The Optimal Linear Least Squares Problem and Methods of Nonlinear Optimization with Constraints

Erik Jacobsen
740500

December 13, 1992

Contents

Chapter 1

Overview

This report investigates various ways of solving a minimization problem with constraints.

1.1 The Problem

As testproblem an optimal linear least squares problem is used. Let A and b be defined as

$$A = \begin{bmatrix} 22 & 10 & 2 & 3 & 7 \\ 14 & 7 & 10 & 0 & 8 \\ -1 & 13 & -1 & -11 & 3 \\ -3 & -2 & 13 & -2 & 4 \\ 9 & 8 & 1 & -2 & 4 \\ 9 & 1 & -7 & 5 & -1 \\ 2 & -6 & 6 & 5 & 1 \\ 4 & 5 & 0 & -2 & 2 \end{bmatrix} \text{ and } b = \begin{bmatrix} 646 \\ 454 \\ 94 \\ -38 \\ 302 \\ 202 \\ 18 \\ 146 \end{bmatrix}$$

The linear least squares problem consists of finding an x such that $||Ax - b||_2$ is minimal. The *optimal* linear least squares problem is to find an x among the solutions to the linear least squares problem, such that $||x||_2$ is minimal.

In chapter 2 the theory behind linear least squares problems is investigated, and from the results I mention:

- There is a unique solution to the optimal linear least squares problem.
- The optimal linear least squares problem can be transformed to a normal nonlinear optimization problem with constraints.
- The problem may also be formulated in terms of the Lagrange-function.

Standard methods for solving linear least squares problems are not investigated here.

Some results from linear algebra used in this report are found in appendix B.

1.2 An Analytic Solution

To be able to check the numerical solutions in this report, I have made an analytic solution in appendix A.

It turns out that the (unique) solution is

$$x = (22, 10, 2, 3, 7)$$

which also happens to be the first row in A. It seems likely that the test-problem has been constructed...

Also from appendix A: A has rank 3, and thus $A^\top A$ is singular.

1.3 The Penalty Method

A nonlinear optimization problem with constraints:

$$\text{Minimize } f(x), \text{ where } g(x) = 0$$

can be solved using the penalty function

$$f_\mu(x) = f(x) + \mu g(x)^\top g(x)$$

For increasing values of the penalty parameter μ, this optimization problem will have solutions closer and closer to the minimum of the original problem — provided f and g are reasonably wellbehaved. Unfortunately large values of μ make the problem illconditioned.

Chapter 3 describes this in further detail.

1.4 Direct Methods

A direct method is characterized by using only the values of f — or in our case f_μ — and not any derivatives.

I have tried two methods:

- The simplex method.
- Pattern search.

Both methods use heuristic arguments to find a minimum, and as it turns out, do not perform well in illconditioned problems. Used with the penalty method they get close to the true solution, and here we note that pattern search uses 4 times as many functionevaluations compared to the simplex method.

1.5 Newton-Raphson

The Newton-Raphson method finds zeros of a function, using the derivative. Used to find minima for a function f, it finds zeros of the gradient using the Hessian.

It turns out that the testproblem is well suited for the Newton-Raphson method. In general it is known that the method can fail to find minima, e.g. because a zero of the gradient gives a maximum or saddlepoint.

1.6 Hestenes-Powell

The Hestenes-Powell method exploits a different way of minimizing with constraints. In the penalty method the problem becomes illconditioned for large μ, but here we need only a μ "sufficiently large". Then the minimum we find is the true minimum of the original problem.

This is illustrated in examples 3.1 and 7.1.

For the testproblem this method works well — it would be interesting to try it on other and larger testproblems.

1.7 The Programs

All programs have been written in TurboPascal 5.0, but I have not included all the various versions, with changing parameters and testprints, into this report. Instead I have put the main procedures in appendix D.

In all the programs I have — arbitrarily — chosen the IEEE floatingpoint format *extended*, with the highest precision. The results in the following chapters show that it is not enough just to use a high precision.

The programs included are:

- The testproblem (section D.1, page 69), containing the functions defining f, ∇f and H_f. These are used in the simplex search, the pattern search and the Newton-Raphson.

- The simplex search (section D.2, page 73), containing the procedure SimplexSearch.

- The pattern search (section D.3, page 81), containing the procedure PatternSearch.

- Solve a system of linear equations with Gaussean elimination (section D.4, page 84), used in the Newton-Raphson method and in the Hestenes-Powell method.

- The Newton-Raphson method (section D.5, page 85), containing the procedure Newton.

- The Hestenes-Powell method (section D.6, page 87). This method defines new f, ∇f and H_f, and uses a Newton-Raphson method for intermediate minimization.

Chapter 2

The Optimal Linear Least Squares Problem

This chapter defines the problem and proves basic properties. A knowledge of linear algebra is required, e.g. as presented in [FB74] or [FB90]. Some results used in this report are presented in appendix B.

2.1 The Linear Least Squares Problem

Let $A \in \mathbf{R}^{m \times n}$ be a real $m \times n$ matrix, and let $b \in \mathbf{R}^m$ be a real vector:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{2.1}$$

A linear least squares solution $\hat{x} \in \mathbf{R}^n$ to the equation

$$Ax = b \tag{2.2}$$

is a solution to the problem

$$\min_{x \in \mathbf{R}^n} ||Ax - b||_2 \tag{2.3}$$

We call $r = b - Ax$ the residual vector, and can formulate the problem as minimizing the 2-norm of $r = (r_1, \ldots, r_n)$:

$$||r||_2 = \sqrt{r_1^2 + \cdots + r_n^2} \tag{2.4}$$

This equation indicates why we have a *least squares* problem.

Linear least squares problems are used for several purposes, e.g. in approximation of empirical data — discussions on this can be found in [Ste73, p. 218], [RR78, p. 249f] and [FB90, sect. 5.5].

2.1.1 Solutions

In the special case where $m = n$, A is a square matrix and we know that there is exactly one solution to (2.2) if and only if A is non-singular, and this solution is $\hat{x} = A^{-1}b$. This is then a solution to (2.3). The general case is a bit more complicated.

Exisistence of Solutions

Using our knowledge of linear algebra we can write a vectorspace as a direct sum of a subspace and its complement:

$$\mathbf{R}^m = \mathcal{R}(A) \oplus \mathcal{R}(A)^{\perp} \tag{2.5}$$

so that we can uniquely write $b = b_1 + b_2$, where $b_1 \in \mathcal{R}(A)$ and $b_2 \in \mathcal{R}(A)^{\perp}$.

Since $\mathcal{R}(A)$ is a subspace we know that $Ax - b_1 \in \mathcal{R}(A)$ for all $x \in \mathbf{R}^n$, and of course $Ax - b_1 \perp \pm b_2$. The Pythagorean theorem gives:

$$||Ax - b||_2^2 = ||Ax - (b_1 + b_2)||_2^2 = ||Ax - b_1 + (-b_2)||_2^2 = ||Ax - b_1||_2^2 + ||b_2||_2^2 \tag{2.6}$$

The minimum of $||Ax - b||_2$ is thus achieved for the x that minimizes $||Ax - b_1||_2$. But since $b_1 \in \mathcal{R}(A)$ we will always be able to find an $\hat{x}$ such that $A\hat{x} = b_1$, giving $||A\hat{x} - b_1||_2 = 0$, which is a minimum.

The conclusion is that we can always find a solution to a linear least squares problem.

Uniqueness of Solutions

Let $\hat{x}$ be a solution to (2.3), i.e. $A\hat{x} = b_1$, where b_1 is defined as above.

Let $\hat{y}$ be another solution to (2.3), and let $x' = \hat{y} - \hat{x}$, i.e. $\hat{y} = \hat{x} + x'$. We get:

$$\begin{aligned} & A\hat{y} = b_1 \\ \Updownarrow & \\ & A(\hat{x} + x') = b_1 \\ \Updownarrow & \\ & A\hat{x} + Ax' = b_1 \\ \Updownarrow & \\ & Ax' = 0 \\ \Updownarrow & \\ & x' \in \mathcal{N}(A) \end{aligned} \tag{2.7}$$

so that all solutions to (2.3) are of the form $\hat{x} + \mathcal{N}(A)$.

So there is a unique solution if and only if $\text{null}(A) = 0$.

From linear algebra we have the lemma that $\text{null}(A) = 0$ if and only if A has linearly independent columns. A corollary is that if $A \in \mathbf{R}^{m \times n}$ and $n > m$, then $\text{null}(A) \neq 0$, in which case there is no unique solution.

2.1.2 Normal Equations

The linear least squares problem can also be formulated in terms of the so-called normal equations:

$$\{x \mid ||Ax - b||_2 = \min_{x' \in \mathbf{R}^n} ||Ax' - b||_2\} = \{x \mid A^\top Ax = A^\top b\} \tag{2.8}$$

To prove this we observe that if x is a solution to (2.3) we have that $Ax = b_1 \Leftrightarrow Ax - b = -b_2$. Since $-b_2 \in \mathcal{R}(A)^\perp = \mathcal{N}(A^\top)$ we have

$$\begin{aligned} & A^\top(-b_2) = 0 \\ \Updownarrow & \\ & A^\top(Ax - b) = 0 \\ \Updownarrow & \\ & A^\top Ax = A^\top b \end{aligned} \tag{2.9}$$

We can solve the normal equations (using e.g. Gaussian elimination), but if there is no unique solution we have to choose among an infinite number of solutions.

2.2 The Optimal Linear Least Squares Problem

In the preceeding section we saw that the solutions to the linear least squares problem (2.3) are of the form $\hat{x} + \mathcal{N}(A)$.

We define the optimal[1] linear least squares solution to be the $x \in \hat{x} + \mathcal{N}(A)$ with the smallest l_2-norm, i.e. a solution to the problem:

$$\min_{x \in \mathbf{R}^n} x^\top x, \text{ where } ||Ax - b||_2 \text{ is minimal} \tag{2.10}$$

From the discussion in section 2.1.2 we see that this is equivalent to the problem:

$$\min_{x \in \mathbf{R}^n} x^\top x, \text{ where } A^\top Ax = A^\top b \tag{2.11}$$

2.2.1 Solutions

When there is a unique solution to (2.3) there is of course a unique solution to (2.10) (and thus to (2.11)). But we must consider the general case.

[1] Sometimes called *general*.

Existence and Uniqueness of Solutions

The set of solutions to (2.3) form an affine subspace $U = \hat{x} + \mathcal{N}(A)$, where $\hat{x}$ is a particular solution and $\mathcal{N}(A)$ is the null space for A.

To minimize $x^\top x = ||x||_2^2$ in U is thus to find a vector or vectors with the smallest 2-norm. Since $||v||_2 = ||0 - v||_2$ this is the same as finding a vector with minimum distance from the zero-vector, 0. The orthogonal projection of 0 onto U is the vector with minimum distance ([FB90, p. 276]).

It is a slight complication that U is an affine subspace, but we can transform it into a linear subspace, do the projection and transform it back. Let $U' = \{x - \hat{x} | x \in U\} = \mathcal{N}(A)$. Then

$$x = P_{\mathcal{N}(A)}(-\hat{x}) + \hat{x} = (I - P_{\mathcal{N}(A)})\hat{x} \in \mathcal{N}(A)^\top = \mathcal{R}(A^\top)$$

So a solution certainly exists, and since orthogonal projections are unique, the solution is also unique.

However these arguments do not give any direct way to find the solution to an optimal linear least squares problem, but theorem (B.7) states that

$$\hat{x} = A^+ b$$

where A^+ is the pseudo-inverse of A (see section B.3).

2.3 The Lagrange-function

Now consider the Lagrange-function

$$L(x, \lambda) = x^\top x + \lambda^\top (A^\top A x - A^\top b) \tag{2.12}$$

The gradient of $L(x, \lambda)$ is

$$\nabla L(x, \lambda) = \begin{bmatrix} \frac{\partial L}{\partial x}(x, \lambda) \\ \frac{\partial L}{\partial \lambda}(x, \lambda) \end{bmatrix} = \begin{bmatrix} 2x + A^\top A \lambda \\ A^\top A x - A^\top b \end{bmatrix}$$

When $\nabla L(x, \lambda) = 0$ we get these equations

$$\begin{aligned} x &= -\tfrac{1}{2} A^\top A\lambda \\ A^\top Ax &= A^\top b \end{aligned}$$

If $\hat{x}$ and $\hat{\lambda}$ exist such that $\nabla L(\hat{x}, \hat{\lambda}) = 0$ we see that $\hat{x}$ satisfies the normal equations, and that $\hat{x} \in \mathcal{R}(A^\top A) = \mathcal{R}(A^\top)$, which is exactly the necessary conditions for $\hat{x}$ to be the solution to (2.11).

2.3.1 Lagrange's Theorem

A wellknown theorem is useful in this context (e.g. see [WCT72, p. 595]):

Theorem 2.1 [Lagrange] *Let the function $g\colon \mathbf{R}^n \to \mathbf{R}^m$, $m < n$, be continuously differentiable. Suppose the equations*

$$\begin{aligned} g_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ g_m(x_1, \ldots, x_n) &= 0 \end{aligned}$$

implicitly define a surface S in $\mathbf{R}^n$, and that at a point x_0 of S the matrix $\nabla g(x_0)$ has some m columns linearly independent. If x_0 is an extreme point of a differentiable function $f : \mathbf{R}^n \to \mathbf{R}$, when restricted to S, then x_0 is a critical point of the function

$$f(x) + \lambda_1 g_1(x) + \cdots + \lambda_m g_m(x)$$

for some constants $\lambda_1, \ldots, \lambda_m$. □

At a first glance this theorem does not apply in the case of an optimal linear least squares problem, since there are n constraint functions in the equation $A^\top Ax - A^\top b = 0$.

However this is only the case if $\operatorname{rank}(A) = n$, in which case $\operatorname{null}(A) = 0$ and there is a unique solution even without the constraints. When $\operatorname{rank}(A) = m < n$ we can reduce $A^\top Ax - A^\top b = 0$ to m equations, and use theorem 2.1.

2.4 Methods for Linear Least Squares Problems

This report does not cover the usual methods for solving linear least squares problems. The reader is referred to other literature, e.g. [GvL83, ch. 6] and [RR78, ch. 6].

Chapter 3

Solving with Penaltyfunction

The problem as formulated in (2.11) is

$$\min_{x \in \mathbf{R}^n} x^\top x, \text{ where } A^\top Ax - A^\top b = 0 \tag{3.1}$$

This is now a problem of the form

$$\min_x f(x), \text{ where } h(x) = 0 \tag{3.2}$$

so that we can use the techniques discussed in [Ram76, chapter 15]. The problem of minimizing a function with nonlinear equality constraints is reformulated to a normal minimization problem

$$f_\mu(x) = f(x) + \mu h(x)^\top h(x) \tag{3.3}$$

where $h(x)^\top h(x)$ is the penalty function. In theory the solution of $f_\mu(x)$ converges to the correct solution for (3.2) as μ goes to infinity[1].

3.1 The Penalty Method

In [Fle81, p. 122] an iteration method is suggested, which I will call "The Penalty Method":

[1] [Ram76] uses $1/r$ where r goes to 0.

1. Choose an initial $\mu^{(0)}$, and make a sequence $\{\mu^{(k)}\} \to \infty$, e.g. powers of 10:
$$\{\mu^{(0)}, \ldots, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, \ldots\}$$

2. For each $\mu^{(k)}$ we find a local minimum for $f_{\mu^{(k)}}(x)$, say $x^{(k)}$. All iterative minimization processes require an initial startpoint, so when minimizing $f_{\mu^{(k)}}(x)$ we use $x^{(k-1)}$ as startpoint. For the minimization of $f_{\mu^{(0)}}$ we choose an arbitrary $x^{(-1)}$, e.g. 0.

3. We stop the process when we think we are sufficiently close to the minimum we want.

It should be noted that none of the $x^{(k)}$ will be a minimum point for the constrained problem, but we can only hope that $x^{(k)} \to \hat{x}$ for $k \to \infty$.

The penalty method does not specify which minimization method to be used in step 2, and part of the purpose of this report is to try some of the standard methods.

There are several ways to decide when to terminate the process in step 3:

- When $||h(x^{(k)})||$ is sufficiently small ([Fle81, p. 122]). In general this only means that the constraints are satisfied, and not that that we have a solution to (3.2). However for the penalty method, starting with a sufficiently small value of μ, it may work.

  For smaller values of μ, the minimization is mainly done with respect to $f(x)$, but with increasing values of μ the constraints become more important, and thus $h(x^{(k)})$ will be smaller and smaller.

  So assuming that the $x^{(k)}$ do converge, this criterion seems to be useful, but used on its own we have no indication of the quality of the solution, and it might happen that we have $h(x^{(k)}) \approx 0$ far from the solution to (3.2).

- [Ram76, p. 216] suggest that we stop the process early on and extrapolate to the true solution.

I haven't tried these two methods. Instead I will consider some that are based on successive values of $f_\mu(x^{(k)})$. Let $f_k = f_\mu(x^{(k)})$:

- (Absolute error) $|f_{k+1} - f_k| < \epsilon$. An absolute criterion is dangerous if we don't know the magnitude of f.
- (Relative error) $|f_{k+1} - f_k|/|f_{k+1}| < \epsilon$. This is the most general criterion.
- (Increasing differences) $|f_{k+1} - f_k| > |f_k - f_{k-1}|$. I.e. when the convergence starts to jump away.

As stated before f_μ becomes more illconditioned as μ gets larger. This means that the sequence $x^{(k)}$ may behave in a way such that it is difficult to decide when we are close to the wanted solution.

3.1.1 Advantages and Disadvantages

The advantage of this method is that we can use normal, unconstrained minimization techniques.

But, as noted, a disadvantage is that the problem becomes more illconditioned as μ gets larger.

Another disadvantage is that $h(x)$ may be undefined for certain x, e.g. containing squareroots or logarithms which are undefined for negative values. To avoid this we could construct a new function $\bar{h}(x)$ identical to $h(x)$, except that it returns large penaltyvalues for illegal x-values, but it might be difficult to make $\bar{h}$ differentiable.

Anyway this is not a problem for the testproblem in this report.

