Business Analytics

– SEMINAR SUMMER SEMESTER 2014 –

An Introduction to Optimization with the Statistical Software “R”

– SEMINAR PAPER –

Submitted by: Fabian Sprengholz

Advisor: Prof. Dr. Dirk Neumann

Table of contents

1. Introduction

Optimization is omnipresent. Although most people relate it intuitively to economic questions, it also appears in a lot of other contexts such as, for example, engineering, sports and science. And even if we do not call them this way, we are constantly solving optimization problems in our daily lives. We do so when making a plan for the next day or searching the fastest way through the city to get back to our car. Optimization simply means the search for the best solution of a problem. And a problem can be defined as a discrepancy between the actual state and the state we wish to have (e.g. Glass 2014). Such problems in real life are normally very complex and not easy to overview. With the help of mathematics, we can create formal models of those real life problems, which naturally are highly abstract and simplified. However, this reduction enables us to clearly define the goal of the optimization, the wished state, and the frame conditions of the optimization. Indeed the model helps us to overview the problem and to find the best solution.

In this paper, we do not discuss the modeling of optimization problems but focus on how to solve them. Although we can solve small optimization problems by hand, it becomes hard, even impossible, with big and complex problems. This paper aims to give an introduction to optimization with the statistical software "R" and wants to show how "R" helps to solve even complex problems in an easy manner.

The paper consists of three main parts. The first part is dedicated to the general structure of optimization problems and the description of characteristics allowing us to build up classes of optimization problems (see Section 2). Section 3 gives a survey of how "R" can be used for optimization and introduces the basic argument structure of a solver-function. The last part (Sections 4-6) describes basic optimization problems of linear, quadratic and non-linear programming as well as the appropriate solver functions in "R".

"R" commands, which can be directly implemented to "R", are marked in **`red`** and output is indicated in **`blue`**.

2. Classification of optimization problems

Optimization problems can appear in very different forms and thus require different solution approaches. It makes sense to create classes of optimization problems and develop class-specific solution methods which are appropriate for the solution of all problems in one class. There is no universally accepted classification, but Figure 1 shows a possible way of structuring the main distinctive features of optimization problems (e.g. Sauer 2003) that will be discussed in the following.

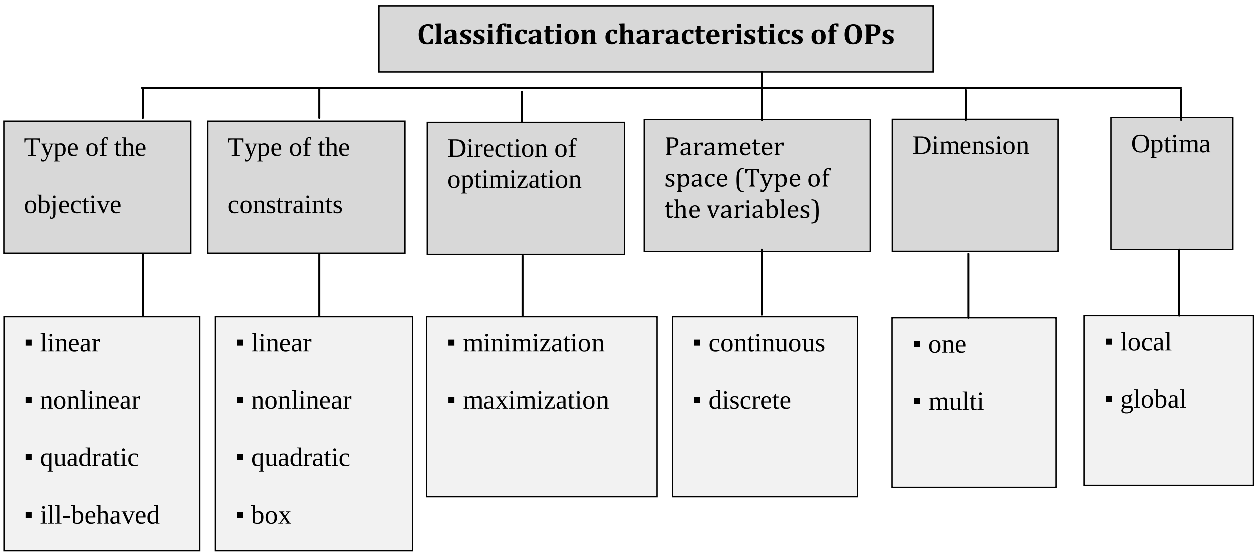

Figure 1: Diagram showing a possible structure of classification characteristics for OPs.

Type of the objective and direction of optimization

Objectives generally can be divided into two types: linear objectives and non-linear objectives. A third common type is the quadratic objective, a special case of the non-linear objective. If x is the vector of the variable $x = (x_1, \ldots\ldots, x_n)$, c is a numeric vector $c = (c_1, \ldots. c_n)$ and $F(x) = c^T x = c_1 x_1 + \ldots + c_n x_n$, then $F(x)$ is a linear objective.

When A is a symmetric matrix $A = a_{ij} \neq 0$ and $F(x) = x^T A x + c^T x = \sum_{i,j=0}^{n} a_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i$ then $F(x)$ is a quadratic objective. In all other cases $F(x)$ is a non-linear objective. The denomination ‘ill-behaved’ or ‘badly behaved’ objective can be also found but has no clear definition. Often it means objectives, which can not be analyzed with the methods of analysis (Zoonekynd 2012).

