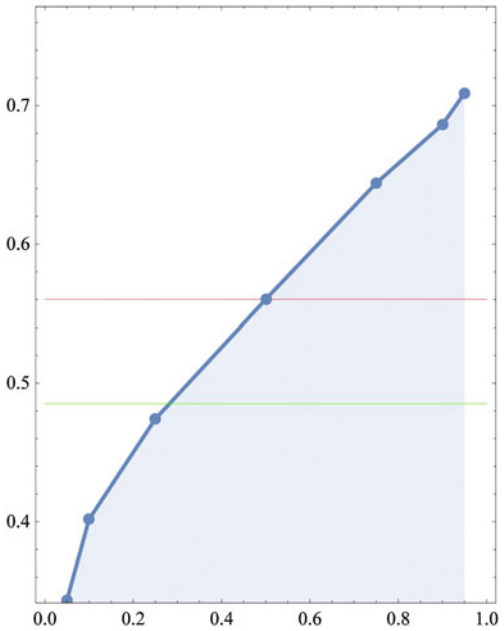


Fig. 12.20 The quantile regression lines for a set of $p \in (0, 1)$ with the OLS approach

Fig. 12.21 The slope of the regression lines as function of the quantile p



Now let us display the $\{y_i, p_i\}$ pairs (Fig. 12.23)

Since we want to also derive $PDF_{x_0}(y)$ from the estimated $CDF_{x_0}(y)$, via differentiation, it is better to use an interpolation object of $CDF_{x_0}(y)$:

Here is a plot of the derived $PDF_{x_0}(y)$ (Fig. 12.24).

These last two Figs provide more insight into the input-output relation, and they can be used for *Monte-Carlo simulation*—namely generating further artificial data points.

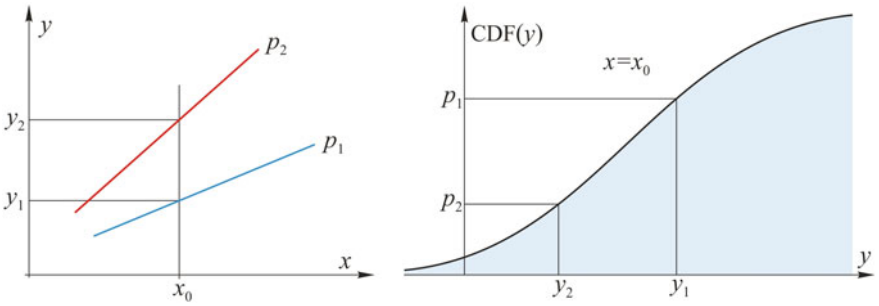


Fig. 12.22 The computation of the conditional probability density function $(y|x)$ at $x = x_0$

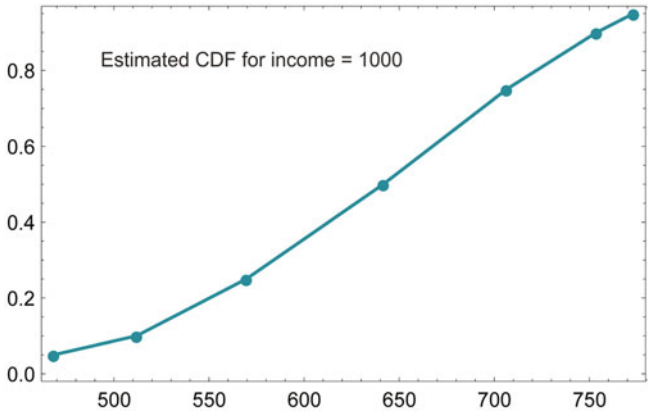


Fig. 12.23 The computed points of CDF of the conditional probability $(y|x)$ at $x = x_0 = 1000$

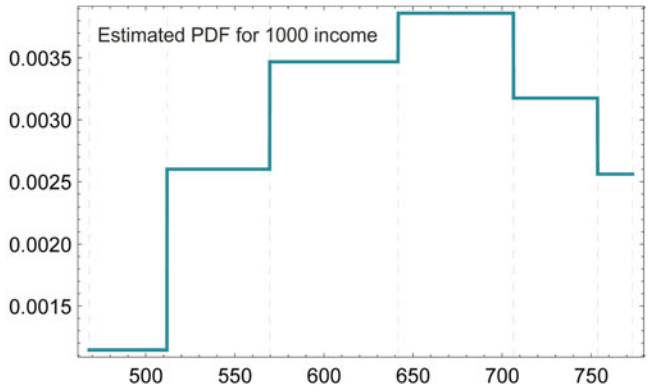


Fig. 12.24 The computed points of PDF of the conditional probability $(y|x)$ at $x = x_0 = 1000$

12.4 Computing Quantile Regression

12.4.1 Quantile Regression via Linear Programming

Employing direct global minimization is a very time consuming technique. It is desirable, therefore, to transform the optimization problem of the quantile regression into a linear programming task of a linear regression model. As we have seen the linear quantile regression can be formulated as the following optimization problem, see Koenker (2005):

$$G(x,y,p,\beta_0,\beta_1) = \sum_{i\in\{i:y_i<\beta_0+\beta_1 x_i\}} (1-p)|(\beta_0+\beta_1 x_i - y_i)| + \sum_{i\in\{i:y_i\geq\beta_0+\beta_1 x\}} p|(\beta_0+\beta_1 x_i - y_i)|$$

and

$$\min_{\beta_0,\beta_1}(G(x,y,p,\beta_0,\beta_1) \to \beta_0,\beta_1$$

In order to convert this problem into a linear programming task, let us introduce new non-negative variables u_i and v_i. Then the following linear equations as constraints can be given,

$$y_i - (\beta_0+\beta_1 x_i) + u_i = 0 \quad \text{for } i \in \{i : y_i \geq \beta_0+\beta_1 x_i\}$$

$$u_i = 0 \quad \text{for } i \notin \{i : y_i \geq \beta_0+\beta_1 x_i\}$$

$$(\beta_0+\beta_1 x_i) - y_i + v_i = 0 \quad \text{for } i \in \{i : y_i < \beta_0+\beta_1 x_i\}$$

$$v_i = 0 \quad \text{for } i \notin \{i : y_i < \beta_0+\beta_1 x_i\}$$

The linear objective function is

$$G(x,y,p,\beta_0,\beta_1,u,v) = \sum_{i\in\{i:y_i<\beta_0+\beta_1 x_i\}} (1-p)u_i + \sum_{i\in\{i:y_i\geq\beta_0+\beta_1 x_i\}} pv_i$$

which should be minimized

$$\min_{\beta_0,\beta_1,u,v}(G(x,y,p,\beta_0,\beta_1) \to \beta_0,\beta_1,u,v$$

So we have now a linear problem, but we pay for it with having a system with a large number of variables. To compute the further illustration examples, we employed the *Mathematica* functions developed by Antonov (2013).

12.4.2 Boscovich's Problem

Between 1735 and 1754 the French Academy carried out four measurements of the length of an arc of a meridian at widely different latitudes with the purpose of determining the figure of the Earth, expressed as its ellipticity. Pope Benedict XIV contributing this project commissioned *Boscovich* and *Christopher Maire* to measure an arc of the meridian near Rome and constract a new map, see Hald (2007). The five measurements appearing in Table 12.1 had been made. It was clear from these measurements that *Arc Length* was increasing as one moved towards the pole from the equator, thus qualitatively confirming Newton's conjecture that the earth's rotation could be expected to make it bulge at the equator with a corresponding flattening at the poles, yielding an oblate spheroid, "more like a grapefruit than a lemon."

But how the five measurements should be combined to produce one estimate of earth's ellipticity was unclear. The form of the approximation is suggested by Koenker (2005)

$$y = \beta_0 + \beta_1 \sin^2 \lambda,$$

where y is the *Arc Length* and λ is the latitude. Then the ellipticity could be computed as

$$\frac{1}{\text{ellipticity}} = \frac{3\beta_0}{\beta_1}$$

Now let us formulate the quantile regression problem for this data as a linear regression task. First we illustrate the solution for quantile value $p = 0.5$.

Input data for $X_i = \sin^2(\lambda_i)$ are

```
X = {0, 0.2987, 0.4648, 0.5762, 0.8386};
```

and for $y_i =$ Arc Length

```
y = {56751, 57037, 56979, 57074, 57422};
```

Table 12.1 Measured data used by Boscovich

Location (i)	λ_i = Latitude	$X_i = \sin^2$(Latitude)	y_i = arc length
Quito	0° 00′	0	56,751
Cape of good hope	33° 18′	0.2987	57,037
Rome	42° 59′	0.4648	56,979
Paris	49° 23′	0.5762	57,074
Lapland	66° 19′	0.8386	57,422

Introducing new variables

```
U = Table[u_i, {i, 1, 5}]
{u_1, u_2, u_3, u_4, u_5}
```

```
V = Table[v_i, {i, 1, 5}],
{v_1, v_2, v_3, v_4, v_5}
```

Constraints with equations are

$$56751 + u_1 - v_1 - \beta_0 = 0$$
$$57037 + u_2 - v_2 - \beta_0 - 0.2987\,\beta_1 = 0$$
$$56979 + u_3 - v_3 - \beta_0 - 0.4648\,\beta_1 = 0$$
$$57074 + u_4 - v_4 - \beta_0 - 0.5762\,\beta_1 = 0$$
$$57422 + u_5 - v_5 - \beta_0 - 0.8386\,\beta_1 = 0$$

Constraints with inequalities are

$$u_1 \geq 0$$
$$u_2 \geq 0$$
$$u_3 \geq 0$$
$$u_4 \geq 0$$
$$u_5 \geq 0$$
$$v_1 \geq 0$$
$$v_2 \geq 0$$
$$v_3 \geq 0$$
$$v_4 \geq 0$$
$$v_5 \geq 0$$

All constrains can be joined together,

$$56751 + u_1 - v_1 - \beta_0 = 0$$
$$57037 + u_2 - v_2 - \beta_0 - 0.2987\,\beta_1 = 0$$
$$56979 + u_3 - v_3 - \beta_0 - 0.4648\,\beta_1 = 0$$
$$57074 + u_4 - v_4 - \beta_0 - 0.5762\,\beta_1 = 0$$
$$57422 + u_5 - v_5 - \beta_0 - 0.8386\,\beta_1 = 0$$

$$u_1 \geq 0$$
$$u_2 \geq 0$$
$$u_3 \geq 0$$
$$u_4 \geq 0$$
$$u_5 \geq 0$$
$$v_1 \geq 0$$
$$v_2 \geq 0$$
$$v_3 \geq 0$$
$$v_4 \geq 0$$
$$v_5 \geq 0$$

The linear objective function is

$$(1-p)\,u_1 + (1-p)u_2 + (1-p)u_3 + (1-p)u_4 + (1-p)u_5 + pv_1 + pv_2 + pv_3 + pv_4 + pv_5$$

The variables of the problem,

$$\{\beta_0, \beta_1, u_1, u_2, u_3, u_4, u_5, v_1, v_2, v_3, v_4, v_5\}$$

When we minimize these we get

$$\{164.473,\ \{\beta_0 \to 56751.,\ \beta_1 \to 800.143, u_1 \to 0.,\ u_2 \to 0.,\ u_3 \to 143.907,\ u_4 \to 138.042, u_5 \to 0.,\ v_1 \to 0., v_2 \to 46.9973,\ v_3 \to 0., v_4 \to 0., v_5 \to 0.\}\}$$

Now let us compute the parameters β's for different quantile values $p \in (0, 1)$.

Figure 12.25 shows the distinct regression quantile solutions for the slope parameter, β_1.

This figure shows an interesting step-wise feature! It turns out that these steps correspond to the pairwise solution of the regression problem. For example, the interval (0, 0.21) yields as a unique solution the line passing through the towns *Quito* and *Rome*, indeed,

$$\{\{\beta_0 \to 56751.,\ \beta_1 \to 490.534\}\}$$

At $p = 0.21$ the solution jumps, and throughout the interval (0.21, 0.48), we have a solution characterized by the line passing through *Quito* and *Paris*. The process continues until we get to $p = 0.78$, where the solution through *Lapland* and *Cape of Good Hope* prevails up to $p = 1$.

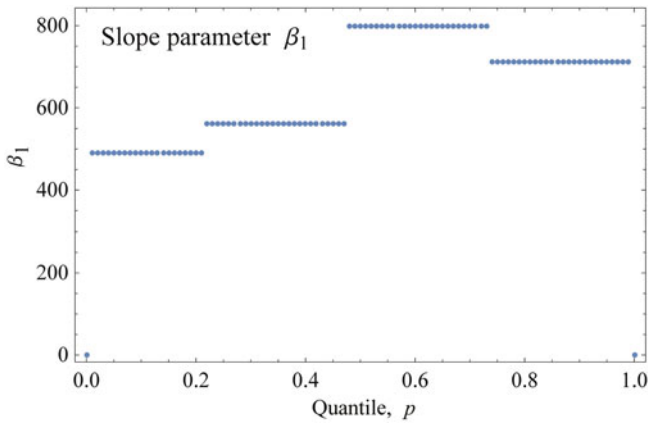


Fig. 12.25 The computed slope parameter as function of the quantile value p

The pairs of points now play the role of order statistics and serve to define the estimated linear conditional quantile functions. Again, in the terminology of linear programming, such solutions are "*basic*" and constitute extreme points of the polyhedral constraint set. If we imagine the plane represented by the objective function (F) rotating as p increases, we may visualize the solutions of F as passing from one vertex of the constraint set to another, see Fig. 12.26. Each vertex represents an exact fit of a line to *a pair of sample observations*. At a few isolated points, the plane will make contact with an entire edge of the constraint set and we will get a set-valued solution. One occasionally encounters the view that quantile regression estimators must "ignore sample information" since they are inherently determined by a small subset of the observations.

12.4.3 Extension to Linear Combination of Nonlinear Functions

The quantile regression formulation as a linear programming problem can be done for any model consisting of a linear combination of nonlinear functions, namely

$$y(x) = \sum_{i=1}^{n} \beta_i g_i(x)$$

where $g_i(x)$ nonlinear functions and β_i coefficients to be computed. As an illustration, let the data to be considered be (Fig. 12.27).
The histogram of the output values y shows anomalies of the data. Let us consider the following quantiles:

```
qs = {0.05, 0.25, 0.5, 0.75, 0.95};
```

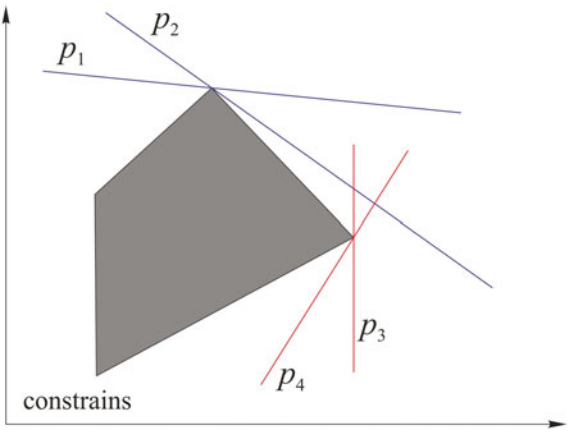


Fig. 12.26 Explanation of the stepwise solutions of the problem

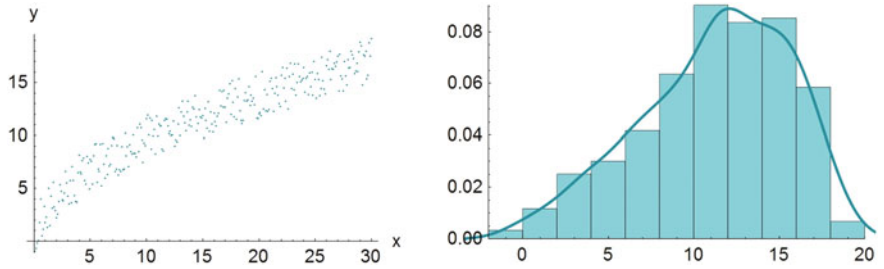


Fig. 12.27 Data for quantile regression with combination of nonlinear functions

We want to find curves that separate the data according the quantiles. Those curves are called "regression quantiles".

Pretending that we do not know how the data is generated, just by looking at the plot we assume that the model for the data is

$$y = \beta_0 + \beta_1 x + \beta_2 \sqrt{x} + \beta_3 \log(x).$$

Here we find the regression quantiles:

```
-0.675199 + 1.28437√x + 0.15953 x + 1.17151 Log[x]
-0.675199 + 2.03544√x + 0.114855 x + 0.67658 Log[x]
0.523285 + 1.37319√x + 0.170873 x + 1.26068 Log[x]
3.01222 + 4.83723*10^-7√x + 0.252894 x + 2.21582 Log[x]
1.53156 + 3.12814√x + 0.015158 x + 1.59654*10^-7 Log[x]
```

We also apply `Fit` to the data and the model functions to compare the regression quantiles with the least-squares regression fit:

Therefore the OLS results in

```
-0.522654 + 3.39992 √x - 0.0329017 x - 0.0430765 Log[x]
```

Here is a plot that combines the found regression quantiles and least squares fit (Fig. 12.28).

Let us check how good the regression quantiles are for separating the data according to the quantiles they were computed for. In case of the ith quantile (p_i), we compute the fraction of the data,

$$f_i = \sum_{k=j} y_k / \sum_{k=1}^{n} y_k, j \in \left\{ j : y_j \geq \beta_0^{(i)} + \beta_1^{(i)} x_j + \beta_2^{(i)} \sqrt{x_j} + \beta_3^{(i)} \log(x_j) \right\}.$$

and $p_i \approx f_i$ should be true (Table 12.2).

$$p_i + f_i \approx 1$$

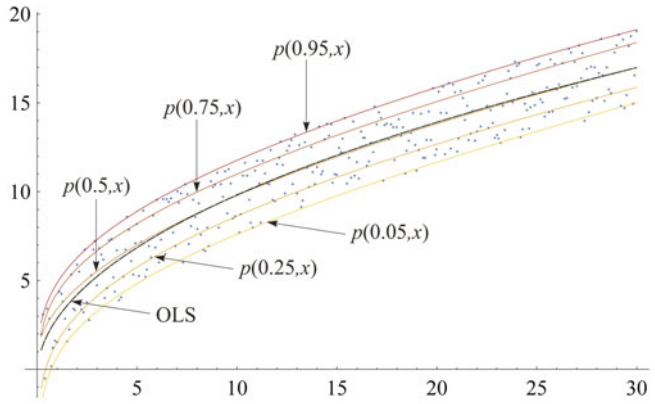


Fig. 12.28 Application of quantile regression with linear combination of nonlinear functions

Table 12.2 Quantile values (P_i) versus fractions above (P_i)

Quantile (P_i)	Fraction above (P_i)
0.05	0.949833
0.25	0.745819
0.50	0.505017
0.75	0.254181
0.95	0.046823

12.4.4 B-Spline Application

Splines, as basic functions in the linear combination, provide more flexibility of the regression line. Here we use B-splines.

We are going to consider two implementations of the Quantile Regression (QR) calculation with B-spline bases. The first implementation is based on the Linear Programming (LP) formulation of the quantile minimization problem. The second implementation is a direct translation of the non-LP minimization formulation, see Antonov (2013).

Let us see an example. We generate data with a sinusoidal function and add some random noise: (Fig. 12.29).

If the number of knots of our spline is 15, then the quantile B-spline functions can be computed as third order polynomials,

$$
\begin{aligned}
&0. + 8.95237 \begin{Bmatrix} -2752.84 + 88.4778\#1 - 0.9479088\#1^2 + 0.00338515\#1^3 & 93.34 \le \#1 \le 100.0 \\ 0 & \text{True} \end{Bmatrix} + \\
&9.19095 \begin{Bmatrix} 4954.52 - 157.88\#1 + 1.67575\#1^2 - 0.005924\#1^3 & 93.34 \le \#1 \le 100.0 \\ -551.155 + 19.0755\#1 - 0.220068\#1^2 + 0.000846286\#1^3 & 86.68 \le \#1 < 93.34 \\ 0 & \text{True} \end{Bmatrix} + \\
&2.79210 \begin{Bmatrix} 2.04538 - 0.457239\#1 + 0.0340715\#1^2 - 0.000846286\#1^3 & 6.76 \le \#1 \le 13.42 \\ -0.0460655 + 0.470919\#1 - 0.10323\#1^2 + 0.005924\#1^3 & 0.1 \le \#1 < 6.76 \\ 0 & \text{True} \end{Bmatrix} + \\
&2.00006 \begin{Bmatrix} 0.00034128 - 0.0068566\#1 + 0.034749\#1^2 - 0.00310305\#1^3 & 0.1 \le \#1 < 6.76 \\ 4.56791 - 0.682456\#1 + 0.0339869\#1^2 - 0.000564191\#1^3 & 13.42 \le \#1 \le 20.08 \\ -1.56825 + 0.689262\#1 - 0.0682276\#1^2 + 0.00197467\#1^3 & 6.76 \le \#1 < 13.42 \\ 0 & \text{True} \end{Bmatrix} + \\
&8.82548 \begin{Bmatrix} 1364.38 - 46.3887\#1 + 0.524765\#1^2 - 0.00197467\#1^3 & 86.68 \le \#1 < 93.34 \\ -289.082 + 10.8379\#1 - 0.13544\#1^2 + 0.000564191\#1^3 & 80.02 \le \#1 < 86.68 \\ -2764.87 + 86.328\#1 - 0.897097\#1^2 + 0.00310305\#1^3 & 93.34 \le \#1 \le 100.0 \\ 0 & \text{True} \end{Bmatrix} + \\
&5.71623 \begin{Bmatrix} 285.724 - 15.597\#1 + 0.282321\#1^2 - 0.00169257\#1^3 & 53.38 \le \#1 < 60.04 \\ 222.743 - 9.1089\#1 + 0.124167\#1^2 - 0.000564191\#1^3 & 66.7 \le \#1 \le 73.36 \\ -57.5353 + 3.69448\#1 - 0.079077\#1^2 + 0.000564191\#1^3 & 46.72 \le \#1 < 53.38 \\ -446.931 + 21.0114\#1 - 0.327411\#1^2 + 0.00169257\#1^3 & 60.04 \le \#1 < 66.70 \\ 0 & \text{True} \end{Bmatrix} + \\
&4.11158 \begin{Bmatrix} 193.87 - 12.0617\#1 + 0.248504\#1^2 - 0.00169257\#1^3 & 46.72 \le \#1 < 53.38 \\ 167.419 - 7.53007\#1 + 0.112895\#1^2 - 0.000564191\#1^3 & 60.04 \le \#1 \le 66.70 \\ -36.2709 + 2.71625\#1 - 0.0678045\#1^2 + 0.000564191\#1^3 & 40.06 \le \#1 < 46.72 \\ -321.018 + 16.8755\#1 - 0.293594\#1^2 + 0.00169257\#1^3 & 53.38 \le \#1 < 60.04 \\ 0 & \text{True} \end{Bmatrix} + \\
&7.71789 \begin{Bmatrix} 402.622 - 19.5827\#1 + 0.316139\#1^2 - 0.00169257\#1^3 & 60.04 \le \#1 < 66.70 \\ 289.082 - 10.8379\#1 + 0.13544\#1^2 - 0.000564191\#1^3 & 73.36 \le \#1 \le 80.02 \\ -85.8147 + 4.82286\#1 - 0.0903495\#1^2 + 0.000564191\#1^3 & 53.38 \le \#1 < 60.04 \\ -601.889 + 25.5977\#1 - 0.361229\#1^2 + 0.00169257\#1^3 & 66.7 \le \#1 < 73.36 \\ 0 & \text{True} \end{Bmatrix} + \\
&2.60176 \begin{Bmatrix} 16.908 - 2.425\#1 + 0.113233\#1^2 - 0.00169257\#1^3 & 20.08 \le \#1 < 26.74 \\ 36.2709 - 2.71625\#1 + 0.0678045\#1^2 - 0.000564191\#1^3 & 33.4 \le \#1 \le 40.06 \\ -1.36359 + 0.304826\#1 - 0.0227143\#1^2 + 0.000564191\#1^3 & 13.42 \le \#1 < 20.08 \\ -47.8154 + 4.83642\#1 - 0.158323\#1^2 + 0.00169257\#1^3 & 26.74 \le \#1 < 33.40 \\ 0 & \text{True} \end{Bmatrix} +
\end{aligned}
$$

$$
4.52253 \begin{cases} 124.062 - 8.97682\#1 + 0.214686\#1^2 - 0.00169257\#1^3 & 40.06 \le \#1 < 46.72 \\ 122.109 - 6.10139\#1 + 0.101622\#1^2 - 0.000564191\#1^3 & 53.38 \le \#1 \le 60.04 \\ -21.0216 + 1.88817\#1 - 0.0565319\#1^2 + 0.000564191\#1^3 & 33.4 \le \#1 < 40.06 \\ -221.15 + 13.19\#1 - 0.259776\#1^2 + 0.00169257\#1^3 & 46.72 \le \#1 < 53.38 \\ 0 & \text{True} \end{cases} +
$$

$$
6.45381 \begin{cases} 933.587 - 34.2426\#1 + 0.417592\#1^2 - 0.00169257\#1^3 & 80.02 \le \#1 < 86.68 \\ 564.191 - 16.9257\#1 + 0.169257\#1^2 - 0.000564191\#1^3 & 93.34 \le \#1 \le 100.0 \\ -222.743 + 9.1089\#1 - 0.124167\#1^2 + 0.000564191\#1^3 & 73.36 \le \#1 < 80.02 \\ -1271.03 + 42.0595\#1 - 0.462682\#1^2 + 0.00169257\#1^3 & 86.68 \le \#1 < 93.34 \\ 0 & \text{True} \end{cases} +
$$

$$
4.61042 \begin{cases} 38.581 - 4.15849\ \#1 + 0.147051\ \#1^2 - 0.00169257\ \#1^3 & 26.74 \le \#1 < 33.40 \\ 57.5353 - 3.69448\ \#1 + 0.079077\ \#1^2 - 0.000564191\ \#1^3 & 40.06 \le \#1 \le 46.72 \\ -4.56791 + 0.682456\ \#1 - 0.0339869\ \#1^2 + 0.000564191\ \#1^3 & 20.08 \le \#1 < 26.74 \\ -87.5485 + 7.17051\ \#1 - 0.192141\ \#1^2 + 0.00169257\ \#1^3 & 33.40 \le \#1 < 40.06 \\ 0 & \text{True} \end{cases} +
$$

$$
2.06603 \begin{cases} 0.697149 - 0.309369\ \#1 + 0.0455979\ \#1^2 - 0.00169257\ \#1^3 & 6.76 \le \#1 < 13.42 \\ 10.7872 - 1.21024\ \#1 + 0.0452594\ \#1^2 - 0.000564191\ \#1^3 & 20.08 \le \#1 \le 26.74 \\ -5.642 \times 10^{-7} + 0.000017\ \#1 - 0.0001693\ \#1^2 + 0.000564\ \#1^3 & 0.1 \le \#1 < 6.76 \\ -7.48439 + 1.51959\ \#1 - 0.090688\ \#1^2 + 0.00169257\ \#1^3 & 13.42 \le \#1 < 20.08 \\ 0 & \text{True} \end{cases} +
$$

$$
1.66144 \begin{cases} 5.28007 - 1.14196\ \#1 + 0.0794155\ \#1^2 - 0.00169257\ \#1^3 & 13.42 \le \#1 < 20.08 \\ 21.0216 - 1.88817\ \#1 + 0.0565319\ \#1^2 - 0.000564191\ \#1^3 & 26.74 \le \#1 \le 33.40 \\ -0.174287 + 0.077346\ \#1 - 0.011442\ \#1^2 + 0.000564191\ \#1^3 & 6.76 \le \#1 < 13.42 \\ -22.1274 + 2.95278\ \#1 - 0.124506\ \#1^2 + 0.00169257\ \#1^3 & 20.08 \le \#1 < 26.74 \\ 0 & \text{True} \end{cases} +
$$

$$
5.93117 \begin{cases} 73.2991 - 6.34243\#1 + 0.180868\#1^2 - 0.00169257\#1^3 & 33.4 \le \#1 < 40.06 \\ 85.8147 - 4.82286\#1 + 0.0903495\#1^2 - 0.000564191\#1^3 & 46.72 \le \#1 \le 53.38 \\ -10.7872 + 1.21024\#1 - 0.0452594\#1^2 + 0.000564191\#1^3 & 26.74 \le \#1 < 33.40 \\ -144.327 + 9.95505\#1 - 0.225958\#1^2 + 0.00169257\#1^3 & 40.06 \le \#1 < 46.72 \\ 0 & \text{True} \end{cases} +
$$

$$
6.35067 \begin{cases} 723.553 - 28.9055\#1 + 0.383774\#1^2 - 0.00169257\#1^3 & 73.36 \le \#1 < 80.02 \\ 458.806 - 14.7463\#1 + 0.157985\#1^2 - 0.000564191\#1^3 & 86.68 \le \#1 \le 93.34 \\ -167.419 + 7.53007\#1 - 0.112895\#1^2 + 0.000564191\#1^3 & 66.7 \le \#1 < 73.36 \\ -1010.94 + 36.1218\#1 - 0.428864\#1^2 + 0.00169257\#1^3 & 80.02 \le \#1 < 86.68 \\ 0 & \text{True} \end{cases} +
$$

$$
8.01989 \begin{cases} 547.565 - 24.0189\#1 + 0.349956\#1^2 - 0.00169257\#1^3 & 66.7 \le \#1 < 73.36 \\ 367.437 - 12.717\#1 + 0.146712\#1^2 - 0.000564191\#1^3 & 80.02 \le \#1 \le 86.68 \\ -122.109 + 6.10139\#1 - 0.101622\#1^2 + 0.000564191\#1^3 & 60.04 \le \#1 < 66.7 \\ -788.893 + 30.6345\#1 - 0.395046\#1^2 + 0.00169257\#1^3 & 73.36 \le \#1 < 80.02 \\ 0 & \text{True} \end{cases} \&
$$

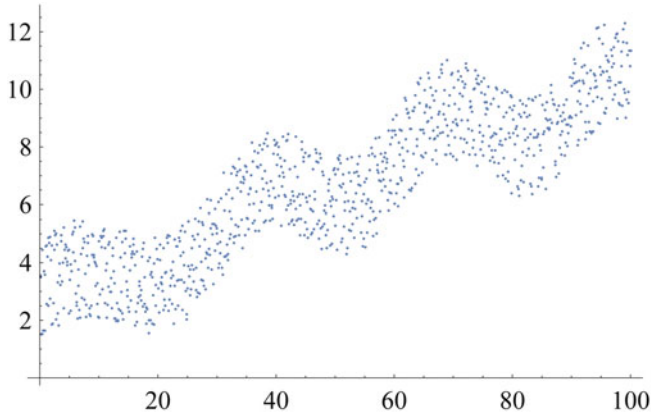


Fig. 12.29 Generated data for B-spline quantile regression

To compare them with the OLS method, here we employ trigonometric polynomials with a linear tail

$$y(x) = \sum_{i=5}^{10} \beta_i \sin\left(\frac{x}{i}\right) + \beta_1 x + \beta_0.$$

```
y = 3.03703 + 0.0711411 x - 3.90944 Sin[x/10] + 7.64373 Sin[x/9] -
 6.6894 Sin[x/8] + 3.14463 Sin[x/7] - 1.33602 Sin[x/6] + 1.14319 Sin[x/5]
```

Let us visualize the result, (Fig. 12.30).

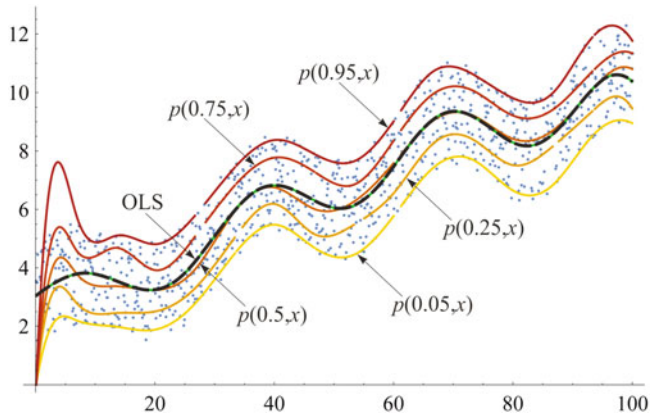


Fig. 12.30 Application of B-spline quantile regression

At small x values, $x < 20$, the values of the OLS method are somewhat worse than that of the B-spline approximation.

Let us demonstrate the robustness of the regression quantiles with the data of the example from Sect. 12.4.3, but suppose that for some reason 50% of the data y-values greater than 10.5 are altered by multiplying them with some factor greater than 1, say, $\alpha = 2.5$. Then the altered data looks like Fig. 12.31. This figure even shows us the histogram of the data on the right hand side.

```
data = Map[{#, (Sqrt[10 x]) + RandomReal[{-2.6, 1.9}]} /. x → #} &,
  Range[0.2, 30, 0.1]];
α = 2.5;
dataAlt = Map[If[RandomInteger[{0, 1}] == 1 && #[[2]] > 10.5,
  {#[[1]], α #[[2]]}, #] &, data];
```

The histogram of the output values are indicating outliers. Let us employ the following quantiles,

```
qs = {0.05, 0.25, 0.5, 0.75, 0.95};
```

The basic functions are,

```
funcs = {1, x, √x, Log[x]};
```

Let us compute the five (qs) regression quantiles for the altered data:

$$1.29453 \times 10^{-9} + 0.396145\sqrt{x} + 0.225675\,x + 1.80078\,\text{Log}[x]$$
$$2.16463\sqrt{x} + 0.14238\,x + 0.204086\,\text{Log}[x]$$
$$0.698279 + 2.49018\sqrt{x} + 0.168249\,x + 3.3233*10^{-9}\,\text{Log}[x]$$
$$1.73343 + 1.45241\,x$$
$$7.82056\sqrt{x} + 0.0819461\,x + 0.547324\,\text{Log}[x]$$

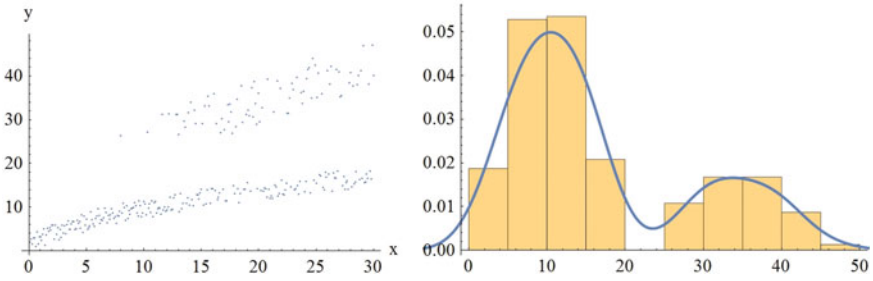


Fig. 12.31 Data generated with outliers

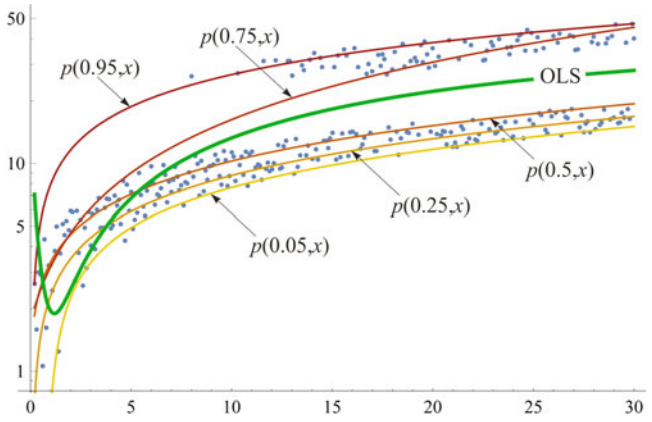


Fig. 12.32 Quantile regression in case of outliers

We also compute the least squares fit of the model consisting of the basic functions (OLS),

```
-16.1728 + 19.0888 √x - 0.958974 x - 9.26759 Log[x]
```

Here is a plot of the altered data and all fitted functions, see Fig. 12.32.
Figure 12.32 shows that regression lines with quantile $p < 0.5$ can reject outliers, while OLS provides a misleading result.

12.5 Applications

12.5.1 Separate Outliers in Cloud Points

LiDAR (Light Detection and Ranging) point clouds contain abundant spatial (3D) information. A dense distribution of scanned points on an object's surface strongly implies surface features. In particular, plane features commonly appear in a typical LiDAR dataset of artificial structures. Plane fitting is the key process for extracting plane features from LiDAR data.

Let's consider a s*ynthetic dataset* generated from a multivariate Gauss distribution. The 3D points have means of (2, 8, 6) and variances (5, 5, 0.01). The ideal plane is $z = 6$; see Fig. 12.33.

In this case let us consider $N = 20{,}000$ points.
Figure 12.33 shows the data points with the ideal plane. Now, let us generate 3000 outliers randomly, see Fig. 12.34.
Let us see the histogram of all the data points, see Fig. 12.35,

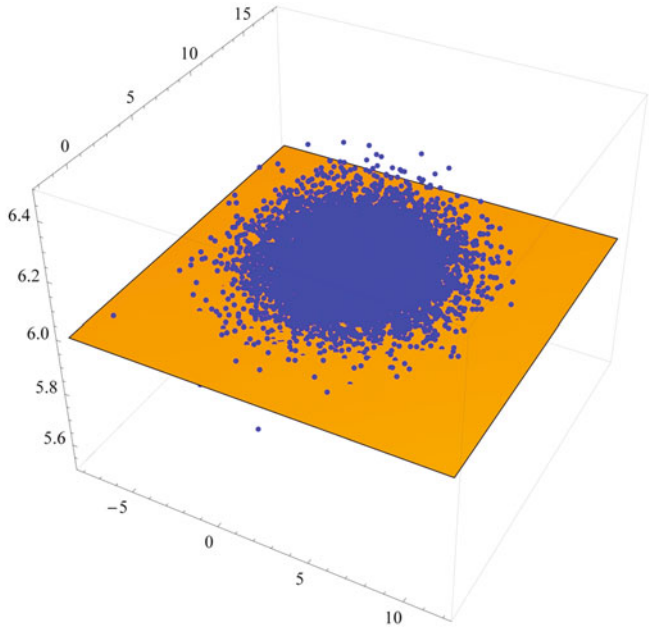


Fig. 12.33 Random point data and the ideal plane

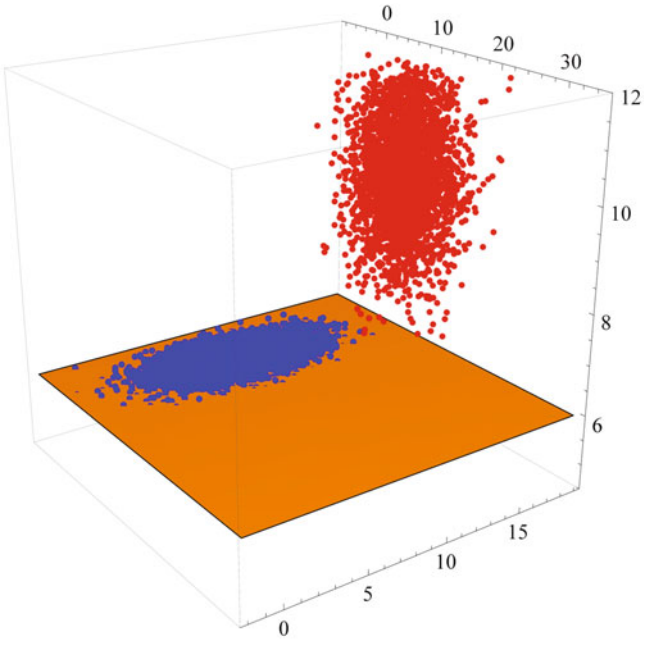


Fig. 12.34 Random point data and outliers with the ideal plane

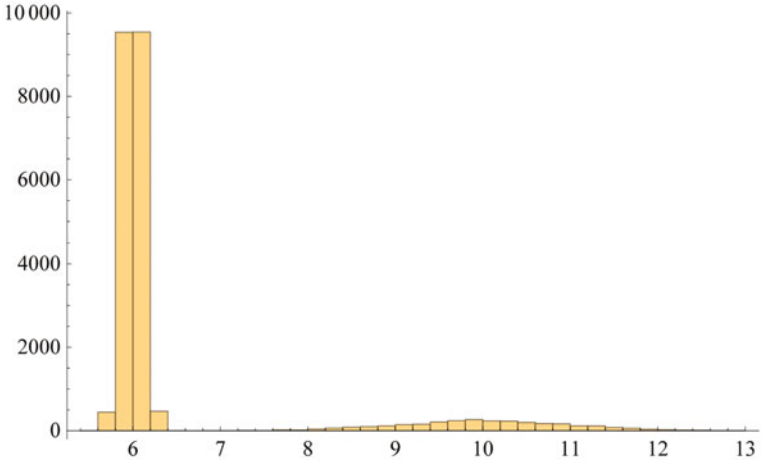


Fig. 12.35 Histogram of all data points

```
XYZC = Join[XYZ, XYZOutLiers];
```

The Fig. 12.35 indicates the anomaly caused by outliers. In order to eliminate the outlier points from the plane fitting process, let us use $p = 0.9$ quantile value (Fig. 12.36).

We get:

```
{2887.81, {α → 0.218001, β → 0.15376, γ → 5.20335}}
```

Here we used the perpendicular distance of points from the plane as the error definition

Now let us see the result of the case of $p = 0.1$.

```
p = 0.1;
```

The objective function to be minimized in order to set the plane parameters is, see Sect. 12.3.3,

```
G = Total[MapThread[If[#3 - (α#1 + β#2 + γ)/Sqrt[1 + α^2 + β^2] < 0,
  (1 - p) Abs[#3 - (α#1 + β#2 + γ)/Sqrt[1 + α^2 + β^2]], p Abs[#3 - (α#1 + β#2 + γ)/Sqrt[1 + α^2 + β^2]]] &, {X, Y, Z}]];
sol = NMinimize[G, {α, β, γ}]
{1535.13, {α → 0.0241206, β → 0.0128114, γ → 5.71022}}
```

Figure 12.37 shows the resulting plane.

How can one find the proper quantile value? Let us compute the histogram of this plane fitting using $p = 0.1$; see Fig. 12.38.

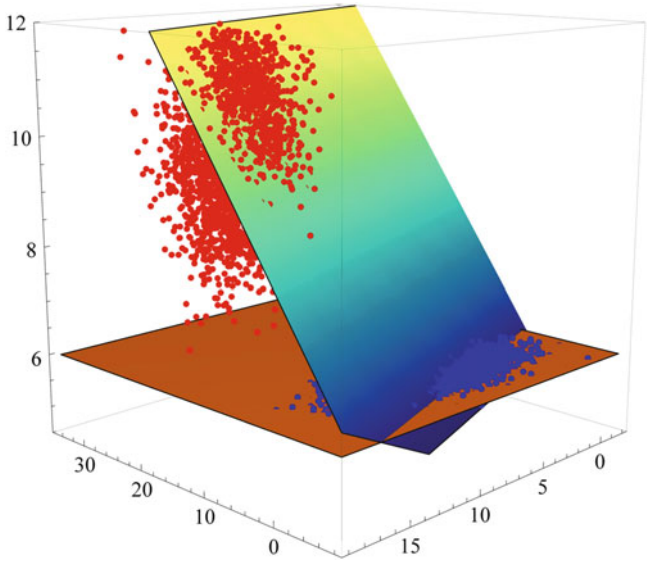
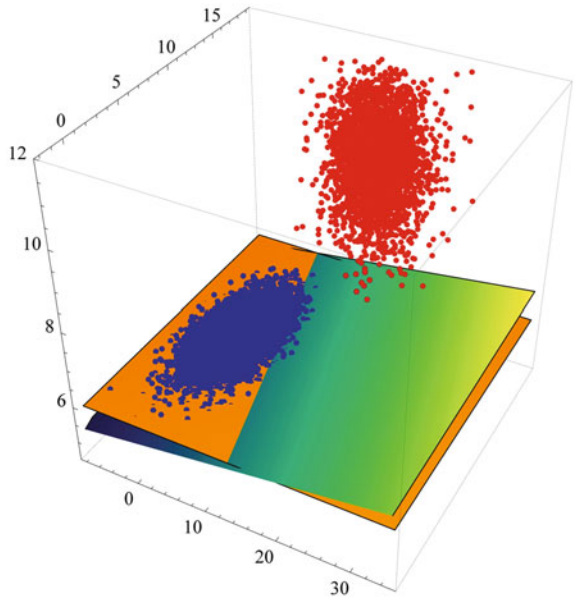


Fig. 12.36 The result of quantile regression ($p = 0.9$)

Fig. 12.37 The result of quantile regression ($p = 0.1$)



We see that the left hand side of the histogram represents the errors of the inliers. By inspection we can determine the maximum error (Abs(Error) < 0.5), or more precisely and automatically one can do it with the *Expectation Maximization*

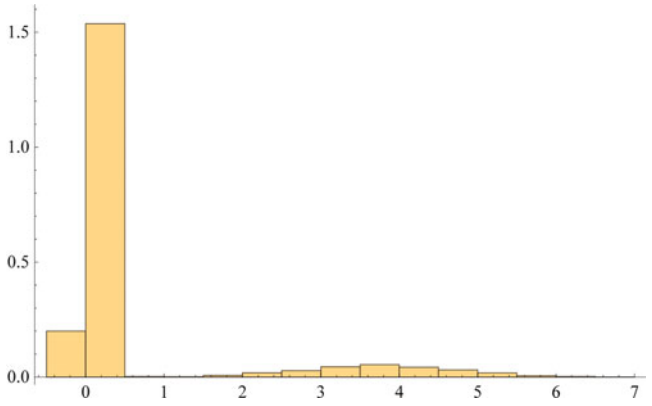


Fig. 12.38 The error histogram of the plane fitted by quantile regression ($p = 0.1$)

(EM) algorithm. Let us select data points which give smaller error for the model fitted with $p = 0.1$.

```
XYZS =
  Select[XYZC, Abs[(#[[3]] - α#[[1]] - β#[[2]] - γ)/Sqrt[1 + α^2 + β^2] /. sol[[2]]] < 0.5&];
```

In this way, most of the outliers could be eliminated. The number of points remaining is 19,979. After minimizing now we get

```
{198.172, {α → 0.000232049, β → 0.0000462067, γ → 5.99977}}
```

See Fig. 12.39.

Now, let us introduce another example with *real LiDAR measurement*. We have data points of the downhill covered by bushes as vegetation. We are going to find the slope of the hill by fitting a plane to the data points. In this context the bushes represent outliers, see Fig. 12.40.

The number of the points is 41,392. But we remove some corrupted data and have 33,292 left.

Let us see the histogram of the data in Fig. 12.41.

Employing quantile regression, we can successfully find the proper plane; see Fig. 12.42. Let

```
p = 0.05;
G = Total[MapThread[If[(#3 - (α#1 + β#2 + γ))/Sqrt[1 + α^2 + β^2] < 0,
  (1 - p) Abs[(#3 - (α#1 + β#2 + γ))/Sqrt[1 + α^2 + β^2]], p Abs[(#3 - (α#1 + β#2 + γ))/Sqrt[1 + α^2 + β^2]]]&, {X, Y, Z}]];
```

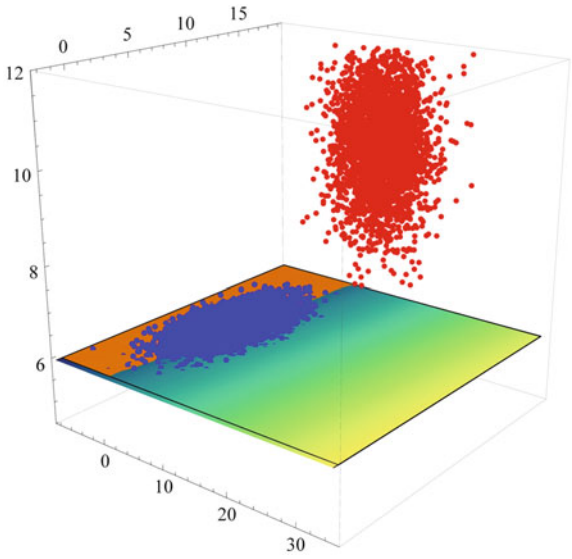


Fig. 12.39 The corrected plane via separating in- and outliers

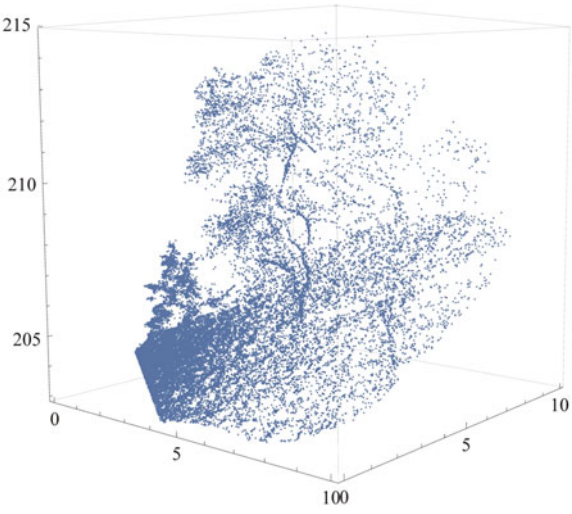


Fig. 12.40 Original LiDAR data

The parameters of the fitted plane are

$$\{1652.71, \{\alpha \to 0.107359, \beta \to 0.508726, \gamma \to 202.536\}\}$$

The fitting is quite successful, see Fig. 12.42.

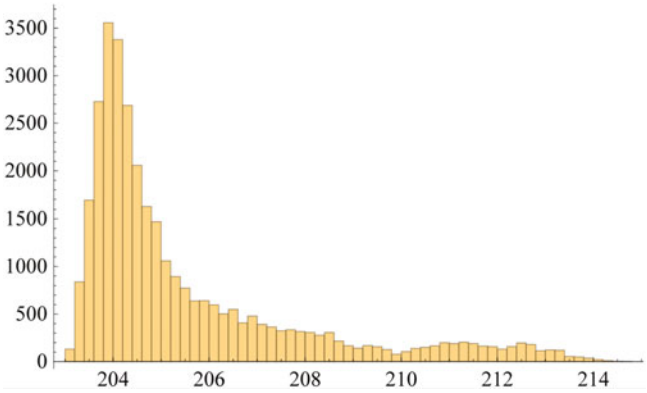


Fig. 12.41 Histogram of the output data

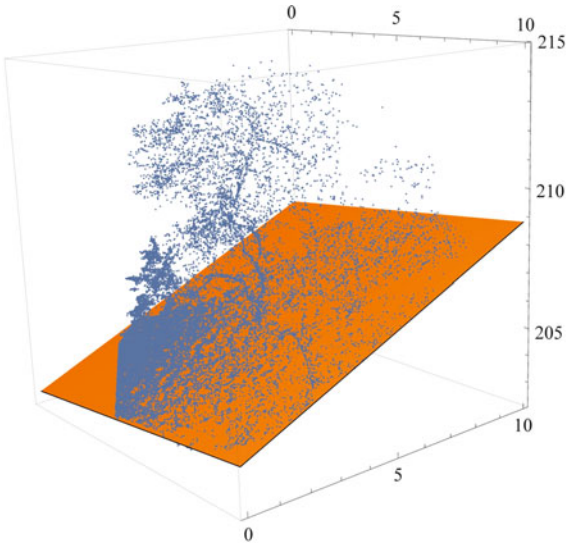


Fig. 12.42 Estimation of the plane by quantile regression with $p = 0.05$

12.5.2 Modelling Time-Series

Let us consider an application of the quantile regression to time series. Here we employ the daily temperature data of Melbourne from 1 Jan 2010 up to 21 Dec 2014, see Fig. 12.43.

First we are looking for the conditional (at a certain day) cumulative density function of the daily temperature. For example we should like to estimate the probability that temperature is less than a given value, or between two given values. Let us consider the following quantile values:

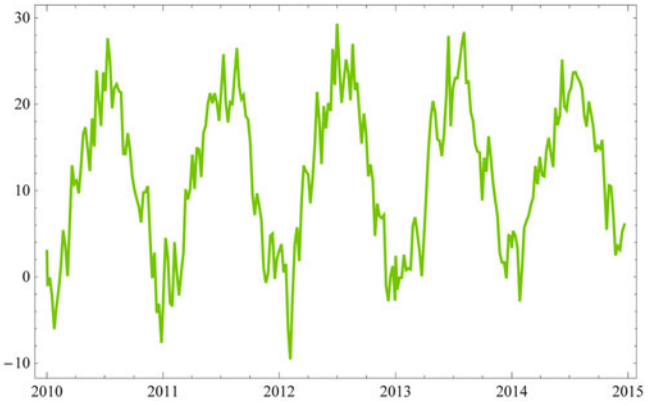


Fig. 12.43 Daily temperature history in Melbourne in a five year period

```
qs = {0.05, 0.25, 0.5, 0.75, 0.95};
```

Let us carry out the quantile regression with B-splines having 20 knots.

Plot regression quantiles and time series data; see Fig. 12.44.

Let us select a day, $i = 123$

```
x0 = 123;
yi = tempq[[x0]];
```

Then using the five different quantile values, we get the p1-temperature diagram, which is the discrete points of the CDF of the temperature as condition (i = 123) random variable; see Fig. 12.45.

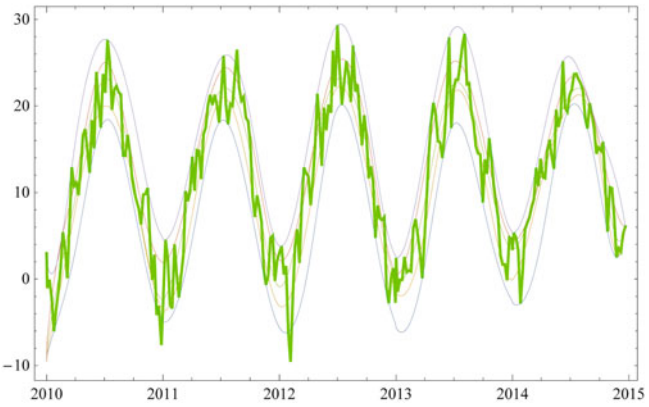


Fig. 12.44 Daily temperature history in Melbourne in a five year period, with the five different quantile B-Splines

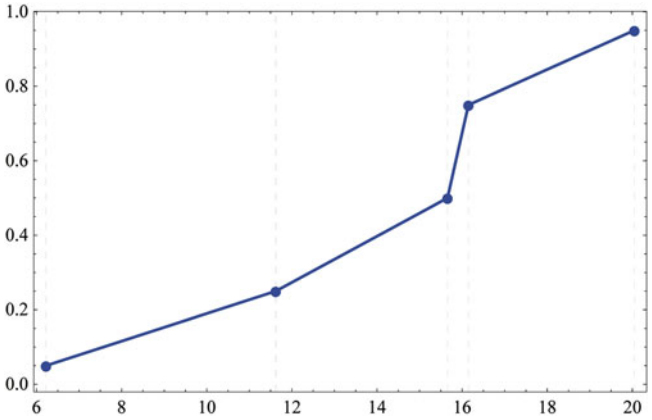


Fig. 12.45 The estimated CDF of the daily temperature in Melbourne at the 123rd day

Let us employ linear interpolation for this discrete CDF,

```
tempq[[123]]
{6.22348,11.6218,15.6562,16.1488,20.0468}
```

Then for example the average temperature can be computed as

```
NSolve[qCDFInt[x] = = 0.5,x]//Quiet
{{x → 15.6562}}
```

The probability that the temperature will be less than temperature is 6.22 is

```
tempq[[123,1]]
6.22348
```

or

```
qCDFInt[%]
0.05
```

The probability that the temperature will be between

```
tempq[[123,1]]
6.22348
```

and

```
tempq[[123,2]]
11.6218
```

is

```
qCDFInt[%] - qCDFInt[%%]
0.2
```

and so on. We can visualize the PDF too, via differentiating CDF; see Fig. 12.46. Secondly, we consider a more difficult forecasting problem. Assume that we have temperature data for a given location. We want to predict today's temperature at that location using yesterday's temperature. More generally, the problem discussed in this section can be stated as "How to estimate the conditional density of the predicted variable given a value of the conditioning covariate?" see *McMillen* (2012).

One way to answer this question is to provide a family of regression quantiles and from them to estimate the Cumulative Distribution Function (CDF) for a given value of the covariate. In other words, given data pairs $\{x_i, y_i\}_{i=1}^{n}$ and a value x_0 we find $\mathrm{CDF}_{x_0}(y)$. From the estimated CDF we can estimate the Probability Density Function (PDF). We can go further and use a Monte Carlo type of simulation with the obtained PDF's (this will be discussed elsewhere).

Using the time series data let us make pairs of yesterday and today data. The resulting pairs are in Fig. 12.47.
Let us use quantile regression with the following quantile values, qsi

```
qs = Join[{0.02}, FindDivisions[{0, 1}, 10][[2;; - 2]], {0.98}]//N
{0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.98}
```

We apply $n = 11$ B-spline with 5 knots, (Fig. 12.48).
Given yesterday's temperature value, let us say $t_0 = 5$ °C, we can estimate $\mathrm{CDF}_{t_0}(t)$ using the values of the quantile at t_0 of the corresponding regression quantile functions.

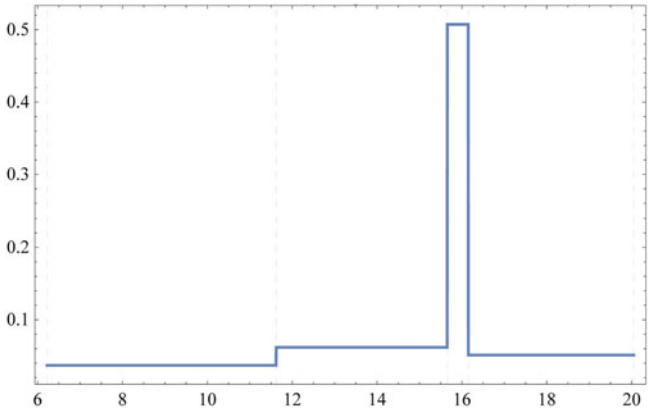


Fig. 12.46 The estimated PDF of the daily temperature in Melbourne at the 123rd day

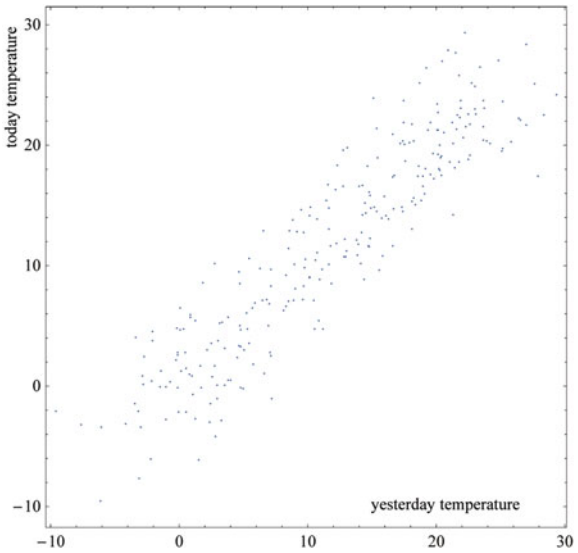


Fig. 12.47 Correlation of the temperatures of two consecutive days

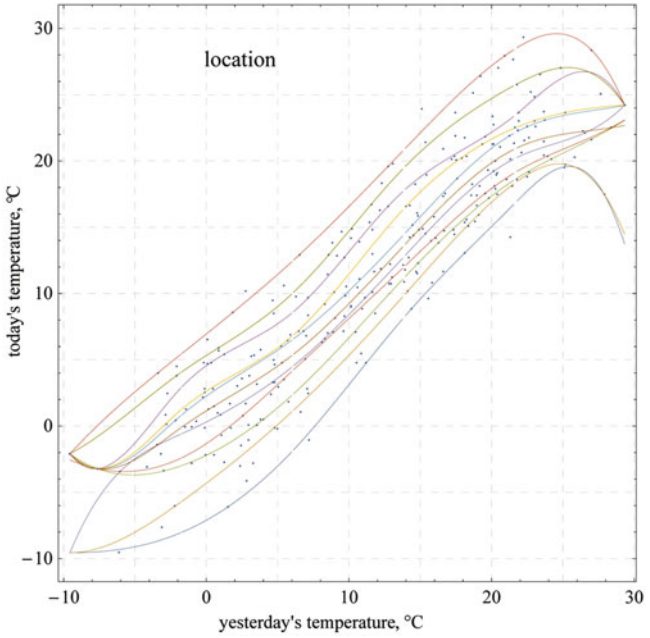


Fig. 12.48 Regression lines of different quantiles

```
t0 = 5;
xs = Through[qFuncs[t0]];
```

We can get a first order (linear) approximation by simply connecting the consecutive points of $\{x_i^{t_0}, q_i\}_{i=1}^{|qs|}$ as it is shown on the following plot, Fig. 12.49.
Since we want to also derive $\mathrm{PDF}_{t_0}(t)$ from the estimated $\mathrm{CDF}_{t_0}(t)$, make plots, and do other manipulations, it is better to make and use an interpolation object for $\mathrm{CDF}_{t_0}(t)$.

```
qCDFInt = Interpolation[cdfPairs, InterpolationOrder
  → 1];
```

Here is a plot with both $\mathrm{CDF}_{t_0}(t)$ and the $\mathrm{PDF}_{t_0}(t)$ derived from it; see Fig. 12.50.

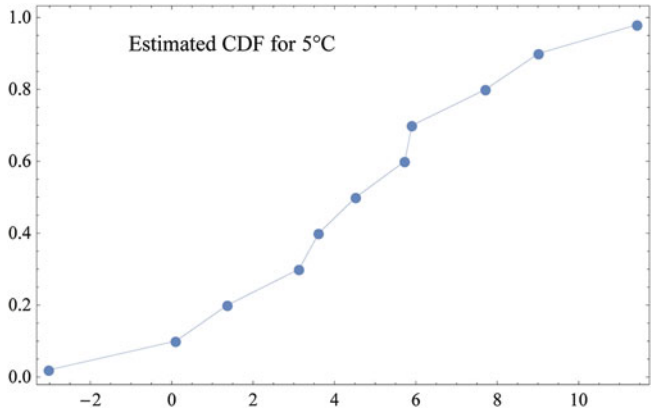


Fig. 12.49 Constructed conditional CDF of the today temperature based on the yesterday temperature of 5 °C

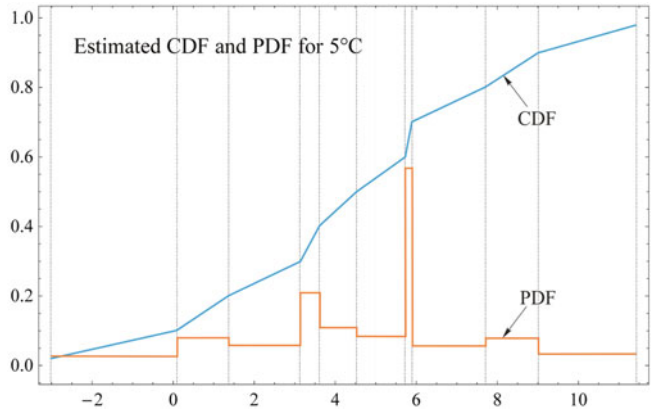


Fig. 12.50 Constructed conditional PDF and CDF of the today temperature based on the yesterday temperature of 5 °C

On the plot the dashed vertical grid lines are for the quantiles {0.05, 0.25, 0.5, 0.75, 0.95}; the solid vertical gray grid line is for t_0.

Using this technique let us plot the estimated CDF's and PDF's for a collection of temperatures. Note that the quantiles are given with vertical dashed grid lines; the median is colored with green. The values of the conditioning covariate are given on the plot labels (Figs. 12.51 and 12.52).