3.1.2 Feasible Points

A feasible point is one for which the constraints are satisfied. Some minimization methods for constrained problems may require a that the starting point is feasible (e.g. [Sub89]). If one is not immediate from the equations, it can be found by minimizing $g(x) = h(x)^\top h(x)$ (see [BDS69, p. 44]).

3.2 An Example

In [Hes75, p. 255] the following example is given:

Example 3.1 We want to minimize $f(x) = x_1^2 - x_2^2 - 4x_2$ subject to the constraint $h(x) = x_2 = 0$. We define

$$\begin{aligned} f_\mu(x) &= f(x) + \mu h(x)^2 \\ &= x_1^2 - x_2^2 - 4x_2 + \mu x_2^2 \\ &= x_1^2 - 4x_2 + (\mu - 1)x_2^2 \end{aligned}$$

which, for $\mu > 1$, has as minimum $x_\mu = (0, \frac{2}{\mu - 1})$, so that $x_\mu \to (0, 0)$ for $\mu \to \infty$. And indeed $(0, 0)$ is the minimum we want. □

3.3 Gradient and Hessian

We will need the gradient and Hessian of f_μ. So we calculate

$$\begin{aligned} f_\mu(x) &= x^\top x + \mu(A^\top Ax - A^\top b)^\top (A^\top Ax - A^\top b) \\ &= x^\top x + \mu(x^\top A^\top AA^\top Ax - b^\top AA^\top Ax - x^\top A^\top AA^\top b + b^\top AA^\top b) \\ &= x^\top x + \mu x^\top A^\top AA^\top Ax - 2\mu b^\top AA^\top Ax + \mu b^\top AA^\top b \\ &= x^\top (I + \mu A^\top AA^\top A)x + (-2\mu b^\top AA^\top A)x + (\mu b^\top AA^\top b) \end{aligned} \tag{3.4}$$

This is a quadratic function of the form

$$f_\mu(x) = x^\top Q x + v^\top x + s$$

where Q is a $n \times n$ matrix, v is an n-dimensional vector, and s is a scalar. In this case we get gradient and Hessian as

$$\begin{aligned} \nabla f_\mu(x) &= 2Qx + v \\ H_{f_\mu}(x) &= 2Q \end{aligned}$$

For the given f this gives

$$\begin{aligned} \nabla f_\mu(x) &= 2(I + \mu A^\top A A^\top A)x + (-2\mu b^\top A A^\top A)^\top \\ H_{f_\mu}(x) &= 2(I + \mu A^\top A A^\top A) \end{aligned}$$

3.3.1 The Program

The program in section D.1 contains the 3 functions to calculate f_μ, the gradient and the Hessian.

In the procedure `TestInit` $A^\top A$ and $A^\top b$ are calculated, and so are these useful quantities:

$$\begin{aligned} s' &= (A^\top b)^\top A^\top b \\ v' &= -2(A^\top b)^\top A^\top A \\ Q' &= A^\top A A^\top A \end{aligned}$$

Now

$$\begin{aligned} f_\mu(x) &= x^\top x + \mu(x^\top Q' x + {v'}^\top x + s') \\ \nabla f_\mu(x) &= 2(x + \mu Q' x) + \mu {v'}^\top \\ H_{f_\mu}(x) &= 2(I + \mu Q') \end{aligned}$$

We note that the Hessian is independent of x. This could be used in the implementations of the minimization processes, but the programs presented in appendix D are general, and recalculate the Hessian each time it is used.

Chapter 4

The Simplex Method

A simple and generally useful method is the simplex method first described in [NM65]. This is a generalization of the method in [SHH62], and has been the subject of numerous articles.

The simplex method is a direct method, since it only uses the values of the function to minimize and not any derivatives. As with all direct methods the strategy is based on heuristics, and — as we shall see — will not always work.

4.1 Description of the Method

The figures accompanying the description will be in two dimensions, but the description will be general for dimension n. Let $f(x) : \mathbf{R}^n \rightarrow \mathbf{R}$ be the function to minimize.

In [Kui62] an *n-simplex* is defined as the point set in $\mathbf{R}^n$:

$$S = \{x_1 + \sum_{i=2}^{n+1} k_i x_i \mid k_i > 0, \sum_{i=2}^{n+1} k_i < 1, x_i \in \mathbf{R}^n\}$$

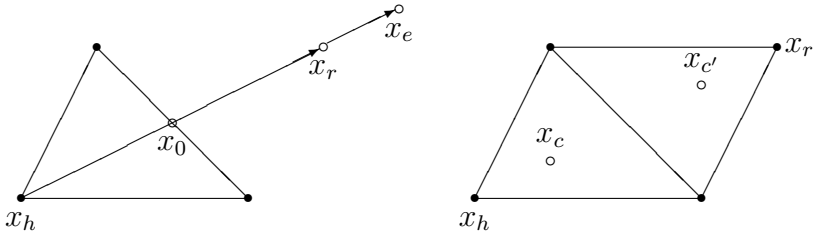

Figure 4.1: Reflection, expansion and contraction

We will think of the simplex[1] as the $n+1$ vertices or points $x_1, \ldots, x_{n+1}$. In $\mathbf{R}^2$ it is a triangle, in $\mathbf{R}^3$ a tetrahedron. The function values in the points will be called $f_1, \ldots, f_{n+1}$, where $f_i = f(x_i)$.

3 of these points are special:

x_l	is the point with the lowest function value,
x_h	is the point with the highest function value and
$x_{h'}$	is the point with the second highest function value.

and we also define $f_l = f(x_l)$, $f_h = f(x_h)$ and $f_{h'} = f(x_{h'})$.

After choosing an initial simplex, the principle is to gradually move the simplex towards a local minimum.

In figure 4.1 the simplex is a triangle. It seems reasonable to look for the minimum in a direction away from x_h. Calculating the "average" (or *centroid*) of the remaining points:

$$x_0 = \frac{1}{n} \sum_{i=1, i \neq h}^{n+1} x_i \tag{4.1}$$

we can calculate the *reflection point* as

$$x_r = (1+\alpha)x_0 - \alpha x_h \text{ and } f_r = f(x_r)$$

Here $\alpha > 0$ is a parameter controlling how far the reflection goes.

[1] In plural: *simplices*

If $f_r < f_l$ we can assume that we are going in the right direction, i.e. "downhill". Perhaps going a little further in that direction will bring us further down, so we calculate an *expansion point* as

$$x_e = \gamma x_r + (1-\gamma)x_0 \text{ and } f_e = f(x_e)$$

The parameter $\gamma > 1$ controls how far we go.

If $f_e < f_r$ we now have a new minimum of all the points considered, and set $x_h = x_e$. Otherwise the minimum is in x_r and we set $x_h = x_r$. In either case we restart the process with a new reflection from a new x_h. Reflection and expansion are illustrated in the left side of figure 4.1.

Otherwise if $f_r < f_{h'}$ we simply set $x_h = x_r$. This should give a sligthly better simplex.

When $f_r \geq f_{h'}$ there are two possibilities: $f_r \geq f_h$ or $f_r < f_h$. In the latter case setting $x_h = x_r$ should make a slightly better simplex, but since we haven't found a significantly smaller function value around either simplex, we will reduce it by making a *contraction point* inside the simplex

$$x_c = \beta x_h + (1-\beta)x_0 \text{ and } f_c = f(x_c)$$

Here $0 < \beta < 1$ is the parameter controlling how far we go. In the right side of figure 4.1 x_c and $x_{c'}$ illustrate that the contraction point is chosen inside the "best" simplex.

When the function is convex x_c will be a new minimum, and "around" a minimum point most functions are convex. So when $f_c < f_h$ we set $x_h = x_c$, and start again.

Otherwise we have found no new minimum point, and there might be a maximum point inside the simplex. In this case we *shrink* the simplex towards x_l, as in figure 4.2, hoping to get around the maximum point.

$$x_i = \delta x_i + (1-\delta)x_l, \text{ for } 1 \leq i \leq n+1$$

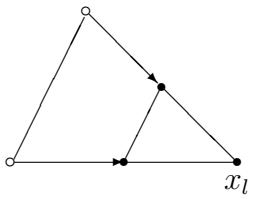

Figure 4.2: Shrinkage

4.1.1 When to Stop

This infinite process must stop eventually. We can look at the points in the simplex, or at the function values (or a combination), and we can choose an absolute or relative criterion. Different criteria have been suggested in the literature. In [Sub89] the process stops when

$$\sum_{i=1}^{n+1} ||x_i - x_l||_1 < \epsilon \tag{4.2}$$

Both [NM65], [Ram76] and [PH72] want to stop when all the function values f_i are close:

$$\sqrt{\sum_{i=1}^{n+1}(f_i - \bar{f})^2} < \epsilon, \text{ where } \bar{f} = \frac{1}{n+1}\sum_{i=1}^{n+1} f_i \tag{4.3}$$

A similar criterion is used in [NRP]:

$$\frac{2|f_h - f_l|}{|f_h| + |f_l|} < \epsilon \tag{4.4}$$

and since we already have f_h and f_l this is easier to compute.

In the following I have used the criterion from [NM65], i.e. (4.3).

4.1.2 Where to Start

It is obvious that if the simplex is close to the minimum point from the start, the process is likely to converge faster. In the general case we don't have any

information about the minimum point.

Since all new points (x_r, x_e, x_c and the x_i from the shrinking step) are linear combinations of old points we must have that the initial simplex span $\mathbf{R}^n$ — otherwise we would search for a minimum in a subspace. In theory every new simplex will also span $\mathbf{R}^n$.

A simple way to choose the starting simplex is to use $(0, 0, \ldots, 0)$ as initial vector, and use the n unitvectors, e_i, to produce the remaining n points:

$$x_1 = 0, \quad x_i = x_1 + \sigma e_{i-1} \text{ for } 2 \leq i \leq n+1 \tag{4.5}$$

Here σ controls the initial size of the simplex.

The choice of parameters $(\alpha, \beta, \gamma, \delta)$ in the literature is suggested as either

$$\begin{array}{ll} (1 \quad , \frac{1}{2}, 2 \quad , \frac{1}{2}) & \text{[PH72] and [NM65]} \\ (2 \quad , \frac{1}{4}, 2\frac{1}{2}, \frac{1}{2}) & \text{[PH72]} \\ (0.95, \frac{1}{2}, 2 \quad , \frac{1}{2}) & \text{[Sub89]} \end{array} \tag{4.6}$$

These parameters control the speed of convergence, and also if the process converges to a correct minimum. It is possible to choose parameters and testfunctions in many ways, and use lots of time and energy experimenting with them.

4.1.3 Variants

Both [Ram76] and [PH72] mention that the expansion process can be repeated as long as smaller function values are found in that direction. The simplex should then be moved to this new minimum, instead of just replacing one point. This avoids getting a simplex with greatly varying sidelengths.

Instead of finding the centroid as in (4.1) we can calculate a *weighted centroid*, which will be closer to the points in the simplex with the smaller function values.

$$x_0 = \frac{1}{c} \sum_{i=1}^{n+1} (f_h - f_i) x_i$$

where

$$c = (n+1)f_h - \sum_{i=1}^{n+1} f_i$$

Neither of these variants have been tried here.

4.1.4 The Original Method

The original method in [SHH62] — as described in [BDS69] — uses only reflection and shrinkage. It starts with a regular simplex, and uses reflections with $\alpha = 1$, thus keeping the simplex regular.

Normally the point with the highest function value is reflected, but if the new reflection point also has the highest function value, the point with the second highest function value is reflected (otherwise the process would oscillate).

When approaching a minimum it is claimed that the simplex will rotate round one of its points, so when one of the points has been unchanged for M iterations, where $M \approx 1.65n + 0.05n^2$, the simplex is shrunk towards that point. The simplex stays regular.

There is no way no increase the size of the simplex, and as we will see in chapter 5, this can give a slow convergence. Of course someone has suggested a way to do this — according to [BDS69].

4.2 The Program

The main procedures implementing the simplex algorithm can be found in section D.2.

It is a good idea to keep the points in the simplex sorted according to the function values. This makes it easy to find the smallest, the largest and the second largest function value.

Updating the sorted points is easy when we find a new minimum (no comparisons), a little harder when we find a point between the highest and lowest value ($O(\log n)$ comparisons in a binary search), and hard when a shrinkage is performed ($O(n \log n)$ comparisons sorting the values).

But I must admit that I have used simpler algorithms, with other — and worse — timecomplexities. Also all n floatingpoint values in a point are moved when sorting, instead of moving pointers. However it makes no difference for this report, since no execution times are measured.

4.3 Experimental Results

To get an idea of the performance I have made a run of the testproblem with these parameters:

$$(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.5, 1.5, 0.5, 10^{-4}, 1)$$

and with varying μ. The parameters are chosen sligthly different from the recommendations in (4.6), since the usefulness of the method should be largely independent of this choice. As will be shown later on, this is not always the case.

In each case the simplex starts in 0 as described in (4.5) — the penalty method will be used later.

The results are found in table 4.1.

As we expect the results show that x does converge towards the true solution (which we know is $x = (22, 10, 2, 3, 7)$, where $f(x) = 646$), as μ increases, but also that at some point it converges to a false minimum.

The False Minimum

From table 4.2 we see that the residue gets smaller as μ gets larger. This is quite reasonable since the penaltyfactor, $\mu||A^\top Ax - A^\top b||_2^2$, becomes more

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	106
10^{-6}	12.4474	7.0336	2.1872	0.4026	4.7468	382.9335	216
10^{-5}	20.0006	10.0434	2.5876	1.8263	6.9253	599.6577	276
10^{-4}	21.7530	10.0408	2.1060	2.8155	7.0070	640.6690	309
10^{-3}	21.9766	10.0035	2.0125	2.9781	6.9968	645.4568	480
10^{-2}	21.9968	9.9999	2.0001	2.9982	7.0030	645.9458	544
10^{-1}	21.9967	10.0035	1.9999	3.0049	7.0024	645.9947	698
10^{0}	21.5791	8.2333	0.1069	2.3903	11.6489	674.8661	501
10^{1}	21.7380	7.4891	-0.3544	1.7973	12.5985	690.7057	556
10^{2}	21.6580	7.7992	-0.1761	2.0582	12.2444	684.0861	660
10^{3}	27.2824	1.1417	-0.0995	-7.4497	8.1312	867.2566	367
10^{4}	27.2824	1.1417	-0.0995	-7.4497	8.1312	867.2580	386
10^{5}	27.2824	1.1417	-0.0995	-7.4497	8.1312	867.2580	403
10^{6}	27.2824	1.1417	-0.0995	-7.4497	8.1312	867.2579	428
10^{7}	*Not converging?*						

Table 4.1: Simplex with $(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.5, 1.5, 0.5, 10^{-4}, 1)$

important. But why does the process converge to the false minimum $\bar{x} = (27.2824, 1.1417, -0.0995, -7.4497, 8.1312)$ for the larger μ?

More investigations — omitted here — of the processes from table 4.1 show that for values of $\mu \geq 10^{-4}$ the point x_l will get close to $\bar{x}$ after around 100–140 iterations, and closer for higher values of μ. The process will stay around this point for a while, and then either converge in the direction of the true minimum, or stop. For $\mu = 10^7$ the point x_l is also very close to $\bar{x}$ when the process is in the infinite loop.

A possible explanation is that from the initial simplex it is "downhill" towards $\bar{x}$, for which the residue is small. $\bar{x}$ is in a "valley"[2] with another orientation, and the process must spend some time around $\bar{x}$ in order to change the orientation of the simplex. For smaller values of μ this succeeds, but for larger values of μ the simplex gets so small that we think it is a minimum. Again intuitively it could be that larger μ give steeper "sides" in the "valley".

[2]Whatever "downhill" and "valley" means in higher dimensions.

μ	$\|\|x\|\|_2$	$\|\|A^\top Ax - A^\top b\|\|_2$
0	0.0000	31229.0244
10^{-6}	15.2278	12290.1110
10^{-5}	23.6407	2019.2701
10^{-4}	25.2085	228.0328
10^{-3}	25.3952	23.2098
10^{-2}	25.4143	2.4224
10^{-1}	25.4163	0.2289
10^{0}	25.9782	0.0359
10^{1}	26.2813	0.0065
10^{2}	26.1550	0.0008
10^{3}	29.4492	0.0003
10^{4}	29.4492	0.0001
10^{5}	29.4492	0.0000
10^{6}	29.4492	0.0000

Table 4.2: Norms of x and the residue from table 4.1

To check this theory I have another testrun, similar to table 4.1, but with a smaller ϵ. See table 4.3

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^1	21.7380	7.4892	-0.3544	1.7973	12.5985	690.7057	580
10^2	21.6608	7.7958	-0.1760	2.0534	12.2423	684.0860	686
10^3	21.5993	8.0913	0.0087	2.2873	11.8605	677.9015	793
10^4	27.2824	1.1417	-0.0995	-7.4497	8.1312	867.2580	401

Table 4.3: Simplex with $(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.5, 1.5, 0.5, 10^{-5}, 1)$

Here for $\mu = 10^3$ we get away from $\bar{x}$, to stop somewhat closer to the true minimum, which supports my theory. It seems that we can't choose ϵ too small, since this may introduce infinite loops — this happens e.g. for $\mu = 10^3$ and $\epsilon = 10^{-8}$ with the rest of the parameters unchanged.

Starting with a Larger Simplex

[KO68, p. 119] state that it is easier to contract than to expand the simplex, so it is important that the initial simplex be large enough. So let's choose a larger starting simplex, with $\sigma = 100$ which in our case "contains" the minimum. This could avoid getting into steep valleys.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^{-1}	22.0017	9.9993	2.0015	2.9974	6.9955	645.9946	589
10^{0}	21.9980	10.0011	1.9990	3.0026	7.0037	645.9996	585
10^{1}	22.0009	9.9995	2.0004	2.9988	6.9984	646.0000	659
10^{2}	22.0006	10.0001	2.0007	2.9994	6.9979	646.0000	905
10^{3}	11.5630	21.9524	1.5233	20.4092	15.4024	1271.7001	575

Table 4.4: Simplex with $(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.5, 1.5, 0.5, 10^{-4}, 100)$

The results in table 4.4 show that this is a good idea. The illconditioned problems for $\mu = 1, 10, 100$, that couldn't be solved with the smaller simplex, now converge. And for $\mu = 0.1$ it even converges faster.