The objective shapes the decision problem of the optimization. It defines the relation between the decision parameters, the parameters that can be varied during the search for the best solu-

tion, and the result of the decision. For example, an enterprise has to decide which goods to produce (decision parameters) for maximizing profit (result).
Mathematically, the optimization problem can be described as the search for an extremum of the objective (Sauer 2003), i.e. This search, for an objective F, can be described, for example, by the following Equation (1) where $D^{'}$ is the feasible region and D the search space.

$$Find\ x^*, x \in \mathbb{R}\ with \left\{\begin{matrix} F(x^*) \geq F(x)\ (maximization) \\ F(x^*) \leq F(x)\ (minimization) \end{matrix}\ x \in D^{'} \in D\right\} \tag{1}$$

The direction of the optimization, the search for a maximum or minimum, is not relevant for the solution as the minimum of F is the maximum of $-F$, which is shown by Equation (2).

$$\underset{x^* \in D}{F}(x^*) = min\, F(x) \leftrightarrow F(x^*) \leq F(x) \leftrightarrow -F(x^*) \geq -F(x) \leftrightarrow -F(x^*) = max\, -F(x) \tag{2}$$

The solution of an optimization problem is a vector x^* that contains a value for each decision parameter.

Constraints and the feasible region

The feasible region of an optimization problem can be limited by constraints. Thus, first distinction is made between constrained and unconstrained optimization problems. Within the constrained problems, further distinction can be made depending on the sort of constraints. There are linear, quadratic, non-linear, integer and box constraints to be found.

The constraints of an optimization problem define which values the decision parameters may adopt and thus determine the feasible region and the possible solutions of the problem. The feasible region is a subspace of the search space. If the search space D contains all $\mathbb{R}$, the constraints define a subspace D' of the element D and the parameters are now allowed to take only values $x \in D'$. When an enterprise thinks about the goods to produce, it has to take into account their restricted resources, their budget, the capacity of the machines etc.

The constraints in a model can take the form of equalities or inequalities as it is shown by Equation (3).

$$\text{Find } x^* \text{ to maximize } F(x) \text{ with } x \in \mathbb{R}\ s.t. \left\{\begin{matrix} g(x) = 0 \\ h(x) \geq 0 \\ i(x) \leq 0 \end{matrix}\right\} \tag{3}$$

Constraints can complicate the search for solutions. So it can be appropriate to transform or dissolve constraints to simplify the problem. The inequality sign can be reversed by a multiplication with (-1) on both sides. The introduction of slack variables can transform inequalities into equalities. And sometimes constraints can be dissolved by reparametrization, where one variable is replaced by an unknown function. For example, the constraint $x \geq 0$ can be replaced by $x \rightarrow y^2$ (Zoonekynd 2012).

The parameter space

The parameter space contains all possible values for the parameters of a problem. One can distinguish between continuous and discrete optimization problems. In a continuous problem the variables are allowed to take any value permitted by the constraints. Thus, it can take arbitrary real numbers as values, and methods of analysis can often solve the problem. In a discrete problem this is not the case, the parameters can take only discrete values (Gould 2006).

Dimension

The dimension of an optimization problem depends on the amount of decision variables in the objective and the constraints (e.g. Glass 2014). It is a measure of the complexity of the problem. If there is only one variable, it is a matter of a one-dimensional optimization problem. If there are more variables, it is called multidimensional optimization.

Local and global optima

There are two types of optima, which can be distinguished: local and global optima. A local optimum exists in $x*$ when there is an interval $[a, b]$ around $x*$ so that $F(x) \leq F(\mathrm{x}^*)$ or $F(x) \geq F(x^*)$ for all $x \in D'$ in $[a, b]$. When $F(x^*)$ has the smallest or biggest function value in the whole feasible region $x \in D'$ and not only in a certain interval, it is called a global optimum.

Depending on the context of the search, it has to be decided whether to search a global or local solution. Here, the concept of convexity becomes relevant for optimization: If we are dealing with convex optimization problems, having a convex objective, such as linear and quadratic objectives, a local optimum is always a global optimum, too.

3. Basics of optimization with “R”

Simple optimization problems can be solved by hand. However, with increasing complexity, resulting from a big amount of variables and constraints, this becomes a hard and time consuming task. The statistical software “R” is a free and powerful tool that is easy to manage and helps to solve even complex optimization problems.

3.1 Levels of using “R”

Basically, “R” offers three different levels for solving optimization problems, which are represented in Figure 2. Firstly, a problem-solving algorithm can be implemented manually, defining each step of the algorithm and writing the commands to the command line. This needs a deep understanding of the mathematical background of the problem and of the solving method as well as the skill to transfer this knowledge into adequate commands.

The second way is much easier since it accesses work done by people who wrote scripts for most of current solving algorithms and provide these to the public as built-in functions in “R” packages. The CRAN Task View: optimization and mathematical programming (Theußl 2014) gives an overview of the multitude of packages available for optimization. For solving a problem with an in-built function, only the knowledge of the right solver function and its argument structure is needed.

The different solvers quite often vary in their argument structure. Because of that, the “R” Optimization Infrastructure (ROI), a framework “that promotes the development and use of interoperable (open source) optimization problem solvers for R” (Theußl 2011) was developed. The idea is to uniform the structure of optimization problems by defining them as objects and making them accessible for different solvers. This makes it easier for users because they do not need to know the solver-specific argument structure of every single solver but only the argument structure of the ROI framework.

In this work we concentrate on the in-built function level.