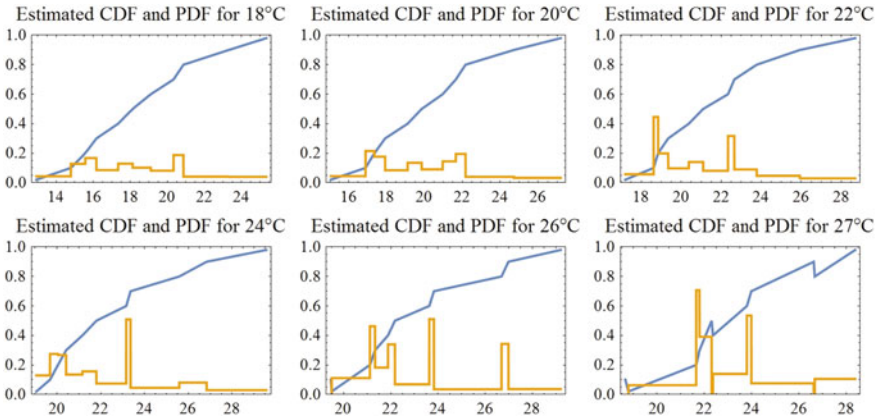


Fig. 12.51 Conditional CDF and PDF for different daily temperatures

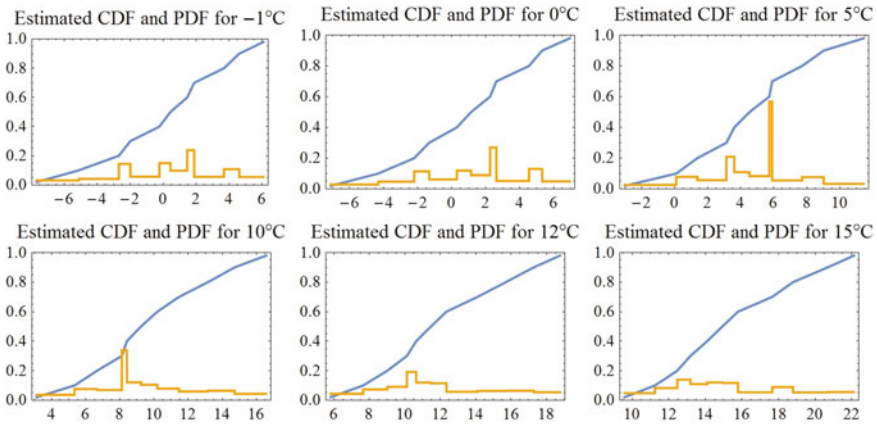


Fig. 12.52 Constructed conditional CDF and PDF of the today temperature based on different yesterday temperatures

12.6 Exercise

12.6.1 Regression of Implicit-Functions

Let us consider the following point cloud; see Fig. 12.53.

```
npoints = 12000;
data = {RandomReal[{0, 2 π}, npoints],
 RandomVariate[SkewNormalDistribution[0.6, 0.3, 4], npoints]};
data = MapThread[#2*{Cos[#1], Sin[#1]}&, data];
rmat = RotationMatrix[ - π/2.5].DiagonalMatrix[{2, 1}];
data = Transpose[rmat.Transpose[data]];
data = TranslationTransform[{ - Norm[Abs[#]/3], 0}][#]&/@data;
sndData = Standardize[data];
hup1 = ListPlot[sndData, PlotRange → All, AspectRatio → 1,
 PlotTheme → " Detailed", GridLines → Map[Quantile[#,
 Range[0, 1, 1/19]]&, Transpose[sndData]], ImageSize → 300]
```

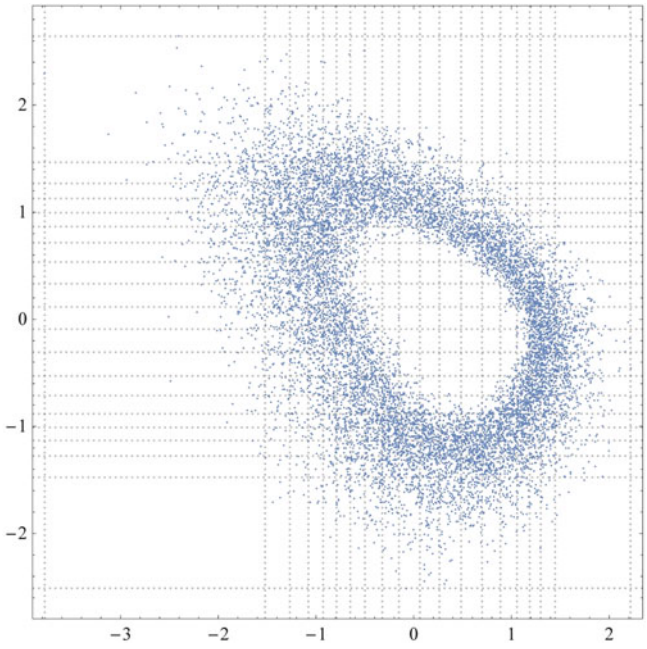


Fig. 12.53 Point cloud may be represented by implicit function

This cloud of points cannot be represented by a single explicit function. Let us try to carry out quantile regression with B-splines having 15 knots. The quantile values are:

```
qs = Range[0.05, 0.95, 0.1]
{0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95}

AbsoluteTiming[
 qFuncs = QuantileRegression[sndData, 15, qs,
 Method → {LinearProgramming, Method →" CLP"}];
]
```

Let us visualize the results (Fig. 12.54).

```
SS = Through[qFuncs[x]];
hup2 = Plot[Evaluate[SS], {x, - 2.1, 2}, PlotPoints → 130,
 ImageSize → 300, AspectRatio → 1];
Show[{hup2, hup1}]
```

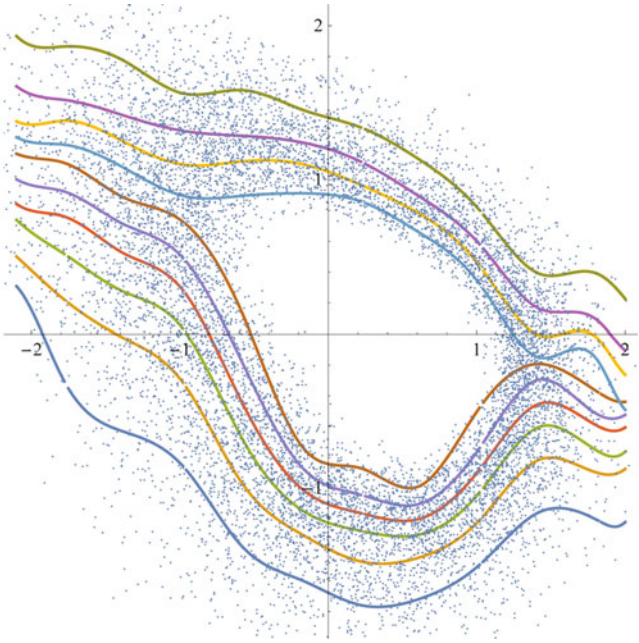


Fig. 12.54 Point cloud represented by explicit functions (B-Splines)

Directional quantile envelope technique

However we can employ a so called *directional quantile envelope* technique, see Antonov (2013). The idea of directional quantile envelopes is conceptually simple and straightforward to derive. The calculation is also relatively simple: over a set of uniformly distributed directions we find the lines that separate the data according to a quantile parameter q, $0 < q < 1$, and with those lines we approximate the enveloping curve for data that corresponds to q. The quantile regression is carried out from different angles using rotation of the data; see Fig. 12.55.

Now let us employ our function. The quantiles are

```
qs = {0.7, 0.75, 0.8, 0.85, 0.90, 0.95, 0.98, .99};
```

One can use the quantile regression function with the option, see Antonov (2013)

```
Options [QuantileEnvelope]
{Tangents → True}
```

```
?QuantileEnvelope
```

```
QuantileRegression[data_?MatrixQ,qs:(_?NumberQ|{_?NumberQ..}),n_Integer]
   experimental implementation of quantile envelopes points finding.
```

Let us visualize the results in Fig. 12.56.

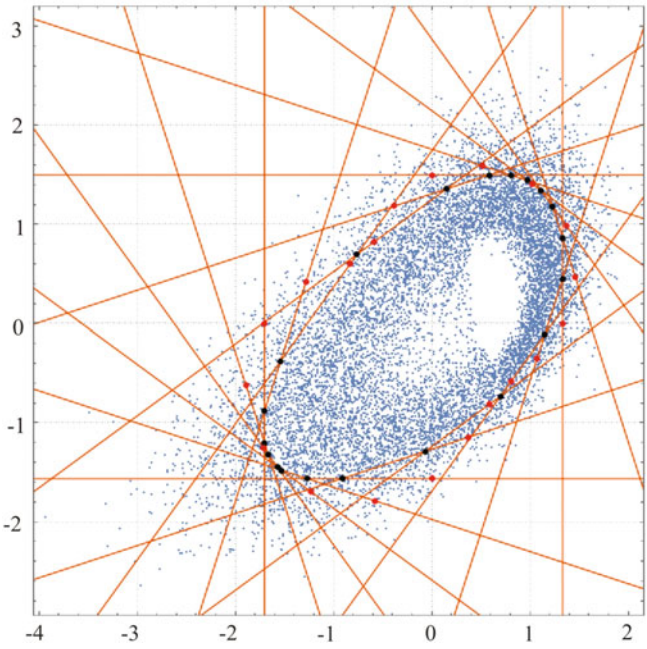


Fig. 12.55 The principle of the directional quantile envelope technique

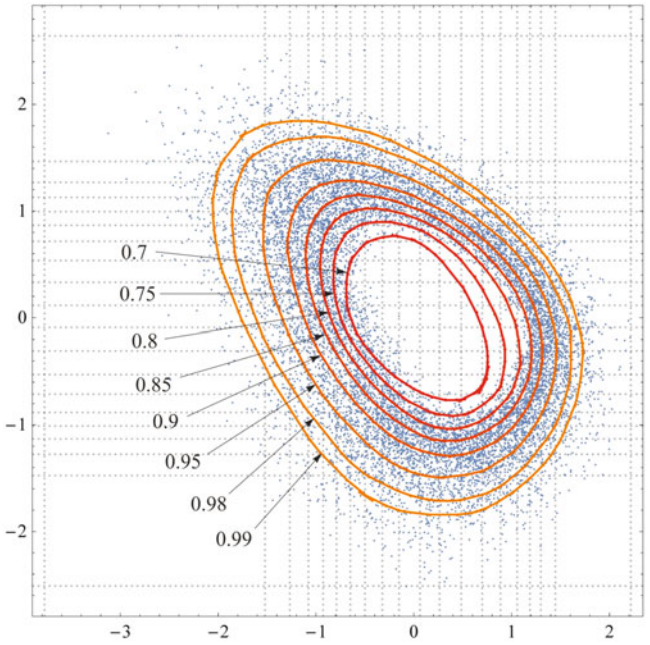


Fig. 12.56 The results of the directional quantile envelope technique

```
Dimensions[qsPoints]
{8,60,2}
Block[{data = sndData},
 Show[{ListPlot[data,AspectRatio → 1,PlotTheme → {"Detailed"},
 GridLines → Map[Quantile[#,Range[0,1,1/(20 - 1)]]&,
 Transpose[data]],PlotLegends → SwatchLegend[
 Blend[{Red,Orange},Rescale[#1,{Min[qs],Max[qs]},
 {0,1}]]&/@qs,qs]],Graphics[{PointSize[0.005],
 Thickness[0.0035],MapThread[{Blend[{Red,Orange},Rescale[#1,
 {Min[qs],Max[qs]},{0,1}]],Tooltip[Line[Append[#2,#2[[1]]]],#1],
 Point[#2]}&,{qs,qsPoints}]}]},ImageSize → 300]]
```

References

Antonov A (2013) Quantile regression mathematica package, source code at GitHub, https://github.com/antononcube/MathematicaForPrediction, package QuantileRegression.m

Davino C, Furno M, Vistocco D (2014) Quantile regression—theory and applications. Wiley Series in Probability and Statistics

Hald A (2007) A history of parametric statistical inference from Bernoulli to fisher 1713–1935. Springer, Berlin

Koenker R (2005) Quantile regression. econometric society monograph. Cambridge University Press, Cambridge. http://www2.sas.com/proceedings/sugi30/213-30.pdf

Logan J, Petscher Y (2013) An introduction to quantile regression. In: Presentation for modern modeling methods conference: 23 May 2013

McMillen DP (2012) Quantile regression for spatial data (SpringerBriefs in Regional Science) Springer

Chapter 13
Robust Regression

13.1 Basic Methods in Robust Regression

13.1.1 Concept of Robust Regression

In many fields such as robotics (Poppinga et al. 2008), computer vision (Mitra and Nguyen 2003), digital photogrammetry (Yang and Förtsner 2010), surface reconstruction (Nurunnabi et al. 2012), computational geometry (Lukács et al. 1998) as well as in the increasing applications of laser scanning (Stathas et al. 2003), it is a fundamental task for extracting features from 3D point cloud. Since the physical limitations of the sensors, the occlusions, multiple reflectance and noise can produce off-surface points, robust fitting techniques are required. *Robust fitting* means an estimation technique which is able to estimate accurate model parameters not only despite small-scale noise in the data set but occasionally large scale measurement errors (outliers). *Outliers definition* is not easy. Perhaps considering the problem from the practical point of the view, we can say that data points, which appearance in the data set causes dramatically change in the result of the parameter estimation can be labeled as outliers.

Basically there are two different methods to handle outliers:

(a) weighting out outliers
(b) discarding outliers

Weighting outliers means that we do not kick out certain data points labeled as outlier but during the parameter estimation process we take them with a low weight into consideration in the objective function. Such a technique is the good old *Danish method*.

The other technique will try to identify data points, which make "troubles" during the parameter estimation process. Troubles mean that their existence in the data set

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_13

change the result of the parameter estimation considerably. One of the representative of this technique is the RANdom SAmple Consensus (RANSAC) method.

Both these techniques eliminate outliers in a way. However there are softer methods used in case of presence of outliers, too.

The simplest methods of estimating parameters in a regression model that are less sensitive to outliers than the least squares estimates, is to use least absolute deviations. Even then, gross outliers can still have a considerable impact on the model, motivating research into even more robust approaches.

In 1973, Huber introduced M-estimation for regression. The M in M-estimation stands for "maximum likelihood type". The method is robust to outliers in the response variables, but turned out not to be resistant to outliers in the explanatory variables (leverage points). In fact, when there are outliers in the explanatory variables, the method has no advantage over least squares. This handicap maybe eliminated employing total least square technique.

First let us see the application of the maximum likelihood estimation technique to fitting cloud data to a plane.

13.1.2 Maximum Likelihood Method

13.1.2.1 Symbolic Solution

Let a point be P_i with coordinates $\{x_i, y_i, z_i\}$ and d_i is its distance from a general point $\{x, y, z\}$ of the plane $z = \alpha x + \beta y + \gamma$, then

```
di = (zi - αx - βy - γ)^2 + (xi - x)^2 + (yi - y)^2
```

The distance of P_i from its perpendicular projection to the plane is the shortest distance, which represents its regression error $\varepsilon_{\underline{MI}}$. From the necessary conditions we get

```
deq1 = D[di, x]//Expand
2x + 2xα^2 + 2yαβ + 2αγ - 2xi - 2αzi

deq2 = D[di, y]//Expand
2y + 2xαβ + 2yβ^2 + 2βγ - 2yi - 2βzi

xy = Solve[{deq1 == 0, deq2 == 0}, {x, y}]//Flatten
{x → -(αγ - xi - β^2 xi + αβyi - αzi)/(1 + α^2 + β^2), y → -(βγ + αβxi - yi - α^2 yi - βzi)/(1 + α^2 + β^2)}
```

Then the square of the error is

```
Δi = di /. xy // Simplify
```

$$\frac{(\gamma + \alpha x_i + \beta y_i - z_i)^2}{1 + \alpha^2 + \beta^2}$$

Consequently our error model is

$$\varepsilon = \frac{z_i - x_i\alpha - y_i\beta - \gamma}{\sqrt{1 + \alpha^2 + \beta^2}};$$

Now let us develop the likelihood with this error model and assuming that the model error has a Gaussian distribution with zero mean.

Then the Maximum Likelihood (ML) estimator for Gaussian-type noise is

```
L = ∏_{i=1}^{N} PDF[NormalDistribution[0, σ], eMi]
```

$$\prod_{i=1}^{\mathbb{N}} \frac{e^{-\frac{(e_M)_i^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}$$

Then the likelihood function—considering its logarithm—should be maximized

```
LogL = log[L /. eMi → wi ε] // Simplify
```

$$-\frac{1}{2}\mathbb{N}\log[2] - \frac{1}{2}\mathbb{N}\log[\pi] - \mathbb{N}\log[\sigma] + \sum_{i=1}^{\mathbb{N}} -\frac{\gamma^2 w_i^2}{2(1 + \alpha^2 + \beta^2)\sigma^2} +$$
$$\sum_{i=1}^{\mathbb{N}} -\frac{\alpha\gamma w_i^2 x_i}{(1 + \alpha^2 + \beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{\alpha^2 w_i^2 x_i^2}{2(1 + \alpha^2 + \beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{\beta\gamma w_i^2 y_i}{(1 + \alpha^2 + \beta^2)\sigma^2} +$$
$$\sum_{i=1}^{\mathbb{N}} -\frac{\alpha\beta w_i^2 x_i y_i}{(1 + \alpha^2 + \beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{\beta^2 w_i^2 y_i^2}{2(1 + \alpha^2 + \beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{\gamma w_i^2 z_i}{(1 + \alpha^2 + \beta^2)\sigma^2} +$$
$$\sum_{i=1}^{\mathbb{N}} \frac{\alpha w_i^2 x_i z_i}{(1 + \alpha^2 + \beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{\beta w_i^2 y_i z_i}{(1 + \alpha^2 + \beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{w_i^2 z_i^2}{2(1 + \alpha^2 + \beta^2)\sigma^2}$$

Here the weighted error with weights w_i is used. In order to avoid global maximization we transform the problem of the global solution of a multivariate polynomial system. From the necessary conditions one can obtain

```
eq1 = D[LogL, α]
```

$$\sum_{i=1}^{\mathbb{N}} \frac{\alpha\gamma^2 w_i^2}{(1 + \alpha^2 + \beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \left(\frac{2\alpha^2\gamma w_i^2 x_i}{(1 + \alpha^2 + \beta^2)^2\sigma^2} - \frac{\gamma w_i^2 x_i}{(1 + \alpha^2 + \beta^2)\sigma^2}\right) +$$
$$\sum_{i=1}^{\mathbb{N}} \left(\frac{\alpha^3 w_i^2 x_i^2}{(1 + \alpha^2 + \beta^2)^2\sigma^2} - \frac{\alpha w_i^2 x_i^2}{(1 + \alpha^2 + \beta^2)\sigma^2}\right) + \sum_{i=1}^{\mathbb{N}} \frac{2\alpha\beta\gamma w_i^2 y_i}{(1 + \alpha^2 + \beta^2)^2\sigma^2} +$$
$$\sum_{i=1}^{\mathbb{N}} \frac{\alpha\beta^2 w_i^2 y_i^2}{(1 + \alpha^2 + \beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \left(\frac{2\alpha^2\beta w_i^2 x_i y_i}{(1 + \alpha^2 + \beta^2)^2\sigma^2} - \frac{\beta w_i^2 x_i y_i}{(1 + \alpha^2 + \beta^2)\sigma^2}\right) +$$

$$\sum_{i=1}^{\mathbb{N}} -\frac{2\alpha\gamma w_i^2 z_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{2\alpha\beta w_i^2 y_i z_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{\alpha w_i^2 z_i^2}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \left(-\frac{2\alpha^2 w_i^2 x_i z_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \frac{w_i^2 x_i z_i}{(1+\alpha^2+\beta^2)\sigma^2}\right)$$

```
eq2 = D[LogL, β]
```

$$\sum_{i=1}^{\mathbb{N}} \frac{\beta\gamma^2 w_i^2}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{2\alpha\beta\gamma w_i^2 x_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{\alpha^2\beta w_i^2 x_i^2}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \left(\frac{2\beta^2\gamma w_i^2 y_i}{(1+\alpha^2+\beta^2)^2\sigma^2} - \frac{\gamma w_i^2 y_i}{(1+\alpha^2+\beta^2)\sigma^2}\right) + \sum_{i=1}^{\mathbb{N}} \left(\frac{2\alpha\beta^2 w_i^2 x_i y_i}{(1+\alpha^2+\beta^2)^2\sigma^2} - \frac{\alpha w_i^2 x_i y_i}{(1+\alpha^2+\beta^2)\sigma^2}\right) + \sum_{i=1}^{\mathbb{N}} \left(\frac{\beta^3 w_i^2 y_i^2}{(1+\alpha^2+\beta^2)^2\sigma^2} - \frac{\beta w_i^2 y_i^2}{(1+\alpha^2+\beta^2)\sigma^2}\right) + \sum_{i=1}^{\mathbb{N}} -\frac{2\beta\gamma w_i^2 z_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{2\alpha\beta w_i^2 x_i z_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{\beta w_i^2 z_i^2}{(1+\alpha^2+\beta^2)^2\sigma^2} + \sum_{i=1}^{\mathbb{N}} \left(-\frac{2\beta^2 w_i^2 y_i z_i}{(1+\alpha^2+\beta^2)^2\sigma^2} + \frac{w_i^2 y_i z_i}{(1+\alpha^2+\beta^2)\sigma^2}\right)$$

```
eq3 = D[LogL, γ]
```

$$\sum_{i=1}^{\mathbb{N}} -\frac{\gamma w_i^2}{(1+\alpha^2+\beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{\alpha w_i^2 x_i}{(1+\alpha^2+\beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} -\frac{\beta w_i^2 y_i}{(1+\alpha^2+\beta^2)\sigma^2} + \sum_{i=1}^{\mathbb{N}} \frac{w_i^2 z_i}{(1+\alpha^2+\beta^2)\sigma^2}$$

We get a multivariate algebraic system for the unknown parameters α, β and γ. In compact form

```
eq1 = jαγ^2 + 2α^2γa - γa(1 + α^2 + β^2) + α^3b - αb(1 + α^2 + β^2) +
 2αβγc + αβ^2d + 2α^2βe - βe(1 + α^2 + β^2) - 2αγf - 2αβg + αh -
 2α^2i + i(1 + α^2 + β^2)//Expand
i - bα + hα - iα^2 - eβ - 2gαβ + eα^2β + iβ^2 - bαβ^2 +
 dαβ^2 - eβ^3 - aγ - 2fαγ + aα^2γ + 2cαβγ - aβ^2γ + jαγ^2

eq2 = jβγ^2 + 2αβγa + α^2βb + 2β^2γc - γc(1 + α^2 + β^2) + 2αβ^2e -
 αe(1 + α^2 + β^2) + β^3d - βd(1 + α^2 + β^2) - 2βγf - 2αβi + βh -
 2β^2g + g(1 + α^2 + β^2)//Expand
g - eα + gα^2 - eα^3 - dβ + hβ - 2iαβ + bα^2β - dα^2β -
 gβ^2 + eαβ^2 - cγ - cα^2γ - 2fβγ + 2aαβγ + cβ^2γ + jβγ^2

eq3 = f - jγ - αa - βc
f - aα - cβ - jγ
```

where the constants are

$$a=\sum_{i=1}^{\mathbb{N}} w_i^2 x_i,\ b=\sum_{i=1}^{\mathbb{N}} w_i^2 x_i^2,\ c=\sum_{i=1}^{\mathbb{N}} w_i^2 y_i,\ d=\sum_{i=1}^{\mathbb{N}} w_i^2 y_i^2,\ e=\sum_{i=1}^{\mathbb{N}} w_i^2 x_i y_i,$$
$$f=\sum_{i=1}^{\mathbb{N}} w_i^2 z_i,\ g=\sum_{i=1}^{\mathbb{N}} w_i^2 y_i z_i,\ h=\sum_{i=1}^{\mathbb{N}} w_i^2 z_i^2,\ i=\sum_{i=1}^{\mathbb{N}} w_i^2 x_i z_i,\ j=\sum_{i=1}^{\mathbb{N}} w_i^2$$

Similarly the compact form of objective function, which will be required later

$$\begin{aligned}L=&-\frac{j\gamma^2}{2(1+\alpha^2+\beta^2)\sigma^2}-\frac{1}{2}\mathrm{Nlog}[2]-\frac{1}{2}\mathrm{Nlog}[\pi]-\mathrm{Nlog}[\sigma]-\\&a\frac{\alpha\gamma}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{\alpha^2 b}{2(1+\alpha^2+\beta^2)\sigma^2}-\frac{\beta\gamma c}{(1+\alpha^2+\beta^2)\sigma^2}-\\&\frac{\alpha\beta e}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{\beta^2 d}{2(1+\alpha^2+\beta^2)\sigma^2}+\frac{\gamma f}{(1+\alpha^2+\beta^2)\sigma^2}+\\&\frac{\alpha i}{(1+\alpha^2+\beta^2)\sigma^2}+\frac{\beta g}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{h}{2(1+\alpha^2+\beta^2)\sigma^2}\\&-\frac{h}{2(1+\alpha^2+\beta^2)\sigma^2}+\frac{i\alpha}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{b\alpha^2}{2(1+\alpha^2+\beta^2)\sigma^2}+\\&\frac{g\beta}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{e\alpha\beta}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{d\beta^2}{2(1+\alpha^2+\beta^2)\sigma^2}+\\&\frac{f\gamma}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{a\alpha\gamma}{(1+\alpha^2+\beta^2)\sigma^2}-\frac{c\beta\gamma}{(1+\alpha^2+\beta^2)\sigma^2}-\\&\frac{j\gamma^2}{2(1+\alpha^2+\beta^2)\sigma^2}-\frac{1}{2}\mathbb{N}\,\mathrm{log}[2]-\frac{1}{2}\mathbb{N}\,\mathrm{log}[\pi]-\mathbb{N}\,\mathrm{log}[\sigma]\end{aligned}$$

This technique is similar to finding regression parameters of a straight line, see Paláncz (2014).

This polynomial system can be solved in symbolic way via Gröbner basis and Sylvester resultant. The polynomial system can be reduced to monomials of higher order. First let us eliminate γ via Gröbner basis, we get

```
gbαβ = GroebnerBasis[{eq1, eq2, eq3}, {α, β, γ}, {γ},
 MonomialOrder → EliminationOrder]
{-afα + ijα + a^2α^2 - f^2α^2 - bjα^2 + hjα^2 +
  afα^3 - ijα^3 - cfβ + gjβ + 2acαβ - 2ejαβ +
 cfα^2β - gjα^2β + c^2β^2 - f^2β^2 -
  djβ^2 + hjβ^2 + afαβ^2 - ijαβ^2 + cfβ^3 - gjβ^3, -
 af + ij + a^2α - f^2α - bjα + hjα + afα^2 - ijα^2 +
  acβ - ejβ + 2cfαβ - 2gjαβ -
 acα^2β + ejα^2β - afβ^2 + ijβ^2 + a^2αβ^2 -
  c^2αβ^2 - bjαβ^2 + djαβ^2 + acβ^3 - ejβ^3,
  cf - gj - acα + ejα + cfα^2 - gjα^2 - acα^3 + ejα^3 -
  c^2β + f^2β + djβ - hjβ - 2afαβ +
  2ijαβ + a^2α^2β - c^2α^2β -
  bjα^2β + djα^2β - cfβ^2 + gjβ^2 + acαβ^2 - ejαβ^2, -
  ag + ci - bcα + aeα - fgα + chα + efα^2 - agα^2 - bcα^3 + aeα^3 -
```

$$
\begin{aligned}
&fg\alpha^3 + ch\alpha^3 + ef\alpha^4 - ci\alpha^4 + \\
&ad\beta - ce\beta - ah\beta + fi\beta - bf\alpha\beta + df\alpha\beta - cg\alpha\beta + ai\alpha\beta + \\
&ad\alpha^2\beta - ce\alpha^2\beta - ah\alpha^2\beta + fi\alpha^2\beta - \\
&bf\alpha^3\beta + df\alpha^3\beta - cg\alpha^3\beta + ai\alpha^3\beta - ef\beta^2 + ci\beta^2 - \\
&bc\alpha\beta^2 + ae\alpha\beta^2 - fg\alpha\beta^2 + ch\alpha\beta^2 + ag\alpha^2\beta^2 - \\
&ci\alpha^2\beta^2 + ad\beta^3 - ce\beta^3 - ah\beta^3 + fi\beta^3 - \\
&bf\alpha\beta^3 + df\alpha\beta^3 - cg\alpha\beta^3 + ai\alpha\beta^3 - ef\beta^4 + ag\beta^4, - \\
&afg + 2cfi - gij - bcf\alpha + 2aef\alpha - f^2g\alpha + cfh\alpha - aci\alpha - \\
&a^2e\alpha^2 + 2ef^2\alpha^2 - afg\alpha^2 + cfi\alpha^2 + \\
&bej\alpha^2 - ehj\alpha^2 - gij\alpha^2 - bcf\alpha^3 - f^2g\alpha^3 + \\
&cfh\alpha^3 - aci\alpha^3 + 2eij\alpha^3 + ef^2\alpha^4 - cfi\alpha^4 + \\
&adf\beta - afh\beta - c^2i\beta + 2f^2i\beta - \\
&egj\beta + dij\beta - hij\beta - 2ace\alpha\beta - bf^2\alpha\beta + df^2\alpha\beta - cfg\alpha\beta + \\
&2e^2j\alpha\beta + i^2j\alpha\beta + adf\alpha^2\beta - 2cef\alpha^2\beta - afh\alpha^2\beta - c^2i\alpha^2\beta + \\
&2f^2i\alpha^2\beta + egj\alpha^2\beta + dij\alpha^2\beta - hij\alpha^2\beta - \\
&bf^2\alpha^3\beta + df^2\alpha^3\beta - cfg\alpha^3\beta + i^2j\alpha^3\beta - c^2e\beta^2 + \\
&cfi\beta^2 + dej\beta^2 - ehj\beta^2 - bcf\alpha\beta^2 - f^2g\alpha\beta^2 + cfh\alpha\beta^2 - \\
&aci\alpha\beta^2 + 2eij\alpha\beta^2 + afg\alpha^2\beta^2 - 2cfi\alpha^2\beta^2 + gij\alpha^2\beta^2 + adf\beta^3 - \\
&2cef\beta^3 - afh\beta^3 - c^2i\beta^3 + 2f^2i\beta^3 + \\
&egj\beta^3 + dij\beta^3 - hij\beta^3 - bf^2\alpha\beta^3 + df^2\alpha\beta^3 - cfg\alpha\beta^3 + \\
&i^2j\alpha\beta^3 - ef^2\beta^4 + afg\beta^4 - cfi\beta^4 + gij\beta^4, - \\
&bcf + aef - a^2g + cfh + aci + bgj - ghj - eij + ef^2\alpha - 2afg\alpha - \\
&cfi\alpha + 2gij\alpha - bcf\alpha^2 + \\
&aef\alpha^2 - f^2g\alpha^2 + cfh\alpha^2 + aci\alpha^2 - eij\alpha^2 + ef^2\alpha^3 - cfi\alpha^3 + \\
&bc^2\beta + a^2d\beta - 2ace\beta - bf^2\beta - 2cfg\beta - \\
&a^2h\beta - c^2h\beta + f^2h\beta + 2afi\beta - bdj\beta + e^2j\beta + \\
&2g^2j\beta + bhj\beta + dhj\beta - h^2j\beta - i^2j\beta + 2adf\alpha\beta - \\
&cef\alpha\beta + 2acg\alpha\beta - 2afh\alpha\beta + c^2i\alpha\beta - 2egj\alpha\beta - \\
&2dij\alpha\beta + 2hij\alpha\beta + bc^2\alpha^2\beta - 2ace\alpha^2\beta - \\
&bf^2\alpha^2\beta + df^2\alpha^2\beta - c^2h\alpha^2\beta + 2afi\alpha^2\beta + e^2j\alpha^2\beta - i^2j\alpha^2\beta - \\
&cef\alpha^3\beta + c^2i\alpha^3\beta + cdf\beta^2 + 2c^2g\beta^2 - \\
&2f^2g\beta^2 - cfh\beta^2 - 3dgj\beta^2 + 3ghj\beta^2 - 2acd\alpha\beta^2 + \\
&ef^2\alpha\beta^2 + 2afg\alpha\beta^2 + 2ach\alpha\beta^2 - cfi\alpha\beta^2 + \\
&2dej\alpha\beta^2 - 2ehj\alpha\beta^2 - 2gij\alpha\beta^2 + bcf\alpha^2\beta^2 - \\
&cdf\alpha^2\beta^2 - aef\alpha^2\beta^2 + c^2g\alpha^2\beta^2 - aci\alpha^2\beta^2 + \\
&eij\alpha^2\beta^2 + bc^2\beta^3 + a^2d\beta^3 - c^2d\beta^3 - 2ace\beta^3 - \\
&bf^2\beta^3 + df^2\beta^3 + 2cfg\beta^3 - a^2h\beta^3 + 2afi\beta^3 - \\
&bdj\beta^3 + d^2j\beta^3 + e^2j\beta^3 - 2g^2j\beta^3 + bhj\beta^3 - dhj\beta^3 - \\
&i^2j\beta^3 - cef\alpha\beta^3 - 2acg\alpha\beta^3 + c^2i\alpha\beta^3 \\
&+ \ 2egj\alpha\beta^3 + bcf\beta^4 - cdf\beta^4 - aef\beta^4 + \\
&a^2g\beta^4 - aci\beta^4 - bgj\beta^4 + dgj\beta^4 + eij\beta^4, - cdf + cfh + \\
&dgj - ghj + acd\alpha + 2afg\alpha - ach\alpha - 2cfi\alpha - dej\alpha + ehj\alpha + \\
&bcf\alpha^2 - cdf\alpha^2 - 2aef\alpha^2 - a^2g\alpha^2 + \\
&2f^2g\alpha^2 + aci\alpha^2 + bgj\alpha^2 + dgj\alpha^2 - 2ghj\alpha^2 + acd\alpha^3 +
\end{aligned}
$$

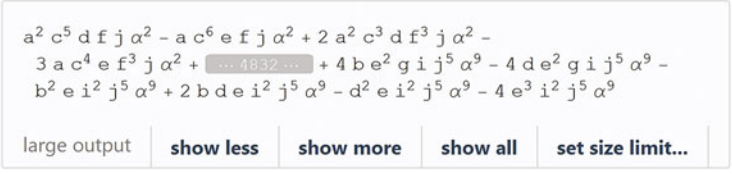
```
a^2 e α^3 - 2 e f^2 α^3 - a c h α^3 - c f i α^3 - b e j α^3 -
d e j α^3 + 2 e h j α^3 + 2 g i j α^3 + b c f α^4 + f^2 g α^4 -
c f h α^4 + a c i α^4 - 2 e i j α^4 - e f^2 α^5 + c f i α^5 + c^2 d β -
d f^2 β - c^2 h β + f^2 h β - d^2 j β + 2 d h j β - h^2 j β - a c g α β + c^2
i α β - 2 f^2 i α β + 2 e g j α β - 2 d i j α β +
2 h i j α β + c^2 d α^2 β + 2 a c e α^2 β + b f^2 α^2 β - 2 d f^2 α^2 β -
c f g α^2 β - c^2 h α^2 β + f^2 h α^2 β - d^2 j α^2 β -
2 e^2 j α^2 β + 2 g^2 j α^2 β + 2 d h j α^2 β - h^2 j α^2 β - i^2 j α^2 β + 2 c e f α^3 β +
a c g α^3 β + c^2 i α^3 β - 2 f^2 i α^3 β -
2 e g j α^3 β - 2 d i j α^3 β + 2 h i j α^3 β + b f^2 α^4 β - d f^2 α^4 β
+ c f g α^4 β - i^2 j α^4 β + a c d α β^2 + c^2 e α β^2 -
a c h α β^2 - c f i α β^2 - 2 d e j α β^2 + 2 e h j α β^2 + b c f α^2 β^2 +
c d f α^2 β^2 + c^2 g α^2 β^2 - 2 c f h α^2 β^2 +
a c i α^2 β^2 - 2 d g j α^2 β^2 + 2 g h j α^2 β^2 - 2 e i j α^2 β^2 + 2 c f i α^3 β^2 -
2 g i j α^3 β^2 + c^2 d β^3 - d f^2 β^3 - c f g β^3 -
c^2 h β^3 + f^2 h β^3 - d^2 j β^3 + g^2 j β^3 + 2 d h j β^3 -
h^2 j β^3 + 2 c e f α β^3 + a c g α β^3 + c^2 i α β^3 - 2 f^2 i α β^3 -
2 e g j α β^3 - 2 d i j α β^3 + 2 h i j α β^3 + b f^2 α^2 β^3 - d f^2 α^2 β^3 +
2 c f g α^2 β^3 - g^2 j α^2 β^3 - i^2 j α^2 β^3 + c d f β^4 +
c^2 g β^4 - f^2 g β^4 - c f h β^4 - 2 d g j β^4 + 2 g h j β^4 + e f^2 α β^4 +
c f i α β^4 - 2 g i j α β^4 + c f g β^5 - g^2 j β^5 }
```

The basis consists of 7 polynomials of variables α and β. The lengths of these polynomials are

```
Map[Length[#]&, gbαβ]
{22, 22, 22, 48, 75, 102, 120}
```

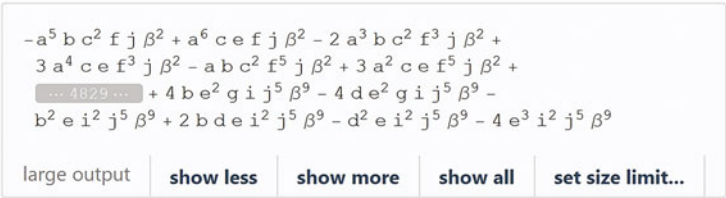
Let us consider the first and the third one and eliminate β via Sylvester resultant,

```
alfa = Resultant[gbαβ[[1]], gbαβ[[3]], β]
```



so we get a univariate polynomial for α. Similarly eliminate α from the first and second basis,

```
beta = Resultant[gbαβ[[1]], gbαβ[[2]], α]
```



which leads to a univariate polynomial of β.

The coefficients of the univariate polynomial for α are

```
ca0 = Coefficient[alfa, α, 0]
0

ca1 = Coefficient[alfa, α, 1]
0

ca2 = Coefficient[alfa, α, 2]
a^2 c^5 dfj - ac^6 efj + 2a^2 c^3 df^3 j - 3ac^4 ef^3 j + a^2 cdf^5 j -
 3ac^2 ef^5 j - aef^7 j - a^2 c^6 gj - a^2 c^4 f^2 gj +
 a^2 c^2 f^4 gj + a^2 f^6 gj - a^2 c^5 fhj - 2a^2 c^3 f^3 hj - a^2 cf^5 hj +
 ac^7 ij + 3ac^5 f^2 ij + 3ac^3 f^4 ij + acf^6 ij -
 2a^2 c^3 d^2 fj^2 + ac^4 defj^2 + c^5 e^2 fj^2 + 2a^2 cd^2 f^3 j^2 -
 2ac^2 def^3 j^2 + 2c^3 e^2 f^3 j^2 - 3adef^5 j^2 + ce^2 f^5 j^2 +
 2a^2 c^4 dgj^2 + 2ac^5 egj^2 - 12a^2 c^2 df^2 gj^2 + 12ac^3 ef^2 gj^2 +
 2a^2 df^4 gj^2 + 10acef^4 gj^2 + 8a^2 c^3 fg^2 j^2 -
 8a^2 cf^3 g^2 j^2 + 4a^2 c^3 dfhj^2 - ac^4 efhj^2 - 4a^2 cdf^3 hj^2 +
 2ac^2 ef^3 hj^2 + 3aef^5 hj^2 - 2a^2 c^4 ghj^2 +
 12a^2 c^2 f^2 ghj^2 - 2a^2 f^4 ghj^2 - 2a^2 c^3 fh^2 j^2 + 2a^2 cf^3 h^2 j^2 -
 3ac^5 dij^2 - c^6 eij^2 - 2ac^3 df^2 ij^2 -
 c^4 ef^2 ij^2 + acdf^4 ij^2 + c^2 ef^4 ij^2 + ef^6 ij^2 - 10ac^4 fgij^2 -
 12ac^2 f^3 gij^2 - 2af^5 gij^2 + 3ac^5 hij^2 +
 2ac^3 f^2 hij^2 - acf^4 hij^2 - c^5 fi^2 j^2 - 2c^3 f^3 i^2 j^2 - cf^5 i^2 j^2
  + a^2 cd^3 fj^3 + ac^2 d^2 efj^3 - 2c^3 de^2 fj^3 -
 3ad^2 ef^3 j^3 + 2cde^2 f^3 j^3 - a^2 c^2 d^2 gj^3 - 4ac^3 degj^3 - c^4 e^2 gj^3 +
 a^2 d^2 f^2 gj^3 + 12acdef^2 gj^3 -
 10c^2 e^2 f^2 gj^3 - e^2 f^4 gj^3 + 4a^2 cdfg^2 j^3 - 20ac^2 efg^2 j^3 -
 4aef^3 g^2 j^3 - 4a^2 c^2 g^3 j^3 + 4a^2 f^2 g^3 j^3 -
 3a^2 cd^2 fhj^3 - 2ac^2 defhj^3 + 2c^3 e^2 fhj^3 + 6adef^3 hj^3 -
 2ce^2 f^3 hj^3 + 2a^2 c^2 dghj^3 + 4ac^3 eghj^3 -
 2a^2 df^2 ghj^3 - 12acef^2 ghj^3 - 4a^2 cfg^2 hj^3 + 3a^2 cdfh^2 j^3 +
 ac^2 efh^2 j^3 - 3aef^3 h^2 j^3 - a^2 c^2 gh^2 j^3 +
 a^2 f^2 gh^2 j^3 - a^2 cfh^3 j^3 + 3ac^3 d^2 ij^3 + 3c^4 deij^3 -
 acd^2 f^2 ij^3 - 2c^2 def^2 ij^3 + 3def^4 ij^3 +
```

```
12ac^2dfgij^3 + 8c^3efgij^3 - 4adf^3gij^3 - 8cef^3gij^3 +
4ac^3g^2ij^3 + 20acf^2g^2ij^3 - 6ac^3dhij^3 -
3c^4ehij^3 + 2acdf^2hij^3 + 2c^2ef^2hij^3 - 3ef^4hij^3 -
12ac^2fghij^3 + 4af^3ghij^3 + 3ac^3h^2ij^3 -
acf^2h^2ij^3 + 2c^3dfi^2j^3 - 2cdf^3i^2j^3 + c^4gi^2j^3 +
10c^2f^2gi^2j^3 + f^4gi^2j^3 - 2c^3fhi^2j^3 + 2cf^3hi^2j^3 -
ad^3efj^4 + cd^2e^2fj^4 + 2acd^2egj^4 + 2c^2de^2gj^4 -
2de^2f^2gj^4 - 4adefg^2j^4 + 12ce^2fg^2j^4 +
8aceg^3j^4 + 3ad^2efhj^4 - 2cde^2fhj^4 -
4acdeghj^4 - 2c^2e^2ghj^4 + 2e^2f^2ghj^4 + 4aefg^2hj^4 -
3adefh^2j^4 + ce^2fh^2j^4 + 2acegh^2j^4 + aefh^3j^4 -
acd^3ij^4 - 3c^2d^2eij^4 + 3d^2ef^2ij^4 - 2ad^2fgij^4 -
8cdefgij^4 - 4acdg^2ij^4 - 4c^2eg^2ij^4 + 4ef^2g^2ij^4 -
8afg^3ij^4 + acd^2hij^4 + 6c^2dehij^4 -
6def^2hij^4 + 4adfghij^4 + 8cefghij^4 + 4acg^2hij^4 -
3acdh^2ij^4 - 3c^2eh^2ij^4 + 3ef^2h^2ij^4 -
2afgh^2ij^4 + ach^3ij^4 - cd^2fi^2j^4 - 2c^2dgi^2j^4 +
2df^2gi^2j^4 - 12cfg^2i^2j^4 + 2cdfhi^2j^4 +
2c^2ghi^2j^4 - 2f^2ghi^2j^4 - cfh^2i^2j^4 - d^2e^2gj^5 - 4e^2g^3j^5 +
2de^2ghj^5 - e^2gh^2j^5 + d^3eij^5 +
4deg^2ij^5 - 3d^2ehij^5 - 4eg^2hij^5 + 3deh^2ij^5 - eh^3ij^5 +
d^2gi^2j^5 + 4g^3i^2j^5 - 2dghi^2j^5 + gh^2i^2j^5

ca3 = Coefficient[alfa, α, 3]; Short[ca3, 10]
-abc^7j - a^3c^5dj + 2a^2c^6ej - 3abc^5f^2j - 6a^3c^3df^2j +
2ac^5df^2j + 10a^2c^4ef^2j - c^6ef^2j -
3abc^3f^4j - 5a^3cdf^4j + 4ac^3df^4j +
14a^2c^2ef^4j - 3c^4ef^4j - abcf^6j + 2acdf^6j +
6a^2ef^6j - 3c^2ef^6j - ef^8j ≪ 550 ≫ + 3deh^3j^5 - eh^4j^5 -
2bd^2gij^5 + d^3gij^5 - 12de^2gij^5 -
8bg^3ij^5 + 4dg^3ij^5 + 4bdghij^5 - d^2ghij^5 +
12e^2ghij^5 + 4g^3hij^5 - 2bgh^2ij^5 -
dgh^2ij^5 + gh^3ij^5 + 5d^2ei^2j^5 - 4eg^2i^2j^5 -
10dehi^2j^5 + 5eh^2i^2j^5 + 4dgi^3j^5 - 4ghi^3j^5

ca4 = Coefficient[alfa, α, 4]; Short[ca4, 10]
5a^2bc^5fj - bc^7fj + 6a^4c^3dfj - 11a^3c^4efj + ac^6efj +
10a^2bc^3f^3j - 3bc^5f^3j + 10a^4cdf^3j -
10a^2c^3df^3j + c^5df^3j - 26a^3c^2ef^3j +
8ac^4ef^3j + 5a^2bcf^5j - 3bc^3f^5j - 10a^2cdf^5j + 2c^3df^5j -
15a^3ef^5j + 13ac^2ef^5j - bcf^7j +
cdf^7j + ≪ 665 ≫ + d^3eij^5 + 8de^3ij^5 + 12beg^2ij^5 -
16deg^2ij^5 + 14bdehij^5 + 4d^2ehij^5 - 8e^3hij^5 +
4eg^2hij^5 - 7beh^2ij^5 - 11deh^2ij^5 + 6eh^3ij^5 -
10bdgi^2j^5 + 5d^2gi^2j^5 - 8e^2gi^2j^5 + 10bghi^2j^5 -
5gh^2i^2j^5 + 8dei^3j^5 - 8ehi^3j^5 + 4gi^4j^5
```

```
ca5 = Coefficient[alfa, α, 5]; Short[ca5, 10]
 - 2a^3bc^5j - 2abc^7j - 2a^5c^3dj - 2a^3c^5dj +
  4a^4c^4ej + 4a^2c^6ej - 12a^3bc^3f^2j +
  abc^5f^2j - 10a^5cdf^2j + 6a^3c^3df^2j + 3ac^5df^2j +
  24a^4c^2ef^2j - 3a^2c^4ef^2j - c^6ef^2j -
  10a^3bcf^4j + 8abc^3f^4j + 20a^3cdf^4j -
  2ac^3df^4j + ≪769≫ + 2bdghij^5 + 6d^2ghij^5 -
  12e^2ghij^5 + 8g^3hij^5 + 7bgh^2ij^5 - 7dgh^2ij^5 - 14bdei^2j^5 +
  d^2ei^2j^5 + 8e^3i^2j^5 -
  8eg^2i^2j^5 + 14behi^2j^5 + 12dehi^2j^5 - 13eh^2i^2j^5 -
  12bgi^3j^5 + 4dgi^3j^5 + 8ghi^3j^5 + 4ei^4j^5

ca6 = Coefficient[alfa, α, 6]; Short[ca6, 10]
6a^4bc^3fj + 4a^2bc^5fj - 2bc^7fj +
  5a^6cdfj + 2a^4c^3dfj - 3a^2c^5dfj -
  11a^5c^2efj - 6a^3c^4efj + 5ac^6efj + 10a^4bcf^3j -
  6a^2bc^3f^3j - 4bc^5f^3j - 20a^4cdf^3j -
  6a^2c^3df^3j + 2c^5df^3j -
  15a^5ef^3j + 18a^3c^2ef^3j ≪770≫ + 9d^2ehij^5 + 4e^3hij^5 -
  4eg^2hij^5 + 8beh^2ij^5 - 8deh^2ij^5 + 13b^2gi^2j^5 -
  12bdgi^2j^5 - d^2gi^2j^5 + 8e^2gi^2j^5 -
  8g^3i^2j^5 - 14bghi^2j^5 + 14dghi^2j^5 -
  8bei^3j^5 - 4dei^3j^5 + 12ehi^3j^5 - 4gi^4j^5

ca7 = Coefficient[alfa, α, 7]; Short[ca7, 10]
 - a^5bc^3j - 2a^3bc^5j - abc^7j - a^7cdj - 2a^5c^3dj -
  a^3c^5dj + 2a^6c^2ej + 4a^4c^4ej +
  2a^2c^6ej - 5a^5bcf^2j + 5abc^5f^2j + 10a^5cdf^2j +
  10a^3c^3df^2j + 6a^6ef^2j - 7a^4c^2ef^2j -
  12a^2c^4ef^2j + c^6ef^2j + 10a^3bcf^4j + ≪669≫ +
  11b^2dgij^5 - 4bd^2gij^5 - d^3gij^5 - 4be^2gij^5 +
  16de^2gij^5 + 8bg^3ij^5 - 8dg^3ij^5 + 7b^2ghij^5 -
  14bdghij^5 + 7d^2ghij^5 - 12e^2ghij^5 +
  5b^2ei^2j^5 - 5d^2ei^2j^5 + 8eg^2i^2j^5 -
  10behi^2j^5 + 10dehi^2j^5 + 8bgi^3j^5 - 8dgi^3j^5 - 4ei^4j^5

ca8 = Coefficient[alfa, α, 8]; Short[ca8, 10]
a^6bcfj + a^4bc^3fj - a^2bc^5fj - bc^7fj - 2a^6cdfj -
  4a^4c^3dfj - 2a^2c^5dfj - a^7efj + a^5c^2efj +
  5a^3c^4efj + 3ac^6efj - 5a^4bcf^3j - 6a^2bc^3f^3j - bc^5f^3j +
  5a^4cdf^3j + 6a^2c^3df^3j +
  c^5df^3j + 6a^5ef^3j + 4a^3c^2ef^3j - 2ac^4ef^3j + ≪554≫ +
  4de^2ghj^5 - b^3eij^5 + b^2deij^5 +
  bd^2eij^5 - d^3eij^5 - 4be^3ij^5 - 4de^3ij^5 -
  12beg^2ij^5 + 12deg^2ij^5 + 2b^2ehij^5 - 4bdehij^5 +
  2d^2ehij^5 + 8e^3hij^5 - 5b^2gi^2j^5 + 10bdgi^2j^5 -
  5d^2gi^2j^5 + 4e^2gi^2j^5 + 4bei^3j^5 - 4dei^3j^5
```

```
ca9 = Coefficient[alfa, α, 9]; Short[ca9, 10]
a^5bcf^2j + 2a^3bc^3f^2j + abc^5f^2j - a^5cdf^2j - 2a^3c^3df^2j -
 ac^5df^2j - a^6ef^2j - a^4c^2ef^2j + a^2c^4ef^2j +
 c^6ef^2j + a^7fgj + 3a^5c^2fgj + 3a^3c^4fgj + ac^6fgj - a^6cfij -
 3a^4c^3fij - 3a^2c^5fij - c^7fij -
 2a^3b^2cf^2j^2 + 2ab^2c^3f^2j^2 + ≪211≫ + acd^2i^2j^4 +
 2a^2bei^2j^4 - 2bc^2ei^2j^4 - 2a^2dei^2j^4 +
  2c^2dei^2j^4 + 12ace^2i^2j^4 + b^2eg^2j^5 - 2bdeg^2j^5 + d^2eg^2j^5 +
 4e^3g^2j^5 + b^3gij^5 - 3b^2dgij^5 +
 3bd^2gij^5 - d^3gij^5 + 4be^2gij^5 - 4de^2gij^5 - b^2ei^2j^5 +
 2bdei^2j^5 - d^2ei^2j^5 - 4e^3i^2j^5

ca10 = Coefficient[alfa, α, 10]
0
```

Therefore our polynomial for α can be written as

```
A = (ca2 + ca3α + ca4α^2 + ca5α^3 + ca6α^4 + ca7α^5 + ca8α^6 + ca9α^7)α^2;
```

Let us check these coefficients

```
(alfa - A)//Simplify
0
```

Now, the coefficients of the polynomial of β

```
cb0 = Coefficient[beta, β, 0]
0

cb1 = Coefficient[beta, β, 1]
0

cb2 = Coefficient[beta, β, 2]; Short[cb2, 10]
 - a^5bc^2fj + a^6cefj - 2a^3bc^2f^3j + 3a^4cef^3j - abc^2f^5j +
 3a^2cef^5j + cef^7j -
 a^7cgj - 3a^5cf^2gj - 3a^3cf^4gj - acf^6gj + a^5c^2fhj +
 2a^3c^2f^3hj + ac^2f^5hj + a^6c^2ij +
 a^4c^2f^2ij - a^2c^2f^4ij - c^2f^6ij + ≪214≫ +
 12afg^2i^2j^4 - 4cefhi^2j^4 - 4acghi^2j^4 -
 8acei^3j^4 + 8cfgi^3j^4 - b^3egj^5 + 3b^2eghj^5 - 3begh^2j^5 +
 egh^3j^5 + b^2e^2ij^5 - b^2g^2ij^5 -
 2be^2hij^5 + 2bg^2hij^5 + e^2h^2ij^5 - g^2h^2ij^5 -
 4begi^2j^5 + 4eghi^2j^5 + 4e^2i^3j^5 - 4g^2i^3j^5
```

```
cb3 = Coefficient[beta, β, 3]; Short[cb3, 10]
a^5 bc^3 j + a^7 cdj - 2a^6 c^2 ej - 2a^5 bcf^2 j + 6a^3 bc^3 f^2 j +
 3a^5 cdf^2 j + a^6 ef^2 j - 10a^4 c^2 ef^2 j -
 4a^3 bcf^4 j + 5abc^3 f^4 j + 3a^3 cdf^4 j + 3a^4 ef^4 j -
 14a^2 c^2 ef^4 j - 2abcf^6 j + acdf^6 j + 3a^2 ef^6 j -
 6c^2 ef^6 j + ef^8 j - a^7 fgj + 4a^5 c^2 fgj - 3a^5 f^3 gj + ≪553≫ +
 12be^2 gij^5 - 4bg^3 ij^5 + b^2 ghij^5 -
 4bdghij^5 - 12e^2 ghij^5 + 4g^3 hij^5 + bgh^2 ij^5 + 2dgh^2 ij^5 -
 gh^3 ij^5 + 2b^2 ei^2 j^5 + 4bdei^2 j^5 -
 12e^3 i^2 j^5 + 4eg^2 i^2 j^5 - 8behi^2 j^5 - 4dehi^2 j^5 +
 6eh^2 i^2 j^5 - 4bgi^3 j^5 + 8dgi^3 j^5 - 4ghi^3 j^5 + 8ei^4 j^5

cb4 = Coefficient[beta, β, 4]; Short[cb4, 10]
 - 6a^3 bc^4 fj + a^7 dfj - 5a^5 c^2 dfj - a^6 cefj + 11a^4 c^3 efj -
 a^5 bf^3 j + 10a^3 bc^2 f^3 j - 10abc^4 f^3 j +
 3a^5 df^3 j - 10a^3 c^2 df^3 j - 8a^4 cef^3 j + 26a^2 c^3 ef^3 j -
 2a^3 bf^5 j + 10abc^2 f^5 j + 3a^3 df^5 j -
 5ac^2 df^5 j - 13a^2 cef^5 j + 15c^3 ef^5 j - abf^7 j + ≪665≫ +
 10de^2 hij^5 - 10dg^2 hij^5 + 2b^2 h^2 ij^5 -
 bdh^2 ij^5 - d^2 h^2 ij^5 - 10e^2 h^2 ij^5 + 5g^2 h^2 ij^5 - bh^3 ij^5 + dh^3 ij^5 +
 16begi^2 j^5 - 12degi^2 j^5 -
 4eghi^2 j^5 + b^2 i^3 j^5 + 4bdi^3 j^5 - 4d^2 i^3 j^5 -
 20e^2 i^3 j^5 - 6bhi^3 j^5 + 4dhi^3 j^5 + h^2 i^3 j^5 + 4i^5 j^5

cb5 = Coefficient[beta, β, 5]; Short[cb5, 10]
2a^5 bc^3 j + 2a^3 bc^5 j + 2a^7 cdj + 2a^5 c^3 dj - 4a^6 c^2 ej -
 4a^4 c^4 ej - 3a^5 bcf^2 j - 6a^3 bc^3 f^2 j +
 10abc^5 f^2 j - a^5 cdf^2 j + 12a^3 c^3 df^2 j + a^6 ef^2 j + 3a^4 c^2 ef^2 j -
 24a^2 c^4 ef^2 j + 2a^3 bcf^4 j -
 20abc^3 f^4 j - 8a^3 cdf^4 j + 10ac^3 df^4 j +
 2a^4 ef^4 j + ≪771≫ + 8d^2 ghij^5 + 12e^2 ghij^5 -
 8g^3 hij^5 + 7bgh^2 ij^5 - 7dgh^2 ij^5 + 3b^2 ei^2 j^5 - 8bdei^2 j^5 +
 8d^2 ei^2 j^5 + 12e^3 i^2 j^5 +
 8eg^2 i^2 j^5 + 2behi^2 j^5 - 8dehi^2 j^5 + 3eh^2 i^2 j^5 + 4bgi^3 j^5 +
 4dgi^3 j^5 - 8ghi^3 j^5 - 4ei^4 j^5

cb6 = Coefficient[beta, β, 6]; Short[cb6, 10]
3a^5 bc^2 fj - 2a^3 bc^4 fj - 5abc^6 fj + 2a^7 dfj -
 4a^5 c^2 dfj - 6a^3 c^4 dfj - 5a^6 cefj + 6a^4 c^3 efj +
 11a^2 c^5 efj - 2a^5 bf^3 j + 6a^3 bc^2 f^3 j + 20abc^4 f^3 j +
 4a^5 df^3 j + 6a^3 c^2 df^3 j - 10ac^4 df^3 j -
 9a^4 cef^3 j - 18a^2 c^3 ef^3 j + ≪768≫ +
 8be^2 hij^5 + 8de^2 hij^5 - 14bg^2 hij^5 + 14dg^2 hij^5 +
 2b^2 h^2 ij^5 - 4bdh^2 ij^5 + 2d^2 h^2 ij^5 - 8e^2 h^2 ij^5 +
 8begi^2 j^5 - 12degi^2 j^5 + 4eghi^2 j^5 +
 2b^2 i^3 j^5 + 2bdi^3 j^5 - 4d^2 i^3 j^5 - 12e^2 i^3 j^5 +
 8g^2 i^3 j^5 - 6bhi^3 j^5 + 6dhi^3 j^5 + 4i^5 j^5
```

```
cb7 = Coefficient[beta, β, 7]; Short[cb7, 10]
a^5bc^3j + 2a^3bc^5j + abc^7j + a^7cdj + 2a^5c^3dj + a^3c^5dj -
 2a^6c^2ej - 4a^4c^4ej - 2a^2c^6ej -
 10a^3bc^3f^2j - 10abc^5f^2j - 5a^5cdf^2j + 5ac^5df^2j -
 a^6ef^2j + 12a^4c^2ef^2j + 7a^2c^4ef^2j -
 6c^6ef^2j + 6a^3bcf^4j + ≪668≫ + 6d^3gij^5 -
 16be^2gij^5 + 4de^2gij^5 + 8bg^3ij^5 -
 8dg^3ij^5 - 7b^2ghij^5 + 14bdghij^5 - 7d^2ghij^5 +
 12e^2ghij^5 - 10bdei^2j^5 + 10d^2ei^2j^5 +
 20e^3i^2j^5 - 8eg^2i^2j^5 + 10behi^2j^5 -
 10dehi^2j^5 + 8bgi^3j^5 - 8dgi^3j^5 - 12ei^4j^5

cb8 = Coefficient[beta, β, 8]; Short[cb8, 10]
2a^5bc^2fj + 4a^3bc^4fj + 2abc^6fj + a^7dfj + a^5c^2dfj -
 a^3c^4dfj - ac^6dfj - 3a^6cefj - 5a^4c^3efj -
 a^2c^5efj + c^7efj - a^5bf^3j - 6a^3bc^2f^3j -
 5abc^4f^3j + a^5df^3j + 6a^3c^2df^3j + 5ac^4df^3j +
 2a^4cef^3j - 4a^2c^3ef^3j - 6c^5ef^3j + a^7cgj + ≪550≫ +
 8bde^2ij^5 - 6d^2e^2ij^5 - 8e^4ij^5 +
 5b^2g^2ij^5 - 10bdg^2ij^5 + 5d^2g^2ij^5 -
 4e^2g^2ij^5 - b^3hij^5 + 3b^2dhij^5 - 3bd^2hij^5 + d^3hij^5 -
 4be^2hij^5 + 4de^2hij^5 - 12begi^2j^5 +
 12degi^2j^5 + b^2i^3j^5 - 2bdi^3j^5 + d^2i^3j^5 + 12e^2i^3j^5

cb9 = Coefficient[beta, β, 9]; Short[cb9, 10]
a^5bcf^2j + 2a^3bc^3f^2j + abc^5f^2j - a^5cdf^2j -
 2a^3c^3df^2j - ac^5df^2j - a^6ef^2j - a^4c^2ef^2j +
 a^2c^4ef^2j + c^6ef^2j + a^7fgj + 3a^5c^2fgj + 3a^3c^4fgj +
 ac^6fgj - a^6cfij - 3a^4c^3fij - 3a^2c^5fij -
 c^7fij - 2a^3b^2cf^2j^2 + 2ab^2c^3f^2j^2 + ≪211≫ +
 acd^2i^2j^4 + 2a^2bei^2j^4 - 2bc^2ei^2j^4 - 2a^2dei^2j^4 +
 2c^2dei^2j^4 + 12ace^2i^2j^4 + b^2eg^2j^5 - 2bdeg^2j^5 +
 d^2eg^2j^5 + 4e^3g^2j^5 + b^3gij^5 - 3b^2dgij^5 +
 3bd^2gij^5 - d^3gij^5 + 4be^2gij^5 - 4de^2gij^5 - b^2ei^2j^5 +
 2bdei^2j^5 - d^2ei^2j^5 - 4e^3i^2j^5

cb10 = Coefficient[beta, β, 10]
0
```

then

```
B = (cb2 + cb3β + cb4β^2 + cb5β^3 + cb6β^4 + cb7β^5 + cb8β^6 + cb9β^7) β^2;
(β - B)//Simplify
0
```

As illustration let us consider a synthetic dataset generated from a multivariate Gauss distribution. Regular 3D points have means of (2, 8, 6) and variances (5, 5, 0.01), see Nurunnabiet al. (2012). The ideal plane is $z = 6$. In this case let us consider here $N = 20{,}000$ points. There are additional 3000 points, which represent outliers.

```
Clear[XYZ]
N = 20000;
XYZ = Table[{Null,Null,Null},{i,1,N}];
Do[XYZ[[i]] = RandomVariate[MultinormalDistribution[{2.,8,6},
 {{5.,0,0},{0,5.,0},{0,0,0.01}}]],{i,1,N}];
p1 = Plot3D[6,{x,-8,12},{y,-2,18},BoxRatios → {1,1,0.5},
 ClippingStyle → None,Mesh → None,PlotRange → {5.5,6.5}];
p2 = ListPointPlot3D[XYZ,PlotStyle → Blue];
NOutLiers = 3000;
```

Now, let us generate 3000 outliers randomly (Fig. 13.1).

```
XYZOutLiers = Table[{Null,Null,Null},{i,1,NOutLiers}];
Do[XYZOutLiers[[i]] =
 RandomVariate[MultinormalDistribution[{15.,15,10},
 {{10.,0,0},{0,2.,0},{0,0,1}}]],{i,1,NOutLiers}];
p1 = Plot3D[6,{x,-8,35},{y,-2,19},BoxRatios → {1,1,0.5},
 ClippingStyle → None,Mesh → None,PlotRange → {4.5,12}];
p3 = ListPointPlot3D[XYZOutLiers,PlotStyle → Red];
Show[{p1,p2,p3}]

XYZC = Join[XYZ,XYZOutLiers];
X = XYZC[[All,1]];Y = XYZC[[All,2]];Z = XYZC[[All,3]];
```

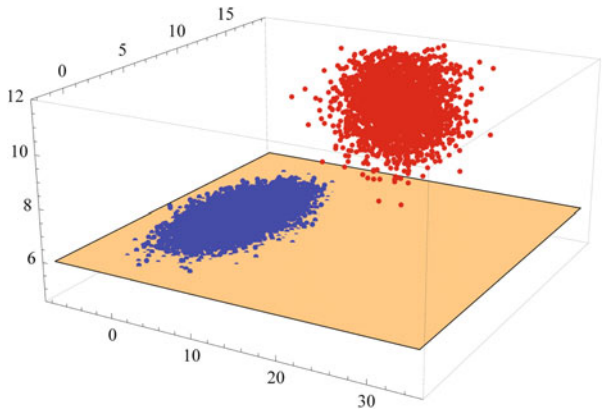


Fig. 13.1 The plane and the artificial measured points with outliers (red)

The number of data

```
N = Length[XYZC]
23000
```

We use the same weight for all data, $W_i = 1$.

```
W = Table[1, {N}];
W2 = Map[#^2 &, W];
```

The coefficients are

```
a,b,c,d,e,f,g,h,i,j = Total/@
 Map[MapThread[#1 #2&, #, W2]&, X, X^2, Y, Y^2, XY, Z, YZ, Z^2, XZ, W2];
```

The monomial for α

```
AbsoluteTiming[pα =
 ca2 + ca3α + ca4α^2 + ca5α^3 + ca6α^4 + ca7α^5 + ca8α^6 + ca9α^7;]
{0.00911141, Null}
```

```
pα
1.23806 × 10^48 - 7.11611 × 10^48 α + 6.94251 × 10^48 α^2 -
 2.76003 × 10^49 α^3 + 3.84629 × 10^49 α^4 - 2.98505 × 10^49 α^5 -
 8.5202 × 10^48 α^6 + 5.1252 × 10^47 α^7
```

The real roots of this monomial are

```
AbsoluteTiming[solα = Reduce[pα == 0, α, Reals];]
{0.00498473, Null}
```

```
solα
α == - 3.94057 || α == 0.188435 || α == 19.4301
```

We select the smaller positive one

```
ToRules[solα[[2]]]
{α → 0.188435}
```

Similarly the monomial for β and γ, and their solutions are

```
AbsoluteTiming[pβ =
 cb2 + cb3β + cb4β^2 + cb5β^3 + cb6β^4 + cb7β^5 + cb8β^6 + cb9β^7;]
{0.00841855, Null}
```

```
pβ
-5.02172 × 10^48 + 3.87559 × 10^49 β - 3.22113 × 10^47 β^2 +
  6.86247 × 10^49 β^3 + 1.75699 × 10^49 β^4 + 2.78598 × 10^49 β^5 +
  1.96989 × 10^49 β^6 + 5.1252 × 10^47 β^7

solβ = Reduce[pβ == 0, β, Reals]
β == -36.9881 || β == -2.04298 || β == 0.126022

ToRules[solβ[[3]]]
{β → 0.126022}

pγ = Solve[eq3 == 0, γ]//Flatten
{γ → -0.0000434783 (-149980. + 84667.8 α + 204930. β)}

%/.ToRules[solα[[2]]]/.ToRules[solβ[[3]]]
{γ → 4.70437}
```

13.1.2.2 Solution via Numerical Gröbner Basis

Instead of solving the equation system developed from the necessary conditions in symbolic way, we can solve this algebraic system in numeric form, too.

```
eq1
688361.2 + 138096.3α - 688361.2α^2 - 993513.5β - 2819366.6αβ +
 993513.5α^2β + 688361.2β^2 + 1175652.4αβ^2 - 993513.5β^3 - 84667.86γ -
 299960.98αγ + 84667.8α^2γ + 409859.5αβγ - 84667.8β^2γ + 23000αγ^2

eq2
1409683.3 - 993513.5α + 1409683.3α^2 - 993513.5α^3 - 1037556.1β -
 1376722.4αβ - 1175652.4α^2β - 1409683.3β^2 + 993513.5αβ^2 - 204929.78γ -
 204929.7α^2γ - 299960.9βγ + 169335.6αβγ + 204929.7β^2γ + 23000βγ^2

eq3
149980.5 - 84667.8α - 204929.7β - 23000γ

AbsoluteTiming[solαβγ = NSolve[{eq1, eq2, eq3}, {α, β, γ}, Reals];]
{0.0869599, Null}
```

The running time is acceptably short.

```
solαβγ
{{α → -1.10422 × 10^15, β → -5.71182 × 10^14, γ → 9.15411 × 10^15},
 {α → 2.16013 × 10^13, β → -4.17602 × 10^13, γ → 2.92564 × 10^14},
 {α → 19.4301, β → -36.9881, γ → 264.558},
 {α → -3.94057, β → -2.04298, γ → 39.2299},
 {α → 0.1884345, β → 0.126022, γ → 4.70437}}
```

How can we select the proper real solution? Let us consider the values of the formal objective function based on the different real solutions

```
Lαβγ = Map[L/.#&, solαβγ]/.σ → 1//N
{-369601., -76614.4, -76645., -387030., -26180.4}
```

The real solution will be selected according to which maximizes the objective function. We seek for the real global maximum of the maximum likelihood function. The value of σ will not influence the relative comparison of the real solutions, therefore we consider $\sigma = 1$. The different real solutions of the polynomial system represent the different real local maximums. Therefore the optimal solution, which gives the highest value for the maximum likelihood function is

```
solopt = solαβγ[[Position[Lαβγ, max[Lαβγ]]//Flatten//First]]
{α → 0.188435, β → 0.126022, γ → 4.70437}
```

It is reasonable to encapsulate these statements in a function, namely

```
PlaneTLSW[XYZ_, W_] :=
 Module[{X, Y, Z, a, b, c, d, e, f, g, h, i, j, N, W2, eq1, eq2, eq3, solαβγ,
  Lαβγ, σ, L, solopt},
 X = XYZ[[All, 1]]; Y = XYZ[[All, 2]]; Z = XYZ[[All, 3]];
  N = Length[XYZ];
  W2 = Map[#^2 &, W];
  {a, b, c, d, e, f, g, h, i, j} =
   Total/@Map[MapThread[#1#2&, {#, W2}]&,
    X, X^2, Y, Y^2, XY, Z, YZ, Z^2, XZ, W2];
  eq1 = jαγ^2 + 2α^2γa - γa(1 + α^2 + β^2) + α^3b - αb(1 + α^2 + β^2) + 2αβγc +
    αβ^2d + 2α^2βe - (1 + α^2 + β^2) - 2α - 2αβg + αh - 2α^2i +
    i(1 + α^2 + β^2);
  eq2 = jβγ^2 + 2αβγa + α^2βb + 2β^2γc - (1 + α^2 + β^2) + 2αβ^2e -
    αe(1 + α^2 + β^2) + β^3d - (1 + α^2 + β^2) - 2β - 2αβi + βh - 2β^2g +
    g(1 + α^2 + β^2);
  eq3 = f - jγ - αa - βc;
  solαβγ = NSolve[{eq1, eq2, eq3}, {α, β, γ}, Reals];
  L = -jγ^2/(2(1 + α^2 + β^2)σ^2) - 1/2 Nlog[2] - 1/2 Nlog[π] - Nlog[σ] -
   a αγ/((1 + α^2 + β^2)σ^2) - α^2b/(2(1 + α^2 + β^2)σ^2) - βγc/((1 + α^2 + β^2)σ^2) -
   αβe/((1 + α^2 + β^2)σ^2) - β^2d/(2(1 + α^2 + β^2)σ^2) + γf/((1 + α^2 + β^2)σ^2) +
   αi/((1 + α^2 + β^2)σ^2) + βg/((1 + α^2 + β^2)σ^2) - h/(2(1 + α^2 + β^2)σ^2);
  Lαβγ = Map[L/.#&, solαβγ]/.σ → 1//N;
solopt = solαβγ[[Position[Lαβγ, max[Lαβγ]]//Flatten//First]]]
```

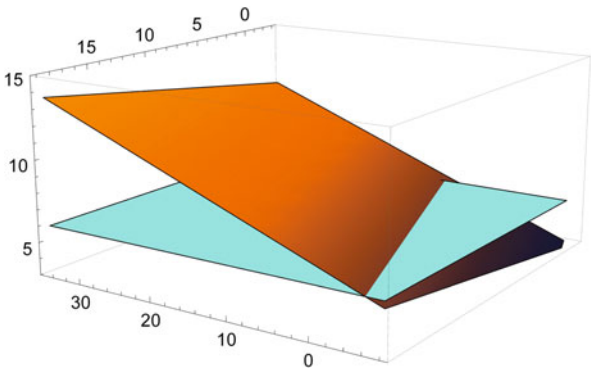


Fig. 13.2 The original plane (green) and the estimated plane (rust) in case of outliers

Now, let us employ it,

```
PlaneTLSW[XYZC,W]
{α → 0.188435, β → 0.126022, γ → 4.70437}
```

Then the estimated plane can be visualized,

```
p4 = Plot3D[αx + βy + γ/.%,{x, -8,35},{y, -3,19},
 BoxRatios → {1,1,0.5},ClippingStyle → None,Mesh → None,
 PlotRange → {3,15},ColorFunction →"RustTones"];
Show[{p4,p1}]
```

The result of this soft method is not good. Although it would have been possible to give ad hoc values for the weights, it is difficult to find their proper ones (Fig. 13.2).