This could be because a small simplex must start with a number of reflections and expansions, which are likely to head off in a direction away from the minimum.

4.3.1 Collapsing into a Subspace

In [Sub89] it is stated that the simplex method is useful in the case of unconstrained optimization problems, but that using penalty functions could lead the simplex to flatten against itself and collapsing into a subspace. The same is indicated in [Box66].

In two dimensions this would happen if the 3 points end up in a line, in which case the optimization would give a false minimum on this line.

Does this happen in this testproblem? To check this we need a few concepts from linear algebra (and numerical analysis).

The simplex with $n+1$ points, when properly made, spans an affine subspace of dimension n in $\mathbf{R}^{n+1}$. If this affine subspace is made into a linear subspace, by appointing one point to be the origo — by subtracting this point from the rest, and then throw it away — the remaining n points will then span $\mathbf{R}^n$, or in other words: be linearly independent.

If the simplex flattens against itself and collapses into a subspace (as suggested in [Sub89]), we would have that the n points are linearly dependent, or almost linearly dependent.

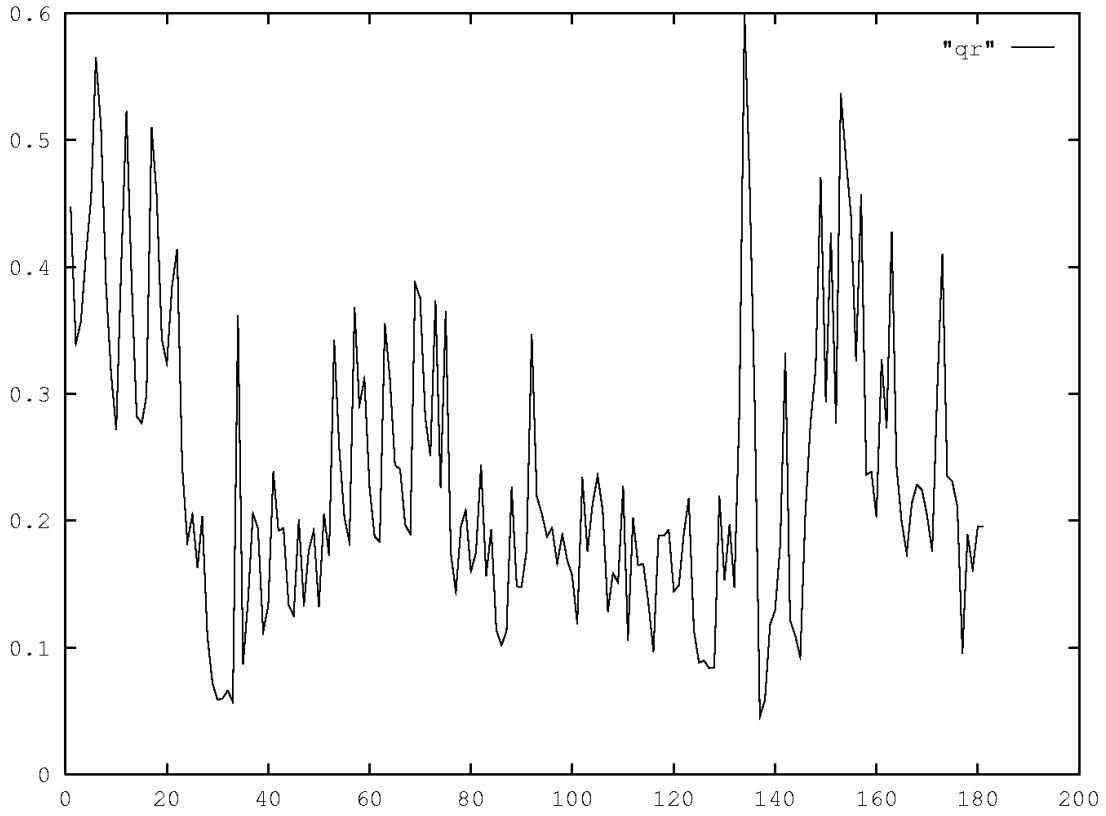

Figure 4.3: Ratios of distances in the simplices.

Distances in the Simplices

A way to check for this is to calculate the distances in the simplex, i.e. the distance from one point to the subspace spanned by the rest of the points. If one of these distances gets too small, compared with the other distances, we have a degenerate simplex.

A way to compute these distances is this: For each point s in the simplex, appoint one of the remaining n points as origo (by subtracting it from all the points). Find an orthonormal basis for the span of the remaining n points. This can be done by putting the n points into a matrix, do a QR-decomposition using Householder transformations, after which the columns in Q will be an orthonormal basis — see e.g. [RR78, p. 522 and 454]. The projection of s onto the subspace spanned by the basis is now $QQ^\top s$ — see e.g. [GvL83, p. 21]. Then calculate the distance between s and it's projection.

When we have distances for all $n+1$ points, the ratio of the smallest to the largest can be calculated, Figure 4.3 shows this plot for the entry from table 4.1 with $\mu = 10^3$ (the "qr" line).

Again we see that the simplex is reasonably regular, since the smallest distance encountered is only about 20 times smaller than the largest.

Conclusion

These experiments have not ruled out the possibility of a simplex collapsing into a subspace, but it doesn't seem to happen for this problem.

4.3.2 Sensitivity to Parametervalues

It turns out that for this problem the simplex method is sensitive to small changes in the chosen parameters. If we e.g. choose to use a larger contraction parameter, say $\beta = 0.6$, we can get a quite different result, as shown in table 4.5.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^1	22.0027	9.9976	2.0007	2.9959	6.9965	646.0002	1061

Table 4.5: Simplex with $(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.6, 1.5, 0.5, 10^{-4}, 1)$

Not only do we get close to the true minimum, but we also use about twice as many function evaluations.

Changing the expansion parameter, say $\gamma = 1.4$, also gives a surprising result, as seen in table 4.6.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^3	34.2444	-1.7642	4.4410	-16.1068	-7.1853	1506.5731	338

Table 4.6: Simplex with $(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.5, 1.4, 0.5, 10^{-4}, 1)$

This is neither the right minimum, nor close to the point from table 4.1.

These two results probably have something to do with the things [PH72] discuss, e.g. that the orientation of the initial simplex is important. Changing these parameters do not change the orientation of the initial simplex, but of the first few simplices. Thus we will head in a different direction, and perhaps, perhaps not, find the true minimum.

4.4 Using the Penalty Method

The previous experiments in this paragraph have been used to investigate the simplex method, but have not used the penalty method described in chapter 3, i.e. to use the previous $\hat{x}$ as starting point for the next μ-value. This is done in table 4.7, and all parameters are the same as in table 4.1.

A new simplex is made from the previous $\hat{x}$ as described in (4.5), where $x_1 = \hat{x}$.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^{-6}	12.4474	7.0336	2.1872	0.4026	4.7468	382.9335	216
10^{-5}	20.0037	10.0420	2.5921	1.8253	6.9165	599.6578	244
10^{-4}	21.7517	10.0387	2.1076	2.8163	7.0118	640.6689	184
10^{-3}	21.9753	10.0037	2.0115	2.9796	7.0009	645.4567	247
10^{-2}	21.9937	9.9729	1.9745	2.9858	7.0643	645.9513	201
10^{-1}	21.9977	9.9722	1.9748	2.9856	7.0595	645.9998	190
10^{0}	21.9978	9.9725	1.9749	2.9860	7.0592	646.0046	179
10^{1}	21.9900	9.9818	1.9748	2.9994	7.0649	646.0053	168
10^{2}	21.9936	9.9774	1.9748	2.9933	7.0625	646.0053	196
10^{3}	21.9937	9.9774	1.9748	2.9932	7.0624	646.0052	200
10^{4}	21.9929	9.9784	1.9748	2.9945	7.0629	646.0052	212
10^{5}	21.9917	9.9798	1.9749	2.9965	7.0637	646.0052	227
10^{6}	21.9907	9.9811	1.9749	2.9983	7.0643	646.0052	242
10^{7}	*Not converging?*						

Table 4.7: Penalty method: Simplex with $(\alpha, \beta, \gamma, \delta, \epsilon, \sigma) = (1, 0.5, 1.5, 0.5, 10^{-4}, 1)$

Comparing table 4.7 and table 4.1, we see that for larger μ we are now close to the true minimum, and that for each μ fewer function evaluations are required.

Stopcriteria for the Penalty Method

We still need to discuss a criterion for stopping the process in the penalty method.

Investigating the criteria from page 15 we get:

- (Absolute error) For $\epsilon = 10^{-4}$ we would stop in table 4.7 with $\mu = 10^2$, which looks like a good place.
- (Relative error) For $\epsilon = 10^{-4}$ this would make us stop with $\mu = 10^{-1}$. Here a total of 1281 functionevaluations are used.

- (Increasing differences) This would again (as far as can be read from the table) stop at $\mu = 10^2$.

From the few experiments made here it is hard to say which method would be better, and it certainly seems that either may fail. In the next chapters we will look at these methods again.

4.5 Conclusion

Using the simplex method for an illconditioned problem seems to be problematic, but used in the penalty method it works. However when the simplex or penalty method stops we have no indication of how good the solution is.

Chapter 5

Pattern Search

Pattern search is another "direct search", using only values of f. Of several variants in the literature I describe 2 for which I have made experiments.

5.1 Description of The Method

Pattern search starts with an initial point x_0, which will be our first primary basepoint, or x. We will now gradually move x closer to a minimum. We will also use a secondary basepoint called y.

The basic operation is the *Exploratory Test* (ET), which consists of evaluating the function in a number of points in a fixed distance from a basepoint.

Whenever an ET finds a point, y', with $f(y') < f(x)$, this points becomes the new primary basepoint. So we have found a direction "downhill" from x to y'. As in the simplex method we assume this "downhill" direction is good (forms a pattern — hence the name), and the method will look further in this direction.

When an ET does not find a new minimum we decrease the stepsize and try again.

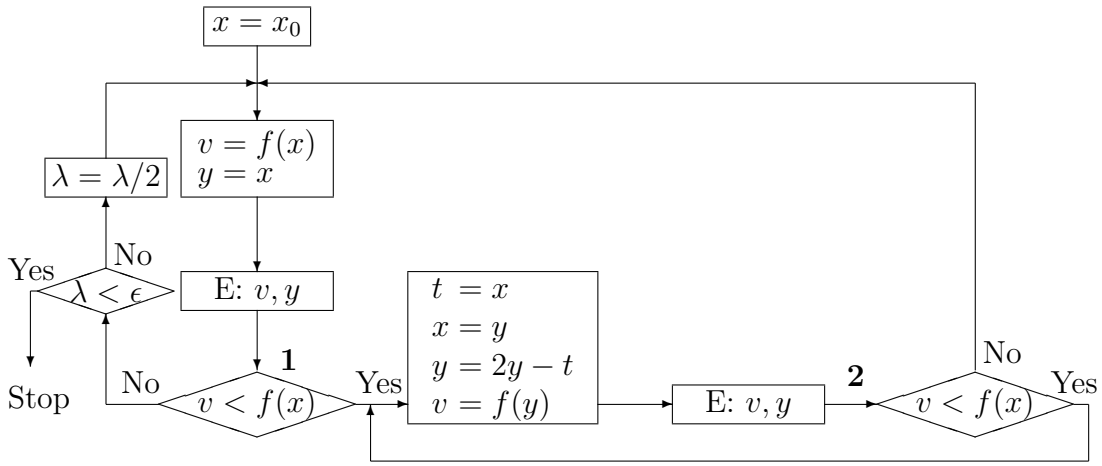

Figure 5.1: Pattern search.

5.1.1 Details

The pattern search algorithm as described in [HJ61] and [Ram76] is slightly complicated, and require a flowchart — see figure 5.1.

In this algorithm the ET consists of evaluating the function f in $2n$ new points, defined from either x or y as $x \pm \lambda e_i$ or $y \pm \lambda e_i$, where e_i is the i'th unitvector, and λ is the current stepsize.

The choice of $2n$ test-evaluations in the directions of the unitvectors is arbitrary, but they are easy to compute, and being orthogonal they are intuitively good.

In the figure $\boxed{\text{E: } v, y}$ indicates such an ET. If the ET is successful, v is changed to the new minimum value and y to the new minimum point. Otherwise they are unchanged.

When pattern search starts, and whenever an ET fails, we make the ET from an x. So at point **1** in figure 5.1 we may have the situation in figure 5.2, i.e.

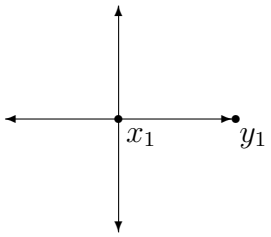

Figure 5.2: Pattern search at **1**.

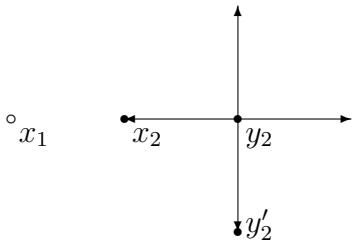

Figure 5.3: Pattern search at **2**.

starting from x_1 we have a new minimum in y_1. The new x_2 will be y_1, and the new y_2 will be at twice the distance from the old x_1. Now a new ET is tried from y_2, and perhaps a new minimum is found at y_2', as shown in figure 5.3.

An ET starting in an x can only move in the directions of the unitvectors, but an ET starting in a y can move in other directions. Figure 5.4 shows what happens after figure 5.3. The new x_3 will be y_2' and the new y_3 will once again be at twice the distance from the old x_2.

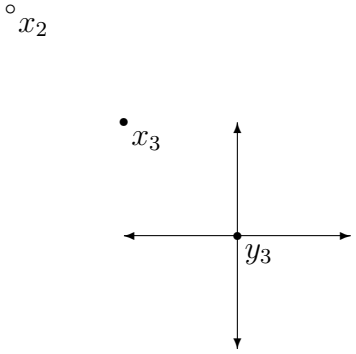

Figure 5.4: Pattern search at **2**.

A few things should be noted here:

1. Each new x will have a smaller function value than the previous one. However this does not guarantee that the process converges.

2. Even though we make an ET in the directions of the unitvectors, we may move the basepoints in other directions as well.

3. When we have found a good direction "downhill", we look further in that direction.

4. The process may compute the function value more than once for a given point.

5. We never increase λ.

5.1.2 Variants

We have a choice making an ET. We can either compute all $2n$ new values, and choose the smallest, or we can just compute the new values in some arbitrary order, and stop as soon as we find a smaller value. The latter seems to be standard, and is the one used in [HJ61] and [Ram76]. Below I try both.

When decreasing λ we could divide by some other number than 2.

5.2 The Program

In section D.3 the main routine in pattern search is shown. The ET shown in the program uses the variant where all $2n$ points are checked. The program should be equivalent to figure 5.1.

5.3 Experimental Results

First testrun of pattern search was with the standard method, i.e. use the first smaller value found. λ starts as 1, initial $x_0 = (0, 0, 0, 0, 0)$, and $\epsilon = 10^{-4}$ for each μ. Results are found in table 5.1.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	141
10^{-6}	12.4463	7.0321	2.1884	0.4044	4.7434	382.9335	562
10^{-5}	20.0046	10.0452	2.5863	1.8267	6.9216	599.6575	608
10^{-4}	21.7501	10.0411	2.1063	2.8188	7.0122	640.6688	1556
10^{-3}	21.9775	10.0083	2.0177	2.9796	6.9849	645.4570	5474
10^{-2}	21.9930	10.0714	2.0559	3.0437	6.8773	645.9708	49223
10^{-1}	22.0342	10.6143	2.5464	3.3236	5.7195	648.4160	243668
10^{0}	23.0000	12.0999	4.7499	3.2249	-0.0248	708.3689	554
10^{1}	23.0000	12.1000	4.7500	3.2250	-0.0250	708.3790	564
10^{2}	23.0000	12.1000	4.7500	3.2250	-0.0250	708.4325	545
10^{3}	23.0000	12.1000	4.7500	3.2250	-0.0250	708.9681	545
10^{4}	23.0000	12.1000	4.7500	3.2250	-0.0250	714.3239	545
10^{5}	23.0000	12.1000	4.7500	3.2250	-0.0250	767.8823	545
10^{6}	23.0000	12.1000	4.7500	3.2250	-0.0250	1303.4658	545
10^{7}	23.0000	12.1000	4.7500	3.2250	-0.0250	6659.3007	545

Table 5.1: Pattern search with $(\lambda, \epsilon, x_0) = (1, 10^{-4}, \mathbf{0})$, *Tryfirst*

We have the same tendency as in the simplex method, i.e. it does seem to converge correctly, and as the problem becomes illconditioned it converges to a false minimum. For $\mu = 10^{-1}$ it uses an extreme number of functioncalls. Most of these calls are made for the smallest values of λ (see also table 5.3), where we proceed very slowly to the right point.

In table 5.2 we try all points in an ET, and choose the one with the smallest value. For small values of μ this require more evalutions, but for larger μ, and thus more illconditioned problems, it is shows an advantage — especially for $\mu = 10^{-1}$. For this value of μ I have counted the number of evaluations for the different values of λ as shown in table 5.3, and it can be seen that most of the evaluations are used towards the end of the process.

As with the simplex method, we seem to have a fixed false minimum for higher values of μ. The reason seems to be the same, since an investigation the iteration processes for lower values of μ, shows that these processes get close to the false minimum, but manage to "turn around" and approach the true minimum. For higher values of μ this turning process requires λ to be

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	141
10^{-6}	12.4463	7.0321	2.1884	0.4044	4.7434	382.9335	902
10^{-5}	20.0046	10.0453	2.5864	1.8267	6.9214	599.6575	1015
10^{-4}	21.7500	10.0411	2.1062	2.8190	7.0126	640.6688	1153
10^{-3}	21.9751	10.0035	2.0114	2.9792	7.0012	645.4567	2402
10^{-2}	22.0430	9.9822	2.0314	2.9419	6.8987	645.9623	3220
10^{-1}	22.0607	9.9827	2.0465	2.9887	6.8506	646.0291	7547
10^{0}	24.9999	8.9393	4.1161	-0.6187	0.0332	722.2329	881
10^{1}	25.0000	8.9412	4.1177	-0.6177	0.0294	722.2839	608
10^{2}	25.0000	8.9412	4.1177	-0.6177	0.0294	722.3003	608
10^{3}	25.0000	8.9412	4.1177	-0.6177	0.0294	722.4651	608
10^{4}	25.0000	8.9412	4.1177	-0.6177	0.0294	724.1131	608
10^{5}	25.0000	8.9412	4.1177	-0.6177	0.0294	740.5926	608
10^{6}	25.0000	8.9412	4.1177	-0.6177	0.0294	905.3875	608
10^{7}	25.0000	8.9412	4.1177	-0.6177	0.0294	2553.3367	608

Table 5.2: Pattern search with $(\lambda, \epsilon, x_0) = (1, 10^{-4}, \mathbf{0})$, *Tryall*

reduced so much that the stop criterion is achieved.