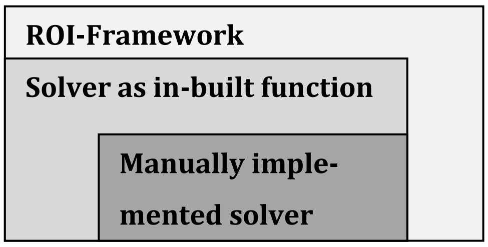

Figure 2: Levels of using “R” to solve optimization problems.

3.2 Basic argument structure of a solver in “R”

According to the features of an optimization problem described in Section 2, the basic argument structure of a solver function in “R” is as follows (Theußl 2011):

```
function(objective,constraints,bounds=NULL,types=NULL,maximum=FALSE)
```

The objective and the constraints have to be specified for every solver, other arguments often have default values.

- The objective has to be described depending on its type. So, for example, the coefficient c of a linear objective has to be a numeric vector. A quadratic objective has to be described with a symmetric matrix A and a numeric vector c for the linear part. The non-linear objective can be an arbitrary function (compare to examples in Sections 4-6).
- The constraints normally are divided into three arguments: one for the left-hand side (`lhs`), one for the right-hand side (`rhs`) and one for the direction of the constraints (`dir`). Usually, `lhs` is a numeric matrix and `rhs` is a numeric vector. The argument `dir` can be a vector of the character strings of (in)equalities signs such as “==”, “>=”, ”<=”, ”<”, ”>”. There are also other ways to indicate the direction of the constraints (see Section 5).
- If bounds (box constraints) have to be specified, this is often done with two arguments containing a numeric vector: one for the lower bound (`lower`), the other for the upper bound (`upper`). For example, the box constraint $l \leq x \leq u$ where $x = (x_1 \dots x_n)$, $l = (l_1 \dots l_n), u = (u_1 \dots u_n)$ are numeric vectors. Here $lower = l$ and $upper = u$ would be the arguments that specify the constraint (e.g. National Computational Infrastructure 2014).
- The parameters can take values of different types such as continuous, integer, binary or mixed. The default type normally is continuous. When functions can treat parameters of different types, the type has to be specified in an argument. This has to be done in very different ways depending on the function. Sometimes, there are arguments of the sort `all.int = TRUE`, which define the type of all parameters. In other functions, the type of every single parameter can be specified in a vector. One possibility could be an argument such as `type = c` where c is, for example, the vector $\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$ that indicates that the first and the fourth parameter are of the type integer.
- The usual default direction of optimization in “R” is minimization. However, some functions have an argument where the direction can be specified. One example for such an argument could be `maximum = TRUE`, in this manner maximization is here defined as di-

rection. Often, there is no specification of the direction because it is not relevant for the solution of an optimization problem (see Section 2).

In this section, a “R”-solver was described in a general, abstract way. The following three sections give an introduction to common optimization problems and show how to solve them in “R” with the help of specific in-built functions.

4. Linear programming with `lp()` and `solveLP()`

If the objective function and the constraints of an optimization problem are both linear, we call the problem linear programming. The standard form for a minimization problem (see Equation (4)) is given by Braun, Murdoch (2007).

$$\min_{x1,x2\ldots,xk} C(x) = c_1x_1 + \ldots + c_kx_k \; s.t. \left\{ \begin{matrix} a_{11}x_1 + \ldots + a_{1k}x_k \geq b_1 \\ a_{21}x_1 + \ldots + a_{2k}x_k \geq b_2 \\ a_{m1}x_1 + \ldots + a_{mk}x_k \geq b_m \\ x_1 \geq 0, \ldots, x_k \geq 0 \end{matrix} \right\} \tag{4}$$

If the variables can adopt only integer values, solving gets harder. This kind of problem often appears in production or traffic planning.

The most popular algorithm for solving such a problem is the simplex algorithm. It utilizes the knowledge that the optimal solution lies at a vertex of the feasible region, which is constricted by the linear constraints. The optimum has to be at an intersection of constraint boundaries (basic solution) and has to lie in the feasible region. The idea now is to compare the vertices of the basic solutions not randomly, but by searching from a starting-point always the adjacent vertex being closest to the optimum. In this manner the algorithm finds the shortest way from the starting point to the optimal solution. For a detailed description of the simplex method see Sauer (2003) and Nagesh Kumar (2014).

Example of programming – A farmer

A farmer has three kinds of animals: cows, sheep and pigs. To raise these animals, different resources are required. And with every animal a different profit is realized. Table 1 shows all considered parameters.

	Cow	**Sheep**	**Pig**	**Capacity**
Units of land	2	0.5	0.3	100
Units of additional food	1.3	0.8	1	200
Hours of work	120	30	60	5000
Hours of machine time	7	10	11	1100
Profit in $	200	100	120	

Table 1: Parameters for the optimization problem of the farmer.

How many animals of every kind should the farmer raise to maximize the profit?

The “R” packages `lpSolve` (Berkelaar et al. 2014) and `linprog` (Henningsen 2012) provide the built-in functions `lp()` and `solveLP()`, which are based on the simplex algorithm and help to solve the problem. The main arguments of these functions are:

- `objective.in:` the coefficient vector of the objective.
- `const.mat:` matrix of the coefficients in the left-hand side of the constraints; one row per constraint and one column per variable.
- `const.dir:` vector of character strings such as "<=", "==", ">=" that indicate the direction of the constraints.
- `const.rhs:` a numeric vector that contains the right-hand side values of the constraints.
- `direction:` defines the direction of the optimization „min“(default) or “max”.

