Nonlinear Optimization

Benny Yakir

May 25, 2010

Chapter 1

Background

1.1 Introduction

The general optimization problem has the form:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$$

subject to:

$$\begin{aligned} g_i(\mathbf{x}) &= 0 \qquad i = 1, \ldots, m_e \\ g_i(\mathbf{x}) &\leq 0 \qquad i = m_e + 1, \ldots, m \\ \mathbf{x}_l \leq \mathbf{x} &\leq \mathbf{x}_u \end{aligned}$$

In particular, if $m = 0$, the problem is called an unconstrained optimization problem. In this course we intend to introduce and investigate algorithms for solving this problem. Algorithms will be programmed and implemented using the R computational environment, which will be introduced in class.

We intend to cover the following topics:

Basic properties of solutions and algorithms: In this section we consider the abstract notion of an iterative algorithm and and discuss the convergence of descending algorithms to a solution.

Basic R: Here we introduce the basic features and structure of the R system.

Line search methods: Here we deal with algorithms for finding the minimum in the case where $d = 1$. These algorithms are the basic building blocks for solving more complex optimization problems.

Properties of solutions in unconstarint minimization: In this section we discuss sufficient and necesary conditions that solutions to an optimization problem must obey.

The method of steepest descent: In each iteration, a line search is performed in the direction of the steepest descent.

Newton and Quasi-Newton methods: In the Newton method the function is approximated (locally) by a quadratic form. The direction of the search is chosen based on this form. Quasi-Newton methods replace the Hessian with terms which are easier to evaluate.

Properties of solutions in constraint minimization: The conditions that were considered for unconstrained problems are modified in order to deal with constraints. The Lagrange multipliers and the Kuhn-Tucker conditions will be discussed.

Lagrange methods: These methods are based on the Lagrange first-order conditions of a solution. The method is applied for quadratic programming.

Sequential Quadratic Programming: At each iteration the function and Lagrange multipliers are approximated by a quadratic programming problem.

Optimization in statistical settings: In the closing part of the course we will discuss special algorithms for optimization that are commonly used in statistical problems.

1.2 Global Convergence of Descending Algorithms

The algorithms we will typically consider are iterative descending algorithms. By iterative we mean, roughly, that the algorithm generates a series of points, each point being calculated on the basis of the points preceding it. By descending we mean that the sequence of values of some function, calculated at the points generated by the algorithm, is a monotone decreasing sequence. More precisely, we define:

Algorithm: An algorithm A is a mapping that assigns, to each point, a subset of the space.

Solution set: An identified subset Γ of the space.

Iterative algorithm: The specific sequence is constructed by choosing a point in the subset and iterating the process. Thus algorithm generates a series of points, and each point is calculated on the basis of the points preceding it.

Descending algorithm: There exists a continuous function Z such that if A is the algorithm and Γ is the solution set then

1. If $\mathbf{x} \notin \Gamma$ and $\mathbf{y} \in A(\mathbf{x})$, then $Z(\mathbf{y}) < Z(\mathbf{x})$.
2. If $\mathbf{x} \in \Gamma$ and $\mathbf{y} \in A(\mathbf{x})$, then $Z(\mathbf{y}) \leq Z(\mathbf{x})$.

Globally convergent algorithm: An algorithm is globally convergent if, for any starting point, it generates a sequence that converges to a point in the solution set.

A closed map: A point-to-set map A is said to be closed at $\mathbf{x}$ if $1+2 \Rightarrow 3$, where

1. $\mathbf{x}_k \to \mathbf{x}$.
2. $\mathbf{y}_k \in A(\mathbf{x}_k)$ and $\mathbf{y}_k \to \mathbf{y}$.
3. $\mathbf{y} \in A(\mathbf{x})$.

The map A is closed if it is closed at each point of the space.

Example: Suppose for $x \in \mathbb{R}$ we define $A(x) = [-|x|/2, |x|/2]$. Starting at $x_0 = 100$, each of the sequences

$$100, 50, 25, 12, -6, -2, 1, 1/2, \ldots$$
$$100, -40, 20, -5, -2, 1, 1/4, 1/8, \ldots$$
$$100, 10, 1/16, 1/100, -1/1000, 1/10000, \ldots$$

might be generated from iterative application of the algorithm. The given algorithm is closed.

Example: If A is point-to-point and continuous them A is closed.

Theorem 1.2.1. *If A is a descending iterative algorithm which is closed outside of the solution set Γ and if the sequence of points is contained in a compact set then any converging subsequence converges to a solution.*

1.3 Rate of convergence

We say that an algorithm converges at rate p at least to a solution x^* if

$$\limsup_n \frac{\|x_{n+1} - x^*\|}{\|x_n - x^*\|^p} < \infty,$$

where $\|\cdot\|$ is an appropriate norm. Note that the rate of convergence when $p = 1$ is actually exponential.

1.4 Homework

1. Define the point-to-set mapping on $\mathbb{R}^n$ by

   $$A(\mathbf{x}) = \{\mathbf{y} : \mathbf{y}'\mathbf{x} \leq b\},$$

   where b is a fixed constant. Is A closed?

2. Consider the iterative process

   $$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right),$$

   where $a > 0$. Assuming the process converges, to what does it converge? What is the order of convergence.

Chapter 2

Basic R

Many sources for learning R are available on the web. A good source can be found at: `http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf` Below is a short introduction to some of the basic features of R. We recommend that you consult a more comprehensive text.

2.1 Starting and quitting R

After installing R under the `Windows` operating system an icon will be added to the desktop. Double clicking on that icon will open the window of the `R` system, which contains the `R Console` sub-window. We found it convenient to have a separate working directory for each project. It is convenient to copy the `R` icon into that directory and to set the working directory by coping its path (in double quotes) in the appropriate box (`"start in:"`) in the `Shortcuts` slip of the `Properties` of the icon (which can be selected by right-clicking the icon.)

The `R` language is an interactive expression-oriented programming langauge. The elementary commands may consist of expressions, which are immediately evaluated, printed to the standard output and lost. Alternatively, expressions can be assigned to object, which store the evaluation of the expression. In the later case the result is not printed out to the screen. These objects are accessible for the duration of the session, and are lost at the end of the session, unless they are actively stored. At the end of the session the user is prompted to store the entire workspace image, including all objects that were created during the session. If "`Yes`" is selected then the objects used in the current session will be available in the next. If "`No`" is selected then only objects from the last saved image will remain.

Help can be found via the "`Help`" menu at the upper bar. Convenient possibilities are "`Search help...`" and "`R functions (text)...`". The former opens a window for searching the help files for an appropriate string. The latter can be used in order to produce the help page when the name of the function help is sought for is known. The searching for help can be conducted with a web browser using the "`Help html`" option.

2.2 Matrices

Consider the following example that involves a magic matrix:

```
> A <- matrix(c(16,3,2,13,5,10,11,8,9,6,7,12,4,15,14,1),
+       4,4,byrow=TRUE)
> A
     [,1] [,2] [,3] [,4]
[1,]   16    3    2   13
[2,]    5   10   11    8
[3,]    9    6    7   12
[4,]    4   15   14    1
> sum(A)
[1] 136
> colSums(A)
[1] 34 34 34 34
> rowSums(A)
[1] 34 34 34 34
> sum(diag(A))
[1] 34
> A[1,4]+A[2,3]+A[3,2]+A[4,1]
[1] 34
> sum(A[outer(1:4,1:4,"+")==5])
[1] 34
```

We learn from this example that:

- Matrices can be created with the function "`matrix`".
- Memory is allocated automatically.
- The function "`c`" concatenates objects into a vector.
- Arguments of a function can be assigned either by location or by name.

- The default method of creating matrices from vectors is column by column.
- The assignment operator is "`<-` ".
- A matrix is also a vector.
- The function "`diag` extracts or replace the diagonal of a matrix.
- Items within a matrix can be referred to via the indexing system.

Consider next:

```
> B <- rbind(matrix(ceiling(10*runif(10)),2),
+         matrix(rnorm(10),2))
> B
           [,1]       [,2]      [,3]       [,4]        [,5]
[1,]  5.000000  5.0000000  3.000000  2.0000000  7.00000000
[2,]  1.000000  9.0000000  6.000000 10.0000000  7.00000000
[3,] -1.049742 -0.2892459 -1.725665  1.1280271 -0.60114631
[4,] -0.813596 -1.9501419 -1.089035  0.1293922  0.09231302
>
```

Which tells us that:

- One can generate random numbers.
- Matrices may be combined together to form a new matrix.

Next,

```
> A*A
     [,1] [,2] [,3] [,4]
[1,]  256    9    4  169
[2,]   25  100  121   64
[3,]   81   36   49  144
[4,]   16  225  196    1
> A%*%A
     [,1] [,2] [,3] [,4]
[1,]  341  285  261  269
[2,]  261  301  309  285
[3,]  285  309  301  261
[4,]  269  261  285  341
> t(A)%*%A
```

```
     [,1] [,2] [,3] [,4]
[1,]  378  212  206  360
[2,]  212  370  368  206
[3,]  206  368  370  212
[4,]  360  206  212  378
```

What is the difference between the three cases?

Finally, consider the algebraic properties of the matrix:

```
> det(A)
[1] 4.86413e-13
> eigen(A)
$values
[1]  3.400000e+01  8.000000e+00 -8.000000e+00 -1.249098e-15

$vectors
     [,1]          [,2]          [,3]       [,4]
[1,] -0.5 -8.164966e-01 -4.082483e-01  0.2236068
[2,] -0.5  4.082483e-01  2.782422e-16 -0.6708204
[3,] -0.5  3.133596e-16 -4.082483e-01  0.6708204
[4,] -0.5  4.082483e-01  8.164966e-01 -0.2236068
```

2.3 Functions

Apart from the many functions that are available as part of the basic distribution of R or are part of contributed packages one can easily write functions for conducting specific tasks. Functions are stored as objects. The basic format of a function is: "`fun.name <- function(args) expression`". The function returns the evaluation of the expression. Alternatively, the output of the function may be specified with the function "`return`". Multiple expressions can be combined with curly brackets. Default values for the arguments may be set using the format "`arg=value`".

```
> falling <- function(t,gravity=32) gravity*t^2/2
> falling(0:5)
[1]   0  16  64 144 256 400
> falling(0:5,8)
[1]   0   4  16  36  64 100
```

Consider another example.

```
> g <- function(x,a) 0.5*(x + a/x)
> x <- 1
> for(i in 1:10^2) x <- g(x,4)
> x
[1] 2
```

Observe that the function computed the square root of 4, which is 2. More generaly, the iteration of the function will converge to the square root of the second argument `a`. Let us wrap the algorithm for the computation of the square root inside another function and add a stopping criteria:

```
> my.sqrt <- function(a,tol=0/10^6){
+   x0 <- x <- 1
+   d <- 10^6
+   while(d > tol){
+      x <- g(x0,a)
+      d <- abs(x-x0)
+      x0 <- x
+   }
+   return(x)
+ }
> my.sqrt(2)
[1] 1.414214
> sqrt(2)
[1] 1.414214
```

2.4 Graphics

R has many graphical abilities. Broadly speaking, plotting function may either be *high level* or *low level*. The former type produces a plot:

```
> t <- seq(0,2*pi,length=100)
> y <- sin(t)
> plot(t,y,type="l")
```

and the latter type adds to an existing plot:

```
> y2 <- sin(t-0.25)
> lines(t,y2,col=2)
> y3 <- sin(t-0.5)
> lines(t,y3,col=3)
```

Three dimensional plotting is also possible:

```
> y <- x <- seq(0,1,by=0.01)
> z <- outer(x,y,function(x,y) exp(- (x^2 + y^2)))
> contour(x,y,z)
```

Observe that:

- The function "`seq`" produces regular sequences.
- The "`type`" refers to the type of plotting (points, lines, etc.).
- Functions can be applied to vectors.
- Any binary function can be used in the function "`outer`".