Reducing ϵ to 10^{-5} for $\mu = 1$ gives the result in table 5.4, and as can be seen, the process has escaped the false minimum from before and stops close to the true minimum.

I should also note that the norm of the residual, i.e. $||A^\top Ax - A^\top b||_2$, for the points in table 5.2, look the same as in table 4.2 for the simplex method. It decreases for larger μ, and for the false minimum it is close to zero.

5.4 Variants

The general problem for pattern search seems to be that if we early on arrive at a point that looked like a minimum, but turned out not to be, it takes forever to get to the right minimum, because λ is small.

λ	$\#f_\mu$
2^0	205
2^{-1}	31
2^{-2}	10
2^{-3}	52
2^{-4}	31
2^{-5}	31
2^{-6}	10
2^{-7}	63
2^{-8}	52
2^{-9}	31
2^{-10}	3506
2^{-11}	42
2^{-12}	31
2^{-13}	3452

Table 5.3: Expanding the entry for $\mu = 10^{-1}$ from table 5.2.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^0	22.2243	9.8788	2.1233	2.7050	6.5594	646.3608	27320

Table 5.4: Pattern search with $(\lambda, \epsilon, x_0) = (1, 10^{-5}, \mathbf{0})$, *Tryall*

Also pattern search seems to suffer from the same problem we had with the simplex method, i.e. it has problems changing directions in narrow "valleys".

More complicated strategies to overcome these problems has been suggested in [BM69] and [EO66]. They use or suggest algorithms that among other things will

- select random points in the neighborhood of a failed pattern search.
- use two or more failed pattern searches to create a new pattern.
- increase or decrease λ in accordance with previous successes or failures of pattern searches.

- look in the opposite direction, when a pattern search fails.
- make ETs in random, but orthogonal, directions.
- use a line search in the direction of a pattern.

These suggestions appear to be good for our problem, but I haven't done any testing.

5.5 Using the Penalty Method

The results in table 5.5 differ from those of table 5.2 in that for each new μ we start in the point from the previous μ, as the penalty method prescribes.

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	$\#f_\mu$
10^{-6}	12.4463	7.0321	2.1884	0.4044	4.7434	382.9335	902
10^{-5}	20.0046	10.0452	2.5863	1.8267	6.9216	599.6575	927
10^{-4}	21.7507	10.0405	2.1063	2.8180	7.0117	640.6688	997
10^{-3}	21.9783	10.0044	2.0151	2.9767	6.9902	645.4569	1264
10^{-2}	21.9783	10.0046	1.9855	3.0197	7.0496	645.9491	425
10^{-1}	21.9783	10.0007	1.9791	3.0217	7.0632	646.0001	321
10^{0}	21.9783	10.0007	1.9789	3.0221	7.0637	646.0051	225
10^{1}	21.9783	10.0007	1.9789	3.0221	7.0637	646.0126	141
10^{2}	21.9783	10.0007	1.9789	3.0221	7.0637	646.0876	141
10^{3}	21.9783	10.0007	1.9789	3.0221	7.0637	646.8378	141
10^{4}	21.9783	10.0007	1.9789	3.0221	7.0637	654.3394	141
10^{5}	21.9783	10.0007	1.9789	3.0221	7.0637	729.3554	141
10^{6}	21.9783	10.0007	1.9789	3.0221	7.0637	1479.5156	141
10^{7}	21.9783	10.0007	1.9789	3.0221	7.0637	8981.1178	141

Table 5.5: Penalty method: Pattern search with $(\lambda, \epsilon, x_0) = (1, 10^{-4}, \mathbf{0})$, *Tryall*

The process doesn't get close to the false minimum from table 5.2, and stays close to the true minimum. The three stop-criteria from page 15 give the following results.

1. (Absolute error) No two successive function values are close by a difference of 10^{-4}.

2. (Relative error) With an $\epsilon = 10^{-4}$ we would stop with $\mu = 10^{-1}$, which is very close to the minimum value. Here 4836 functionevaluations are used.

3. (Decreasing differences) Up until $\mu = 1$ the differences between function values decrease, after which they start to increase. So here we would stop at $\mu = 1$, which isn't too bad.

5.6 Conclusion

As in the simplex method we have problems for illconditioned problems, but used with the penalty method it works.

Pattern search however uses about 4 times as many functionevaluations as the simplex method (with relative error as stopcriterion). Most of these are used for the many iterations for small λ, so a way to increase λ would probably make it more competitive.

Chapter 6

Newton-Raphson

The Newton-Raphson method differs from the previous two in that it does not use the values of f, but only the first and second derivatives. In general we are not guaranteed to find a minimum.

6.1 Description of the Method

The Newton-Raphson method is a one-point iterative method for solving systems of equations of the form

$$g(x) = 0 \tag{6.1}$$

We can use this for minimizing a function f, by solving

$$\nabla f(x) = 0 \tag{6.2}$$

since this is a necessary condition for a minimum. Unfortunately a solution to (6.2) can also be a maximum or a saddle point. For our test example we have exactly one minimum and no maximum or saddle point, so Newton-Raphson will be worth trying.

Solving (6.1) by Newton-Raphson consists of the following procedure. Choose an initial point, $x^{(0)}$, and perform a number of iterations:

$$x^{(k+1)} = x^{(k)} - (\nabla g(x^{(k)}))^{-1} g(x^{(k)}) \tag{6.3}$$

until successive points are sufficiently close. (6.3) involves an expensive matrixinversion, but the process can be rewritten:

$$\begin{array}{lcl} \nabla g(x^{(k)}) r^{(k+1)} & = & -g(x^{(k)}) \\ x^{(k+1)} & = & x^{(k)} + r^{(k+1)} \end{array} \tag{6.4}$$

which involves solving a system of linear equations. $\nabla g(x)$ is the Jacobian matrix for g.

When solving (6.2), we observe that $\nabla\nabla f(x)$ is in fact the Hessian of f, so that the equations (6.3) and (6.4) can be rewritten as:

$$x^{(k+1)} = x^{(k)} - (H_f(x^{(k)}))^{-1} \nabla f(x^{(k)}) \tag{6.5}$$

and

$$\begin{array}{lcl} H_f(x^{(k)}) r^{(k+1)} & = & -\nabla f(x^{(k)}) \\ x^{(k+1)} & = & x^{(k)} + r^{(k+1)} \end{array} \tag{6.6}$$

The following theorem is very useful when minimizing quadratic functions — as in the test problem.

Theorem 6.1 [See [PSU88, th. 3.1.4, p. 89]] *Suppose that A is a positive definite $n \times n$-matrix, that $b \in \mathbf{R}^n$, $x^{(0)} \in \mathbf{R}^n$, and that $a \in \mathbf{R}$. Then the quadratic function $f(x)$ of n variables defined by*

$$f(x) = a + bx + \tfrac{1}{2} x^\top A x \tag{6.7}$$

is strictly convex and has a unique global minimizer at the point $\hat{x}$ that is the unique solution of the system

$$Ax = -b$$

Moreover, the Newton's method sequence $\{x^{(k)}\}$ with initial point $x^{(0)}$ for minimizing $f(x)$ reaches $\hat{x}$ in one step, that is,

$$x^{(1)} = \hat{x}.$$

Proof: First we observe that for f in (6.7) we have

$$\nabla f(x) = b + Ax \text{ and } H_f(x) = A$$

Since A is positive definite, f is convex[1] and thus has x as global minimizer[2] when x is the solution to:

$$\begin{array}{c} \nabla f(x) = 0 \\ \Updownarrow \\ b + Ax = 0 \\ \Updownarrow \\ Ax = -b \end{array}$$

Choose any $x^{(0)}$ as initial point, and let $\hat{x}$ be the unique minimum point:

$$\begin{aligned} x^{(1)} &= x^{(0)} - (H_f(x^{(0)}))^{-1}\nabla f(x^{(0)}) \\ &= x^{(0)} - A^{-1}(b + Ax^{(0)}) \\ &= x^{(0)} - A^{-1}(-A\hat{x} + Ax^{(0)}) \\ &= x^{(0)} + \hat{x} - x^{(0)} \\ &= \hat{x} \end{aligned}$$

□

A problem with Newton-Raphson is that the system of linear equations in (6.6) may not have a unique solution or may be illconditioned.

The equation (3.4) is of the form (6.7), so we expect Newton-Raphson's method on the testproblem to be fast and accurate, provided we have a positive definite matrix.

6.2 The Program

The program in section D.5 is general, and does not use special properties of the given testproblem. Most specifically the Hessian is recalculated for every new x, and the resulting system of linear equations solved. In the

[1] [Ram76, p. 39] theorem 2.7
[2] [Ram76, p. 40] theorem 2.8 and [Ram76, p. 30] theorem 2.2

testproblem the Hessian is constant, and we could do an LU-decomposition once.

And even though we know the problem to be quadratic, requiring in theory one iteration, the program continues until the residues are sufficiently small (which is why the number of iterations in table 6.1 for most μ is 2).

The Newton-Raphson program uses the program in section D.4 to solve a system of linear equations.

6.3 Experimental Results

μ	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$	$f_\mu(\hat{x})$	#iter.
0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1
10^{-6}	12.4463	7.0321	2.1884	0.4044	4.7434	382.9335	2
10^{-5}	20.0046	10.0453	2.5864	1.8267	6.9215	599.6575	2
10^{-4}	21.7501	10.0411	2.1062	2.8188	7.0125	640.6688	2
10^{-3}	21.9742	10.0047	2.0114	2.9808	7.0016	645.4567	2
10^{-2}	21.9974	10.0005	2.0012	2.9981	7.0002	645.9456	2
10^{-1}	21.9997	10.0000	2.0001	2.9998	7.0000	645.9946	2
10^{0}	22.0000	10.0000	2.0000	3.0000	7.0000	645.9995	2
10^{1}	22.0000	10.0000	2.0000	3.0000	7.0000	645.9999	2
10^{2}	22.0000	10.0000	2.0000	3.0000	7.0000	646.0000	2
10^{3}	22.0000	10.0000	2.0000	3.0000	7.0000	646.0000	2
10^{4}	22.0000	10.0000	2.0000	3.0000	7.0000	646.0000	2
10^{5}	22.0000	10.0000	2.0000	3.0000	7.0000	646.0000	3
10^{6}	22.0000	10.0000	2.0000	3.0000	7.0000	646.0000	263
10^{7}	*Not converging?*						

Table 6.1: Newton-Raphson

A testrun of Newton-Raphson's method with $\epsilon = 10^{-8}$, gives the results in table 6.1. As expected convergence is quick, but as μ grows large we seem to get problems.

If we look at the iteration process for $\mu = 10^6$ we can see that we get close to the true minimum very fast (in fact with $\epsilon = 10^{-4}$ the last 3 entries in the table would need only 2 iterations). For the next 260 iterations x wanders aimlessly around the true minimum, and then suddenly jumps to the right value — and even with high precision.

For $\mu = 10^7$ the process quickly enters a loop of 3 different points, close to the true minimum.

Calculating the singular values of $I + \mu(A^\top A)^2$ for $\mu = 10^7$ using the routine from [NRP] gives:

$$\sigma = (1.5575\text{e}^{13}, 1.6000\text{e}^{12}, 1.4746\text{e}^{12}, 1, 1)$$

showing that the matrix is "almost" singular and "almost" has rank 3, since there are two relatively small singular values. This is no big surprise, since $(A^\top A)^2$ is singular and we only add small numbers to the diagonal. The 2-norm (or spectral) condition number is $\kappa(I + \mu(A^\top A)^2) \approx 1.5575\text{e}^{13}$ which is large and should make us wonder how the Gaussean elimination performs.

A simple way to check the Gaussean elimination is to look for small pivots. The implementation I have chosen uses partial pivoting, and for $\mu = 10^6$ we get these 4 pivots:

$$2.05337600000200\text{e}^{12}$$
$$2.20495375641558\text{e}^{11}$$
$$2.68592030072611\text{e}^{11}$$
$$3.93595043197274$$

Here 4 is a small pivot since the size of the elements in the matrix is around $10^9 \ldots 10^{12}$. So the Gaussean elimination cannot be expected to give an accurate answer. This means that the $r^{(k+1)}$ from (6.6) is not accurate, and thus we cannot expect theorem 6.1 to hold.

6.4 Solving with the Penalty Method

In view of theorem 6.1 it makes little sense to repeat the testruns from table 6.1 with the penalty method. No matter what start point we choose, we

would converge to exactly the same point. Perhaps for the last 3 values of μ the results would differ a little, but we would have stopped the process before.

Either of the 3 stop-criteria from page 15 would work here with good results.

- (Absolute error) For $\epsilon = 10^{-4}$ we would stop at $\mu = 10^2$.
- (Relative error) For $\epsilon = 10^{-4}$ we would stop at $\mu = 10^{-1}$.
- (Increasing differences) We would stop around $\mu = 10^3$.

6.5 Conclusion

The Newton-Raphson method is fast for convex functions, as theorem 6.1 shows. Even for moderately illconditioned problems it converges, but it has a limit. Of the 3 methods considered it is certainly the best for the given testproblem, and will be used in the Hestenes-Powell method (next chapter), where we need to minimize a sequence of problems.

Chapter 7

Hestenes-Powell

A way to avoid the illconditioning encountered using a penaltyfunction was proposed in [Hes69]. This method will converge to the true minimum, provided the penaltyparameter is large enough.

7.1 The Theory Behind

For a problem of the form

$$\min_x \ f(x), \text{ where } h(x) = 0$$

a new function is constructed

$$f_\mu(x, \lambda) = f(x) + \lambda^\top h(x) + \frac{1}{2}\mu h(x)^\top h(x)$$

This function is used to find $\hat{x}$ and $\hat{\lambda}$, such that the Lagrange function $L(x, \lambda) = f(x) + \lambda^\top h(x)$ (see section 2.3) has a minimum, or rather, such that

$$\nabla L(x, \lambda) = \begin{bmatrix} \nabla_x L(x, \lambda) \\ \nabla_\lambda L(x, \lambda) \end{bmatrix} = \begin{bmatrix} \nabla f(x) + J_h(x)^\top \lambda \\ h(x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad (7.1)$$

where $J_h(x)$ is the Jacobian matrix for h at x (see e.g. [WCT72, p. 255]).

Choose an initial $\lambda^{(0)}$, and perform these iteration steps

$$\begin{array}{rcl} x^{(k)} & = & \min_x f_\mu(x, \lambda^{(k)}) \\ \lambda^{(k+1)} & = & \lambda^{(k)} + \mu h(x^{(k)}) \end{array}$$

until $||x^{(k+1)} - x^{(k)}|| < \epsilon$.

After each iteration we have

$$\begin{array}{rcl} \nabla_x f_\mu(x^{(k)}, \lambda^{(k)}) & = & 0 \\ \Updownarrow & & \\ \nabla f(x^{(k)}) + J_h(x^{(k)})^\top \lambda^{(k)} + \mu J_h(x^{(k)})^\top h(x^{(k)}) & = & 0 \\ \Updownarrow & & \\ \nabla f(x^{(k)}) + J_h(x^{(k)})^\top (\lambda^{(k)} + \mu h(x^{(k)})) & = & 0 \\ \Updownarrow & & \\ \nabla f(x^{(k)}) + J_h(x^{(k)})^\top \lambda^{(k+1)} & = & 0 \end{array}$$

so that for $x^{(k)}$ and $\lambda^{(k+1)}$ the first condition from (7.1) is satisfied. However in general $h(x^{(k)}) \neq 0$. When the process converges for $\lambda^{(k)}$, we have — at least in the limit — that $\lambda^{(k+1)} = \lambda^{(k)}$, and thus that $h(x^{(k)}) = 0$.

In this way (7.1) is satisfied. In [Ram76, p. 223f] these criteria are listed for the functions involved:

1. $f, h \in C^2$ in a neighbourhood around the minimum.
2. The minimum is a regular point.
3. The Hessian of the Lagrange function is positive definite for some points.

When these 3 criteria are satisfied [Ram76] prove that there exists a μ_0 such that $H^x_{f_\mu}$ is positive definite for $\mu > \mu_0$, such that we have a convex problem.

And for our wellbehaved testproblem they are satisfied. f, h are C^2 for all x, all points are regular when the constraints are linear ([Ram76, p. 180]) and the Hessian here is a constant matrix, which is positive definite.

So we will expect the testproblem to perform well with the method of Hestenes-Powell.

7.2 An Example

Continuing example 3.1 we get ([Hes75, p. 267]):

Example 7.1 We want to minimize $f(x) = x_1^2 - x_2^2 - 4x_2$ subject to the constraint $h(x) = x_2 = 0$. Now we define

$$\begin{aligned} f_\mu(x) &= f(x) + \lambda h(x) + \frac{1}{2}\mu h(x)^2 \\ &= x_1^2 - x_2^2 - 4x_2 + \lambda x_2 + \frac{1}{2}\mu x_2^2 \\ &= x_1^2 + (\lambda - 4)x_2 + \frac{\mu - 2}{2}x_2^2 \end{aligned}$$

which has a minimum at $(0, 0)$ if $\lambda = 4$ and $\mu > 2$.