The function `lp()` accepts only positive values for the variables: $x_1, x_2, x_3 \geq 0$.

The main task is to set up the problem and bring it into the required form so that the functions can solve it. The maximization problem of the farmer is formulated by Equation (5) where $x_1 = \text{cow};\ x_2 = \text{sheep};\ x_3 = \text{pig}$.

$$max\ 200x_1 + 100x_2 + 120x_3 \text{ s.t.} \left\{ \begin{array}{c} 2x_1 + 0.5x_2 + 0.3x_3 \leq 100 \text{ (land constraint)} \\ 1.3x_1 + 0.8x_2 + x_3 \leq 200 \text{ (food constraint)} \\ 120x_1 + 30x_2 + 60x_3 \leq 5000 \text{ (work constraint)} \\ 7x_1 + 10x_2 + 11x_3 \leq 1100 \text{ (machine constraint)} \end{array} \right\} \tag{5}$$

In Box 1 this problem is implemented and solved with the function `lpSolve.` In Box 2 the same is done with `solveLP`.

```
> library(lpSolve)

> objective.in <- c(200,100,120)
> const.mat <- matrix(nrow=4, ncol=3, c(2,0.5,0.3,1.3,0.8,1,120,30,60,7,10,11), byrow = TRUE)
> const.dir <- c("<=","<=","<=","<=")
> const.rhs <- c(100,200,5000,1100)
> lp <- lp(direction="max",objective.in,const.mat,const.dir,const.rhs)
> lp
Success: the objective function is 13232.32
> lp$solution
[1] 17.17172 97.97980  0.00000
```

Box 1: “R” code snippet for solving the problem “A farmer” with `lp()`.

```
>library(linprog)
>
>solveLP(objective.in,const.rhs,const.mat, maximum=TRUE,const.dir)

Objective function (Maximum): 13232.3   Constraints
                                             actual dir bvec    free    dual  dual.reg
Iterations in phase 1: 0                1   83.3333  <=  100 16.6667 0.00000   16.6667
Iterations in phase 2: 2                2  100.7071  <=  200 99.2929 0.00000   99.2929
Solution                                3 5000.0000  <= 5000  0.0000 1.31313 1700.0000
      opt                               4 1100.0000  <= 1100  0.0000 6.06061  808.3333
1 17.1717
2 97.9798                               All Variables (including slack variables)
3  0.0000                                       opt cvec   min.c     max.c      marg marg.reg
                                        1   17.1717  200 106.667 400.00000        NA       NA
Basic Variables                         2   97.9798  100  72.000 285.71429        NA       NA
        opt                             3    0.0000  120    -Inf 145.45455 -25.45455   62.963
1   17.1717                             S 1 16.6667    0      NA  36.36364   0.00000       NA
2   97.9798                             S 2 99.2929    0      NA 105.26316   0.00000       NA
S 1 16.6667                             S 3  0.0000    0    -Inf   1.31313  -1.31313 1700.000
S 2 99.2929                             S 4  0.0000    0    -Inf   6.06061  -6.06061  808.333
```

Box 2: “R” code snippet for solving the problem “A farmer” with `solveLP()`.

The function `lp()` shows the value of the maximized objective and, with the command `lp$solution`, the optimal levels of the variables. The farmer should keep 17.17 cows, 97.98 sheep and no pigs and his maximized profit would be 13232.32 $.

The function `solveLP()` from the package `linprog` has a more detailed output printing the whole final simplex table. It needs the same arguments as `lp()` but in a different order. Here, the values of all variables are printed and it becomes visible that 16.67 units of land and 99.29 units of additional food will be left and can be used in another way.

Despite its less detailed output, `lp()` often is a better choice because `solveLP()` is slow and can not handle integer programming (Braun 2013). It is evident that there is no fraction of a living animal, so that it is appropriate to define the variables x_1, x_2, x_3 to be of the type integer. With `lp()` this is very easy, only one more argument is needed, which is demonstrated in Box 3. It results that keeping 17 cows and 98 sheep is the solution for the farmer’s optimization problem.

```
> lp<- lp(direction="max",objective.in,const.mat,const.dir,const.rhs)
> int.vec<- c(1,2,3) #vector that gives the indices of the variables, which have to be integer
> lp<- lp(direction="max",objective.in,const.mat,const.dir,const.rhs, int.vec=int.vec)
> lp
Success: the objective function is 13200
>lp$solution
[1] 17 98  0
```

Box 3: “R” code snippet for solving the problem “A farmer” as integer programming with `lp()`.

5. Quadratic programming with `solve.QP()`

If the objective function of an optimization problem is quadratic and the constraints are linear, it is called quadratic programming. A basic form is: $\min_x \frac{1}{2} x^T Q x + c^T x \quad s.t.\ A^T x \geq b$ with x being the vector of n decision variables, Q a symmetric $n \times n$ matrix, c a numeric vector of length n and A being a $n \times k$ matrix where k is the number of constraints.

For solving such problems with "R", the `quadprog` package (Turlach, Weingessel 2013) provides the function `solve.QP()`, which takes the following main arguments:

- **`Dmat`**: Matrix of the quadratic objective (Q).
- **`dvec`**: Coefficient vector of the objective (c).
- **`Amat`**: Matrix defining the constraints (A).
- **`bvec`**: Vector containing the right-hand side values of the constraints (b).
- **`meq`**: a number m, which indicates that the first m constraints will be treated as equality constraints and all further constraints as inequality constraints.