In the next section we shall demonstrate how the Danish method can provide the proper weights.

13.1.3 Danish Algorithm

13.1.3.1 Basic Idea

The Danish method was proposed by Krarup et al. (1980) and is purely heuristic with no rigorous statistical theory. The method works with adaptive weights, which are altered iteratively. The weight of an observation is computed according to its error (residual), which is different from 1 if the error of this observation is greater than the standard deviation of the error distribution, σ_ϵ, namely

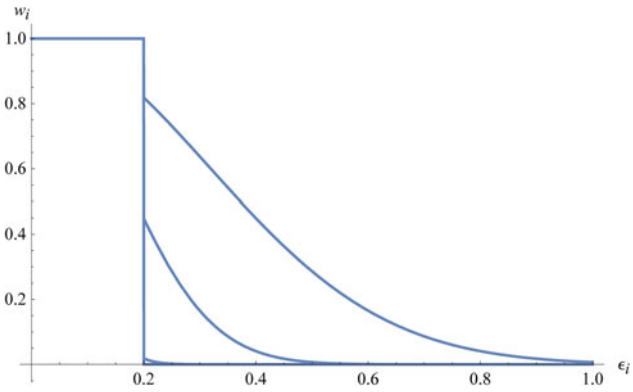


Fig. 13.3 The actual weights of the observations in case of $s_? = 0.1$ and with different c_1 values

$$w_i^{(k+1)} = \exp\left(-c_1\left(\varepsilon_i^{(k)}\right)^2\right) \quad \textit{if} \quad \varepsilon_i^{(k)} \geq \frac{1}{c_2}\sigma_\epsilon, \quad \textit{otherwise} \quad w_i^{(k+1)} = 1.$$

where $\varepsilon_i^{(k)}$ is the error of the i-th observation computed with weight $w_i^{(k)}$ in the k-th iteration step. The error definition is the same as before,

$$\varepsilon_i = \frac{|z_i - \alpha x_i - \beta y_i - \gamma_s|}{\sqrt{1+\alpha^2+\beta^2}}.$$

Here c_2 is a suitable constant. For example in this study, let $\sigma_\varepsilon = 0.1$ and $c_2 = 1/2$. Therefore $(1/c_2)\sigma_\varepsilon = 0.2$, so $\varepsilon_i^{(k)} \geq 0.2$. In this study c_1 will be 5, 20, 100, 1000; see Fig. 13.3.

```
Plot[Map[If[εi ≥ 0.2, exp[-#εi^2], 1]&, {5, 20, 100, 1000}],
 {εi, 0, 1}, PlotRange → All, AxesLabel → {"εi", "wi"}]
```

So when $c_1 = 1000$, we have $w_i \approx 0$ weight for outliers.

13.1.3.2 Danish Algorithm with Gröbner Basis

We want to *combine numerical Gröbner basis solution* into with the solution of the maximum likelihood estimation with *Danish algorithm* therefore we need to introduce a certain initialization phase. First we define the maximum number of the iterations,

```
N = 10; (*Number of iterations*)
```

The number of observations,

```
NC = Length[XYZC]
23000
```

The initial weights of the observations

```
W = Table[1, {NC}]; (*initialization of weights *)
```

In order to follow the change of weights during the iteration, these weights will be collected in every iteration step,

```
CollectorW = {W}; (*initialization of the weights collertor*)
```

Similarly, we should like to follow the convergence of the solution of the estimated parameters. These values will be collected in every iteration step, too.

```
CollectorSols = {};
```

The parameter c_1 is

```
c1 = 1000;
```

Then the cycle for the iteration can be written as

```
AbsoluteTiming[For[J = 1, J <= N, J++,
 sols = PlaneTLSW[XYZC, W];
 Error = Map[((#[[3]] - α#[[1]] - β#[[2]] - γ)/(√(1 + α^2 + β^2)))/.
  sols&, XYZC];
 StD = StandardDeviation[Error];
 W = Table[If[(Abs[Error[[i]]] < StD), 1,
  Exp[-c1 Error[[i]]^2]], {i, 1, NC}];
 CollectorW = Append[CollectorW, W];
 CollectorSols = Append[CollectorSols, {α, β, γ}/.sols];
];]
{9.55748, Null}
```

The next Figs. 13.4, 13.5, 13.6 show the convergence of the different estimated parameters as a function of the number of the iterations.

```
ListPlot[Transpose[CollectorSols][[1]],
 Joined → True, Joined → True, PlotRange → All,
 AxesLabel → {"Number of iterations", "α"}]
```

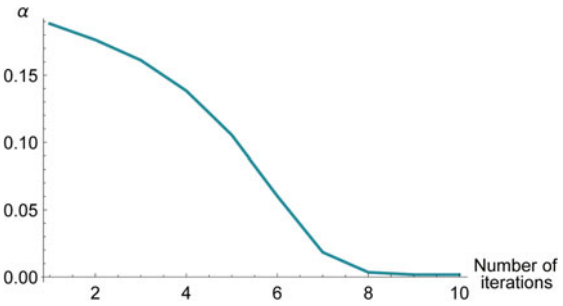


Fig. 13.4 The convergence of the parameter α

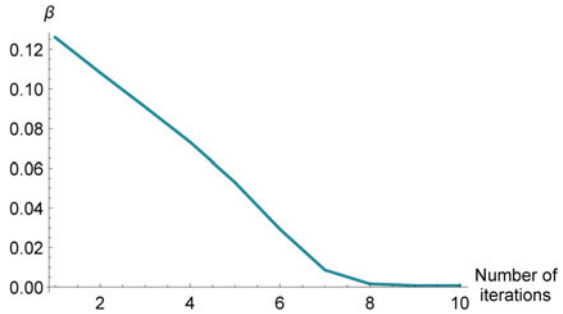


Fig. 13.5 The convergence of the parameter β

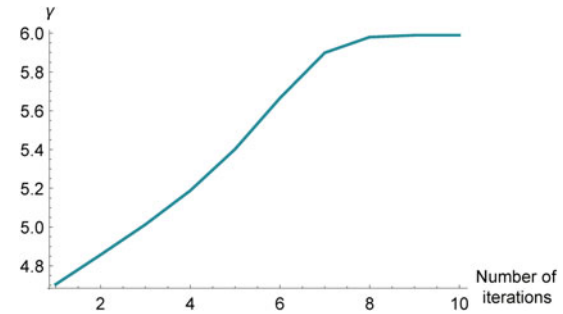


Fig. 13.6 The convergence of the parameter γ

```
ListPlot[Transpose[CollectorSols][[2]],
 Joined → True,Joined → True, PlotRange → All,
 AxesLabel → {"Number of iterations","β"}]

ListPlot[Transpose[CollectorSols][[3]],
 Joined → True,Joined → True, PlotRange → All,
 AxesLabel → {"Number of iterations","γ"}]
```

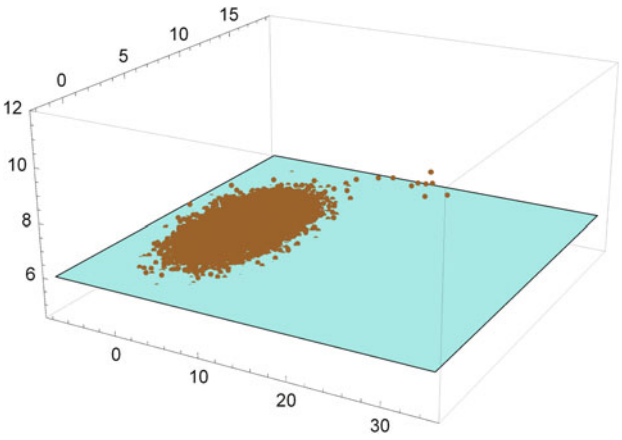


Fig. 13.7 The original plane and the inliers computed via Danish method

The result of the iteration

```
W = Last[CollectorW];
sols = PlaneTLSW[XYZC,W]
{α → 0.00169951, β → 0.000787971, γ → 5.98921}
```

The observations, the inlier data points which remained active,

```
inliers = Select[MapThread[If[#1 < 0.1,0,#2]&,{W,XYZC}],
 Length[#] == 3&];
Length[inliers]
20011

p5 = ListPointPlot3D[inliers,PlotStyle → Brown];
Show[{p1,p5}]
```

Only eleven points remained from the outliers (Fig. 13.7).

13.1.4 Danish Algorithm with PCA

The Danish algorithm can cure other soft algorithms too, and make them proper for robust estimation. Let us consider the Principal Component Analysis (PCA), which is a popular statistical technique that describes the covariance structure of data by means of a small number of components. These components are linear combinations of the original variables that rank the variability in the data through the variances, and produce directions using the eigenvectors of the covariance matrix.

The eigenvector corresponding to the smallest eigenvalue is exactly the normal of the best-fitted plane.

In order to employ PCA in the Danish algorithm, one should extend the traditional PCA algorithm to weighted PCA (PCAW), see Weingarten (2004).

First, we compute the weighted center of gravity,

$$c_0 = \frac{\sum_{i=1}^{N} \tilde{w}_i \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}}{\sum_{i=1}^{N} \tilde{w}_i}.$$

Let us shift the data of observations to this point,

$$C_{c_0 i} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - c_0, \quad i = 1, 2, \ldots N.$$

The weighted covariance matrix is symmetric,

$$A = \begin{pmatrix} \sum_i \tilde{w}_i (C_{c_0 i})_1^2 & \sum_i \tilde{w}_i (C_{c_0 i})_1 (C_{c_0 i})_2 & \sum_i \tilde{w}_i (C_{c_0 i})_1 (C_{c_0 i})_3 \\ \cdot & \sum_i \tilde{w}_i (C_{c_0 i})_2^2 & \sum_i \tilde{w}_i (C_{c_0 i})_2 (C_{c_0 i})_3 \\ \cdot & \cdot & \sum_i \tilde{w}_i (C_{c_0 i})_3^2 \end{pmatrix}.$$

Considering the Hesse form, the weighted residual to be minimized can be written,

$$R\big(n_x, n_y, n_z, d\big) = \sum_{i=1}^{N} \tilde{w}_i \big(n_x x_i + n_y y_i + n_z z_i - d\big)^2.$$

Remark Remember that in the estimator based on the maximization of the likelihood function we applied $w_i = \sqrt{\tilde{w}_i}$.

Therefore considering that

$$\frac{\partial R}{\partial d} = 0$$

we get

$$n_x \sum_{i=1}^{N} \tilde{w}_i x_i + n_y \sum_{i=1}^{N} \tilde{w}_i y_i + n_z \sum_{i=1}^{N} \tilde{w}_i z_i - d \sum_{i=1}^{N} \tilde{w}_i = 0.$$

Since the plane normal can be computed as the eigenvectors of the weighted covariance matrix A, the surface parameter d is determined. Consequently the surface is

$$z = -\frac{n_x}{n_z}x - \frac{n_y}{n_z}y + \frac{d}{n_z}.$$

Now, let us check our algorithm with the weights provided by the computation done above in Sect. 13.1.3.2.

First let us compute $\tilde{w}_i = w_i^2$,

```
Ws = Map[#^2 &, W];
```

The weighted data of the observations,

```
XYZW = MapThread[#1 #2&, {XYZC, Ws}];
```

therefore the weighted center of gravity,

```
c0 = Map[Total[#]&, Transpose[XYZW]]/Total[Ws]
{1.98748, 8.01922, 6.00089}
```

Then the shifted data

```
Cc0 = Map[# - c0&, XYZC];
```

Preparation of the shifted data in order to compute their weighted covariance matrix,

```
Cc0W = MapThread[#1 Sqrt[#2] &, {Cc0, Ws}];
```

Then the covariance matrix

```
AbsoluteTiming[A = Covariance[Cc0W]; MatrixForm[A]]
```

$$\left\{0.0903777, \begin{pmatrix} 4.40013 & 0.0429781 & 0.00899076 \\ 0.0429781 & 4.39533 & 0.002346 \\ 0.00899076 & 0.002346 & 0.0093104 \end{pmatrix}\right\}$$

The eigenvector having $nz = 1$ is choosen, see the expression of $z = z(x,y)$ above,

```
{nx, ny, nz} = Eigenvectors[A][[3]]
{-0.00204257, -0.000514862, 0.999998}
```

Therefore the parameter d can be computed

```
sold = Solve[{nx,ny,nz}.Total[XYZW] - d Total[Ws] == 0,d]]//
 Flatten
{d → 5.99269}
```

Consequently the coefficients of surface model are

$$\left\{-\frac{nx}{nz}, -\frac{ny}{nz}, \frac{d}{nz}\right\}/.sold$$

```
{0.00204258, 0.000514863, 5.9927}
```

We want to implement the weighted PCA into the Danish method.

First let us summarize the steps of computations of the PCAW algorithm in a *Mathematica* function,

```
PlanePCAW[XYZ_,W_]:=
 Module[{Ws,XYZW,co,Cco,CcoW,nx,ny,nz,d,sold},
 Ws = Map[#^2&,W];
 XYZW = MapThread[#1 #2&,{XYZ,Ws}];
 co = Map[Total[#]&,Transpose[XYZW]]/Total[Ws];
 Cco = Map[# - c0&,XYZ];
 CcoW = MapThread[#1 #2&,{Cc0,W}];
 {nx,ny,nz} = Eigenvectors[Covariance[CcoW]][[3]];
 sold =
  Solve[{nx,ny,nz}.Total[XYZW] - d Total[Ws] == 0,d]//Flatten;
 {α → -nx/nz, β → -ny/nz, γ → d/nz}/.sold]
```

Let us check it

```
PlanePCAW[XYZC,W]
{α → 0.00204258, β → 0.000514863, γ → 5.9927}
```

Now, the integration of PCAW into the Danish algorithm is similar to that of the numerical Gröber basis solution, see above

```
N = 10;(*Number of iterations*)
```

The number of observations,

```
NC = Length[XYZC]
23000
```

The initial weights of the observations

```
W = Table[1, {NC}];(*initialization of weights *)
```

In order to follow the change of weights during the iteration, these weights will be collected in every iteration step,

```
CollectorW = {W};(*initialization of the weights collertor*)
```

Similarly, we should like to follow the convergence of the solution of the estimated parameters. These values will be collected in every iteration step, too.

```
CollectorSols = {};
```

The parameter c_1 is

```
c1 = 1000;
```

Then the cycle for the iteration can be written as

```
AbsoluteTiming[For[j = 1, j ≤ N, j + + ,
 sols = PlanePCAW[XYZC, W];
 Error = Map[((#[[3]] - α#[[1]] - β#[[2]] - γ)/(√(1 + α^2 + β^2)))/.
 sols&, XYZC];
 StD = StandardDeviation[Error];
 W = Table[If[(Abs[Error[[i]]] < StD), 1, Exp[-c1]Error[[i]]^2]],
 {i, 1, NC}];
 CollectorW = Append[CollectorW, W];
 CollectorSols = Append[CollectorSols, {α, β, γ}/.sols];
];]
{7.23155, Null}
```

The computation time is somewhat longer than the Gröbner solution' time was. The next figures show the convergence of the different parameters as a function of the number of the iterations (Figs. 13.8, 13.9, 13.10).

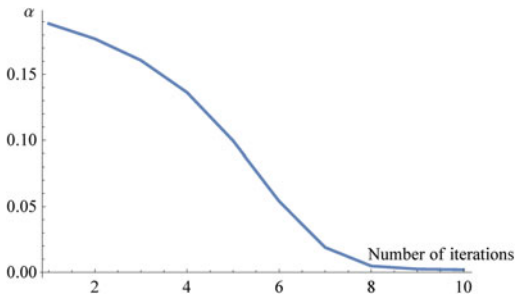


Fig. 13.8 The convergence of the parameter α

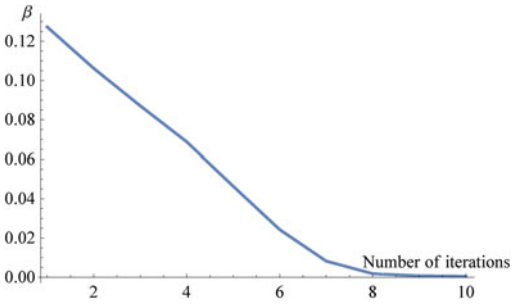


Fig. 13.9 The convergence of the parameter β

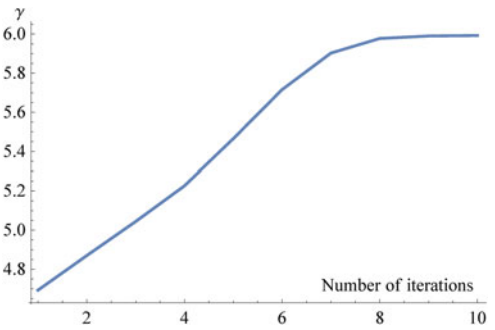


Fig. 13.10 The convergence of the parameter γ

```
ListPlot[Transpose[CollectorSols][[1]],Joined → True,
 Joined → True,PlotRange → All,
 AxesLabel → {"Number of iterations","α"}]

ListPlot[Transpose[CollectorSols][[2]],Joined → True,
 Joined → True,PlotRange → All,
 AxesLabel → {"Number of iterations","β"}]

ListPlot[Transpose[CollectorSols][[3]],Joined → True,
 Joined → True,PlotRange → All,
 AxesLabel → {"Number of iterations","γ"}]
```

The result of the iteration

```
W = Last[CollectorW];
sols = PlanePCAW[XYZC,W]
{α → 0.00204258,β → 0.000514863,γ → 5.9927}
```

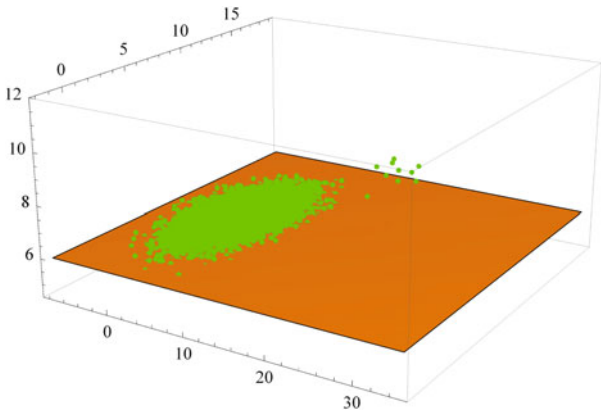


Fig. 13.11 The original plane and the inliers computed via Danish method with weighted PCA

The observations which remained active (inliers),

```
inliers = Select[MapThread[If[#1 < 0.1, 0, #2] &, {W, XYZC}],
 Length[#] == 3 &];
Length[inliers]
20010

p6 = ListPointPlot3D[inliers, PlotStyle → Green];
Show[{p1, p6}]
```

The result is the same as that of the Danish method combined with M-estimation (Fig. 13.11).

13.1.5 RANSAC Algorithm

13.1.5.1 Basic Idea of RANSAC

Let us try to apply the RANSAC method, given in Zuliani (2012), which has proven to be successful for detecting outliers. The basic RANSAC algorithm is as follows:

(1) Pick up a model type (M)
(2) Input data as

`dataQ`	data corrupted with outliers (cardinality (`dataQ`) = n)
s	number of data elements required per subset
N	number of subsets to draw from the data
t	threshold which defines if data element, $d_i \in$ `dataQ`, agrees with the model M

Remarks In special case s can be the minimal number of the data which results in a determined system for the unknown parameters of the model.

The number of subsets to draw from the data, N is chosen high enough to ensure that at least one of the subsets of the random examples does not include an outlier (with the probability p, which is usually set to 0.99). Let u represent the probability that any selected data point is an inlier and $v = 1 - u$ the probability of observing an outlier. Then the iterations N can be computed as

$$N = \frac{\log(1 - p)}{\log(1 - (1 - v)^s)}$$

(3) maximalConsensusSet ← Ø
(4) Iterate N times:

   (a) ConsensusSet ← Ø
   (b) Randomly draw a subset containing s elements and estimate the parameters of the model M
   (c) For each data element, $d_i \in$ `dataQ`:
   if agree (d_i, M, t), ConsensusSet ← d_i
   (d) if cardinality (maximalConsensusSet) < cardinality(ConsensusSet), maximalConsensusSet ← ConsensusSet

(5) Estimate model parameters using maximalConsensusSet.

13.1.5.2 The Necessary Number of Iterations

In our case we have 3 parameters of the plane to determine, so we need minimum 3 equations, therefore

```
s = 3;
```

In this case the parameters can be computed directly from a linear system. However, in general if $s > 3$ we need linear least squares solution. Let

```
p = 0.99;
```

In addition let us assume that the probability of observing outlier is

```
ν = 1/2;
```

Then the number of the necessary iterations is

```
N = Round[Log[1 - p]/Log[1 - (1 - ν)^s]]
34
```

13.1.5.3 Threshold

Let the threshold t

```
τ = 10;
```

An element $d_i \in$ `dataQ` will be accepted if the local error is less than the pre-specified threshold

$$\frac{|z_i - \alpha_s x_i - \beta_s y_i - \gamma_s|}{\sqrt{1 + \alpha_s^2 + \beta_s^2}} < \tau$$

Remarks d_i is represented by the triplet (x_i, y_i, z_i) and the parameters (a_s, β_s, γ_s) computed from a subset of s elements.

Here ordinary least square (OLS_z) estimation could be also applied as maximum support, see Fischler and Bolles (1981).

13.1.5.4 Computation of the Method Step by Step

```
inliers = Table[{}, {N}](* the consensus sets, j = 1, 2, ...N*)
{{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {},
 {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}}
```

Let us consider a subset (j = 1, 2, … N).

```
j = 2(*second iteration means second subset*)
2
```

Assigning noisy data to `dataQ`

```
dataQ = XYZC;
```

Let us select a triplet randomly

```
dataS = RandomSample[dataQ, s]
{{2.93055, 7.24291, 5.96181}, {1.26074, 5.85424, 5.93},
 {-0.126164, 7.10004, 6.10541}}
```

For further computation we need a function form of our algorithm.

13.1.5.5 RANSAC with Gröbner Basis

We want to integrate numerical Gröbner basis solution into the RANSAC algorithm, therefore it is reasonable to write a function for this suggested algorithm,

```
PlaneTLS[XYZ_] :=
 Module[{X, Y, Z, a, b, c, d, e, f, g, h, i, N, eq1, eq2, eq3, solαβγ,
  selsol, Lαβγ, L, solopt},
 X = XYZ[[All, 1]]; Y = XYZ[[All, 2]]; Z = XYZ[[All, 3]];
 {a, b, c, d, e, f, g, h, i} = Total/@{X, X^2, Y, Y^2, XY, Z, YZ, Z^2, XZ};
 N = Length[XYZ];
 eq1 = Nαγ^2 + 2α^2γa - γa(1 + α^2 + β^2) + α^3b - αb(1 + α^2 + β^2) + 2αβγc +
  αβ^2d + 2α^2βe - βe(1 + α^2 + β^2) - 2αγf - 2αβg + αh - 2α^2i + i(1 + α^2 + β^2);
 eq2 = Nβγ^2 + 2αβγa + α^2βb + 2β^2γc - γc(1 + α^2 + β^2) + 2αβ^2e - αe(1 + α^2 +
  β^2) + β^3d - βd(1 + α^2 + β^2) - 2βγf - 2αβi + βh - 2β^2g + g(1 + α^2 + β^2);
 eq3 = f - Nγ - αa - βc;
 solαβγ = NSolve[{eq1, eq2, eq3}, {α, β, γ}];
 selsol = Select[solαβγ, Im[#[[1]][[2]]] == 0&];
 L = -h/(2(1 + α^2 + β^2)σ^2) + iα/((1 + α^2 + β^2)σ^2) - bα^2/(2(1 + α^2 + β^2)σ^2) +
  gβ/((1 + α^2 + β^2)σ^2) - eαβ/((1 + α^2 + β^2)σ^2) - dβ^2/(2(1 + α^2 + β^2)σ^2) + fγ/((1 + α^2 + β^2)σ^2) -
  aαγ/((1 + α^2 + β^2)σ^2) - cβγ/((1 + α^2 + β^2)σ^2) - Nγ^2/(2(1 + α^2 + β^2)σ^2) -
  1/2 Nlog[2] - 1/2 Nlog[π] - Nlog[σ];
 Lαβγ = Map[L/.#&, selsol]/.σ → 1//N;
 solopt = selsol[[Position[Lαβγ, max[Lαβγ]]//Flatten//First]]]
```

Now let us compute the parameters from the randomly selected subset

```
sols = PlaneTLS[dataS]
{α → -0.0509108, β → 0.0841216, γ → 5.50172}
```

Then check for all elements of dataQ (i = 1,2, …, n) whether it can satisfy the threshold. For example let

```
i = 1131;
dataQ[[i]]
{0.320363, 4.46075, 5.90436}
```

```
If[(((1/√(1 + α^2 + β^2)) Abs[dataQ[[i]][[3]] - α dataQ[[i]][[1]] -
 β dataQ[[i]][[2]] - γ])/.sols) < τ,
 inliers[[j]] = Append[inliers[[j]], dataQ[[i]]]]
{{0.320363, 4.46075, 5.90436}}
```

```
inliers
{{}, {{0.320363, 4.46075, 5.90436}}, {}, {}, {}, {}, {}, {}, {},
 {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {},
 {}, {}, {}, {}, {}, {}, {}}
```

Employing the following double For cycle

```
For j = 1, 2, ... N
 For i = 1, 2, ... NC
```

we can get the maximal consensus whose size is between s and NC. Here NC is the number of observations, see Sect. 13.1.3.2. Then the computation is straightforward.

13.1.5.6 Application the RANSAC Method

Now let us use for threshold τ = 0.75

```
τ = 0.75;
```

and let

```
s = 3;
```

therefore

```
N = Round[Log[1 - p]/Log[1 - (1 - v)^s]]
34
```

Preparation of the sets of inliers for every iteration step—at the beginning they are empty sets:

```
inliers = Table[{},{N}](*the consensus sets, j = 1,2,...N*);
```

Then let us carry out the double `For` cycle to find the maximum consensus set

```
AbsoluteTiming[For[j = 1, j ≤ N, j + + ,
 (*get four random points*)
 dataS = RandomSample[dataQ, s];
 (*determine model parameters*)
 sols = PlaneTLS[dataS];
 (*determine inliers*)
 For[i = 1, i ≤ NC, i + + ,
  If[(((1/√(1 + α^2 + β^2))Abs[dataQ[[i]][[3]] - αdataQ[[i]][[1]] -
   βdataQ[[i]][[2]] - γ])/.sols) < τ,
   inliers[[j]] = Append[inliers[[j]], dataQ[[i]]]];
 ];];]
{39.9638, Null}
```

Select the consensus set which has the highest number of inliers (Fig. 13.12)

```
trimmeddata = inliers[[Ordering[inliers, -1]]][[1]];
Length[trimmeddata]
20048

p5 = ListPointPlot3D[trimmeddata, PlotStyle → Green];
Show[{p1, p5}]
```

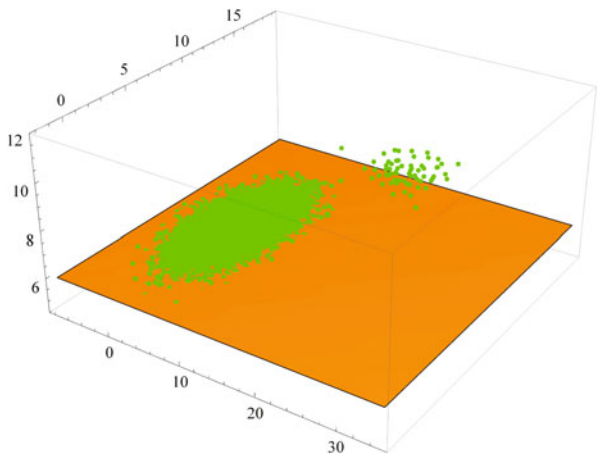


Fig. 13.12 The original plane and the inliers computed via RANSAC (green)

Then we can compute the parameters

```
PlaneTLS[trimmeddata]
{α → 0.0137893, β → 0.00655942, γ → 5.92509}
```

Having a multicore machine, one can evaluate `For` cycles in parallel. Let us compute the internal `For` cycle parallel, then `ParallelTable` should be used used instead of `For`. Now we launch twice as many kernels as the number of cores.

Let us refresh the sets of inliers for every iteration step - at the beginning they are empty sets:

```
inliers = Table[{},{N}](*the consensus sets, j = 1,2,...N*);
AbsoluteTiming[For[j = 1, j ≤ N, j + + ,
 (*get three random points*)
 dataS = RandomSample[dataQ, s];
 (*determine model parameters*)
sols = planeTLS[dataS];
(*determine inliers*)
S = ParalleTable[
If[(((1/√(1 + α^2 + β^2))Abs[dataQ[[i]][[3]] - αdataQ[[i]][[1]] -
  βdataQ[[i]][[2]] - γ])/.sols) < τ, dataQ[[i]]];
 {i, 1, NC}];
 inliers[[j]] = Append[inliers[[j]],
  Select[S, (Head[#] == List)&]][[1]];
];]
{39.9638, Null}
```

Select the consensus set which contains the highest number of inliers (Fig. 13.13)

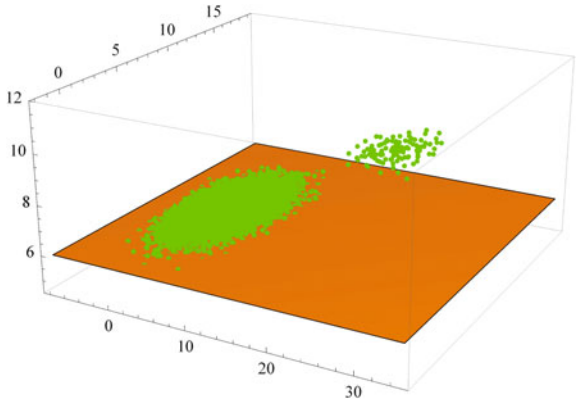


Fig. 13.13 The original plane and the inliers computed via RANSAC

```
trimmeddata = inliers[[Ordering[inliers, -1]]][[1]];
Length[trimmeddata]
20077
p5 = ListPointPlot3D[trimmeddata, PlotStyle → Green];
Show[{p1, p5}]
```

Then we can compute the parameters

```
PlaneTLS[trimmeddata]
{α → 0.0277224, β → .0114123, γ → 5.86191}
```

We can improve the method—increase the precision and reduce the running time—by preparating the data set via a soft computing technique, like neural network algorithm, see in the next section.

13.1.5.7 Application of Self-organizing Map (SOM) to the RANSAC Algorithm

The Self-Organizing Map (SOM) is a neural network algorithm, which uses a competitive learning technique to train itself in an unsupervised manner. SOMs are different from other artificial neural networks in the sense that they use a neighborhood function to preserve the topological properties of the input space and they are used to create an ordered representation of multi-dimensional data that simplifies complexity and reveals meaningful relationships.

In the last decade a lot of research has been carried out to develop surface reconstruction methods. A more recent approach to the problem of surface reconstruction is that of learning based methods see e.g. DalleMole et al. (2010). Learning algorithms are able to process very large or noisy data, such as point clouds obtained from 3D scanners and are used to construct surfaces. Following this approach some studies have been employed SOM and their variants for surface reconstruction. SOM is suitable for this problem because it can form topological maps representing the distribution of input data. In our case this mapping occurs from 3D space to 2D, see Kohonen (1998).

Therefore we try to represent this cloud of observed data points by considerably fewer points using the Kohonen-map with a lattice of SOM of size 6×6 code-book vectors with symmetrical neighborhood (Fig. 13.14).

```
≪NeuralNetworks′
AbsoluteTiming[{som, fitrecord} =
 UnsupervisedNetFit[XYZC, 36, 35, SOM → {6, 6}];//Quiet]
```

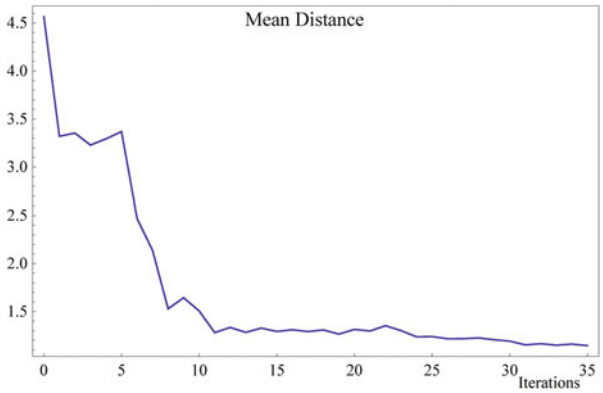


Fig. 13.14 The error of the Kohonen-map with a lattice of SOM of size 6 × 6 code-book vectors during the training of the network

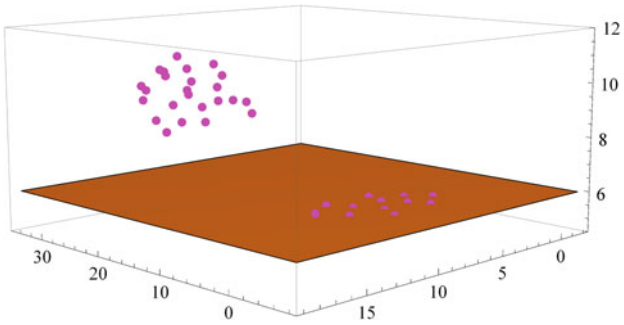


Fig. 13.15 The original plane and the code-book vectors of the SOM of 6 × 6

Figure 13.15 shows the original plane with the 6 × 6 = 36 codebook vector points.

```
dataSOM = som[[1]];
p6 = ListPointPlot3D[dataSOM,
 PlotStyle → Directive[Magenta, PointSize[0.015]]];
Show[{p1, p6}]
```

Now let us apply RANSAC to these 36 code-book vectors with a low threshold

```
τ = 0.05;
NSOM = Length[dataSOM];
inliers = Table[{}, {N}] (* the consensus sets, j = 1, 2, ... N*);
AbsoluteTiming[For[j = 1, j ≤ N, j + + ,
 (*get three random points*)
 dataS = RandomSample[dataSOM, s];
 (*determine model parameters*)
 sols = PlaneTLS[dataS];
 (* determine inliers*)
 For[i = 1, i → NSOM, i + + ,
  If[(((1/√(1 + α^2 + β^2)) Abs[dataQ[[i]][[3]] - αdataQ[[i]][[1]] -
   βdataQ[[i]][[2]] - γ])/.sols) < τ,
   inliers[[j]] = Append[inliers[[j]], dataQ[[i]]]];
 ];];]
{1.46515, Null}
```

Select the consensus set which has the highest number of inliers

```
trimmeddata = inliers[[Ordering[inliers, -1]]][[1]];
Length[trimmeddata]
14
```

Figure 13.16 shows the result. These 14 codevectors are practically on the plane.

```
p7 = ListPointPlot3D[trimmeddata, PlotStyle → Directive[Green,
 PointSize[0.015]]];
Show[{p1, p7}]
```

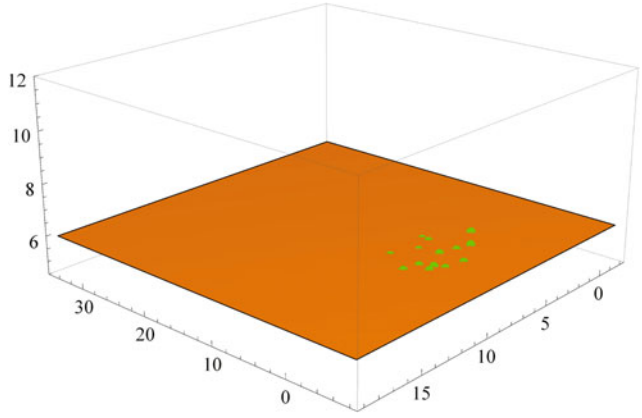


Fig. 13.16 The original plane and the inliers of RANSAC (magenta)

Then we can compute the parameters

```
PlaneTLS[trimmeddata]
{α → -0.0119509, β → -0.00791939, γ → 6.05779}
```

The error of the method is quite acceptable. Now the total computation time of SOM + RANSAC parallel combination is just the same as that of the direct RANSAC with the original number of points. However having a more than four core machine, this is not the case, since parallel processing can reduce the computation time further.

13.2 Application Examples

13.2.1 Fitting a Sphere to Point Cloud Data

In order to understand the geometry of an object as a function of its depth values relative to the sensor, it is vital to have both a smooth and accurate estimate of the depth map. Due to the limitations of the depth sensing technology used by inexpensive devices, a significant loss occurs when capturing depth information. Unlike depth measuring technologies, like laser based LIDAR or range finding cameras, theses sensors contain much missing data and are incapable of measuring depth for shiny objects or objects of certain orientation. On the flip side, traditional depth capturing devices cost in the order of several thousand dollars, while low resolution sensors costs around $100–200. These sensors mainly are intended for close distance human pose recognition, like Microsoft's Kinect, and are operated in an environment with a conspicuous foreground (human) and background (room), so a clean and smooth depth map is not necessary.

However, if the ability of such sensors is to be extended to a more general setting, such as the inclusion in consumer laptops for human-computer interaction, robotics for robot motion planning, or graphics for environment reconstruction, both smooth and complete depth information are essential and could significantly enhance the accuracy of object detection.

In our example the measurement has been carried out with *Microsoft Kinect XBOX* see Fig. 13.17. Microsoft's Kinect contains a diverse set of sensors, most notably a depth camera based on *PrimeSense's* infrared structured light technology. With a proper calibration of its color and depth cameras, the Kinect can capture detailed color point clouds at up to 30 frames per second. This capability uniquely positions the Kinect for use in fields such as robotics, natural user interfaces, and three-dimensional mapping, see i.e. Draelos (2012).

This device provides 2D RGB color imaging and RGB data representing the depth—the distance of the object—in 11 bits (0.2048).

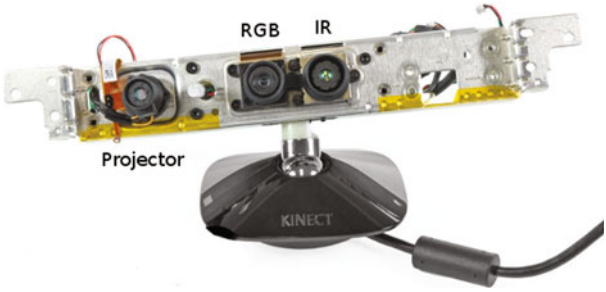


Fig. 13.17 Microsoft Kinect XBOX

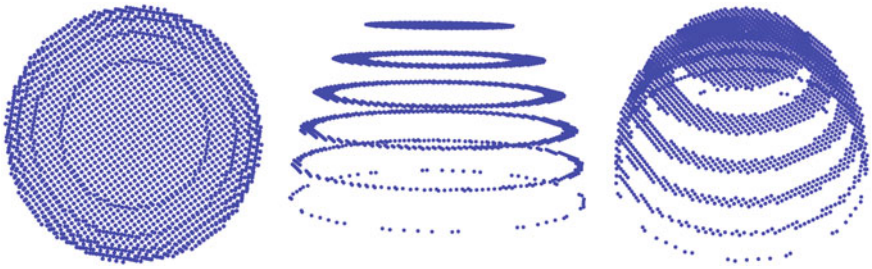


Fig. 13.18 Simulated synthetic quantized depth data points in case of a sphere from 3 different view points

This low resolution causes a discontinuity effect, which can be even 10 cm above a 4 m object distance, and is about 2 cm when the object distance is less than 2.5 m. In our case a spherical object, having radius $R = 0.152$ m was placed in the real world position $x = 0$, $y = 0$ and the object distance $z = 3$ m. Since the intensity threshold resolution is low, one may have quantized levels caused by round-off processes. Figure 13.18 represents the quantized depth data points simulated as synthetic data for illustration.
Real measurements suffer not only from quantization but from random errors. Let us load a real measured data set adopted from Molnár et al. (2012).

```
dataQ = Import["H:\\f_03_05.dat"]/1000;
```

The number of the data points is,

```
n = Length[dataQ]
2710
```

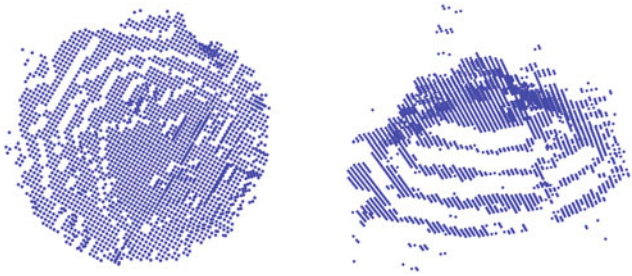


Fig. 13.19 Real measured data points from different view points

The cloud of measured points is shown in Fig. 13.19.
There are three basic approaches for estimating the position (a, b, c) and the radius (R) of a sphere from data point clouds, namely algebraic, geometric and directional least squares estimate.

13.2.1.1 Algebraic Least Square When R Is Unknown

In this case, we want to fit given n points to a sphere such that the sum of the squared algebraic distance is minimized then the local error is defined as

$$\delta_i = (x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 - R^2$$

where (x_i, y_i, z_i) are the coordinates of the measured point. First let us suppose that the position as well as the radius of the sphere is unknown.

The parameters to be estimated are R, a, b and c. Let us introduce a new parameter,

$$d = a^2 + b^2 + c^2 - R^2.$$

Then d_i can be expressed as

$$\delta_i = (-2x_i, -2y_i, -2z_i, 1) \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \left(x_i^2 + y_i^2 + z_i^2\right).$$

Consequently the parameter estimation problem is linear, namely one should solve an overdetermined system. In matrix form

$$Mp = h$$

where

$$M = \begin{pmatrix} -2x_1 & -2y_1 & -2z_1 & 1 \\ . & . & . & . \\ -2x_i & -2y_i & -2z_i & 1 \\ . & . & . & . \\ -2x_n & -2y_n & -2z_n & 1 \end{pmatrix}, \quad p = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \quad and \quad h = \begin{pmatrix} -\left(x_1^2 + y_1^2 + z_1^2\right) \\ . \\ -\left(x_i^2 + y_i^2 + z_i^2\right) \\ . \\ -\left(x_n^2 + y_n^2 + z_n^2\right) \end{pmatrix}.$$

Then the solution in least square sense

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = M^{-1}h$$

where M^{-1} is the pseudoinverse of M.

The attractive feature of the algebraic parameter estimation is that its formulation results in a linear least squares problem which has non-iterative closed-form / HypSlash> solution. Therefore the algebraic distance based fitting can be used as a quick way to calculate approximate parameter values without requiring an initial guess.

First we create the M matrix.

```
M = Map[{-2#[[1]], -2#[[2]], -2#[[3]], 1}&, dataQ];
Dimensions[M]
{2710, 4}
```

The right hand side vector is

```
h = Map[-(#[[1]]^2 + #[[2]]^2 + #[[3]]^2)&, dataQ];
```

Then we can compute the parameters *a, b, c* and *d,*

```
{a, b, c, d} = PseudoInverse[M].h
{-0.0413512, -0.00851136, 3.00166, 8.99215}
```

and the estimated radius of the sphere is

```
R = √(a^2 + b^2 + c^2 - d)
0.139948
```

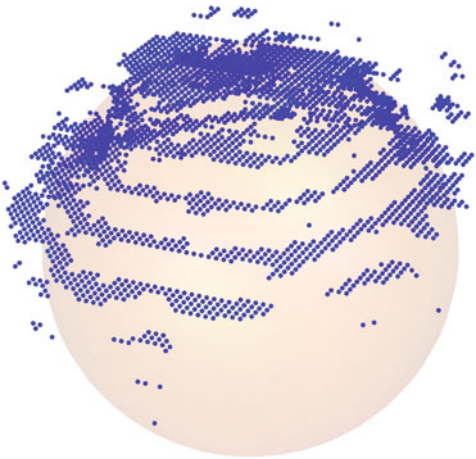


Fig. 13.20 The measured data with the estimated object

Since we want to use these parameter values as initial values for other methods, we assign them to the following variables,

```
a0 = a; b0 = b; c0 = c; d0 = d; R0 = R;
```

Let us display this estimated sphere with the data points, see Fig. 13.20.
The disadvantages of this method are its sensitivity for noise, outliers and partial occlusion, namely when the data points cover only a part of the surface of the sphere.

13.2.1.2 Algebraic Least Square When *R* Is Known

Sometimes the radius of sphere is known i.e. in case of calibration, and we want to find only the position of the sphere. Therefore there are 3 parameters *a, b* and *c* to be estimated. Now we have an overdetermined polynomial system to be solved,

$$(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 - R^2 = 0 \quad i = 1, 2, \ldots n.$$

The equations can be easily generated:

```
Clear[a,b,c]; R = 0.152;
```

the prototype of the equation,

```
G = (x - a)^2 + (y - b)^2 + (z - c)^2 - R^2;
```

Then the system can be created,

```
eqs = Map[G/.{x → #[[1]], y → #[[2]], z → #[[3]]}&, dataQ];
```

For example the first equation is,

```
eqs[[1]]
-0.023104 + (-0.1958 - a)^2 + (-0.0309 - b)^2 + (2.967 - c)^2
```

The number of the equations is,

```
Length[eqs]
2710
```

The solution of this problem is not unique since the reduced Gröbner basis for the determined polynomial subsystems ($n = 3$) are second order polynomials, see Appendix A1 at the end of this section. To solve the overdetermined system one can use local or global techniques. The local methods need an initial guess for the parameter values, which can be the solution of the linear problem. One of the most effective local methods is the *Extended Newton method.* Extended Newton method can handle nonlinear overdetermined systems by using the pseudoinverse of the Jacobian.

Let us employ the result of the algebraic solution as initial guess

```
NewtonExtended[eqs, {a, b, c}, {a0, b0, c0}]
{-0.0403983, -0.00727025, 3.0184}
```

Another local method is based on the *direct minimization of the squares of the residuals of the equations*,

$$\rho = \sum_{i=1}^{n} \delta_i^2 = \sum_{i=1}^{n} \Big((x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2 - R \Big)^2 .$$

Now we employ *local minimization* with the initial guess again computed for the linear problem. The objective function is

```
W = Apply[Plus, Map[#^2&, eqs]];
```

Then

```
FindMinimum[W, {{a, a0}, {b, b0}, {c, c0}}]
{0.0355584, {a → -0.0403983, b → -0.00727025 → 3.0184}}
```

The solution of the two methods is the same, but the computation time of the local minimization is somewhat shorter.

Although as we mentioned, the reduced Gröbner basis of the determined system ($n = 3$) can be computed in symbolic form represented by second order polynomials, the high number of the subsystems, namely

```
Binomial[n,3]
3313414020
```

hinders the realization of the Gauss-Jacobi combinatorial computation as a global method. However, later we shall demonstrate, that if one can represent the cloud of points with considerably fewer points, for example via SOM (Self Organizing Map), then the symbolic solution is a reasonable alternative to the local methods.

Appendix A1 *Algebraic approach: Computation of the Gröbber basis, when R is known*

If we have three exact noiseless measurement points, then the prototype system for 3 points:

```
Clear[a,b,c,R];
```

The prototype of the equations,

```
G=(x-a)^2 + (y-b)^2 + (z-c)^2 - R^2;
proto=Table[(G/.{x → η_i, y → ξ_i, z → χ_i}),{i,1,3}]
{-R^2 + (-a + η_1)^2 + (-b + ξ_1)^2 + (-c + χ_1)^2,
 -R^2 + (-a + η_2)^2 + (-b + ξ_2)^2 + (-c + χ_2)^2,
 -R^2 + (-a + η_3)^2 + (-b + ξ_3)^2 + (-c + χ_3)^2}
```

The polynomial for a is

```
grba=GroebnerBasis[proto,{a,b,c},{b,c}]//Simplify;
Coefficient[grba,a,2]//Simplify
{4(η_1^2ξ_2^2 - 2η_1^2ξ_2ξ_3 + η_1^2ξ_3^2 + ξ_2^2χ_1^2 -
 2ξ_2ξ_3χ_1^2 + ξ_3^2χ_1^2 + η_3^2(ξ_1^2 - 2ξ_1ξ_2 + ξ_2^2 + (χ_1 - χ_2)^2) - 2ξ_1ξ_2χ_1χ_2 +
 2ξ_1ξ_3χ_1χ_2 + 2ξ_2ξ_3χ_1χ_2 - 2ξ_3^2χ_1χ_2 + η_1^2χ_2^2 + ξ_1^2χ_2^2 - 2ξ_1ξ_3χ_2^2 +
 ξ_3^2χ_2^2 + η_2^2(ξ_1^2 - 2ξ_1ξ_3 + ξ_3^2 + (χ_1 - χ_3)^2) +
 2η_1η_3(-ξ_2^2 + ξ_1(ξ_2 - ξ_3) + ξ_2ξ_3 + (χ_1 - χ_2)(χ_2 - χ_3)) +
 2ξ_1ξ_2χ_1χ_3 - 2ξ_2^2χ_1χ_3 - 2ξ_1ξ_3χ_1χ_3 + 2ξ_2ξ_3χ_1χ_3 -
 2η_1^2χ_2χ_3 - 2ξ_1^2χ_2χ_3 + 2ξ_1ξ_2χ_2χ_3 + 2ξ_1ξ_3χ_2χ_3 -
 2ξ_2ξ_3χ_2χ_3 + η_1^2χ_3^2 + ξ_1^2χ_3^2 - 2ξ_1ξ_2χ_3^2 + ξ_2^2χ_3^2 - 2η_2(η_3(ξ_1^2 +
 ξ_2ξ_3 - ξ_1(ξ_2 + ξ_3) + (χ_1 - χ_2)(χ_1 - χ_3)) + η_1(ξ_1(ξ_2 - ξ_3) -
 ξ_2ξ_3 + ξ_3^2 + χ_1χ_2 - χ_1χ_3 - χ_2χ_3 + χ_3^2)))}
Coefficient[grba,a,3]//Simplify
{0}
```

This means we have a second order polynomial indicating non-unique solution.

13.2.1.3 Geometric Fitting When *R* Is Unknown

In this case, we want to minimize the sum of squared geometric distances to the point set. Then the local error is defined as

$$\delta_i = \sqrt{(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2} - R.$$

Now we have an overdetermined *nonlinear equation system* to be solved for the parameter estimation problem even in case of unknown *R*. However in this case, the solution of the system is unique, since the Gröbner basis of the determined subsystems are first order polynomials. In general the use of geometric distance results in better solution compared to the algebraic distance. This is especially true in the presence of noise and small outliers. Similarly to the algebraic approach, local methods like Extended Newton method as well as direct local minimization can be employed.

Let us create the system of the equations. First we reset *a, b, c* and *R* to be symbolic variables again,

```
Clear[a,b,c,R]
```

The prototype of the equation,

```
G = √((x - a)^2 + (y - b)^2 + (z - c)^2) - R;
```

Then the system can be created as,

```
eqs = Map[G/.{x → #[[1]],y → #[[2]],z → #[[3]]}&,dataQ];
```

For example the first equation is

```
eqs[[1]]
√((-.1958 - a)^2 + (-.0309 - b)^2 + (2.967 - c)^2) - R
```

Let us employ the Extended Newton method with the result of the algebraic solution as initial guess. We get:

```
{-0.0411326,-0.00844824,3.01198,0.145917}
```

Indeed, the geometric approach provides better estimation than algebraic one, however it needs a good initial guess.

The other possible solution is minimization of the squares of the residuals of the equations,

$$\rho = \sum_{i=1}^{n} \delta_i^2 = \sum_{i=1}^{n} \left(\sqrt{(x_i - a)^2 + (y_i - b)^2 + (z_i - c)^2} - R \right)^2 .$$

Now again we can employ *local minimization* with the initial guess computed from the linear problem,

```
W = Apply[Plus, Map[#^2 &, eqs]];
FindMinimum[W, {{a, a0}, {b, b0}, {c, c0}, {R, R0}}]
{0.351, {a → -0.0411326, b → -0.00844824, c → 3.01198, R → 0.145917}}
```

Again this computation required less time than the Newton method.

13.2.1.4 Geometric Fitting When R Is Known

Again one can use the Extended Newton method,

```
NewtonExtended[eqs/.R → 0.152, {a, b, c}, {a0, b0, c0}]
{-0.040658, -0.00793428, 3.02032}
```

The solution with the direct local minimization

```
FindMinimum[W/.R → 0.152, {{a, a0}, {b, b0}, {c, c0}}]
{0.360787, {a → -0.040658, b → -0.00793428, c → 3.02032}}
```

In this case the nonlinear system has more solutions since the Gröbner basis of the determined subsystems are second order polynomials, see Appendix A2. Generally speaking, the algebraic as well as the geometric approach with known radius always lead to nonlinear problems, where in case of geometric model the solution is not unique.

Appendix A2 *Geometric approach: Computation of the Gröbber basis, when R is known*

If we have three exact noiseless measurement points, then the prototype system for 3 points:

```
Clear[a, b, c, R];
```

The prototype of the equations,

```
G = Sqrt[(x - a)^2 + (y - b)^2 + (z - c)^2] - R;
proto = Table[(G /. {x → η_i, y → ξ_i, z → χ_i}), {i, 1, 3}]
{-R^2 + Sqrt[(-a + η_1)^2 + (-b + ξ_1)^2 + (-c + χ_1)^2],
 -R^2 + Sqrt[(-a + η_2)^2 + (-b + ξ_2)^2 + (-c + χ_2)^2],
 -R^2 + Sqrt[(-a + η_3)^2 + (-b + ξ_3)^2 + (-c + χ_3)^2]}
```

The polynomial for a is

```
grba = GroebnerBasis[proto, {a, b, c}, {b, c}] // Simplify;
Length[grba]
1
ca2 = Coefficient[grba[[1]], a, 2] // Simplify
{4 (η_1^2 ξ_2^2 - 2 η_1^2 ξ_2 ξ_3 + η_1^2 ξ_3^2 + ξ_2^2 χ_1^2 - 2 ξ_2 ξ_3 χ_1^2 +
  ξ_3^2 χ_1^2 + η_3^2 (ξ_1^2 - 2 ξ_1 ξ_2 + ξ_2^2 + (χ_1 - χ_2)^2) - 2 ξ_1 ξ_2 χ_1 χ_2 +
  2 ξ_1 ξ_3 χ_1 χ_2 + 2 ξ_2 ξ_3 χ_1 χ_2 - 2 ξ_3^2 χ_1 χ_2 + η_1^2 χ_2^2 + ξ_1^2 χ_2^2 -
  2 ξ_1 ξ_3 χ_2^2 + ξ_3^2 χ_2^2 + η_2^2 (ξ_1^2 - 2 ξ_1 ξ_3 + ξ_3^2 + (χ_1 - χ_3)^2) +
  2 η_1 η_3 (-ξ_2^2 + ξ_1 (ξ_2 - ξ_3) + ξ_2 ξ_3 + (χ_1 - χ_2) (χ_2 - χ_3)) +
  2 ξ_1 ξ_2 χ_1 χ_3 - 2 ξ_2^2 χ_1 χ_3 - 2 ξ_1 ξ_3 χ_1 χ_3 + 2 ξ_2 ξ_3 χ_1 χ_3 -
  2 η_1^2 χ_2 χ_3 - 2 ξ_1^2 χ_2 χ_3 + 2 ξ_1 ξ_2 χ_2 χ_3 + 2 ξ_1 ξ_3 χ_2 χ_3 -
  2 ξ_2 ξ_3 χ_2 χ_3 + η_1^2 χ_3^2 + ξ_1^2 χ_3^2 - 2 ξ_1 ξ_2 χ_3^2 + ξ_2^2 χ_3^2 - 2 η_2 (η_3 (ξ_1^2 +
  ξ_2 ξ_3 - ξ_1 (ξ_2 + ξ_3) + (χ_1 - χ_2) (χ_1 - χ_3)) +
  η_1 (ξ_1 (ξ_2 - ξ_3) - ξ_2 ξ_3 + ξ_3^2 + χ_1 χ_2 - χ_1 χ_3 - χ_2 χ_3 + χ_3^2)))}
ca3 = Coefficient[grba[[1]], a, 3] // Simplify
0
```

Again, we have a second order polynomial indicating non-unique solution.