The method of Hestenes-Powell will not work in this case, since the λ's will diverge, unless $\lambda^{(0)} = 4$. □

7.3 The Program

The program in section D.6 contains definitions of the following 3 functions:

$$\begin{array}{lcl} f_\mu(x,\lambda) & = & x^\top x + \lambda^\top (A^\top Ax - A^\top b) + \frac{1}{2}\mu (A^\top Ax - A^\top b)^\top (A^\top Ax - A^\top b) \\ \nabla_x f_\mu(x,\lambda) & = & 2x + \lambda^\top A^\top A + \frac{1}{2}\mu (2(A^\top A)^2 x - 2b^\top AA^\top A) \\ H^x_{f_\mu}(x,\lambda) & = & 2 + \mu (A^\top A)^2 \end{array}$$

In the procedure `TestInit` $A^\top A$ and $A^\top b$ are calculated, and also:

$$\begin{aligned} Q' &= (A^\top A)^2 \\ P' &= b^\top AA^\top A \end{aligned}$$

7.4 Experimental Results

For the results in table 7.1 $\lambda^{(0)} = 0$ and $\epsilon = 10^{-4}$. For the embedded Newton-Raphson an $\epsilon = 10^{-8}$ was chosen, but as seen in the previous chapter this choice is unimportant for the μ we see in the table.

μ		1	2	3	4	5	#it.
0	λ	*0.0000*	*0.0000*	*0.0000*	*0.0000*	*0.0000*	1
	x	0.0000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000	
10^{-7}	λ	*-0.0471*	*-0.0123*	*0.0030*	*-0.0150*	*-0.0097*	473
	x	21.9451215716	10.0568912722	2.0708527407	2.9216759355	7.0314327923	
10^{-6}	λ	*-0.0937*	*-0.0376*	*-0.0043*	*-0.0175*	*-0.0268*	83
	x	21.9949191105	10.0039022607	2.0074381427	2.9929679801	7.0031297592	
10^{-5}	λ	*-0.5571*	*-0.2928*	*-0.0806*	*-0.0389*	*-0.1996*	17
	x	21.9996741800	10.0002378870	2.0005016858	2.9995552365	7.0002068764	
10^{-4}	λ	*-5.1911*	*-2.8444*	*-0.8438*	*-0.2520*	*-1.9276*	7
	x	21.9999909872	10.0000071085	2.0000128208	2.9999874331	7.0000054583	
10^{-3}	λ	*-51.5303*	*-28.3612*	*-8.4758*	*-2.3832*	*-19.2076*	5
	x	21.9999998900	10.0000000923	2.0000001449	2.9999998439	7.0000000637	
10^{-2}	λ	*-514.9223*	*-283.5292*	*-84.7958*	*-23.6952*	*-192.0076*	4
	x	21.9999999911	10.0000000077	2.0000000110	2.9999999873	7.0000000049	
10^{-1}	λ	*-5148.8423*	*-2835.2092*	*-847.9958*	*-236.8152*	*-1920.0076*	4
	x	22.0000000000	10.0000000000	2.0000000000	3.0000000000	7.0000000000	
10^{0}	λ	*-51488.0423*	*-28352.0092*	*-8479.9958*	*-2368.0152*	*-19200.0076*	3
	x	21.9999999993	10.0000000006	2.0000000008	2.9999999990	7.0000000003	

Table 7.1: Hestenes-Powell

It seems that the process converges to the true minimum for sufficiently large μ, i.e. $\mu = 10^{-1}$. With more decimals than shown we can see that we are not exactly at the optimum, but probably as close as we can get, remembering that the underlying Newton-Raphson uses a Gaussean elimination which is not accurate for the Hessian for this problem. For $\mu > 1$ the solutions look like the last two in the table, i.e. they do not improve further.

7.5 Conclusion

It looks as if the method of Hestenes-Powell could perform well for more complicated functions and constraints, but this will definitely require more experiments.

If μ is chosen large enough from the start, the amount of work will be minimal. The method is more expensive than Newton-Raphson for one value of μ, but

since Newton-Raphson combined with the penalty method would need a number of different μ's, the Hestenes-Powell method could easily be more efficient.

Any comparison between the two should really not be done with a quadratic problem, since this favours Newton-Raphson.

Appendix A

An Analytic Solution of the Testproblem

Let A and b be defined as:

$$A = \begin{bmatrix} 22 & 10 & 2 & 3 & 7 \\ 14 & 7 & 10 & 0 & 8 \\ -1 & 13 & -1 & -11 & 3 \\ -3 & -2 & 13 & -2 & 4 \\ 9 & 8 & 1 & -2 & 4 \\ 9 & 1 & -7 & 5 & -1 \\ 2 & -6 & 6 & 5 & 1 \\ 4 & 5 & 0 & -2 & 2 \end{bmatrix} \text{ and } b = \begin{bmatrix} 646 \\ 454 \\ 94 \\ -38 \\ 302 \\ 202 \\ 18 \\ 146 \end{bmatrix} \tag{A.1}$$

A.1 The Linear Least Squares Problem

If we can solve the equation $Ax = b$ we have a solution that will minimize $||Ax - b||_2$, since in that case the norm is 0.

The equation $Ax = b$ defines an overdetermined system of linear equations,

which will be solved by Gaussian elimination.

$$\begin{array}{rcrcrcrcrcr}
22x_1 & + & 10x_2 & + & 2x_3 & + & 3x_4 & + & 7x_5 & = & 646 \\
14x_1 & + & 7x_2 & + & 10x_3 & & & + & 8x_5 & = & 454 \\
-1x_1 & + & 13x_2 & - & 1x_3 & - & 11x_4 & + & 3x_5 & = & 94 \\
-3x_1 & - & 2x_2 & + & 13x_3 & - & 2x_4 & + & 4x_5 & = & -38 \\
9x_1 & + & 8x_2 & + & 1x_3 & - & 2x_4 & + & 4x_5 & = & 302 \\
9x_1 & + & 1x_2 & - & 7x_3 & + & 5x_4 & - & 1x_5 & = & 202 \\
2x_1 & - & 6x_2 & + & 6x_3 & + & 5x_4 & + & 1x_5 & = & 18 \\
4x_1 & + & 5x_2 & & & - & 2x_4 & + & 2x_5 & = & 146
\end{array} \tag{A.2}$$

$$\begin{array}{rcrcrcrcrcr}
22x_1 & + & 10x_2 & + & 2x_3 & + & 3x_4 & + & 7x_5 & = & 646 \\
 & & 7x_2 & + & 96x_3 & - & 21x_4 & + & 39x_5 & = & 472 \\
 & & 296x_2 & - & 20x_3 & - & 239x_4 & + & 73x_5 & = & 2714 \\
 & & -14x_2 & + & 292x_3 & - & 35x_4 & + & 109x_5 & = & 1102 \\
 & & 86x_2 & + & 4x_3 & - & 71x_4 & + & 25x_5 & = & 830 \\
 & & -68x_2 & - & 172x_3 & + & 83x_4 & - & 85x_5 & = & -1370 \\
 & & -76x_2 & + & 64x_3 & + & 52x_4 & + & 4x_5 & = & -448 \\
 & & 35x_2 & - & 4x_3 & - & 28x_4 & + & 8x_5 & = & 314
\end{array} \tag{A.3}$$

$$\begin{array}{rcrcrcrcrcr}
22x_1 & + & 10x_2 & + & 2x_3 & + & 3x_4 & + & 7x_5 & = & 646 \\
 & & 7x_2 & + & 96x_3 & - & 21x_4 & + & 39x_5 & = & 472 \\
 & & & & 44x_3 & - & 7x_4 & + & 17x_5 & = & 186 \\
 & & & & 44x_3 & - & 7x_4 & + & 17x_5 & = & 186 \\
 & & & & 44x_3 & - & 7x_4 & + & 17x_5 & = & 186 \\
 & & & & 44x_3 & - & 7x_4 & + & 17x_5 & = & 186 \\
 & & & & 44x_3 & - & 7x_4 & + & 17x_5 & = & 186 \\
 & & & & 44x_3 & - & 7x_4 & + & 17x_5 & = & 186
\end{array} \tag{A.4}$$

So we have 2 degrees of freedom and choose arbitrarily to let x_4 and x_5 be free variables. We get from (A.4):

$$\begin{array}{rcrcrcr}
x_3 & = & \frac{7}{44}x_4 & - & \frac{17}{44}x_5 & + & \frac{186}{44} \\
x_2 & = & \frac{9}{11}x_4 & - & \frac{3}{11}x_5 & + & \frac{104}{11} \\
x_1 & = & -\frac{23}{44}x_4 & - & \frac{7}{44}x_5 & + & \frac{1086}{44}
\end{array} \tag{A.5}$$

or with s and t as parameters:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} \frac{1086}{44} \\ \frac{104}{11} \\ \frac{93}{22} \\ 0 \\ 0 \end{bmatrix} + s \begin{bmatrix} -\frac{23}{44} \\ \frac{9}{11} \\ \frac{7}{44} \\ 1 \\ 0 \end{bmatrix} + t \begin{bmatrix} -\frac{7}{44} \\ -\frac{3}{11} \\ -\frac{17}{44} \\ 0 \\ 1 \end{bmatrix} \tag{A.6}$$

So (A.6) gives all the solutions in the form $S = \hat{x} + \mathcal{N}(A)$, and all of these will make $||Ax - b||_2 = 0$ and thus minimize it. But which of these will have minimum l_2-norm?

A.2 The Optimal Linear Least Squares Problem

We wish to minimize $||x||_2 = \sqrt{x^\top x}$, where $x \in S$. This is of course the same as minimizing $x^\top x$. So let $x \in S$.

$$\begin{aligned}
&x^\top x = \\
&(\tfrac{1086}{44} - \tfrac{23}{44}s - \tfrac{7}{44}t, \tfrac{104}{11} + \tfrac{9}{11}s - \tfrac{3}{11}t, \tfrac{93}{22} + \tfrac{7}{44}s - \tfrac{17}{44}t, s, t)^\top (\tfrac{1086}{44} - \tfrac{23}{44}s - \tfrac{7}{44}t, \tfrac{104}{11} + \tfrac{9}{11}s - \tfrac{3}{11}t, \\
&\tfrac{93}{22} + \tfrac{7}{44}s - \tfrac{17}{44}t, s, t) = \\
&(\tfrac{1086}{44} - \tfrac{23}{44}s - \tfrac{7}{44}t)^2 + (\tfrac{104}{11} + \tfrac{9}{11}s - \tfrac{3}{11}t)^2 + (\tfrac{93}{22} + \tfrac{7}{44}s - \tfrac{17}{44}t)^2 + s^2 + t^2
\end{aligned}$$

Many tedious calculations result in:

$$\begin{array}{rcrcrcrcrcrcr}
(\frac{1086}{44})^2 & - & 2\frac{23\cdot 1086}{44\cdot 44}s & - & 2\frac{7\cdot 1086}{44\cdot 44}t & + & \frac{23\cdot 23}{44\cdot 44}s^2 & + & 2\frac{7\cdot 23}{44\cdot 44}st & + & \frac{7\cdot 7}{44\cdot 44}t^2 & + \\
(\frac{104}{11})^2 & + & 2\frac{9\cdot 104}{11\cdot 11}s & - & 2\frac{3\cdot 104}{11\cdot 11}t & + & \frac{9\cdot 9}{11\cdot 11}s^2 & - & 2\frac{9\cdot 3}{11\cdot 11}st & + & \frac{3\cdot 3}{11\cdot 11}t^2 & + \\
(\frac{93}{22})^2 & + & 2\frac{7\cdot 93}{22\cdot 44}s & - & 2\frac{93\cdot 17}{22\cdot 44}t & + & \frac{7\cdot 7}{44\cdot 44}s^2 & - & 2\frac{7\cdot 17}{44\cdot 44}st & + & \frac{17\cdot 17}{44\cdot 44}t^2 & + \\
 & & & & & & s^2 & & & + & t^2 & = \\
\frac{1387048}{44\cdot 44} & - & \frac{17400}{44\cdot 44}s & - & \frac{31512}{44\cdot 44}t & + & \frac{3810}{44\cdot 44}s^2 & - & \frac{780}{44\cdot 44}st & + & \frac{2418}{44\cdot 44}t^2 &
\end{array}$$

After disposing of the constant and reducing the coefficients we have this quadratic equation, which has the same minimum points.

$$f(s,t) = -2900s - 5252t + 635s^2 - 130st + 403t^2$$

A necessary condition for a minimum point for f is that[1] the gradient is 0:

$$\nabla f(s,t) = \begin{bmatrix} \frac{\partial f}{\partial s}(s,t) \\ \frac{\partial f}{\partial t}(s,t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{A.7}$$

So we calculate

$$\nabla f(s,t) = \begin{bmatrix} \frac{\partial f}{\partial s}(s,t) \\ \frac{\partial f}{\partial t}(s,t) \end{bmatrix} = \begin{bmatrix} -2900 & + & 2 \cdot 635s & - & 130t \\ -5252 & - & 130s & + & 2 \cdot 403t \end{bmatrix}$$

and thus need to solve these linear equations:

$$\begin{array}{rcrcl} 1270s & - & 130t & = & 2900 \\ -130s & + & 806t & = & 5252 \end{array}$$

and this gives:

$$\begin{array}{rcl} s & = & 3 \\ t & = & 7 \end{array} \tag{A.8}$$

But is this point a minimum, a maximum — or neither?

To check this we need the Hessian matrix:

$$H_f(s,t) = \begin{bmatrix} \frac{\partial^2 f}{\partial s,s} & \frac{\partial^2 f}{\partial s,t} \\ \frac{\partial^2 f}{\partial t,s} & \frac{\partial^2 f}{\partial t,t} \end{bmatrix} = \begin{bmatrix} 1270 & -130 \\ -130 & 806 \end{bmatrix}$$

Since this matrix has positive diagonal elements and is diagonally dominant, then it is positive definite, and thus[2] $(s,t) = (3,7)$ is a local minimum.

Since $H_f(x,y)$ is positive definite on $\mathbf{R}^2$, it is also[3] convex on $\mathbf{R}^2$. And then[4] every local minimum is also a global minimum.

---

[1] [Ram76, p. 29] corollary to theorem 2.1
[2] [Ram76, p. 30] theorem 2.2
[3] [Ram76, p. 39] theorem 2.7
[4] [Ram76, p. 40] theorem 2.8

So we have that

$$\hat{x} = \begin{bmatrix} \frac{1086}{44} \\ \frac{104}{11} \\ \frac{93}{22} \\ 0 \\ 0 \end{bmatrix} + 3 \begin{bmatrix} -\frac{23}{44} \\ \frac{9}{11} \\ \frac{7}{44} \\ 1 \\ 0 \end{bmatrix} + 7 \begin{bmatrix} -\frac{7}{44} \\ -\frac{3}{11} \\ -\frac{17}{44} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 22 \\ 10 \\ 2 \\ 3 \\ 7 \end{bmatrix} \tag{A.9}$$

is the solution to the optimal least squares problem for the given A and b, and we also have that

$$\begin{array}{rcl} ||\hat{x}||_2^2 & = & 646 \\ ||\hat{x}||_2 & \approx & 25.4165 \end{array}$$

A.2.1 More Numbers

We might as well continue with some calculations needed elsewhere in the report.

$$A^\top A = \begin{bmatrix} 872 & 400 & 104 & 112 & 288 \\ 400 & 448 & 16 & -160 & 192 \\ 104 & 16 & 360 & -16 & 160 \\ 112 & -160 & -16 & 192 & -32 \\ 288 & 192 & 160 & -32 & 160 \end{bmatrix} \tag{A.10}$$

$$(A^\top A)^2 = \begin{bmatrix} 1026688 & 567040 & 178816 & 44288 & 387072 \\ 567040 & 42324 & 87808 & -64000 & 239616 \\ 178816 & 87808 & 166528 & -4864 & 116736 \\ 44288 & -64000 & -4864 & 76288 & -12288 \\ 387072 & 239616 & 116736 & -12288 & 172032 \end{bmatrix} \tag{A.11}$$

$A^\top A$ is singular, and — no surprise — so is $(A^\top A)^2$. We also need:

$$A^\top b = \begin{bmatrix} 25744 \\ 14176 \\ 4240 \\ 1184 \\ 9600 \end{bmatrix} \tag{A.12}$$

A.3 Singular Value Decomposition

Using a routine from [NRP] I have made the following singular value decomposition. The calculations were done with extended IEEE-reals, and is shown with 5 digits precision in table A.1.

$$U \approx \begin{bmatrix} -7.0711\text{e}^{-01} & 1.5811\text{e}^{-01} & 1.7678\text{e}^{-01} & 5.4307\text{e}^{-01} & -2.4731\text{e}^{-01} \\ -5.3033\text{e}^{-01} & 1.5811\text{e}^{-01} & -3.5355\text{e}^{-01} & -5.4699\text{e}^{-01} & 6.7877\text{e}^{-02} \\ -1.7678\text{e}^{-01} & -7.9057\text{e}^{-01} & -1.7678\text{e}^{-01} & 3.0029\text{e}^{-01} & 4.2010\text{e}^{-01} \\ -3.7181\text{e}^{-20} & 1.5811\text{e}^{-01} & -7.0711\text{e}^{-01} & 1.0071\text{e}^{-01} & 2.0937\text{e}^{-01} \\ -3.5355\text{e}^{-01} & -1.5811\text{e}^{-01} & -1.1615\text{e}^{-19} & -1.5579\text{e}^{-01} & -2.1655\text{e}^{-01} \\ -1.7678\text{e}^{-01} & 1.5811\text{e}^{-01} & 5.3033\text{e}^{-01} & -1.8845\text{e}^{-01} & 6.8860\text{e}^{-01} \\ 1.3666\text{e}^{-20} & 4.7434\text{e}^{-01} & -1.7678\text{e}^{-01} & 3.6857\text{e}^{-01} & 4.2515\text{e}^{-01} \\ -1.7678\text{e}^{-01} & -1.5811\text{e}^{-01} & -1.9872\text{e}^{-19} & -3.3157\text{e}^{-01} & 1.1001\text{e}^{-01} \end{bmatrix}$$

$$\text{diag}(\Sigma) \approx \begin{bmatrix} 3.5327\text{e}^{+01} & 2.0000\text{e}^{+01} & 1.9596\text{e}^{+01} & 9.6917\text{e}^{-19} & 7.2893\text{e}^{-19} \end{bmatrix}$$

$$V^\top \approx \begin{bmatrix} -8.0064\text{e}^{-01} & -4.8038\text{e}^{-01} & -1.6013\text{e}^{-01} & 1.2134\text{e}^{-19} & -3.2026\text{e}^{-01} \\ 3.1623\text{e}^{-01} & -6.3246\text{e}^{-01} & 3.1623\text{e}^{-01} & 6.3246\text{e}^{-01} & 2.9463\text{e}^{-19} \\ 2.8868\text{e}^{-01} & -5.1712\text{e}^{-19} & -8.6603\text{e}^{-01} & 2.8868\text{e}^{-01} & -2.8868\text{e}^{-01} \\ -4.1910\text{e}^{-01} & 4.4051\text{e}^{-01} & -5.2005\text{e}^{-02} & 6.7606\text{e}^{-01} & 4.1298\text{e}^{-01} \\ 0.0000\text{e}^{+00} & 4.1855\text{e}^{-01} & 3.4879\text{e}^{-01} & 2.4415\text{e}^{-01} & -8.0222\text{e}^{-01} \end{bmatrix}$$

Table A.1: Singular value decomposition of A.