2.5 Homework

1. Let $f(x) = ax^2 - 2bx + c$. Under which conditions does f has a minimum? What is the minimizing x?
2. Let $f(\mathbf{x}) = \mathbf{x}'\mathbf{A}\mathbf{x} - 2\mathbf{b}'\mathbf{x} + c$, with $\mathbf{A}$ an $n \times n$ matrix, $\mathbf{b}$ an n-vectors and c a scaler. Under which conditions does f has a minimum? a unique minimum? What is the minimizing $\mathbf{x}$?
3. Write an R function that finds the location and value of the minimum of a quadratic function.
4. Plot a contour plot of the function f with $\mathbf{A} = \begin{pmatrix} 1 & 3 \\ -1 & 2 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$ and $c = 5$. Mark, on the plot, the location of the minimum.

Chapter 3

Line search methods

We prepare for the consideration of algorithms for locating a local minimum in the optimization problem with no constrains. All methods have in common the basic structure: in each iteration a direction $\mathbf{d}_n$ is chosen from the current location $\mathbf{x}_n$. The next location, $\mathbf{x}_{n+1}$, is the minimum of the function along the line that passes through $\mathbf{x}_n$ in the direction $\mathbf{d}_n$. Before discussing the different approaches for choosing directions, we will deal with the problem of finding the minimum of a function of one variable, a problem terms *line search*.

3.1 Fibonacci and Golden Section Search

These approaches assume only that the function is unimodal. Hence, if the interval is divided by the points $x_0 < x_1 < \cdots < x_N < x_{N+1}$ and we find that, among these points, x_k minimizes the function then the over-all minimum is in the interval $[x_{k-1}, x_{k+1})$.

The Fibonacci sequence ($F_n = F_{n-1} + F_{n-2}$, $F_0 = F_1 = 1$) is the basis for choosing sequentially N points such that the discrepancy $x_{k+1} - x_{k-1}$ is minimized. The length of the final interval is $(x_{N+1} - x_0)/F_N$.

The solution of the Fibonacci recursion is $F_N = A\tau_1^N + B\tau_2^N$, where

$$\tau_1 = \frac{1+\sqrt{5}}{2} = 1/0.618, \quad \tau_2 = \frac{1-\sqrt{5}}{2}.$$

It follows that $F_{N-1}/F_N \sim 0.618$, a number called the *golden ratio*, and the rate of convergence of this line search approach is exponential with an exponential rate that equals the log of this number. Note that the proposed Fibonacci algorithm is linear.

The golden section is an approximation of the optimal procedure in which a point is added between two previous points according to the golden ratio. The asymptotic rate of the golden section is the same as that of the approach based on the Fibonacci sequence. It is the standard algorithm in the initial stage of a line search. At the vicinity of the solution it is replaced by an algorithm with a higher rate of convergence.

3.2 Newton's method

The best known method of line search is Newton's method. Assume not only that the function is continuous but that it is also smooth. Given the first and second derivatives of the function at x_n, one can write the Taylor expansion:

$$f(x) \approx q(x) = f(x_n) + f'(x_n)(x - x_n) + f''(x_n)(x - x_n)^2/2.$$

The minimum of $q(x)$ is obtained at

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

(Note that this approach can be associated with the problem of finding the zeros of the function $g(x) = q'(x)$.)

We can expect that the solution of an iterative procedure of this type will satisfy

$$x^* = x^* - \frac{f'(x^*)}{f''(x^*)} \Rightarrow f'(x^*) = 0.$$

We claim that the rate of convergence of the Newton algorithm is quadratic:

Theorem 3.2.1. *Let the function g have a continuous second derivative and let x^* be such that $g(x^*) = 0$ and $g'(x^*) \neq 0$. Then the Newton method converges with an order of convergence of at least two, provided that x_0 is sufficiently close to x^*.*

Proof. Denote $G(x) = x - f'(x)/f''(x)$ and Let x^* be a solution of $G(x) = x$. A second order approximation of the drivative f' around x_n produces:

$$f'(x_n) - f'(x^*) = -[f'(x^*) - f'(x_n)] = -\left[f''(x_n)(x^* - x_n) + \frac{f^{(3)}(\tilde{x})}{2}(x^* - x_n)^2\right],$$

for some $\tilde{x}$ between x_n and x^*. Therefore,

$$\frac{f'(x_n) - f'(x^*)}{f''(x_n)} = (x_n - x^*) - \frac{f^{(3)}(\tilde{x})}{2f''(x_n)}(x^* - x_n)^2 .$$

It follows, from the assumed continuity of the derivatives that

$$x_{n+1} - x^* = x_n - x^* - \frac{f'(x_n) - f'(x^*)}{f''(x_n)} \approx \frac{f^{(3)}(x^*)}{2f''(x^*)}(x_n - x^*)^2.$$

□

3.3 Applying line-search methods

Let us exemplify the problem of finding the minimum of a function on a line with R.

```
> humps <- function(x) 1/((x-0.3)^2+0.01)+1/((x-0.9)^2+0.04)-6
> x <- seq(-5,5,by=0.01)
> plot(x,humps(x),type="l")
```

The function doing line search is `optimize`:

```
> optimize(humps,c(0.3,1))
$minimum
[1] 0.6370261

$objective
[1] 11.25275
> abline(v=c(0.3,1),lty=2)
```

In the next exercise we try to trace down the steps of the algorithm:

```
> humps.plot <- function(x)
+ {
+     y <- 1/((x-0.3)^2 + 0.01) + 1/((x-0.9)^2+0.04) - 6
+     lines(x,y,col=2,type="h")
+     print(c(x,y))
+     return(y)
+ }
> i <- 1
> x <- seq(0.3,1,by=0.01)
> plot(x,humps(x),type="l")
> abline(v=c(0.3,1),lty=2)
> optimize(humps.plot,c(0.3,1))
[1]  0.5673762 12.9098442
```

```
[1]  0.7326238 13.7746201
[1]  0.4652476 25.1714186
[1]  0.6444162 11.2692834
[1]  0.6412996 11.2583212
[1]  0.6376181 11.2528668
[1]  0.6369854 11.2527543
[1]  0.6370261 11.2527542
[1]  0.6370668 11.2527551
[1]  0.6370261 11.2527542
$minimum
[1] 0.6370261

$objective
[1] 11.25275
```

Can you say at which stage it switched from the golden section to a different algorithm?

3.4 Quadratic interpolation

Assume we are given $x_1 < x_2 < x_3$ and the values of $f(x_i)$, $i = 1, 2, 3$, which satisfy

$$f(x_2) < f(x_1) \quad \text{and} \quad f(x_2) < f(x_3).$$

The quadratic interpolation passing through these points is given by

$$q(x) = \sum_{i=1}^{3} f(x_i) \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}.$$

The minimum of this function is obtained at the point

$$x_4 = \frac{1}{2} \frac{\beta_{23} f(x_1) + \beta_{31} f(x_2) + \beta_{12} f(x_3)}{\gamma_{23} f(x_1) + \gamma_{31} f(x_2) + \gamma_{12} f(x_3)},$$

with $\beta_{ij} = x_i^2 - x_j^2$ and $\gamma_{ij} = x_i - x_j$. An algorithm $A : \mathbb{R}^3 \to \mathbb{R}^3$ can be defined by such a pattern. If we start from an initial 3-points pattern $\mathbf{x} = (x_1, x_2, x_3)$ the algorithm A can be constructed in such a way that $A(\mathbf{x})$ has the same pattern. The algorithm is continuous, hence closed. It is descending with respect to the function $Z(\mathbf{x}) = f(x_1) + f(x_2) + f(x_3)$. If follows that the algorithm converges to the solution set $\Gamma = \{\mathbf{x}^* : f'(x_i^*) = 0, i = 1, 2, 3.\}$. It can be shown that the order of convergence to the solution is (approximately) 1.3. Unlike the Newton method, the algorithm does not require knowledge of the derivatives of the function.

3.5 Cubic fit

Given x_1 and x_2, together with $f(x_1)$, $f'(x_1)$, $f(x_2)$ and $f'(x_2)$, one can consider a cubic interpolation of the form

$$q(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

The local minimum is determined by the solution of the equation

$$q'(x) = a_1 + 2a_2 x + 3a_3 x^2 = 0,$$

which satisfies

$$q''(x) = 2a_2 x + 6a_3 x > 0.$$

It follows that the appropriate interpolation is given by

$$x_3 = x_2 - (x_2 - x_1)\frac{f'(x_2) + \beta_2 - \beta_1}{f'(x_2) - f'(x_1) + 2\beta_2},$$

where

$$\begin{aligned} \beta_1 &= f'(x_1) + f'(x_2) - 3\frac{f(x_1) - f(x_2)}{x_1 - x_2} \\ \beta_2 &= (\beta_1^2 - f'(x_1)f'(x_2))^{1/2}. \end{aligned}$$

The order of convergence of this algorithm is 2 and it uses only first order derivative.

3.6 Homework

1. Find the minimum of the function `-humps`. Use different ranges.
2. (a) Given $f(x_n), f'(x_n)$ and $f'(x_{n-1})$, show that

$$q(x) = f(x) + f'(x_n)(x - x_n) + \frac{f'(x_{n-1}) - f'(x_n)}{x_{n-1} - x_n} \cdot \frac{(x - x_n)^2}{2},$$

   has the same derivatives as f at x_n and x_{n-1} and is equal to f at x_n.

   (b) Construct a line search algorithm based on this quadratic fit.
3. What conditions on the values and derivatives at two points guarantee that a cubic fit will have a minimum between the two points? Use the answer to develop a search scheme that is globally convergent for unimodal functions.

4. Consider the function

$$f(x, y) = e^x(4x^2 + 2y^2 + 4xy + 2y + 1).$$

Use the function `optimize` to plot the function

$$g(y) = \min_x f(x, y).$$

Chapter 4

Conditions for Unconstraint Solutions

4.1 First Order Necessary Conditions

Assume that the function f is defined over $\Omega \subset \mathbb{R}^d$.

Definition: A point $\mathbf{x}^* \in \Omega$ is said to be a *relative minimum point* or a *local minimum point* of f if there is an $\epsilon > 0$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x}$ such that $\|\mathbf{x} - \mathbf{x}^*\| < \epsilon$. If the inequality is strict for all $\mathbf{x} \neq \mathbf{x}^*$ then $\mathbf{x}^*$ is said to be a *strict relative minimum point.*

Definition: A point $\mathbf{x}^* \in \Omega$ is said to be a *global minimum point* of f if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \Omega$. If the inequality is strict for all $\mathbf{x} \neq \mathbf{x}^*$ then $\mathbf{x}^*$ is said to be a *strict global minimum point.*

In practice, the algorithms we will consider in most of this course converge to a local minimum. We may indicate in some cases how the global minimum can be obtained.

Definition: Given $\mathbf{x} \in \Omega$, a vector $\mathbf{d}$ is a *feasible direction at* $\mathbf{x}$ if there exists an $\bar{\alpha} > 0$ such that $\mathbf{x} + \alpha\mathbf{d} \in \Omega$ for all $0 \leq \alpha \leq \bar{\alpha}$.