13.2.1.5 Directional Fitting

For directional fitting, the error function is based on the distance between the measured point $P_i = P\{x_i, y_i, z_i\}$ and its projection onto the surface of the sphere closer to the instrument along the direction of P_i, see Fig. 13.21.
In Cartesian coordinates, the expressions for r_i, p_i and q_i (see Fig. 13.21) can be written as, see i.e. Franaszek et al. (2009) (Fig. 13.22).

$$r_i = \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2 + (z_c - z_i)^2},$$

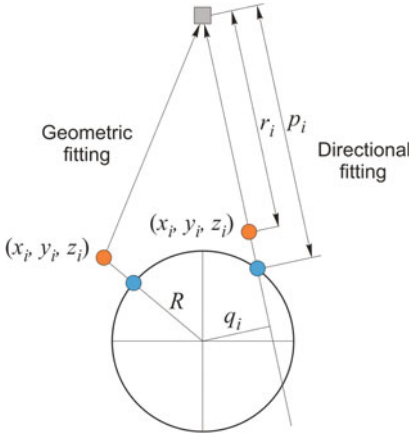


Fig. 13.21 Comparison of the geometric and directional fitting

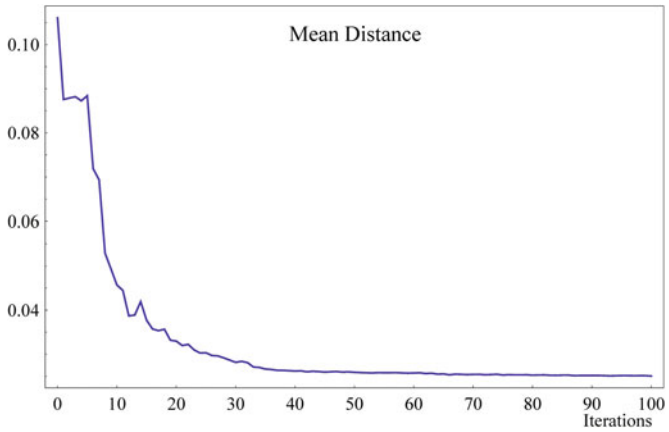


Fig. 13.22 The error of the Kohonen-map with a lattice of SOM of size 5 × 5 code-book vectors

where x_c, y_c and z_c are the coordinates of the center point. Let,

$$p_i = au_i + bv_i + cw_i$$

and

$$q_i = \sqrt{(v_i c - w_i b)^2 + (w_i a - u_i c)^2 + (u_i b - v_i a)^2},$$

where

$$u_i = \frac{x_i}{r_i}, \quad v_i = \frac{y_i}{r_i} \quad and \quad w_i = \frac{z_i}{r_i}.$$

The systems of the nonlinear equations are

$$\left(p_i - \sqrt{R^2 - q_i^2} - r_i\right)^2 = 0 \quad if \quad q_i < R$$
$$(p_i - r_i)^2 + (q_i - R)^2 = 0 \quad if \quad q_i \geq R.$$

In our case the camera is in the origin,

$$x_c = 0, \quad y_c = 0 \quad and \quad z_c = 0.$$

Now, let us create the nonlinear system of the problem,

```
Eqs = Res[a,b,c,R,dataQ];
```

The number of equations are

```
Length[Eqs]
2710
```

For example the first equation is,

```
Eqs[[1]]
If[Sqrt[((0.0103914a - 0.0658458b)^2
 + (-0.997776b - 0.0103914c)^2 + (0.997776a + 0.0658458c)^2)]
<R,((-2.97361 - 0.0658458a - 0.0103914b
 + 0.997776c - Sqrt[(-(0.0103914a - 0.0658458b)^2
 -(-0.997776b - 0.0103914c)^2 - (0.997776a + 0.0658458c)^2
 + R^2)])^2),(-2.97361 - 0.0658458a - 0.0103914b +
0.997776c)^2 + [Sqrt[((0.0103914a - 0.0658458b)^2 +
(-0.997776b - 0.0103914c)^2 + (0.997776a + 0.0658458c)^2)] - R)^2]
```

The Extended Newton method can not handle these equations properly because the derivatives can not be compute symbolically, therefore local minimization is used.

The objective function is the sum of the residual of the equations,

```
W = Apply[Plus,Eqs];
```

Then the local minimization is employed with a derivative free method,

```
FindMinimum[W,{{a,a0},{b,b0},{c,c0},{R,R0}},
 Method → "PrincipalAxis"]
{1.56588,{a → -0.036743,b → -0.00738915,c → 3.00028,
 R → 0.140142}}
```

Now let us assume that R is known. Then we have the same equation system where $R = 0.152$.

Then the local minimization gives

```
FindMinimum[W/.R → 0.152,{{a,a0},{b,b0},{c,c0}},
 Method → "PrincipalAxis"]
{1.38047,{a → -0.0369308,b → -0.00800386,c → 3.0196}}
```

In the following sections we shall demonstrate the application of Self-Organizing Map (SOM) which can reduce the number of data points and open the way for the application of the Gauss-Jacobi combinatoric solution.

13.2.1.6 Application Self-organizing Map to Reducing Data

Using a Kohonen-map with a lattice of SOM (Self Organizing Map) of size 5×5 code-book vectors with symmetrical neighborhood.
The 25 representing code-book vectors are

```
dataSOM = som[[1]];
```

Let us display the code-book vectors with the sphere to be estimated, see Fig. 13.23.

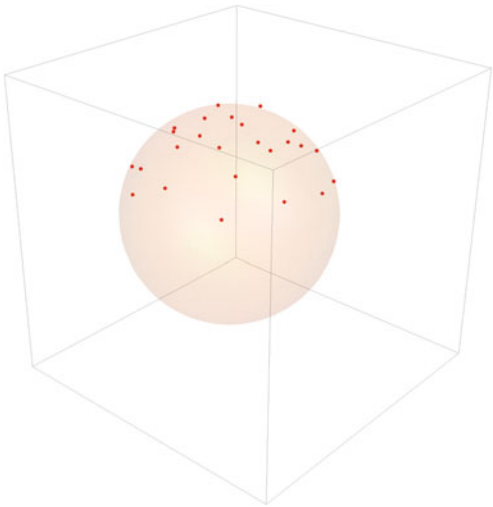


Fig. 13.23 The code-book vectors with the exact target object

At this time the application of the Gauss-Jacobi algorithm as a global method to solve polynomial systems is reasonable, since now the number of the subsets is "only"

```
Binomial[25,4]
12650
```

when R is unknown, and

```
Binomial[25,3]
2300
```

when R is known.

But before we do that, let us compute the parameters from these code-book vectors as input data in order to see if precise results can be expected at all with a reduced data set. We use the earlier methods. First we consider geometric fitting with unknown and then with known radius.

The parameters when R is unknown:

```
Clear[a,b,c,R]
```

Then the system can be created

```
eqs = Map[G/.{x → #[[1]],y → #[[2]],z → #[[3]]}&,dataSOM];
```

Let us employ the Extended Newton method with the result of the algebraic solution as initial guess

```
NewtonExtended[eqs,{a,b,c,R},{a0,b0,c0,R0}]
{-0.040915,-0.00897106,3.00846,0.141147}
```

Then for known radius, we get

```
NewtonExtended[eqs/.R → 0.152,{a,b,c},{a0,b0,c0}]
{-0.0397396,-0.00766308,3.02364}
```

It means that the code-book vectors provide a quite good representation of the point cloud.

Let us compute the parameters from the directional approach, too. Creating the nonlinear system of the problem for R is unknown,

```
Eqs = Res[a,b,c,R,dataSOM];
```

The number of equations is

```
Length[Eqs]
25
```

For example the first equation is,

```
Eqs[[1]]
If[Sqrt[((0.0333223a - 0.0476156b)^2 + (-0.99831b - 0.0333223c)^2 +
 (0.99831a + 0.0476156c)^2)] < R, (-2.97169 - 0.0476156a - 0.0333223b +
 0.99831c - Sqrt[(-(0.0333223a - 0.0476156b)^2 -
 (-0.99831b - 0.0333223c)^2 - (0.99831a + 0.0476156c)^2 + R^2)])^2,
 ((-2.97169 - 0.0476156a - 0.0333223b + 0.99831c)^2) +
 Sqrt[((0.0333223a - 0.0476156b)^2 + (-0.99831b - 0.0333223c)^2 +
 (0.99831a + 0.0476156c)^2)] - R)^2)]
```

Again let us employ local minimization

```
W = Apply[Plus, Eqs];
FindMinimum[W, {{a, a0}, {b, b0}, {c, c0}, {R, R0}},
 Method → "PrincipalAxis"]
{0.00139774, {a → -0.0398059, b → -0.0102369, c → 3.0147,
 R → 0.144784}}
```

However, now SOM provides a smooth and well arranged data point set, therefore Newton method can be also successful,

```
NewtonExtended[Eqs, {a, b, c, R}, {a0, b0, c0, R0}]
{-0.0398873, -0.00998192, 3.01548, 0.145522}
```

Now let us compute the Gröbner basis for the equation system resulted from the geometric approach, when R is unknown.

13.2.1.7 Symbolic Computation of Gröbner Basis for Geometric Fitting

Considering 4 exact noiseless measurement points, then the prototype system for these 4 points is,

```
proto = Table[(G/.{x → η_i, y → ξ_i, z → χ_i}), {i, 1, 4}]
{-R + Sqrt[(-a + η1)^2 + (-b + ξ1)^2 + (-c + χ1)^2],
 -R + Sqrt[(-a + η2)^2 + (-b + ξ2)^2 + (-c + χ2)^2],
 -R + Sqrt[(-a + η3)^2 + (-b + ξ3)^2 + (-c + χ3)^2],
 -R + Sqrt[(-a + η4)^2 + (-b + ξ4)^2 + (-c + χ4)^2]}
```

One can realize, that the elimination of R reduces the system

```
protoR = Take[proto, {2, 4}] /
.R → √((-a + η1)^2 + (-b + ξ1)^2 + (-c + χ1)^2)
{-√((-a + η1)^2 + (-b + ξ1)^2 + (-c + χ1)^2)
 + √((-a + η2)^2 + (-b + ξ2)^2 + (-c + χ2)^2),
 -√((-a + η1)^2 + (-b + ξ1)^2 + (-c + χ1)^2)
 + √((-a + η3)^2 + (-b + ξ3)^2 + (-c + χ3)^2),
 -√((-a + η1)^2 + (-b + ξ1)^2 + (-c + χ1)^2)
 + √((-a + η4)^2 + (-b + ξ4)^2 + (-c + χ4)^2)}
```

To get univariate polynomial for parameter a let us eliminate b and c from the basis

```
grba = GroebnerBasis[protoR, {a, b, c}, {b, c}] // Simplify;
```

This basis contains

```
Length[grba]
1
```

polynomials. We take the first one

```
grba[[1]]
2aη2ξ3χ1 - η2^2ξ3χ1 - ξ2^2ξ3χ1 + ξ2ξ3^2χ1 -
 2aη2ξ4χ1 + η2^2ξ4χ1 + ξ2^2ξ4χ1 - ξ3^2ξ4χ1 - ξ2ξ4^2χ1 + ξ3ξ4^2χ1 - 2aη1ξ3χ2 +
 η1^2ξ3χ2 + ξ1^2ξ3χ2 - ξ1ξ3^2χ2 +
 2aη1ξ4χ2 - η1^2ξ4χ2 - ξ1^2ξ4χ2 + ξ3^2ξ4χ2 + ξ1ξ4^2χ2 - ξ3ξ4^2χ2 + ξ3χ1^2χ2 - ξ4χ1^2χ2 -
 ξ3χ1χ2^2 + ξ4χ1χ2^2 - 2aη2ξ1χ3 + η2^2ξ1χ3 +
 2aη1ξ2χ3 - η1^2ξ2χ3 - ξ1^2ξ2χ3 + ξ1ξ2^2χ3 - 2aη1ξ4χ3 + η1^2ξ4χ3 +
 2aη2ξ4χ3 - η2^2ξ4χ3 + ξ1^2ξ4χ3 - ξ2^2ξ4χ3 -
 ξ1ξ4^2χ3 + ξ2ξ4^2χ3 - ξ2χ1^2χ3 + ξ4χ1^2χ3 + ξ1χ2^2χ3 - ξ4χ2^2χ3 + ξ2χ1χ3^2 -
ξ4χ1χ3^2 - ξ1χ2χ3^2 + ξ4χ2χ3^2 +
 η4^2(ξ3(χ1 - χ2) + ξ1(χ2 - χ3) + ξ2(-χ1 + χ3)) + 2aη4(ξ3(-χ1 + χ2) +
ξ2(χ1 - χ3) + ξ1(-χ2 + χ3)) + 2aη2ξ1χ4 - η2^2ξ1χ4 -
 2aη1ξ2χ4 + η1^2ξ2χ4 + ξ1^2ξ2χ4 - ξ1ξ2^2χ4 + 2aη1ξ3χ4 -
η1^2ξ3χ4 - 2aη2ξ3χ4 + η2^2ξ3χ4 - ξ1^2ξ3χ4 + ξ2^2ξ3χ4 +
 ξ1ξ3^2χ4 - ξ2ξ3^2χ4 + ξ2χ1^2χ4 - ξ3χ1^2χ4 - ξ1χ2^2χ4 + ξ3χ2^2χ4 +
  ξ1χ3^2χ4 - ξ2χ3^2χ4 - ξ2χ1χ4^2 + ξ3χ1χ4^2 + ξ1χ2χ4^2 - ξ3χ2χ4^2 - ξ1χ3χ4^2 + ξ2χ3χ4^2 -
 2aη3(ξ4(-χ1 + χ2) + ξ2(χ1 - χ4) + ξ1(-χ2 + χ4)) +
 η3^2(ξ4(-χ1 + χ2) + ξ2(χ1 - χ4) + ξ1(-χ2 + χ4))
```

It turns out that this is a first order polynomial, since

```
ca2 = Coefficient[grba[[1]],a,2]
0
```

Therefore the solution for parameter a is simple,

```
aG = - Coefficient[grba[[1]],a,0] / Coefficient[grba[[1]],a,1] //Simplify
```

$$
\begin{aligned}
&(-\eta_2^2\xi_3\chi_1 - \xi_2^2\xi_3\chi_1 + \xi_2\xi_3^2\chi_1 + \eta_2^2\xi_4\chi_1 + \xi_2^2\xi_4\chi_1 - \xi_3^2\xi_4\chi_1 - \\
&\xi_2\xi_4^2\chi_1 + \xi_3\xi_4^2\chi_1 + \eta_1^2\xi_3\chi_2 + \xi_1^2\xi_3\chi_2 - \xi_1\xi_3^2\chi_2 - \eta_1^2\xi_4\chi_2 - \\
&\xi_1^2\xi_4\chi_2 + \xi_3^2\xi_4\chi_2 + \xi_1\xi_4^2\chi_2 - \xi_3\xi_4^2\chi_2 + \xi_3\chi_1^2\chi_2 - \xi_4\chi_1^2\chi_2 - \\
&\xi_3\chi_1\chi_2^2 + \xi_4\chi_1\chi_2^2 + \eta_2^2\xi_1\chi_3 - \eta_1^2\xi_2\chi_3 - \xi_1^2\xi_2\chi_3 + \xi_1\xi_2^2\chi_3 + \\
&\eta_1^2\xi_4\chi_3 - \eta_2^2\xi_4\chi_3 + \xi_1^2\xi_4\chi_3 - \xi_2^2\xi_4\chi_3 - \xi_1\xi_4^2\chi_3 + \xi_2\xi_4^2\chi_3 - \xi_2\chi_1^2\chi_3 + \\
&\xi_4\chi_1^2\chi_3 + \xi_1\chi_2^2\chi_3 - \xi_4\chi_2^2\chi_3 + \xi_2\chi_1\chi_3^2 - \xi_4\chi_1\chi_3^2 - \\
&\xi_1\chi_2\chi_3^2 + \xi_4\chi_2\chi_3^2 + \eta_4^2(\xi_3(\chi_1 - \chi_2) + \xi_1(\chi_2 - \chi_3) + \xi_2(-\chi_1 + \chi_3)) - \eta_2^2\xi_1\chi_4 + \\
&\eta_1^2\xi_2\chi_4 + \xi_1^2\xi_2\chi_4 - \xi_1\xi_2^2\chi_4 - \eta_1^2\xi_3\chi_4 + \\
&\eta_2^2\xi_3\chi_4 - \xi_1^2\xi_3\chi_4 + \xi_2^2\xi_3\chi_4 + \xi_1\xi_3^2\chi_4 - \\
&\xi_2\xi_3^2\chi_4 + \xi_2\chi_1^2\chi_4 - \xi_3\chi_1^2\chi_4 - \xi_1\chi_2^2\chi_4 + \xi_3\chi_2^2\chi_4 + \xi_1\chi_3^2\chi_4 - \xi_2\chi_3^2\chi_4 - \xi_2\chi_1\chi_4^2 + \\
&\xi_3\chi_1\chi_4^2 + \xi_1\chi_2\chi_4^2 - \xi_3\chi_2\chi_4^2 - \xi_1\chi_3\chi_4^2 + \\
&\xi_2\chi_3\chi_4^2 + \eta_3^2(\xi_4(-\chi_1 + \chi_2) + \xi_2(\chi_1 - \chi_4) + \xi_1(-\chi_2 + \chi_4)))/ \\
&(2(-\eta_2\xi_3\chi_1 + \eta_2\xi_4\chi_1 + \eta_1\xi_3\chi_2 - \eta_1\xi_4\chi_2 + \eta_2\xi_1\chi_3 - \eta_1\xi_2\chi_3 + \\
&\eta_1\xi_4\chi_3 - \eta_2\xi_4\chi_3 + \eta_4(\xi_3(\chi_1 - \chi_2) + \xi_1(\chi_2 - \chi_3) + \\
&\xi_2(-\chi_1 + \chi_3)) - \eta_2\xi_1\chi_4 + \eta_1\xi_2\chi_4 - \eta_1\xi_3\chi_4 + \\
&\eta_2\xi_3\chi_4 + \eta_3(\xi_4(-\chi_1 + \chi_2) + \xi_2(\chi_1 - \chi_4) + \xi_1(-\chi_2 + \chi_4))))
\end{aligned}
$$

This means that from the coordinates of the corresponding 4 points the parameter can be directly computed. Similar expressions can be developed for the other two parameters b and c, too.

For parameter b:

```
grbb = GroebnerBasis[protoR,{a,b,c,R},{a,c}];
bG = - Coefficient[grbb[[1]],b,0] / Coefficient[grbb[[1]],b,1] //Simplify
```

$$
\begin{aligned}
&(\eta_4\xi_2^2\chi_1 - \eta_4\xi_3^2\chi_1 - \eta_1^2\eta_4\chi_2 + \eta_1\eta_4^2\chi_2 - \eta_4\xi_1^2\chi_2 - \\
&\eta_1\xi_3^2\chi_2 + \eta_4\xi_3^2\chi_2 + \eta_1\xi_4^2\chi_2 - \eta_4\chi_1^2\chi_2 + \eta_4\chi_1\chi_2^2 + \eta_1^2\eta_4\chi_3 - \eta_1\eta_4^2\chi_3 + \\
&\eta_4\xi_1^2\chi_3 + \eta_1\xi_2^2\chi_3 - \eta_4\xi_2^2\chi_3 - \eta_1\xi_4^2\chi_3 + \eta_4\chi_1^2\chi_3 + \eta_1\chi_2^2\chi_3 - \\
&\eta_4\chi_2^2\chi_3 - \eta_4\chi_1\chi_3^2 - \eta_1\chi_2\chi_3^2 + \eta_4\chi_2\chi_3^2 - \eta_1\xi_2^2\chi_4 + \eta_1\xi_3^2\chi_4 - \\
&\eta_1\chi_2^2\chi_4 + \eta_1\chi_3^2\chi_4 + \eta_1\chi_2\chi_4^2 - \eta_1\chi_3\chi_4^2 + \eta_2(\xi_3^2\chi_1 - \xi_4^2\chi_1 - \eta_1^2\chi_3 - \\
&\xi_1^2\chi_3 + \xi_4^2\chi_3 - \chi_1^2\chi_3 + \chi_1\chi_3^2 + \eta_4^2(-\chi_1 + \chi_3) + \\
&\eta_3^2(\chi_1 - \chi_4) + \eta_1^2\chi_4 + \xi_1^2\chi_4 - \xi_3^2\chi_4 + \chi_1^2\chi_4 - \chi_3^2\chi_4 - \chi_1\chi_4^2 + \\
&\chi_3\chi_4^2) + \eta_2^2(\eta_4(\chi_1 - \chi_3) + \eta_1(\chi_3 - \chi_4) + \eta_3(-\chi_1 + \chi_4)) + \\
&\eta_3(\xi_4^2\chi_1 + \eta_4^2(\chi_1 - \chi_2) + \eta_1^2\chi_2 + \xi_1^2\chi_2 - \xi_4^2\chi_2 + \chi_1^2\chi_2 - \chi_1\chi_2^2 -
\end{aligned}
$$

$$\eta_1^2\chi_4 - \xi_1^2\chi_4 - \chi_1^2\chi_4 + \chi_2^2\chi_4 + \chi_1\chi_4^2 - \chi_2\chi_4^2 + \xi_2^2(-\chi_1 + \chi_4)) + \eta_3^2(\eta_4(-\chi_1 + \chi_2) + \eta_1(-\chi_2 + \chi_4)))/(2(\eta_2\xi_3\chi_1 - \eta_2\xi_4\chi_1 - \eta_1\xi_3\chi_2 + \eta_1\xi_4\chi_2 - \eta_2\xi_1\chi_3 + \eta_1\xi_2\chi_3 - \eta_1\xi_4\chi_3 + \eta_2\xi_4\chi_3 + \eta_4(\xi_3(-\chi_1 + \chi_2) + \xi_2(\chi_1 - \chi_3) + \xi_1(-\chi_2 + \chi_3)) + \eta_2\xi_1\chi_4 - \eta_1\xi_2\chi_4 + \eta_1\xi_3\chi_4 - \eta_2\xi_3\chi_4 + \eta_3(\xi_4(\chi_1 - \chi_2) + \xi_1(\chi_2 - \chi_4) + \xi_2(-\chi_1 + \chi_4))))$$

For parameter c:

```
grbc = GroebnerBasis[protoR,{a,b,c,R},{a,b}];
cG = -Coefficient[grbc[[1]],c,0]/Coefficient[grbc[[1]],c,1] //Simplify
```

$$(\eta_1^2\eta_4\xi_2 - \eta_1\eta_4^2\xi_2 + \eta_4\xi_1^2\xi_2 - \eta_4\xi_1\xi_2^2 - \eta_1^2\eta_4\xi_3 + \eta_1\eta_4^2\xi_3 - \eta_4\xi_1^2\xi_3 - \eta_1\xi_2^2\xi_3 + \eta_4\xi_2^2\xi_3 + \eta_4\xi_1\xi_3^2 + \eta_1\xi_2\xi_3^2 - \eta_4\xi_2\xi_3^2 + \eta_3^2(\eta_4(\xi_1 - \xi_2) + \eta_1(\xi_2 - \xi_4)) + \eta_1\xi_2^2\xi_4 - \eta_1\xi_3^2\xi_4 - \eta_1\xi_2\xi_4^2 + \eta_1\xi_3\xi_4^2 + \eta_2^2(\eta_4(-\xi_1 + \xi_3) + \eta_3(\xi_1 - \xi_4) + \eta_1(-\xi_3 + \xi_4)) + \eta_4\xi_2\chi_1^2 - \eta_4\xi_3\chi_1^2 - \eta_4\xi_1\chi_2^2 - \eta_1\xi_3\chi_2^2 + \eta_4\xi_3\chi_2^2 + \eta_1\xi_4\chi_2^2 + \eta_4\xi_1\chi_3^2 + \eta_1\xi_2\chi_3^2 - \eta_4\xi_2\chi_3^2 - \eta_1\xi_4\chi_3^2 - \eta_1\xi_2\chi_4^2 + \eta_1\xi_3\chi_4^2 + \eta_3(-\xi_1^2\xi_2 + \xi_1\xi_2^2 + \eta_4^2(-\xi_1 + \xi_2) + \xi_1^2\xi_4 - \xi_2^2\xi_4 - \xi_1\xi_4^2 + \xi_2\xi_4^2 + \eta_1^2(-\xi_2 + \xi_4) - \xi_2\chi_1^2 + \xi_4\chi_1^2 + \xi_1\chi_2^2 - \xi_4\chi_2^2 - \xi_1\chi_4^2 + \xi_2\chi_4^2) + \eta_2(\eta_4^2(\xi_1 - \xi_3) + \eta_1^2\xi_3 + \xi_1^2\xi_3 - \xi_1\xi_3^2 - \eta_1^2\xi_4 - \xi_1^2\xi_4 + \xi_3^2\xi_4 + \xi_1\xi_4^2 - \xi_3\xi_4^2 + \eta_3^2(-\xi_1 + \xi_4) + \xi_3\chi_1^2 - \xi_4\chi_1^2 - \xi_1\chi_3^2 + \xi_4\chi_3^2 + \xi_1\chi_4^2 - \xi_3\chi_4^2))/(2(\eta_2\xi_3\chi_1 - \eta_2\xi_4\chi_1 - \eta_1\xi_3\chi_2 + \eta_1\xi_4\chi_2 - \eta_2\xi_1\chi_3 + \eta_1\xi_2\chi_3 - \eta_1\xi_4\chi_3 + \eta_2\xi_4\chi_3 + \eta_4(\xi_3(-\chi_1 + \chi_2) + \xi_2(\chi_1 - \chi_3) + \xi_1(-\chi_2 + \chi_3)) + \eta_2\xi_1\chi_4 - \eta_1\xi_2\chi_4 + \eta_1\xi_3\chi_4 - \eta_2\xi_3\chi_4 + \eta_3(\xi_4(\chi_1 - \chi_2) + \xi_1(\chi_2 - \chi_4) + \xi_2(-\chi_1 + \chi_4))))$$

13.2.1.8 Geometric Fitting via Gauss-Jacobi Method with SOM Data

We can get a good approximation to compute the arithmetic average of the solution of the $\binom{25}{4} = 12\ 650$ subsets. Let us generate the indices of the equations belonging to the same subset,

```
index = Partition[Map[#&,Flatten[Subsets[Range[25],{4}]]],4];
```

For example the 10th subset consists of the following equations

```
index[[10]]
{1,2,3,13}
```

Now we apply parallel computation to reduce computation time. The coordinates of the points of the subsets are

```
dataSub = ParallelMap[Map[dataSOM[[#]]&,#]&,index];
```

For example

```
dataSub = [[10]]
{{-0.141499,-0.0990233,2.96667},{-0.0517075,-0.136479,2.9448},
 {-0.00368423,-0.136832,2.94386},{-0.089566,-0.0862473,2.91522}}
```

In order to avoid ill-posed subsets, we compute the product of the distances of every vertex of a quadrant representing a subset,

```
avgSide[x_]:=Apply[Times,Map[Norm[x[[#[[1]]]]-x[[#[[2]]]]]&,
 {{1,2},{1,3},{1,4},{2,3},{2,4},{3,4}}]]
```

Let us select the only subsets for which this product is higher than 2.5×10^{-5}

```
dataSubS = Parallelize[Select[dataSub,avgSide[#]>.000025&]];
```

Generating a prototype for the equations of the subsets

```
psi = Table[({η_i,ξ_i,χ_i}),{i,1,4}];
```

Substituting numerical values

```
dataS =
 ParallelMap[Flatten[MapThread[MapThread[#1 → #2&,{#1,#2}]&,
 {psi,#}]]&,dataSubS];
```

Then the number of the equations to be considered is

```
Length[dataS]
1231
```

instead of the 12 650.

Employing the result of the Gröbner basis the parameter a is

```
aS = ParallelMap[aG/.#&,dataS];
```

Then the average is

```
aaS = Mean[aS]
 - 0.0382357
```

The parameters b and c can be computed in the same way. b is -0.0108443 and c is 3.01347.

Now employing these averages of computed parameters to compute the radius of the sphere, we get 0.143481.

13.2.1.9 *Application of RANdom SAmple Consensus (RANSAC)*

In order to see how this data reduction technique can work in our case, let us select the simplest model, namely the algebraic least squares estimation when R is known.

The number of data elements required per subset s is

```
s = 4;
```

The solution of a subsystem of size s (which is a determined system) is

$$M = \begin{pmatrix} -2x_1 & -2y_1 & -2z_1 & 1 \\ . & . & . & . \\ -2x_i & -2y_i & -2z_i & 1 \\ . & . & . & . \\ -2x_s & -2y_s & -2z_s & 1 \end{pmatrix}, \quad p = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \quad \textit{and}$$

$$h = \begin{pmatrix} -\left(x_1^2 + y_1^2 + z_1^2\right) \\ . \\ -\left(x_i^2 + y_i^2 + z_i^2\right) \\ . \\ -\left(x_s^2 + y_s^2 + z_s^2\right) \end{pmatrix}$$

and

$$M\,p = h$$

Remark Although it is a determined system, subsets representing ill-posed configuration can arise. Therefore the best way to avoid this situation is to restrict the condition number of M. Let's say if this condition number is higher than 1000 we leave out the computation of the subset and draw another one. Here we use another less efficient method, namely we compute pseudoinverse.

The total number of the data is,

```
n
2710
```

Let

```
p = 0.99;
```

In addition let us assume that the probability of observing an outlier is

```
v = 1/2;
```

Then the number of the necessary iterations is

```
N = Round[Log[1 - p]/Log[1 - (1 - v)^s]]
71
```

Remark The higher this probability, the large the number of necessary iterations.

Thinking of Gauss-Jacobi technique, which can also be employed to kick out outliers, the possible subset of which can be drawn from `dataQ`:

```
Binomial[n,s]
2242352938035
```

Let the threshold t be 1.

```
τ = 1;
```

An element $d_i \in$ `dataQ` will be accepted if

$$\left|(x_i - a_s)^2 + (y_i - b_s)^2 + (z_i - c_s)^2 - R_s^2\right| < \tau^2$$

Remark d_i is represented by the triplet (x_i, y_i, z_i) and the parameters (a_s, b_s, c_s, R_s) computed from a subset of s elements. We get four random points:

```
{{0.0201, - 0.0553, 2.893}, { - 0.0766, - 0.1431, 2.942},
 {0.0153, 0.092, 2.942}, { - 0.0703, 0.0804, 2.893}}
```

First we compute the parameters from this subset. Let us create the M matrix.

$$\begin{pmatrix} -0.0402 & 0.1106 & -5.786 & 1 \\ 0.1532 & 0.2862 & -5.884 & 1 \\ -0.0306 & -0.184 & -5.884 & 1 \\ 0.1406 & -0.1608 & -5.786 & 1 \end{pmatrix}$$

Then we can compute the parameters *a, b, c* and *d,* by multiplying the pseudoinverse of *M* by *h:*

```
{-0.0631953,-0.0128281,3.0033,9.00307}
```

and the estimated radius of the sphere is

```
R = √(a^2 + b^2 + c^2 - d)
0.144597
```

Then check each element of `dataQ` (i = 1, 2,…, n) to see whether it can satisfy the threshold. We get one inlier:

```
{-0.1958,-0.0412,2.967}
```

Now let us use for threshold t = 1.5 cm

```
τ = 0.015;
173
```

As a last step we apply algebraic least squares with the maximal consensus set to checkout previous results. The estimated radius of the sphere is

```
0.1414112
```

The selected points—the elements of the maximal consensus set−are close to the surface of the sphere, see Fig. 13.24.
This seems to be a better result than the direct algebraic approach without RANSAC, and quite close to the result of the directional approach.

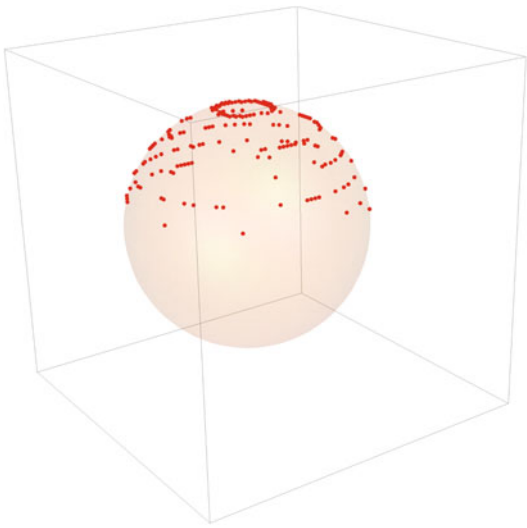


Fig. 13.24 Elements of the maximal consensus set

Table 13.1 Estimated parameters employing different methods (R is unknown)

Type of estimation	a [mm]	b [mm]	c [mm]	R [mm]
Algebraic	−41.4	−8.5	300.2	140.0
Geometric	−41.1	−8.5	301.2	145.9
Directional	−36.7	−7.4	300.0	140.1
Geometric based on SOM	−41.0	−8.9	300.6	141.0
Directional based on SOM	−40.3	−10.1	301.5	144.8
Gauss-Jacobi with Gröbner based on SOM	−38.2	−11.0	301.3	144.0
RANSAC with Algebraic	−34.0	−5.7	300.8	141.0

Table 13.2 Estimated parameters employing different methods (R = 152 mm)

Type of estimation	a [mm]	b [mm]	c [mm]
Algebraic	−40.4	−7.3	301.8
Geometric	−40.7	−7.9	302.0
Directional	−36.7	−7.9	300.0
Geometric based on SOM	−39.7	−7.7	302.4

To find the sphere parameters, different methods—algebraic, geometrical and directional—have been employed. The algebraic approach results in a linear least square problem which does not require initial guess. The geometric as well as the directional approach lead to a nonlinear problem that requires iteration with an initial guess. This initial guess can be the result of the algebraic method. One also may employ direct global minimization of the sum of the residual of the geometrical model equations; however, it may need restrictions for the search space.

As an alternative method, Gauss-Jacobi combinatorial method based on Gröbner basis solution utilizing SOM representation of the measured data points has been demonstrated. This method, similar to the algebraic solution, does not require an initial guess. Parallel computation on a multi-core machine can be utilized decreasing the computation time considerably. The results of the different methods are in the following Tables (13.1 and 13.2).

13.2.2 Fitting a Cylinder

Fitting real circular cylinders is an important problem in the representation of geometry of man-made structures such as industrial plants Vosselman et al. (2004), deformation analysis of tunnels, Stal et al. (2012), detecting and monitoring the deformation in deposition holes, Carrea et al. (2014), positioning of femur pieces for surgical fracture reduction, Winkelbach et al. (2003), estimating tree stems, Khameneh (2013) and so on. Since planes and cylinders compose up to 85% of all objects in industrial scenes, research in 3D reconstruction and modeling—see CAD-CAM applications—have largely focused on these two important geometric primitives, e.g. Petitjean (2002).

In general, to define a cylinder we need 5 parameters: 4 for the axis and 1 for the radius, so one requires at last 5 points to determine parameters. It goes without saying, that in special cases, like the cylinder is parallel to an axis or to a plane, fewer parameters are enough; see Beder and Förstner (2006).

In case of more than 5 points, one has to find the 5 parameters of the cylinder so that the sum of the distances of data points from the cylinder surface is minimum in the least squares sense. Basically, two approaches can be followed:

- find the direction vector of the cylinder center-axis and transform the data points into vertical position to the $x - y$ plane, then the remaining 3 parameters of the cylinder oriented parallel to z axis—2 shifting parameters and the radius—can be computed; see Beder and Förstner (2006).
- All of the 5 parameters are computed via the least squares method using local optimization techniques, e.g. Lukacs et al. (1998), Lichtblau (2006, 2012).

There are different methods to find the orientation of the cylinder center axis:

- one may use Principal Component Analysis (PCA),
- considering the general form of a cylinder as a second order surface, the direction vector can be partially extracted, and computed via linear least squares; see Khameneh (2013),
- the PCA method can be modified such as instead of single points, a local neighborhood of randomly selected points are employed; see Ruiz et al. (2013).
- employing Hough transformation; see Su and Bethel (2010), Rabbani and van den Heuvel (2005).

All of these methods consider a least squares method assuming that model error has normal Gaussian distribution with zero mean value. In this study we consider realistic model error distribution applying maximization likelihood technique for parameter estimation, where this distribution is represented by a Gaussian mixture of the in- and outliers error, and identified by an expectation maximization algorithm. For geometric modeling of a general cylinder, a vector algebraic approach is followed; see Lichtblau (2012).

13.2.2.1 Vector Algebraic Definition

Given a cylinder with axis line L in R^3. We may parametrize it using five model parameters (a, b, c, d, r), where r stands for the radius of the cylinder. If $P(x, y, z)$ is a point of the cylinder and the vector of the axis line of L is $vec = \{1,\ a,\ c\}$, let us consider L' to pass through the origin of the coordinate system and parallel to L. The translation vector is $offset = \{0,\ b,\ d\}$. The result of this translation is $P' \rightarrow P$ and $L' \rightarrow L$. We project this P' point onto $L'$and denote the length of the orthogonal projection *perp*. It is computed as follows.

The vector of P' is projected onto L' and the projection will be subtracted from the vector of P'. Then for the magnitude of *perp*, its norm, one can write

$$\|perp\|^2 - r^2 = 0.$$

Let us apply this definition to set the equation of the cylinder. Consider a point

```
P = {x, y, z};
```

The vector of the locus of this point on the cylinder having model parameters (a, b, c, d, r) can be computed as follows, see Fig. 13.25.

The vector of the axis line

```
vec = {1, a, c};
```

The vector translating the axis line to pass the origin is

```
offset = {0, b, d};
```

The function computing the orthogonal projection *perp*, carries out projection onto the translated axis line L' and subtracts it from the vector of locus P':

```
perp[vec1_, vec_, offset_] :=
 vec1 - offset - Projection[vec1 - offset, vec, Dot]
```

Applying it to point *P:*

```
perp[P, vec, offset] // Simplify
{x - (x + a(-b + y) + c(-d + z))/(1 + a^2 + c^2),
 -b + y - (a(x + a(-b + y) + c(-d + z)))/(1 + a^2 + c^2),
 -d + z - (c(x + a(-b + y) + c(-d + z)))/(1 + a^2 + c^2)}
```

Clearing denominators and formulating of the equation for *perp*

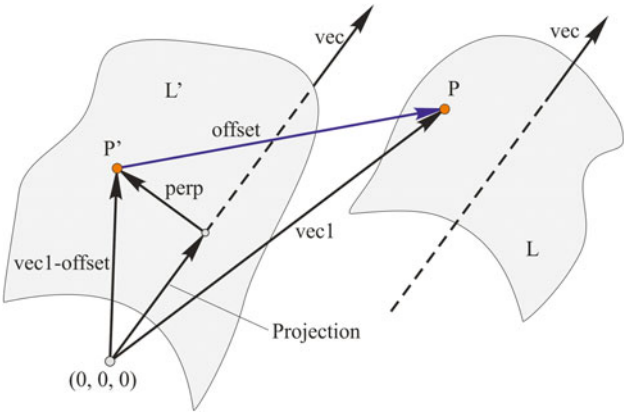


Fig. 13.25 Explanation of perp function

```
vector = Numerator[Together[%.% - r^2]]
b^2 + b^2c^2 - 2abcd + d^2 + a^2d^2 - r^2 -
 a^2r^2 - c^2r^2 + 2abx + 2cdx + a^2x^2 + c^2x^2 - 2by -
 2bc^2y + 2acdy - 2axy + y^2 + c^2y^2 +
 2abcz - 2dz - 2a^2dz - 2cxz - 2acyz + z^2 + a^2z^2
```

This is the implicit equation of the cylinder with model parameters (*a, b, c, d, r*).

It is important to realize that creating this equation, the algebraic error definition, $\|perp\|^2 - r^2$ has been used.

13.2.2.2 Parametrized form of the Cylinder Equation

In order to employ these model parameters to define the parametric equation of the cylinder, let us develop the usual parametric equation in form of $x = x\ (u,\ v)$, $y = y\ (u,\ v)$ and $z = z\ (u,\ v)$ with actual scaling parameters (u, v) and using the model parameters (*a, b, c, d, r*).

The locus of point P is obtained as the sum of a vector on L' plus a vector of length r perpendicular to L. Let v be a parameter along the length of axis L'. Then the projection of P ($u,\ v$) on L is a vector

$$\mu = v\,vec + offset.$$

All vectors perpendicular to L are spanned by any independent pair. We can obtain an orthonormal pair $\{w_1,\ w_2\}$ in the standard way by finding the null space to the matrix whose one row is the vector along the axial direction, that is *vec*, and then using Gram-Schmidt to orthogonalize that pair. Using parameter u, this vector having length r and perpendicular to L can be written as

$$\rho = r\,\cos(u)w_1 + r\,\sin(u)w_2$$

Then the locus vector of a general point of the cylinder is, see Fig. 13.26,

$$\lambda = \mu + \rho.$$

Let us carry out this computation step by step in symbolic way employing *Mathematica*,

```
pair = NullSpace[{vec}];
{w1,w2} = Orthogonalize[pair,Dot];
```

Then the parametric equation of a general circular cylinder using {*a, b, c, d, r*) parameters is

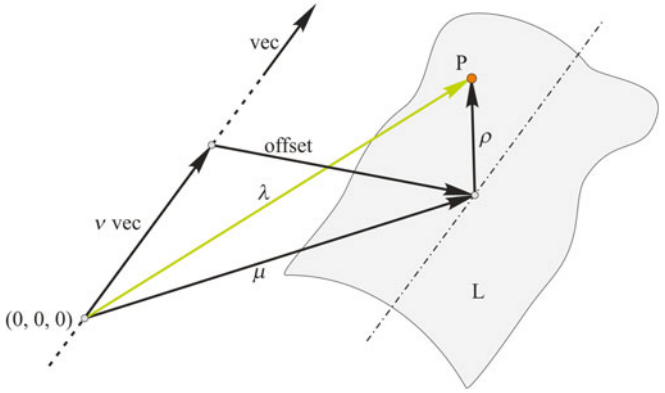


Fig. 13.26 Explanation of general locus vector γ

$$\{v - \frac{cr\cos[u]}{\sqrt{1+c^2}} - \frac{ar\sin[u]}{(1+c^2)\sqrt{1+a^2/(1+c^2)}},\ b + av + \frac{r\sin[u]}{\sqrt{1+a^2/(1+c^2)}},$$
$$d + cv + \frac{r\cos[u]}{\sqrt{1+c^2}} - \frac{acr\sin[u]}{(1+c^2)\sqrt{1+a^2/(1+c^2)}}\}$$

where u and v the actual scaling parameters.

13.2.2.3 Implicit Equation from the Parametric One

One can develop the implicit equation from the parametric one, too. This implicitazion requires that $\sin(u)$, $\cos(u)$ and v all be eliminated from the algebraic system.

$$x = v - \frac{cr\,\cos(u)}{\sqrt{1+c^2}} - \frac{ar\,\sin(u)}{(1+c^2)\sqrt{1+\frac{a^2}{1+c^2}}},$$
$$y = b + av + \frac{r\,\sin(u)}{\sqrt{1+\frac{a^2}{1+c^2}}}$$
$$z = d + cv + \frac{r\,\cos(u)}{\sqrt{1+c^2}} - \frac{acr\,\sin(u)}{(1+c^2)\sqrt{1+\frac{a^2}{1+c^2}}},$$

and

$$\sin^2(u) + \cos^2(u) = 1.$$

In order to carry out this computation in *Mathematica,* for practical reasons let $\sin(u) = U$ and $\cos(u) = V$. Then our system is

```
polys =
Append[vvec + offset + rVw1 + rUw2 - x, y, z, U^2 + V^2 - 1]//Simplify
```

$$\left\{-\frac{arU}{(1+c^2)\sqrt{(1+a^2+c^2)/(1+c^2)}}+v-\frac{crV}{\sqrt{1+c^2}}-x,\ b+\frac{rU}{\sqrt{(1+a^2+c^2)/(1+c^2)}}+av-y,\right.$$
$$\left.d-\frac{acrU}{(1+c^2)\sqrt{(1+a^2+c^2)/(1+c^2)}}+cv+\frac{rV}{\sqrt{1+c^2}}-z,\ -1+U^2+V^2\right\}$$

Let us clear denominators:

$$\left\{-a\sqrt{1+c^2}rU-(1+c^2)\sqrt{\frac{1+a^2+c^2}{1+c^2}}\left(-\sqrt{1+c^2}v+crV+\sqrt{1+c^2}x\right),\right.$$
$$b\sqrt{\frac{1+a^2+c^2}{1+c^2}}+rU+\sqrt{\frac{1+a^2+c^2}{1+c^2}}(av-y),(1+c^2)^{3/2}\sqrt{\frac{1+a^2+c^2}{1+c^2}}d-ac\sqrt{1+c^2}rU+$$
$$\left.(1+c^2)\sqrt{\frac{1+a^2+c^2}{1+c^2}}\left(c\sqrt{1+c^2}v+rV-\sqrt{1+c^2}z\right),\ -1+U^2+V^2\right\}$$

and get

$$\left\{-a\sqrt{1+c^2}rU+\sqrt{1+a^2+c^2}\left(v+c^2v-c\sqrt{1+c^2}rV-x-c^2x\right),\right.$$
$$b\sqrt{1+a^2+c^2}+\sqrt{1+c^2}rU+\sqrt{1+a^2+c^2}(av-y),$$
$$(1+c^2)\sqrt{1+a^2+c^2}d-ac\sqrt{1+c^2}rU+\sqrt{1+a^2+c^2}(cv+$$
$$\left.c^3v+\sqrt{1+c^2}rV-z-c^2z\right),\ -1+U^2+V^2\right\}$$

Applying Gröbner basis to eliminate U and V and v, the implicit form is:

$$b^2+b^2c^2-2abcd+d^2+a^2d^2-r^2-a^2r^2-c^2r^2$$
$$+2abx+2cdx+a^2x^2+c^2x^2-2by-$$
$$2bc^2y+2acdy-2axy+y^2+c^2y^2+2abcz-2dz-$$
$$2a^2dz-2cxz-2acyz+z^2+a^2z^2$$

This is the same equation, which was computed in Sect. 13.2.2.1.

13.2.2.4 Computing Model Parameters

Let us consider five triples of points

```
points5 = Table[{x_i, y_i, z_i}, {i, 1, 5}]
{{x_1, y_1, z_1}, {x_2, y_2, z_2}, {x_3, y_3, z_3}, {x_4, y_4, z_4}, {x_5, y_5, z_5}}
```

Substituting these points into the implicit form and setting $r^2 = rsqr$, we get

```
eqs = Map[implicit/.x → #[[1]],y → #[[2]],z → #[[3]]&,
 points5]/.r^2 → rsqr
{b^2 + b^2c^2 - 2abcd + d^2 + a^2d^2 - rsqr - a^2rsqr -
 c^2rsqr + 2abx1 + 2cdx1 + a^2x1^2 + c^2x1^2 - 2by1 -
 2bc^2y1 + 2acdy1 - 2ax1y1 + y1^2 + c^2y1^2 + 2abcz1 - 2dz1 -
 2a^2dz1 - 2cx1z1 - 2acy1z1 + z1^2 + a^2z1^2,
 b^2 + b^2c^2 - 2abcd + d^2 + a^2d^2 - rsqr -
 a^2rsqr - c^2rsqr + 2abx2 + 2cdx2 + a^2x2^2 + c^2x2^2 - 2by2 -
 2bc^2y2 + 2acdy2 - 2ax2y2 + y2^2 + c^2y2^2 +
 2abcz2 - 2dz2 - 2a^2dz2 - 2cx2z2 - 2acy2z2 + z2^2 + a^2z2^2,
 b^2 + b^2c^2 - 2abcd + d^2 + a^2d^2 - rsqr - a^2rsqr - c^2rsqr + 2abx3 + 2cdx3 +
 a^2x3^2 + c^2x3^2 - 2by3 -
 2bc^2y3 + 2acdy3 - 2ax3y3 + y3^2 + c^2y3^2 + 2abcz3 - 2dz3 - 2a^2dz3 -
 2cx3z3 - 2acy3z3 + z3^2 + a^2z3^2,
 b^2 + b^2c^2 - 2abcd + d^2 + a^2d^2 - rsqr - a^2rsqr - c^2rsqr +
 2abx4 + 2cdx4 + a^2x4^2 + c^2x4^2 - 2by4 -
 2bc^2y4 + 2acdy4 - 2ax4y4 + y4^2 + c^2y4^2 + 2abcz4 - 2dz4 -
 2a^2dz4 - 2cx4z4 - 2acy4z4 + z4^2 + a^2z4^2,
 b^2 + b^2c^2 - 2abcd + d^2 + a^2d^2 - rsqr - a^2rsqr -
 c^2rsqr + 2abx5 + 2cdx5 + a^2x5^2 + c^2x5^2 - 2by5 -
 2bc^2y5 + 2acdy5 - 2ax5y5 + y5^2 + c^2y5^2 + 2abcz5 - 2dz5 -
 2a^2dz5 - 2cx5z5 - 2acy5z5 + z5^2 + a^2z5^2}
```

This is a polynomial system for the parameters based on the algebraic error definition. In order to create the determined system for the geometric error model, let us consider *geometric error model*, namely $\|perp\| - r$,

$$\Delta_i = \|perp\| - r \rightarrow \Delta_i^2 = \left(\sqrt{\#.\#} - r\right)^2$$

which can be applied to the `points5` data, then

```
eqs = Map[Numerator[Together[Sqrt[#.#] - r]]&,
 Map[perp[#,vec,offset]&,points5]]
{-r + Sqrt[(x1 - (x1 + a(-b + y1) + c(-d + z1))/(1 + a^2 + c^2))^2 +
  (-b + y1 - (a(x1 + a(-b + y1) + c(-d + z1)))/(1 + a^2 + c^2))^2 +
  (-d + z1 - (c(x1 + a(-b + y1) + c(-d + z1)))/(1 + a^2 + c^2))^2],
 -r + Sqrt[(x2 - (x2 + a(-b + y2) + c(-d + z2))/(1 + a^2 + c^2))^2 +
  (-b + y2 - (a(x2 + a(-b + y2) + c(-d + z2)))/(1 + a^2 + c^2))^2
  + (-d + z2 - (c(x2 + a(-b + y2) + c(-d + z2)))/(1 + a^2 + c^2))^2],
```

$$
\begin{aligned}
&-r+\mathrm{Sqrt}\left[\left(x_3-\frac{x_3+a(-b+y_3)+c(-d+z_3)}{1+a^2+c^2}\right)^2+\right.\\
&\left(-b+y_3-\frac{a(x_3+a(-b+y_3)+c(-d+z_3))}{1+a^2+c^2}\right)^2\\
&\left.+\left(-d+z_3-\frac{c(x_3+a(-b+y_3)+c(-d+z_3))}{1+a^2+c^2}\right)^2\right],\\
&-r+\mathrm{Sqrt}\left[\left(x_4-\frac{x_4+a(-b+y_4)+c(-d+z_4)}{1+a^2+c^2}\right)^2+\right.\\
&\left(-b+y_4-\frac{a(x_4+a(-b+y_4)+c(-d+z_4))}{1+a^2+c^2}\right)^2\\
&\left.+\left(-d+z_4-\frac{c(x_4+a(-b+y_4)+c(-d+z_4))}{1+a^2+c^2}\right)^2\right],\\
&-r+\mathrm{Sqrt}\left[\left(x_5-\frac{x_5+a(-b+y_5)+c(-d+z_5)}{1+a^2+c^2}\right)^2+\right.\\
&\left(-b+y_5-\frac{a(x_5+a(-b+y_5)+c(-d+z_5))}{1+a^2+c^2}\right)^2\\
&\left.+\left(-d+z_5-\frac{c(x_5+a(-b+y_5)+c(-d+z_5))}{1+a^2+c^2}\right)^2\right]\}
\end{aligned}
$$

13.2.2.5 Computing Model Parameters in Overdetermined Case

Let us consider an *algebraic error model* first. This means to minimize the residual of the implicit form

$$G(a,b,c,d,r)=\sum_{i=1}^{n}\Delta_i^2$$

where the algebraic error is, see `implicit` or `vector`,

$$
\begin{aligned}
\Delta_i = {} & b^2+b^2c^2-2abcd+d^2+a^2d^2-rsqr-a^2rsqr-\\
& c^2rsqr+2abx_i+2cdx_i+a^2x_i^2+c^2x_i^2-2by_i-\\
& 2bc^2y_i+2acdy_i-2ax_iy_i+y_i^2+c^2y_i^2+2abcz_i-2dz_i-\\
& 2a^2dz_i-2cx_iz_i-2acy_iz_i+z_i^2+a^2z_i^2,
\end{aligned}
$$

while the geometric error is,

$$
\begin{aligned}
&\Delta_i=\mathtt{Map[Numerator[Together[}\sqrt{\#.\#}-r\mathtt{]]\&,Map[perp[\#,vec,offset]\&,}\\
&\mathtt{\{\{}x_i,y_i,z_i\mathtt{\}\}]]}\\
&\left\{-r+\mathrm{Sqrt}\left[\left(x_i-\frac{x_i+a(-b+y_i)+c(-d+z_i)}{1+a^2+c^2}\right)^2+\right.\right.\\
&\left(-b+y_i-\frac{a(x_i+a(-b+y_i)+c(-d+z_i))}{1+a^2+c^2}\right)^2+\\
&\left.\left(-d+z_i-\frac{c(x_i+a(-b+y_i)+c(-d+z_i))}{1+a^2+c^2}\right)^2\right]\}
\end{aligned}
$$

In order to carry out this minimization problem via local minimization, since that is much faster than the global one, we may solve the determined system ($n = 5$) for randomly selected data points to get initial guess values.

13.2.2.6 Application to Estimation of Tree Stem Diameter

Outdoor laser scanning measurements have been carried out in the backyard of Budapest University of Technology and Economics, see Fig. 13.27 left. In order to get a simple fitting problem rather than a segmentation one, the test object, the lower part of the trunk of a tree was preselected by segmentation, see Fig. 13.27 right. The experiment has been carried out with a Faro Focus 3D terrestrial laser scanner, see Fig. 13.28.

The scanning parameters were set to 1/2 resolution that equals to 3 mm/10 m point spacing. The test data set was cropped from the point cloud; moreover, further resampling was applied in order to reduce the data size. The final data set is composed of 16 434 points in ASCII format, and only the x, y, z coordinates were kept (no intensity values). Let us load the measured data of 16434 points: (Fig. 13.29).

We shall compute the parameters from the geometric error model employing local minimization of the sum of the squares of the residual. To do that we compute the initial guess values from five randomly chosen points of data using the algebraic

Fig. 13.27 Test environment (to left) and test object (to right)

Fig. 13.28 Faro Focus 3D scanner

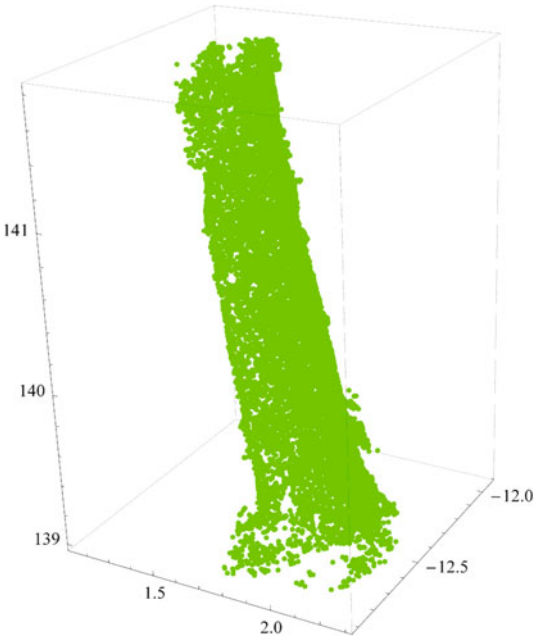


Fig. 13.29 The points of cloud of data

error model, since the resulting polynomial system can be solved easily via numerical Gröbner basis. In the deterministic case, the number of real solutions must be an even number (0, 2, 4, or 6). According to our numerical experiences, to ensure reliable initial values, one needs such a five points that provide at least four real solutions.

```
dataPR = RandomSample[dataP, 5]
{{1.5402, -12.2018, 140.461}, {1.5536, -12.1603, 141.285},
 {1.4512, -12.319, 141.43}, {1.6294, -12.3794, 140.952},
 {1.6821, -12.2508, 140.86}}
```

The equations are

```
perps = Map[perp[#, vec, offset] &, dataPR]
{{1.5402 - (1.5402 + a(-12.2018 - b) + c(140.461 - d))/(1 + a^2 + c^2),
  -12.2018 - b - (a(1.5402 + a(-12.2018 - b)
  + c(140.461 - d)))/(1 + a^2 + c^2),
 140.461 - (c(1.5402 + a(-12.2018 - b)
  + c(140.461 - d)))/(1 + a^2 + c^2) - d},
{1.5536 - (1.5536 + a(-12.1603 - b) + c(141.285 - d))/(1 + a^2 + c^2),
 -12.1603 - b - (a(1.5536 + a(-12.1603 - b)
  + c(141.285 - d)))/(1 + a^2 + c^2),
 141.285 - (c(1.5536 + a(-12.1603 - b)
  + c(141.285 - d)))/(1 + a^2 + c^2) - d},
```

```
 {1.4512 - (1.4512 + a(-12.319 - b) + c(141.43 - d))/(1 + a^2 + c^2),
 -12.319 - b - (a(1.4512 + a(-12.319 - b)
  + c(141.43 - d)))/(1 + a^2 + c^2),
 141.43 - (c(1.4512 + a(-12.319 - b)
  + c(141.43 - d)))/(1 + a^2 + c^2) - d},
{1.6294 - (1.6294 + a(-12.3794 - b) + c(140.952 - d))/(1 + a^2 + c^2),
 -12.3794 - b - (a(1.6294 + a(-12.3794 - b)
  + c(140.952 - d)))/(1 + a^2 + c^2),
 140.952 - (c(1.6294 + a(-12.3794 - b)
  + c(140.952 - d)))/(1 + a^2 + c^2) - d},
{1.6821 - (1.6821 + a(-12.2508 - b) + c(140.86 - d))/(1 + a^2 + c^2),
 -12.2508 - b - (a(1.6821 + a(-12.2508 - b)
  + c(140.86 - d)))/(1 + a^2 + c^2),
 140.86 - (c(1.6821 + a(-12.2508 - b)
  + c(140.86 - d)))/(1 + a^2 + c^2) - d}}
```

In order to avoid round off error, one may use higher precision or alternatively employ integer coefficients

```
exprs = Map[Numerator[Together[Rationalize[#.#, 0] - rsqr]]&, perps]
{248477205553 + 469830309a + 246645809213a^2 +
 305045000b + 38505000ab + 12500000b^2 -
 5408450805c + 42846925746ac + 3511525000abc +
 1890701741c^2 + 305045000bc^2 +
 12500000b^2c^2 + 3511525000d + 3511525000a^2d +
 38505000cd + 305045000acd + 25000000abcd +
 12500000d^2 + 12500000a^2d^2 - 12500000rsqr -
 12500000a^2rsqr - 12500000c^2rsqr,
 2010932412109 + 3778448416a + 1996386489796a^2 +
 2432060000b + 310720000ab +
 100000000b^2 - 43900075200c + 343613597100ac +
 28257000000abc + 15028656905c^2 +
 2432060000bc^2 + 100000000b^2c^2 -
 28257000000d - 28257000000a^2d + 310720000cd -
 2432060000acd - 200000000abcd + 100000000d^2 +
 100000000a^2d^2 - 100000000rsqr -
 100000000a^2rsqr - 100000000c^2rsqr,
 503855066525 + 893866640a + 500113772036a^2 + 615950000b +
 72560000ab + 25000000b^2 -
 10262160800c + 87113808500ac + 7071500000abc +
 3846593561c^2 + 615950000bc^2 +
 25000000b^2c^2 - 7071500000d - 7071500000a^2d +
 72560000cd - 615950000acd - 50000000abcd +
 25000000d^2 + 25000000a^2d^2 -
```

```
25000000rsqr - 25000000a^2rsqr - 25000000c^2rsqr,
500517896209 + 1008549718a + 496753031209a^2 +
618970000b + 81470000ab + 25000000b^2 -
11483359440c + 87245059440ac + 7047600000abc +
3897612218c^2 + 618970000bc^2 +
25000000b^2c^2 - 7047600000d - 7047600000a^2d +
81470000cd - 618970000acd - 50000000abcd +
25000000d^2 + 25000000a^2d^2 - 25000000rsqr -
25000000a^2rsqr - 25000000c^2rsqr,
1999162170064 + 4121414136a + 1984436906041a^2 +
2450160000b + 336420000ab +
100000000b^2 - 47388121200c + 345129537600ac +
28172000000abc + 15291156105c^2 +
2450160000bc^2 + 100000000b^2c^2 - 28172000000d -
28172000000a^2d + 336420000cd -
2450160000acd - 200000000abcd + 100000000d^2 +
100000000a^2d^2 - 100000000rsqr -
100000000a^2rsqr - 100000000c^2rsqr}
```

Solving the system via numerical Gröbner basis yields six solutions:

```
sol5 = NSolve[exprs, {a, b, c, d, rsqr}]
{{a → - 0.745866, b → - 11.1109, c → - 22.1205,
 d → 175.724, rsqr → 0.0124972},
 {a → - 0.674735, b → - 11.2742, c → - 3.34268,
 d → 145.884, rsqr → 0.0316293},
 {a → - 0.0533048 + 0.95706 i, b → - 12.1959 - 1.50844 i,
 c → 0.372737 + 0.0412819 i,
 d → 140.563 - 0.0688756 i, rsqr → 0.097419 - 0.00128866 i},
 {a → - 0.0533048 - 0.95706 i, b → - 12.1959 + 1.50844 i,
 c → 0.372737 - 0.0412819 i,
 d → 140.563 + 0.0688756 i, rsqr → 0.097419 + 0.00128866 i},
 {a → 2.43015, b → - 14.9782, c → - 1.54214,
 d → 142.734, rsqr → 0.268222},
 {a → - 2.03283, b → - 9.11152, c → 5.59169,
 d → 132.397, rsqr → 0.0299727}}
```

Of these, the real solutions are:

```
solnsR = Select[sol5, Im[#[[1, 2]]] == 0&]
{{a → - 0.745866, b → - 11.1109, c → - 22.1205,
 d → 175.724, rsqr → 0.0124972},
 {a → - 0.674735, b → - 11.2742, c → - 3.34268,
 d → 145.884, rsqr → 0.0316293},
```

```
{a → 2.43015, b → -14.9782, c → -1.54214,
d → 142.734, rsqr → 0.268222},
{a → -2.03283, b → -9.11152, c → 5.59169,
d → 132.397, rsqr → 0.0299727}}
```

We compute the corresponding *r*:

```
{{a → -0.745866, b → -11.1109, c → -22.1205,
d → 175.724, r → 0.111791},
{a → -0.674735, b → -11.2742, c → -3.34268,
d → 145.884, r → 0.177846},
{a → 2.43015, → -14.9782, c → -1.54214,
d → 142.734, r → 0.517901},
{a → > -2.03283, b → -9.11152, c → 5.59169,
d → 132.397, r → 0.173126}}
```

From these solutions we can select the one that gives the smallest residual. It is:

```
{a → -0.674735, b → -11.2742, c → -3.34268, d → 145.884, r → 0.177846}
```

This solution will be used as initial guess values for the local minimization. The result, with plot, is: (Fig. 13.30)

```
{25.041, {a → -0.604342, b → -11.3773, c → -3.68704,
d → 146.479, r → 0.178783}}
```

The distribution of the model error

```
error = exprs/.sol[[2]];
p5 = Histogram[error, Automatic, "PDF"]
```

```
Mean[error]
2.08444 × 10^-16
```

```
Min[error]
-0.0813939
```

```
Max[error]
0.224278
```

```
StandardDeviation[error]
0.0390362
```

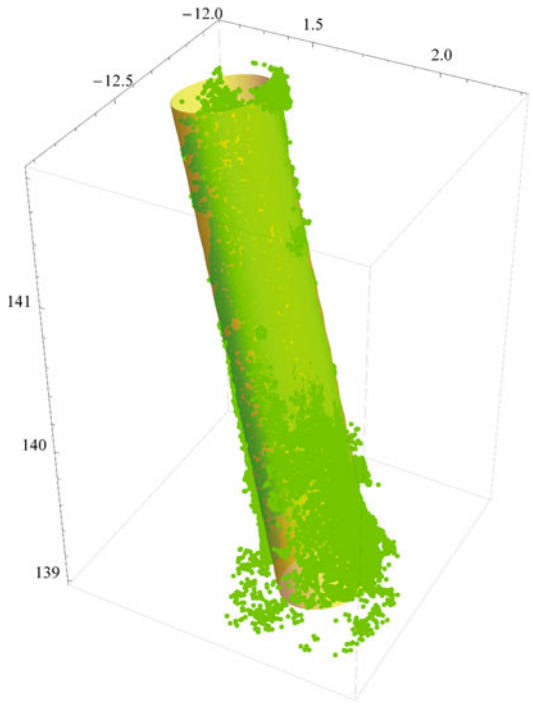


Fig. 13.30 The fitted cylinder via geometrical error model

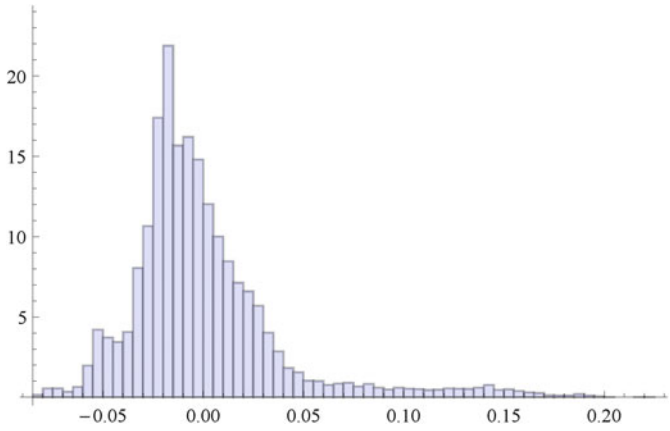


Fig. 13.31 Histogram of the model error of the standard approach assuming $N(0, s)$

It is clear from Fig. 13.31 that the assumption for the model error is not true. Therefore one should employ a likelihood function for the parameter estimation. The real model error distribution as well as the model parameters can be computed

in an iterative way. We can consider the distribution shown in Fig. 13.31 as a first guess for this iteration. In order to handle an empirical distribution in the likelihood function there are at least three ways:

- Gaussian kernel functions can be applied to the values of different bins,
- Employing a known type of distribution to approximate the empirical histogram,
- Considering the empirical distribution as a Gaussian mixture of errors corresponding to inliers and outliers

Here, we considered the third approach, and employed an expectation maximization algorithm to compute the parameters of the component distributions.