Some of the singular values are very small, around 10^{-18}. It is reasonable to assume that they should be 0. This leaves 3 nonzero singular values, meaning that the rank of A is 3 (which we knew already).

Calculating A^+ I set the two small values explicitly to 0. In table A.2 A^+ (or rather its transpose for typographical reasons) is shown, and calculating $\hat{x} = A^+ b$ we get the expected result from (A.9) with at least 15 correct digits (not shown).

$$A^{+\top} = \begin{bmatrix} 2.1130\text{e}^{-02} & 4.6154\text{e}^{-03} & -2.1074\text{e}^{-03} & 7.6042\text{e}^{-03} & 3.8061\text{e}^{-03} \\ 9.3109\text{e}^{-03} & 2.2115\text{e}^{-03} & 2.0529\text{e}^{-02} & -2.0833\text{e}^{-04} & 1.0016\text{e}^{-02} \\ -1.1098\text{e}^{-02} & 2.7404\text{e}^{-02} & -3.8862\text{e}^{-03} & -2.7604\text{e}^{-02} & 4.2067\text{e}^{-03} \\ -7.9167\text{e}^{-03} & -5.0000\text{e}^{-03} & 3.3750\text{e}^{-02} & -5.4167\text{e}^{-03} & 1.0417\text{e}^{-02} \\ 5.5128\text{e}^{-03} & 9.8077\text{e}^{-03} & -8.9744\text{e}^{-04} & -5.0000\text{e}^{-03} & 3.2051\text{e}^{-03} \\ 1.4319\text{e}^{-02} & -2.5962\text{e}^{-03} & -2.0136\text{e}^{-02} & 1.2813\text{e}^{-02} & -6.2099\text{e}^{-03} \\ 4.8958\text{e}^{-03} & -1.5000\text{e}^{-02} & 1.5313\text{e}^{-02} & 1.2396\text{e}^{-02} & 2.6042\text{e}^{-03} \\ 1.5064\text{e}^{-03} & 7.4038\text{e}^{-03} & -1.6987\text{e}^{-03} & -5.0000\text{e}^{-03} & 1.6026\text{e}^{-03} \end{bmatrix}$$

Table A.2: The pseudo-inverse of A

Appendix B

Results from Linear Algebra

In this appendix I summarize some of the definitions and results from linear algebra used previously, and I prove some of these.

B.1 Matrices and Linear Transformations

A real $m \times n$ matrix A has an associated linear transformation f_A: $\mathbf{R}^n \to \mathbf{R}^m$, defined by $f_A(x) = Ax$. Definitions and results can be formulated in terms of either matrices or linear transformation.

Definition B.1 *The* range *of a matrix A is the subspace defined as*

$$\mathcal{R}(A) = \{Ax \in \mathbf{R}^m \mid x \in \mathbf{R}^n\} \subseteq \mathbf{R}^m$$

The null space *of A is the subspace defined as*

$$\mathcal{N}(A) = \{x \in \mathbf{R}^n \mid Ax = 0\} \subseteq \mathbf{R}^n$$

The rank *of A is defined as*

$$\text{rank}(A) = \dim(\mathcal{R}(A))$$

and similarly the nullity *of A is*

$$\text{null}(A) = \dim(\mathcal{N}(A))$$

□

Lemma B.2 *For any A:*

$$\begin{aligned} \mathcal{N}(A) &= \mathcal{R}(A^\top)^\perp && \text{(B.1)} \\ \mathcal{N}(A)^\perp &= \mathcal{R}(A^\top) && \text{(B.2)} \\ \mathcal{N}(A^\top) &= \mathcal{R}(A)^\perp && \text{(B.3)} \\ \mathcal{N}(A^\top)^\perp &= \mathcal{R}(A) && \text{(B.4)} \end{aligned}$$

Proof: We only need to prove (B.1):

$$\begin{array}{l} x \in \mathcal{N}(A) \\ \Updownarrow \\ Ax = 0 \\ \Updownarrow \\ \forall y:\ y^\top Ax = 0 \\ \Updownarrow \\ \forall y:\ (A^\top y)^\top x = 0 \\ \Updownarrow \\ \forall z \in \mathcal{R}(A^\top):\ x \perp z \\ \Updownarrow \\ x \in \mathcal{R}(A^\top)^\perp \end{array}$$

Now (B.2)–(B.4) follows from $A^{\top\top} = A$ and $S^{\perp\perp} = S$ for A a matrix and S a linear subspace. □

Lemma B.3 *For any A:*

$$\begin{aligned} \mathcal{N}(A^\top) &= \mathcal{N}(AA^\top) && \text{(B.5)} \\ \mathcal{N}(A) &= \mathcal{N}(A^\top A) && \text{(B.6)} \\ \mathcal{R}(A^\top) &= \mathcal{R}(A^\top A) && \text{(B.7)} \\ \mathcal{R}(A) &= \mathcal{R}(AA^\top) && \text{(B.8)} \end{aligned}$$

Proof: I first prove (B.5):

$$
\begin{array}{ll}
 & x \in \mathcal{N}(A^\top) \\
\Updownarrow & \\
 & A^\top x = 0 \\
\Downarrow & \\
 & AA^\top x = 0 \\
\Updownarrow & \\
 & x \in \mathcal{N}(AA^\top)
\end{array}
\qquad
\begin{array}{ll}
 & x \in \mathcal{N}(AA^\top) \\
\Updownarrow & \\
 & AA^\top x = 0 \\
\Downarrow & \\
 & x^\top AA^\top x = 0 \\
\Updownarrow & \\
 & \left(A^\top x\right)^\top A^\top x = 0 \\
\Updownarrow & \\
 & ||A^\top x||_2^2 = 0 \\
\Updownarrow & \\
 & A^\top x = 0 \\
\Updownarrow & \\
 & x \in \mathcal{N}(A^\top)
\end{array}
$$

Now the rest is easy. First (B.6):

$$\mathcal{N}(A) = \mathcal{N}((A^\top)^\top) = \mathcal{N}((A^\top)(A^\top)^\top) = \mathcal{N}(A^\top A)$$

(B.7):

$$\mathcal{R}(A^\top) = \mathcal{N}(A)^\perp = \mathcal{N}(A^\top A)^\perp = \mathcal{R}((A^\top A)^\top) = \mathcal{R}(A^\top A)$$

(B.8):

$$\mathcal{R}(A) = \mathcal{N}(A^\top)^\perp = \mathcal{N}(AA^\top)^\perp = \mathcal{R}((AA^\top)^\top) = \mathcal{R}(AA^\top)$$

□

B.2 Singular Value Decomposition

It is known from linear algebra that a real $m \times n$ matrix A, can be written as

$$A = U\Sigma V^\top \tag{B.9}$$

where

$$U^\top U = V^\top V = VV^\top = I_n \text{ and } \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n).$$

I.e. V is a $n \times n$ orthogonal matrix, U is an $m \times n$ matrix with orthonormal columns, and Σ is a $n \times n$ diagonal matrix.

The diagonal elements of Σ are the non-negative square roots of the eigenvalues of $A^\top A$, and are called the *singular values* of A. Rank(A) is the number of non-zero singular values. The decomposition in (B.9) is called the *singular value decomposition* of A (in [GR71, p. 134] other ways to define the singular value decomposition can be found).

The decomposition is not unique, and we may choose it so that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \tag{B.10}$$

The non-zero singular values will be the first in this list, followed by those that are zero.

B.3 The Pseudo-inverse of a Matrix

Definition B.4 *The pseudo-inverse A^+ of $A = U\Sigma V^\top$ is defined as*

$$A^+ = V\Sigma^+ U^\top \tag{B.11}$$

where the elements of Σ^+ are the inverses of the non-zero singular values, i.e. $1/\sigma_i$, if $\sigma_i \neq 0$, and 0 otherwise[1]. □

The pseudo-inverse has a number of interesting properties:

Lemma B.5 *Let A^+ be the pseudo-inverse of A. Then*

$$\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \\ (AA^+)^\top &= AA^+ \\ (A^+A)^\top &= A^+A \end{aligned}$$

[1] Of course Σ^+ is the pseudo-inverse of Σ.

Proof: Assume $\text{rank}(A) = r$ and let $I_{r,n}$ be a diagonal matrix with the first r diagonal elements equal to 1, and the rest zero.

$$\begin{aligned} AA^+A &= U\Sigma V^\top V\Sigma^+ U^\top U\Sigma V^\top \\ &= U\Sigma\Sigma^+\Sigma V^\top \\ &= U\Sigma I_{r,n} V^\top \\ &= U\Sigma V^\top \\ &= A \end{aligned}$$

$$\begin{aligned} A^+AA^+ &= V\Sigma^+ U^\top U\Sigma V^\top V\Sigma^+ U^\top \\ &= V\Sigma^+\Sigma\Sigma^+ U^\top \\ &= V\Sigma^+ I_{r,n} U^\top \\ &= V\Sigma^+ U^\top \\ &= A^+ \end{aligned}$$

$$\begin{aligned} (AA^+)^\top &= (U\Sigma V^\top V\Sigma^+ U^\top)^\top \\ &= U{\Sigma^+}^\top V^\top V\Sigma^\top U^\top \\ &= U\Sigma^+ V^\top V\Sigma U^\top \\ &= U\Sigma^+\Sigma U^\top \\ &= U\Sigma\Sigma^+ U^\top \\ &= U\Sigma V^\top V\Sigma^+ U^\top \\ &= AA^+ \end{aligned}$$

$$\begin{aligned} (A^+A)^\top &= (V\Sigma^+ U^\top U\Sigma V^\top)^\top \\ &= V\Sigma^\top U^\top U{\Sigma^+}^\top V^\top \\ &= V\Sigma U^\top U\Sigma^+ V^\top \\ &= V\Sigma\Sigma^+ V^\top \\ &= V\Sigma^+\Sigma V^\top \\ &= V\Sigma^+ U^\top U\Sigma V^\top \\ &= A^+A \end{aligned}$$

□

The 4 conditions of (B.5) uniquely define the pseudo-inverse, and are in fact used to define the pseudo-inverse in [Pen55].

Lemma B.6

$$A^\top A A^+ = A^\top$$

Proof: $(A^\top A A^+)^\top = A^{+\top} A^\top A = (AA^+)^\top A = AA^+A = A$ □

Theorem B.7 *The unique solution to the optimal linear least squares problem $Ax = b$ is*

$$\hat{x} = A^+ b \tag{B.12}$$

□

We first need:

Lemma B.8 *For $A \in \mathbf{R}^{m\times n}$, $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$:*

$$\begin{aligned} ||Ax + (I - AA^+)y||_2^2 &= ||Ax||_2^2 + ||(I - AA^+)y||_2^2 \\ ||A^+y + (I - A^+A)x||_2^2 &= ||A^+y||_2^2 + ||(I - A^+A)x||_2^2 \end{aligned}$$

Proof: First we observe that

$$\begin{aligned} &||Ax + (I - AA^+)y||_2^2 = \\ &(Ax + (I - AA^+)y)^\top (Ax + (I - AA^+)y) = \\ &((Ax)^\top + y^\top (I - AA^+)^\top)(Ax + (I - AA^+)y) = \\ &(Ax)^\top Ax + (Ax)^\top (I - AA^+)y + y^\top (I - AA^+)^\top Ax + ((I - AA^+)y)^\top (I - AA^+)y = \\ &(Ax)^\top Ax + ((I - AA^+)y)^\top (I - AA^+)y = \\ &||Ax||_2^2 + ||(I - AA^+)y||_2^2 \end{aligned}$$

where the last but one equality comes from two calculations: $(Ax)^\top (I - AA^+)y = x^\top A^\top y - x^\top A^\top AA^+ y = x^\top A^\top y - x^\top A^\top y = 0$ and $y^\top (I - AA^+)^\top Ax = y^\top Ax - y^\top AA^+Ax = y^\top Ax - y^\top Ax = 0$ using lemmas B.5 and B.6.

The second equation in the lemma comes from a similar calculation, and both are just special cases of the Pythagorean theorem since $Ax \perp (I - AA^+)y$ and $A^+x \perp (I - A^+A)y$. □

Proof of theorem B.7:

$$\begin{aligned} ||Ax-b||_2^2 &= ||Ax-b+AA^+b-AA^+b||_2^2 \\ &= ||A(x-A^+b)+(I-AA^+)(-b)||_2^2 \\ &= ||A(x-A^+b)||_2^2+||(I-AA^+)(-b)||_2^2 \\ &= ||Ax-AA^+b||_2^2+||AA^+b-b||_2^2 \end{aligned}$$

So $||Ax-b||_2^2$ is minimal when $||Ax-AA^+b||_2^2=0$, i.e. when $Ax=AA^+b$. For such an x we get

$$\begin{aligned} ||x||_2^2 &= ||x+A^+b-A^+b||_2^2 \\ &= ||x+A^+b-A^+AA^+b||_2^2 \\ &= ||x+A^+b-A^+Ax||_2^2 \\ &= ||A^+b+(I-A^+A)x||_2^2 \\ &= ||A^+b||_2^2+||(I-A^+A)x||_2^2 \\ &= ||A^+b||_2^2+||x-A^+Ax||_2^2 \\ &= ||A^+b||_2^2+||x-A^+AA^+b||_2^2 \\ &= ||A^+b||_2^2+||x-A^+b||_2^2 \end{aligned}$$

which means that $||x||_2^2$ is minimal when $||x-A^+b||_2^2=0$, i.e. when $x=A^+b$. □

Appendix C

Reals in TurboPascal 5.0

The possible real-types available in TurboPascal 5.0 are

Name	Range	Digits	Binary digits	Storage	80x87	1 mult
single	$1.5e^{-45} \ldots 3.4e^{38}$	7–8	24	4 bytes	X	0.25 msec
real	$2.9e^{-39} \ldots 1.7e^{38}$	11–12	40	6 bytes		0.34 msec
double	$5.0e^{-324} \ldots 1.7e^{308}$	15–16	53	8 bytes	X	0.26 msec
extended	$3.4e^{-4932} \ldots 1.1e^{4932}$	19–20	64	10 bytes	X	0.28 msec

Table C.1: Reals in TurboPascal 5.0.

An 'X' in the 80x87-column means that this representation is only available when compiling for 80x87-mode, i.e. using the compiler directive `{$N+}`. If an 80x87 numeric co-processor is not available on the machine, the compiler directive `{$E+}` will include a 80x87 software emulator. This will add appr. 10 Kbytes extra to the executable code. The code will then use the 80x87 if present, and otherwise emulate it in software.

The 3 80x87-modes correspond to IEEE floating point standards.

The last column shows how long it takes to perform 1 multiplication on a 12.5 Mhz 80286 without numeric co-processor.

It is interesting to note that the standard TurboPascal representation (`real`) is slower than all the others. This is either because this representation is awkward for processing, or because better code exists for the IEEE standards.

It is also interesting to note that there is almost no penalty for using high IEEE precision. This is because internally the 80x87 (and thus the emulator) always works in the highest precision. The extra time required is probably just because more bytes need to be moved.

The execution times would be much faster with a numeric co-processor.

Appendix D

Programs

These are the Pascal-programs mentioned earlier in the report. They can compile with Turbo Pascal 5.0, provided they are embedded in a proper program. The various counters, test-prints etc. used to produce the tables in this report are omitted.