Theorem 4.1.1 (First-order necessary conditions.)**.** *Let $f \in C^1$. If $\mathbf{x}^*$ is a relative minimum, then for any vector $\mathbf{d}$ which is feasible at $\mathbf{x}^*$, we have $\dot{f}(\mathbf{x}^*)'\mathbf{d} \geq 0$. ($\dot{f}(\mathbf{x}^*)$ is the* gradient*, i.e. the vector of partial derivatives of f at $\mathbf{x}^*$.)*

Proof. By a one-sided Taylor expansion of the function $h(t) = f(\mathbf{x}^* + t\mathbf{d})$, which is defined in the interval $[0, \bar{\alpha}]$, we obtain that $h(t) = h(0) + th'(0) + o(t)$. Since 0 is a relative minimum of h we obtain that $th'(0) \geq 0$, for all t small enough, which implies that $h'(0) \geq 0$. The claim now follows from the chain rule. □

Corollary 4.1.1. *If $\mathbf{x}^*$ is a relative minimum and if $\mathbf{x}^* \in \Omega^0$ then $\dot{f}(\mathbf{x}^*) = \mathbf{0}$.*

Proof. If $\mathbf{x}^*$ is an interior point then all directions are feasible. It follows that $\dot{f}(\mathbf{x}^*)'\mathbf{d} \geq 0$ and $\dot{f}(\mathbf{x}^*)'(-\mathbf{d}) = -\dot{f}(\mathbf{x}^*)'\mathbf{d} \geq 0$. Hence, $\dot{f}(\mathbf{x}^*)'\mathbf{d} = 0$, for all $\mathbf{d} \in \mathbb{R}^d$. In particular, $\|\dot{f}(\mathbf{x}^*)\|^2 = 0$ and the claim follows. □

Example: Consider the function $f(x, y) = x^2 - xy + y^2 - 3y$, with $\Omega = \mathbb{R}^2$. From the first order conditions we get that $x^* = 1$ and $y^* = 2$. This is a global minimum. (Why?)

Example: Consider the function $f(x, y) = x^2 - x + y + xy$, with $\Omega = (\mathbb{R}^+)^2$. The global minimum is at $x^* = 0.5$ and $y^* = 0$. At this point, $\dot{f}(0.5, 0) = (0, 3/2)'$.

Example: We observe $g(x)$ at the points $x_1, \ldots, x_m$. We want to approximate the function with a polynomial of the form $h(x) = \sum_{j=0}^{d-1} a_j x^j$, for $d < m$. Consider the minimization problem

$$\min_{\mathbf{a}\in\mathbb{R}^d} \sum_{k=1}^{m} [g(x_k) - h(x_k)]^2 = \min_{\mathbf{a}\in\mathbb{R}^d} \sum_{k=1}^{m} [g(x_k) - \textstyle\sum_{j=0}^{d-1} a_j x_k^j]^2 = \min_{\mathbf{a}\in\mathbb{R}^d} f(\mathbf{a}).$$

Let $q_{ij} = \sum_{k=1}^{m} (x_k)^{i+j}$, $b_j = \sum_{k=1}^{m} g(x_k)(x_k)^j$ and $c = \sum_{k=1}^{m} g(x_k)^2$. Then

$$f(\mathbf{a}) = \mathbf{a}'\mathbf{Q}\mathbf{a} - 2\mathbf{b}'\mathbf{a} + c,$$

and the first order conditions take the form $\mathbf{Q}\mathbf{a} = \mathbf{b}$.

4.2 Second Order Necessary Conditions

Second order conditions deal with functions with continuous second partial derivatives and uses the Hessian matrix $\ddot{f}(\mathbf{x}^*)$ of (mixed) partial derivatives.

Theorem 4.2.1 (Second-order necessary conditions.)**.** *Let $f \in C^2$. Let $\mathbf{x}^*$ be a relative minimum. For any vector $\mathbf{d}$ which is feasible at $\mathbf{x}^*$, if $\dot{f}(\mathbf{x}^*)'\mathbf{d} = 0$ then $\mathbf{d}'\ddot{f}(\mathbf{x}^*)\mathbf{d} \geq 0$.*

Proof. Consider a one-sided second-order Taylor expansion of the function $h(t) = f(\mathbf{x}^* + t\mathbf{d})$, which is defined in the interval $[0, \bar{\alpha}]$. One obtains in this case that $h(t) = h(0) + th'(0) + t^2h''(0)/2 + o(t^2)$. By assumption and the chain rule $h'(0) = 0$. Hence, since 0 is a relative minimum of h we obtain that $t^2h''(0)/2 \geq 0$, for all t small enough, which implies that $h''(0) \geq 0$. The claim now follows from an iterative application of the chain rule. □

Corollary 4.2.1. *If* $\mathbf{x}^* \in \Omega^0$ *is a relative minimum and if* $\mathbf{x}^* \in \Omega^0$ *then* $\dot{f}(\mathbf{x}^*)'\mathbf{d} = 0$ *and* $\mathbf{d}'\ddot{f}(\mathbf{x}^*)\mathbf{d} \geq 0$ *for all* $\mathbf{d}$.

Proof. Straightforward. □

Example: Consider the function $f(x, y) = x^3 - x^2y + 2y^2$, with $\Omega = (\mathbb{R}^+)^2$. The first order conditions are

$$3x^2 - 2xy = 0, \quad -x^2 + 4y = 0.$$

There are two solutions: $(0, 0)$ and $(6, 9)$. However, the second is not a relative minimum since the Hessian matrix

$$\ddot{f}(6, 9) = \begin{bmatrix} 18 & -12 \\ -12 & 4 \end{bmatrix}$$

is not positive semi-definite.

Theorem 4.2.2 (Second-order sufficient conditions.)**.** *Let* $f \in C^2$*. Assume that* $\mathbf{x}^* \in \Omega^0$*. If* $\dot{f}(\mathbf{x}^*) = \mathbf{0}$ *and* $\ddot{f}(\mathbf{x}^*)$ *is positive definite then* $\mathbf{x}^*$ *is a strict relative minimum.*

Proof. See Problem 2.4.1. □

4.3 Homework

1. Prove Theorem 4.2.2.

2. To approximate the function g over the interval $[0, 1]$ by a polynomial h of degree n (or less), we use the criterion

$$f(\mathbf{a}) = \int_0^1 [g(x) - h(x)]^2 dx,$$

where $\mathbf{a} \in \mathbb{R}^{n+1}$ are the coefficients of h. Find the equations satisfied by the optimal solution.

3. (a) Using first-order necessary conditions, find the minimum of the function

$$f(x, y, z) = 2x^2 + xy + y^2 + yz + z^2 - 6x - 7y - 8z + 9.$$

   (b) Verify the point is a relative minimum by checking the second-order conditions.

4. In control problem one is interested in finding numbers $u_0, \ldots, u_n$ that minimize the objective function

$$J = \sum_{k=0}^{n} \{(x_0 + u_0 + \cdots + u_{k-1})^2 + u_k^2\},$$

   for a given x_0. Find the equations that determine the first order conditions.

Chapter 5

The Method of Steepest Decent

The method of steepest decent is a method of searching for the minimum of a function of many variables f. In in each iteration of this algorithm a line search is performed in the direction of the steepest decent of the function at the current location. In other words,

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \dot{f}(\mathbf{x}_n),$$

where α_n is the nonnegative scalar that minimizes $f(\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n))$. It can be shown that relative to the solution set $\{\mathbf{x}^* : \dot{f}(\mathbf{x}^*) = \mathbf{0}\}$, the algorithm is descending and closed, thus converging.

5.1 The rate of convergence in the quadratic case

Assume

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}'Q\mathbf{x} - \mathbf{x}'\mathbf{b} = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)'Q(\mathbf{x} - \mathbf{x}^*) - \frac{1}{2}\mathbf{x}^{*\prime}Q\mathbf{x}^*,$$

were Q a positive definite and symmetric matrix and $\mathbf{x}^* = Q^{-1}\mathbf{b}$ is the minimizer of f. Note that in this case $\dot{f}(\mathbf{x}) = Q\mathbf{x} - \mathbf{b}$. and

$$\begin{aligned} f(\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n)) &= \frac{1}{2}(\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n))'Q(\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n)) - (\mathbf{x}_n - \alpha \dot{f}(\mathbf{x}_n))'\mathbf{b} \\ &= \frac{\alpha^2}{2}(\dot{f}(\mathbf{x}_n))'Q(\dot{f}(\mathbf{x}_n)) - \alpha(\dot{f}(\mathbf{x}_n))'(Q\mathbf{x}_n - \mathbf{b}) + \frac{1}{2}\mathbf{x}_n'Q\mathbf{x}_n - \mathbf{x}_n'\mathbf{b} \end{aligned}$$

which is minimized at

$$\alpha_n = \frac{\dot{f}(\mathbf{x}_n)'\dot{f}(\mathbf{x}_n)}{\dot{f}(\mathbf{x}_n)'Q\dot{f}(\mathbf{x}_n)}.$$

It follows, since

$$\mathbf{x}_{n+1} - \mathbf{x}^* = \mathbf{x}_n - \mathbf{x}^* - \alpha_n \dot{f}(\mathbf{x}_n)$$

and

$$Q(\mathbf{x}_n - \mathbf{x}^*) = \dot{f}(\mathbf{x}_n) \quad \Longrightarrow \quad \mathbf{x}_n - \mathbf{x}^* = Q^{-1}\dot{f}(\mathbf{x}_n)$$

that

$$\begin{aligned}
&\frac{1}{2}(\mathbf{x}_{n+1} - \mathbf{x}^*)'Q(\mathbf{x}_{n+1} - \mathbf{x}^*) = \\
&\qquad \frac{1}{2}(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*) - 2\alpha_n f(\mathbf{x}_n)'\dot{f}(\mathbf{x}_n) + \alpha_n^2 \dot{f}(\mathbf{x}_n)'Q\dot{f}(\mathbf{x}_n) \\
&= \quad \left\{1 - \frac{(\dot{f}(\mathbf{x}_n)'\dot{f}(\mathbf{x}_n))^2}{\dot{f}(\mathbf{x}_n)'Q\dot{f}(\mathbf{x}_n)\dot{f}(\mathbf{x}_n)'Q^{-1}\dot{f}(\mathbf{x}_n)}\right\} \times \frac{1}{2}(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*) \ .
\end{aligned}$$

Observe that we have used the fact that

$$(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*) = f(\mathbf{x}_n)'Q^{-1}\dot{f}(\mathbf{x}_n) \ .$$

The corrolary of the following theorem will establish the linear convergence of the Steepest Decent Algorithm:

Theorem 5.1.1 (Kantorovich inequality)**.** *Let Q be a positive definite and symmetric matrix and let $0 < a = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n = A$ be the eigenvalues. Then*

$$\frac{(\mathbf{x}'\mathbf{x})^2}{(\mathbf{x}'Q\mathbf{x})(\mathbf{x}'Q^{-1}\mathbf{x})} \geq \frac{4aA}{(a+A)^2}.$$

Proof: By a change of variables Q is diagonal. We assume it is. In which case

$$\frac{(\mathbf{x}'\mathbf{x})^2}{(\mathbf{x}'Q\mathbf{x})(\mathbf{x}'Q^{-1}\mathbf{x})} = \frac{(\sum_{i=1}^d x_i^2)^2}{(\sum_{i=1}^d \lambda_i x_i^2)(\sum_{i=1}^d x_i^2/\lambda_i)}.$$

Denoting $\xi_i = x_i^2 / \sum_{j=1}^d x_j^2$ the above becomes

$$= \frac{1/\sum_{i=1}^d \xi_i\lambda_i}{\sum_{i=1}^d (\xi_i/\lambda_i)} = \frac{\phi(\xi_1, \ldots, \xi_d)}{\psi(\xi_1, \ldots, \xi_d)}.$$

The numerator is a point on the curve $1/x$. The denominator is a convex combination of points from that curve. The minimal ratio is achieved for some $\lambda = \xi_1\lambda_1 + \xi_d\lambda_d$, where $\xi_1 + \xi_d = 1$. Hence,

$$\frac{\phi(\xi_1, \ldots, \xi_d)}{\psi(\xi_1, \ldots, \xi_d)} \geq \min_{\lambda_1 \leq \lambda \leq \lambda_d} \frac{(1/\lambda)}{(\lambda_1 + \lambda_d - \lambda)/(\lambda_1\lambda_d)}.$$

The minimum is achieved when $\xi_1 = \xi_d = 1/2$, and the proof follows.