13.2.2.7 Expectation Maximization

Let us consider a two-component Gaussian mixture represented by the mixture model in the following form,

$$\mathcal{N}_{12}(x) = \eta_1 \mathcal{N}(\mu_1, \sigma_1, x) + \eta_2 \mathcal{N}(\mu_2, \sigma_2, x)$$

where

$$\mathcal{N}(\mu_i, \sigma_i, x) = \frac{e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}}{\sqrt{2\pi}\sigma_i}, \quad i = 1, 2$$

and η_i 's are the membership weights constrained with

$$\eta_1 + \eta_2 = 1.$$

We are looking for the parameter values (μ_1, s_1) and (μ_2, s_2). The log-likelihood function in case of N samples is

$$\begin{aligned} Log\mathcal{L}(x_i, \theta) &= \sum_{i=1}^{N} \log(\mathcal{N}_{12}(x_i, \theta)) \\ &= \sum_{i=1}^{N} \log(\eta_1 \mathcal{N}(\mu_1, \sigma_1, x_i) + \eta_2 \mathcal{N}(\mu_2, \sigma_2, x_i)), \end{aligned}$$

where θ = (μ_1, s_1, μ_2, s_2) the parameters of the normal densities. The problem is the direct maximization of this function, because of the sum of terms inside the logarithm. In order to solve this problem let us introduce the following alternative log-likelihood function

$$Log\mathcal{L}(x_i,\theta,\Delta)=\sum_{i=1}^{N}(1-\Delta_i)\log(\mathcal{N}(\mu_1,\sigma_1,x_i))+\Delta_i\log(\mathcal{N}(\mu_2,\sigma_2,x_i))$$
$$+\sum_{i=1}^{N}(1-\Delta_i)\log(\alpha_1)+\Delta_i\log(\alpha_2)$$

Here Δ_i 's are considered as unobserved latent variables taking values 0 or 1. If x_i belongs to the first component then $\Delta_\mathrm{i}=0$, so

$$Log\mathcal{L}(x_i,\theta,\Delta)=\sum_{i\in N_1(\Delta)}\log(\mathcal{N}(\mu_1,\sigma_1,x_i))+N_1\log(\alpha_1),$$

otherwise x_i belongs to the second component then $\Delta_\mathrm{i}=1$, therefore

$$Log\mathcal{L}(x_i,\theta,\Delta)=\sum_{i\in N_2(\Delta)}\log(\mathcal{N}(\mu_2,\sigma_2,x_i))+N_2\log(\alpha_2),$$

where N_1 and N_2 are the number of the elements of the mixture, which belong to the first and to the second component, respectively.

Since the values of the Δ_i 's are actually unknown, we proceed in an iterative fashion, substituting for each Δ_i its expected value,

$$\xi_i(\theta)=E(\Delta_i|\theta,x)=\Pr(\Delta_i=1|\theta,x)\approx\frac{\eta_2\mathcal{N}(\mu_2,\sigma_2,x_i)}{(1-\eta_2)\mathcal{N}(\mu_1,\sigma_1,x_i)+\eta_2\mathcal{N}(\mu_2,\sigma_2,x_i)},$$

This expression is also called the responsibility of component two for observation i. Then the procedure, called the EM algorithm for two-component Gaussian mixture, is the following:

- Take initial guess for the parameters: $\theta=(\tilde{\mu}_1,\tilde{\sigma}_1,\tilde{\mu}_2,\tilde{\sigma}_2)$ and for $\tilde{\eta}_2$
- *Expectation Step*: compute the responsibilities:

$$\tilde{\xi}_2=\frac{\tilde{\eta}_2\mathcal{N}(\tilde{\mu}_2,\tilde{\sigma}_2,x_i)}{(1-\tilde{\eta}_2)\mathcal{N}(\tilde{\mu}_1,\tilde{\sigma}_1,x_i)+\tilde{\eta}_2\mathcal{N}(\tilde{\mu}_2,\tilde{\sigma}_2,x_i)},\quad fo\,i=1,2,\ldots,N,$$

- *Maximization Step*: compute the weighted means and variances for the two components:

$$\tilde{\mu}_1=\sum_{i=1}^{N}\Big(1-\tilde{\xi}_i\Big)x_i/\sum_{i=1}^{N}\Big(1-\tilde{\xi}_i\Big),$$
$$\tilde{\sigma}_1=\sum_{i=1}^{N}\Big(1-\tilde{\xi}_i\Big)(x_i-\tilde{\mu}_1)^2/\sum_{i=1}^{N}\Big(1-\tilde{\xi}_i\Big),$$

$$\tilde{\mu}_2 = \sum_{i=1}^{N} \tilde{\xi}_i x_i / \sum_{i=1}^{N} \tilde{\xi}_i,$$
$$\tilde{\sigma}_2 = \sum_{i=1}^{N} \tilde{\xi}_i (x_i - \tilde{\mu}_1)^2 / \sum_{i=1}^{N} \tilde{\xi}_i,$$

and the mixing probability

$$\tilde{\eta}_2 = \sum_{i=1}^{N} \tilde{\xi}_i / N,$$

- Iterate these steps until convergence.

This algorithm is implemented for *Mathematica* see, Fox et al. (2013) as a *Mathematica* Demonstration project. The code has been modified and applied here. In the next section we illustrate how this function works.

The result of the parameter values is:

```
{{0.0813029,0.0535765},{-0.00940588,0.0226878},
{0.103693,0.896307}}
```

These can be displayed in a table form (Table 13.3)
Figure 13.32 shows the density functions of the two component (Figs. 13.33, 13.34).

Table 13.3 Parameters of the Gaussian mixture after zero iteration

η	μ	s
0.103693	0.0813029	0.0535765
0.896307	−0.00940588	0.0226878

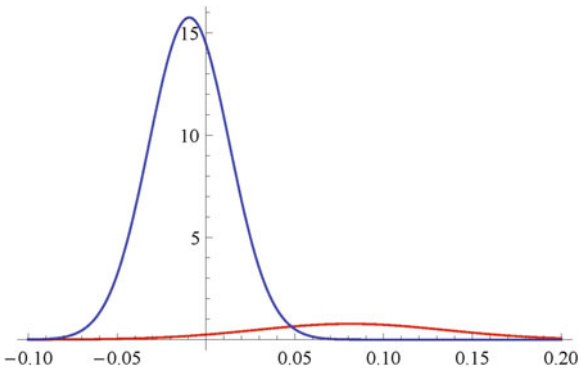


Fig. 13.32 The PDF of the two components

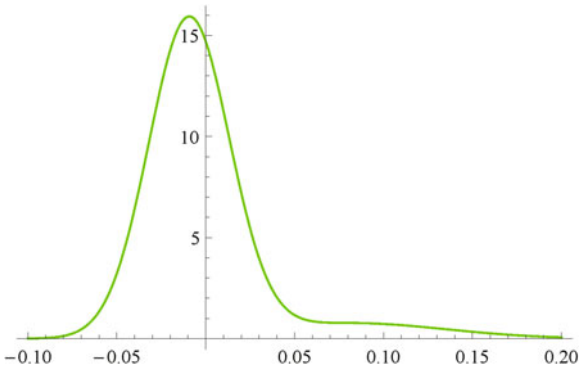


Fig. 13.33 The joint PDF of the mixture

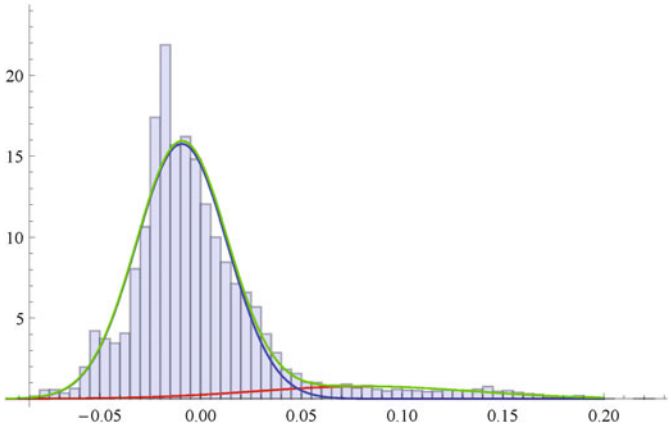


Fig. 13.34 The PDF of the two components and the normalized histogram of the data

The following figures show the convergence of the different parameters (Figs. 13.35, 13.36, 13.37).
Next, we separate the mixture of the samples into two clusters: cluster of outliers and cluster of inliers. Membership values of the first cluster

```
Short[Δ[[1]],10]
{0.163639,0.00848004,0.0116534,1.,0.00851251,0.0156639,0.00894375,
 0.00945738,≪16418≫,1.,1.,1.,1.,1.,0.999997,0.999995}
```

Membership values of the second cluster

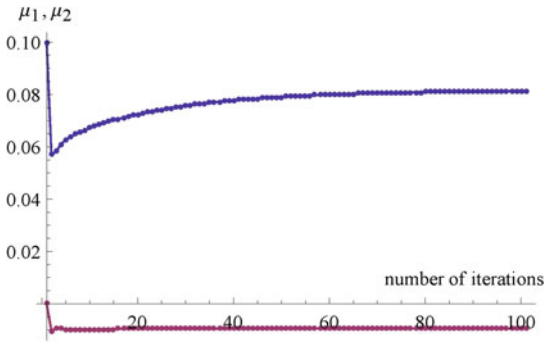


Fig. 13.35 The convergence of the means of the two components

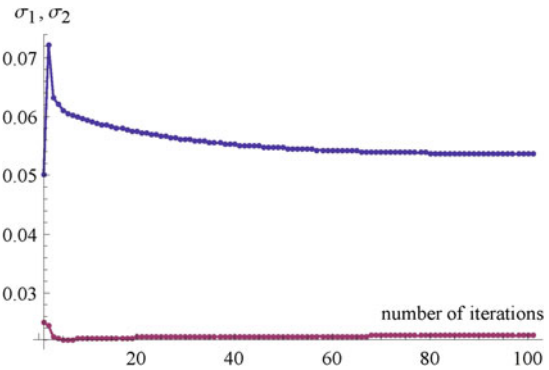


Fig. 13.36 The convergence of the standard deviations

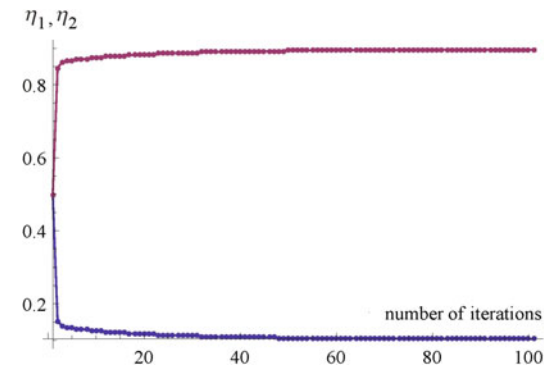


Fig. 13.37 The convergence of the membership weights

```
Short[Δ[[2]],10]
0.836361,0.99152,0.988347,7.93708 × 10^-8,0.991487,0.984336,0.991056,
 0.990543,≪16419≫,5.63356 × 10^-11,4.26408 × 10^-11,1.24908 × 10^-10,
 1.40798 × 10^-11,2.28746 × 10^-8,3.32431 × 10^-6,5.16318 × 10^-6
```

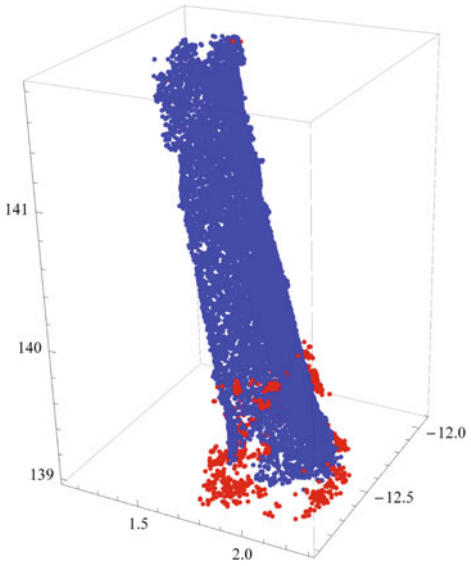


Fig. 13.38 The inliers (blue) and outliers (red) as a first guess resulted standard minimization of the total residual

In order to get Boolean (crisp) clustering let us round the membership values

```
S1 = Round[Δ[[1]]]; S2 = Round[Δ[[2]]];
```

The elements in the first cluster (outliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S1)

```
XYZOut = Map[dataP[[#]]&, Position[S1, 1]//Flatten];
```

The number of these elements (outliers) is 1318.

Similarly, the elements in the second cluster (inliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S2). There are 15116.

Let us display the ouliers and the inliers (Fig. 13.38).

13.2.2.8 Maximum Likelihood for Gaussian Mixture

The likelihood function for a two-component Gaussian mixture can be written as,

$$Log\mathcal{L}(x_i, \theta) = \sum_{i \in N_1} \log(\mathcal{N}(\mu_1, \sigma_1, x_i)) + \sum_{i \in N_2} \log(\mathcal{N}(\mu_2, \sigma_2, x_i)) + N_1 \log(\eta_1) + N_2 \log(\eta_2),$$

where the likelihood function for one of the components can be developed as it follows.

We have seen that the geometric error is

$$\Delta_i = -r + Sqrt\left\{\left(x_i - \frac{x_i + a(-b+y_i) + c(-d+z_i)}{1+a^2+c^2}\right)^2 + \left(-b+y_i - \frac{a(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\right)^2 + \left(-d+z_i - \frac{c(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\right)^2\right\}$$

The probability density function for a single Gaussian

$$\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma}.$$

Let us substitute the expression of the geometric error, $(x \to \Delta_i)$

$$pdf = \frac{e^{-\left(\frac{-r-\mu+\sqrt{\left(x_i - \frac{x_i + a(-b+y_i) + c(-d+z_i)}{1+a^2+c^2}\right)^2 + \left(-b+y_i - \frac{a(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\right)^2 + \left(-d+z_i - \frac{c(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\right)^2}}{2\sigma^2}\right)^2}}{\sqrt{2\pi}\sigma}.$$

We apply a *Mathematica* function developed by Rose and Smith (2000) which is entitled as `SuperLog`. This function utilizes pattern-matching code that enhances *Mathematica*'s ability to simplify expressions involving the natural logarithm of a product of algebraic terms.

Then the likelihood function is

$$\mathcal{L} = \prod_{i=1}^{N} pdf = \prod_{i=1}^{N} \frac{e^{-\left(\frac{-r-\mu+\sqrt{\left(x_i - \frac{x_i + a(-b+y_i) + c(-d+z_i)}{1+a^2+c^2}\right)^2 + \left(-b+y_i - \frac{a(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\right)^2 + \left(-d+z_i - \frac{c(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\right)^2}}{2\sigma^2}\right)^2}}{\sqrt{2\pi}\sigma}.$$

Considering its logarithm which should be maximized

$$\text{Log}\mathcal{L} = \text{Log}[\mathcal{L}]$$

$$-\frac{b^2N}{2\sigma^2} - \frac{a^2b^2N}{2(1+a^2+c^2)^2\sigma^2} - \frac{a^4b^2N}{2(1+a^2+c^2)^2\sigma^2} - \frac{a^2b^2c^2N}{2(1+a^2+c^2)^2\sigma^2} + \frac{a^2b^2N}{(1+a^2+c^2)\sigma^2} -$$
$$\frac{abcdN}{(1+a^2+c^2)^2\sigma^2} - \frac{a^3bcdN}{(1+a^2+c^2)^2\sigma^2} - \frac{abc^3dN}{(1+a^2+c^2)^2\sigma^2} + \frac{2abcdN}{(1+a^2+c^2)\sigma^2} - \frac{d^2N}{2\sigma^2} -$$
$$\frac{c^2d^2N}{2(1+a^2+c^2)^2\sigma^2} - \frac{a^2c^2d^2N}{2(1+a^2+c^2)^2\sigma^2} - \frac{c^4d^2N}{2(1+a^2+c^2)^2\sigma^2} + \frac{c^2d^2N}{(1+a^2+c^2)\sigma^2} - \frac{r^2N}{2\sigma^2} - \frac{rN\mu}{\sigma^2} -$$
$$\frac{N\mu^2}{2\sigma^2} - \frac{1}{2}N\log[2] - \frac{1}{2}N\log[\pi] - N\log[\sigma] + \sum_{i=1}^{N}\frac{abx_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^3bx_i}{(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{abc^2x_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{2abx_i}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N}\frac{cdx_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^2cdx_i}{(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{c^3dx_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{2cdx_i}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N} - \frac{x_i^2}{2\sigma^2} + \sum_{i=1}^{N} - \frac{x_i^2}{2(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N} - \frac{a^2x_i^2}{2(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{c^2x_i^2}{2(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{x_i^2}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N}\frac{by_i}{\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{a^2by_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^4by_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^2bc^2y_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{2a^2by_i}{(1+a^2+c^2)\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{acdy_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^3cdy_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{ac^3dy_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{2acdy_i}{(1+a^2+c^2)\sigma^2} +$$
$$\sum_{i=1}^{N} - \frac{ax_iy_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{a^3x_iy_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{ac^2x_iy_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{2ax_iy_i}{(1+a^2+c^2)\sigma^2} +$$
$$\sum_{i=1}^{N} - \frac{y_i^2}{2\sigma^2} + \sum_{i=1}^{N} - \frac{a^2y_i^2}{2(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{a^4y_i^2}{2(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{a^2c^2y_i^2}{2(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{a^2y_i^2}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N}\frac{abcz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^3bcz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{abc^3z_i}{(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N} - \frac{2abcz_i}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N}\frac{dz_i}{\sigma^2} + \sum_{i=1}^{N}\frac{c^2dz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{a^2c^2dz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{c^4dz_i}{(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N} - \frac{2c^2dz_i}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N} - \frac{cx_iz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{a^2cx_iz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{c^3x_iz_i}{(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{2cx_iz_i}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N} - \frac{acy_iz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{a^3cy_iz_i}{(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{ac^3y_iz_i}{(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{2acy_iz_i}{(1+a^2+c^2)\sigma^2} + \sum_{i=1}^{N} - \frac{z_i^2}{2\sigma^2} + \sum_{i=1}^{N} - \frac{c^2z_i^2}{2(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N} - \frac{a^2c^2z_i^2}{2(1+a^2+c^2)^2\sigma^2} +$$
$$\sum_{i=1}^{N} - \frac{c^4z_i^2}{2(1+a^2+c^2)^2\sigma^2} + \sum_{i=1}^{N}\frac{c^2z_i^2}{(1+a^2+c^2)\sigma^2} +$$
$$\sum_{i=1}^{N}\frac{1}{\sigma^2}r\,\text{Sqrt}\Big\{\Big(x_i - \frac{x_i + a(-b+y_i) + c(-d+z_i)}{1+a^2+c^2}\Big)^2 +$$
$$\Big\{\Big(-b + y_i - \frac{a(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\Big)^2 + \Big(-d + z_i - \frac{c(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\Big)^2\Big\} +$$
$$\sum_{i=1}^{N}\frac{1}{\sigma^2}\mu\,\text{Sqrt}\Big\{\Big(x_i - \frac{x_i + a(-b+y_i) + c(-d+z_i)}{1+a^2+c^2}\Big)^2 +$$
$$\Big(-b + y_i - \frac{a(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\Big)^2 + \Big(-d + z_i - \frac{c(x_i + a(-b+y_i) + c(-d+z_i))}{1+a^2+c^2}\Big)^2\Big\}$$

It is a rather complicated expression, but fortunately *Mathematica* has a built in function to compute this expression numerically (`Likelihood[]`).

Now we can create the corresponding likelihood functions for the identified outliers and inliers. We do not display the *Mathematica* code here.

In order to carry out local maximization, the result of the standard parameter estimation (from above) can be considered

```
{a → -0.604342,b → -11.3773,c → -3.68704,d → 146.479,
 r → 0.178783}
```

The result of the computation is:

```
{a → -0.643829,b → -11.3064,c → -3.60796,d → 146.386,
 r → 0.175172}
```

Now, let us compute the model error distribution and show the histogram:

This means that employing the model error distribution represented by Fig. 13.31, and using maximum likelihood method for estimating the parameters, with these parameters we get a model error distribution represented by Fig. 13.39. If the two distributions are the same, than the parameter estimation is correct. To answer this question let us employ EM for this later distribution.

The result of the parameter values is

```
{{0.0826151,0.0596793},{-0.0111856,0.018304},{0.118863,0.881137}}
```

These can be displayed in a table form (Table 13.4)
Figure 13.40 shows the density functions of the two components (Figs. 13.41, 13.42, 13.43, 13.44, 13.45).

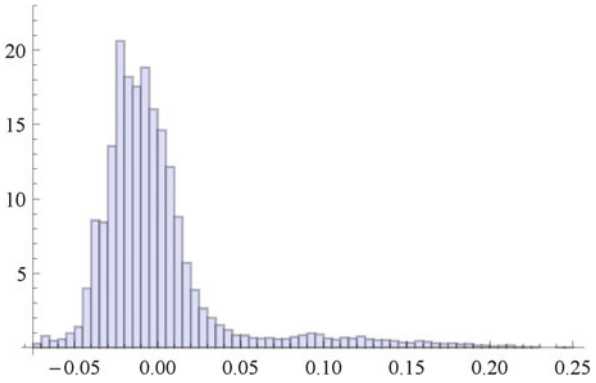


Fig. 13.39 Histogram of the model error employing maximum likelihood method

Table 13.4 Parameters of the Gaussian mixture after zero iteration

η	μ	s
0.118863	0.0826151	0.0596793
0.881137	−0.0111856	0.018304

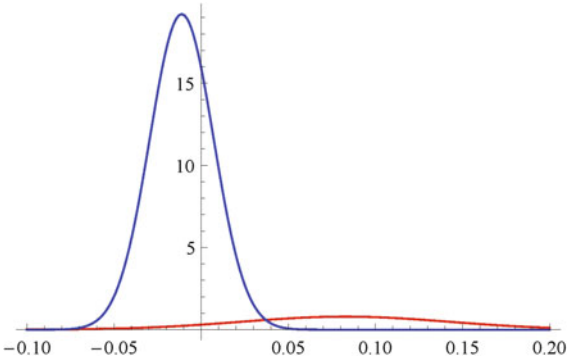


Fig. 13.40 The PDF of the two components

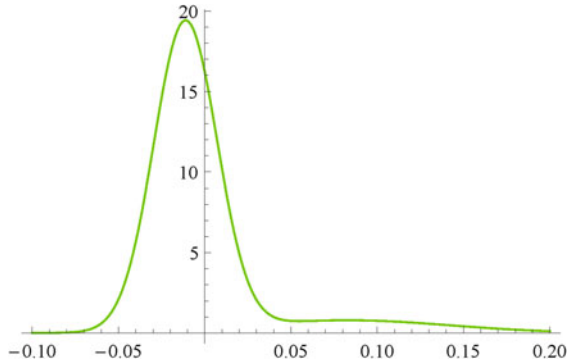


Fig. 13.41 The joint PDF of the mixture

We now separate the mixture of the samples into two clusters: cluster of outliers and cluster of inliers. Membership values of the first cluster:

```
Short[Δ[[1]],10]
{0.0825756,0.0117538,0.0342882,1.,0.0125805,0.0147983,0.0104937,
 0.0196103,0.0308094,0.0161561,0.0126288,≪16412≫,0.99999,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.}
```

Membership values of the second cluster:

```
Short[Δ[[1]],10]
{0.917424,0.988246,0.965712,1.78221 × 10^-11,0.98742,0.985202,
 0.989506,≪16420≫,2.3251 × 10^-21,8.37015 × 10^-24,4.22025 × 10^-21,
 1.97601 × 10^-24,1.26222 × 10^-17,1.2341 × 10^-13,2.78243 × 10^-13}
```

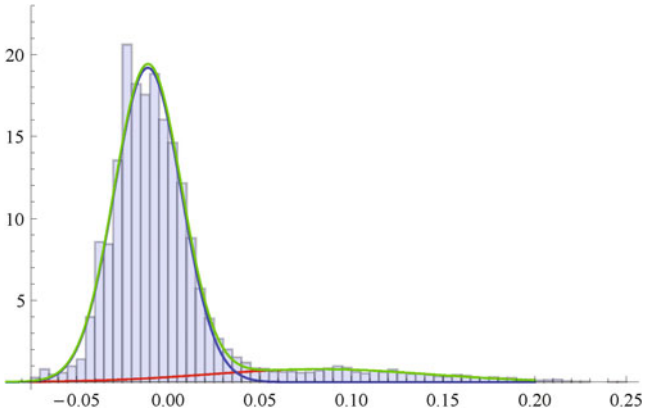


Fig. 13.42 The PDF of the two components and the normalized histogram of the data

Fig. 13.43 The convergence of the means of the two components

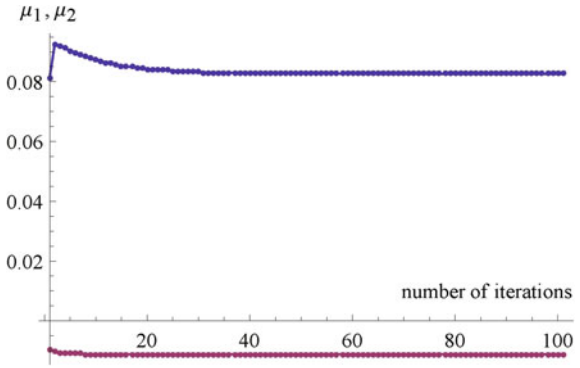
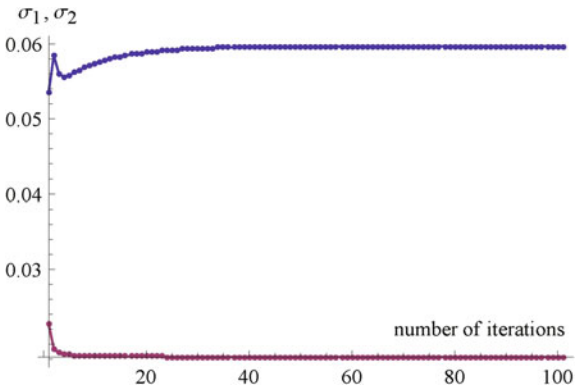


Fig. 13.44 The convergence of the standard deviations



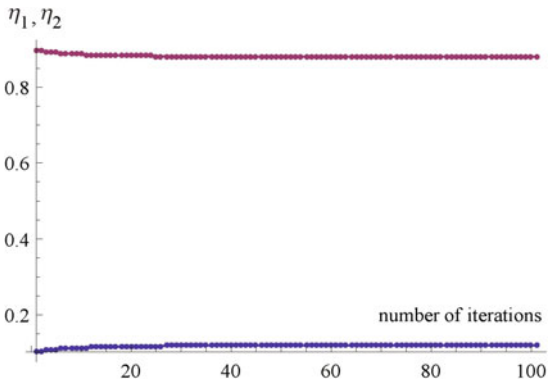


Fig. 13.45 The convergence of the membership weights

In order to get Boolean (crisp) clustering let us round the membership values.

```
S1 = Round[Δ[[1]]]; S2 = Round[Δ[[2]]];
```

The elements in the first cluster (outliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S1).

```
XYZOut = Map[dataP[[#]]&, Position[S1, 1]//Flatten];
```

The number of these elements (outliers) is 1577.

Similarly, the elements in the second cluster (inliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S2).

```
XYZIn = Map[dataP[[#]]&, Position[S2, 1]//Flatten];
```

and there are 14857 inliers.

Let us display the ouliers and the inliers (Fig. 13.46).

Since the two distributions are different, we need to carry out a further iteration steps. The results after three iteration steps can be seen on Fig. 13.47 and in Tables 13.5 and 13.6.

13.2.2.9 Application to Leafy Tree

In the example above there were only $\sim$10% outliers. Now let us see another example where the ratio of the outliers are much more higher, nearly 40%. This situation is closer to segmentation than to a simple fitting problem. Here the test object is the same tree, but with foliage, see Fig. 13.48.

Let us load the data, 91,089 points, and show their plot.

The first guess will be the result, which we obtained without foliage (Figs. 13.49, 13.50).

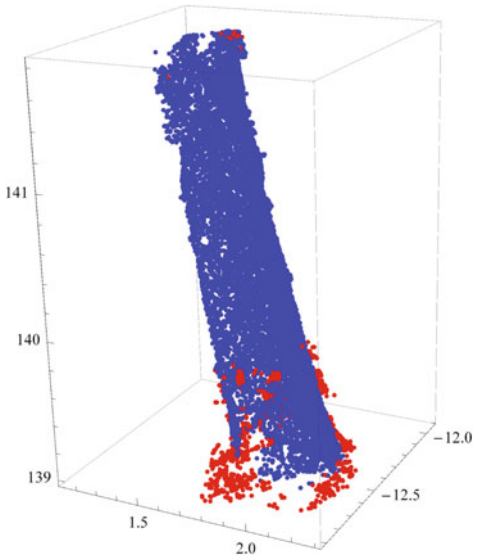


Fig. 13.46 The inliers (blue) and outliers (red) as a first guess resulted standard minimization of the total residual

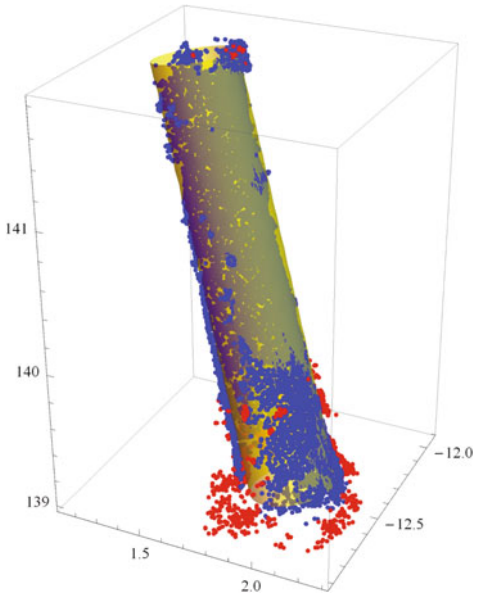


Fig. 13.47 The inliers (blue) and outliers (red) as a first guess resulted by the maximum likelihood technique

$$\{a \to -0.6619, b \to -11.2767, c \to -3.57789, d \to 146.344, r \to 0.175\}$$

Applying the geometric error model, the model error distribution of this first approach (Fig. 13.51).

Table 13.5 The computed cylinder parameters

Iteration	a	b	c	d	r
0	−0.6043	−11.3773	−3.6870	146.479	0.1788
1	−0.6438	−11.3064	−3.6080	146.386	0.1752
2	−0.6556	−11.2872	−3.5783	146.343	0.1749
3	−0.6619	−11.2767	−3.5779	146.344	0.1750

Table 13.6 Parameters of the components of the Gaussian mixture

Iteration	μ_1	μ_2	s_1	s_2	$?_1$	$?_2$	N_1	N_2
0	0.0813	−0.010	0.0536	0.0227	0.1037	0.8962	1318	15116
1	0.0826	−0.011	0.0597	0.0183	0.1188	0.8811	1577	14857
2	0.0814	−0.0120	.0631	0.0176	0.1250	0.8749	1641	14793
3	0.0837	−0.012	0.0627	0.0176	0.1229	0.8771	1637	14797

Fig. 13.48 The inliers (blue) and outliers (red) as a first guess resulted standard minimization of the total residual

Using all of the points we have:
Now let us identify the components of the Gaussian mixture.

```
{Δ,param}=ExpectationMaximization[error,100,
{{0.6,0.15},{0,0.1},{0.5,0.5}}];
```

The result of the parameter values is

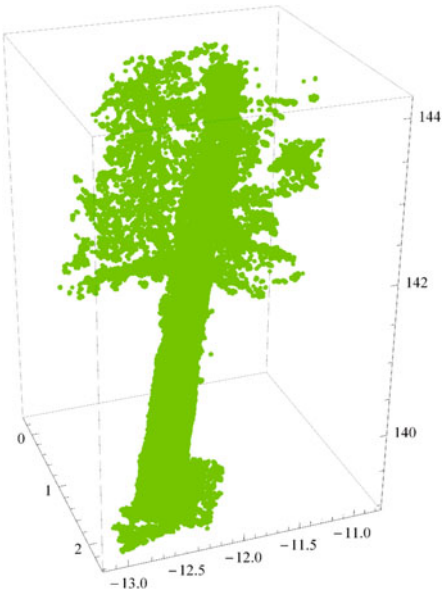


Fig. 13.49 The points of cloud of data

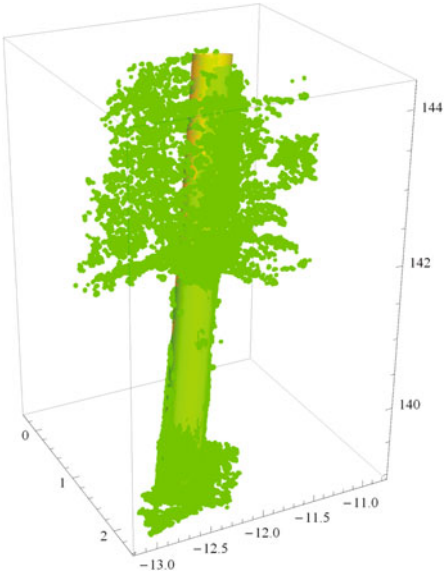


Fig. 13.50 The first estimation of the cylinder fitting to leafy tree

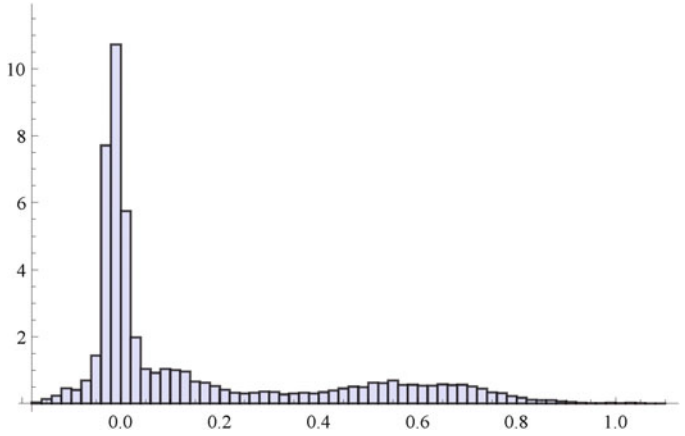


Fig. 13.51 The model error distribution of the first estimation

```
{{0.341165,0.275897},{-0.0109171,0.0204849},{0.463995,0.536005}}
```

These can be displayed in a table form (Table 13.7; Figs. 13.52, 13.53).The density functions of the two component: (Figs. 13.54, 13.55, 13.56, 13.57)

Table 13.7 Parameters of the Gaussian mixture after zero iteration

η	μ	s
0.463995	0.341165	0.275897
0.536005	−0.0109171	0.0204849

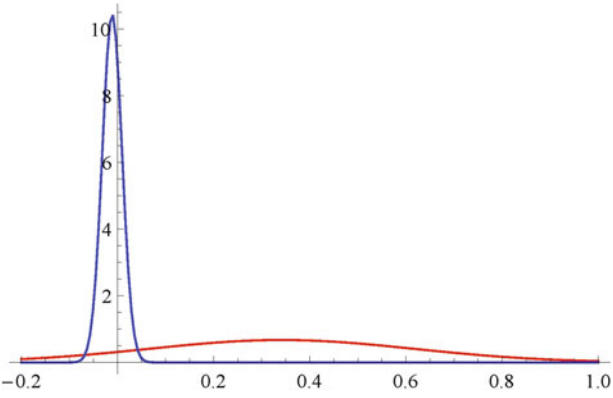


Fig. 13.52 The PDF of the two components

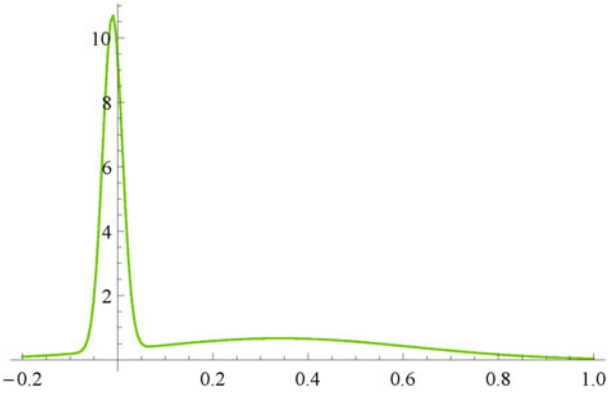


Fig. 13.53 The joint PDF of the mixture

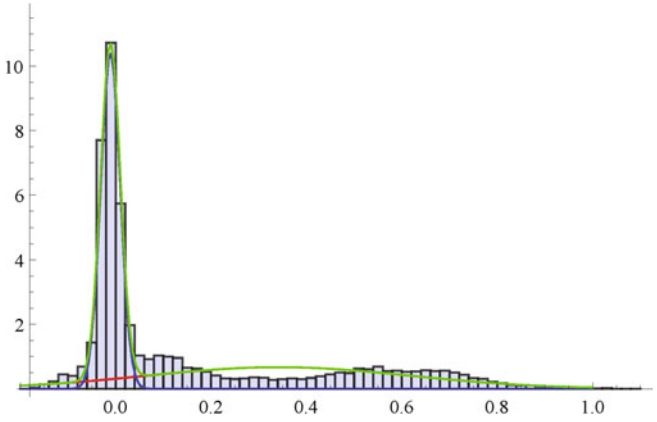


Fig. 13.54 The PDF of the two components and the normalized histogram of the data

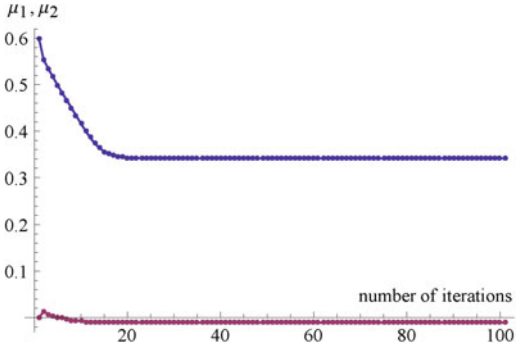


Fig. 13.55 The convergence of the means of the two components

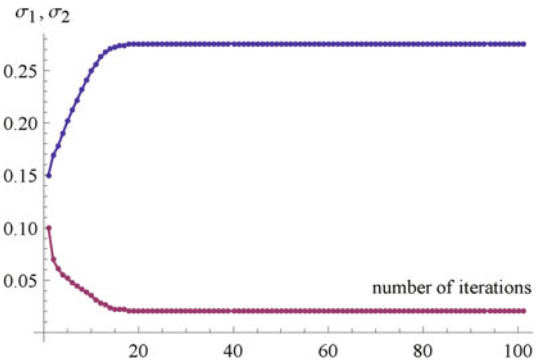


Fig. 13.56 The convergence of the standard deviations

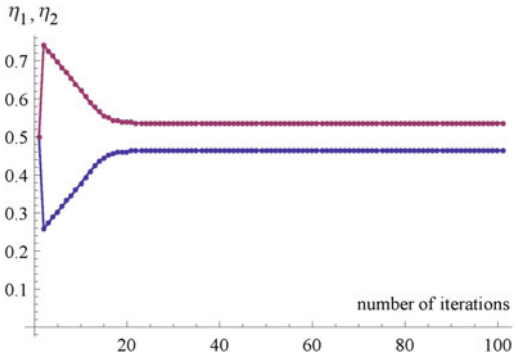


Fig. 13.57 The convergence of the membership weights

```
Show[{p5,p3,p4}]
```

Now, we separate the mixture of the samples into two clusters: cluster of outliers and cluster of inliers. Membership values of the first cluster

```
Short[Δ[[1]],10]
{1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,≪91009≫,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.}
```

Membership values of the second cluster:

```
Short[Δ[[2]],10]
{2.168186518114028 × 10^-569, 5.715327153955006 × 10^-563,
 4.439837476761972 × 10^-375, 6.094575596255934 × 10^-374,
 1.526341682623958 × 10^-585, 2.869284522501804 × 10^-372,
 ≪91078≫, 9.51406 × 10^-40, 6.5817 × 10^-32,
 8.02675 × 10^-35, 1.28721 × 10^-30, 1.99815 × 10^-38}
```

In order to get Boolean (crisp) clustering let us round the membership values.

```
S1 = Round[Δ[[1]]]; S2 = Round[Δ[[2]]];
```

The elements in the first cluster (outliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S1). We get 39,982 outliers.

Similarly, the elements in the second cluster (inliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S2). There are 51,107 (Figs. 13.58, 13.59).

Let us display the ouliers and the inliers:

Now let us compute the first iteration employing maximum likelihood method. The parameters become:

```
{a → -0.660978, b → -11.28, c → -3.60862, d → 146.398, r → 0.17553}
```

Now the parameters of the Gaussian mixture:

The result of the parameter values:

```
{{0.343552, 0.275066}, {-0.0109826, 0.0207809}, {0.458853, 0.541147}}
```

These can be displayed in a table form Table (13.8); (Figs. 13.60, 13.61, 13.62, 13.63, 13.64, 13.65).

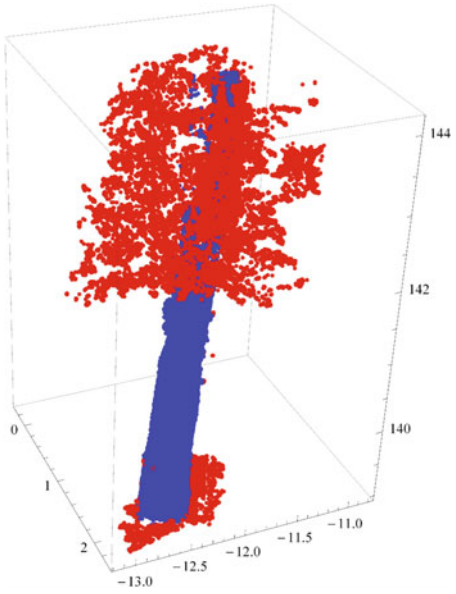


Fig. 13.58 The inliers (blue) and outliers (red) as a first guess resulted by the first approach

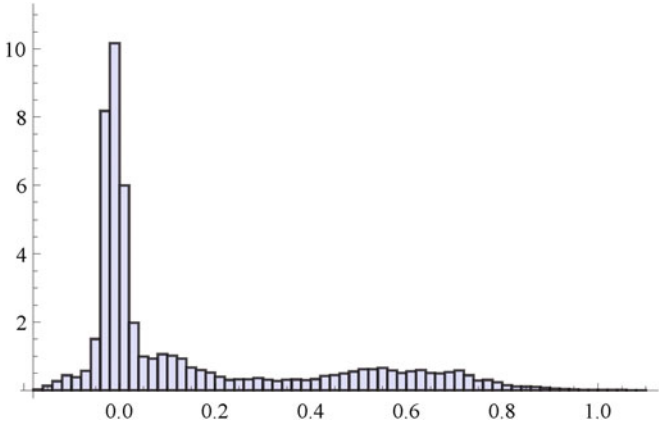


Fig. 13.59 The PDF of the two components and the normalized histogram of the data

Table 13.8 Parameters of the Gaussian mixture after zero iteration

η	μ	s
0.458853	0.343552	0.275066
0.541147	−0.0109826	0.0207809

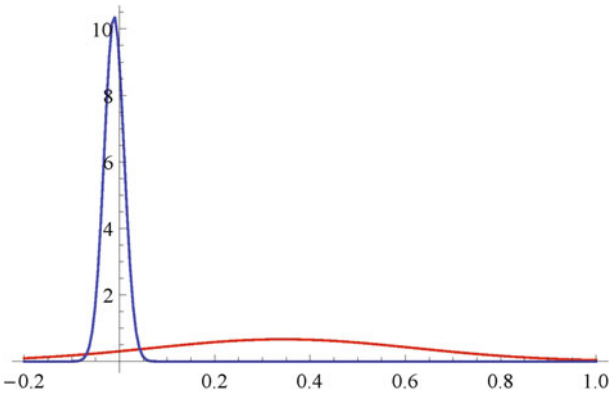


Fig. 13.60 The PDF of the two components

The density functions of the two components:
We separate the mixture of the samples into two clusters: cluster of outliers and cluster of inliers. Membership values of the first cluster:

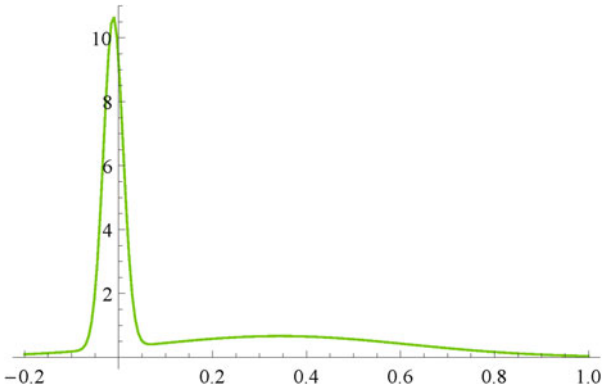


Fig. 13.61 The joint PDF of the mixture

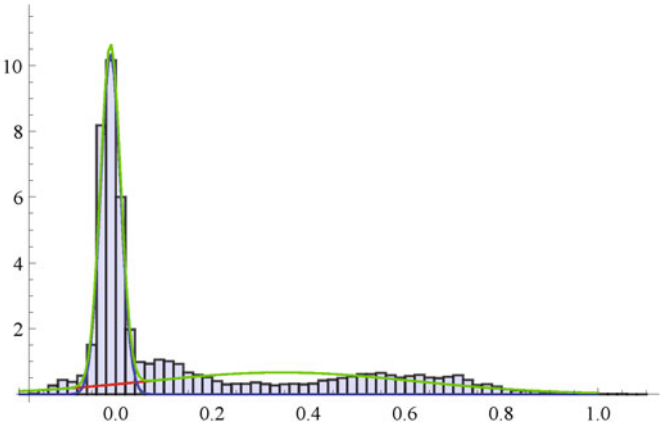


Fig. 13.62 The PDF of the two components and the normalized histogram of the data

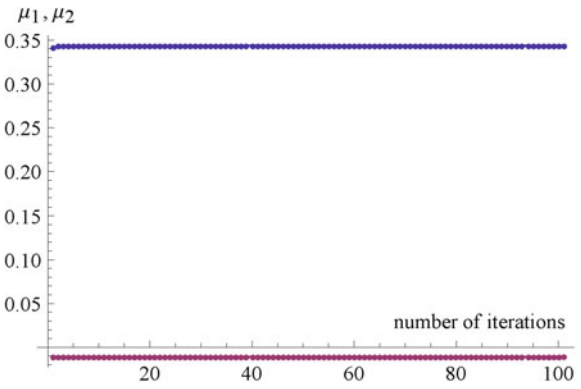


Fig. 13.63 The convergence of the means of the two components

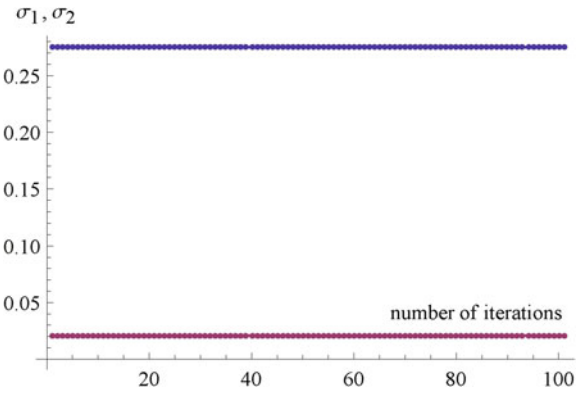


Fig. 13.64 The convergence of the standard deviations

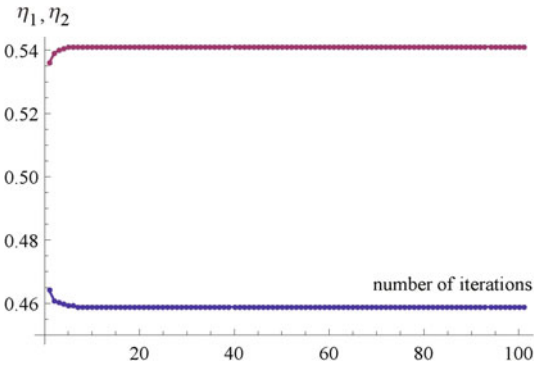


Fig. 13.65 The convergence of the membership weights

```
Short[Δ[[1]],10]
{1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,≪91009≫,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,
 1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,}
```

Membership values of the second cluster:

```
Short[Δ[[2]],10]
{5.197294751685767 × 10^-563,9.62586576863179 × 10^-557,
 2.44052537654221 × 10^-371,3.138647064118679 × 10^-370,
 7.696701760704474 × 10^-579,1.319339597606332 × 10^-368,
 ≪91077≫,5.26593 × 10^-38,1.12943 × 10^-38,4.7876 × 10^-31,
 6.62412 × 10^-34,7.8142 × 10^-30,1.95385 × 10^-37}
```

In order to get Boolean (crisp) clustering let us round the membership values.

```
S1 = Round[Δ[[1]]]; S2 = Round[Δ[[2]]];
```

The elements in the first cluster (outliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S1). There are 39404 outliers.

Similarly, the elements in the second cluster (inliers) are the corresponding elements of those having value 1 in the set containing the rounded member values (S2). There are 51685 inliers (Fig. 13.66).

Let us display the outliers and the inliers (Fig. 13.67).

The inlier points,

We consider it the last iteration step. The parametric equation of the fitted cylinder is (Figs. 13.68, 13.69).

```
{v + 0.168963 Cos[u] + 0.00813893 Sin[u], - 11.28 - 0.660978 v +
 0.172661 Sin[u], 146.398 - 3.60862 v + 0.0468219 Cos[u] -
 0.0293703 Sin[u]}
```

```
Show[{p1,p2,pIn,pOut}]
```

Tables (13.9, 13.10)

The results our computation illustrate that in case of noisy measurements, the frequently assumed $\mathcal{N}(0, \sigma)$ model error distribution is not true. Therefore to estimate the model parameters, iteration is necessary with the proper representation of the

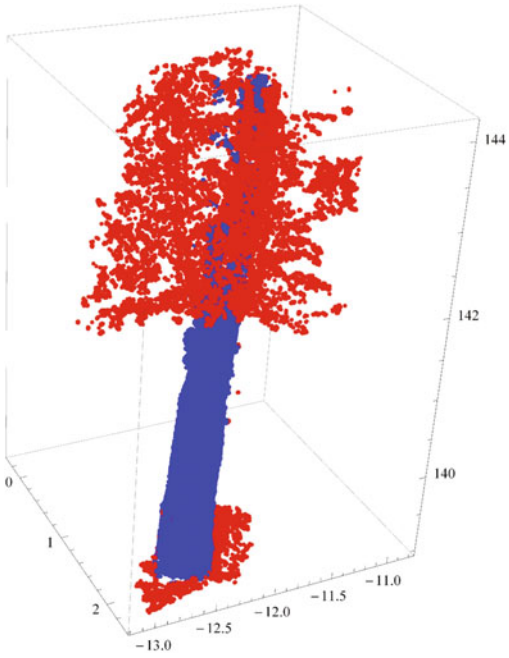


Fig. 13.66 The inliers (blue) and outliers (red) as a first guess resulted by the first iteration

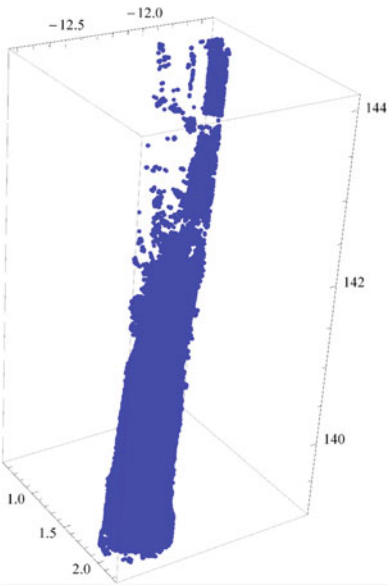


Fig. 13.67 The inliers

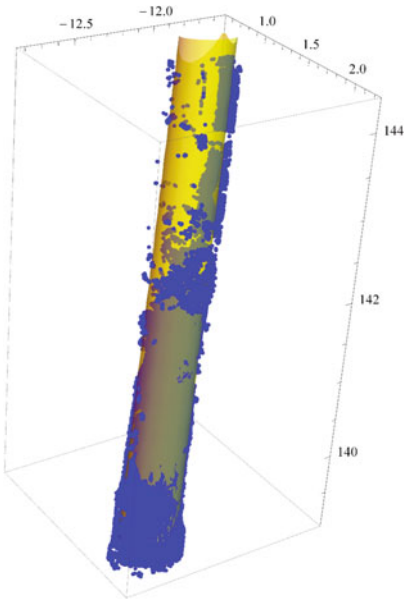


Fig. 13.68 The inliers (blue) and the fitted cylinder

non-Gaussian distribution. In this study an expectation maximization algorithm was employed. In case of general cylinder geometry, the five model parameters can be computed via local maximization of the maximum likelihood function. The implicit equation based on an algebraic error definition, and solved easily by Gröbner basis, can provide a proper initial guess value for the maximization problem.

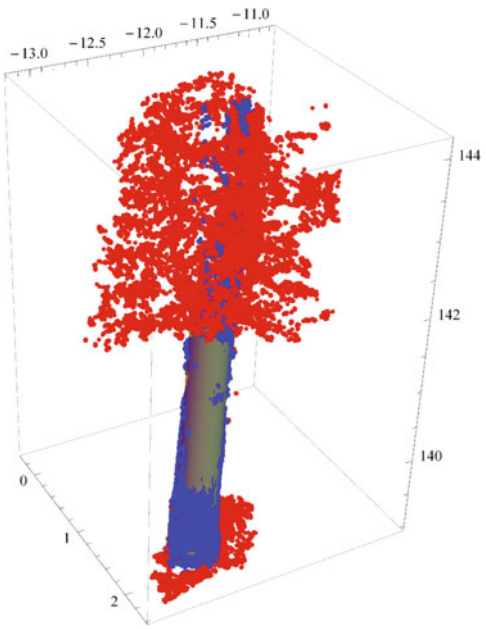


Fig. 13.69 The inliers (blue) and outliers (red) and the fitted cylinder

Table 13.9 The computed cylinder parameters

Iteration	a	b	c	d	r
0	−0.6619	−11.2767	3.5779	146.344	0.1750
1	−0.6611	−11.2791	3.6086	146.398	0.1753

Table 13.10 Parameters of the components of the Gaussian mixture

Iteration	μ_1	μ_2	s_1	s_2	$?_1$	$?_2$	N_1	N_2
0	0.3412	−0.011	0.2759	0.0205	0.4640	0.5360	39982	51107
1	0.3435	−0.010	0.2751	0.0208	0.4588	0.5418	39404	51685

13.3 Problem

13.3.1 Fitting a Plane to a Slope

13.3.1.1 Test Area, Equipment and Measured Data

Outdoor laser scanning measurements have been carried out in a hilly park of Budapest, see Fig. 13.70 The test area is on a steep slope, covered with dense but low vegetation.

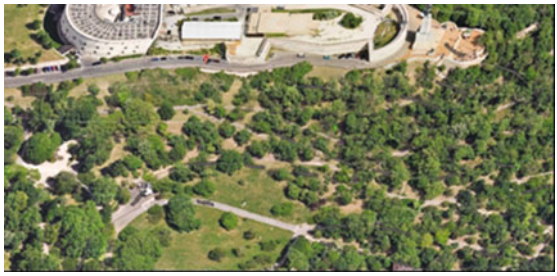


Fig. 13.70 The test area in Budapest

Fig. 13.71 The scanner at the top of the measured steep slope with the different sizes of white spheres as control points in the background

The experiment has been carried out with a Faro Focus 3D terrestrial laser scanner, as before. The test also aimed to investigate the tie point detection capabilities of the scanner's processing software; different types of spheres were deployed all over the test area. In case of multiple scanning positions these spheres can be used for registering the point clouds.

The measurement range of the scanner is 120 m, the ranging error is ±2 mm, according to the manufacturer's technical specification (Fig. 13.71).

The scanning parameters were set to " resolution, which equals 3 mm/10 m point spacing. This measurement resulted in 178.8 million points that were acquired in five and half minutes. The test data set was cropped from the point cloud; moreover, further resampling was applied in order to reduce the data size. The final data set is composed of 38,318 points in ASCII format, and only the x, y, z coordinates were kept (no intensity values). Let us load the measured data: (Figs. 13.72, 13.73).

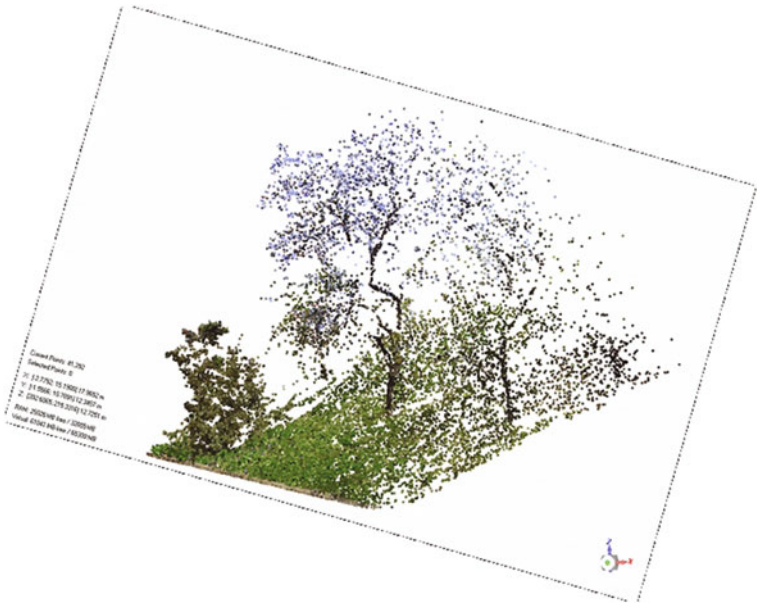
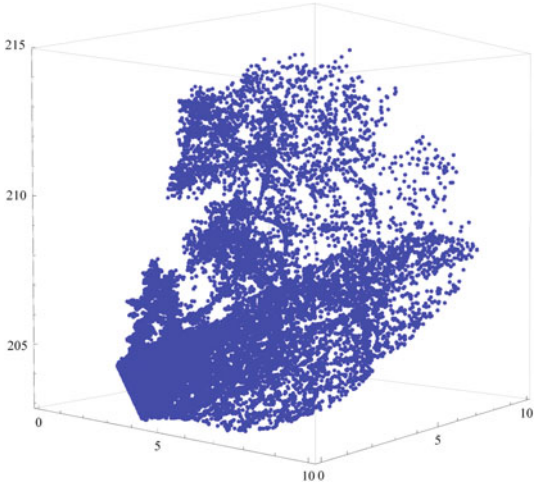


Fig. 13.72 The measured colored LiDAR point cloud

Fig. 13.73 The measured LiDAR point cloud




```
Clear[XYZ,N]
XYZ = Import["H:\\Tamas_adatok//output_41_392.dat"];
n = Length[XYZ]
41392
```

```
XYZP = Select[XYZ, And[NumberQ[#[[1]]], NumberQ[#[[2]]],
 NumberQ[#[[3]]]]&];
N = Length[XYZ]
33292
```

```
p8 = ListPointPlot3D[XYZP, PlotStyle → Blue, BoxRatios → {1, 1, 1},
 ViewPoint → {1.8698922647815255‘, - 2.32400793103699‘,
 - 0.1931120274143106‘}, ViewVertical → {0.47714644303899606‘,
 - 0.5674527451525903‘, 0.6710653127036321‘}]
```

13.3.1.2 Application SVD

Skipping the details, after 9.1 s SVD finds that the equation for the plane is:

```
{z → 198.698 + 0.495306x + 1.68771y}
```

13.3.1.3 Solution via PCA

```
AbsoluteTiming[A = Covariance[Cc0]; MatrixForm[A]]
```

$$\left\{0.0376042,\begin{pmatrix} 5.87736 & -0.0359188 & 1.96269 \\ -0.0359188 & 2.86103 & 1.78577 \\ 1.96269 & 1.78577 & 5.77838 \end{pmatrix}\right\}$$

```
{nx, ny, nz} = Eigenvectors[A][[3]]
{ - 0.244802, - 0.834143, 0.494245}

sold = Solve[{nx, ny, nz}.Total[XYZ] - dN == 0, d]
{d → 98.2054}
```

Consequently the coefficients of surface model are

```
{0.495306, 1.68771, 198.698}
```

13.3.1.4 Gröbner Basis Solution

$\{\alpha \to 0.49530, \beta \to 1.68771, \gamma \to 198.698\}$

See Figure (13.74).