Using `solve.QP`, one has to take into account that inequality constraints must be of the type "$\geq$ ". Inequalities of the form "$\leq$" have to be transformed by multiplication with (-1). Furthermore, the non-negativity constraints have to be indicated explicitly and the direction of the optimization is always minimization.

Example of quadratic optimization – Portfolio Selection

The following example is taken from Braun and Murdoch (2007). A stock investor wants to invest his fortune so that it maximizes his benefit (U). The benefit depends on the expected daily return (μ) and the risk (σ^2) of the portfolio so that the optimization problem can be described as $\max U(\mu, \sigma) = \mu - \frac{k}{2}\sigma^2$. The risk attitude of the investor is k ($k = 0$ means indifference towards risk). In this example, the investor has a risk tolerance of $k = 4$. The three stocks with the following parameters in Figure 3 are available. These values are usually won from historical data.

Expected daily return	s1	s2	s3
μ	0.002	0.005	0.01

Covariances	s1	s2	s3
s1	0.01	0.002	0.002
s2	0.002	0.01	0.002
s3	0.002	0.002	0.01

Figure 3: Parameters of the example "Portfolio Selection".

The question for the investor is now how to weigh the different stocks in the optimal portfolio. The problem expressed in the needed matrix notation is

$$\max U(p) = c^T x - 2x^T Q x \text{ with } c^T = [0.002\ 0.005\ 0.01] \text{ and } Q = \begin{bmatrix} 0.010 & 0.002 & 0.002 \\ 0.002 & 0.010 & 0.002 \\ 0.002 & 0.002 & 0.010 \end{bmatrix}.$$

The expected daily return of the three stocks is represented by the vector c, and Q is the covariance matrix that gives the variances and the covariance of the stocks. The variable x is the decision variable that describes how the fortune is divided between the stocks. It follows that:

- $c^T x = \mu_p$ is the expected daily return of the portfolio
- $2 = \frac{k}{2}$ is the risk attitude
- $x^T Q x = {\sigma_p}^2$ is the expected variance of the portfolio

The constraints are $\sum_{i=1}^{3} x_i = 1$ and $x_i \geq 0$ for $i = 1,2,3$. They indicate that the sum of the fractions of the fortune should be 1. Moreover, no fraction can be negative. This means that short selling is excluded. The constraints in matrix notation are $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{matrix} = \\ \geq \\ \geq \\ \geq \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$.

Given the needed form, it is easy to solve the problem with the function `solve.QP` (see Box 4).

```
> library(quadprog)
> Q<- matrix (c(0.01,0.002,0.002,0.002,0.01,0.002,0.002,0.002,0.01), nrow=3) #Dmat
> A<- t(matrix(c(1,1,0,0,1,0,1,0,1,0,0,1),nrow=4)) #Amat transposed
> c<- c(0.002,0.005,0.01) #dvec
> b<- c(1,0,0,0) #bvec
> solve.QP(2*Q,c,AA,b,meq =1) #meq=1 because the first constraint is an equality
$solution
[1] 0.1041667 0.2916667 0.6041667
$value
[1] -0.002020833
$unconstrained.solution
[1] -0.02678571  0.16071429  0.47321429
$iterations
[1] 2 0
$Lagrangian
[1] 0.003666667 0.000000000 0.000000000 0.000000000
$iact
[1] 1
```

Box 4: “R” code snippet for solving the problem “Portfolio Selection” with `solve.QP()`.

The output shows that the investor should invest 10.42% of his fortune in stock 1, 29.16 % in stock 2 and 60.42% in stock 3. The optimal value of the objective, the benefit, is 0.002020833. It appears as a negative value because the function always does minimizations. In the output, `$iact` = 1 indicates that the constraints are activated. If this is not the case, constrained optimization equals unconstrained optimization.

If we need to solve a quadratic objective subjected to quadratic constraints the package `Rcplex` (Bravo 2013) provides a possible solution.

6. Non-linear programming with `optimize()` and `optimx()`

All optimization problems with nor linear neither quadratic objective are called nonlinear. The methods to solve them can be divided into groups. Some methods use derivations, if the problem allows it, while others are derivative-free. Furthermore, one-dimensional and multidimensional methods exist.

The following methods are helpful for unconstrained optimization problems. If there are constraints, a new objective has to be constructed with the constraints being integrated (e.g. Lagrange method) so that still the methods of unconstrained optimization can be applied.

6.1 One dimensional non-linear programming with `optimize()`

One method for solving one-dimensional optimization problems without knowing their derivations is the “golden section search”. With this method, problems that have a single optimum in a specified interval can be solved. The algorithm starts with an interval $[a, b]$ containing the optimum. Afterwards, it narrows the interval by comparing function values until a predefined size is reached. Due to the direction being minimization, it is obvious that if $f(x_1) < f(x_2)$, the minimum has to be left of x_1 and the new search interval is $[a, x_1]$. Otherwise, it would be $[x_2, b]$. Within the new interval, the last step is repeated until the demanded interval size is attained. For a detailed explanation and a more efficient version with a constant reduction factor see Verschelde (2005).

The function `optimize()` from the package `stats` (R Core Team 2014) provides an implemented variation of the golden section search. It can be used to solve the following problem

$$\min_x f(x) = |x - 2| + 2|x - 1|. \tag{6}$$

The function f is not differentiable for $x = 1$ and $x = 2$, which can be seen in the plot that is shown in Figure 4.