Theorem 5.1.2. *For the quadratic case*

$$\frac{1}{2}(\mathbf{x}_{n+1} - \mathbf{x}^*)'Q(\mathbf{x}_{n+1} - \mathbf{x}^*) \leq \left(\frac{A-a}{A+a}\right)^2 \frac{1}{2}(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*).$$

Proof:

$$1 - \frac{4aA}{(a+A)^2} = \left(\frac{A-a}{A+a}\right)^2.$$

5.2 Applying the method in R

Let us investigate the application of the steepest decent method in the quadratic setting. Consider the matrix Q and the vector $\mathbf{b}$:

```
> Q <- matrix(c(0.78,-0.02,-0.12,-0.14,-0.02,0.86,-0.04,0.06,
+     -0.12,-0.04,0.72,-0.08,-0.14,0.06,-0.08,0.74),
+     4,4,byrow=TRUE)
> Q
      [,1]  [,2]  [,3]  [,4]
[1,]  0.78 -0.02 -0.12 -0.14
[2,] -0.02  0.86 -0.04  0.06
[3,] -0.12 -0.04  0.72 -0.08
[4,] -0.14  0.06 -0.08  0.74
> b <- c(0.76,0.08,1.12,0.68)
```

We define the quadratic function `quad` using the matrix and the vector and find the minimum of the function using `R`'s built-in general propose optimizer `optim`:

```
> quad <- function(x,Q,b) t(x)%*%Q%*%x/2 - sum(x*b)
>
> optim(rep(1,4),quad,Q=Q,b=b)
$par
[1] 1.5349646 0.1220379 1.9751225 1.4129248

$value
[1] -2.174660

$counts
function gradient
```

```
     217       NA

$convergence
[1] 0

$message
NULL

> solve(Q)%*%b
          [,1]
[1,] 1.5349650
[2,] 0.1220096
[3,] 1.9751564
[4,] 1.4129555
```

In its default application the function `optim` takes as argument a function to be minimized and an initial point and produces the minimizing value and the value of the target function at that value. Extra arguments to the function in the argument may be passed on using the "`...`" argument. Observe that the function produces a minimizing value which is almost identical to the theoretical value.

Next we want to program the algorithm of steepest decent. For that we need a function that computes the gradient:

```
> quad.grad <- function(x,Q,b) Q%*%x - b
>
> steepest.decent <- function(fun,grad,x,...)
+ {
+     g <- function(a,x,...) fun(x - a*grad(x,...),...)
+     opt <- optimize(g,c(0,10),x=x,...)
+     alpha <- opt$minimum
+     objective <- opt$objective
+     x <- x - alpha*grad(x,...)
+     return(list(x=x,alpha=alpha,objective=objective))
+ }
> steepest.decent(quad,quad.grad,rep(1,4),Q=Q,b=b)
$x
             [,1]
[1,] 1.331422136
[2,] 0.005733593
[3,] 1.815808334
```

```
[4,] 1.127470052

$alpha
[1] 1.274701

$objective
          [,1]
[1,] -2.128281
```

Observe that functions can be treated as any other argument. Observe also that we apply the line-search procedure `optimmize` in order to find the minimizing value in the direction determined by the gradient.

Let us apply several iterations of the algorithm:

```
> xn <- x <- rep(1,4)
> fn <- quad(x,Q,b)
> for (i in 1:10)
+ {
+     out <- steepest.decent(quad,quad.grad,x,Q=Q,b=b)
+     x <- out$x
+     xn <- cbind(xn,x)
+     fn <- c(fn,out$obj)
+ }
> fn
 [1] -1.430000 -2.128281 -2.171504 -2.174440 -2.174644 -2.174658 -2.174659
 [8] -2.174660 -2.174660 -2.174660 -2.174660
> xn
     xn
[1,]  1 1.331422136 1.4813488 1.5235040 1.5302238 1.5342309 1.5345837 1.5349140
[2,]  1 0.005733593 0.1700348 0.1144736 0.1250617 0.1215216 0.1222107 0.1219760
[3,]  1 1.815808334 1.9125511 1.9623510 1.9713649 1.9741493 1.9749043 1.9750791
[4,]  1 1.127470052 1.4000565 1.3926856 1.4122049 1.4114599 1.4129046 1.4128446

[1,] 1.5349357 1.5349614 1.5349628
[2,] 0.1220236 0.1220072 0.1220106
[3,] 1.9751387 1.9751506 1.9751551
[4,] 1.4129518 1.4129472 1.4129552
```

We can see that the algorithm converged to the minimizing value after 8 iterations.

An assessment of the rate of convergence of the algorithm can be obtained via the examination of the extreme eigenvalues:

```
> eigen(Q)
$values
[1] 0.94 0.88 0.76 0.52

$vectors
              [,1]          [,2]          [,3]         [,4]
[1,]  5.773503e-01  5.773503e-01  9.313285e-16 5.773503e-01
[2,] -5.773503e-01  5.773503e-01 -5.773503e-01 7.530868e-16
[3,]  8.592302e-17 -5.773503e-01 -5.773503e-01 5.773503e-01
[4,] -5.773503e-01  9.761208e-16  5.773503e-01 5.773503e-01

> U <- eigen(Q)$vector
> L <- diag(eigen(Q)$val)
> U%*%t(U)
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00 -2.918210e-17 -1.397266e-16 -1.099923e-16
[2,] -2.918210e-17  1.000000e+00 -1.318634e-16 -3.651837e-16
[3,] -1.397266e-16 -1.318634e-16  1.000000e+00 -1.780802e-16
[4,] -1.099923e-16 -3.651837e-16 -1.780802e-16  1.000000e+00
> t(U)%*%U
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00  3.255039e-16 -2.019327e-16  2.976677e-16
[2,]  3.255039e-16  1.000000e+00  7.358552e-17 -5.778077e-18
[3,] -2.019327e-16  7.358552e-17  1.000000e+00  1.847481e-16
[4,]  2.976677e-16 -5.778077e-18  1.847481e-16  1.000000e+00
> L
     [,1] [,2] [,3] [,4]
[1,] 0.94 0.00 0.00 0.00
[2,] 0.00 0.88 0.00 0.00
[3,] 0.00 0.00 0.76 0.00
[4,] 0.00 0.00 0.00 0.52
> U%*%L%*%t(U) - Q
              [,1]          [,2]          [,3]          [,4]
[1,]  1.110223e-16 -2.428613e-17 -1.387779e-17 -5.551115e-17
[2,] -2.775558e-17 -8.881784e-16 -2.914335e-16 -5.342948e-16
[3,] -8.326673e-17 -3.608225e-16 -3.330669e-16 -2.498002e-16
[4,] -1.110223e-16 -5.551115e-16 -2.498002e-16 -3.330669e-16
> ((0.94 - 0.52)/(0.94 + 0.52))^2
[1] 0.08275474
```

5.2.1 Homework

1. Investigate the the performance of the steepest decent algorithm for

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2.$$

   (a) Compute and program the gradient of the function.

   (b) Apply the function `steepest.decent` in order to find the minimum.

   (c) Plot the function and the points produced by the algorithm.

   (d) Why doesn't the steepest decent algorithm converge?

Chapter 6

Newton and Quasi-Newton Methods

6.1 Newton's method

Based on the Taylor expansion

$$f(\mathbf{x}_n) \approx f(\mathbf{x}_n) + \dot{f}(\mathbf{x}_n)'(\mathbf{x} - \mathbf{x}_n) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_n)'\ddot{f}(\mathbf{x}_n)(\mathbf{x} - \mathbf{x}_n)$$

one can derive, in the same spirit that led to the derivation that was presented in the case of the one-dimensional line-search problem, a multi-dimensional Newton's method:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - (\ddot{f}(\mathbf{x}_n))^{-1}\dot{f}(\mathbf{x}_n).$$

Theorem 6.1.1 (Newton's method)**.** *Assume the target function is in C^3, $\mathbf{x}^*$ is a local minimum, and the Hessian $\ddot{f}(\mathbf{x}^*)$ is positive definite. If $\mathbf{x}_0$ is close enough to $\mathbf{x}^*$ then the order of convergence is at least 2.*

Proof.

$$\begin{aligned} \|\mathbf{x}_{n+1} - \mathbf{x}^*\| &= \|\mathbf{x}_n - \mathbf{x}^* - \ddot{f}(\mathbf{x}_n)^{-1}\dot{f}(\mathbf{x}_n)\| \\ &= \|\ddot{f}(\mathbf{x}_n)^{-1}[\dot{f}(\mathbf{x}^*) - \dot{f}(\mathbf{x}_n) - \ddot{f}(\mathbf{x}_n)(\mathbf{x}^* - \mathbf{x}_n)]\| \\ &\leq C\|\mathbf{x}_n - \mathbf{x}^*\|^2, \end{aligned}$$

for some constant C. □

A modification of this approach is to set

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n(\ddot{f}(\mathbf{x}_n))^{-1}\dot{f}(\mathbf{x}_n),$$

where α_n minimizes the function $f(\mathbf{x}_n - \alpha(\ddot{f}(\mathbf{x}_n))^{-1}\dot{f}(\mathbf{x}_n))$.

6.2 Extensions

Consider the approach of choosing

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n S_n \dot{f}(\mathbf{x}_n),$$

where S_n is some symmetric and positive-definite matrix and α_n is the non-negative scalar that minimizes $f(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n))$. It can be shown that when S_n is positive-definite the algorithm is descending.

Consider the quadratic case as an example:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}'Q\mathbf{x} - \mathbf{x}'\mathbf{b} = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)'Q(\mathbf{x} - \mathbf{x}^*) - \frac{1}{2}\mathbf{x}^{*\prime}Q\mathbf{x}^*,$$

were Q a positive definite and symmetric matrix and $\mathbf{x}^* = Q^{-1}\mathbf{b}$ is the minimizer of f. Note that in this case $\dot{f}(\mathbf{x}) = Q\mathbf{x} - \mathbf{b}$. and

$$f(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n)) = \frac{1}{2}(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n))'Q(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n)) - (\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n))'\mathbf{b},$$

which is minimized at

$$\alpha_n = \frac{\dot{f}(\mathbf{x}_n)' S_n \dot{f}(\mathbf{x}_n)}{\dot{f}(\mathbf{x}_n)' S_n Q S_n \dot{f}(\mathbf{x}_n)}.$$

It follows that

$$\frac{1}{2}(\mathbf{x}_{n+1} - \mathbf{x}^*)'Q(\mathbf{x}_{n+1} - \mathbf{x}^*) = \left\{1 - \frac{(\dot{f}(\mathbf{x}_n)' S_n \dot{f}(\mathbf{x}_n))^2}{\dot{f}(\mathbf{x}_n)' S_n Q S_n \dot{f}(\mathbf{x}_n) \dot{f}(\mathbf{x}_n)' Q^{-1} \dot{f}(\mathbf{x}_n)}\right\} \times \frac{1}{2}(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*).$$

Theorem 6.2.1. *For the quadratic case*

$$\frac{1}{2}(\mathbf{x}_{n+1} - \mathbf{x}^*)'Q(\mathbf{x}_{n+1} - \mathbf{x}^*) \leq \left(\frac{B_n - b_n}{B_n + b_n}\right)^2 \frac{1}{2}(\mathbf{x}_n - \mathbf{x}^*)'Q(\mathbf{x}_n - \mathbf{x}^*),$$

where B_n and b_n are the largest and smallest eigenvalues of SQ.

6.3 The Davidon-Fletcher-Powell (DFP) method

This is a rank-two correction procedure. The algorithm is initiated with the starting point $\mathbf{x}_0$ and some initial positive-definite algorithm S_0. In each iteration:

1. Minimizes $f(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n))$ to obtain $\mathbf{x}_{n+1}$ and $\dot{f}(\mathbf{x}_{n+1})$. Define
$$\begin{aligned} \Delta_n \mathbf{x} &= \mathbf{x}_{n+1} - \mathbf{x}_n = -\alpha_n S_n \dot{f}(\mathbf{x}_n) \ , \\ \Delta_n \dot{f} &= \dot{f}(\mathbf{x}_{n+1}) - \dot{f}(\mathbf{x}_n) \ . \end{aligned}$$
2. Set
$$S_{n+1} = S_n + \frac{(\Delta_n \mathbf{x})(\Delta_n \mathbf{x})'}{(\Delta_n \mathbf{x})'(\Delta_n \dot{f})} - \frac{S_n(\Delta_n \dot{f})(\Delta_n \dot{f})' S_n}{(\Delta_n \dot{f})' S_n (\Delta_n \dot{f})}.$$
3. Go to 1.

