ROI: The R Optimization Infrastructure Package

Stefan Theußl, Florian Schwendinger, Kurt Hornik

Research Report Series
Report 133, November 2017

Institute for Statistics and Mathematics
http://statmath.wu.ac.at/

ROI: The R Optimization Infrastructure Package

Stefan Theußl
Raiffeisen Bank International AG

Florian Schwendinger
WU (Wirtschafts-universität Wien)

Kurt Hornik
WU (Wirtschafts-universität Wien)

Abstract

Optimization plays an important role in many methods routinely used in statistics, machine learning and data science. Often, implementations of these methods rely on highly specialized (non-reusable) optimization algorithms. However, in many instances recent advances, in particular in the field of convex optimization, make it possible to conveniently and straightforwardly use modern solvers (reusable) instead with the advantage of enabling broader usage scenarios and thus promoting reusability. This paper introduces the R Optimization Infrastructure package which provides an extensible infrastructure to model linear, quadratic, conic and general nonlinear optimization problems in a consistent way. Furthermore, the infrastructure administers many different solvers, reformulations, problem collections and functions to read and write optimization problems in various formats.

Keywords: optimization, mathematical programming, linear programming, quadratic programming, convex programming, nonlinear programming, mixed integer programming, R.

1. Introduction

Optimization is at the core of inference in modern statistics since solving statistical inference problems goes hand in hand with solving optimization problems (OPs). As such statisticians, data scientists, and others who regularly employ computational methods ranging from various types of regression (e.g., constrained least squares, regularized least squares, nonlinear least squares), and classification (e.g., support vector machines, convex clustering) to covariance estimation and low rank approximations (e.g., multidimensional scaling, non-negative matrix factorization) benefit from advances in optimization, in particular in mixed integer and convex optimization. For example, Bertsimas, King, and Mazumder (2016) show that, due to a striking speedup factor of 450 billion in mixed integer optimization in the period of 1991-2015, the NP-hard best subset problem (Miller 2002) can now be solved reasonably fast (number of observations in the 100s and number of variables in the 1000s is solved within minutes). O'Donoghue, Chu, Parikh, and Boyd (2016) introduce the **SCS** solver for convex optimization problems, which can be used to solve among others (logistic) regression with $l_{\{1,2\}}$- regularization, support vector machines, convex clustering, non-negative matrix factorization and graphical lasso.

For R (R Core Team 2017), being a general-purpose tool for scientific computing and data science, optimization and access to highly efficient solvers play an important role. The field of optimization has already many resources to offer, like software for modeling, solving and randomly generating optimization problems, as well as optimization problem collections used to benchmark optimization solvers. In order to exploit the available resources more conveniently, over the years many modeling tools have emerged. One of the first systems used to model linear optimization problems is the so-called Mathematical Programming System (MPS) format (see Kallrath 2004). Developed in the 1960's, the MPS format today seems rather archaic but it is still widely used to store and exchange linear problems and is supported by most of the linear optimization solvers. Later, algebraic modeling languages (AML) (e.g., GAMS (Bisschop and Meeraus 1982) and AMPL (Fourer, Gay, and Kernighan 1989)) became available. AMLs are domain specific languages (DSL) dedicated to optimization. Today modern optimization systems are typically implemented in high-level programming languages like Julia (Bezanson, Edelman, Karpinski, and Shah 2017), MATLAB(The MathWorks Inc. 2017), Python (Python Software Foundation 2017) or R. Many of these systems are DSLs specially suited for convex optimization, such as **YALMIP** (Löfberg 2004) and **CVX** (Grant and Boyd 2014) in MATLAB, **CVXPY** (Diamond and Boyd 2016) and **CVXOPT** (Andersen, Dahl, and Vandenberghe 2016) in Python, **Convex.jl** (Udell, Mohan, Zeng, Hong, Diamond, and Boyd 2014) in Julia and **CVXR** (Fu 2017) in R. **JuMP** (Lubin and Dunning 2015) is a DSL implemented in Julia designed for mixed-integer programming. **pyOpt** (Perez, Jansen, and Martins 2012) is a Python package for nonlinear constrained optimization.

Despite R having access to many modern optimization solvers which are capable of solving a wide class of optimization problems (see e.g., the CRAN optimization and mathematical programming task view by Theußl and Borchers 2017), it is still commonplace to develop highly sophisticated special purpose code (SPC) for many statistical problems. The reasons are manyfold. To name but a few: 1) *efficiency*, i.e., SPC tends to be faster, 2) *reusability*, i.e., many solvers have not been easily reusable (at least without the knowledge of a DSL) and 3) *capability*, i.e., problems could not be solved due to a lack of adequate solvers.

This paper introduces the R Optimization Infrastructure (**ROI**), which is composed of package **ROI** (Theußl, Schwendinger, Hornik, and Meyer 2017) and its (at the time of this writing) 22 companion packages. The companion packages equip **ROI** with state of the art optimization solvers, benchmark collections and functions to read and write optimization problems in various formats (increase capability). In contrast to DSLs, the **ROI** package does not aim to create a new language but provides a modeling mechanism borrowing its strength from the rich language features R has to offer (facilitate reusability). Another key feature of **ROI** is that it is designed to be extensible, thus allowing package developers to *plug-in* new solvers and make use of their highly efficient code (eliminate efficiency detriments). Already a wide range of suitable solvers exist for which interfaces have been written and they are typically made available via so-called plug-ins. Currently **ROI** can be used to model and solve linear, quadratic, second order cone, semidefinite, exponential cone, power cone and general nonlinear optimization problems as well as mixed integer problems. This makes it applicable to many optimization problems encountered in statistics, machine learning and data science. Such problems are then formulated and manipulated by using provided R functions instead of special syntax from DSLs for which highly specialized knowledge would be required.