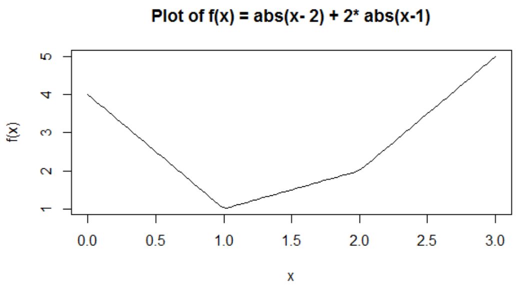

Figure 4: Finding a minimum of a one-dimensional function which is not differentiable.

The function `optimize()` needs the following four arguments:

- `f`: the objective
- `lower` : the minimum of the interval
- `upper`: the maximum of the interval
- **tol** : the tolerance, which defines the size of the final interval(there is an default value)

```
> f <- function(x) return(abs(x- 2)+ 2* abs(x-1))
>optimize(n.deriv.f, lower= 0, upper=3)
$minimum
[1] 1.000009
$objective
[1] 1.000009
```

Box 5: “R” code that applies the function `optimize()` to the problem in Equation(6).

In Box 5 `optimize()` was applied on the problem in Equation (6). The output shows 1.000009 as solution for the minimized x-value and the objective value. It seems to be a good approximation for the exact solution, which is 1.

6.2 Multidimensional non-linear programming

For the multidimensional, non-linear programming there are plenty of solving algorithms. In this section, three basic main methods as well as the “R”-function `optimx()` using solvers of all three methods will be shortly presented.

6.2.1 Basic solving algorithms

The solving algorithms, whose basic ideas will be presented in the following, are: the Nelder-Mead simplex method as a common non-derivative method, the steepest descent method as primary gradient method and the Newton-Raphson method as base for the hessian methods.

Non-derivative method – Nelder-Mead simplex algorithm

The Nelder-Mead simplex is a derivative-free method for solving non-linear optimization problems with dimensions greater than 1. For a more detailed introduction see Geiger, Kanzow (1999) and Braun, Murdoch (2007). Basically, three steps are taken:

1. $n+1$ points $x_1, x_2, \dots x_{n+1}$ are chosen with n being the number of variables. The points are arranged that they build an n-dimensional simplex with $n+1$ vertices.
2. The values of $f(x_i)$ are calculated and arranged in order according to its size $f(x_1) \leq f(x_2) \leq \cdots f(x_{n+1})$.
3. If the best value – for minimization the smallest – is good enough, the algorithm stops. If this is not the case, the worst value – for minimization the highest – is replaced and the algorithm continues with step 2.

The essential idea of Nelder-Mead is, replacing the ‘worst’ point with the following operations:

- Reflection to the center of gravity of the simplex formed by the other points and further expansion in the same direction if the resulted point is better.
- Contraction of the ‘worst ’point x_{n+1} towards the center of the simplex.
- Compression which is the contraction of all points towards the ‘best’ point.

Gradient method – steepest descent method

In contrast to the Nelder-Mead-algorithm, the method of the steepest descent uses the first derivation of the function to find the right direction. The search direction s_n of the next point results from the negative gradient of the latest point (e.g. Graichen 2012)

$$s_n = -\nabla f(x_n). \tag{7}$$

The next point is calculated by going one step with a certain step size a_n, which can be fixed or adapted, in the search direction

$$x_{n+1} = x_n + a_n \cdot s_n. \tag{8}$$

The process is repeated until $\nabla f(x_n) = 0$ or until a stopping criterion is satisfied.

Figure 5 shows the “zig-zagging” of the steepest descent algorithm searching the minimum of the Himmelblau’s function.

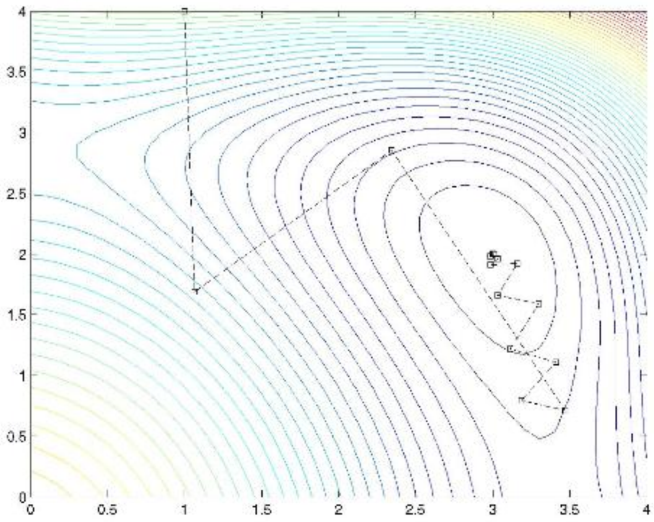

Figure 5: Finding a minimum of the Himmelblau’s function with steepest descent (Gaile 2008).

Based on the simple steepest descent method, the conjugate gradient method, which uses the information about the derivation of the prior iteration steps, was developed.

Newton-Raphson – the base of hessian methods

The basic idea of the Newton method is using the knowledge that the minimizer will fulfill the conditions $f'(x^*) = 0$ and $f''(x^*) > 0$.

With a Taylor series the tangent of a starting point is linearized

$$f'(x) \approx f'(x_0) + (x - x_0)f''(x_0). \tag{9}$$

Further it is equalized with zero to find an approximation for $f'(x^*) = 0$. At the next step, x_0 is replaced by x_1. Finally, the algorithm $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$ is repeated until $f'(x_n)$ is close enough to 0. In contrast to the steepest descent method, it uses the second derivation to find the optimal direction.