Theorem 6.3.1. *For the DFP algorithm. If S_n is positive definite then so is S_{n+1}.*

Proof. Let
$$g(\alpha) = f(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n)) \ .$$
By the chain rule we get that the derivative of g with respect to α is:
$$\dot{g}(\alpha) = -\dot{f}(\mathbf{x}_n - \alpha S_n \dot{f}(\mathbf{x}_n))' S_n \dot{f}(\mathbf{x}_n) \ .$$
The minimizing α_n is obtained by solving $\dot{g}(\alpha_n) = 0$. It follows that
$$0 = \dot{g}(\alpha_n) = -\dot{f}(\mathbf{x}_n - \alpha_n S_n \dot{f}(\mathbf{x}_n))' S_n \dot{f}(\mathbf{x}_n) = -\dot{f}(\mathbf{x}_{n+1})' S_n \dot{f}(\mathbf{x}_n) \ .$$
Define $\Delta \mathbf{x} = \mathbf{x}_{n+1} - \mathbf{x}_n$, $\Delta \dot{f} = \dot{f}(\mathbf{x}_{n+1}) - \dot{f}(\mathbf{x}_n)$. Since
$$S_{n+1} = S_n + \frac{(\Delta \mathbf{x})'(\Delta \mathbf{x})}{(\Delta \mathbf{x})'(\Delta \dot{f})} - \frac{S_n(\Delta \dot{f})(\Delta \dot{f})' S_n}{(\Delta \dot{f})' S_n (\Delta \dot{f})},$$
it follows that
$$\begin{aligned} \mathbf{y}' S_{n+1} \mathbf{y} &= \mathbf{y}' S_n \mathbf{y} - \frac{(\mathbf{y}' S_n(\Delta \dot{f}))^2}{(\Delta \dot{f})' S_n (\Delta \dot{f})} + \frac{(\mathbf{y}'(\Delta \mathbf{x}))^2}{(\Delta \mathbf{x})'(\Delta \dot{f})} \\ &= \mathbf{a}'\mathbf{a} - \frac{(\mathbf{a}'\mathbf{b})^2}{\mathbf{b}'\mathbf{b}} + \frac{(\mathbf{y}'(\Delta \mathbf{x}))^2}{(\Delta \mathbf{x})'(\Delta \dot{f})}, \end{aligned} \tag{6.1}$$
where $\mathbf{a} = S_n^{1/2}\mathbf{y}$ and $\mathbf{b} = S_n^{1/2}(\Delta \dot{f})$.

We saw that, since $\mathbf{x}_{n+1}$ is computed by minimizing the function f in the given direction, then $(\dot{f}(\mathbf{x}_n))' S_n (\dot{f}(\mathbf{x}_{n+1})) = 0$. Observe that $\Delta \mathbf{x} = -\alpha_n S_n \dot{f}(\mathbf{x}_n)$. It can be concluded that
$$(\Delta \mathbf{x})'(\Delta \dot{f}) = -\alpha_n (\dot{f}(\mathbf{x}_{n+1}) - \dot{f}(\mathbf{x}_n))' S_n (\dot{f}(\mathbf{x}_n)) = \alpha_n (\dot{f}(\mathbf{x}_n))' S_n (\dot{f}(\mathbf{x}_n)) > 0 \ .$$

The first difference in (6.1) is strictly positive, unless $\mathbf{y} \propto (\Delta \dot{f})$. assume it is. In which case the second term is proportional to $(\dot{f}(\mathbf{x}_n))' S_n (\dot{f}(\mathbf{x}_n))$, thus positive. □

6.4 The Broyden-Flecher-Goldfarb-Shanno (BFGS) method

In this method the Hessian is approximated and not the inverse thereof. This is again a rank-two correction procedure. The algorithm starts with some positive-definite algorithm H_0 and an initial point $\mathbf{x}_0$:

1. Minimizes $f(\mathbf{x}_n - \alpha H_n^{-1}\dot{f}(\mathbf{x}_n))$ to obtain $\mathbf{x}_{n+1}$ and $\dot{f}(\mathbf{x}_{n+1})$. Define

$$\begin{aligned} \Delta_n \mathbf{x} &= \mathbf{x}_{n+1} - \mathbf{x}_n = -\alpha_n H_n^{-1}\dot{f}(\mathbf{x}_n)\ , \\ \Delta_n \dot{f} &= \dot{f}(\mathbf{x}_{n+1}) - \dot{f}(\mathbf{x}_n)\ . \end{aligned}$$

2. Set

$$H_{n+1} = H_n + \frac{\Delta_n \dot{f}' \Delta_n \dot{f}}{\Delta_n \dot{f}' \Delta_n \mathbf{x}} - \frac{H_n \Delta_n \mathbf{x} \Delta_n \mathbf{x}' H_n}{\Delta_n \mathbf{x}' H_n \Delta_n \mathbf{x}}.$$

3. Go to 1.

6.5 Examples

Recall the quadratic function that was used in order to illustrate the algorithm of steepest decent:

```
> Q <- matrix(c(0.78,-0.02,-0.12,-0.14,-0.02,0.86,-0.04,0.06,
+     -0.12,-0.04,0.72,-0.08,-0.14,0.06,-0.08,0.74),
+     4,4,byrow=TRUE)
> b <- c(0.76,0.08,1.12,0.68)
> quad <- function(x,Q,b) t(x)%*%Q%*%x/2 - sum(x*b)
> quad.grad <- function(x,Q,b) Q%*%x - b
>
> solve(Q)%*%b
          [,1]
[1,] 1.5349650
[2,] 0.1220096
[3,] 1.9751564
[4,] 1.4129555
```

Let us program a function that carries out a step of the DFP algorithm:

```
> DFP <- function(fun,grad,x,S,...)
+ {
```

```
+     g <- function(a,x,S,...) fun(x - a*S%*%grad(x,...),...)
+     opt <- optimize(g,c(0,10),x=x,S=S,...)
+     alpha <- opt$minimum
+     objective <- opt$objective
+     x1 <- x - alpha*S%*%grad(x,...)
+     dx <- x1-x
+     ddf <- grad(x1,...) - grad(x,...)
+     term1 <- dx %*% t(dx)/sum(dx*ddf)
+     term2 <- S %*% ddf %*% t(ddf) %*% S
+     term2 <- term2/as.numeric(t(ddf) %*% S %*% ddf)
+     S <- S + term1 - term2
+     x <- x1
+     return(list(x=x,S=S,alpha=alpha,objective=objective))
+ }
```

Test the program:

```
> x0 <- rep(1,4)
> S0 <- diag(4)
> DFP(quad,quad.grad,x0,S0,Q=Q,b=b)
$x
            [,1]
[1,] 1.331422136
[2,] 0.005733593
[3,] 1.815808334
[4,] 1.127470052

$S
            [,1]        [,2]        [,3]         [,4]
[1,]  1.05568042 -0.1107316  0.1120896  0.04112339
[2,] -0.11073155  1.0252274 -0.1364487 -0.15002460
[3,]  0.11208958 -0.1364487  1.1873048  0.11304641
[4,]  0.04112339 -0.1500246  0.1130464  1.00648790

$alpha
[1] 1.274701

$objective
          [,1]
[1,] -2.128281
```

and compare the convergence of the DFP algorithm to the steepest decent:

```
> x <- xn <- x0
> S <- S0
> fn <- quad(x0,Q,b)
> for (i in 1:3)
+ {
+     out <- DFP(quad,quad.grad,x,S,Q=Q,b=b)
+     x <- out$x
+     S <- out$S
+     xn <- cbind(xn,x)
+     fn <- c(fn,out$obj)
+ }
> fn
[1] -1.430000 -2.128281 -2.174356 -2.174658
> xn
     xn
[1,]  1 1.331422136 1.5131097 1.5355306
[2,]  1 0.005733593 0.1152711 0.1210948
[3,]  1 1.815808334 1.9727640 1.9736910
[4,]  1 1.127470052 1.4264535 1.4136988
```

Finally, compare the inverse of the hessian with its approximation after convergence:

```
> S
          [,1]        [,2]        [,3]        [,4]
[1,] 1.3511501  0.06182109  0.32254369  0.25582426
[2,] 0.0618211  1.11110888 -0.03773244 -0.03450682
[3,] 0.3225437 -0.03773244  1.30214508  0.28169782
[4,] 0.2558243 -0.03450682  0.28169782  1.39402136
> solve(Q)
           [,1]        [,2]       [,3]        [,4]
[1,] 1.37442345  0.02417795 0.26223776  0.28641571
[2,] 0.02417795  1.17199430 0.05980861 -0.08398656
[3,] 0.26223776  0.05980861 1.45841001  0.20242915
[4,] 0.28641571 -0.08398656 0.20242915  1.43423206
```

6.6 Homework

1. Investigate the the performance of the DFP algorithm on the function

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

that was investigated in the previous homework.

2. Investigate the rate of convergence of the algorithm

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [\delta I + (\ddot{f}(\mathbf{x}_n))^{-1}]\dot{f}(\mathbf{x}_n).$$

What is the rate if δ is larger than the smallest eigenvalue of $(\ddot{f}(\mathbf{x}^*))^{-1}$?

3. Use the formula

$$[A + \mathbf{b}\mathbf{a}']^{-1} = A^{-1} - \frac{A^{-1}\mathbf{a}\mathbf{b}'A^{-1}}{1 + \mathbf{b}'A^{-1}\mathbf{a}},$$

in order to get a direct updating formula for the inverse of H_n in the BFGS method.

4. Read the help file on the function "`optim`". Investigate the effect of supplying the gradients with the parameter "`gr`" on the performance of the procedure.

Chapter 7

Conditions for Constraint Minimization

The general (constrained) optimization problem has the form:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$$

subject to:

$$\begin{aligned} g_i(\mathbf{x}) &= 0 \qquad i = 1, \ldots, m_e \\ g_i(\mathbf{x}) &\leq 0 \qquad i = m_e + 1, \ldots, m \\ \mathbf{x}_l \leq \mathbf{x} &\leq \mathbf{x}_u \end{aligned}$$

The first m_e constraints are called *equality constraints* and the last $m - m_e$ constraints are the *inequality constraints*.

7.1 Necessary conditions (equality constraints)

We assume first that $m_e = m$, i.e. all constraints are equality constraints. Let $\mathbf{x}^*$ be a solution of the optimization problem. Let $\mathbf{g} = (g_1, \ldots, g_m)$. Note that $\mathbf{g}$ is a (non-linear) transformation from $\mathbb{R}^d$ into $\mathbb{R}^m$. The set $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{g}(\mathbf{x}) = \mathbf{0}\}$ is a surface in $\mathbb{R}^n$. This surface is approximated near $\mathbf{x}^*$ by $\mathbf{x}^* + M$, where

$$M = \{\mathbf{y} : \dot{\mathbf{g}}(\mathbf{x}^*)'\mathbf{y} = \mathbf{0}\}.$$

In order for this approximation to hold, $\mathbf{x}^*$ should be a *regular point* of the constraint, i.e. $(\dot{g}_1(\mathbf{x}^*), \ldots, \dot{g}_m(\mathbf{x}^*))$ should be linearly independent.