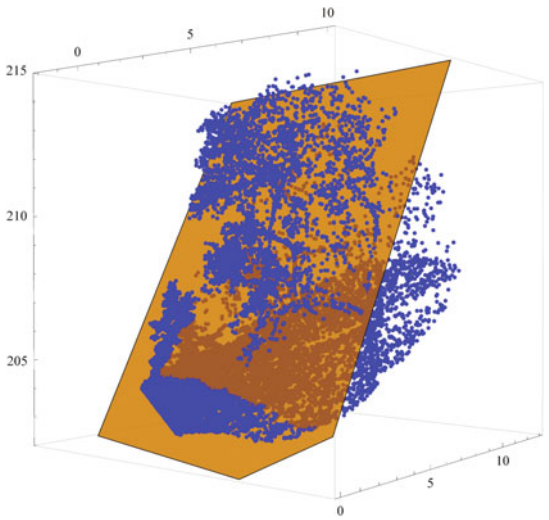


Fig. 13.74 The measured LiDAR point cloud

13.3.1.5 Model Error Analysis

```
Error = Map[(#[[3]] - α#[[1]] - β#[[2]] - γ)/Sqrt[1 + α^2 + β^2]) /.solopt&, XYZ];
Min[Error]
-5.06393
Max[Error]
5.46962
Mean[Error]
-1.27578 × 10^-14
StandardDeviation[Error]
1.3388
Histogram[Error, 40]
```

See Figure (13.75).

13.3.1.6 Application Danish Algorithm with Embedded Gröbner Basis Solution

We will use 20 iterations with 33292 points.

The initial weights of the observations

```
W = Table[1, {N}]; (*initialization of weights *)
```

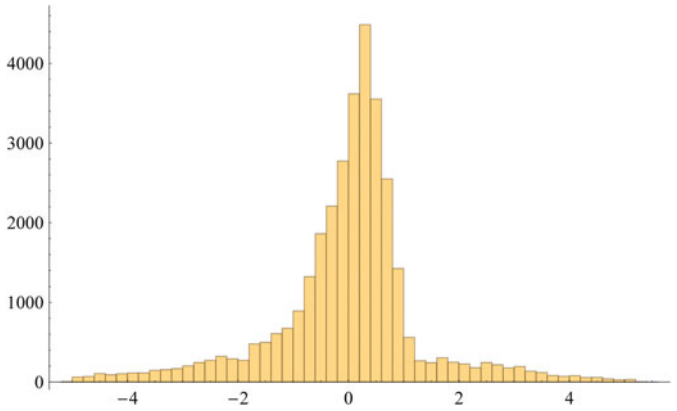


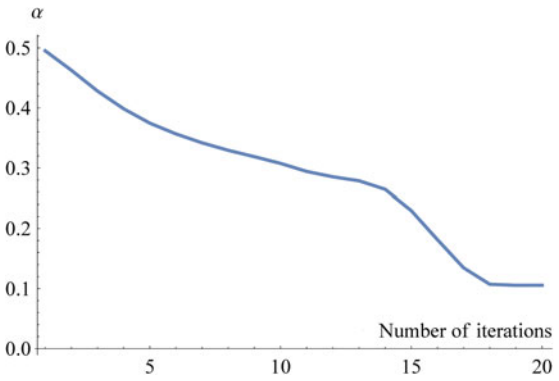
Fig. 13.75 The histogram of the errors

The next figures show the convergence of the different parameters as a function of the number of the iterations (Figs. 13.76, 13.77, 13.78).
The result of the iteration

$$\{\alpha \to 0.105602, \beta \to 0.504716, \gamma \to 202.664\}$$

The observations that remained active (inliers) are total 24,576.We plot them:
The inliers of the Danish solution (Figs. 13.79, 13.80).

Fig. 13.76 The convergence of the parameter α



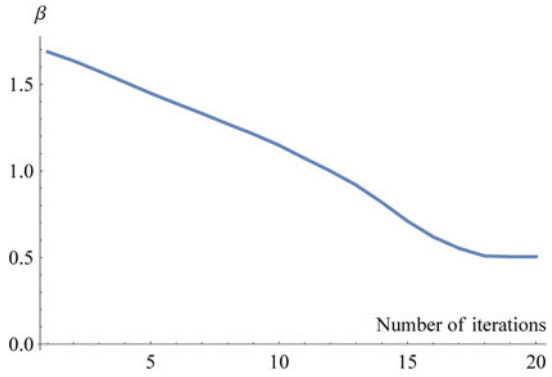


Fig. 13.77 The convergence of the parameter β

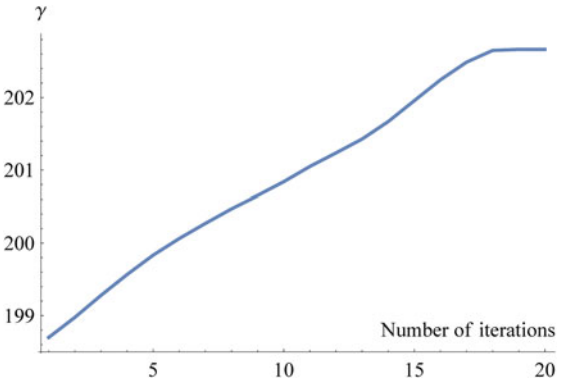


Fig. 13.78 The convergence of the parameter γ

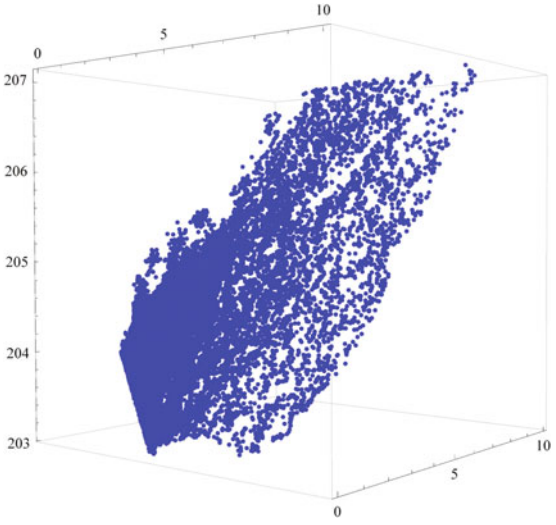


Fig. 13.79 The inliers of the Danish solution

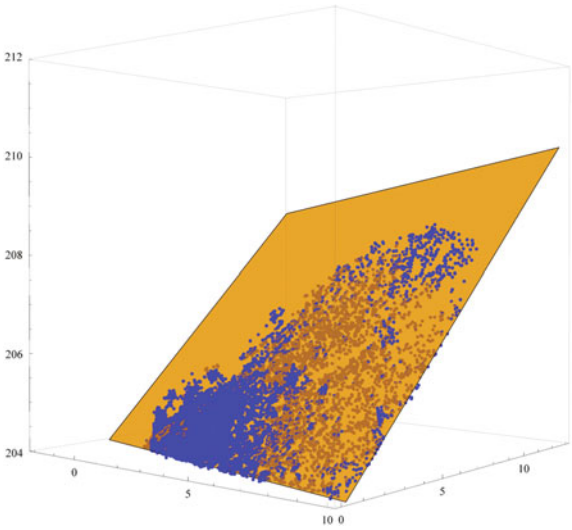


Fig. 13.80 The original plane and the inliers of the Danish solution

13.3.1.7 Model Error Analysis

$$\texttt{Error} = \texttt{Map}\left[\left(\frac{\#[[3]] - \alpha\#[[1]] - \beta\#[[2]] - \gamma}{\sqrt{1 + \alpha^2 + \beta^2}}\right)/.\texttt{sols\&}, \texttt{inliers}\right];$$

```
Min[Error]
-0.219962
Max[Error]
0.370434
Mean[Error]
-9.44382 × 10^-15
StandardDeviation[Error]
0.0703037
pHT = Histogram[Error, 20]
```

See Figure (13.81).

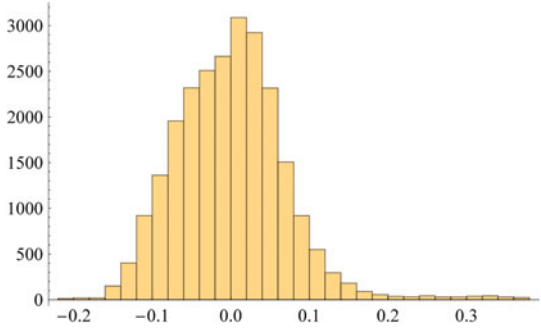


Fig. 13.81 The histogram of the errors

13.3.1.8 Application Danish Method with Embedded Weighted PCA (PCAW)

```
N = 50; (*Number of iterations*)
```

The next figures show the convergence of the different parameters as a function of the number of the iterations (Figs. 13.82, 13.83, 13.84).
The result of the iteration:

$$\{\alpha \to 0.102968, \beta \to 0.566614, \gamma \to 202.536\}$$

The observations that remained active (inliers) are total 26089 and they look like: (Fig. 13.85)
The plane that results from the Danish algorithm: (Fig. 13.86)

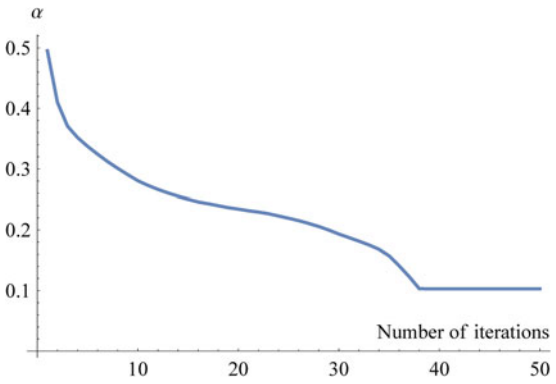


Fig. 13.82 The convergence of the parameter α

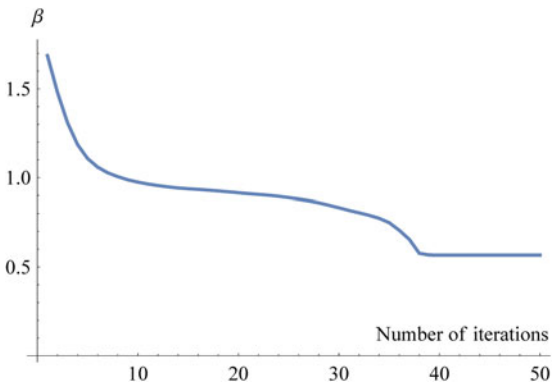


Fig. 13.83 The convergence of the parameter β

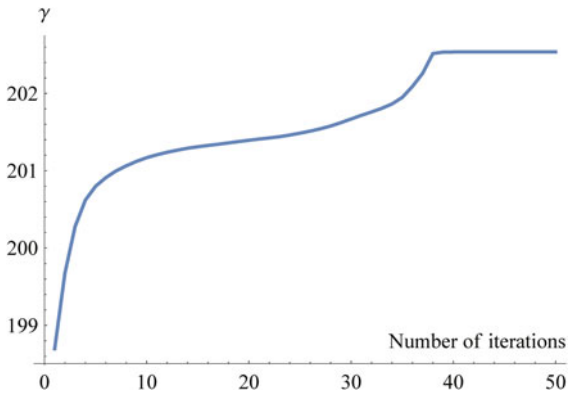


Fig. 13.84 The convergence of the parameter γ

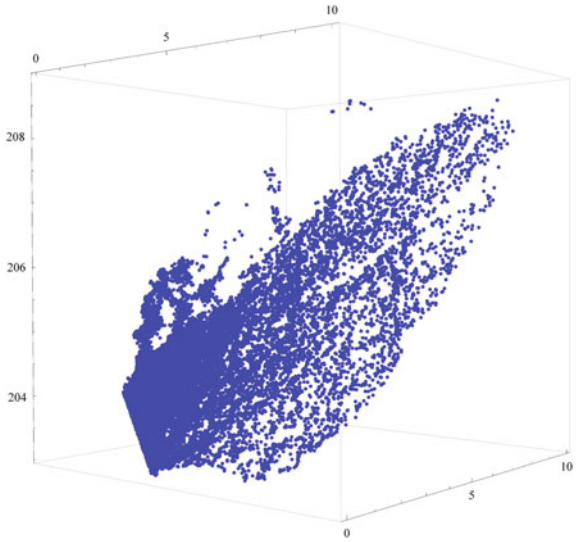


Fig. 13.85 The inliers of the Danish solution

13.3.1.9 Model Error Analysis

```
Error = Map[(#[[3]] - α#[[1]] - β#[[2]] - γ)/√(1 + α^2 + β^2) /.sols&, inliers];
Min[Error]
-0.460056
Max[Error]
0.945997
Mean[Error]
-1.68568 × 10^-14
StandardDeviation[Error]
0.185912
pHS = Histogram[Error, 20]
```

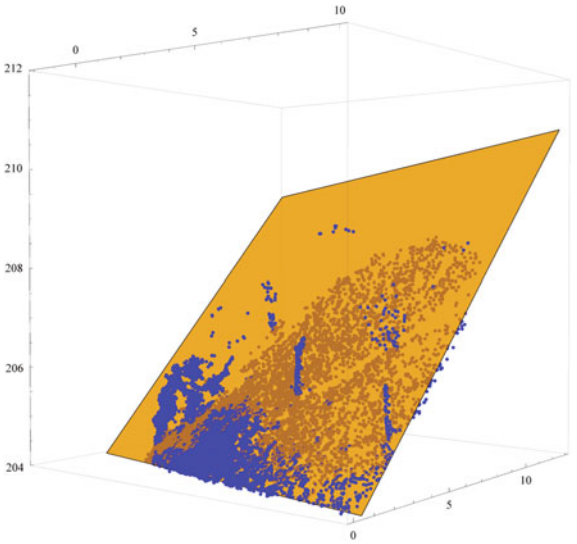


Fig. 13.86 The original plane and the inliers of the Danish solution

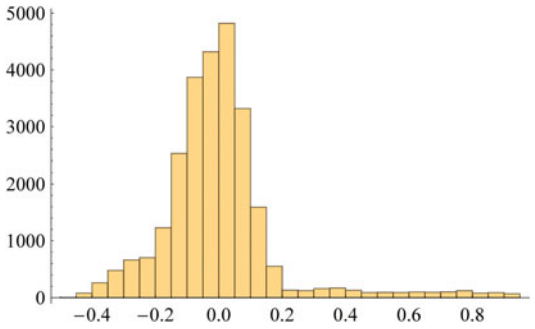


Fig. 13.87 The histogram of the errors in case of PCAW method

See Figure (13.87).

In order to compare the solution without and with the Danish robust algorithm, let us consider the inliers selected by the Danish algorithm for the error computation of the result given *without robust estimation*,

$$\texttt{Error} = \texttt{Map}\left[\left(\frac{\#[[3]] - \alpha\#[[1]] - \beta\#[[2]] - \gamma}{\sqrt{1 + \alpha^2 + \beta^2}}\right)/.\texttt{solopt\&}, \texttt{inliers}\right];$$

```
Min[Error]
-5.06393
Max[Error]
1.1246
Mean[Error]
-0.32325
```

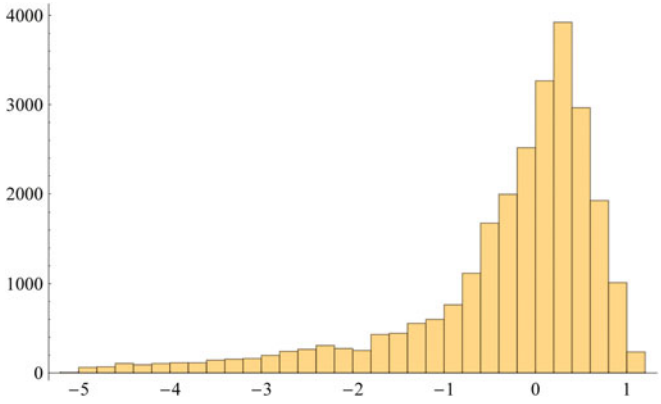


Fig. 13.88 The histogram of the errors in case of non-rubust method

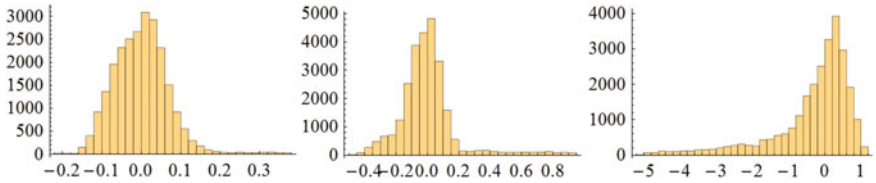


Fig. 13.89 Distributions of the model error in the 3 different cases: **a** Danish—Gröbner basis—Maximum likelihood, **b** Danish—PCAW, **c** Non-robust estimation

```
StandardDeviation[Error]
0.10649
pHW = Histogram[Error, 20]
```

See Figure (13.88).
The 3 histograms are (Fig. 13.89)

```
GraphicsGrid[{{pHT, pHS, pHW}}]
```

In this section a new algorithm for plane fitting to large cloud of data set was presented.

References

Beder C, Förstner W (2006) Direct solutions for computing cylinders from minimal sets of 3d points, computer vision-ECCV 2006. Lect Notes Comput Sci 3951:135–146

Carrea D, Jaboyedoff M, Derron MH (2014) Feasibility study of point cloud data from test deposition holes for deformation analysis. In: Working report 2014–01, Universite de Lausanne (UNIL)

DalleMole V, do Rego R, Araújo A (2010) The self-organizing approach for surface reconstruction from unstructured point clouds. In: Matsopoulos GK (Ed) Self-Organizing Maps
Draelos MT (2012) The kinect up close: modifications for short-range depth imaging. In: A thesis Master of Science, Electrical Engineering, Raleigh, North Carolina
Fischler MA, Bolles RC (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 24(6):381–395
Fox A, Smith J, Ford R, Doe J (2013) Expectation Maximization for gaussian mixture distributions. In: Wolfram Demonstration Project
Franaszek M, Cheok G, Saidi KS, Witzgall C (2009) Fitting spheres to range data from 3-D imaging systems. IEEE T. Instru Meas 58(10):3544–3553
Khameneh M (2013) Tree detection and species identification using LiDAR data. In: MSc. Thesis, KTH, Royal Institute of Technology, Stockholm
Kohonen T (1998) The self-organizing map. Neurocomputing 21:1–6
Krarup T, Kubik K, Juhl J (1980) Götterdammerung over least squares. In: Proceeding of international society of photogrammetry 14th Congress, Hamburg, pp 370–378
Lichtblau D (2006) Cylinders through five points: complex and real enumerative geometry, Conference Paper August 2006, DOI:10.1007/978-3-540-77356-6_6 Source: DBLP conference: automated deduction in geometry, 6th international workshop, ADG 2006, Pontevedra, Spain, August 31-September 2, 2006
Lichtblau D (2012) Cylinders Through five points computational algebra and geometry automated deduction in geometry. J. Mathe Res 4:65–82 Published by Canadian Center of Science and Education
Lukacs G, Martin R, Marshall D (1998) Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In: Burkhardt H, Neumann B (eds) Computer Vision - ECCV'98, vol I. LNCS 1406, Springer-Verlag, pp 671–686
Mitra NJ, Nguyen (2003) SoCG'03, June 8–10, San Diego, California, USA, ACM 1-58113-663-3/03/0006, pp 322–328
Molnár B, Toth CK, Detrekoi A (2012) Accuracy test of microsoft kinect for human morphological measurements. In: International archives of the photogrammetry, remote sensing and spatial information sciences, vol XXXIX-B3, 2012 XXII ISPRS Congress, 25 August—01 September 2012, Melbourne, Australia
Nurunnabi A, Belton D, West G. (2012) Diagnostic—robust statistical analysis for local surface fitting in 3D point cloud data. In: ISPRS annals of the photogrammetry, remote sensing and spatial information sciences, vol. I-3, 2012 XXII ISPRS Congress, 25 Aug. 2012, Melbourne, Australia, pp 269–275
Paláncz B (2014) Fitting data with different error models. Mathe J 16:1–22
Petitjean S (2002) A survey of methods for recovering quadrics in triangle meshes. ACM Comput Surv 34(2):211–262
Poppinga J, Vaskevicius N, Birk A, Pathak K (2008) Fast plane detection and polygonalization in noisy 3D range images. In: International conference on intelligent robots and systems (IROS), Nice, France, IEEE Press 2008
Rose C, Smith D (2000) Symbolic Maximum Likelihood Estimation with Mathematica. Statistician 49:229–240
Ruiz O, Arroyave S, Acosta D (2013) Fitting Analytic Surfaces to Noisy Point Clouds. Am. J. Comput Mathe 3:18–26
Stal C, Timothy Nuttens T, Constales D, Schotte K, De Backer H, De Wulf A (2012) Automatic filtering of terrestrial laser scanner data from cylindrical tunnels, TS09D—Laser Scanners III, 5812. In: FIG Working Week 2012, Knowing to manage the territory, protect the environment, evaluate the cultural heritage, Rome, Italy, May 2012, pp 6–10
Stathas D, Arabatzi O, Dogouris S, Piniotis G, Tsini D, Tsinis D (2003) New monitoring techniques on the determination of structure deformation. In: Proceedings 11th international fig symposium on deformation measurements, Santorini, Greece
Su YT, Bethel J (2010) Detection and robust estimation of cylinder features in point clouds. In: ASPRS 2010 annual conference, San Diego, California April 26–30

Vosselman G, Gorte B, Sithole G, Rabbani T (2004) Recognizing structure in laser scanner point clouds. In Int. Arch Photogrammetry Remote Sensing Spatial Inf Sci 46:33–38

Winkelbach S, Westphal R, Goesling T (2003) Pose estimation of cylinder fragments for semi-automatic bone fracture reduction. In: Pattern Recognition (DAGM 2003). Michaelis B and Krell G eds. Lecture Notes in Computer Science, vol 2781, Springer-Verlag, pp 566–573

Yang MY, Förtsner W (2010) Plane detection in point cloud data, TR-IGG-P-2010-01. In: Technical report. Nr.1, department of photogrammetry institute of geodesy and geo-information, Uni., Bonn, Germany

Zuliani M (2012) RANSAC for dummies, www.vision.ece.ucsb.edu/∼zuliani

Chapter 14
Stochastic Modeling

14.1 Basic Stochastic Processes

14.1.1 Concept of Stochastic Processes

A *stochastic* or *random process* is a mathematical object usually defined as a collection of random variables. Historically, the random variables were associated with or indexed by a set of numbers, usually viewed as points in time, giving the interpretation of a stochastic process representing numerical values of some system randomly changing over time. From this point of view a random process can be discrete or continuous in time. Consequently stochastic processes are widely used as mathematical models of systems and phenomena that appear to vary in a random manner.

The term *random function* is also used to refer to a stochastic or random process, because a stochastic process can also be interpreted as a random element in a function space.

14.1.2 Examples for Stochastic Processes

Bernoulli process

One of the simplest stochastic processes is the *Bernoulli process*, which is a sequence of independent and identically distributed random variables, where each random variable takes either the value one with probability, say, p and value zero with probability $1 - p$. This process can be likened to somebody flipping a coin, where the probability of obtaining a head is p and its value is one, while the value of a tail is zero. In other words, a Bernoulli process is a sequence of Bernoulli random variables, where each coin flip is a Bernoulli trial. Using a *Mathematica* function for $p = 0.25$ could result in:

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_14

Using built in function

```
data = RandomFunction[BernoulliProcess[0.25], {0, 100}]
```

If we plot this from time 0 to time 100 we could get (Fig. 14.1).

Random walk

Random walks are stochastic processes that are usually defined as sums of random variables or random vectors in Euclidean space, so they are processes that change in discrete time. But some also use the term to refer to processes that change in continuous time.

A classic example of a random walk is known as the simple random walk, which is a stochastic process in discrete time with the integers as the state space, and is based on a Bernoulli process, where each Bernoulli variable takes either the value positive one or negative one. In other words, the simple random walk takes place on the integers, and its value increases by one with probability, say, p, or decreases by one with probability $1 - p$, so the index set of this random walk is the natural numbers, while its state space is the integers. If the $p = 0.5$, this random walk is called a symmetric random walk. Here is a graph with time 0 to 100 (Fig. 14.2).

A random process realization results a single trajectory. However, to get statistics of the process at single time stamp, one needs many realizations. Here are four (Fig. 14.3).

Wiener process

The Wiener process is a stochastic process with stationary and independent increments that are normally distributed based on the size of the increments. The Wiener process is named after Norbert Wiener, who proved its mathematical existence, but the process is also called the Brownian motion process or just Brownian motion due to its historical connection as a model for Brownian movement in liquids. It can be considered a continuous version of the simple random

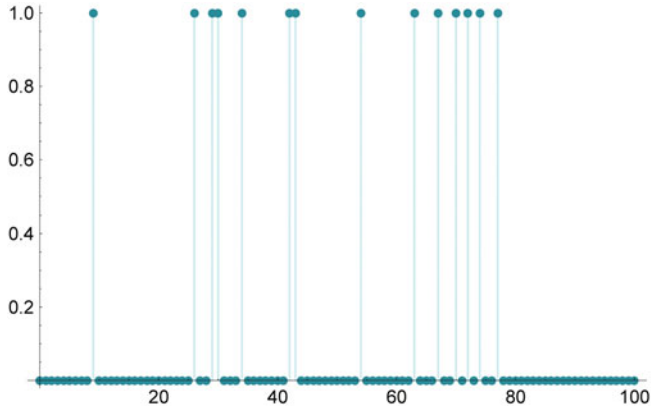


Fig. 14.1 Bernoulli process

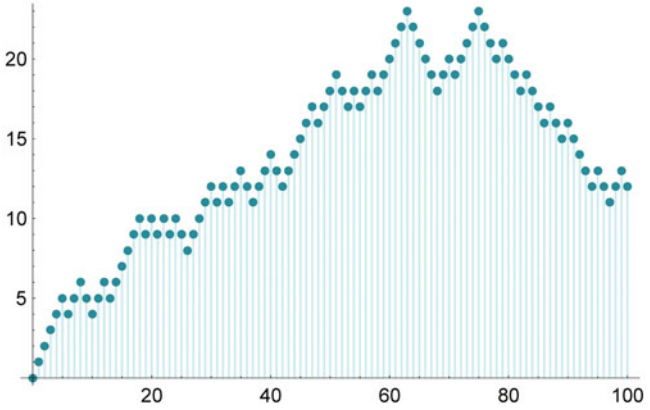


Fig. 14.2 Random walk process

walk. It plays a central role in the stochastic calculus of the stochastic differential equations (Fig. 14.4).

```
data1 = RandomFunction[WienerProcess[.8, .5], {0, 1, 0.01}];

data2 = RandomFunction[WienerProcess[0, .5], {0, 1, 0.01}];
```

14.1.3 Features of Stochastic Processes

Slice distribution
This is the distribution of the random process values at a fix time stamp.

Simulate 500 paths from a Wiener process and plot paths and histogram distribution of the slice distribution at $t = 1$ (Fig. 14.5).

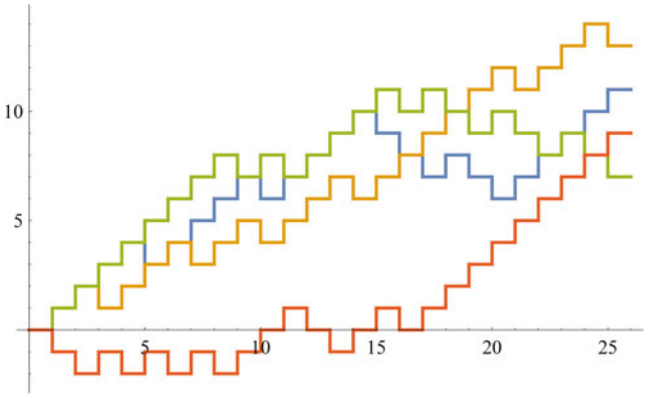


Fig. 14.3 Random walk process in two dimensions with four realizations

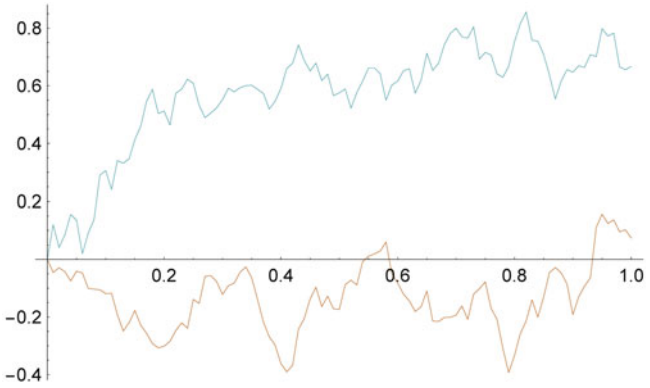


Fig. 14.4 Wiener process with drift and without drift four realizations

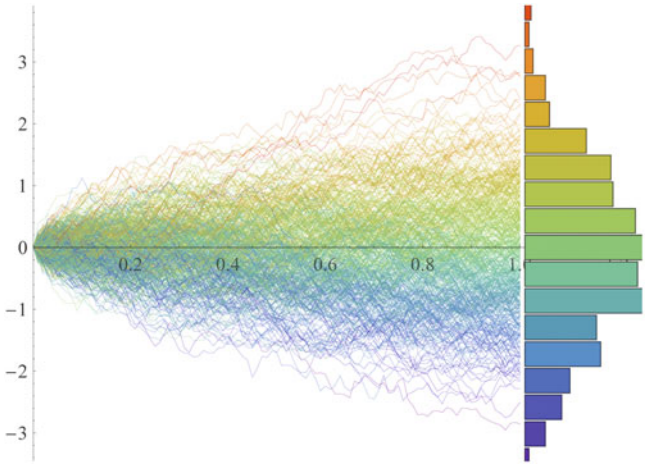


Fig. 14.5 Simulate 500 paths from a Wiener process and plot paths and histogram distribution of the slice distribution at $t = 1$

Mean value function:
Mean value of a slice distribution of the random process

Variance function:
Variance of a slice distribution of the random process

Stationarity:
Stationary distribution is also known as steady-state distribution, its slice distribution is independent on the time.

Weak stationary:
A random process is weakly stationary if its mean function is independent of time, and its covariance function is independent of time translation.

14.2 Time Series

14.2.1 Concept of Time Series

A time series is *a series of data points* indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data. Examples of time series are heights of ocean tides, counts of sunspots.

Time series analysis comprises methods for *analyzing time series* data in order to extract meaningful statistics and other characteristics of the data. *Time series forecasting* is the use of a model to predict future values based on previously observed values.

For example here is the average temperature on the first day of a month in Chicago, IL. We have 133 data points from 2001 to 2012 (Fig. 14.6).

14.2.2 Models of Time Series

Models for time series data can have many forms and represent different stochastic processes. When modeling variations in the level of a process, three broad classes of practical importance are the autoregressive (AR) models, the integrated (I) models, and the moving average (MA) models. These three classes depend linearly on previous data points. Combinations of these ideas produce autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models. A further extension of the SRIMA model for seasonal time series is called as SARIMA model.

For example, an autoregressive (AR) model is a representation of a type of random process; as such, it is used to describe certain time-varying processes in

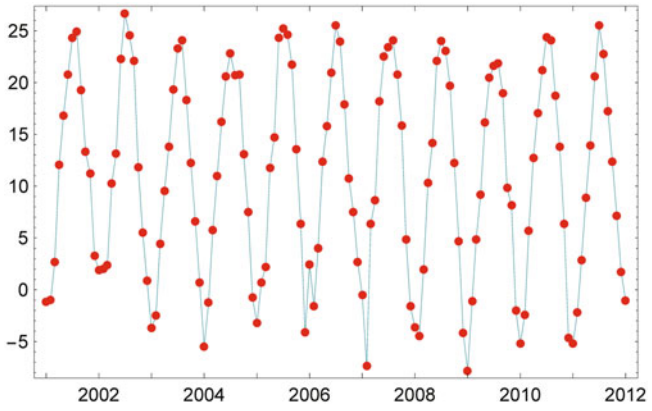


Fig. 14.6 The average temperature on the first day of a month in Chicago

nature, economics, etc. The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term); thus the model is in the form of a stochastic difference equation.

$$y_t = c + \sum_{i=1}^{p} \alpha_i y_{t-i} + \varepsilon_t$$

where p is the order of the autoregressive model, c is constant α_i are the parameters of the method and ε_t is white noise.

To demonstrate the application of this type of random process, let us fit an AR process to the average temperature of Chicago,

```
ARproc = TimeSeriesModelFit[temp, "AR"]
```

The fitted AR model of order six is,

```
ARProcess[11.2784, {0.607721, 0.0381793, - 0.165581, - 0.140076,
  - 0.100544, - 0.267987}, 8.05676]
```

The statistics of the model parameter can be seen in Table 14.1.
Now we may forecast the average temperature for one year ahead (Fig. 14.7),

```
forecast = TimeSeriesForecast[ARproc, {1, 12}];
```

The Fig. 14.8 shows the known data with the forecasted temperature values.
Our model represents a random model consequently many representations of the process trajectory are possible. One hundred trajectories of the process with their mean can be seen on Fig. 14.9.
Let us calculate the standard error bands for the forecasted trajectory; see Fig. 14.10.
Similarly the 95% confidence band can be also computed, see Fig. 14.11.
Now we can try to employ different types of time series models and select the best one. In our case the SARIMA {(1, 0, 0), (2, 1, 1)} model proved to be the model that fits with smaller error, see Table 14.2.

Table 14.1 Statistics of the fitted AR(6) model

	Estimate	Standard error	*t*-statistic	*P*-value
α_1	0.607721	0.0835393	7.27467	1.34616×10^{-11}
α_2	0.0381793	0.0983854	0.388058	0.349297
α_3	−0.165581	0.0976889	−1.69498	0.0462095
α_4	−0.140076	0.0976889	−1.4339	0.076974
α_5	−0.100544	0.0983854	−1.02194	0.154332
α_6	−0.267987	0.0835393	−3.20792	0.00083767

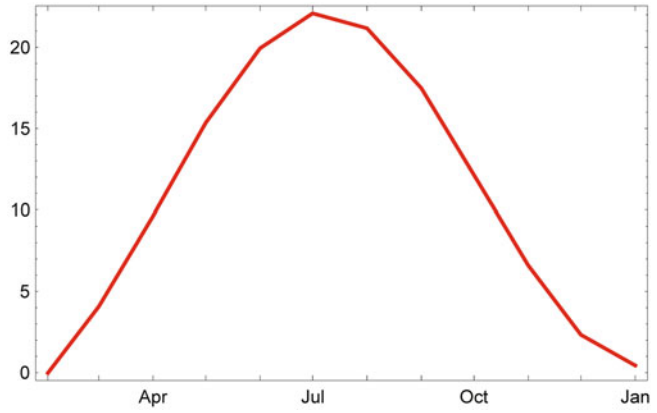


Fig. 14.7 The estimated of the average temperature for the first consecutive year

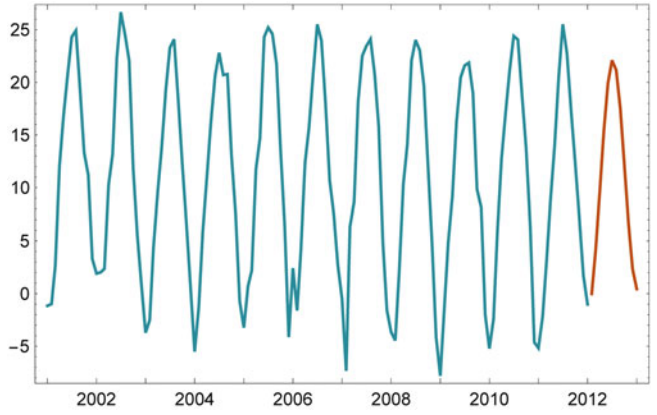


Fig. 14.8 The old and the forecasted temperature values

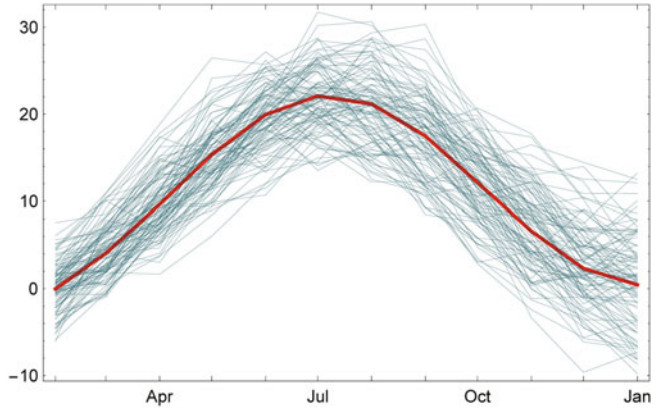


Fig. 14.9 Hundred generated trajectories with the average value

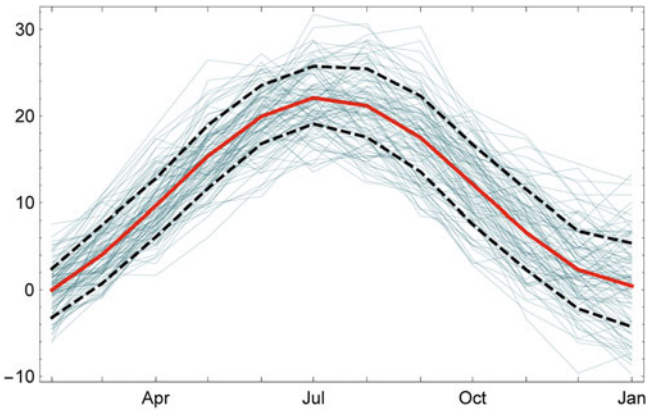


Fig. 14.10 The standard error band

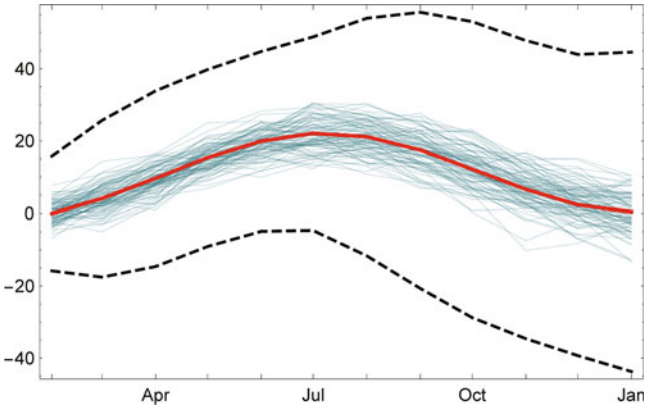


Fig. 14.11 The 95% confidence band

```
TimeSeriesModel[ ⊞ Family: SARIMA
                 Order: {{1, 0, 0}, {2, 1, 1}12} ]
```

The best model is

```
Normal[tsp]
SARIMAProcess[-0.21347,{0.151342},0,{},
 {12,{-0.292346,-0.347534},1,{-0.549841}},4.61513]
```

To compare the different models we used the Akaike criterion. The Akaike information criterion (AIC) is a measure of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the

quality of each model, relative to each of the other models. Hence, AIC provides a means for model selection.

Suppose that we have a statistical model of some data. Let T be the maximum value of the likelihood function for the model (M) and let k be the number of estimated parameters in the model. Then the AIC value of the model is the following,

$$AIC = 2k - 2\ln(T)$$

Given a set of candidate models for the data, the preferred model is the one with the minimum AIC value. AIC rewards goodness of fit (as assessed by the likelihood function), but it also includes a penalty that is an increasing function of the number of estimated parameters. The penalty discourages over fitting, because increasing the number of parameters in the model almost always improves the goodness of the fit.

The AIC value of our previous selected AR model as well as that of the SARIMA model can be seen in Tables 14.2 and 14.3 respectively.
Then the probability that AR model is better than the SARIMA one is,

```
Exp[(215.402 - 293.506)/2]
1.09631 × 10^-17
```

The statistics of the main parameters of the selected SARIMA model can be seen in Table 14.4.
Figures 14.12, 14.13, 14.14, 14.15 and 14.16 shows the same features of the fitted time series in case of the SARIMA model, similarly to the ARIMA model.

Table 14.2 Comparison of the different SARIMA models

	Candidate	AIC
1	SARIMAProcess[{1,0,0},{2,1,1}$_{12}$]	215.402
2	SARIMAProcess[{1,0,0},{2,1,2}$_{12}$]	215.558
3	SARIMAProcess[{0,0,1},{2,1,2}$_{12}$]	215.656
4	SARIMAProcess[{0,0,0},{2,1,2}$_{12}$]	216.444
5	SARIMAProcess[{0,0,0},{2,1,1}$_{12}$]	216.526
6	SARIMAProcess[{1,0,0},{3,1,1}$_{12}$]	217.228
7	SARIMAProcess[{1,0,1},{2,1,2}$_{12}$]	217.423
8	SARIMAProcess[{1,0,1},{2,1,1}$_{12}$]	217.739
9	SARIMAProcess[{2,0,0},{2,1,2}$_{12}$]	217.889
10	SARIMAProcess[{0,0,0},{1,1,2}$_{12}$]	218.016

Table 14.3 The AIC measure of the AR(6) process

	Candidate	AIC
1	ARProcess[6]	293.506
2	ARProcess[5]	301.418

Table 14.4 Statistics of the selected SARIMA model

	Estimate	Standard error	*t*-statistic	*P*-value
α_1	0.151342	0.221397	0.683579	0.247715
α_1	−0.292346	0.121123	−2.41363	0.00857806
α_2	−0.347534	0.102341	−3.39583	0.000451239
β_1	−0.549841	0.1131	−4.86155	1.61578×10^{-6}

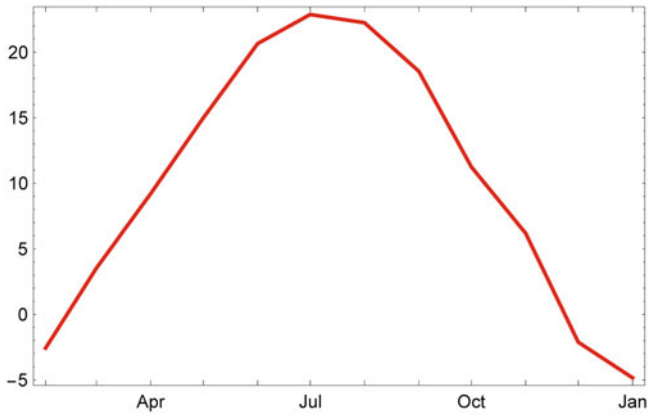


Fig. 14.12 The estimated average temperature for the first consecutive year in case of SARIMA model

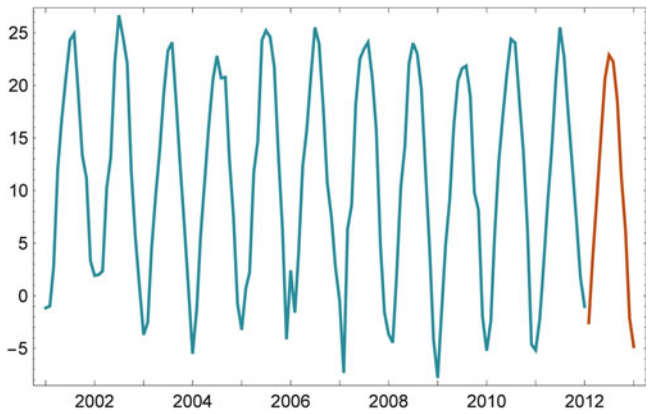


Fig. 14.13 The old and the forecasted temperature values of the SARIMA model

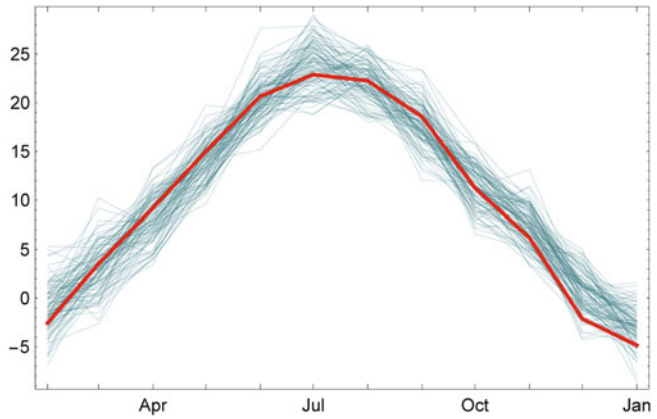


Fig. 14.14 One hundred generated trajectories with the average value

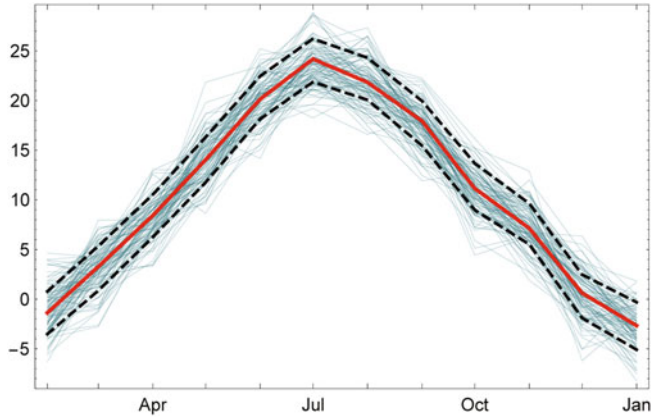


Fig. 14.15 The standard error band for the SARIMA model

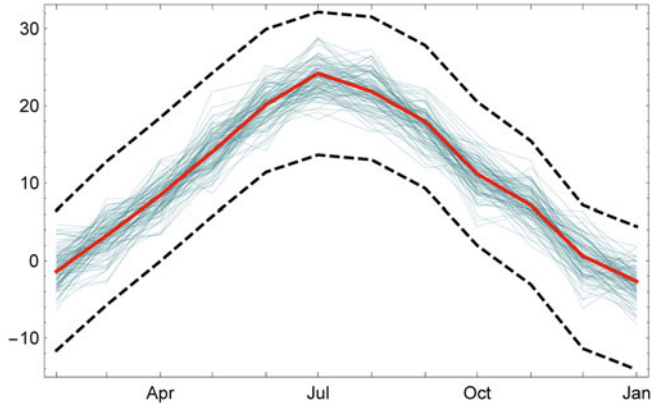


Fig. 14.16 The 95% confidence band in case of the SARIMA model

```
forecast = TimeSeriesForecast[tsp, {1, 12}];
```

Let's calculate the standard error bands for each time stamp:

```
sd1 = TimeSeriesThread[Mean[#] - StandardDeviation[#] &, simul];
sd2 = TimeSeriesThread[Mean[#] + StandardDeviation[#] &, simul];
```

```
err = forecast['MeanSquaredErrors']
```

```
TemporalData[ Time: 01 Feb 2012 to 01 Jan 2013
              Data points: 12   Paths: 1 ]
```

These Figs. also demonstrate the superiority of the SARIMA model.

14.3 Stochastic Differential Equations (SDE)

The time series mainly represent *discrete* stochastic processes. Now we consider stochastic differential equations as a *continuous* model of the random processes.

14.3.1 Ito Process

A typical form of the stochastic differential equations is

$$dy(t) = a(y(t), t)dt + b(y(t), t)dW(t)$$

where $dW(t)$ is a Wiener process, a continuous-time random walk.

14.3.2 Ito Numerical Integral

The solution of the SDE is

$$y(t) = y_0 + \int_0^t a(y(s), s)\, ds + \int_0^t b(y(s), s)\, dW(s)$$

In most practical situations there is no analytical solution for this integral, therefore numerical approximation is required.

14.3.3 Euler-Maruyama Method

To solve a model numerically, one needs—controversially—to discretize the model. The most simple discretized approximation is the *Euler-Maruyama* method,

$$y(t_{n+1}) = y(t_n) + a(y(t_n), t_n)\,\Delta t + b(y(t_n), t_n)\,\Delta W(t_n)$$

where $t_n = n\Delta t$ and $\Delta W(t_n) = W(t_n) - W(t_{n-1})$. The order of this schema is 1/2. To have higher order numerical approximation one may use the Stochastic Runge Kutta Method, too.

In the next section the numerical solution of SDE will be illustrated employing the stochastic form of the *FitzHugh–Nagumo* model.

14.4 Numerical Solution of (SDE)

The *FitzHugh–Nagumo* model for excitable media is a nonlinear model describing the reciprocal dependencies of the voltage $V(t)$ across an exon membrane and a recovery variable $R(t)$ summarizing outward currents. The model is general and is also to model excitable media, for example reaction-diffusion models. The deterministic ODE model (white-box model) is described by the following system of ordinary differential equations

$$\frac{dV(t)}{dt} = \gamma\left(V(t) - \frac{V(t)^3}{3} + R(t)\right),$$
$$\frac{dR(t)}{dt} = -\frac{1}{\gamma}(V(t) - \alpha + \beta R(t)),$$

with parameters α, β, and γ, initial condition $V(0) = -1$ and $R(0) = 1$.

White-box models are mainly constructed on the basis of knowledge of physics about the system. Solutions to ODE's are deterministic functions of time, and hence these models are built on the assumption that the future value of the state variables can be predicted exactly. An essential part of model validation is the analysis of the residual errors (the deviation between the true observations and the one-step predictions provided by the model). This validation method is based on the fact that a correct model leads to uncorrelated residuals. This is rarely obtainable for white-box models. Hence, in these situations, it is not possible to validate ODE models using standard statistical tools. However, by using a slightly more advanced type of equation, this problem can be solved by replacing ODE's with SDE's that incorporate the stochastic behavior of the system: modeling error, unknown disturbances, system noise, and so on.

The stochastic SDE gray-box model can be considered as an extension of the ODE model by introducing system noise:

$$dV(t) = \gamma\left(V(t) - \frac{V(t)^3}{3} + R(t)\right)dt + \sigma dW(t),$$
$$dR(t) = -\frac{1}{\gamma}(V(t) - \alpha + \beta R(t))dt + \sigma dW(t),$$

where W(t) is a Wiener process (also known as Brownian motion), a continuous-time random walk. The next section carries out the numerical simulation of the SDE model using the parameter settings $\alpha = \beta = 0.2$, $\gamma = 3$ and $\sigma = 0.1$.

These equations represent an Ito-stochastic process that can be simulated with a stochastic Runge-Kutta method.

```
procVR = ItoProcess[{dV[t] == γ(V[t] - V[t]^3/3 + R[t])dt + σdwV[t],
  dR[t] == - (1/γ)(V[t] - α + βR[t])dt + σdwRσdwR[t]},
  {V[t],R[t]},{{V,R},{-1,1}},t,{wV ≈ WienerProcess[],
  wR ≈ WienerProcess[]}];
param = {α → 0.2, β → 0.2, γ → 3, σ → 0.1};
```

14.4.1 Single Realization

First, a single realization is simulated in the time interval $0 \leq t \leq 20$ (Fig. 14.17),

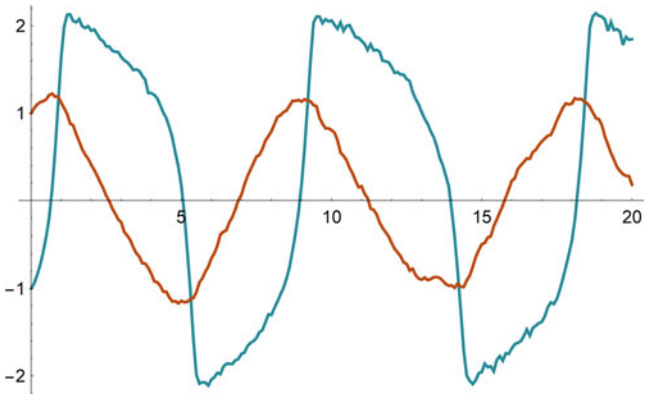


Fig. 14.17 The trajectories of the state variables $V(t)$ (blue) and $R(t)$ (brown), in case of a single realization of the Ito process

```
dataVR = RandomFunction[procVR /.param, {0, 20, 0.1},
 Method → " StochasticRungeKutta"];
```

The values of V and R can be found at any time point, for example,

```
dataVR["SliceData", τ] /.τ → 12.
{{1.52811, - 0.477377}}
```

14.4.2 Many Realizations

Now let us compute the trajectories for 100 realizations (Fig. 14.18).
Slice distributions of the state variables can be computed at any time point. First, let us simulate the trajectories in a slightly different and faster way, now using 1000 realizations.

```
td = RandomFunction[procVR/.param, {0., 20., 0.1}, 1000]
```

```
TemporalData[ Time: 0. to 20.
              Data points: 201 000   Paths: 1000 ]
```

14.4.3 Slice Distribution

At the time point $\tau = 12$, we can compute the mean and standard deviation for both state variables $V(\tau)$ and $R(\tau)$ as well as their histograms (Fig. 14.19).

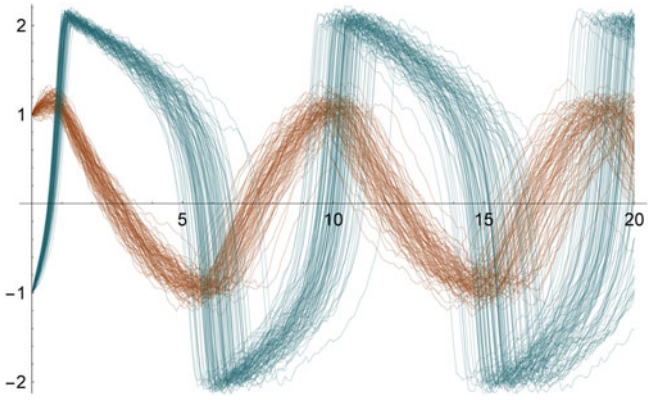


Fig. 14.18 The trajectories of the state variables $V(t)$ (blue) and $R(t)$ (brown), in case of 100 realizations of the Ito process

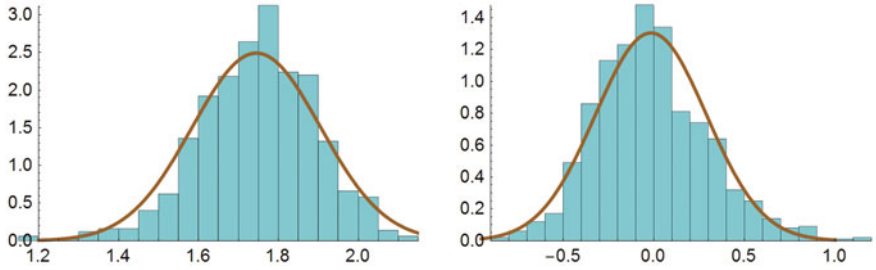


Fig. 14.19 The distribution of $V(\tau)$ (left) and $R(\tau)$ (right), in case of 100 realizations of the Ito process, at time $\tau = 12$

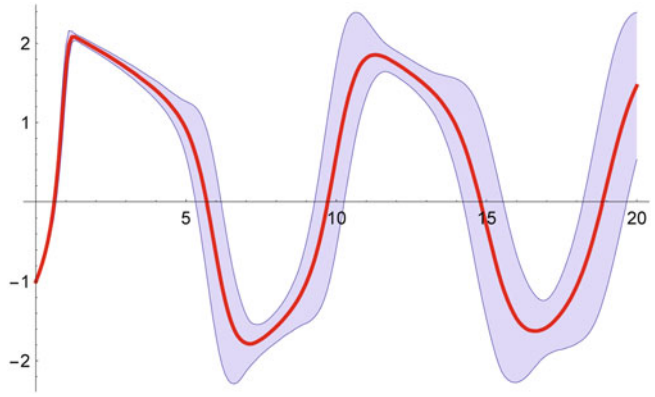


Fig. 14.20 The $\mu_V(t)$, the mean value of $V(t)$ (red) with its standard deviation, $\mu_V(t) \pm \sigma_V(t)$ (blue)

14.4.4 Standard Error Band

The mean value and the standard deviation of the trajectories along the simulation time, $0 \leq t \leq 20$ can be also computed and visualized (This may take a few minutes.) (Fig. 14.20).

14.5 Parameter Estimation

Parameter estimation is critical since it decides how well the model compares to the measurement data. The measurement process itself may also have serially uncorrelated errors due to the imperfect accuracy and precision of the measurement equipment.

14.5.1 Measurement Values

Let us write the measurement equation as

$$y_k = V(t_k) + e_k,$$

where $e_k \sim N(0,\ \sigma_m)$. The voltage $V(t)$ is assumed to be sampled between $t = 0$ and $t = 20$ at discrete time points t_k, where $k = 0,\ 1,\ 2,\ldots,\ N$, and $N = 40$ with an additive measurement noise σ_m. To get $V(t_k)$, we consider a single realization and sample it at time points t_k. Then we add white noise with $\sigma_m = 0.1$ (Fig. 14.21).

14.5.2 Likelihood Function

As we have seen, the solutions to SDE's are stochastic processes that are described by probability distributions. This property allows for maximum likelihood estimation. Let the deviation of the measurements from the model be

$$\varepsilon_k(\theta) = y_k - \mu_V(t_k, \theta),$$

where $\theta = \{\alpha, \beta, \gamma\}$. Assuming that the density function of ε_k can be approximated reasonably well by Gaussian density, the likelihood function to be maximized is

$$L(\theta) = \frac{1}{2\pi}\exp\left(-\sum_{k=1}^{N}\varepsilon_k(\theta)^2\right),$$

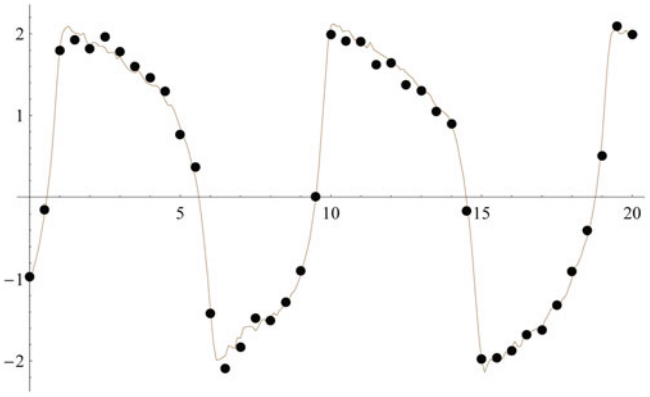


Fig. 14.21 A single realization of V(t) (brown) and the simulated measured values y_k (black points)

For computation we use its logarithm. Now we accept the model parameter γ and estimate α and β from the SDE model employing maximum likelihood method.

```
parm = {γ → 3, σ → 0.1};
```

Here are the values of the y_k.

```
yk = Transpose[dataSV][[2]];
```

We use a logarithmic likelihood function.

14.5.3 Maximization of the Likelihood Function

The optimization of the likelihood function is not an easy task, since we have frequently a flat objective function, with non-differentiable terms and more local optimums. In addition there is a long evaluation time of the model. Instead of using direct global optimization, first we compute the values of the objective function on a 25 × 25 grid. In this way one can employ parallel computation in order to decrease the running time. We are looking for the parameters in a range $-0.1 \leq \alpha, \beta \leq 0.5$. Let us create the grid points

```
XY = Table[{ - 0.1 + i0.025, - 0.1 + j0.025}, {i, 0, 24}, {j, 0, 24}];
```

We compute the function values at the grid points in parallel. Then we apply interpolation for the grid point data and visualize the likelihood function (Fig. 14.22).

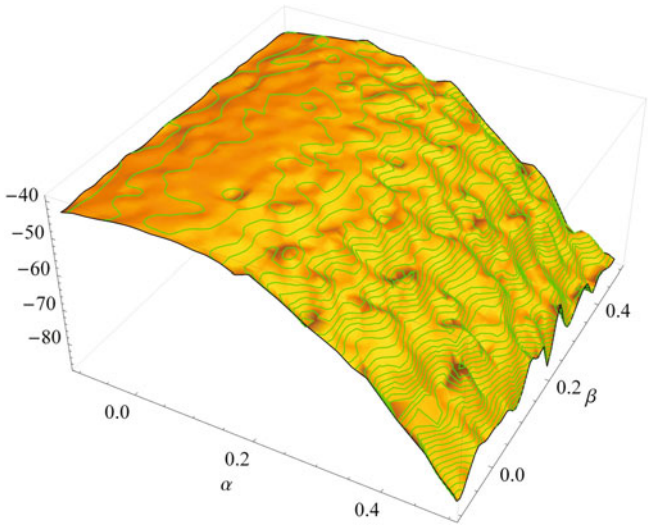


Fig. 14.22 The likelihood function of the parameter estimation problem

Table 14.5 Results of the different global methods

DifferentialEvolution	RandomSearch	SimulatedAnnealing	NelderMead
−39.7162	−39.4906	−39.4907	−40.6119
$u\rightarrow$ 0.0500002	$u\rightarrow$ 0.125	$u\rightarrow$ 0.125001	$u\rightarrow$ −0.00364276
$v\rightarrow$ 0.25	$v\rightarrow$ 0.0273485	$v\rightarrow$ 0.027331	$v\rightarrow$ 0.225

Employing different global optimization methods, we compute the parameters (Table 14.5).

There are many local maximums. Let us choose the global one.

14.5.4 Simulation with the Estimated Parameters

Here is a new parameter list.

```
paramN = {α → u, β → v, γ → 3, σ → 0.1}/.uvN[[2]]
{α → -0.125, β → 0.0273485, γ → 3, σ → 0.1}
```

This computes 500 realizations and visualizes the result and the measurement data (Fig. 14.23).

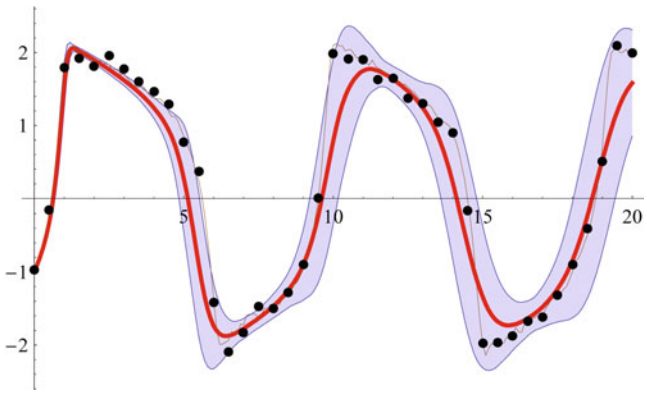


Fig. 14.23 The simulated $V(t)$ process with the estimated parameters

Table 14.6 The results of the parameter estimation for different system noises

σ	$-\log L(\alpha, \beta)$	α	β
0.00	61.060	−0.037	0.230
0.10	54.938	−0.004	0.300
0.25	52.761	0.151	0.125
0.40	54.687	0.044	0.150

14.5.5 Deterministic Versus Stochastic Modeling

In the previous parameter estimation section, we illustrated the technique of the stochastic modeling. The measurement data was simulated with the correct model parameters ($\gamma = 3$), no assumed modeling error, existing system noise $\sigma = 0.1$, and measurement error $\sigma_m = 0.1$. The model to be fitted had two free parameters, α and β, since the system noise $\sigma = 0.1$ was adopted for the model.

Now let us consider a different situation. Suppose that we have modeling error, since the measured values are simulated with $\gamma = 2.5$; however, in the model to be fitted, we use $\gamma = 3$. In addition, let us increase the measure error to $\sigma_m = 0.2$. To compare the efficiencies of the deterministic and stochastic models, we should handle σ as a free parameter in the stochastic model to be fitted. So we have then three free parameters to be estimated: α, β and σ. Let us carry out the parameter estimation for different values of σ.

The results can be seen in Table 14.6; $\sigma = 0$ corresponds to the deterministic model. In order to demonstrate that the stochastic model can provide significant improvement compared with the deterministic model, the likelihood-ratio test can be applied. The likelihood-ratio test has been used to compare two nested models. In our case, one of the models is the ODE model having less parameters than the other one, the SDE model. The null hypothesis is that the two models are basically the same. The test statistic is,

$$R = 2\big((-\log L(\alpha,\beta))_D - ((-\log L(\alpha,\beta))_S\big),$$

where the subscripts D and S stand for the deterministic and stochastic model, respectively.

The distribution of R is $\chi^2(f)$, where f is the difference in the number of parameters between the two models; in our case, $f = 3 - 2 = 1$. Here is the critical value for $\chi^2(1)$ at confidence level of 95%.

```
InverseCDF[ChiSquareDistribution[1],0.95]
3.84146
```

If the value of R is less than the critical value, then the null hypothesis can be accepted. In our case $R = 2 \times (61.060 - 52.761) = 16.598$. Therefore we reject

the null hypothesis, which means the SDE model can be considered as a different model providing a significant improvement compared with the ODE model.