6.2.2 `optimx()` – solving non-linear optimization problems with "R"

The "R" function `optimx()` from the package `optimx` (Nash et al. 2013) has integrated solvers of every above-mentioned type and therefore is a really good and flexible tool. Three basic arguments are required by the function:

- **`par`**: a vector of initial values for the parameters, thus the starting point for the algorithm.
- **`fn`**: the objective function. Important is that the first argument of the objective includes those parameters that will be minimized. Hence, the argument has to be a matrix/vector.
- **`method`**: the function `optimx()` offers different solving methods, amongst them are: a gradient free method ("Nelder Mead"), a gradient based method("CG") and a hessian method ("BFGS") based on Newton-Raphson.

With the help of `optimx()` we will apply now algorithms of the three main methods to minimize the Himmelblau's function given in Equation (10). By this, we hope to give a good impression of the great possibilities offered by `optimx()` as well as of its easy use.
The Himmelblau's function, which will be used in the example, is one of the multi modal functions that are used as benchmark function to test the performance of algorithms. Its plot is shown in Figure 6.

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \tag{10}$$

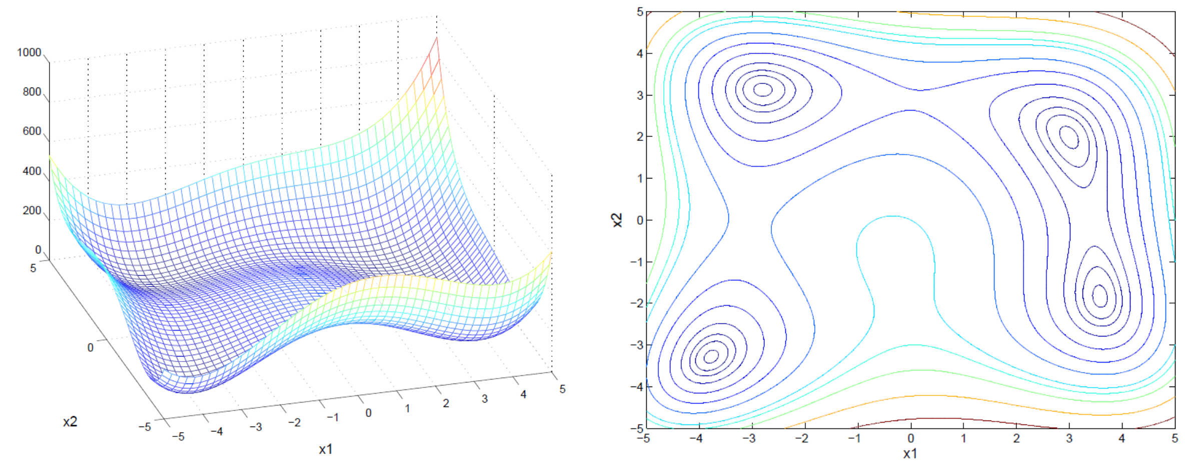

Figure 6: Plot of Himmelblau’s function (Zimmermann 2007).

The function has five local optima under which are four minima with an identical value. The minima are: $f(-3.7793; -3.2832) = 0$; $f(-2.8051;\ 3.1313) = 0$; $f(3;\ 2) = 0$ and $f(3,5844; -1,8481) = 0$. They can be found analytically (Zimmermann 2007).

```
> library(optimx)
> fn<- function(matrix.A){ # The first argument has to be a matrix of the parame-
ters
+   matrix.A <- matrix(matrix.A, ncol=2) # One column per parameter, one row per
point
+   x<- matrix.A [,1]
+   y<- matrix.A [,2]
+   f.x <- (x^2+y-11)^2+(x+y^2-7)^2
+   return(f.x)
+ }
> par<- c(1,1)
> sol.nelder.mead <- optimx(par, fn, method = "Nelder-Mead")
> sol.CG <- optimx(par, fn, method = "CG")
> sol.BFGS <- optimx(par, fn, method = "BFGS")
> sol.nelder.mead #solution Nelder-Mead method
                  p1       p2       value fevals gevals niter convcode  kkt1 kkt2
Nelder-Mead 2.999995 2.000183 5.56163e-07     67     NA    NA        0 FALSE TRUE
            xtimes
Nelder-Mead   0.02
> sol.CG #solution conjugate gradient method
   p1 p2        value fevals gevals niter convcode kkt1 kkt2 xtimes
CG  3  2 1.081231e-12    119     31    NA        0 TRUE TRUE      0
> sol.BFGS # solution Broyden-Fletcher-Goldfarb-Shanno method
     p1 p2        value fevals gevals niter convcode kkt1 kkt2 xtimes
BFGS  3  2 1.354193e-12     32     11    NA        0 TRUE TRUE      0
```

Box 6: “R” code snippet for the minimization of Himmelblau’s function with help of the methods ”Nelder-Mead”, “CG”, “BFGS”, integrated in `optimx()`.