Theorem 7.1.1 (Lagrange multipliers)**.** *Let* $\mathbf{x}^*$ *be a local extremum point of* f *subject to the constraint* $\mathbf{g} = \mathbf{0}$*. Assume that* $\mathbf{x}^*$ *is a regular point of these constraints. Then there is a* $\lambda \in \mathbb{R}^m$ *such that*

$$\dot{f}(\mathbf{x}^*) + \dot{\mathbf{g}}(\mathbf{x}^*)\lambda = \dot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \dot{\mathbf{g}}_j(\mathbf{x}^*) = \mathbf{0}.$$

Given λ, one can consider the *Lagrangian*:

$$l(\mathbf{x}, \lambda) = f(\mathbf{x}) + \mathbf{g}(\mathbf{x})\lambda.$$

The necessary conditions can be formulated as $\dot{l} = \mathbf{0}$. The matrix of partial second derivatives of l (with respect to $\mathbf{x}$) at $\mathbf{x}^*$ is

$$\ddot{l}_{\mathbf{x}}(\mathbf{x}^*) = \ddot{f}(\mathbf{x}^*) + \ddot{\mathbf{g}}(\mathbf{x}^*)\lambda = \ddot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \ddot{\mathbf{g}}_j(\mathbf{x}^*)\lambda_j$$

We say that this matrix is positive semidefinite over M if $\mathbf{x}'\ddot{l}_{\mathbf{x}}(\mathbf{x}^*)\mathbf{x} \geq 0$, for all $\mathbf{x} \in M$.

Theorem 7.1.2 (Second-order condition)**.** *Let* $\mathbf{x}^*$ *be a local extremum point of* f *subject to the constraint* $\mathbf{g} = \mathbf{0}$*. Assume that* $\mathbf{x}^*$ *is a regular point of these constraints, and let* $\lambda \in \mathbb{R}^m$ *be such that*

$$\dot{f}(\mathbf{x}^*) + \dot{\mathbf{g}}(\mathbf{x}^*)\lambda = \dot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \dot{\mathbf{g}}_j(\mathbf{x}^*) = \mathbf{0}.$$

Then the matrix $\ddot{l}_{\mathbf{x}}(\mathbf{x}^*)$ *is positive semidefinite over* M*.*

7.2 Examples

We now give some applications of the above theory.

Example 7.2.1. *Minimize* $f(x, y, z) = xy + yz + xz$*, subject to* $x + y + z = 3$*.*

The necessary conditions become:

$$\begin{aligned} y + z + \lambda &= 0 \\ x + z + \lambda &= 0 \\ x + y + \lambda &= 0 \\ x + y + z &= 3. \end{aligned}$$

Solving this system gives $x = y = z = 1$, $\lambda = -2$.

Example 7.2.2. *A discrete random variable takes the values $x_1, \ldots, x_d$, with probabilities $p_1, \ldots, p_d$. For a given mean value m, find the distribution which minimizes the entropy*

$$f(p_1, \ldots, p_d) = -\sum_{i=1}^{d} p_i \log(p_i).$$

The problem can be formulated as

$$\min f(p_1, \ldots, p_d)$$

subject to:

$$\sum_{i=1}^{d} p_i = 1, \sum_{i=1}^{d} x_i p_i = m$$
$$0 \leq p_i,\ i = 1, \ldots, d.$$

Ignoring the inequality constraints, the Lagrangian becomes

$$l(p_1, \ldots, p_d, \lambda_1, \lambda_2) = \sum_{i=1}^{d} \{-p_i \log(p_i) + \lambda_1 p_i + \lambda_2 x_i p_i\} - \lambda_1 - m\lambda_2.$$

The necessary conditions are

$$-\log(p_i) - 1 + \lambda_1 + \lambda_2 x_i = 0, \quad i = 1, \ldots, d,$$

which leads to

$$p_i = \exp\{(\lambda_1 - 1) + \lambda_2 x_i\},\ \sum_{i=1}^{d} p_i = 1,\ \sum_{i=1}^{d} x_i p_i = m.$$

Note that the solution satisfies the inequality constraints.

Example 7.2.3. *A chain is suspended between two hooks that are t meters apart on a horizontal line. The chain consists of d links. Each link is 1 meter long (measured from the inside). What is the shape of the chain?*

We intend to minimize the potential energy of the chain. We let link i a x_i distance horizontally and y_i distance vertically. The potential energy of a link is its weight times its vertical distance (from some reference point). The potential energy of the chain is the sum of the potential energies of the links. Take the top as the reference and assume that the mass of each link is

concentrated at the center of the link. The potential energy is proportional to

$$\begin{aligned} f(y_1,\ldots,y_d) &= 0.5y_1 + (y_1 + 0.5y_2) + \cdots + (y_1 + \cdots + y_{d-1} + 0.5y_d) \\ &= \sum_{i=1}^{d}(d - i + 0.5)y_i. \end{aligned}$$

The constraints are:

$$\sum_{i=1}^{d} y_i = 0, \sum_{i=1}^{d} x_i = \sum_{i=1}^{d} \sqrt{1 - y_i^2} = t.$$

The first order necessary conditions are

$$(d - i + 0.5) + \lambda_1 - \frac{\lambda_2 y_i}{(1 - y_i^2)^{1/2}} = 0, \text{ for } i = 1,\ldots,d,$$

which leads to the solution

$$y_i = -\frac{d - i + 0.5 + \lambda_1}{[\lambda_2^2 + (d - i + 0.5 + \lambda_1)^2]^{1/2}}.$$

7.3 Necessary conditions (inequality constraints)

We now consider the case where $m_e < m$. Let $\mathbf{x}^*$ be a solution of the constrained optimization problem. A constraint g_j is *active* at $\mathbf{x}^*$ if $g_j(\mathbf{x}^*) = 0$ and it is *inactive* if $g_j(\mathbf{x}^*) < 0$. Note that all equality constraints are active. Denote by J the set of all active constraints.

For the consideration of necessary conditions when inequality constraints are present the definition of a regular point should be extended. We say now that $\mathbf{x}^*$ is regular if $\{\dot{g}_j(\mathbf{x}^*) : j \in J\}$ are linearly independent.

Theorem 7.3.1 (Kuhn-Tucker Conditions)**.** *Let $\mathbf{x}^*$ be a local extremum point of f subject to the constraint $g_j(\mathbf{x}) = 0$, $1 \le j \le m_e$ and $g_j(\mathbf{x}) \le 0$, $m_e + 1 \le j \le m$. Assume that $\mathbf{x}^*$ is a regular point of these constraints. Then there is a $\lambda \in \mathbb{R}^m$ such that $\lambda_j \ge 0$, for all $j > m_e$, and*

$$\begin{aligned} \dot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \dot{\mathbf{g}}_j(\mathbf{x}^*) &= \mathbf{0} \\ \sum_{j=m_e+1}^{m} \lambda_j \mathbf{g}_j(\mathbf{x}^*) &= \mathbf{0}. \end{aligned}$$

Let

$$\ddot{l}_{\mathbf{x}}(\mathbf{x}^*) = \ddot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \ddot{\mathbf{g}}_j(\mathbf{x}^*).$$

Theorem 7.3.2 (Second-order condition)**.** *Let $\mathbf{x}^*$ be a local extremum point of f subject to the constraint $g_j(\mathbf{x}) = 0$, $1 \leq j \leq m_e$ and $g_j(\mathbf{x}) \leq 0$, $m_e + 1 \leq j \leq m$. Assume that $\mathbf{x}^*$ is a regular point of these constraints, and let $\lambda \in \mathbb{R}^m$ be such that $\lambda_j \geq 0$, for all $j > m_e$, and*

$$\dot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \dot{\mathbf{g}}_j(\mathbf{x}^*) = \mathbf{0}.$$

Then the matrix $\ddot{l}_{\mathbf{x}}(\mathbf{x}^*)$ is positive semidefinite on the tangent subspace of the active constraints.

7.4 Sufficient conditions

Sufficient conditions are based on second-order conditions:

Theorem 7.4.1 (Equality constraints)**.** *Suppose there is a point $\mathbf{x}^*$ satisfying $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$, and a $\lambda \in \mathbb{R}^m$ such that*

$$\dot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \dot{\mathbf{g}}_j(\mathbf{x}^*) = \mathbf{0}.$$

Suppose also that the matrix $\ddot{l}_{\mathbf{x}}(\mathbf{x}^*)$ is positive definite on M. Then $\mathbf{x}^*$ is a strict local minimum for the constrained optimization problem.

Theorem 7.4.2 (Inequality constraints)**.** *Suppose there is a point $\mathbf{x}^*$ that satisfies the constraints. A sufficient condition for $\mathbf{x}^*$ to be a strict local minimum for the constrained optimization problem is the existence of a $\lambda \in \mathbb{R}^m$ such that $\lambda_j \geq 0$, for $\quad m_e < j \leq m$, and*

$$\dot{f}(\mathbf{x}^*) + \sum_{j=1}^{m} \lambda_j \dot{\mathbf{g}}_j(\mathbf{x}^*) = \mathbf{0} \tag{7.1}$$

$$\sum_{j=m_e+1}^{m} \lambda_j \mathbf{g}_j(\mathbf{x}^*) = \mathbf{0}, \tag{7.2}$$

and the Hessian matrix $\ddot{l}_{\mathbf{x}}(\mathbf{x}^*)$ is positive on the subspace

$$M' = \{\mathbf{y} : \dot{\mathbf{g}}_j(\mathbf{x}^*)'\mathbf{y} = 0, j \in J\}$$

where $J = \{j : \mathbf{g}_j(\mathbf{x}^*) = 0, \lambda_j > 0\}$

Example 7.4.1. *Consider the problem:*

$$\begin{array}{ll} \textit{minimize} & 2x^2 + 2xy + y^2 - 10x - 10y \\ \textit{subject to} & x^2 + y^2 \leq 5 \\ & 3x + y \leq 6. \end{array}$$

The first order necessary conditions are

$$\begin{aligned} 4x + 2y - 10 + 2\lambda_1 3 + 3\lambda_2 &= 0 \\ 2x + 2y - 10 + 2\lambda_1 y + \lambda_2 &= 0 \\ \lambda_1 \geq 0, \quad \lambda_2 &\geq 0 \\ \lambda_1(x^2 + y^2 - 5) &= 0 \\ \lambda_2(3x + y - 6) &= 0. \end{aligned}$$

One should check different subsets of active and inactive constraints. For example, if we set $J = \{1\}$ then

$$\begin{aligned} 4x + 2y - 10 + 2\lambda_1 3 + 3\lambda_2 &= 0 \\ 2x + 2y - 10 + 2\lambda_1 y + \lambda_2 &= 0 \\ x^2 + y^2 &= 5, \end{aligned}$$

which has the solution $x = 1$, $y = 2$, $\lambda_1 = 1$. This yields $3x + y = 5$, and hence the second constraint is satisfied. Thus, since $\lambda_1 > 0$, we conclude that this solution satisfies the first order necessary conditions.

7.5 Homework

1. Consider the constraints $x_1 \geq 0$, $x_2 \geq 0$ and $x_2 - (x_1 - 1)^2 \leq 0$. Show that $(1, 0)$ is feasible but not regular.

2. Find the rectangle of given perimeter that has greatest area by solving the first-order necessary conditions. Verify that the second-order sufficient conditions are satisfied.

3. Three types of items are to be stored. Item A costs one dollar, item B costs two dollars and item C costs 4 dollars. The demand for the three items are independent and uniformly distributed in the range $[0, 3000]$. How many of each type should be stored if the total budget is 4,000 dollars?

4. Let A be an $n \times m$ matrix of rank m and let L be an $n \times n$ matrix that is symmetric and positive-definite on the subspace $M = \{\mathbf{y} : A\mathbf{y} = \mathbf{0}\}$. Show that the $(n+m) \times (n+m)$ matrix

$$\begin{bmatrix} L & A' \\ A & \mathbf{0} \end{bmatrix}$$

is non-singular.