D.1 TESTPROB

```
(* The testproblem uses:
                float: the floatingpoint format chosen.
                   mu: the penalty parameter.
   The testproblem defines:
                    n: the dimension.
                    f: the (penalty)function to minimize.
                 Grad: calculate gradient.
              Hessian: calculate Hessian.
                Point: a point in R^n.
               Matrix: a matrix in R^(n*n).
             TestInit: initialize variables.
*)

CONST
  rows = 8;
  cols = 5;
  n  = cols;

TYPE
```

```
20   Point = ARRAY [1..cols] OF float;
21   Matrix = ARRAY [1..n,1..n] OF float;
22
23 CONST
24   A: ARRAY [1..rows,1..cols] OF float =
25         ( ( 22, 10,  2,  3,  7),
26           ( 14,  7, 10,  0,  8),
27           ( -1, 13, -1,-11,  3),
28           ( -3, -2, 13, -2,  4),
29           (  9,  8,  1, -2,  4),
30           (  9,  1, -7,  5, -1),
31           (  2, -6,  6,  5,  1),
32           (  4,  5,  0, -2,  2));
33   b: ARRAY [1..rows] OF float =
34         (646,454,94,-38,302,202,18,146);
35
36 VAR (* See section 3.3.1 page 17. *)
37   AtA: ARRAY [1..cols,1..cols] OF float;
38   Atb: ARRAY [1..cols] OF float;
39
40   Q_prime: ARRAY [1..cols,1..cols] OF float;
41   v_prime: ARRAY [1..cols] OF float;
42   s_prime: float;
43
44   PROCEDURE TestInit;
45   VAR
46     i,j,k: WORD;
47     sum: float;
48   BEGIN
49     (* Calculate A^T A *)
50     FOR i := 1 TO cols DO
51       FOR j := 1 TO cols DO
52       BEGIN
53         sum := 0.0;
54         FOR k := 1 TO rows DO
55           sum := sum + A[k,j]*A[k,i];
56         AtA[j,i] := sum
57       END;
58
59     (* Calculate A^T b *)
60     FOR i := 1 TO cols DO
61     BEGIN
62       sum := 0.0;
63       FOR k := 1 TO rows DO
64         sum := sum + A[k,i]*b[k];
65       Atb[i] := sum
66     END;
```

```
67
68      (* Q' = (A^T A)^2 *)
69      FOR i := 1 TO cols DO
70        FOR j := 1 TO cols DO
71        BEGIN
72          sum := 0.0;
73          FOR k := 1 TO cols DO
74            sum := sum + AtA[j,k]*AtA[k,i];
75          Q_prime[j,i] := sum
76        END;
77
78      (* v' = -2(A^T b)^T A^T A *)
79      FOR i := 1 TO cols DO
80      BEGIN
81        sum := 0.0;
82        FOR j := 1 TO cols DO
83          sum := sum + Atb[j]*AtA[j,i];
84        v_prime[i] := -2*sum
85      END;
86
87      (* s' = (A^T b)^T A^T b *)
88      sum := 0.0;
89      FOR i := 1 TO cols DO
90        sum := sum + Atb[i]*Atb[i];
91      s_prime := sum
92
93    END; {TestInit}
94
95    (* f(x) = x^T x + μ x^T Q' x + μ v'^T x + μ s' *)
96    FUNCTION f(VAR x: Point): float;
97    VAR
98      i,j: WORD;
99      s,s1,s2,s3: float;
100     v: Point;
101   BEGIN
102     (* s_1 = x^T x *)
103     s1 := 0.0;
104     FOR i := 1 TO cols DO
105       s1 := s1 + x[i]*x[i];
106
107     (* v = x^T Q' *)
108     FOR i := 1 TO cols DO
109     BEGIN
110       s := 0.0;
111       FOR j := 1 TO cols DO
112         s := s + x[j]*Q_prime[j,i];
113       v[i] := s
```

```
114     END;
115
116     (* s2 = v^T x *)
117     s2 := 0.0;
118     FOR i := 1 TO cols DO
119       s2 := s2 + v[i]*x[i];
120
121     (* s3 = v'^T x *)
122     s3 := 0.0;
123     FOR i := 1 TO cols DO
124       s3 := s3 + v_prime[i]*x[i];
125
126     (* f = s1 + mu(s2 + s3 + s') *)
127     f := s1 + mu*(s2+s3+s_prime)
128
129   END; {f}
130
131   (* ∇f(x) = 2(I + μQ')x + μv' *)
132   PROCEDURE Grad(VAR x: Point; VAR b: Point);
133   VAR
134     i,k: WORD;
135     sum: float;
136   BEGIN
137     FOR i := 1 TO n DO
138     BEGIN
139       sum := 0.0;
140       FOR k := 1 TO n DO
141         sum := sum + Q_prime[i,k]*x[k];
142       sum := x[i]+mu*sum;
143       b[i] := 2*sum + mu*v_prime[i]
144     END
145   END; {Grad}
146
147   (* H_f(x) = 2(I + μQ') *)
148   PROCEDURE Hessian(VAR x: Point; VAR H: Matrix);
149   VAR
150     i,j: WORD;
151     c: float;
152   BEGIN
153     FOR i := 1 TO n DO
154       FOR j := 1 TO n DO
155       BEGIN
156         c := mu*Q_prime[i,j];
157         IF i=j THEN
158           c := c+1;
159         c := 2*c;
160         H[i,j] := c
```

```
161       END
162   END; {Hessian}
```

D.2 SIMPLEX

```
  1 PROGRAM Simplex;
  2 {$N+}  (* Enter 80x87 mode *)
  3 {$E+}  (* Use emulator *)
  4 {$R+}  (* Range checks *)
  5 {$S+}  (* Stack checks *)
  6
  7 (* This program implements the simplex method described in
  8    section 4, page 18.
  9       Author: Erik Jacobsen
 10       Date: October 1992
 11 *)
 12
 13 CONST
 14   Debug1 = TRUE;     (* Print operations *)
 15   Debug2 = TRUE;     (* Print minimum along the way *)
 16   Debug3 = TRUE;     (* Print simplices on a file *)
 17
 18 {$DEFINE BINDUMP}   (* bindump or textdump *)
 19
 20 TYPE
 21   float = EXTENDED;  (* Choices are:  SINGLE, REAL, DOUBLE,
EXTENDED *)
 22
 23 (*$I TESTPROB.INC *)
 24 (* $I ROSENBROCK.INC *)
 25 (* $I SIMPLE.INC *)
 26
 27 CONST
 28   alpha   = 1.0;     (* \alpha > 0 *)
 29   beta    = 0.5;     (* 0 < \beta < 1 *)
 30   gamma   = 1.5;     (* \gamma > 1 *)
 31   delta   = 0.5;     (* 0 < \delta < 1 *)
 32   epsilon = 1.0E-4;  (* 0 < \epsilon small *)
 33   sigma   = 1.0;     (* Size of initial simplex *)
 34
 35 VAR
 36   x: ARRAY [1..n+1] OF point;  (* The simplex — big enough *)
 37   initial_x: point;              (* Initial point *)
 38   x_0: point;                    (* Centroid *)
 39   x_r,x_e,x_c: point;            (* Reflexion, expansion, con-
```

```
traction points *)
  f_: ARRAY [1..n+1] OF float; (* Function values in simplex
points *)
  f_r,f_e,f_c: float;            (* ... and in reflextion,
expansion, *)
                                 (*  contraction points *)
  low,high: WORD;                (* Position of x_l and x_h *)

(* x and f are sorted according to the functionvalues in f
   f[1] is lowest and f[n+1] is highest *)


  NoOfIterations: WORD;
  {$IFDEF TEXTDUMP}
  g: TEXT;
  {$ENDIF}
  {$IFDEF BINDUMP}
  g: FILE;
  Dummy,DummyWord: WORD;
  {$ENDIF}

  PROCEDURE MakeSimplex(p: point);
  VAR
    i: WORD;
  BEGIN
    FOR i := 1 TO n+1 DO
      x[i] := p;
    FOR i := 2 TO n+1 DO
      x[i][i-1] := x[i][i-1] + sigma;
    low := 1;
    high := n+1
  END; {MakeSimplex}

  PROCEDURE Init;
  VAR
    i: WORD;
  BEGIN
(*    FOR i := 1 TO n DO
      initial_x[i] := 0.0; *)
    MakeSimplex(initial_x);
    FOR i := 1 TO n+1 DO
      f_[i] := f(x[i])
  END; {Init}

  PROCEDURE PrintSimplex;
  VAR
    i,j: WORD;
```

```
84  BEGIN
85    FOR i := 1 TO n+1 DO
86      Write('   x[',i,']    ');
87    Writeln;
88    FOR i := 1 TO n+1 DO
89      Write('-----------');
90    Writeln;
91    FOR j := 1 TO n DO
92    BEGIN
93      FOR i := 1 TO n+1 DO
94        Write(x[i][j]:10:6,' ');
95      Writeln
96    END;
97    FOR i := 1 TO n+1 DO
98      Write('-----------');
99    Writeln;
100   FOR i := 1 TO n+1 DO
101     Write(f_[i]:10:6,' ');
102   Writeln
103 END; {PrintSimplex}
104
105 PROCEDURE PrintSimplexOnFile;
106 VAR
107   i,j: WORD;
108 BEGIN
109   IF Debug3 THEN
110   BEGIN
111     {$IFDEF TEXTDUMP}
112       Writeln(g,n,' ',n+1);
113     {$ENDIF}
114     {$IFDEF BINDUMP}
115       DummyWord := n;
116       BlockWrite(g,DummyWord,SIZEOF(WORD),Dummy);
117       DummyWord := n+1;
118       BlockWrite(g,DummyWord,SIZEOF(WORD),Dummy);
119     {$ENDIF}
120     FOR j := 1 TO n DO
121     BEGIN
122       FOR i := 1 TO n+1 DO
123       BEGIN
124       {$IFDEF TEXTDUMP}
125         Write(g,x[i][j],' ');
126       {$ENDIF}
127       {$IFDEF BINDUMP}
128         BlockWrite(g,x[i][j],SIZEOF(float),Dummy);
129       {$ENDIF}
130       END;
```

```
131         {$IFDEF TEXTDUMP}
132           Writeln(g)
133         {$ENDIF}
134       END
135     END
136   END; {PrintSimplex}
137
138   PROCEDURE SortFunctionValues;
139   VAR
140     i,j: WORD;
141     Changed: BOOLEAN;
142     tmpf: float;
143     tmpx: point;
144   BEGIN (* Bubblesort *)
145     Changed := TRUE;
146     j := n+1;
147     REPEAT
148       DEC(j);
149       Changed := FALSE;
150       FOR i := 1 TO j DO
151         IF f_[i] > f_[i+1] THEN
152         BEGIN
153           tmpf := f_[i]; f_[i] := f_[i+1]; f_[i+1] := tmpf;
154           tmpx :=  x[i];  x[i] :=  x[i+1];  x[i+1] := tmpx;
155           Changed := TRUE
156         END
157     UNTIL NOT Changed
158   END; {SortFunctionValues}
159
160   PROCEDURE MoveUp;  (* Discard top element, and move rest up *)
161   VAR
162     i: WORD;
163   BEGIN
164     FOR i:=n+1 DOWNTO 2 DO
165     BEGIN
166       f_[i] := f_[i-1];
167        x[i] :=  x[i-1]
168     END
169   END; {MoveUp}
170
171   PROCEDURE MoveDown; (* Bring topelement in correct position *)
172   VAR
173     i: WORD;
174     tmpf: float;
175     tmpx: point;
176   BEGIN
177     FOR i := n+1 DOWNTO 2 DO
```

```
178     BEGIN
179       IF f_[i] < f_[i-1] THEN
180       BEGIN
181         tmpf := f_[i]; f_[i] := f_[i-1]; f_[i-1] := tmpf;
182         tmpx :=  x[i];  x[i] :=  x[i-1];  x[i-1] := tmpx
183       END
184       ELSE
185         EXIT
186     END
187   END; {MoveDown}
188
189   PROCEDURE FindCentroid;
190   VAR
191     i,j: WORD;
192   BEGIN
193     (* Find the centroid x_0 = 1/n * sum_{i=1,i!=h}^{|s|} x_i *)
194
195     FOR j := 1 TO n DO
196       x_0[j] := 0.0;
197
198     FOR i := 1 TO n DO
199       FOR j := 1 TO n DO
200         x_0[j] := x_0[j]+x[i][j];
201     FOR j := 1 TO n DO
202       x_0[j] := x_0[j] / n
203   END; {FindCentroid}
204
205   (* We stop when sqrt(sum_{i=1}^{|s|}(f_i - f_bar)^2) < epsilon *)
206
207   FUNCTION Stop: BOOLEAN;
208   VAR
209     i: WORD;
210     sum,avg: float;
211   BEGIN
212     avg := 0.0;
213     FOR i := 1 TO n+1 DO
214       avg := avg + f_[i];
215     avg := avg / (n+1);
216     sum := 0.0;
217     FOR i := 1 TO n+1 DO
218       sum := sum + sqr(f_[i] - avg);
219     Stop := sqrt(sum) < epsilon
220   END; {Stop}
221
222   PROCEDURE PerformSimplex;
223   VAR
224     i,j: WORD;
```

```
225  BEGIN
226    SortFunctionValues;
227    NoOfIterations := 0;
228    REPEAT
229      INC(NoOfIterations);
230      IF Debug1 THEN
231        Writeln('Iteration: ',NoOfIterations);
232
233      FindCentroid;
234      IF Debug2 THEN
235        WritePoint(x[low]);
236
237      (* x_r = (1+alpha)x_0 - alpha x_h and f_r = f(x_r) *)
238      FOR j := 1 TO n DO
239        x_r[j] := (1+alpha)*x_0[j] - alpha*x[high][j];
240      f_r := f(x_r);
241
242      IF f_r < f_[low] THEN
243      BEGIN
244
245        (* x_e = gamma x_r + (1-gamma)x_0 and f_e = f(x_e) *)
246        FOR j := 1 TO n DO
247          x_e[j] := gamma*x_r[j] + (1-gamma)*x_0[j];
248        f_e := f(x_e);
249
250        MoveUp;
251
252        IF f_e < f_r THEN
253        BEGIN
254          x[low]  := x_e;
255          f_[low] := f_e;
256          IF Debug1 THEN
257            Writeln('Expansion')
258        END
259        ELSE
260        BEGIN
261          x[low]  := x_r;
262          f_[low] := f_r;
263          IF Debug1 THEN
264            Writeln('Reflexion')
265        END
266      END
267      ELSE
268      BEGIN
269
270        IF f_r <= f_[high-1] THEN
271        BEGIN
```

```
272           x[high]  := x_r;
273           f_[high] := f_r;
274           MoveDown
275         END
276         ELSE (* f_r > f_h' *)
277         BEGIN
278
279           IF f_r < f_[high] THEN
280           BEGIN
281             x[high]  := x_r;
282             f_[high] := f_r;
283           END;
284
285           (* x_c = beta*x_h + (1-beta)*x_0 and f_c = f(x_c) *)
286           FOR i := 1 TO n DO
287             x_c[i] := beta*x[high][i] + (1-beta)*x_0[i];
288           f_c := f(x_c);
289
290           IF f_c < f_[high] THEN
291           BEGIN
292             x[high]  := x_c;
293             f_[high] := f_c;
294             MoveDown;
295             IF Debug1 THEN
296               Writeln('Contraction')
297           END
298           ELSE
299           BEGIN
300
301             (* x_i = delta*x_i + (1-delta)*x_l and f_i = f(x_i) *)
302             FOR i := 2 TO n+1 DO
303             BEGIN
304               FOR j := 1 TO n DO
305                 x[i][j] := delta*x[i][j] + (1-delta)*x[low][j];
306               f_[i] := f(x[i])
307             END;
308             SortFunctionValues;
309             IF Debug1 THEN
310               Writeln('Shrinkage');
311
312           END
313         END
314       END;
315 {       IF NoOfIterations MOD 10 = 0 THEN}
316         PrintSimplexOnFile;
317     UNTIL Stop;
318     PrintSimplexOnFile;
```

```
319   END; {PerformSimplex}
320
321 VAR
322   r: point;
323   i: WORD;
324 BEGIN
325     FOR i := 1 TO n DO
326       initial_x[i] := 0.0;
327
328 REPEAT
329   IF Debug3 THEN
330   BEGIN
331     {$IFDEF TEXTDUMP}
332       Assign(g,'SIMPLEX.DAT');
333       Rewrite(g);
334     {$ENDIF}
335     {$IFDEF BINDUMP}
336       Assign(g,'SIMPLEX.DAT');
337       Rewrite(g,1);
338     {$ENDIF}
339   END;
340   TestInit;
341   Init;
342   PerformSimplex;
343   PrintSimplex;
344   WritePoint(x[low]);
345   Writeln('Value at minimum: ',f_[low]:8:4);
346   Writeln('Norm of \hat{x}: ',Norm(x[low]):8:4);
347   Residue(x[low],r);
348   Writeln('Norm of residue: ',Norm(r):8:4);
349   Writeln('Number of evaluations: ',NoOfEvaluations);
350   Writeln('Value at th.min : ',f(Theoretical_minimum):8:4);
351   IF Debug3 THEN
352   BEGIN
353     {$IFDEF TEXTDUMP}
354       Writeln(g,'0 0');
355     {$ENDIF}
356     {$IFDEF BINDUMP}
357       DummyWord := 0;
358       BlockWrite(g,DummyWord,SIZEOF(WORD),Dummy);
359       BlockWrite(g,DummyWord,SIZEOF(WORD),Dummy);
360     {$ENDIF}
361     Close(g)
362   END;
363
364   Initial_x := x[low];
365   UNTIL FALSE
```

```
END. {Simplex}
```

D.3 PATTERN

```
PROGRAM Pattern;
{$N+}  (* Enter 80x87 mode *)
{$E+}  (* Use emulator *)
{$R-}  (* Range checks *)
{$S-}  (* Stack checks *)

(* This program implements the pattern method described in
   section 5, page 33.
      Author: Erik Jacobsen
      Date: October 1992
*)

CONST
  Debug1 = FALSE;          (* Print number of evaluations for
each λ. *)
  Debug2 = TRUE;          (* Print x along the way. *)

TYPE
  float = EXTENDED;       (* Choices are: SINGLE, REAL, DOUBLE,
EXTENDED *)

(*$I TESTPROB.INC *)
(* $I ROSENBROCK.INC *)
(* $I SIMPLE.INC *)


CONST
  TryAll = TRUE;          (* Try all 2n points in an ET? *)
  epsilon  = 1.0E-04;

VAR
  x,y,t: point;
  InitialPoint: point;
  v,f_x: float;
  lambda: float;
  Evals: LONGINT;

  PROCEDURE Init;
  VAR
```

```
38    i: WORD;
39  BEGIN
40    lambda := 1;
41    Evals := NoOfEvaluations;
42  END; {Init}
43
44  PROCEDURE ExploratoryTest;
45  LABEL
46    done;
47  VAR
48    i: WORD;
49    t,yy: point;
50    m: float;
51  BEGIN
52    yy := y;
53    FOR i:=1 TO n DO  (* Try x ± λe_i *)
54    BEGIN
55      t := y;
56      t[i] := y[i] + lambda;
57      m := f(t);
58      IF m < v THEN
59      BEGIN
60        v := m;
61        yy := t;
62        IF NOT TryAll THEN
63          GOTO done
64      END;
65      t[i] := y[i] - lambda;
66      m := f(t);
67      IF m < v THEN
68      BEGIN
69        v := m;
70        yy := t;
71        IF NOT TryAll THEN
72          GOTO done
73      END
74    END;
75  done:
76    y := yy
77  END; {ExploratoryTest}
78
79  PROCEDURE PerformPattern;
80  VAR
81    Stop: BOOLEAN;
82    i: WORD;
83  BEGIN
84
```

```
 85     x := InitialPoint;
 86     f_x := f(x);
 87
 88     IF Debug2 THEN WritePoint(x);
 89
 90     Stop := FALSE;
 91     REPEAT
 92       v := f_x;
 93       y := x;
 94
 95       ExploratoryTest;
 96
 97       IF v < f_x THEN
 98       BEGIN
 99         REPEAT
100           t := x;
101           x := y;
102           f_x := v;
103
104           IF Debug2 THEN WritePoint(x);
105
106           FOR i := 1 TO n DO
107             y[i] := 2*y[i]-t[i];
108           v := f(y);
109
110           ExploratoryTest
111         UNTIL v >= f_x
112       END
113       ELSE
114       BEGIN
115         IF Debug1 THEN
116           Writeln('lambda=',lambda:10:8,' used ',
117                   NoOfEvaluations-Evals,' evaluations');
118         Evals := NoOfEvaluations;
119         lambda := lambda / 2;
120         IF lambda < epsilon THEN
121           Stop := TRUE
122       END
123     UNTIL Stop
124   END; {PerformPattern}
125
126 VAR
127   i: WORD;
128   r: point;
129 BEGIN
130   FOR i := 1 TO n DO
131     InitialPoint[i] := 0.0;
```

```
132   TestInit;
133   Init;
134   PerformPattern;
135   WritePoint(x);
136   Writeln('Value at minimum: ',f_x:8:4);
137   Writeln('Number of evaluations: ',NoOfEvaluations);
138   Residue(x,r);
139   Writeln('Norm of residue: ',Norm(r):8:4);
140   InitialPoint := x
141
142 END. {Pattern}
143
```