14.6 Applications

14.6.1 Rotating Ellipsoid with a Stochastic Flattening

Experimental data suggest that the Earth's short time dynamics is related to stochastic fluctuation in its shape. Let's use a simple "toy model" to represent the geometric flattening, a rotating ellipsoid in the framework of stochastic differential equations. This allows us to determine an explicit drift component in the dynamical flattening and also the second zonal harmonic. First we summarize the equations of the deterministic toy model.

14.6.1.1 Model for the Deterministic Flattening

In the deterministic case the motion of an ellipsoid can be described by the deterministic perturbed Euler-Liouville (DEL) equations under the following assumptions:

(a) conservation of the ellipsoid (Earth) mass,
(b) conservation of the ellipsoid (Earth) volume.
(c) bounded variation of the flattening.

The variation in time of the semi-major axis $c_t(t)$ is given by

$$\frac{dc_t}{dt} = f(t, c_t).$$

Assuming free motion, the inertia matrix has diagonal elements only, I_i, $i = 1, 2, 3$. Their time dependency can be represented by the following equations

$$\frac{dI_1}{dt} = \frac{M_E}{5}\left(-\frac{3V_\varepsilon}{4}\pi\frac{f(t,c_t)}{c_t^2} + 2c_t f(t,c_t)\right),$$
$$\frac{dI_3}{dt} = \frac{3M_E V_E}{10\pi}\left(-\frac{f(t,c_t)}{c_t^2}\right),$$

where M_E and V_E are the mass and volume the of Earth, respectively, and $I_2(t) \equiv I_1(t)$.

In order to perform numerical simulation an ad hoc admissible deformation function is considered,

$$f(t,x) = \alpha\cos(\gamma t)\,(x-(c_0+d_{\min}))\,((c_0+d_{\max})-x),$$

where α and γ are real numbers. Then the major axis satisfies the following differential equation,

$$\frac{dc_t}{dt} = \alpha\,\cos(\gamma t)(c_t-(c_0+d_{\min}))((c_0+d_{\max})-c_t),$$

where $c_0 = c_t(0)$. The constants $d_{\min}$ and $d_{\max}$ are an admissible deterministic variation, namely,

$$f(t,\ c_0+d_{\min}) \geq 0,$$
$$f(t,\ c_0+d_{\max}) \leq 0, \quad \forall \geq t \geq 0.$$

14.6.1.2 Simulation of the Deterministic Flattening

Parameter values are:

```
param = {a0 → 1, c0 → Sqrt[298/300], ME → 1, α → 10^-3, β → 10^-4, γ
          → 10};
```

in addition

```
paramA = {VE → 4/3 π a0^2 c0, dmin → c0 - a0, dmax → a0 - c0}/.param;
```

then perturbation function is

```
f[t_, x_] := 10^-3 Cos[10t] (x - (2 Sqrt[298/300] - 1)) (1 - x)
```

The system of the differential equations is

```
deqs = {ct'[t] == f[t, ct[t]], I1'[t] == 1/5 (-(3VE/(4Pi)) 1/ct[t]^2 f[t, ct[t]] +
  2ct[t] f[t, ct[t]]),
  I2'[t] == 1/5 (-(3VE/(4Pi)) 1/ct[t]^2 f[t, ct[t]] +
  2ct[t] f[t, ct[t]]), I3'[t] == 31VE/(10Pi) (-f[t, ct[t]]/ct[t]^2)}
```

The initial conditions are

$$\texttt{inits} = \{\texttt{ct[0]} == \sqrt{298/300}, \texttt{I1[0]} == \frac{1}{5}1(1 + 298/300), \texttt{I2[0]} == \frac{1}{5}1(1 + 298/300), \texttt{I3[0]} == \frac{2}{5}11\}$$

Now let us solve the system,

```
sol =
NDSolve[Join[deqs,inits],ct,I1,I2,I3,t,0,2,MaxStepSize → 10^-4];
```

We can visualize the results (Figs. 14.24, 14.25 and 14.26):

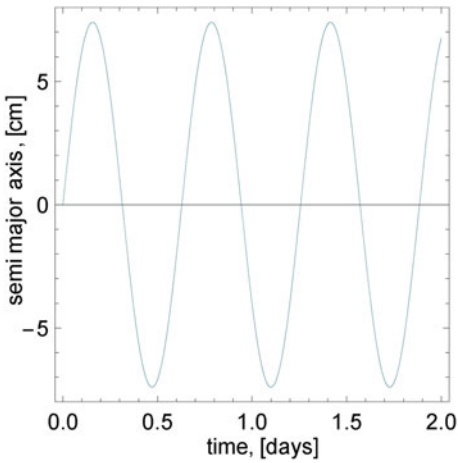


Fig. 14.24 Semi-major axis, $c_t(t) - c_0$

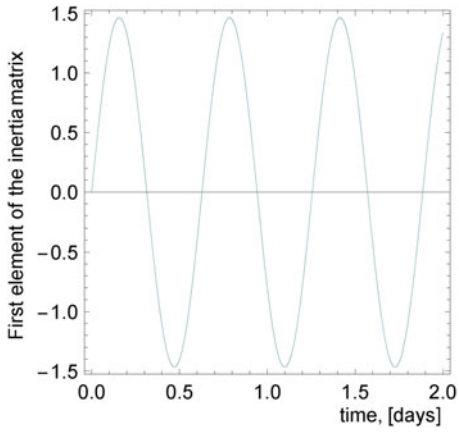


Fig. 14.25 First element of the diagonal inertia matrix, $I_1(t) - I_1(0)$

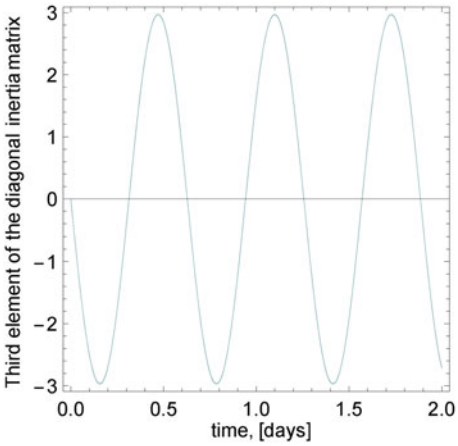


Fig. 14.26 Third element of the diagonal inertia matrix, $I_3(t) - I_3(0)$

Now we can compute the second degree zonal harmonic,

$$J_2(t) = \frac{-I_1(t) + I_3(t)}{M_E a(t)^2}$$

but

$$a(t) = \sqrt{\frac{3V_E}{4\pi c_t(t)}}$$

therefore

$$J_2(t) = \frac{-I_1(t) + I_3(t)}{M_E} \frac{4\pi c_t(t)}{3V_E}$$

on the other hand,

$$J_2(0) = \frac{-I_1(0) + I_3(0)}{M_E a(0)^2} = \frac{-\frac{1}{5} M_E \left(a_0^2 + c_0^2\right) + \frac{2}{3} M_E a_0^2}{M_E a_0^2} = \frac{-\frac{1}{5}\left(a_0^2 + c_0^2\right) + \frac{2}{3} a_0^2}{a_0^2}$$

```
J20 = ( - (1/5)(1 + 298/300) + 2/5 1)/1.
0.00133333
```

more precisely (Fig. 14.27)

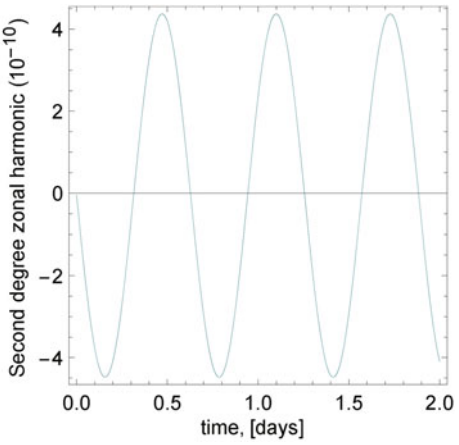


Fig. 14.27 Second degree zonal harmonic, $J_2(t) - J_2(0)$

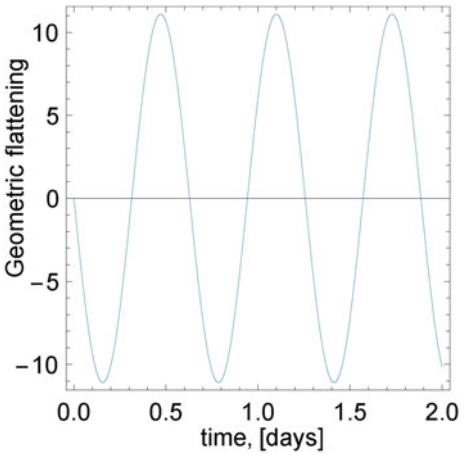


Fig. 14.28 Geometric flattening, $f_G(t) - f_G(0)$

```
J20 = 0.0013333346896384094787149832722 8;
```

as well as geometric flattening (Fig. 14.28):

$$f_G(t) = \frac{a(t) - c_t(t)}{a(t)}.$$

and the dynamical flattening:

$$H(t) = \frac{I_3(t) - I_1(t)}{I_3(t)}.$$

14.6.1.3 Stochastic Model for the Flattening

Let the stochastic process be expressed as

$$dc_t(t) = f(t, c_t)dt + g(t, c_t)dW(t).$$

The function $g(x)$ is designed to reproduce the observed stochastic behavior of the flattening of the Earth.

$$g(c_t) = \beta(c_t - (c_0 + d_{\min}))((c_0 + d_{\max}) - c_t).$$

Now our system is the following,

$$dI_1(t) = \frac{M_E}{5}\left(-\frac{3V_E f(t,c_t)}{4\pi}\frac{1}{c_t^2(t)} + g^2(t,c_t)\left(1 + \frac{3V_E}{4\pi c_t^3(t)}\right) + 2c_t(t)f(t,c_t)\right)dt + \frac{M_E}{5}g(t,c_t)\left(2c_t(t) - \frac{3V_E}{4\pi c_t^2}\right)dW(t)$$

$$dI_2 = \frac{3M_E V_E}{10\pi}\left(-\frac{f(t,c_t)}{c_t^2} + \frac{g^2(t,c_t)}{c_t^3}\right)dt - \frac{3M_E V_E}{10\pi}\frac{g(t,c_t)}{c_t^2(t)}dW(t)$$

$$\frac{dI_3}{dt} = \frac{3M_E V_E}{10\pi}\left(-\frac{f(t,c_t)}{c_t^2} + \frac{g^2(t,c_t)}{c_t^3}\right)dt - \frac{3M_E V_E}{10\pi}\frac{g(t,c_t)}{c_t^2(t)}dW(t).$$

Now our equations are

```
g[x_] := 10^-4 (x - (2 Sqrt[298/300] - 1)) (1 - x)

procFlattening = ItoProcess[{dct[t] == f[t, ct[t]] dt + g[ct[t]] dW[t],
  dI1[t] == 1/5 (-(3VE/(4Pi)) 1/ct[t]^2 f[t, ct[t]] + 2 ct[t] f[t, ct[t]] + g[ct[t]]^2
  (1 + (3VE/(4 Pi)) 1/ct[t]^3)) dt + 1/5 g[ct[t]] (2 ct[t] - (3VE/(4 Pi)) 1/ct[t]^2) dW[t],

  dI2[t] == 3VE/(10 Pi) (-1/ct[t]^2 f[t, ct[t]] + g[ct[t]]^2/ct[t]^3) dt -
   3VE/(10 Pi) g[ct[t]]/ct[t]^2 dW[t],
  dI3[t] == 3VE/(10 Pi) (-1/ct[t]^2 f[t, ct[t]] + g[ct[t]]^2/ct[t]^3) dt -
   3VE/(10 Pi) g[ct[t]]/ct[t]^2 dW[t]} /. paramA /. param, {ct[t], I1[t], I2[t], I3[t]},
  {{ct, I1, I2, I3}, {Sqrt[298/300], 1/5 1 (1 + 298/300), 1/5 1 (1 + 298/300), 2/5 11}},
  t, W ≈ WienerProcess[]];
```

A single path is simulated (Figs. 14.29, 14.30, 14.31, 14.32 and 14.33),

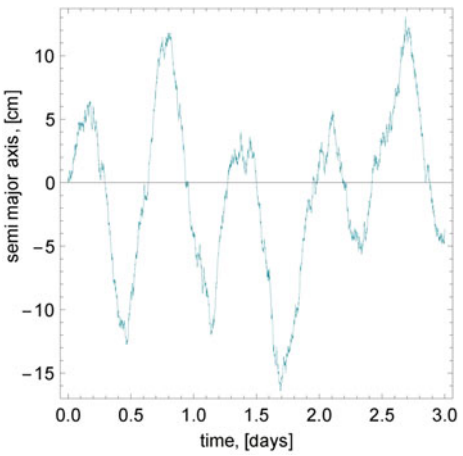


Fig. 14.29 Semi-major axis, $c_t(t) - c_0$

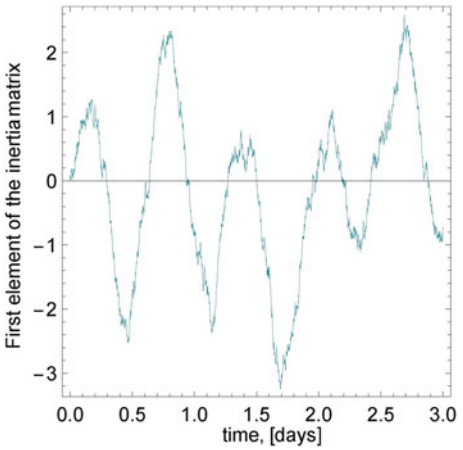


Fig. 14.30 First element of the diagonal inertia matrix, $I_1(t) - I_1(0)$

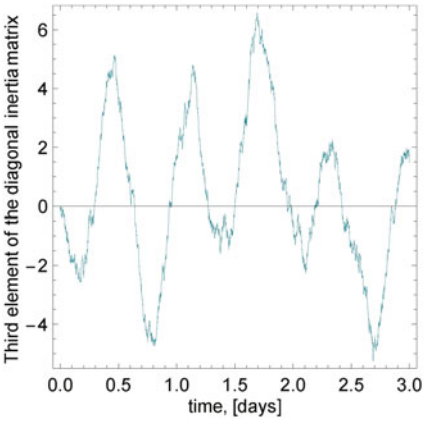


Fig. 14.31 Third element of the diagonal inertia matrix, $I_3(t) - I_3(0)$

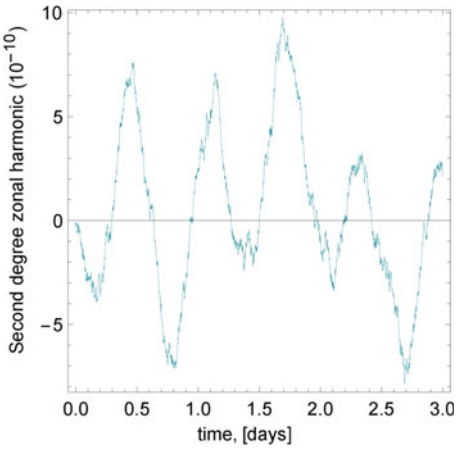


Fig. 14.32 Second degree zonal harmonic, $J_2(t) - J_2(0)$

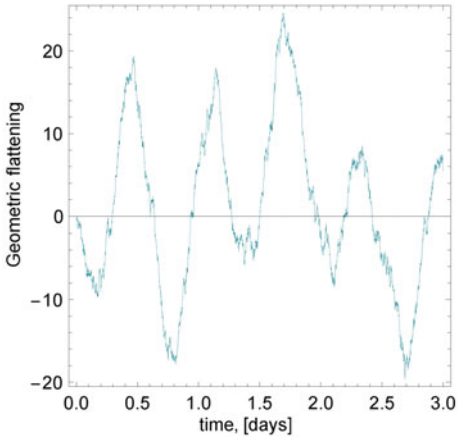


Fig. 14.33 Geometric flattening, $f_G(t) - f_G(0)$

```
dataFlattening = RandomFunction[procFlattening, {0, 3, 0.0001}, 1];
```

```
sl[t_] := Mean[dataFlattening[t]][[1]]
```

$$\texttt{csa[t\_]} := 0.66356752314 \times 10^{10} \left(\frac{\sqrt{\frac{3VE}{4\pi sl[t]}} - sl[t]}{\sqrt{\frac{3VE}{4\pi sl[t]}}} - \frac{a0 - c0}{a0} \right)$$

and the dynamical flattening:

$$H(t) = \frac{I_3(t) - I_1(t)}{I_3(t)}.$$

The stochastic simulation seems to follow qualitatively the tendency of experimental measurements, and illustrates why it is so difficult to predict the rotation motion over a few days, and also why the filtering method seems to work but without providing the physical meaning and origin.

14.6.2 Analysis of Changes in Groundwater Radon

It is well known that the correlation between ground water radon changes and earthquake occurrences is statistically significant, but it is not quite clear what the mechanisms are that lead to changes in groundwater radon before and during an earthquake. We consider a radon detection chamber consisting of two homogeneous phases, gas and liquid.

The stochastic differential equations describing the transport process of the radon from the entering original groundwater into the gas phase of the chamber are

$$\begin{aligned} dC_G(t) &= (\alpha C_G(t) + \beta C_L(t))\,dt, \\ dC_L(t) &= (\delta C_G(t) + \varepsilon C_L(t) + \varphi C_0(t))\,dt, \\ dC_0(t) &= dW(t), \end{aligned}$$

where C_0, C_L and C_G are the radon concentration in the entering original groundwater, in the liquid phase, and in the gas phase of the radon detection chamber respectively. The radon transfer in the chamber between the liquid and gas phases is controlled by Henry's law (H_{Rn}) or by the Ostwald partition coefficient, and depends also on the temperature of the liquid phase, T_L.

$$H_{Rn}(T_L) = 1/(0.425\ \exp(-0.05T_L) + 0.104,$$

The parameters are

$$\begin{aligned} \alpha(T_L) &= -\left(\lambda + \frac{k_L S}{H_{Rn}(T_L)\,V_G}\right), \\ \beta &= \frac{k_L S}{V_G}, \\ \delta(T_L) &= \frac{k_L S}{H_{Rn}(T_L)\,V_G}, \\ \varepsilon &= -\left(\lambda + \frac{Q}{V_L} + \frac{k_L S}{V_L}\right), \\ \varphi &= \frac{Q}{V_L}. \end{aligned}$$

where the constant values are:

- decay constant of ^{222}Rn: $\lambda = 1.26 \times 10^{-4}$ /min
- gas exchange constant: $k_L = 0.144$ cm/min
- volume of the gas phase: $V_G = 1.57 \times 10^3$ cm^3
- volume of the liquid phase: $V_L = 1.57 \times 10^2$ cm^3
- area of the phase boundary: $S = 3.14 \times 10^2$ cm^2
- flow rate of the groundwater: $Q = 24$ cm^3/min

The hourly timely variation of the liquid phase temperature is measured from time 0 to time 2170 h; see Fig. 14.34.

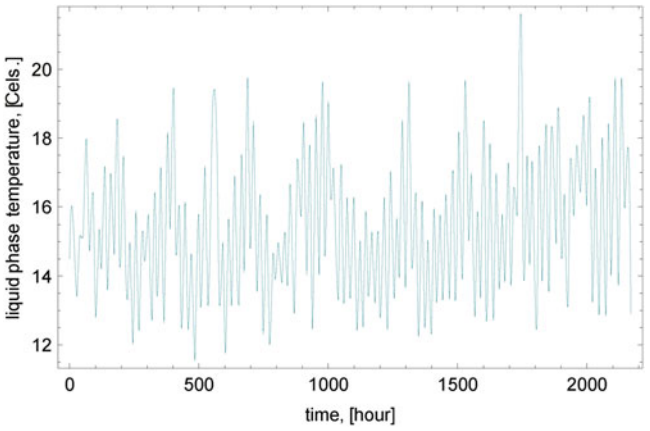


Fig. 14.34 Timely variation of the liquid phase temperature in hour

```
dataT = Map[{24(#[[1]] - 1), #[[2]]},
  Drop[Import["D:\\randon.dat"], {1, 1}]];

TL = Interpolation[dataT]
```

The constants are in hour,

```
dataP = {λ → 601.2610^-4, kL → 600.086, VG → 1.5710^3, VL → 1.5710^2,
  S → 3.14100, Q → 6024};
```

Henry's law is,

```
HRn[T_] := 1/(0.425 Exp[ - 0.05T] + 0.104)
```

The model parameters can be computed,

$$\alpha[T\_] := -\left(\lambda + \frac{kLS}{HRn[T]VG}\right)/.dataP$$

$$\beta = \frac{kLS}{VG}/.dataP$$

1.032

$$\delta[T\_] := \frac{kLS}{HRn[T]VL}/.dataP;$$

$$\varepsilon = -\left(\lambda + \frac{Q}{VL} + \frac{kLS}{VL}\right)/.dataP;$$

$$\varphi = \frac{Q}{VL}/.dataP$$

9.17197

Let us simulate a *close-up of an earthquake-related random change*, where the radon concentration of the inlet groundwater has been changed from 50 Bq/cm^3 down to 45 Bq/cm^3 at t = 0 h.

Then the Ito form of our equation system is,

```
procRadon =
 ItoProcess[{dCG[t] == (α[TL[t]]CG[t] + βCL[t])dt,
 dCL[t] == (δ[TL[t]]CG[t] + εCL[t] + φC0[t])dt, dC0[t] == dW[t]},
 {CG[t], CL[t], C0[t]}, {{CG, CL, C0}, {160, 49, 45}}, t,
 W ≈ WienerProcess[]];
```

with the initial conditions $C_G(0)$ = 160, $C_L(0)$ = 49, $C_0(0)$ = 50 all in Bq/cm^3.

First let us evaluate 100 realizations of the trajectories, Figs. 14.35, 14.36 and 14.37 show the radon concentration variation in time.

The fluctuation in the radon concentration is mostly caused by the changes in the random flux through the interface between the gas and liquid phases in the radon detection chamber, which is caused by temperature changes.

Let us compute the slice distribution of $C_G(t)$ at t = 50 and t = 200 h (Fig. 14.38).

```
SlicePDF[τ_] := Module[{},
 μCG = Mean[td[\[Tau]]][[1]];
 σCG = StandardDeviation[td[τ]][[1]];
 dataτCG = Map[#[[1]], td[τ][[2]][[2]]];
 Show[{
```

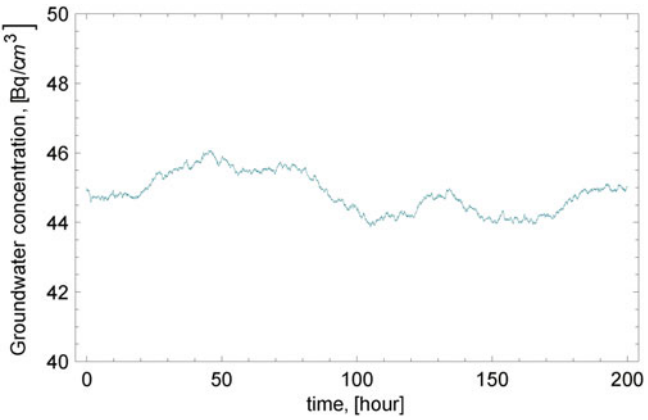


Fig. 14.35 Timely variation of the radon concentration in the entering original groundwater, $C_0(t)$

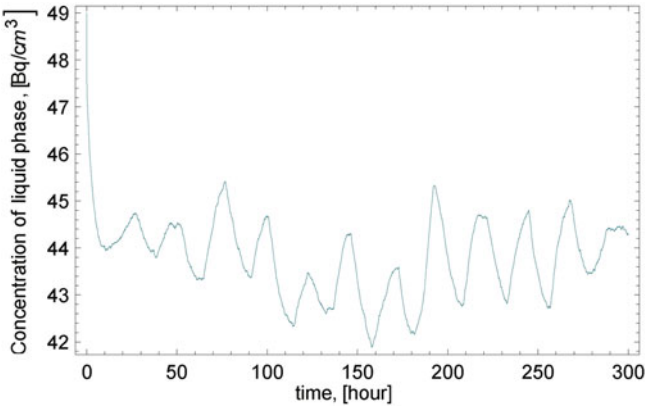


Fig. 14.36 Timely variation of the radon concentration in the liquid phase, $C_L(t)$

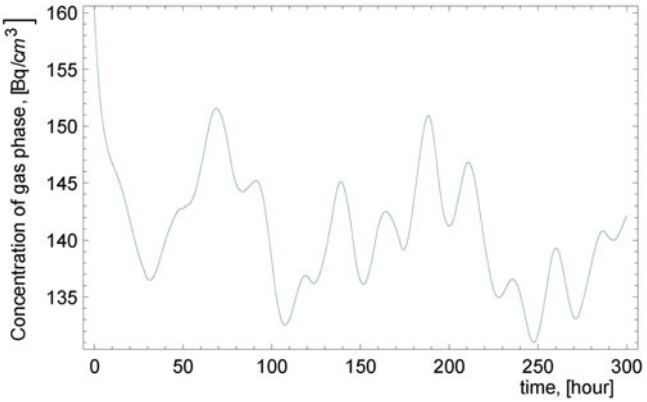


Fig. 14.37 Timely variation of the radon concentration in the gas phase, $C_G(t)$

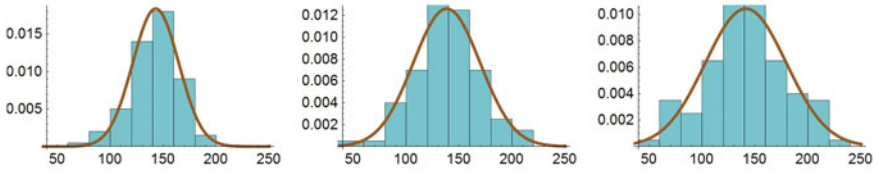


Fig. 14.38 Slice distribution of the radon concentration in the gas phase, $C_G(t)$ at t = 50, 100 and 200 h

```
Plot[PDF[NormalDistribution[μCG,σCG],t], {t,40,250},
PlotStyle → {Thick,Brown}],Histogram[dataτCG,Automatic,'PDF'],
Plot[PDF[NormalDistribution[μCG,σCG],t],{t,0,250},
PlotStyle → {Thick,Brown}]
p20 = SlicePDF[50];p100 = SlicePDF[100];p200 = SlicePDF[200];
```

These figures show that the distributions of the concentration are roughly Gaussian and they are spreading with time.

14.7 Problem

14.7.1 Deterministic Lorenz Attractor

Historically, theoretical developments in climate dynamics have been largely motivated by the work of Lorenz. He was a meteorologist studying weather forecasting and was interested in the predictability of the solutions to hydrodynamics equations. His model, complicated partial differential equations describing fluid motion and heat flow, can be reduced to three ordinary differential equations, which have the following stochastic form,

$$\begin{aligned} \frac{dx}{dt}(t) &= s(y(t) - x(t))\,dt, \\ \frac{dy}{dt}(t) &= (rx(t) - y(t) - x(t)z(t))\,dt, \\ \frac{dz}{dt}(t) &= x(t)y(t) - bz(t)\,dt. \end{aligned}$$

This model may be used for describing atmospheric convection. The usual parameter values are, $r = 28$, $s = 10$ and $b = 8/3$. First let us simulate the deterministic case,

```
param = {r → 8,s → 10,b → 8/3}
{r → 28,s → 10,b → 8/3}
```

The Ito form of our system is (Fig. 14.39, 14.40, 14.41, 14.42 and 14.43).

```
Lorenz
 NDSolve[{x'[t] == s(y[t] - x[t]),y'[t] == rx[t] - y[t] - x[t]z[t],
 z'[t] == b x[t]y[t] - b z[t],x[0] == 1,y[0] == 1,
 z[0] == 1}/.param,{x,y,z},{t,0,50}];
```

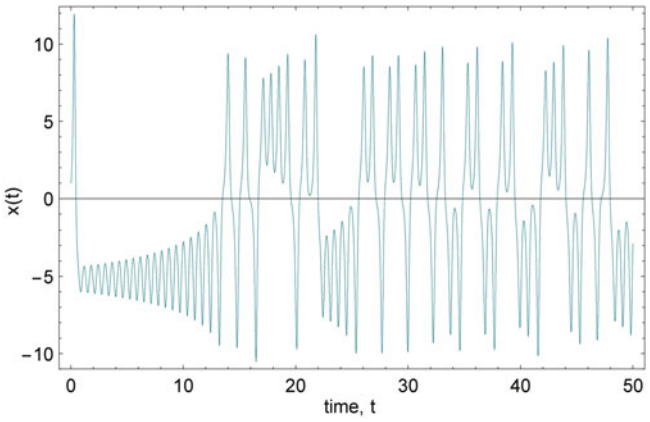


Fig. 14.39 Deterministic trajectory of $x(t)$

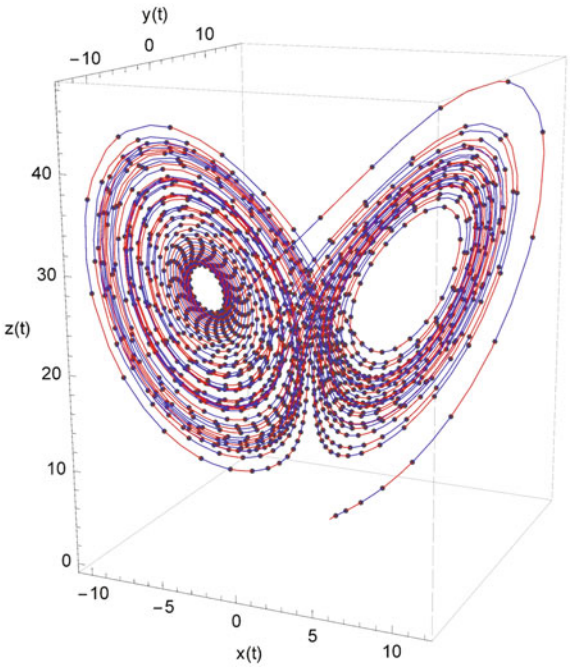


Fig. 14.40 Lorenz deterministic attractor

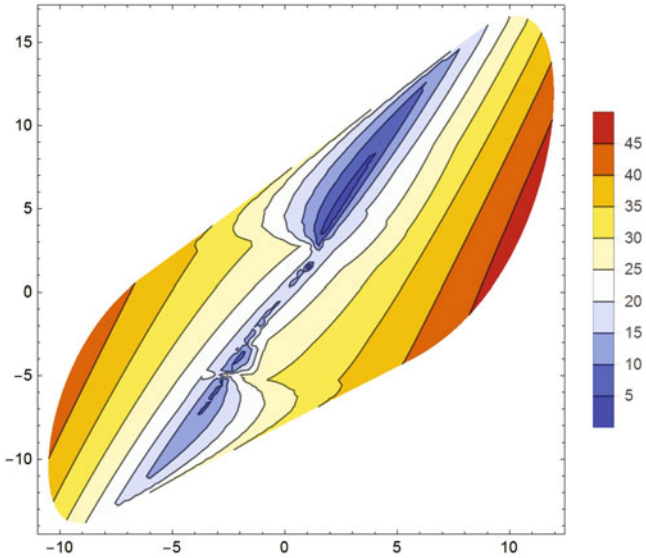


Fig. 14.41 The stochastic trajectory projection into x–y plane

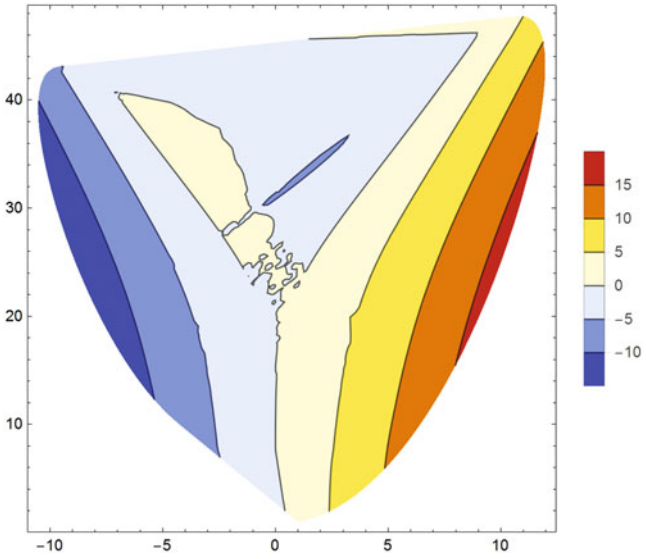


Fig. 14.42 The stochastic trajectory projection into x–z plane

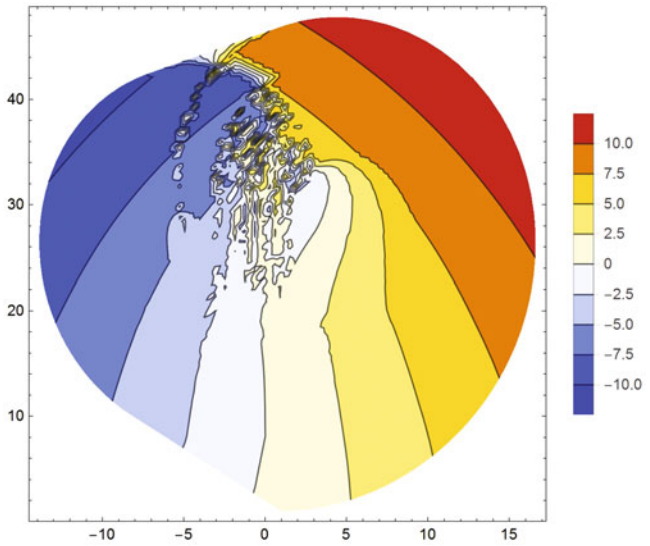


Fig. 14.43 The stochastic trajectory projection into y–z plane

We can project the points of the trajectories to the different planes.

```
data = Table[Evaluate[{x[t], y[t], z[t]}] /. Lorenz, {t, 0, 50, 0.005}];
```

```
data = Table[Evaluate[{x[t], z[t], y[t]}] /. Lorenz, {t, 0, 50, 0.005}];
```

```
data = Table[Evaluate[{y[t], z[t], x[t]}] /. Lorenz, {t, 0, 50, 0.005}];
```

The dynamics at each point in phase space, specified by the "velocity in phase space" vector $v(x, y, z) = (\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt})$ is unique. Figure 14.44 shows the density plot of the norm of these vectors in space. In this way one may detect the different velocity zones of the attractor space.

```
dataV =
 Evaluate[{x[t], y[t], z[t],
 Norm[{s(y[t] - x[t]), rx[t] - y[t] - x[t]z[t],
 b x[t]y[t] - b z[t]}]}]
```

This figure shows the region of the different velocity regions.

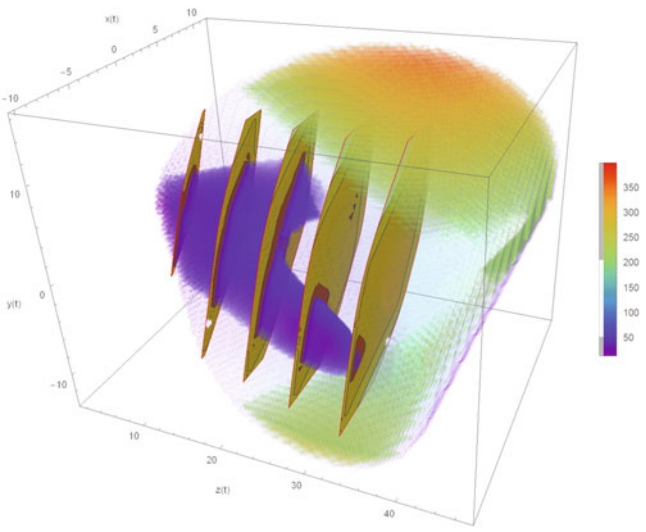


Fig. 14.44 The deterministic "phase velocity" attractor

14.7.2 Stochastic Lorenz Attractor

The stochastically perturbed Lorenz model can be written as

$$\begin{aligned} dx(t) &= \mathrm{s}\,(y(t) - \mathrm{x}(t))\,dt + \sigma x(t)\,dW(t)\,, \\ dy(t) &= (rx(t) - y(t) - x(t)\,z(t))\,dt + \sigma y(t)\,dW(t)\,, \\ dz(t) &= (x(t)\,y(t) - bz(t))\,dt + \sigma z(t)\,dW(t)\,. \end{aligned}$$

When a deterministic system is perturbed by noise, it is often observed that the topological structure of the deterministic attractor becomes fuzzy.

We use these values for the parameters:

$$\{\mathrm{r} \to 28, \mathrm{s} \to 10, \mathrm{b} \to \frac{8}{3}, \sigma \to 1\}$$

The Ito-form of our system is

```
procLorenz =
 ItoProcess[{dx[t] == s(y[t] - x[t])dt + σx[t]dW[t],
 dy[t] == (r x[t] - y[t] - x[t]z[t])dt + σy[t]dW[t],
 dz[t] == b (x[t]y[t] - z[t])dt + σ z[t]dW[t]},
 {x[t],y[t],z[t]},{{x,y,z},{1,1,1}},t,W≫WienerProcess[]]
```

Let us evaluate 100 trajectories, time 0 to 50, with time increment 0.002 (total of 250,0100 pints):

```
td = RandomFunction[procLorenz,{0,50,0.002},100]
```

Visualizing the result (Figs. 14.45, 14.46, 14.47, 14.48, 14.49 and 14.50),

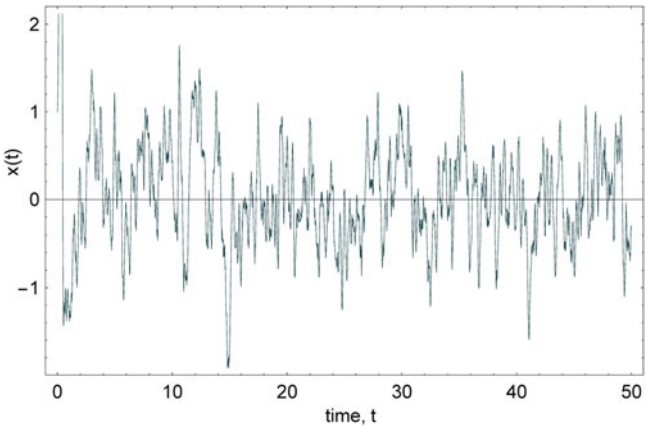
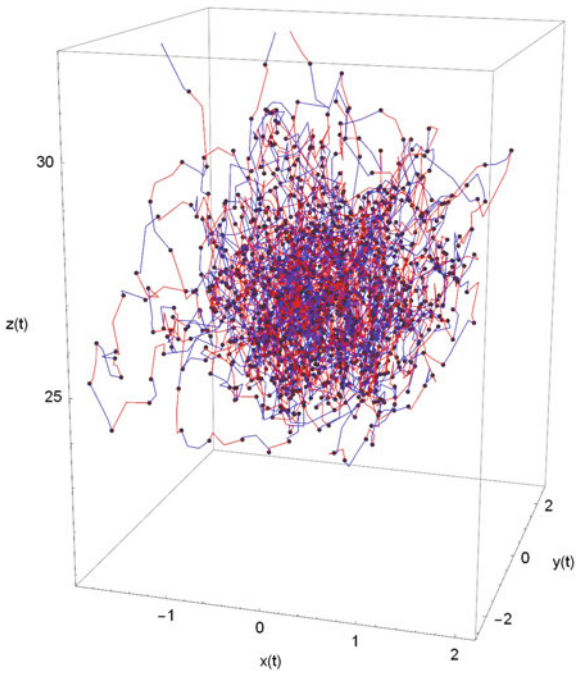


Fig. 14.45 Stochastic trajectory of $x(t)$

Fig. 14.46 The two branches of the deterministic attractor collapse



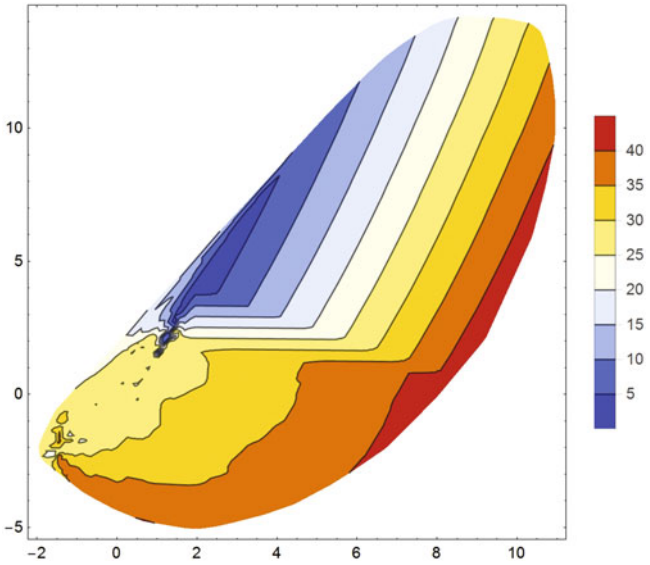


Fig. 14.47 The stochastic trajectory projection into x–y plane

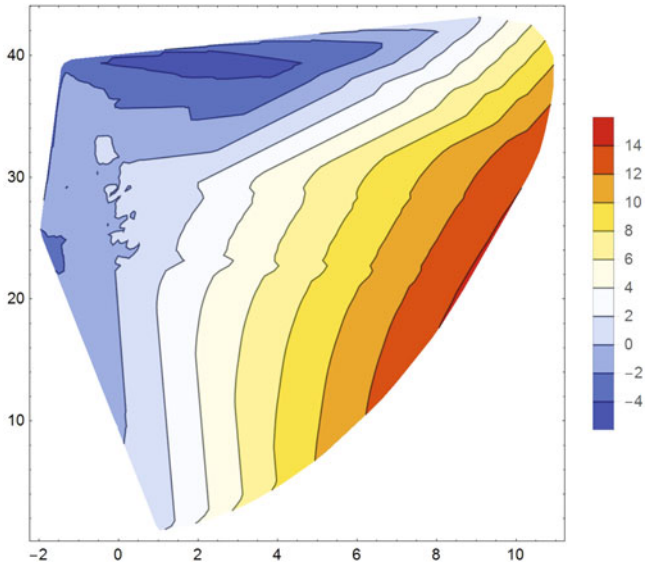


Fig. 14.48 The stochastic trajectory projection into x–z plane

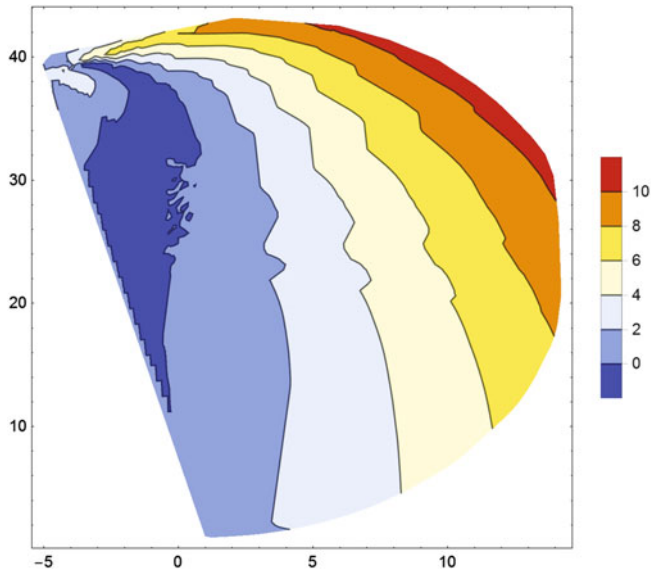


Fig. 14.49 The stochastic trajectory projection into y–z plane

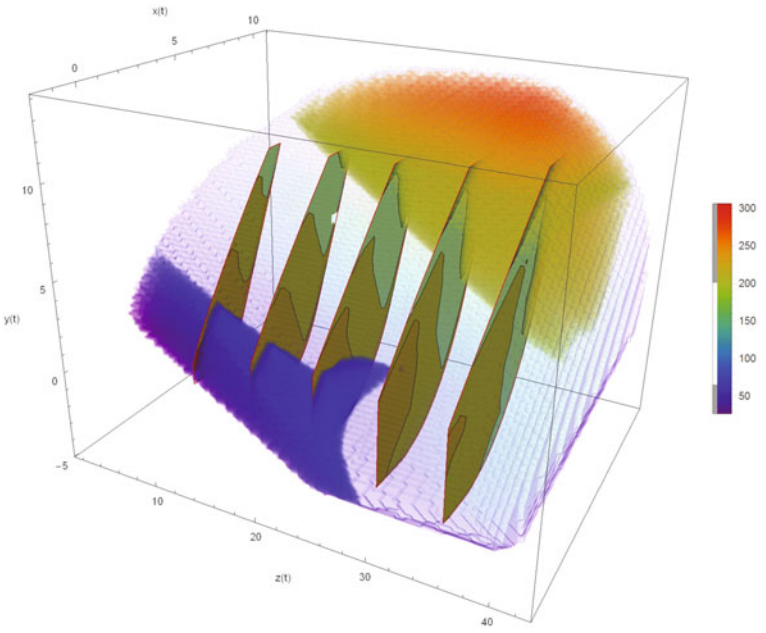


Fig. 14.50 The stochastic "phase velocity" attractor

```
data = Table[{Mean[td[t]][[1]],Mean[td[t]][[2]],
 Mean[td[t]][[3]]},{t,0,50,0.005}];

data = Table[{Mean[td[t]][[1]],Mean[td[t]][[3]],
 Mean[td[t]][[2]]},{t,0,50,0.005}];

data = Table[{Mean[td[t]][[2]],Mean[td[t]][[3]],
 Mean[td[t]][[1]]},{t,0,50,0.005}];

dataV =
 Table[{Mean[td[t]][[1]], Mean[td[t]][[2]],Mean[td[t]][[3]],
 Norm[{s(Mean[td[t]][[2]] - Mean[td[t]][[1]]),
 r Mean[td[t]][[1]] - Mean[td[t]][[2]] -
 Mean[td[t]][[1]]Mean[td[t]][[3]],
 b(Mean[td[t]][[1]]Mean[td[t]][[2]] -
 Mean[td[t]][[3]])}]},{t,0,40,0.005}];
```

This computation shows that the stochastic noise shrinks the volume of the attractor, and makes the trajectories fuzzy, but the structure of the attractor has been only slightly changed.

Chapter 15
Parallel Computations

15.1 Introduction

Although in the earlier chapters parallel computations in symbolic as well as in numeric form have been carried out in many times, in this chapter a more systematic overview is given of this topic. On the one hand, wide spread multicore computers in the market place provide ample proof that we are in the multicore era. In years to come probably the number of cores packed into a single chip will represent the increasing computational power rather than the clock frequency of the processor, which has perhaps reached its limit, and will likely stay below 4 GHz for a while.

On the other hand, popular software systems like MATLAB *Maple*, and *Mathematica* have extended their capabilities with simple instructions and functions providing a comfortable realization of parallel computation on multicore desktop (or even laptop) computers.

Consequently, parallel computing can be utilized by not only experts but engineers and scientists having no special knowledge in this area. However, despite the easily accessible hardware and the simple, user friendly software, there are some pitfalls and tricks, which are good to know in order to avoid unexpected negative effects, as well as to exploit the advantages of a multicore system.

Using illustrative examples, important features of parallel computation will be demonstrated. We show how some frequently employed algorithms in geosciences can be efficiently evaluated in parallel. We use *Mathematica*, but other systems could be used for some of these examples. As usual, we will show some of the *Mathematica* commands but not the more detailed ones. In most of these examples, the time consumed is displaced first.

© Springer International Publishing AG 2018

J. L. Awange et al., *Mathematical Geosciences*,
https://doi.org/10.1007/978-3-319-67371-4_15

15.2 Amdahl's-Law

Multicore processors are processing systems composed of two or more independent "cores" or CPUs. The amount of performance gained by the use of a multicore processor strongly depends on the software algorithms and implementation. In particular, the possible gains are limited by the fraction of the software that can be "parallelized" to run on multiple cores simultaneously; this effect is described by *Amdahl's law*. The modern version of this law states that if one enhances a fraction f of a computation by speedup S, then the overall speedup S_a is,

$$S_a(f,S) = \frac{1}{(1-f) + \frac{f}{S}}.$$

Four decades ago, Amdahl assumed that a fraction f of a program's execution time was infinitely parallelizable, with no overhead, while the remaining fraction, $1-f$, was totally sequential. He noted that the speedup on n processors is governed by

$$S_a(f,n) = \frac{1}{(1-f) + \frac{f}{n}}.$$

By this law, in the best case, so-called embarrassingly parallel problems $f \approx 1$ one may realize speedup factors near the number of cores. Many typical applications, however, do not realize such large speedup factors, and thus the parallelization of software is a significant on-going topic of research.

In the near future, more efficient core design, more complex trade-offs between cores and cache capacity, and highly tuned parallel scientific and database codes may lead to considerable improvement. Table 15.1 shows the speedup limit in the case of $f = 0.99$ for different hardware designs: symmetric, asymmetric and dynamic multi-cores.

15.3 Implicit and Explicit Parallelism

On multicore machines, as many independent processes can be executed in parallel as there are cores in the machine. However, the so-called "multi-threading execution model" allows further parallelization, since within the context of a single

Table 15.1 Upper Bound on Speedup, $f = 0.99$

Basic Core Equivalents	Base Amdahl	Symmetric	Asymmetric	Dynamic
16	14	14	14	<16
64	39	39	49	<60
256	72	80	166	<223
1024	91	161	531	<782

process running on one core, several tasks sharing the same resources can be executed concurrently.

Some of the programming systems (e.g. *Matlab*) can exploit multicore and multi-threading automatically, partly or fully, without any special directives of the programming language. This characteristic of a programming language is called *implicit parallelism*. The implicit parallelism may support only certain statements and may be efficient only in case of large computational loads. *Mathematica* also automatically supports the parallelization of some operations. Already Version 11.1 (2017) added automatic multi-threading when computations are performed on multicore computers. To illustrate implicit threading in *Mathematica,* let us compute the greatest singular value of a random $n \times n$ matrix with $n = 1200$.

```
f[n_]:=Max[SingularValueList[Table[RandomReal[],{n},{n}]]]
```

The time elapsed and result (there is really no value to return) is: (Fig. 15.1)

```
{0.430452,Null}
```

The maximum of the average CPU usage is about 30%.

Increasing the load, for example in case of $n = 5000$, the CPU usage will automatically increase. The computation time yields, (Fig. 15.2)

```
{8.00045,Null}
```

The maximum of the average CPU usage is about 50%.

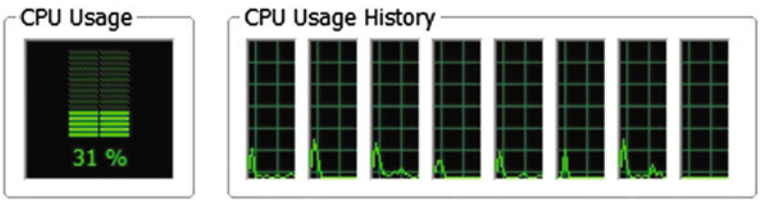


Fig. 15.1 Implicit parallelism at low computation load

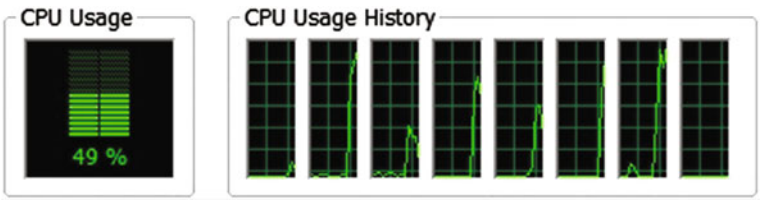


Fig. 15.2 Implicit parallelism at higher computation load

Parallelism provided by the system and controlled by the user, namely using or not using parallel execution, is called *explicit parallelism.* This is a feature that allows or forces the programmer to annotate his program to indicate which parts should be executed as independent parallel tasks. This is obviously more work for the programmer than a system with implicit parallelism (where the system decides automatically which parts to run in parallel) but may provide higher performance. *Mathematica* has many functions supporting explicit parallelism and we will use some of them in the following examples.

15.4 Dispatching Tasks

In this section we will illustrate how important can be the proper size of a task dispatched to the threads in order to avoid overloading the "master," resulting in "idle workers". We shall see that improper size of the tasks can increase the running time of parallel evaluation compared to the non-parallel case.

Let us consider the computation of the value of π with a Monte-Carlo method. The standard technique is to generate random numbers from the uniform distribution [0, 1] as pair of coordinates (x_i, y_i) of points in plane. The approximate value of π can be computed as,

$$\pi \approx 4\frac{N_p}{N},$$

where N_p is the number of the points having distance from the origin less or equal with 1, and N is the number of the all randomly generated points.

The Mathematica function to compute this simulation could be

```
PI[n_]:=4./n Length[Select[Table[{RandomReal[],RandomReal[]},
 {n}],√#[[1]]^2 + #[[2]]^2£1&]]
```

If we carry out the computation for 10 million random points, we get (again with the timing given first)

```
{31.9762,3.14168}
```

To check the CPU usage on a Windows system, open *Windows Task Manager-Performance*. (Similar utilities exist on other platforms.) Average CPU Usage is 12-13%. This means that practically only one core computation power was exploited. Of course the other cores were not completely idle, because of implicit threading. Now let us try to use explicit parallel mode using a modified statement, with `ParallelTable` instead of `Table`,

```
PIParallelNaive[n_]:=4./nLength[Select[ParallelTable[
 {RandomReal[],RandomReal[]},{n}],√#[[1]]^2 + #[[2]]^2£1&]]
```

If we invoke this on $n = 10000000$ we get

```
{35.6434,3.14332}
```

No improvement happened, it even became somewhat worse! It is clear that the master was deeply engaged with dispatching tasks, since the workers completed their small tasks very quickly and were queuing at the front of the master and asking for a new task. So the master was overloaded; it is the bottleneck of the process. (Fig. 15.3)

Now let us divide the total work into two portions and assign them to the workers. We should slightly modify our function,

```
4Length[Select[Table[RandomReal[],RandomReal[],
 {n}],√#[[1]]^2 + #[[2]]^2≤1&]]//N
```

Now the number of the available threads is eight. Every core represents two threads.

```
LaunchKernels[2$ProcessorCount]//Quiet;
```

This function will be available for all workers. We formulate the parallel evaluation as

```
t=AbsoluteTime[];
job1=ParallelSubmit[PIParallel[5000000]];
job2=ParallelSubmit[PIParallel[5000000]];
{a1,a2}=WaitAll[{job1,job2}];
pi=(a1 + a2)/10000000;
time2=AbsoluteTime[]-t
```

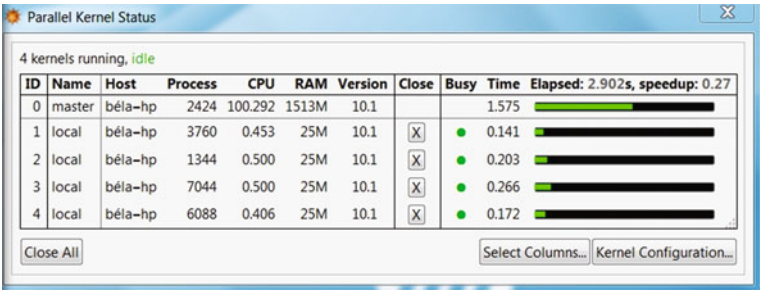


Fig. 15.3 Naive parallelism

```
16.1616284
pi
3.14196
```

Now the average CPU Usage is 25%. The problem is that only two workers were on duty, the others were idle. Let us generalize the function for m sub-tasks: (Fig. 15.4)

```
PIParallelReasonable[n_,m_] := Apply[Plus,
 WaitAll[Map[ParallelSubmit[PIParallel[#]]&,
  Table[n/m, {m}]]]]/n
```

With $m = 4$ we get

```
{10.5612, 3.14105}
```

Average CPU Usage is 50%. There is certainly improvement, but even now half of the workers remained idle. Let us use all the threads, i.e., $m = 8$:

```
{9.34044, 3.14238}
```

We get the best result when the total work is partitioned into as many parts as we have threads. The average processor usage is 100% and all threads are working. (Fig. 15.5)
The efficiency of the parallel computation can be measured as

$$\eta = \frac{singlecoremodetime}{multicoremodetime} \frac{100}{Numberofthreads}\% .$$

In our case it is about (32.5/9.3)×(100/8) = 40%

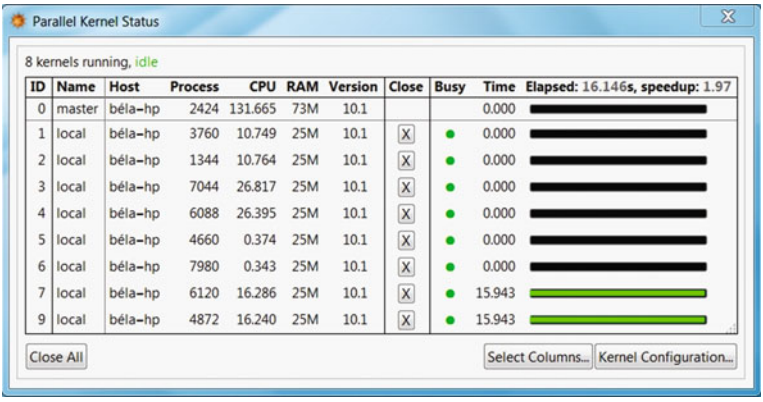


Fig. 15.4 Two workers are active

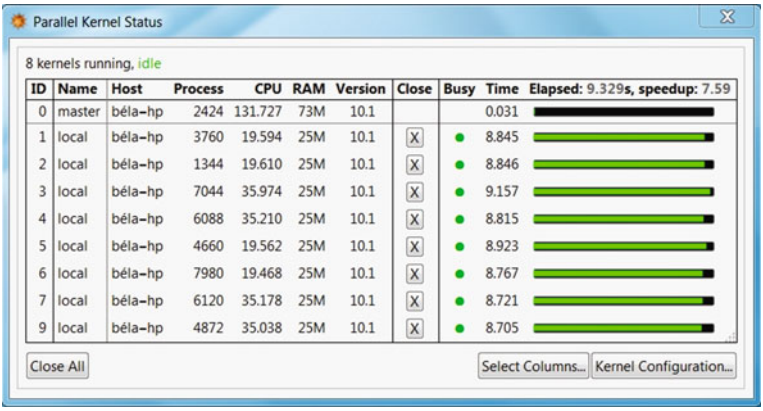


Fig. 15.5 All workers are active

15.5 Balancing Loads

It is also important to distribute tasks among the workers (threads) uniformly. However this is not always an easy job. To illustrate this situation let us consider the Monte-Carlo analysis of the stability of an infinite slope.

Infinite Slope Stability* via *Monte-Carlo Analysis

We are going to compute the safety factor for an infinite slope subject to a given ground acceleration value occurring over a specified time period. The safety factor is the ratio of resisting forces to driving forces. A slope will be stable if the resisting forces exceed the driving forces, and the factor of safety is greater than 1. We will use a simplified model of the infinite slope model. Let ϕ be the angle of internal friction (representing the frictional component of soil shear strength), β be the slope angle in degrees, and $0 \leq H \leq 1$ is the dimensionless height of the phreatic surface above the base of the slide mass; *Haneberg* (2004).

The pseudo static factor of safety of an infinite slope subjected to seismic acceleration is,

```
SeismicsFS[f_,β_,H_,Cs_]:=
 (((1-H/2) Cos[β]-Cs Sin[β])Tan[f])/(Sin[β]+Cs Cos[β])
```

in which Cs is a coefficient of seismic acceleration given in terms of the gravitational acceleration g. This is a generalization of the static model ($Cs = 0$),

```
SeismicsFS[f,β,H,0]
```

```
(1-H/2) Cot[β] Tan[ϕ]
```

According to the USGS national earthquake hazard maps, a peak ground acceleration of 0.12 g has 0.10 probability of being exceeded in 50 years in Socorro, New Mexico. So we define a function for computing the safety factor at a fixed $Cs = 0.12$ value called Slope[n].

We will censor the offensive negative values by simply removing them. Let us try $n = 80\ 000$ trials:

```
{22.573240, Null}
```

Here is the histogram showing the *Monte Carlo* simulation results. (Fig. 15.6)

The mean value is 1.01779. The deviation is .184506. Now let us carry out this simulation in parallel.

```
LaunchKernels[2 $ProcessorCount] //Quiet;

DistributeDefinitions[Slope, SeismicsFS];
```

First we distribute the tasks in equal portions among the eight threads (Figure 15.7).

```
n = Table[10000,{8}]

{10000,10000,10000,10000,10000,10000,10000,10000}

r1 = Flatten[ParallelMap[Slope[#]&,n]]; //AbsoluteTiming

{2.605205, Null}
```

Now we distribute the tasks in an unequal way:

```
n = {1000,2000,5000,8000,11000,15000,18000,20000};
```

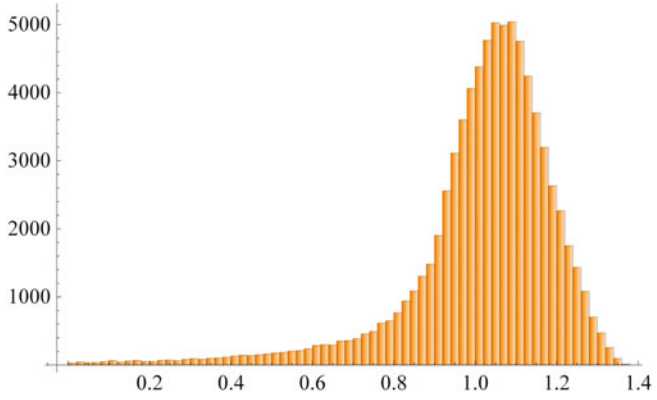


Fig. 15.6 The histogram of the distribution of the safety factor in case Cs = 0.12, employing 80 000 samples

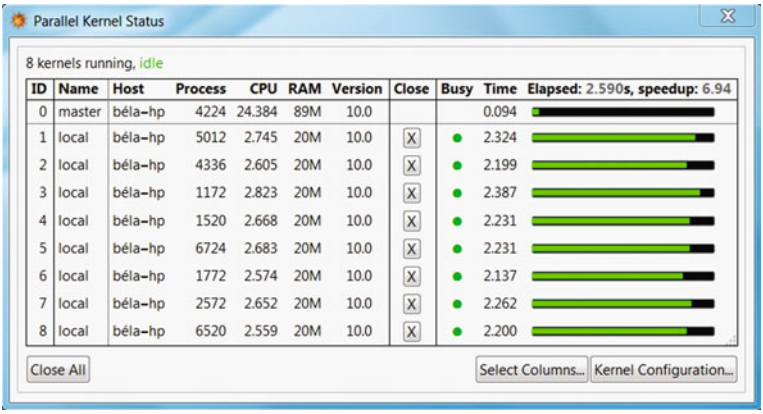


Fig. 15.7 The threads have got the same size of load and they were used nearly equally

The running time is longer and the load of the threads are unbalanced (Fig. 15.8).

```
r1 = Flatten[ParallelMap[Slope[#]&,n]];//AbsoluteTiming
{3.806407, Null}
```

That is why it is important to distribute the tasks into as equal loads as possible!

Basically, there are two types of parallel computation, namely *task parallel* and *data parallel*. In reality these two types turn up in mixed form.

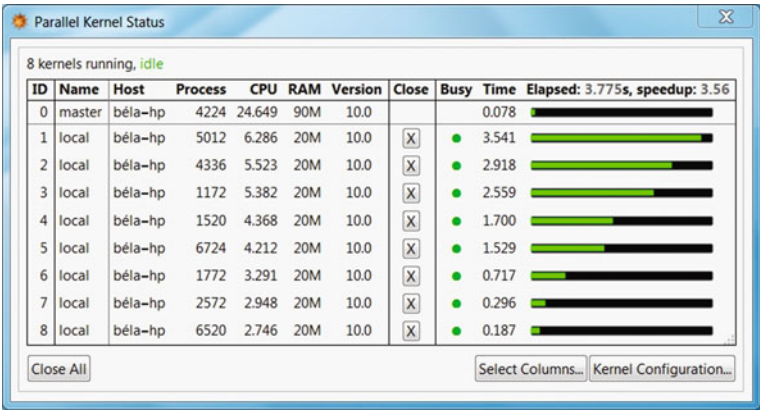


Fig. 15.8 The balance of load of the different threads

15.6 Parallel Computing with GPU

Graphics Process Unit (GPU) alias Visual Processing Unit (VPU) was designed to carry out visualization an image processing independently on CPU. So called dedicated GPU has its own hardware with high speed memory storage and multi-core processor unit to carry out these tasks very fast. This development was mainly motivated by the demand of computer gaming technology.