5. Maximize $14x - x^2 + 6y - y^2 + 7$ subject to $x + y \leq 2$, $x + 2y \leq 3$.

Chapter 8

Quadratic Programming

The Lagrange methods for dealing with constrained optimization problem are based on solving the Lagrange first-order necessary conditions. In particular, for solving the problem with equality constraints only:

$$\begin{array}{rl} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{g}(\mathbf{x}) = \mathbf{0}, \end{array}$$

the algorithms look for solutions of the problem:

$$\begin{aligned} \dot{f}(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j \dot{g}_j(\mathbf{x}) &= \mathbf{0} \\ \mathbf{g}(\mathbf{x}) &= \mathbf{0}, \end{aligned}$$

8.1 Quadratic programming

An important special case is when the target function f is quadratic and the constraints are linear:

$$\begin{array}{rll} \text{minimize} & (1/2)\mathbf{x}'Q\mathbf{x} + \mathbf{x}'\mathbf{c} & \\ \text{subject to} & \mathbf{a}_i'\mathbf{x} = b_i, & 1 \le i \le m_e \\ & \mathbf{a}_i'\mathbf{x} \le b_i, & m_e + 1 \le i \le m \end{array}$$

with Q a symmetric matrix.

8.1.1 Equality constraints

In the particular case where $m_e = m$ the above becomes

$$\begin{array}{rl} \text{minimize} & (1/2)\mathbf{x}'Q\mathbf{x} + \mathbf{x}'\mathbf{c} \\ \text{subject to} & A\mathbf{x} = \mathbf{b}. \end{array}$$

and the Lagrange necessary conditions become

$$\begin{aligned} Q\mathbf{x} + A'\lambda + \mathbf{c} &= \mathbf{0} \\ A\mathbf{x} - \mathbf{b} &= \mathbf{0}. \end{aligned}$$

This system is nonsingular if Q is positive definite on the subspace $M = \{\mathbf{x} : A\mathbf{x} = \mathbf{0}\}$. If Q is nonsingular then the solution becomes:

$$\begin{aligned} \mathbf{x} &= Q^{-1}A'(AQ^{-1}A')^{-1}[AQ^{-1}\mathbf{c} + \mathbf{b}] - Q^{-1}\mathbf{c} \\ \lambda &= -(AQ^{-1}A')^{-1}[AQ^{-1}\mathbf{c} + \mathbf{b}]. \end{aligned}$$

8.2 Quadratic Programming in R

Let us start with the implementation of quadratic programming with equality constraints. Consider the following example:

```
> Q.quad <- matrix(c(0.78,-0.02,-0.12,-0.14,-0.02,0.86,-0.04,0.06,
+     -0.12,-0.04,0.72,-0.08,-0.14,0.06,-0.08,0.74),
+     4,4,byrow=TRUE)
> Q.quad
      [,1]  [,2]  [,3]  [,4]
[1,]  0.78 -0.02 -0.12 -0.14
[2,] -0.02  0.86 -0.04  0.06
[3,] -0.12 -0.04  0.72 -0.08
[4,] -0.14  0.06 -0.08  0.74
> b.quad <- c(0.76,0.08,1.12,0.68)
> A.cons <- matrix(c(1,1,1,1,1,1,-1,-1),nrow=2,byrow=TRUE)
> A.cons
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    1   -1   -1
> b.cons <- c(0,0)
> b.cons
[1] 0 0
```

Observe that $d = 4$ in this example and $m = m_e = 2$. Next, construct a the matrix and the vector which correspond to the solution of the normal equations:

```
> V <- rbind(t(A.cons),matrix(0,nrow(A.cons),nrow(A.cons)))
> V <- cbind(rbind(Q.quad,A.cons),V)
> V
      [,1]  [,2]  [,3]  [,4] [,5] [,6]
[1,]  0.78 -0.02 -0.12 -0.14    1    1
[2,] -0.02  0.86 -0.04  0.06    1    1
[3,] -0.12 -0.04  0.72 -0.08    1   -1
[4,] -0.14  0.06 -0.08  0.74    1   -1
[5,]  1.00  1.00  1.00  1.00    0    0
[6,]  1.00  1.00 -1.00 -1.00    0    0
> u <- c(-b.quad,b.cons)
> u
[1] -0.76 -0.08 -1.12 -0.68  0.00  0.00
> solve(V,u)
[1] -0.3874113  0.3874113 -0.2429078  0.2429078 -0.7009397  0.2557270
```

We wrap it in a function that carries out quadratic programming with equality constraints:

```
> QP <- function(Q.quad,b.quad,A.cons,b.cons)
+ {
+     d <- length(b.quad)
+     m <- length(b.cons)
+     if(m==1) A.cons <- matrix(A.cons,nrow=1)
+     V <- rbind(t(A.cons),matrix(0,m,m))
+     V <- cbind(rbind(Q.quad,A.cons),V)
+     u <- c(-b.quad,b.cons)
+     xl <- solve(V,u)
+     x <- xl[1:d]
+     if(m >= 1) lam <- xl[1:m +d] else lam <- NA
+     return(list(x=x,lam=lam))
+ }
> QP(Q.quad,b.quad,A.cons,b.cons)
$x
[1] -0.3874113  0.3874113 -0.2429078  0.2429078
$lam
[1] -0.7009397  0.2557270
```

Let us compare our program to the implementation of quadratic programming in a contributed package to R. The package is called "quadprog" and it can be installed from the CRAN mirror. (Go to "`packages`" in the upper bar and select "`Install package(s)`". Choose the appropriate package from the list.) We implement that function "`solve.QP`". Consult its help file first:

```
> library(quadprog)
> ?solve.QP
```

Applying the function in the same setting we considered produces:

```
> solve.QP(Q.quad,-b.quad,t(A.cons),b.cons,meq=2)
$solution
[1] -0.3874113  0.3874113 -0.2429078  0.2429078
$value
[1] -0.1851596
$unconstrainted.solution
[1] -1.5349650 -0.1220096 -1.9751564 -1.4129555
$iterations
[1] 3 0
$iact
[1] 1 2
```

Compare the result to the results we got:

```
> solve.QP(Q.quad,-b.quad,t(A.cons),b.cons,meq=2)$solution
[1] -0.3874113  0.3874113 -0.2429078  0.2429078
> QP(Q.quad,b.quad,A.cons,b.cons)$x
[1] -0.3874113  0.3874113 -0.2429078  0.2429078
```

8.2.1 Inequality constraints

For the general quadratic programming problem, the method of *active set* is used. A working set of constraints W_n is updated in each iteration. The set W_n contains all constraints that are suspected to satisfy an equality relation at the solution point. In particular, it contains the equality constraints. An algorithm for solving the general quadratic problem is:

1. Start with a feasible point $\mathbf{x}_0$ and a working set W_0. Set $n = 0$

2. Solve the quadratic problem

$$\begin{aligned} \text{minimize} \quad & (1/2)\mathbf{d}'Q\mathbf{d} + (\mathbf{c} + Q\mathbf{x}_n)'\mathbf{d} \\ \text{subject to} \quad & \mathbf{a}_i'\mathbf{d} = 0, \quad i \in W_n. \end{aligned}$$

   If $\mathbf{d}_n^* = \mathbf{0}$ go to 4.

3. Set $x_{n+1} = \alpha_n \mathbf{d}_n^*$, where

$$\alpha_n = \min_{\mathbf{a}_i'\mathbf{d}_n^* > 0} \left\{ 1, \frac{b_i - \mathbf{a}_i'\mathbf{x}_n}{\mathbf{a}_i'\mathbf{d}_n^*} \right\}.$$

   If $\alpha_n < 1$, adjoin the minimizing index above to W_n to form W_{n+1}. Set $n = n + 1$ and return to step 2.

4. Compute the Lagrange multiplier in step 3 and let $\lambda_n = \min\{\lambda_i : i \in W_n, i > m_e\}$. If $\lambda_n \geq 0$, stop; $\mathbf{x}_n$ is a solution. Otherwise, drop λ_n from W_n to form W_{n+1} and return to step 2.

Example 8.2.1. *Consider the problem*

$$\begin{aligned} \textit{minimize} \quad & 2x^2 + xy + y^2 - 12x - 10y \\ \textit{subject to} \quad & (1) \quad x + y \leq 3.5, \\ & (2) \quad -x \leq 0, \\ & (3) \quad -y \leq 0. \end{aligned}$$

Take $\mathbf{x}_0 = (0, 0)'$, and $W_0 = \{2, 3\}$. Then $\mathbf{d}_0^* = (0, 0)'$. Both Lagrange multipliers are negative, but the one corresponding to (2) is more negative. Drop that constraint, and put $W_1 = \{3\}$. Minimizing along the line $y = 0$ leads to $\mathbf{x}_1 = (3, 0)'$. The Lagrange multiplier of the active constraint is negative, thus $W_2 = \emptyset$. Also, $\mathbf{d}_1^* = (-1, 4)$, the direction to the overall optimum at $(2, 4)'$. We move to the constraint (1), and write $W_3 = \{(1)\}$. Finally, we move along this constraint to the solution.

Let us carry out the above analysis using R an the function `QP`. We initiate by defining the parameters on the quadratic programming and plotting the feasible region and the target function:

```
> Q.quad <- matrix(c(4,1,1,2),2,2)
> Q.quad
     [,1] [,2]
[1,]    4    1
[2,]    1    2
```

```
> b.quad <- -matrix(c(12,10),2,1)
> A.cons <- matrix(c(1,-1,0,1,0,-1),3,2)
> A.cons
     [,1] [,2]
[1,]    1    1
[2,]   -1    0
[3,]    0   -1
> b.cons <- c(3.5,0,0)

> xx <- seq(-1,4,by=0.01)
> yy <- seq(-1,4,by=0.01)
> zz <- outer(xx,yy,function(x,y) 2*x^2+x*y+y^2-12*x-10*y)
> contour(xx,yy,zz,nlev=30)
> abline(3.5,-1,col=2)
> abline(h=0,col=2)
> abline(v=0,col=2)
> x <- seq(0,3.5,length=20)
> segments(x,rep(0,length(x)),x,3.5-x,col=2)
```

The algorithm is initiated by setting a starting value and an initial working collection of active constraints.

```
> # initiate: n=0
> b.0 <- rep(0,3)
> x0 <- c(0,0)
> W0 <- 2:3
> # 2.
> qp <- QP(Q.quad,b.quad+Q.quad%*%x0,A.cons[W0,],b.0[W0])
> # 4.
> qp
$x
[1] 0 0
$lam
[1] -12 -10
> W1 <- 3
> x1 <- x0
> points(x1[1],x1[2],col="blue",cex=2)
```

Observe that most negative Lagrange multiplier is associated with the constraint (2), which is removed from the working set.

```
> # n=1
```

```
> # 2.
> qp <- QP(Q.quad,b.quad+Q.quad%*%x1,A.cons[W1,],b.0[W1])
> d1 <- qp$x
> # 3.
> alpha <- (b.cons - A.cons%*%x1)/A.cons%*%d1
> alpha[A.cons%*%d1 > 0]
[1] 1.166667
> # 4.
> qp$lam
[1] -7
> W2 <- 0
> x2 <- x1+d1
> points(x2[1],x2[2],col="blue",cex=2)
> segments(x1[1],x1[2],x2[1],x2[2],lty=2,col="blue",lwd=2)
```

At this stage no constraint is added. The solution of the associated quadratic programming problem produces a negative lagrange multiplier. Consequently, the working set becomes empty. Note that the function `QP` is still working.

```
> # n=2
> # 2.
> qp <- QP(Q.quad,b.quad+Q.quad%*%x2,A.cons[W2,],b.0[W2])
> d2 <- qp$x
> # 3.
> alpha <- (b.cons - A.cons%*%x2)/A.cons%*%d2
> alpha
          [,1]
[1,] 0.1666667
[2,] 3.0000000
[3,] 0.0000000
> alpha[A.cons%*%d2 > 0]
[1] 0.1666667 3.0000000
> W3 <- c(1)
> x3 <- x2+min(alpha[A.cons%*%d2 > 0])*d2
> points(x3[1],x3[2],col="blue",cex=2)
> segments(x2[1],x2[2],x3[1],x3[2],lty=2,col="blue",lwd=2)
```

Now, the constarint (1) is added to the working set. That produces, in te next iteration, the optimal solution:

```
> # n=3
```

```
> # 2.
> qp <- QP(Q.quad,b.quad+Q.quad%*%x3,A.cons[W3,],b.0[W3])
> d3 <- qp$x
> # 3.
> alpha <- (b.cons - A.cons%*%x3)/A.cons%*%d3
> alpha
           [,1]
[1,]  0.0000000
[2,]  1.9428571
[3,] -0.4571429
> alpha[A.cons%*%d3 > 0]
[1] 0.000000 1.942857
> A.cons%*%d3
              [,1]
[1,]  4.440892e-16
[2,]  1.458333e+00
[3,] -1.458333e+00
> # 4.
> qp$lam
[1] 4.375
> x4 <- x3 + d3
> points(x4[1],x4[2],col="blue",cex=2)
> segments(x3[1],x3[2],x4[1],x4[2],lty=2,col="blue",lwd=2)
```

8.3 Homework

1. Write an R function that implements quadratic programming (QP) when some constraints may be inequality constraints. Compare the implementation of your function to the implementation of the function "`solve.QP`".