Box 6 shows the application of the three above-mentioned methods, which are integrated in `optimx()`. For applying the different methods only the method argument has to be changed. In addition to the solution for the parameter levels and the function value, the output shows the value `fevals`, which indicates how often the function `fn` was called during the computation and gives so the number of needed iterations. The value `gevals` indicates the number of calculated gradients and `xtimes` puts out the required time for the calculation. With Nelder-Mead,

the values $x^* = 2.999995\,; y^* = 2.000183$ with $f(x^*, y^*) = 5.56163 \cdot 10^{-7}$ were found in 67 iterations steps and 2 seconds.
The conjugate gradient method found the exact values of one optimum of the Himmelblau’s function with $x^* = 3; y^* = 2$ and $f(x^*, y^*) = 1.081231 \cdot 10^{-12}$. It needed 119 iterations and less than 1 second. The Newton based method, here the Broyden-Fletcher–Goldfarb-Shanno method, also found the exact solution for one optimum but only needed 32 iterations. Therefore, it seems to be the most appropriate, out of the three methods, for the minimization of the Himmelblau’s function. This is not surprising because it is obvious that the two derivations of the function can be built with little cost.
Starting from the same starting point $x = 1; y = 1$, all methods found, at least approximately, the same optimum in $x^* = 3; y^* = 2$ and only this one and not the other three local minima of the Himmelblau’s function.
Nor such cases of multiple optima or global optimization – more complex problems in general – neither more specific solving algorithms were treated in this paper because this would go beyond the scope of this introduction.

7. Conclusion

Optimization is omnipresent and a key technology for statistics, mathematics and economics. Every person who wants to minimize costs or maximize profit is faced with this topic. The most essential requirements for optimization are: the knowledge of the problem and the knowledge of an appropriate solving method. In this introduction, first the most important classification criteria for optimization problems were discussed. Then, linear, quadratic and non-linear optimization problems were presented together with the adequate solving functions in “R”. This introduction concentrated on really simple and common problems and only described the skills of the “R” functions that are necessary for solving them. Of course, there is a much greater variety of optimization problems and solving methods and also “R” offers far more possibilities than were presented here. However, with the few “R” functions presented in this introduction already a lot of optimization problems can be solved in a really easy manner.

8. References

Berkelaar, M. et al. (2014): lpSolve: Interface to Lp_solve v.5.5 to solve linear/integer programs. R package, version 5.6.9, 2014.

Braun, W.J (2013): Numerical optimization. Lecture Notes, 2013. Available online at http://www.stats.uwo.ca/faculty/braun/ss2864/notes/ch7.pdf.

Braun, W.J; Murdoch, J. D. (2007): A First Course in Statistical Programming with R. Cambridge, New York, e.a.: Cambridge University Press.

Bravo, H. C. (2013): R interface to CPLEX. R package, version 0.3-1., 2013.

Gaile, S. (2008): Übung zur Vorlesung Optimierung I. Universität Erlangen, 2008.

Geiger, C.; Kanzow, C. (1999): Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben. Berlin, Heidelberg, New York: Springer Verlag.

Glass, C. W. (2014): Modellierung,Simulation,Optimierung-Vorlesungsunterlagen am Institut für Höchstleistungsrechnen der Universität Stuttgart. Stuttgart, 2014. Available online at http://www.ihr.uni-stuttgart.de/fileadmin/user_upload/teaching/vorlesungsstoff/Modellierung_Simulation_Optimierungsverfahren/Vorlesung/2013_2014/MSO1_V11_Optimierung1.pdf, checked on 6/29/2014.

Gould, N. (2006): An introduction to algorithms for continuous optimization. Oxford University. Oxford, 2006.

Graichen, K. (2012): Methoden der Optimierung und optimalen Steuerung. Lecture Notes, 2012.

Henningsen, A. (2012): Linear Programming / Optimization. R package, version 0.9-2, 2012.

Nagesh Kumar, D.(2014): Optimization Methods: Linear Programming- Simplex Method. Available online at http://nptel.ac.in/courses/Webcourse-contents/IISc-BANG/OPTIMIZATION%20METHODS/pdf/Module_3/M3L3_LN.pdf, checked on 6/29/2014.

Nash, J.C; Varadhan, R.; Grothendieck, G. (2013): optimx:A Replacement and Extension of the optim function. R package, version 2013.8.6, 2013.

National Computational Infrastructure (2014): Optimization Toolbox. Available online at http://nf.nci.org.au/facilities/software/Matlab/toolbox/optim/tutori6d.html, checked on 6/29/2014.

R Core Team (2014): R: A language and environment for statistical computing. R Foundation for Statistical Computing, 2014. Available online at http://www.R-project.org/.

Sauer, T. (2003): Einführung in die Optimierung für Hörer aller Fachbereiche. Gießen, 2003. Available online at www.staff.uni-giessen.de/tomas.sauer/Skripten/HaFOptimierung.pdf, checked on 6/29/2014.

Theußl, S. (2011): Many Solvers, One Interface. ROI, R Optimization Infrastructure, 2011. Available online at http://statmath.wu.ac.at/courses/optimization/Presentations/ROI-2011.pdf, checked on 6/30/2014.

Theußl, S. (2014): CRAN Task View: Optimization and mathematical programming, 2014. Available online at cran.r-project.org/web/views/Optimization.html, checked on 6/30/2014.

Turlach, B. A.; Weingessel, A. (2013): Quadprog: Functions to solve Quadratic Programming Problems. R package, version 1.5-5, 2013.

Verschelde, J. (2005): The Golden Section Search method. Lecture Notes, 2005. Available online at http://homepages.math.uic.edu/~jan/mcs471f03/Lec9/lec9.html, checked on 6/30/2014.

Zimmermann, U. (2007): Konvexe und Diskrete Optimierung. Die Funktion von Himmelblau. Lecture Notes, 2007.

Zoonekynd, V. (2012): Vincent Zoonekynd's Blog. Optimization, 2012. Available online at http://zoonek.free.fr/blosxom/2012/06/01/.