However, today this technique can be employed in other areas of computing, too. This generalization of the usage of GPU was strongly supported by CUDA (Compute Unified Device Architecture) a parallel computing platform developed by NVIDIA.

The most simple way to utilize this feature is to parametrize special built-in functions employing GPU. Some computational systems like Mathematica provides easy access to this computational ability even for standard users.

15.6.1 Neural Network Computing with GPU

The algorithms of training and evaluating of neural networks are mainly based on matrix operations which can be carried out effectively by using CPU architecture.

Let us illustrate this operation first by a function approximation application.

15.6.1.1 Function Approximation

Assuming Gaussian $N(0,0.2)$ noise, discrete data points of function

$$y(x) = \exp\left(-x^2\right)$$

can be generated in range $x \in [-5, 5]$, see Fig. 15.9

```
data =
 Table[x → Exp[ - x^2] + RandomVariate[NormalDistribution[0,.2]],
 {x, - 5,5,.1}];
plot = ListPlot[List@@@data, PlotStyle → Red, PlotRange → All]
```

Let us employ a feedforward network with two hidden layers containing 150 nodes each employing *Tanh*(*x*) activation function, (Table 15.2)

```
net = NetChain[{150, Tanh, 150, Tanh, 1}, "Input" →" Scalar",
  "Output" →" Scalar"]
```

In order to avoid over learning the 20% of the data points are used as validation and 80% as training set. First we carry out the training with CPU.

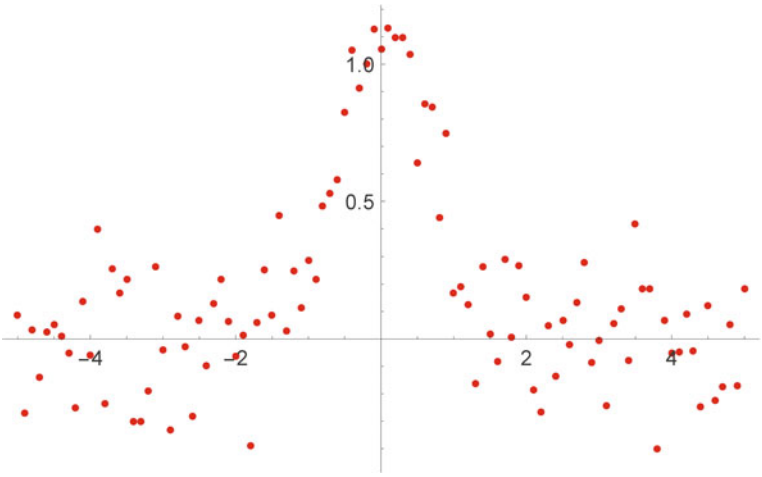


Fig. 15.9 Noisy data points

```
AbsoluteTiming[
 net2 = NetTrain[net, data, ValidationSet → Scaled[0.2]];]
```

```
{24.0289, Null}
```

Let us approximate the data with the network function, see Fig. 15.10

```
Show[Plot[net2[x], {x, -5, 5}], plot]
```

Then we repeat the computation with GPU unit (in our case GeForce GT 630), using parametrization

```
(TargetDevice →″ GPU″),
```

Table 15.2 Structure of the regression neural network

NetChain[			scalar	]
		Input	vector (sixe: 1)	
	1	LinearLayer	vector (sixe: 150)	
	2	Tanh	vector (sixe: 150)	
	3	LinearLayer	vector (sixe: 150)	
	4	Tanh	vector (sixe: 150)	
	5	LinearLayer	vector (sixe: 1)	
		Output	scalar	
		(uninitialized)		

```
AbsoluteTiming[
 net2 = NetTrain[net, data, ValidationSet → Scaled[0.2],
 TargetDevice →" GPU"];]
```

```
{11.2258, Null}
```

The running time is considerably shorter and even the result is much more realistic, see Fig. 15.11.

```
Show[Plot[net2[x], {x, -5, 5}], plot]
```

15.6.1.2 Classification Problem via Deep Learning

Our second example is an image classification problem, which can be solved via deep learning technique employing convolutional neural network. We use the CIFAR-10 database of labeled images of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. There are available 50000 colored RGB images of size 32×32. Let us download the images,

```
obj = ResourceObject["CIFAR-10"];
```

```
trainingData = ResourceData[obj, "TrainingData"];
```

As illustration here are ten randomly selected images,

```
RandomSample[trainingData, 10]
```

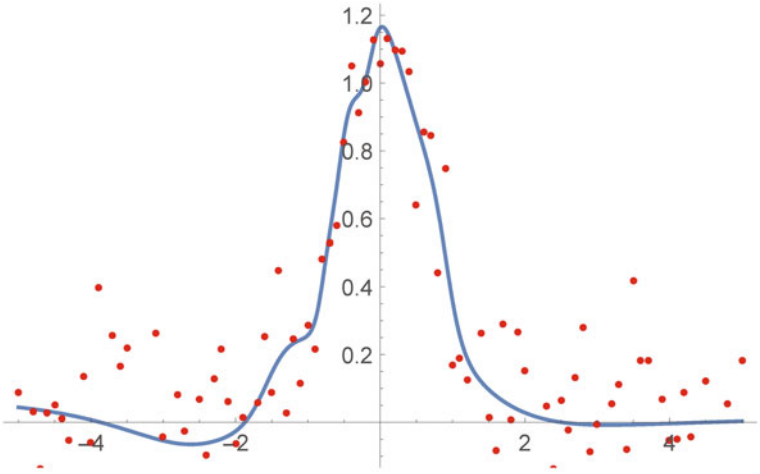


Fig. 15.10 Noisy data points approximated by the neural network

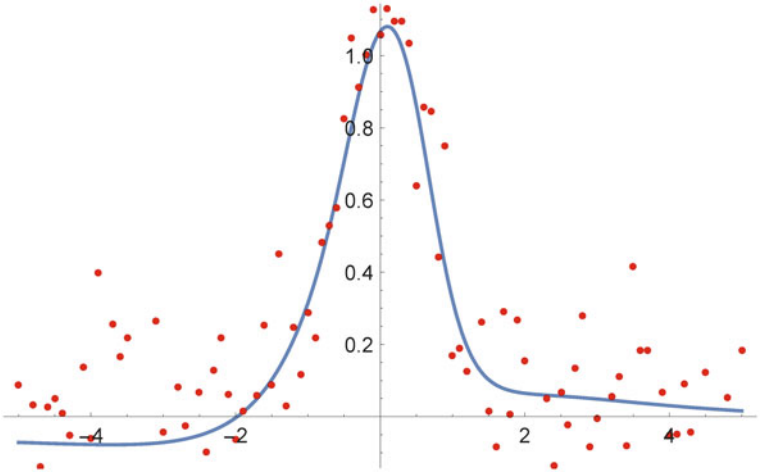
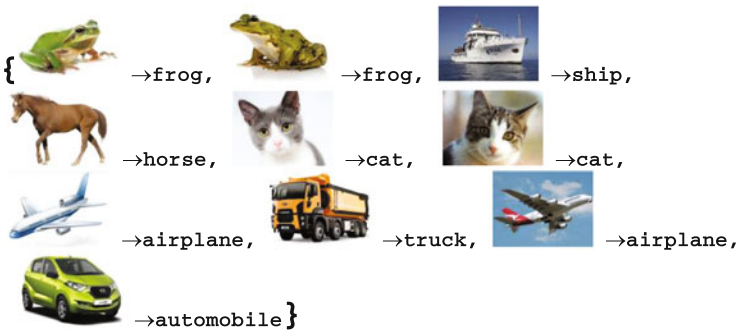


Fig. 15.11 Noisy data points approximated by the neural network



Extract the unique classes,

```
classes = Union@Values[trainingData]
```

```
50000
```

The total number of the images

```
{airplane,automobile,bird,cat,deer,dog,frog,horse,ship,truck}
```

```
Length[trainingData]
```

Actually, we employ the last 1000 images as testing set, therefore the training set,

```
actualTrainingSet = RandomSample[trainingData,49000];
```

and the testing set is,

```
actualTestingSet = Complement[trainingData,actualTrainingSet];
```

The structure of the convolutional network can be seen in Table 15.3. The network has a 3 dimensional tensor since one image has 32×32 pixel size and three RGB colors, consequently the input size is 3×32×32. The output is a ten dimensional vector corresponding the ten different classes. The *i*-th element of the output vector shows the probability of the association of the input image with the *i*-th class.

Let us define the network, see Table 15.3,

```
lenet = NetChain[
 {ConvolutionLayer[20,5],Ramp,PoolingLayer[2,2],
 ConvolutionLayer[50,5],Ramp,PoolingLayer[2,2],
 FlattenLayer[],500,Ramp,10,SoftmaxLayer[]},
 "Output" → NetDecoder[{"Class",classes}],
 "Input" → NetEncoder[{"Image",{32,32}}]
 ]
```

For validation set we use the 20% of the training set. First let us carry out the training with CPU.

Table 15.3 Structure of the convolutional neural network

NetChain[			image	]
		Input	3-tensor (size: 3 × 32 × 32)	
	1	ConvolutionLayer	3-tensor (size: 20 × 28 × 28)	
	2	Ramp	3-tensor (size: 20 × 28 × 28)	
	3	PoolingLayer	3-tensor (size: 20 × 14 × 14)	
	4	ConvolutionLayer	3-tensor (size: 50 × 10 × 10)	
	5	Ramp	3-tensor (size: 50 × 10 × 10)	
	6	PoolingLayer	3-tensor (size: 50 × 5×5)	
	7	FlattenLayer	vector (size: 1250)	
	8	LinearLayer	vector (size: 500)	
	9	Ramp	vector (size: 500)	
	10	LinearLayer	vector (size: 10)	
	11	SoftmaxLayer	vector (size: 10)	
		Output	class	
		(uninitialized)		

```
AbsoluteTiming[
 trained=NetTrain[lenet,actualTrainingSet,
 ValidationSet → Scaled[0.2],MaxTrainingRounds → 5];]
```

```
{1538.4,Null}
```

Let us pick up one image from the testing set randomly

```
images=RandomSample[Keys[actualTestingSet],1]
```

Using our trained network we can classify this image,

```
trained[images,"Probabilities"]
```

```
{(|airplane → 0.0000499916,
 automobile → 0.497519,bird → 4.10959 × 10^-6,cat → 0.000122994,
 deer → 3.23379 × 10^-7,dog → 0.000022407,frog → 4.71051 × 10^-6,
 horse → 0.0000508431,ship → 0.0149342,truck → 0.487292|)}
```

Our classifier identifies this object as an automobile with the highest probability (0.497519), however gives a similar chance for being classified as a truck (0.487292).

Let us check more test images

```
images=RandomSample[Keys[actualTestingSet],10]
```

```
trained[images]
```

```
{cat,dog,airplane,automobile,dog,dog,cat,frog,frog,frog}
```

Now we carry out the training with GPU

```
AbsoluteTiming[
 trained = NetTrain[lenet, actualTrainingSet,
 ValidationSet → Scaled[0.2], MaxTrainingRounds → 5,
 TargetDevice → "GPU"];]
```

```
{128.438, Null}
```

We can reach more than ten times speed up. Considering these two neural network examples, one can see that increasing task size results increasing speed-up.

Checking the test image

trained[{  },"Probabilities"]

```
{(|airplane → 0.0034147,
 automobile → 0.579832, bird → 0.000021244, cat → 0.0000830502,
 deer → 0.0000600025, dog → 0.0000127508, frog → 6.02948 × 10^-7,
 horse → 0.00050736, ship → 0.00266856, truck → 0.4134|)}
```

Remark Images here with improved resolution are illustration however the computation has been carried out with the original images of the CIFAR-10 database.

15.6.2 Image Processing with GPU

The parametrization technique is simple and effective but it is constrained to a limited number of built in functions. Extension can be achieved by using CUDA platform providing functions can be written in C -type compiler languages. These functions then will be linked to a computing system like *Mathematica* via CUDALink. Therefore this method is less efficient than the parametrization technique. Sometimes, surprisingly, if the CPU of our computer is much efficient than its GPU, then the considered function could be executed faster on CPU than on GPU. Here we illustrate this situation in order the call attention to the pitfalls of the GPU application, too.

Let us load CUDA platform,

```
Needs["CUDALink`"]
```

Checking whether our system can support this platform,

```
CUDAQ[]
```

```
True
```

Let us get information about our GPU system,

```
CUDAInformation[]
```

```
{1 → {Name → GeForce GT 630,Clock Rate → 875500,
Compute Capabilities → 3.,GPU Overlap - > 1,
Maximum Block Dimensions → {1024,1024,64},
Maximum Grid Dimensions → {2147483647,65535,65535},
Maximum Threads Per Block → 1024,
Maximum Shared Memory Per Block → 49152,
Total Constant Memory → 65536,Warp Size → 32,
Maximum Pitch → 2147483647,Maximum Registers Per Block - > 65536,
Texture Alignment → 512,Multiprocessor Count → 1, Core Count → 32,
Can Map Host Memory → True,
Compute Mode → Default,Texture1D Width → 65536,
Texture2D Width → 65536,Texture2D Height → 65536,
Texture3DWidth → 4096,Texture3DHeight → 4096,
Texture3D Depth → 4096,Texture2D Array Width → 16384,
Texture2D Array Height → 16384,Texture2D Array Slices → 2048,
Surface Alignment → 512,Concurrent Kernels → True,
ECC Enabled → False,TCC Enabled → False,
Total Memory → 2147483648}}
```

The task is to convolve an RGB image, see Fig. 15.12
Our kernel is an edge detection filter

$$M1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix};$$

Employing CPU, we get (Fig. 15.13)

```
AbsoluteTiming[ImageConvolve[im,M1]]
```

Now let us employ GPU via CUDA function (Fig. 15.14)

```
Needs["CUDALink"]
```

```
AbsoluteTiming[CUDAImageConvolve[im,M1]]
```

The time is somewhat longer, but the quality of the result is definitely better. The explanation is in Tables 15.4. This table shows that our system has a strong CPU, however its average performance index is low, since the low efficiency of the GPU subsystem is the bottle neck.

im=

Fig. 15.12 Budapest panorama with the Danube

{0.0464621,

}

Fig. 15.13 Budapest panorama with the Danube after convolution filtering via CPU

{0.155772,

}

Fig. 15.14 Budapest panorama with the Danube after convolution filtering via CPU

It means that when we have a corresponding built-in function and small task, in addition our GPU is not strong comparing to the CPU then using the built-in function with CPU can be better than employing CUDALink function.

15.7 Applications

The type of parallel jobs can be *task parallel* (embarrassingly parallel) and *data parallel*. In case of task parallel, multiple workers work on different parts of the problem without communication between each other. The result is independent of the execution order. Let us see a geodesics example.

15.7.1 3D Ranging Using the Dixon Resultant

Suppose three stations are given with known coordinates (x_i, y_i, z_i) and their distances, s_i from a point in 3D space. We are looking for the coordinates of the unknown point (x_0, y_0, z_0). (Fig. 15.15).

The corresponding distance observation equations,

$$\texttt{e1} = (\texttt{x}_1 - \texttt{x0})^2 + (\texttt{y}_1 - \texttt{y0})^2 + (\texttt{z}_1 - \texttt{z0})^2 - \texttt{s}_1^2\texttt{//Expand}$$

$$\texttt{x0}^2 + \texttt{y0}^2 + \texttt{z0}^2 - \texttt{s}_1^2 - 2\texttt{x0x}_1 + \texttt{x}_1^2 - 2\texttt{y0y}_1 + \texttt{y}_1^2 - 2\texttt{z0z}_1 + \texttt{z}_1^2$$

$$\texttt{e2} = (\texttt{x}_2 - \texttt{x0})^2 + (\texttt{y}_2 - \texttt{y0})^2 + (\texttt{z}_2 - \texttt{z0})^2 - \texttt{s}_2^2\texttt{//Expand}$$

$$\texttt{x0}^2 + \texttt{y0}^2 + \texttt{z0}^2 - \texttt{s}_2^2 - 2\texttt{x0x}_2 + \texttt{x}_2^2 - 2\texttt{y0y}_2 + \texttt{y}_2^2 - 2\texttt{z0z}_2 + \texttt{z}_2^2$$

$$\texttt{e3} = (\texttt{x}_3 - \texttt{x0})^2 + (\texttt{y}_3 - \texttt{y0})^2 + (\texttt{z}_3 - \texttt{z0})^2 - \texttt{s}_3^2\texttt{//Expand}$$

$$\texttt{x0}^2 + \texttt{y0}^2 + \texttt{z0}^2 - \texttt{s}_3^2 - 2\texttt{x0x}_3 + \texttt{x}_3^2 - 2\texttt{y0y}_3 + \texttt{y}_3^2 - 2\texttt{z0z}_3 + \texttt{z}_3^2$$

Table 15.4 Details of the effectivity of the different system components

Component	What is rated	Subscore	Base score
Processor	Calculations per second	7.8	
Memory (RAM)	Memory operations per second	7.9	
Graphics	Desktop performance for Windows Aero	6.8	
Gaming graphics	3D business and gaming graphics performance	6.8	Determined by lowest subscore
Primary hard disk	Disk data transfer rate	7.9	

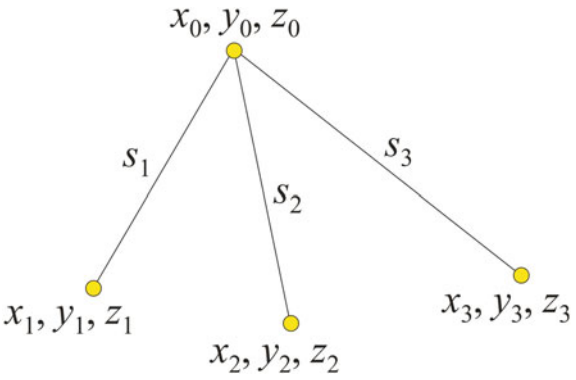


Fig. 15.15 Spacial ranging 3 Point Problem

Instead of trying to solve the original system directly, first we shall transform this system into a linear system of two equations containing $z0$ as parameter. With simple subtractions, we get

```
q = -{e1 - e2, e2 - e3}//Expand
```

$$\{s_1^2 - s_2^2 + 2x0x_1 - x_1^2 - 2x0x_2 + x_2^2 + 2y0y_1 - y_1^2 - 2y0y_2 + y_2^2 + 2z0z_1 - z_1^2 - 2z0z_2 + z_2^2, s_2^2 - s_3^2 + 2x0x_2 - x_2^2 - 2x0x_3 + x_3^2 + 2y0y_2 - y_2^2 - 2y0y_3 + y_3^2 + 2z0z_2 - z_2^2 - 2z0z_3 + z_3^2\}$$

Now, we will write the equations in the following form,

$$eq_1 = \alpha_1 x0 + \beta_1 y0 + \delta_1 z0 + \varepsilon_1$$

$$x0\,\alpha_1 + y0\,\beta_1 + z0\,\delta_1 + \varepsilon_1$$

and

$$eq_2 = eq_1 /. \{1 \to 2\}$$

$$x0\,\alpha_2 + y0\,\beta_2 + z0\,\delta_2 + \varepsilon_2$$

The corresponding coefficients α_i, β_i, δ_i are

```
coeffs0 = Map[({Coefficient[#,x0],Coefficient[#,y0],
 Coefficient[#,z0]}//Factor)&,q]
```

$$\{\{2(x_1 - x_2), 2(y_1 - y_2), 2(z_1 - z_2)\}, \{2(x_2 - x_3), 2(y_2 - y_3), 2(z_2 - z_3)\}\}$$

and the constant values, ε_i

```
coeffs1 =
 Table[{q[[i]] - coeffs0[[i]].{x0,y0,z0}//Simplify},{i,1,2}]
```

```
{{s1^2 - s2^2 - x1^2 + x2^2 - y1^2 + y2^2 - z1^2 + z2^2}{s2^2 - s3^2 - x2^2 + x3^2 - y2^2 + y3^2
- z2^2 + z3^2}}
```

then unify them,

```
coeffs = Table[Union[coeffs0[[i]],coeffs1[[i]]],{i,1,2}]
```

$$\{\{2(x_1-x_2),2(y_1-y_2),2(z_1-z_2),s_1^2-s_2^2-x_1^2+x_2^2-y_1^2+y_2^2-z_1^2+z_2^2\},$$
$$\{2(x_2-x_3),2(y_2-y_3),2(z_2-z_3),s_2^2-s_3^2-x_2^2+x_3^2-y_2^2+y_3^2-z_2^2+z_3^2\}\}$$

These expressions can be assigned to the coefficients in rule form,

```
coeffsn = Flatten[Table[Inner[#1 → #2&,{αi,βi,δi,εi]},
 coeffs[[i]],List],{i,1,2}]]
```

$$\{\alpha_1 \to 2(x_1-x_2),\ \beta_1 \to 2(y_1-y_2),\ \delta_1 \to 2(z_1-z_2),$$
$$\in_1 \to s_1^2-s_2^2-x_1^2+x_2^2-y_1^2+y_2^2-z_1^2+z_2^2,\ \alpha_2 \to 2(x_2-x_3),$$
$$\beta_2 \to 2(y_2-y_3),\ \delta_2 \to 2(z_2-z_3),$$
$$\in_2 \to s_2^2-s_3^2-x_2^2+x_3^2-y_2^2+y_3^2-z_2^2+z_3^2\}$$

The symbolic solution of the *3D Ranging Problem* can be solved via symbolic computation, employing the Dixon resultant.

The sequential solution in *Mathematica* is the following:

```
≪ Resultant`Dixon`
```

Now, we can eliminate $y0$ and $z0$ simultaneously,

```
AbsoluteTiming[solx0D =
 DixonResultant[eq1,eq2,e3},{y0,z0},{Y0,Z0}]//Simplify;]
{1.13882,Null}
```

```
solx0D
```

$$(-\beta_2\delta_1+\beta_1\delta_2)(x0^2\beta_2^2\delta_1^2-s_3^2\beta_2^2\delta_1^2-2x0x_3\beta_2^2\delta_1^2+x_3^2\beta_2^2\delta_1^2+y_3^2\beta_2^2\delta_1^2+z_3^2\beta_2^2\delta_1^2$$
$$+x0^2\alpha_2^2(\beta_1^2+\delta_1^2)-2x0^2\beta_1\beta_2\delta_1\delta_2+2s_3^2\beta_1\beta_2\delta_1\delta_2+4x0x_3\beta_1\beta_2\delta_1\delta_2-2x_3^2\beta_1\beta_2\delta_1\delta_2$$
$$-2y_3^2\beta_1\beta_2\delta_1\delta_2-2z_3^2\beta_1\beta_2\delta_1\delta_2+x0^2\beta_1^2\delta_2^2-s_3^2\beta_1^2\delta_2^2-2x0x_3\beta_1^2\delta_2^2+x_3^2\beta_1^2\delta_2^2$$
$$+y_3^2\beta_1^2\delta_2^2+z_3^2\beta_1^2\delta_2^2+x0^2\alpha_1^2(\beta_2^2+\delta_2^2)+2z_3\beta_2^2\delta_1\in_1-2z_3\beta_1\beta_2\delta_2\in_1-2y_3\beta_2\delta_1\delta_2\in_1$$

$$+2y_3\beta_1\delta_2^2\in_1+\beta_2^2\in_1^2+\delta_2^2\in_1^2-2z_3\beta_1\beta_2\delta_1\in_2+2y_3\beta_2\delta_1^2\in_2+2z_3\beta_1^2\delta_2\in_2$$
$$-2y_3\beta_1\delta_1\delta_2\in_2-2\beta_1\beta_2\in_1\in_2-2\delta_1\delta_2\in_1\in_2+\beta_1^2\in_2^2+\delta_1^2\in_2^2$$
$$-2x0\alpha_2(-y_3\beta_2\delta_1^2+y_3\beta_1\delta_1\delta_2+z_3\beta_1(\beta_2\delta_1-\beta_1\delta_2)+x0\alpha_1(\beta_1\beta_2+\delta_1\delta_2)$$
$$+\beta_1\beta_2\in_1+\delta_1\delta_2\in_1-\beta_1^2\in_2-\delta_1^2\in_2)+2x0\alpha_1(z_3\beta_2(\beta_2\delta_1-\beta_1\delta_2)+y_3\delta_2(-\beta_2\delta_1+\beta_1\delta_2)$$
$$+\beta_2^2\in_1+\delta_2^2\in_1-\beta_1\beta_2\in_2-\delta_1\delta_2\in_2))$$

This is a second order polynomial for $x0$,

```
Exponent[solx0D,x0]
```

```
2
```

Similarly, eliminating $x0$ and $z0$ simultaneously,

```
Clear[X0]
```

```
AbsoluteTiming[soly0D=
 DixonResultant[{eq1,eq2,e3},{x0,z0},{X0,Z0}]//Simplify;]
{1.13374,Null}
```

```
soly0D
```

$$-(\alpha_2\delta_1-\alpha_1\delta_2)(\alpha_2^2(y0^2\beta_1^2+(y0^2-s_3^2+x_3^2-2y0y_3+y_3^2+z_3^2)\delta_1^2+2z_3\delta_1\in_1$$
$$+\in_1^2+2y0\beta_1(z_3\delta_1+\in_1))+2x_3\alpha_1\delta_2(-y0\beta_2\delta_1+y0\beta_1\delta_2+\delta_2\in_1-\delta_1\in_2)$$
$$+(y0\beta_2\delta_1-y0\beta_1\delta_2-\delta_2\in_1+\delta_1\in_2)^2+\alpha_1^2(y0^2\beta_2^2+(y0^2-s_3^2+x_3^2-2y0y_3+y_3^2$$
$$+z_3^2)\delta_2^2+2z_3\delta_2\in_2+\in_2^2+2y0\beta_2(z_3\delta_2+\in_2))-2\alpha_2(x_3\delta_1(-y0\beta_2\delta_1+y0\beta_1\delta_2$$
$$+\delta_2\in_1-\delta_1\in_2)+\alpha_1(y0^2\delta_1\delta_2-s_3^2\delta_1\delta_2+x_3^2\delta_1\delta_2-2y0y_3\delta_1\delta_2+y_3^2\delta_1\delta_2+z_3^2\delta_1\delta_2$$
$$+y0\beta_2\in_1+\in_1\in_2+y0\beta_1(y0\beta_2+z_3\delta_2+\in_2)+z_3(y0\beta_2\delta_1+\delta_2\in_1+\delta_1\in_2))))$$

This is a second order polynomial for $y0$,

```
Exponent[soly0D,y0]
```

```
2
```

To get $z0$, we can eliminate $x0$ and $y0$ simultaneously,

```
AbsoluteTiming[solz0D=
 DixonResultant[{eq1,eq2,e3},{x0,y0},{X0,Y0}]//Simplify;]
```

```
{1.09857,Null}
```

```
solz0D
```

$$-(\alpha_2\beta_1-\alpha_1\beta_2)(\alpha_2^2(z0^2-S_3^2+X_3^2+Y_3^2-2z0z_3+z_3^2)\beta_1^2+2y_3\beta_1(z0\delta_1+\in_1)$$
$$+(z0\delta_1+\in_1)^2)+2x_3\alpha_1\beta_2(\beta_2(z0\delta_1+\in_1)-\beta_1(z0\delta_2+\in_2))+(\beta_2(z0\delta_1+\in_1)$$
$$-\beta_1(z0\delta_2+\in_2)^2)+\alpha_1^2((z0^2-S_3^2+X_3^2+Y_3^2-2z0z_3+z_3^2)\beta_2^2+2y_3\beta_2(z0\delta_2+\in_2)$$
$$+(z0\delta_2+\in_2)^2)-2\alpha_2(X_3\beta_1(\beta_2(z0\delta_1+\in_1)-\beta_1(z0\delta_2+\in_2))$$
$$+\alpha_1((z0\delta_2+\in_2)(Y_3\beta_2+z0\delta_2+\in_2)+\beta_1((z0^2-S_3^2+X_3^2+Y_3^2-2z0z_3+z_3^2)\beta_2$$
$$+Y_3(z0\delta_2+\in_2)))))$$

This is a second order polynomial for $y0$,

```
Exponent[solz0D,z0]
```

```
2
```

These computations can be evaluated in parallel too.

```
xyz = {{y0,z0},{x0,z0},{x0,y0}}; XYZ = {{Y0,Z0},{X0,Z0},{X0,Z0}};
```

```
sys = {eq1,eq2,e3};
```

```
DistributeDefinitions[DixonResultant,xyz,XYZ,sys];
```

```
{solxPD,solyPD,solzPD}=Map[Simplify[#]&,
Parallelize[MapThread[DixonResultant[sys,#1,#2]&,{xyz,XYZ}]]];
//AbsoluteTiming
```

```
{1.64605,Null}
```

The results are the same,

```
solx0D - solxPD//Simplify
```

```
0
```

```
soly0D - solyPD//Simplify
```

```
0
```

```
solz0D - solzPD//Simplify
```

```
0
```

The load is too small, however; the net win is (1.14 + 1.13 + 1.1)/1.65 = 2.042

So using parallel computation even for a symbolic case can reduce the running time by half!

15.7.2 Reducing Colors via Color Approximation

In some problems data being analyzed is too much for one processor. In this case each worker operates on part of the data, and they may or may not communicate with each other. To illustrate, let's consider an example in digital image processing.

On systems with 24-bit color displays, true color images can display up to 16,777,216 (i.e., 2^{24}) colors. On systems with lower screen bit depths, true color images are still displayed reasonably well, using color approximation. Color approximation is the process by which the software chooses replacement colors when direct matches cannot be found. One of the methods to carry out such color approximation is *color quantization*.

An important concept in image quantization is the RGB color cube. The RGB color cube is a three-dimensional array of all the colors that are defined for a particular data type (Fig. 15.16)

Quantization involves dividing the RGB color cube into smaller boxes, and then mapping all colors that fall within each box to the color value at the center of that box.

If the actual image is big in size, one may divide the image into parts, and this quantization process can be carried out in parallel on these image segments. Let us consider an airborne digital photo of Boston (Fig. 15.17).

```
pBoston =
```

Its size is

```
ImageDimensions[pBoston]
```

```
{4481,2881}
```

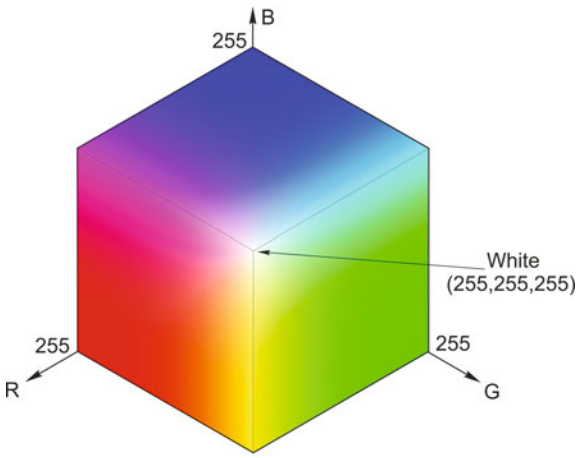


Fig. 15.16 RGB color cube

Fig. 15.17 Digital photo of Boston

So the number of pixels is their product, 12909761.

Let us divide the RGB cube into 10 "rectangular cuboids" (rectangular solids), and carry out the color reduction by color quantization,

```
pQ = ColorQuantize[pBoston,10];//Timing

{7.69085,Null}
```

The result is:

(See Fig. 15.18)

Now let us divide the picture into eight parts,

```
4481/4
1120.25

2881/2
1440.5

pP = ImagePartition[pBoston,{1120,1440}];//Timing

{0.0624004,Null}
```

Fig. 15.18 Digital photo of Boston after color reduction

```
pPF = Flatten[pP];

Magnify[pPF, 0.2]
```

Now let us compute the color approximation of each sub-image simultaneously in parallel, using `ParallelSubmit` (Fig. 15.19,15.20)

```
LaunchKernels[2$ProcessorCount]//Quiet;

DistributeDefinitions[ColorQuantize, pPF];

pS = WaitAll[Map[ParallelSubmit[ColorQuantize[#, 10]]&, pPF]];//
  AbsoluteTiming
{1.66377, Null}

Dimensions[pS]

{8}
```

We put the parts together,

```
pR = ImageAssemble[Partition[pS, 4]];//AbsoluteTiming

{0.0961564, Null}
```

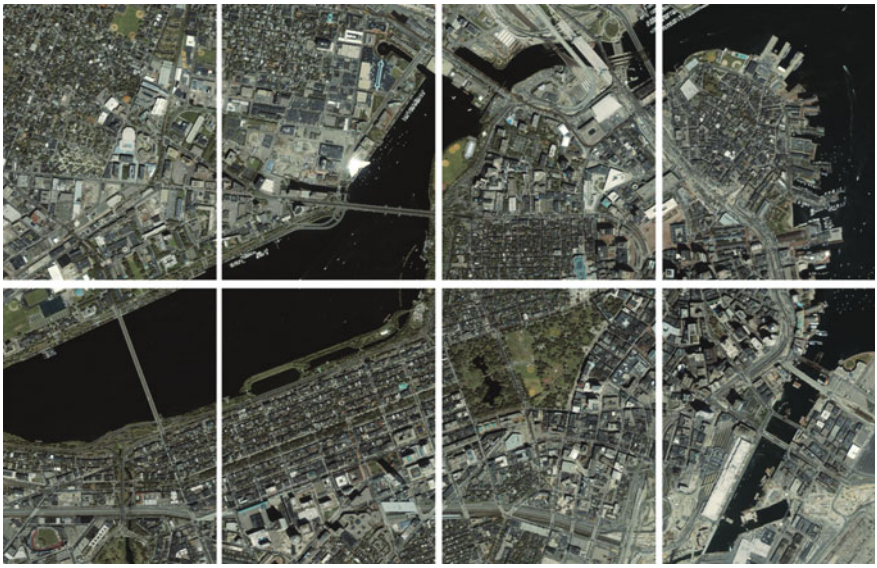


Fig. 15.19 The partitioned image of Boston

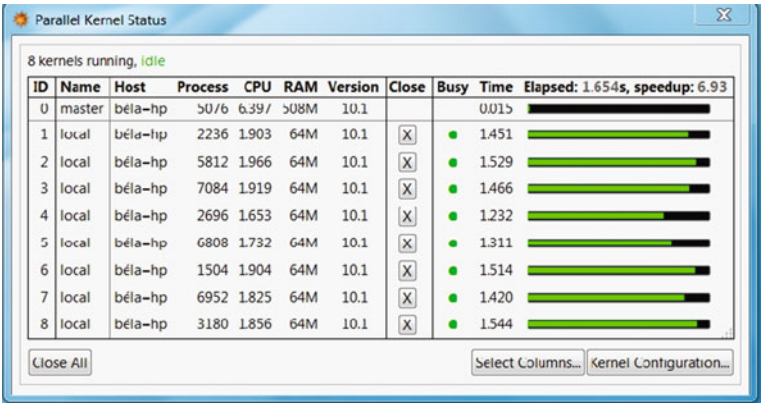


Fig. 15.20 Status of the kernels in case of parallel evaluation of color quantization using ParallelSubmit

```
ImageDimensions[pR]
```

```
{4480,2880}
```

The result is:

(See Fig. 15.21)

The actual net win in running time, 5.865/(0.031 + 1.664 + 0.096) = 3.27471.

The parallelization reduced the running time by a factor of 3.3, which means down to 30% of the traditional computation.

Fig. 15.21 Digital photo of Boston after color reduction employing parallel computation

15.8 Problem

15.8.1 Photogrammetric Positioning by Gauss-Jacobi Method

The relation between the coordinates of a ground point (X_i, Y_i, Z_i) and the coordinates of the corresponding point on the photo plane (x_i, y_i) can be represented by the following linear transformation

$$\begin{pmatrix} x_i - \eta_0 \\ y_i - \xi_0 \\ -f \end{pmatrix} = k_i R \begin{pmatrix} X_i - X_0 \\ Y_i - Y_0 \\ Z_i - Z_0 \end{pmatrix}$$

where

- η_0, ξ_0 are the coordinates of the perspective center on the photo plane,
- f the focal length,
- k_i the scaling factor,
- R the rotation matrix,
- X_0, Y_0, Z_0 the coordinates of the perspective center in the ground system. (Fig. 15.22).

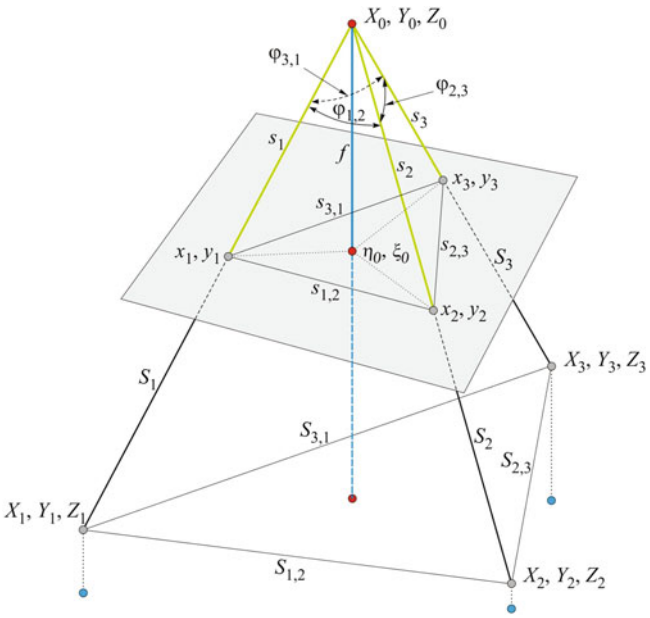


Fig. 15.22 Photogrammetric 3D resection

In general the focal length f is known, and the parameters of the transformation should be determined on the basis of the coordinates of 3 corresponding plane-ground points.

The problem with this representation of the transformation is that the scaling factors k_i differ for the different points, therefore it is reasonable to eliminate them. Let us express the equations of the transformation for the 3 different coordinates,

$$x_i - \eta_0 = k_i\big(R_{1,1}(X_i - X_0) + R_{1,2}(Y_i - Y_0) + R_{1,3}(Z_i - Z_0)\big),$$

$$y_i - \xi_0 = k_i(R_{21}(X_i - X_0) + R_{22}(Y_i - Y_0) + R_{23}(Z_i - Z_0)),$$

$$-f = k_i(R_{21}(X_i - X_0) + R_{22}(Y_i - Y_0) + R_{23}(Z_i - Z_0)).$$

Let us divide the first and second equation by the third one and rearrange them and introducing $r_{ij} = R_{i,j}$ we get

$$x_i = \eta_0 - f\frac{(r_{11}(X_i - X_0) + r_{12}(Y_i - Y_0) + r_{13}(Z_i - Z_0))}{(r_{31}(X_i - X_0) + r_{32}(Y_i - Y_0) + r_{33}(Z_i - Z_0))},$$

$$y_i = \xi_0 - f\frac{(r_{21}(X_i - X_0) + r_{22}(Y_i - Y_0) + r_{23}(Z_i - Z_0))}{(r_{31}(X_i - X_0) + r_{32}(Y_i - Y_0) + r_{33}(Z_i - Z_0))}.$$

Next, we express the elements of the rotation matrix with the elements of the skew matrix,

```
Clear["Global`*"]
```

```
S = {{0, -c, b}, {c, 0, -a}, {-b, a, 0}};
```

The rotation matrix is

$$R = (I_3 - S)^{-1}(I_3 + S),$$

where I_3 is a 3×3 identity matrix.

```
I3 = IdentityMatrix[3];
```

```
R = Inverse[(I3 - S)].(I3 + S)//Simplify; MatrixForm[R]
```

$$\begin{pmatrix} \frac{1+a^2-b^2-c^2}{1+a^2+b^2+c^2} & \frac{2ab-2c}{1+a^2+b^2+c^2} & \frac{2(b+ac)}{1+a^2+b^2+c^2} \\ \frac{2(ab+c)}{1+a^2+b^2+c^2} & \frac{1-a^2+b^2-c^2}{1+a^2+b^2+c^2} & -\frac{2(a-bc)}{1+a^2+b^2+c^2} \\ \frac{2(-b+ac)}{1+a^2+b^2+c^2} & \frac{2(a+bc)}{1+a^2+b^2+c^2} & \frac{1-a^2-b^2+c^2}{1+a^2+b^2+c^2} \end{pmatrix}$$

The numerator of the elements of the rotation matrix,

```
Table[ri,j = Numerator[R[[i,j]]],{i,1,3},{j,1,3}]
```

```
{{1 + a^2 - b^2 - c^2, 2ab - 2c, 2(b + ac)},
 {2(ab + c), 1 - a^2 + b^2 - c^2, -2(a - bc)},
 {2(-b + ac), 2(a + bc), 1 - a^2 - b^2 + c^2}}
```

Considering six corresponding points on the photo plane and on the ground,

```
n = 6;
```

```
Clear[X]
```

the prototypes of the model equations are

```
ei = (xi - η0)(r3,1(Xi - X0) + r3,2(Yi - Y0) + r3,3(Zi - Z0))
  + f(r1,1(Xi - X0) + r1,2(Yi - Y0) + r1,3(Zi - Z0))//Expand
```

```
- fX0 - a^2 fX0 + b^2 fX0 + c^2 fX0 - 2abfY0 + 2cfY0 - 2bfZ0 - 2acfZ0
 - 2bX0η0 + 2acX0η0 + 2aY0η0 + 2bcY0η0 + Z0η0 - a^2 Z0η0 - b^2 Z0η0
 + c^2 Z0η0 + 2bX0x_i - 2acX0x_i - 2aY0x_i - 2bcY0x_i - Z0x_i + a^2 Z0x_i
 + b^2 Z0x_i - c^2 Z0x_i + fX_i + a^2 fX_i - b^2 fX_i - c^2 fX_i + 2bη0X_i - 2acη0X_i
 - 2bx_i X_i + 2acx_i X_i + 2abfY_i - 2cfY_i - 2aη0Y_i - 2bcη0Y_i + 2ax_i Y_i
 + 2bcx_i Y_i + 2bfZ_i + 2acfZ_i - η0Z_i + a^2 η0Z_i + b^2 η0Z_i - c^2 η0Z_i + x_i Z_i
 - a^2 x_i Z_i - b^2 x_i Z_i + c^2 x_i Z_i
```

and

```
e_{i+n} = (y_i - ξ0)(r_{3,1}(X_i - X0) + r_{3,2}(Y_i - Y0) + r_{3,3}(Z_i - Z0)) +
 f(r_{2,1}(X_i - X0) + r_{2,2}(Y_i - Y0) + r_{2,3}(Z_i - Z0))//Expand
```

```
-2abfX0-2cfX0-fY0+a^2 fY0-b^2 fY0+c^2 fY0+2afZ0-2bcfZ0-2bX0ξ0
+2acX0ξ0+2aY0ξ0+2bcY0ξ0+Z0ξ0-a^2 Z0ξ0-b^2 Z0ξ0+c^2 Z0ξ0+2abfX_i
+2cfX_i+2bξ0X_i-2acξ0X_i+2bX0y_i-2acX0y_i-2aY0y_i-2bcY0y_i-Z0y_i
+a^2 Z0y_i+b^2 Z0y_i-c^2 Z0y_i-2bX_i y_i+2acX_i y_i+fY_i-a^2 fY_i+b^2 fY_i-c^2 fY_i
-2aξ0Y_i-2bcξ0Y_i+2ay_i Y_i+2bcy_i Y_i-2afZ_i+2bcfZ_i-ξ0Z_i+a^2 ξ0Z_i
+b^2 ξ0Z_i-c^2 ξ0Z_i+y_i Z_i-a^2 y_i Z_i-b^2 y_i Z_i+c^2 y_i Z_i
```

```
sysn = Table[{e_i, e_{i+n}}/.i → j, {j,1,n}]//Flatten;
Short[sysn,10]
```

```
{-fX0-a^2 fX0+b^2 fX0+c^2 fX0-2abfY0+2cfY0-2bfZ0-2acfZ0-2bX0η0
+2acX0η0+2aY0η0+2bcY0η0+Z0η0-a^2 Z0η0-b^2 Z0η0+c^2 Z0η0+2bX0x_1
-2acX0x_1-2aY0x_1-2bcY0x_1-Z0x_1+≪10≫+2bη0X_1-2acη0X_1-2bx_1 X_1
+2acx_1 X_1+2abfY_1-2cfY_1-2aη0Y_1-2bcη0Y_1+2ax_1 Y_1+2bcx_1 Y_1+2bfZ_1
+2acfZ_1-η0Z_1+a^2 η0Z_1+b^2 η0Z_1-c^2 η0Z_1+x_1 Z_1-a^2 x_1 Z_1-b^2 x_1 Z_1+c^2 x_1 Z_1,
≪1≫,≪8≫,≪1≫,≪1≫}
```

Let us consider the following data: the ground coordinates, (X_i, Y_i, Z_i).
The coordinate values of the ground points,

```
dataXYZ =
 {X_1 → -460., Y_1 → -920, Z_1 → -153, X_2 → 460., Y_2 → -920., Z_2 → 0.,
 X_3 → -460., Y_3 → 0, Z_3 → 0, X_4 → 460., Y_4 → 0, Z_4 → 153,
 X_5 → -460., Y_5 → 920., Z_5 → -153., X_6 → 460., Y_6 → 920, Z_6 → 0};
```

and the image coordinates,

```
dataxy =
 {x_1 → 18996.171, y_1 → -64147.679, x_2 → 113471.749, y_2 → -73694.266,
 x_3 → 16504.609, y_3 → 16331.649, x_4 → 128830.826, y_4 → 21085.172,
 x_5 → 13716.588, y_5 → 106386.802, x_6 → 120577.473, y_6 → 128214.823};
```

In our case the focal length f = 153 000.

Then the equations in numerical form,

```
sysnN = sysn/.dataXYZ/.dataxy/.f → 153000;
```

```
Short[sysnN,20]
```

```
{−7.32864 × 10^7−3.4953 × 10^7a − 6.74736 × 10^7a^2−2.93415 × 10^7b
 −281520000ab + 7.32864 × 10^7b^2 + 281520000c − 6.42945 × 10^7ac
 −3.4953 × 10^7bc + 6.74736 × 10^7c^2−153000X0 − 153000a^2X0
 + 37992.3bX0 + 153000b^2X0 − 37992.3acX0 + 153000c^2X0
 −37992.3aY0 − 306000abY0 + 306000cY0 − 37992.3bcY0
 −18996.2Z0 + 18996.2a^2Z0 − 306000bZ0 + 18996.2bη0 − 153a^2η0
 −920.bη0 − 153b^2η0 + 920.acη0 + 1840bcη0 + 153c^2η0 − 2bX0η0
 + 2acX0η0 + 2aY0η0 + 2bcY0η0 + Z0η0 − a^2Z0η0 − b^2Z0η0 + c^2Z0η0,^2
 −1.30945 × 10^8 + 1.6485 × 10^8a + ≪ 57 ≫ + c^2Z0ξ0, ≪ 1 ≫,
 ≪ 6 ≫, ≪ 1 ≫, ≪ 1 ≫, 140760000 + 2.35915 * 10^8a−
 140760000a^2−1.17958 × 10^8b + 1.4076 × 10^8ab + 140760000b^2
 + 1.4076 × 10^8c + 1.17958 × 10^8ac + ≪ 38 ≫ + 2acX0ξ0
 + 2aY0ξ0 + 2bcY0ξ0 + Z0ξ0 − a^2Z0ξ0 − b^2Z0ξ0 + c^2Z0ξ0}
```

```
Length[%]
```

```
12
```

This means we have twelve equations, but only eight unknown variables (a, b, c, $X0$, $Y0$, $Z0$, $\eta 0$, $\xi 0$), consequently our system is overdetermined.

```
χ = {a,b,c,X0,Y0,Z0,η0,ξ0};
```

Let us employ the *Gauss-Jacobi combinatorial algorithm*. We have 6 corresponding points and every four points represent four equations,

```
n = 6; m = 4;
```

The number of the combinations,

```
mn = Binomial[n,m]
```

```
15
```

And the actual combinations,

```
qs = Partition[Map[#&,Flatten[Subsets[Range[n],{m}]]],m]
```

```
{{1,2,3,4},{1,2,3,5},{1,2,3,6},{1,2,4,5},{1,2,4,6},
 {1,2,5,6},{1,3,4,5},{1,3,4,6},{1,3,5,6},{1,4,5,6},
 {2,3,4,5},{2,3,4,6},{2,3,5,6},{2,4,5,6},{3,4,5,6}}
```

We use data preparation, namely we apply a mask {1, 2, 3, 4} for each combination,

```
datapXYZ =
 Table[Map[Select[dataXYZ,MemberQ[qs[[i]],#[[1,2]]]&]/.
 {#[[1]] → 1,#[[2]] → 2,#[[3]] → 3,#[[4]] → 4}&,
 {qs[[i]]}],{i,1,mn}];
```

For example, for the subset {1, 2, 3, 6},

```
datapXYZ[[3]]
```

$$\{\{X_1 \to -460.,\ Y_1 \to -920,\ Z_1 \to -153,\ X_2 \to 460.,\ Y_2 \to -920.,\ Z_2 \to 0.,\ X_3 \to -460.,\ Y_3 \to 0,\ Z_3 \to 0,\ X_4 \to 460.,\ Y_4 \to 920,\ Z_4 \to 0\}\}$$

similarly,

```
datapxy =
 Table[Map[Select[dataxy,MemberQ[qs[[i]],#[[1,2]]]&]/.
 {#[[1]] → 1,#[[2]] → 2,#[[3]] → 3,#[[4]] → 4}&,
 {qs[[i]]}],{i,1,mn}];
```

and

```
datapXYZ[[3]]
```

$$\{\{x_1 \to 18996.171,\ y_1 \to -64147.679,\ x_2 \to 113471.749,\ y_2 \to -73694.266,\ x_3 \to 16504.609,\ y_3 \to 16331.649,\ x_4 \to 120577.473,\ y_4 \to 128214.823\}\}$$

To solve the square subsets, we need the equations of the determined problem, namely the $n = 4$ points problem,

```
e_{i+4} = (y_i - ξ0)(r_{3,1}(X_i - X0) + r_{3,2}(Y_i - Y0) + r_{3,3}(Z_i - Z0)) +
 f(r_{2,1}(X_i - X0) + r_{2,2}(Y_i - Y0) + r_{2,3}(Z_i - Z0))//Expand
```

$$-2abfX0 - 2cfX0 - fY0 + a^2fY0 - b^2fY0 + c^2fY0 + 2afZ0 - 2bcfZ0 - 2bX0ξ0 + 2acX0ξ0 + 2aY0ξ0 + 2bcY0ξ0 + Z0ξ0 - a^2Z0ξ0 - b^2Z0ξ0 + c^2Z0ξ0 + 2abfX_i + 2cfX_i + 2bξ0X_i - 2acξ0X_i + 2bX0y_i - 2acX0y_i - 2aY0y_i - 2bcY0y_i - Z0y_i + a^2Z0y_i + b^2Z0y_i - c^2Z0y_i - 2bX_iy_i + 2acX_iy_i + fY_i - a^2fY_i + b^2fY_i - c^2fY_i - 2aξ0Y_i - 2bcξ0Y_i + 2ay_iY_i + 2bcy_iY_i - 2afZ_i + 2bcfZ_i - ξ0Z_i + a^2ξ0Z_i + b^2ξ0Z_i - c^2ξ0Z_i + y_iZ_i - a^2y_iZ_i - b^2y_iZ_i + c^2y_iZ_i$$

```
sysn = Table[{e_i, e_{i+4}}/.i → j, {j, 1, 4}]//Flatten;
Short[sysn, 10]
```

$$\{-fX0 - a^2 fX0 + b^2 fX0 + c^2 fX0 - 2abfY0 + 2cfY0 - 2bfZ0 - 2acfZ0 - 2bX0\eta 0 + 2acX0\eta 0 + 2aY0\eta 0 + 2bcY0\eta 0 + Z0\eta 0 - a^2 Z0\eta 0 - b^2 Z0\eta 0 + c^2 Z0\eta 0 + 2bX0x_1 - 2acX0x_1 - 2aY0x_1 - 2bcY0x_1 - Z0x_1 + \ll 10 \gg + 2b\eta 0X_1 - 2ac\eta 0X_1 - 2bx_1X_1 + 2acx_1X_1 + 2abfY_1 - 2cfY_1 - 2a\eta 0Y_1 - 2bc\eta 0Y_1 + 2ax_1Y_1 + 2bcx_1Y_1 + 2bfZ_1 + 2acfZ_1 - \eta 0Z_1 + a^2\eta 0Z_1 + b^2\eta 0Z_1 - c^2\eta 0Z_1 + x_1Z_1 - a^2x_1Z_1 - b^2x_1Z_1 + c^2x_1Z_1, \ll 1 \gg, \ll 4 \gg, \ll 1 \gg, \ll 1 \gg\}$$

Then the solution of the subsets,

```
AbsoluteTiming[solGJ = MapThread[
 NSolve[sys/.Flatten[#1]/.Flatten[#2]/.f → 153000, χ]&,
 {datapXYZ, datapxy}]; ]
{73.866130, Null}
```

This computation takes a long time, therefore it is reasonable to employ parallel computation. Let us define all variables involved in the computation as parallel variables,

```
LaunchKernels[2$ProcessorCount]//Quiet;
```

```
DistributeDefinitions[sys, f, datapXYZ, datapxy, χ]
```

```
{sys, datapXYZ, datapxy, χ}
```

Then the parallel computation could be applied straight away, but here the application of the function `Parallelize` is reasonable, (Fig. 15.23)

```
AbsoluteTiming[solGJ = Parallelize[MapThread[
  NSolve[sys/.Flatten[#1]/.Flatten[#2]/.f → 153000, χ]&,
 {datapXYZ, datapxy}]]; ]
```

```
{22.198839, Null}
```

Now the running time is considerably shorter.

It can be seen that only seven threads were activated and their tasks needed different running times! Indeed, the solutions have different types and different lengths, for example

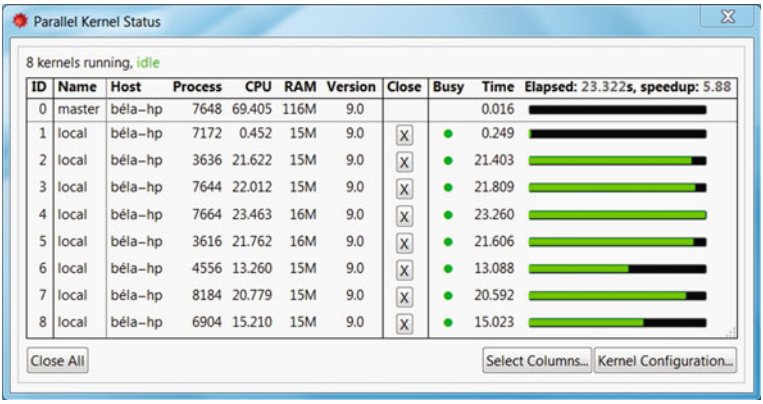


Fig. 15.23 Status of the kernels in case of parallel evaluation of the Gauss-Jacobi algorithm using **Parallelize**

```
solGJ[[1]]
```

```
{{a → 212.292, b → 213.217, c → -22.6786, X0 → 5220.9,
 Y0 → 5503.69, Z0 → 330.045, η0 → -528188., ξ0 → -550286.},
 {a → 2.23966, b → -2.23331, c → 0.0123486, X0 → 4738.68,
 Y0 → 5021.47, Z0 → 3229.65, η0 → -528188., ξ0 → -550286.},
 {a → 11.63, b → 11.8209, c → -52.4359, X0 → 22.2177,
 Y0 → 482.218, Z0 → -1369.61, η0 → 0.0120128, ξ0 → 0.0154296},
 {a → 0.0520854, b → -0.0546015, c → 0.0183141, X0 → -460.,
 Y0 → 0.000080254, Z0 → 1530., η0 → 0.0120128, ξ0 → 0.0154296}}
```

In order to select the admissible solutions from a subset, first we compute the minimum of the residual errors for each subset solution,

```
obj = Apply[Plus, Map[#^2 &, sys]];
```

```
objmin = Map[Min[Abs[#]] &,
 Table[obj/.dataXYZ/.dataxy/.solGJ[[i, j]]/.f → 153000,
 {i, 1, mn}, {j, 1, Length[solGJ[[i]]]}]];
```

then we select the solutions belonging to these minimums,

```
solGJS = Table[Select[solGJ[[i]],
 Abs[obj/.dataXYZ/.dataxy/.#/.f → 153000] == 
 objmin[[i]] &], {i, 1, mn}]
```

```
{{{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
    Y0 → 0.000080254, Z0 → 1530., η0 → 0.0120128, ξ0 → 0.0154296}},
 {{a → 0.0520853, b → −0.0546014, c → 0.0183141, X0 → −460.,
  Y0 → 0.000179128, Z0 → 1530., η0 → 0.0160779, ξ0 → 0.0396817}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → −0.0000230117, Z0 → 1530., η0 → 0.00908422, ξ0 → 0.0047149}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → 0.000021374, Z0 → 1530., η0 → −0.00226708, ξ0 → 0.00521362}},
 {{a → 0.0520854, b → −0.0546014, c → 0.0183141, X0 → −460.,
  Y0 → 0.00015106, Z0 → 1530., η0 → 0.0305765, ξ0 → 0.0287886}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → −0.00005401, Z0 → 1530., η0 → −0.0019466, ξ0 → −0.00323999}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → 0.0000930051, Z0 → 1530., η0 → 0.00217061, ξ0 → 0.0273753}},
 {{a → 0.0520854, b → −0.0546014, c → 0.0183141, X0 → −460.,
  Y0 → 0.000115357, Z0 → 1530., η0 → 0.0132181, ξ0 → 0.0155434}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → −0.0000117604, Z0 → 1530., η0 → −0.00941041, ξ0 → 0.0126929}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → 0., Z0 → 1530., η0 → 0.00455397, ξ0 → 0.0057907}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → 0.0000418975, Z0 → 1530., η0 → −0.0122799, ξ0 → 0.0194434}},
 {{a → 0.0520854, b → −0.0546014, c → 0.0183141, X0 → −460.,
  Y0 → 0.0000871336, Z0 → 1530., η0 → 0.0177359, ξ0 → 0.0148248}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → −0.000432414, Z0 → 1530., η0 → −0.0230733, ξ0 → −0.032863}},
 {{a → 0.0520855, b → −0.0546011, c → 0.0183142, X0 → −460.,
  Y0 → −0.0010589, Z0 → 1530., η0 → 0.0951372, ξ0 → −0.183949}},
 {{a → 0.0520854, b → −0.0546015, c → 0.0183141, X0 → −460.,
  Y0 → 0.000241822, Z0 → 1530., η0 → −0.00770075, ξ0 → 0.0280156}}}
```

The average values of the subset solutions is

```
solGJ = MapThread[#1 -> #2&, {χ, Map[Mean[#]&,
 Transpose[Table[Map[#[[2]]&, Flatten[solGJS[[i]], 1]], {i, 1, mn}]]]}]
```

```
{a → 0.0520854, b → −0.0546014, c → 0.0183141, X0 → −460.,
 Y0 → −0.0000379376, Z0 → 1530., η0 → 0.00959261, ξ0 → −0.000169176}
```

The actual net win in running time 73.9/22.2 = 3.33.

Bibliography

Awange JL, Grafarend EW, Paláncz B, Zaletnyik P (2011) Algebraic geodesy and geoinformatics. Springer, Heidelberg, Berlin

Buchberger B (1965) An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. Leopold-Franzens University, Innsbruck

Buse L, Elkadi M, Mourrain B (2000) Generalized resultants over unirational algebraic varieties. J Symbolic Comp 29:515–526

Chapra SC, Canale RP (1998) Numerical methods for engineers. McGraw-Hill, Boston

Chen CC, Stamos I (2007) Range image segmentation for modeling and object detection in urban scenes. In: 3DIM2007.1—Proceedings 6th international conference on 3-D digital imaging and modeling, p 185–192

Connelly R, Sabitov I, Walz A (1997) The Bellows Conjecture. Contrib. Algebra Geom. 38:1–10

Diebel JR, Thrun S, Brunig M (2006) A bayesian method for probable surface reconstruction and decimation. ACM Trans Graph V(N):1–20

Dixon AL (1908) The eliminant of three quantics in two independent variables. Proc London Math Soc 6:468–478

Faugere JC (1999) A new efficient algorithm for computing Groebner bases (F4). J Pure Appl Algebra (Elsevier Science) 139:61–88

Gentleman WM, Johnson SC (1976) A case study: determinants of matrices with polynomial entries. ACM Trans Math Softw (TOMS) TOMS Homepage Arch 2(3), Sept

Gray A, Abbena E, Salamon S (2006) Modern differential geometry of curves and surfaces with mathematica, chapman & hall/CRC. Taylor & Francis Group, USA

Haneberg WC (2004) Computational Geosciences with Mathematica. Springer, Berlin, Heidelberg

Hubert M, Rousseeuw PJ, Van den Branden K (2005) ROBPCA: a new approach to robust principal component analysis. Technometrics 47(1):64–79

Kalaba R, Tesfatsion L (1991) Solving nonlinear equations by adaptive homotopy continuation. Appl Math Comput 41:99–115

Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the 1995 IEEE international conference on neural networks (Conference proceedings), 1995, pp 1942–1948

Kenwright B (2012) Dual-quaternions, from classical mechanics to computer graphics and beyond, http://www.xbdev.net/misc_demos/demos/dual_quaternions_beyond/paper.pdf

Koenker R, Hallock KF (2001) Quantile regression. J Econ Perspect 15(4):143–156

Lakaemper R, Latecki LJ (2006) Extended EM for planar approximation of 3D data. In: IEEE International conference on robotics and automation (ICRa), Orlando, Florida, May 2006.

Lewis RH, Bridgett S (2003) Conic tangency equations arising from apollonius problems in biochemistry. Math Comput Simul 61(2):101–114

Lewis RH Computer algebra system Fermat. http://home.bway.net/lewis/

Lichtblau D (2008) Approximate Gröbner bases and overdetermined algebraic systems, ACA 2008, Session: symbolic and numeric computation. Linz, Austria

Nievergelt Y (2000) A tutorial history of least square with applications to astronomy and geodesy. J Comp Appl Math 121:37–72

Paláncz B, Awange JL, Zaletnyik P, Lewis RH (2009) Linear homotopy solution of nonlinear systems of equations in geodesy. J Geodesy 84(1):79–95

Poppinga J, Vaskevicius N, Birk A, Pathak K (2006) Fast plane detection and polygonalization in noisy 3D range images. In: international conference on intelligent robots and systems (IROS), IEEE Press, Nice, France, 2008

Proskova J (2011) Application of dual quaternions algorithm for geodetic datum transformation. J Appl Math IV(2):225–235

Rabbani T (2006) Automatic reconstruction of industrial installations using point clouds and images publications on Geodesy 62, NCG Netherlands Commissie voor Geodesie Netherlands Geodetic CommissionDelft

Russeeuw PJ, Van Driessen K (1999) A fast algorithm for the minimum covariance determinant estimator. Technometrics 41(3):212–223

Shijian Z, Yunlan G, Xinwu Z, Tieding L (2008) Robust algorithm for fitting sphere to 3D point clouds in terrestrial laser scanning. In: The international archives of photogrammetry and spatial information science, vol XXXVII Part B5. Beijing. pp 519–522

Sturmfels B (2003) Solving systems of polynomial equations. In: CBMS regional conference series in mathematics 97, American Mathematical Society, Providence

Weingarten JW, Gruener G, Siegwart R (2004) Probabilistic plane fitting in 3D and an application to robotic mapping. IEEE Int Conf 1:927–932

Westfeld P, Mulsow C, Schulze M (2009) Photogrammetric calibration of range imaging sensors using intensity and range information simultaneously. In: Grün A, Kahmen H (eds) Optical 3-d measurement techniques IX. vol II. Research Group Engineering Geodesy, Vienna University of Technology, p 129