D.4 SOLVE

```
 1 (* Solve a system of linear equations by Gauss-elimination. *)
 2 (* The procedure does not do a complete LU-decomposition. *)
 3
 4 PROCEDURE Solve(VAR A: matrix; VAR b: point; VAR x: point);
 5 VAR
 6   R: ARRAY [1..n] OF WORD;  (* Permutation of rows *)
 7   i,j,k,t: WORD;
 8   c: float;
 9 BEGIN
10   FOR i := 1 TO n DO
11     R[i] := i;
12
13   (* Put A on triangular form *)
14   FOR k := 1 TO n-1 DO
15   BEGIN
16     (* Partial pivoting *)
17     j := k;
18     FOR i := k+1 TO n DO
19       IF ABS(A[R[i],k]) > ABS(A[R[j],k]) THEN
20         j := i;
21     t := R[k]; R[k] := R[j]; R[j] := t;  (* Swap rows *)
22
23     IF A[R[k],k] = 0.0 THEN
24       Writeln('Singular matrix in Gaussean Elimination');
25
26     (* Eliminate k'th column below diagonal *)
27     FOR i := k+1 TO n DO
28     BEGIN
29       c := A[R[i],k] / A[R[k],k];
30       A[R[i],k] := 0.0;
31       FOR j := k+1 TO n DO
```

```
32          A[R[i],j] := A[R[i],j] - c*A[R[k],j];
33        b[R[i]] := b[R[i]] - c*b[R[k]]
34      END
35    END;
36
37    IF A[R[n],n] = 0.0 THEN
38      Writeln('Singular matrix in Gaussean Elimination');
39
40    (* Back substitution *)
41    FOR k := n DOWNTO 1 DO
42    BEGIN
43      c := b[R[k]];
44      FOR i := k+1 TO n DO
45        c := c - x[i]*A[R[k],i];
46      x[k] := c / A[R[k],k]
47    END
48  END; {Solve}
```

D.5 NEWTON

```
 1  PROGRAM Newton;
 2  {$N+}  {Enter 80x87 mode}
 3  {$E+}  {Use emulator}
 4  {$R+}  {Range checks}
 5  {$S+}  {Stack checks}
 6
 7  (* This program implements the Newton-Raphson method described in
 8     section 6 page 42.
 9       Author: Erik Jacobsen
10       Date: October 1992
11  *)
12
13
14  TYPE
15    float = EXTENDED;       {Choices are: SINGLE, REAL, DOUBLE, EXTENDED}
16
17  (*$I TESTPROB.INC *)
18  (* $I ROSENBROCK.INC *)
19  (* $I SIMPLE.INC *)
20
21  CONST
22    eps  = 1.0E-08;
23
24  VAR
25    x: point;
26    Iterations: LONGINT;
```

```
27
28   PROCEDURE Init;
29   VAR
30     i: WORD;
31   BEGIN
32     Iterations := 0;
33     FOR i := 1 TO n DO
34       x[i] := 0.0
35   END; {Init}
36
37 (*$I SOLVE.INC*)
38
39 {  FUNCTION Norm(VAR p: point): float;
40   VAR
41     i: WORD;
42     s: float;
43   BEGIN
44     s := 0.0;
45     FOR i := 1 TO n DO
46       s := s + sqr(p[i]);
47     Norm := SQRT(s)
48   END; {Norm}
49
50   PROCEDURE PerformNewton;
51   VAR
52     b,r: point;
53     H: matrix;
54     i: WORD;
55   BEGIN
56     REPEAT
57       WritePoint(x);
58       INC(Iterations);
59       Grad(x,b);
60       FOR i := 1 TO n DO
61         b[i] := -b[i];
62       Hessian(x,H);
63       Solve(H,b,r);
64       FOR i := 1 TO n DO
65         x[i] := x[i] + r[i];
66     UNTIL Norm(r) < eps;
67   END; {PerformNewton}
68
69 BEGIN
70   TestInit;
71   Init;
72   PerformNewton;
73   WritePoint(x);
```

```
74   Writeln('Value at minimum: ',f(x):8:4);
75   Writeln('Number of iterations: ',Iterations);
76 END. {Newton}
77
78
```

D.6 HES-POW

```
 1 PROGRAM Hes_Pow;
 2 {$N+}  {Enter 80x87 mode}
 3 {$E+}  {Use emulator}
 4 {$R+}  {Range checks}
 5 {$S+}  {Stack checks}
 6
 7 (* This program implements the Hestenes-Powell method, as
 8    described in ...somewhere...
 9 *)
10
11 TYPE
12   float = EXTENDED;
13
14 CONST
15   rows = 8;
16   cols = 5;
17   n  = cols;
18   epsilon = 10e-4;
19
20 TYPE
21   point = ARRAY [1..cols] OF float;
22   matrix = ARRAY [1..n,1..n] OF float;
23
24 CONST
25   A: ARRAY [1..rows,1..cols] OF float =
26         ( ( 22, 10,  2,  3,  7),
27           ( 14,  7, 10,  0,  8),
28           ( -1, 13, -1,-11,  3),
29           ( -3, -2, 13, -2,  4),
30           (  9,  8,  1, -2,  4),
31           (  9,  1, -7,  5, -1),
32           (  2, -6,  6,  5,  1),
33           (  4,  5,  0, -2,  2));
34   b: ARRAY [1..rows] OF float =
35         (646,454,94,-38,302,202,18,146);
36   Theoretical_minimum: point = (22,10,2,3,7);
37
38 VAR
```

```
39  AtA: matrix;
40  Atb: point;
41  Q_prime: matrix;  (* Q' = (A^T A)^2 *)
42  P_prime: point;   (* P' = b^T A A^T A *)
43  mu: float;
44  NoOfEvaluations: LONGINT;
45
46  PROCEDURE TestInit;
47  VAR
48    i,j,k: WORD;
49    sum: float;
50  BEGIN
51    NoOfEvaluations := 0;
52
53    (* Calculate A^T A *)
54    FOR i := 1 TO cols DO
55      FOR j := 1 TO cols DO
56      BEGIN
57        sum := 0.0;
58        FOR k := 1 TO rows DO
59          sum := sum + A[k,j]*A[k,i];
60        AtA[j,i] := sum
61      END;
62
63    (* Calculate A^T b *)
64    FOR i := 1 TO cols DO
65    BEGIN
66      sum := 0.0;
67      FOR k := 1 TO rows DO
68        sum := sum + A[k,i]*b[k];
69      Atb[i] := sum
70    END;
71
72    (* Q' = (A^T A)^2 *)
73    FOR i := 1 TO cols DO
74      FOR j := 1 TO cols DO
75      BEGIN
76        sum := 0.0;
77        FOR k := 1 TO cols DO
78          sum := sum + AtA[j,k]*AtA[k,i];
79        Q_prime[j,i] := sum
80      END;
81
82     (* P' = b^T A A^T A *)
83     FOR i := 1 TO n DO
84     BEGIN
85       sum := 0.0;
```

```
86        FOR j := 1 TO n DO
87          sum := sum + Atb[j]*AtA[i,j];
88        P_Prime[i] := sum
89      END
90
91    END; {TestInit}
92
93    (* Define f: R^n -> R *)
94
95    FUNCTION f(x: point; lambda: point): float;
96    VAR
97      i,j: WORD;
98      s,s1,s2,s3: float;
99      v: point;
100   BEGIN
101     INC(NoOfEvaluations);
102
103     (* s1 = x^T x *)
104     s1 := 0.0;
105     FOR i := 1 TO n DO
106       s1 := s1 + x[i]*x[i];
107
108     (* v = A^T Ax - Atb *)
109     FOR i := 1 TO n DO
110     BEGIN
111       s := 0.0;
112       FOR j := 1 TO n DO
113         s := s + AtA[i,j]*x[j];
114       v[i] := s - Atb[i]
115     END;
116
117     (* s2 = lambda^T v *)
118     s2 := 0.0;
119     FOR i := 1 TO n DO
120       s2 := s2 + lambda[i]*v[i];
121
122     (* s3 = v^T v *)
123     s3 := 0.0;
124     FOR i := 1 TO n DO
125       s3 := s3 + v[i]*v[i];
126
127     (* f = s1 + s2 + frac12mus3 *)
128     f := s1 + s2 + (mu*s3)/2
129
130   END; {f}
131
132   (* nabla f(x, lambda) = 2x + lambda^T (A^T A) *)
```

```
133  (*              +μ((A^⊤A)^2 x − b^⊤AA^⊤A) *)
134  PROCEDURE Grad(x: point; lambda: point; VAR b: point);
135  VAR
136    i,j: WORD;
137    sum: float;
138    v1,v2: point;
139  BEGIN
140    (* v_1 = λ^⊤(A^⊤A) *)
141    FOR i := 1 TO n DO
142    BEGIN
143      sum := 0.0;
144      FOR j := 1 TO n DO
145        sum := sum + lambda[j]*AtA[i,j];
146      v1[i] := sum
147    END;
148
149    (* v_2 = (A^⊤A)x *)
150    FOR i := 1 TO n DO
151    BEGIN
152      sum := 0.0;
153      FOR j := 1 TO n DO
154        sum := sum + Q_prime[i,j]*x[j];
155      v2[i] := sum
156    END;
157
158    FOR i := 1 TO n DO
159      b[i] := 2*x[i] + v1[i] + mu*(v2[i] + P_prime[i])
160
161  END; {Grad}
162
163  (* Calculate H_f = 2I + μQ' *)
164  PROCEDURE Hessian(VAR x: point; VAR H: matrix);
165  VAR
166    i,j: WORD;
167    c: float;
168  BEGIN
169    FOR i := 1 TO n DO
170      FOR j := 1 TO n DO
171      BEGIN
172        c := mu*Q_prime[i,j];
173        IF i=j THEN
174          c := c+2;
175        H[i,j] := c
176      END
177  END; {Hessian}
178
179  PROCEDURE WritePoint(x: point);
```

```
180   VAR
181     i: WORD;
182   BEGIN
183     Write('(');
184     FOR i := 1 TO n DO
185     BEGIN
186       Write(x[i]:18:14);
187       IF i<>n THEN
188         Write(',')
189     END;
190     Writeln(')')
191   END; {WritePoint}
192
193 VAR
194   lambda: point;
195   Iterations: LONGINT;
196
197   PROCEDURE Init;
198   VAR
199     i: WORD;
200   BEGIN
201     FOR i := 1 TO n DO
202       lambda[i] := 0.0;
203     Iterations := 0
204   END; {Init}
205
206   FUNCTION Norm(VAR p: point): float;
207   VAR
208     i: WORD;
209     s: float;
210   BEGIN
211     s := 0.0;
212     FOR i := 1 TO n DO
213       s := s + sqr(p[i]);
214     Norm := SQRT(s)
215   END; {Norm}
216
217   PROCEDURE Minimize(lambda: point; VAR x: point);
218   CONST
219     eps = 1.0E-08;
220   VAR
221     Iterations: WORD;
222
223     PROCEDURE Init;
224     VAR
225       i: WORD;
226     BEGIN
```

```
227       Iterations := 0;
228       FOR i := 1 TO n DO
229         x[i] := 0.0
230     END; {Init}
231
232 (*$I SOLVE.INC*)
233
234     PROCEDURE PerformNewton;
235     VAR
236       b,r: point;
237       H: matrix;
238       i: WORD;
239     BEGIN
240       REPEAT
241         INC(Iterations);
242         Grad(x,lambda,b);
243         FOR i := 1 TO n DO
244           b[i] := -b[i];
245         Hessian(x,H);
246         Solve(H,b,r);
247         FOR i := 1 TO n DO
248           x[i] := x[i] + r[i]
249       UNTIL Norm(r) < eps
250     END; {PerformNewton}
251
252   BEGIN
253     Init;
254     PerformNewton
255   END; {Minimize}
256
257 VAR
258   x,v,oldx,r: point;
259   s: float;
260   i,j: WORD;
261
262 BEGIN
263   Write('Mu='); Readln(mu);
264   TestInit;
265   Init; FOR i:=1 TO n DO x[i] := 0.0;
266   REPEAT
267     oldx := x;
268     INC(Iterations);
269     Minimize(lambda,x);
270     Writeln(Iterations,';');
271 {    Write('Lambda = '); WritePoint(lambda);
272     Write('x      = '); WritePoint(x);
273     Writeln('f(x,lambda)=',f(x,lambda):10:4);
```

```
274 }
275     (* v = A^T Ax - Atb *)
276     FOR i := 1 TO n DO
277     BEGIN
278       s := 0.0;
279       FOR j := 1 TO n DO
280         s := s + AtA[i,j]*x[j];
281       v[i] := s - Atb[i]
282     END;
283
284     (* λ = λ + μ(A^T Ax - Atb) *)
285     FOR i := 1 TO n DO
286       lambda[i] := lambda[i] + mu*v[i];
287 {    Writeln('f(x,lambda)=',f(x,lambda):10:4);}
288
289 {    Write('v      = '); WritePoint(v);}
290     FOR i := 1 TO n DO
291       r[i] := oldx[i] - x[i];
292
293   UNTIL Norm(r) < epsilon;
294   Write('Lambda = '); WritePoint(lambda);
295   Write('x      = '); WritePoint(x);
296   Writeln('f(x,lambda)=',f(x,lambda):10:4);
297   Writeln('Iterations: ',Iterations)
298 END. {Hes_Pow}
```

Bibliography

[BDS69] M. J. Box, D. Davies, and W. H. Swann. *Non-Linear Optimization Techniques*. Monograph No. 5 by Imperial Chemical Industries Limited. Oliver & Boyd, 1969.

[BM69] John W. Bandler and Patrick A. MacDonald. Optimization of microwave networks by razor search. *IEEE Transactions on Microwave Theory and Techniques*, MTT-17(8):552–562, August 1969.

[Box66] M. J. Box. A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, 8(1):42–52, 1965–66.

[EO66] F. E. Emery and M. O'Hagan. Optimal design of matching networks for microwave transistor amplifiers. *IEEE Transactions on Microwave Theory and Techniques*, MTT-14:696–698, December 1966.

[FB74] Fr. Fabricius-Bjerre. *Lærebog i Geometri I.* Gjellerup, 1974.

[FB90] John B. Fraleigh and Raymond A. Beauregard. *Linear Algebra.* Addison-Wesley, 2nd edition, 1990.

[Fle81] R. Fletcher. *Practical Methods of Optimization, Constrained Optimization*, volume 2. Wiley, 1981.

[GR71] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. In J. H. Wilkinson and C. Reinsch, editors, *Handbook for Automatic Computation: Linear Algebra*, volume II, pages 134–151. Springer Verlag, 1971.

[GvL83] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. North Oxford Academic, 1983.

[Hes69] Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

[Hes75] Magnus R. Hesteness. *Optimization Theory. The Finite Dimensional Case*. Wiley-Interscience, 1975.

[HJ61] Robert Hooke and T. A. Jeeves. "Direct search" solution of numerical and statistical problems. *JACM*, 8:212–229, 1961.

[KO68] J. Kowalik and M. R. Osborne. *Methods for Unconstrained Optimization Problems*. American Elsevier, New York, 1968.

[Kui62] Nicolaas H. Kuiper. *Linear Algebra and Geometry*. North-Holland Publishing Company, 1962.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965 1964–65.

[NRP] NRPAS13.ZIP. Available by anonymous ftp from e.g. garbo.uwasa.fi. The package contains Pascal procedures originally published as the Pascal Appendix to the FORTRAN book "NUMERICAL RECIPES: THE ART OF SCIENTIFIC COMPUTING" (Cambridge University Press, 1986), and test driver programs originally published as the "NUMERICAL RECIPES EXAMPLE BOOK (PASCAL)" (Cambridge University Press, 1986). Later versions are not freely available.

[Pen55] R. Penrose. A generalized inverse for matrices. *Proc. Cambridge Philos Soc.*, 51:406–413, 1955.

[PH72] J. M. Parkinson and D. Hutchinson. An investigation into the efficiency of variants on the simplex method. In Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 115–135. Academic Press, 1972.

[PSU88] A. L. Peressini, F. E. Sullivan, and J. J. Uhl, Jr. *The Mathematics of Nonlinear Programming*. Undergraduate Texts in Mathematics. Springer-Verlag, 1988.

[Ram76] Håkan Ramsin. *Ickelinjär optimering*. LiberLäromedel, 1976.

[RR78] Anthony Ralston and Philip Rabinowitz. *A First Course in Numerical Analysis*. McGraw-Hill, 2nd edition, 1978.

[SHH62] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of the simplex design in optimization and evolutionary operation. *Technometrics*, 4:441–461, 1962.

[Ste73] G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, 1973.

[Sub89] M. B. Subrahmanyam. An extension of the simplex method to constrained nonlinear optimization. *Journal of Optimization Theory and Applications*, 62(2):311–319, August 1989.

[WCT72] Richard E. Williamson, Richard H. Crowell, and Hale F. Trotter. *Calculus of Vector Functions*. Prentice Hall, 3rd edition, 1972.