Chapter 9

Sequential Quadratic Programming

Let us consider again the general optimization problem with constraints

$$\begin{array}{rl} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{g}_i(\mathbf{x}) = 0, \quad i = 1, \ldots, m_e \\ & \mathbf{g}_i(\mathbf{x}) \leq 0, \quad i = m_e + 1, \ldots, m. \end{array}$$

The SQP method solves this problem by solving a sequence of QP problems where the Lagrangian function l is approximated by a quadratic function and the constraints are approximated by a linear hyper-space.

9.1 Newton's Method

Consider the case of equality constraints only. At each iteration the problem

$$\begin{array}{rl} \text{minimize} & (1/2)\mathbf{d}'\ddot{l}_{\mathbf{x}}(\mathbf{x}_n, \lambda_n)\mathbf{d} + \dot{l}(\mathbf{x}_n, \lambda_n)'\mathbf{d} \\ \text{subject to} & \dot{g}_i(\mathbf{x}_n)'\mathbf{d} + g_i(\mathbf{x}_n) = 0, \quad i = 1, \ldots, m \end{array}$$

is solved. It can be shown that the rate of convergence of this algorithm is 2 (at least) if the starting point $(\mathbf{x}_0, \lambda_0)$ is close enough to the solution $(\mathbf{x}^*, \lambda^*)$. A disadvantage of this approach is the need to compute Hessian.

9.2 Structured Methods

These methods are modifications of the basic Newton method, with approximations replacing Hessian. One can rewrite the solution to the Newton step

in the form

$$\begin{bmatrix} \mathbf{x}_{n+1} \\ \lambda_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_n \\ \lambda_n \end{bmatrix} - \begin{bmatrix} \ddot{l}_n & \dot{\mathbf{g}}_n' \\ \dot{\mathbf{g}}_n & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \dot{l}_n \\ \mathbf{g}_n \end{bmatrix}.$$

Instead, one can use the formula

$$\begin{bmatrix} \mathbf{x}_{n+1} \\ \lambda_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_n \\ \lambda_n \end{bmatrix} - \alpha_n \begin{bmatrix} H_n & \dot{\mathbf{g}}_n' \\ \dot{\mathbf{g}}_n & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \dot{l}_n \\ \mathbf{g}_n \end{bmatrix},$$

with α_n and H_n appropriately chosen.

9.3 The Han–Powell method

The Han–Powell method is what is used by Matlab for SQP. It is a Quasi-Newton method, where H_n is updated using the BFGS approach:

$$H_{n+1} = H_n + \frac{(\Delta l)(\Delta l)'}{(\Delta \mathbf{x})'(\Delta l)} - \frac{H_n(\Delta \mathbf{x})(\Delta \mathbf{x})' H_n}{(\Delta \mathbf{x})' H_n (\Delta \mathbf{x})},$$

where

$$\Delta \mathbf{x} = \mathbf{x}_{n+1} - \mathbf{x}_n, \quad \Delta l = l(\mathbf{x}_{n+1}, \lambda_{n+1}) - l(\mathbf{x}_n, \lambda_n).$$

It can be shown that H_{n+1} is positive-definite if H_n is and if $(\Delta \mathbf{x})'(\Delta l) > 0$.

9.4 Merit function

In order to choose the α_n and to assure that the algorithm will converge a *merit function* is associated with the problem such that a solution of the constrained problem is a (local) minimum of the merit function. The algorithm should be descending with respect to the merit function.

Consider, for example, the problem with inequality constraints only:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m. \end{aligned}$$

The absolute-value merit function is given by

$$Z(\mathbf{x}) = f(\mathbf{x}) + c \sum_{i=1}^{m} g_i(\mathbf{x})^+.$$

The parameter α is chosen by minimizing the merit function in the direction chosen by the algorithm.

Theorem 9.4.1. *If H is positive-definite and if $c > \max_{1\le i\le m} \lambda_i$ then the algorithm is descending with respect to the absolute-value merit function.*

Proof. The optimization problem is solved by solving sequentially problems of the form:

$$\begin{array}{ll} \text{minimize} & (1/2)\mathbf{d}'H\mathbf{d} + \dot{f}(\mathbf{x})'\mathbf{d} \\ \text{subject to} & \dot{g}_j(\mathbf{x})'\mathbf{d} + g_j(\mathbf{x}) \le 0, \quad i = 1, \ldots, m, \end{array}$$

The necessary conditions here are

$$\begin{aligned} H\mathbf{d} + \dot{f}(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j \dot{g}_j(\mathbf{x}) &= \mathbf{0} && (9.1)\\ \dot{g}_j(\mathbf{x})'\mathbf{d} + g_j(\mathbf{x}) &\le 0 && (9.2)\\ \lambda_j[\dot{g}_j(\mathbf{x}) + g_j(\mathbf{x})] &= 0 && (9.3)\\ \lambda_j &\ge 0. && (9.4) \end{aligned}$$

Let $J(\mathbf{x}) = \{g_i(\mathbf{x}) > 0\}$. Now, for $\alpha > 0$,

$$\begin{aligned} Z(\mathbf{x} + \alpha\mathbf{d}) &= f(\mathbf{x} + \alpha\mathbf{d}) + c\sum_{i=1}^{m} g_i(\mathbf{x} + \alpha\mathbf{d})^+ \\ &= f(\mathbf{x}) + \alpha\dot{f}(\mathbf{x})'\mathbf{d} + c\sum_{i=1}^{m}[g_i(\mathbf{x})^+ + \alpha c\dot{g}_j(\mathbf{x})'\mathbf{d} + o(\alpha)]^+ \\ &= f(\mathbf{x}) + \alpha\dot{f}(\mathbf{x})'\mathbf{d} + c\sum_{i=1}^{m} g_i(\mathbf{x})^+ + \alpha c\sum_{j\in J(\mathbf{x})} \dot{g}_j(\mathbf{x})'\mathbf{d} + o(\alpha) \\ &= Z(\mathbf{x}) + \alpha\dot{f}(\mathbf{x})'\mathbf{d} + \alpha c\sum_{j\in J(\mathbf{x})} \dot{g}_j(\mathbf{x})'\mathbf{d} + o(\alpha). \end{aligned}$$

Here we applied condition (9.2) in order to infer that $\dot{g}_j(\mathbf{x})'\mathbf{d} \le 0$ if $g_j(\mathbf{x}) = 0$. Using this condition again we get

$$c\sum_{j\in J(\mathbf{x})} \dot{g}_j(\mathbf{x})'\mathbf{d} \le c\sum_{j\in J(\mathbf{x})} -g_j(\mathbf{x}) = -c\sum_{j=1}^{m} g_j(\mathbf{x})^+. \qquad (9.5)$$

Using (9.1) we can infer that

$$\dot{f}(\mathbf{x})'\mathbf{d} = -\mathbf{d}'H\mathbf{d} - \sum_{j=1}^{m} \lambda_j \dot{g}_j(\mathbf{x})\mathbf{d},$$

which by using condition (9.3) leads to

$$\dot{f}(\mathbf{x})'\mathbf{d} \leq -\mathbf{d}'H\mathbf{d} + \sum_{j=1}^{m} \lambda_j g_j(\mathbf{x})^+ \leq -\mathbf{d}'H\mathbf{d} + \max_j \lambda_j \sum_{j=1}^{m} g_j(\mathbf{x})^+. \quad (9.6)$$

Remark 1. *The first inequality in (9.6) follows from the fact that:*

$$-\sum_{j=1}^{n} \lambda_j \dot{g}_j(\mathbf{x})\mathbf{d} = -\sum_{j=1}^{n} \lambda_j [\dot{g}_j(\mathbf{x})\mathbf{d} + g_j(\mathbf{x})] + \sum_{j=1}^{n} \lambda_j g_j(\mathbf{x}),$$

which by (9.3) equals $\sum_{j=1}^{n} \lambda_j g_j(\mathbf{x})$. *Obviously,* $g_j(\mathbf{x}) \leq g_j(\mathbf{x})^+$. *Thus, the first inequality. The second inequality, and the rest of the proof, are straightforward.*

Summarizing what we got thus far leads to

$$Z(\mathbf{x} + \alpha\mathbf{d}) \leq Z(\mathbf{x}) + \alpha\{-\mathbf{d}'H\mathbf{d} - [c - \max_j \lambda_j] \sum_{j=1}^{m} g_j(\mathbf{x})^+\} + o(\alpha).$$

The conclusion follows from the assumption that H is positive definite and the assumption $c \geq \max_j \lambda_j$. □

9.5 Enlargement of the feasible region

Consider, again, the problem with inequality constraints only:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}_i(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m. \end{aligned}$$

and its solution with a structural SQP algorithm.

Assume that at the current iteration $\mathbf{x}_n = \mathbf{x}$ and $H_n = H$. Then one wants to consider the QP problem:

$$\begin{aligned} \text{minimize} \quad & (1/2)\mathbf{d}'H\mathbf{d} + \dot{f}(\mathbf{x}) \\ \text{subject to} \quad & \dot{g}_i(\mathbf{x})'\mathbf{d} + g(\mathbf{x}) \leq 0, \quad i = 1, \ldots, m. \end{aligned}$$

However, it is possible that this problem is infeasible at the point $\mathbf{x}$. Hence, the original method breaks down. However, one can consider instead the problem

$$\begin{aligned} \text{minimize} \quad & (1/2)\mathbf{d}'H\mathbf{d} + \dot{f}(\mathbf{x}) + c\sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & \dot{g}_i(\mathbf{x})'\mathbf{d} + g(\mathbf{x}) \leq \xi_i, \quad i = 1, \ldots, m. \\ & -\xi_i \leq 0, \quad i = 1, \ldots, m, \end{aligned}$$

which is always feasible.

Theorem 9.5.1. *If H is positive-definite and if $c > \max_{1\leq i\leq m} \lambda_i$ then the algorithm is descending with respect to the absolute-value merit function.*

9.6 Homework

1. Let H be a positive-definite matrix and assume that, throughout some compact set, the quadratic programming has a unique solution, such that the Lagrange multipliers are not larger than c. Let $\{\mathbf{x}_n : n \geq 0\}$ be a sequence generated by the recursion $\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{d}_n$, where $\mathbf{d}$ is the direction found by solving the QP centered at $\mathbf{x}_n$ and with H fixed and α_n is determined by minimization of the function Z. Show that any limit point of $\{\mathbf{x}_n\}$ satisfies the first order necessary conditions for the constrained minimization problem.

   (Hint: Define the solution set to be all points that satisfy the first order necessary conditions. Show that the function Z is strictly decreasing outside of the solution set, and not increasing inside that set. Conclude the result from the appropriate theorem.)

2. Clairaut's Theorem states that if second order partial derivatives are continuous at a point then they commute at that point:

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i}.$$

   Consider the function

$$f(x, y) = \frac{xy(x^2 - y^2)}{x^2 + y^2}.$$

   Observe that $\partial^2 f(0, 0)/\partial x \partial y$ and $\partial^2 f(0, 0)/\partial y \partial x$ both exist but they are not equal.