The remainder of this paper is organized as follows: In Section 2 we discuss the basic optimization problem classes, with a special focus on the newer developments in convex optimization. A survey of the R packages concerned with solving the problem classes introduced in Section 2 is given in Section 3. Section 4 shows how to formulate optimization problems with the **ROI** package. How to solve the previously formulated optimization problems is discussed in

Section 5. Section 6 is dedicated to the extension of **ROI**. Applications in the field of statistics are presented in Section 7. Section 8 concludes this paper.

2. Problem classes

Optimization is the process of allocating scarce resources to a feasible set of alternative solutions in order to minimize (or maximize) the overall outcome. Given a function $f_0 : \mathbb{R}^n \to \mathbb{R}$ and a set $\mathcal{C} \subset \mathbb{R}^n$ we are interested in finding an $x^* \in \mathbb{R}^n$ that solves

$$\begin{array}{rl} \text{minimize} & f_0(x) \\ \text{subject to} & x \in \mathcal{C}. \end{array} \tag{1}$$

The function f_0 is called the *objective function*. A point x is said to be feasible if it satisfies every constraint given by the set $\mathcal{C}$ of all feasible points defining the *feasible region*. If $\mathcal{C}$ is empty then we say that the optimization problem is *infeasible*. Since maximization problems can be expressed as minimization problems by just changing the sign in the objective function, we will mainly deal with minimization problems subsequently.

An OP can be *bounded* or *unbounded*. For the latter, the value of the objective for a given sequence $x^j \in \mathcal{C}$ tends to $-\infty$ in a minimization problem, symbolically $f_0(x^j) \to -\infty$ as $j \to +\infty$. Thus, a problem like in Equation 1 may or may not have a *solution*. If the problem is neither infeasible nor unbounded then we can often find a vector $x^* \in \mathcal{C}$ that satisfies

$$f_0(x^*) \leq f_0(x), \ \forall x \in \mathcal{C},$$

which is commonly referred to as a solution of the OP.

Since any feasible set $\mathcal{C}$ can be expressed by the combination of constraint functions, the OP from Equation 1 can be written as:

$$\begin{array}{rll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq b_i, & i = 1, \ldots, m, \end{array} \tag{2}$$

where $b \in \mathbb{R}^m$ is the so-called right-hand-side. The constraints $f_i,\ i = 1, \ldots, m$ are sometimes referred to as *functional constraints* (Ben-Tal and Nemirovski 2001; Nesterov 2004). Since any equality constraint can be expressed by two inequality constraints and vice versa any inequality constraint can be expressed as an equality constraint by adding additional variables (also called *slack variables*), it is common practice to define OPs only in terms of either equality, less than or equal or greater than or equal constraints, to avoid redundancies.

Equation 2 is also sometimes referred to as the *primal problem*, which highlights the fact that there exists an alternative problem formulation the *dual problem*. The dual problem is typically defined via the Lagrangian function (Lagrange duality) (Nocedal and Wright 2006).

There are several interconnected characteristics which determine how efficiently a given OP can be solved, namely convexity, the functional form of the objective, the functional form of the constraints and if the variable x is binary, integer, or continuous. An OP as displayed in Equation 1 is convex, if f_0 is convex and the set $\mathcal{C}$ is convex. Whereas modern solvers can efficiently solve a wide range of convex OPs and verify that a global solution was obtained, the same is mostly not true for non-convex problems. More information about convex programming can be found in, e.g., Boyd and Vandenberghe (2004); Ben-Tal and Nemirovski (2015).

Based on the functional form of the objective function and of the constraints, OPs can be divided into linear and nonlinear OPs. Thereby, the class of nonlinear OPs can again be subdivided into conic, quadratic and general nonlinear OPs. In the following we give a formal definition of the different classes of OPs and some information about their properties.

2.1. Linear programming

Starting from Equation 2, a linear program (LP) is an OP where all f_i $(i = 0, \dots, m)$ are linear. Thus an LP can be defined as:

$$\begin{array}{rl} \text{minimize} & a_0^\top x \\ \text{subject to} & Ax \leq b \end{array} \tag{3}$$

where x is the vector of objective variables which has to be optimized. The coefficients of the objective function are represented by $a_0 \in \mathbb{R}^n$. $A \in \mathbb{R}^{m \times n}$ is a matrix of coefficients representing the constraints of the LP, hence in accordance with Equation 2 $Ax \leq b$ could also be written as $a_i^\top x \leq b_i$, $i = 1, \dots, m$ (here a_i refers to the $i-$th row of the coefficient matrix A). All LPs are convex and normally solved via interior-point or simplex methods. For more information about the origination and mathematical properties of these methods we refer the reader to the book of Nocedal and Wright (2006).

2.2. Quadratic programming

A quadratic program (QP) is a generalization of the standard LP shown in Equation 3, where the objective function contains a quadratic part in addition to the linear term. The quadratic part is typically represented by a matrix $Q_0 \in \mathbb{R}^{n \times n}$. Therefore QPs can be expressed in the following manner:

$$\begin{array}{rl} \text{minimize} & \frac{1}{2} x^\top Q_0 x + a_0^\top x \\ \text{subject to} & Ax \leq b. \end{array} \tag{4}$$

Unlike LPs not all QPs are convex. A QP is convex if and only if Q_0 is positive semidefinite. A generalization of the QP is the quadratically constrained quadratic program (QCQP):

$$\begin{array}{rl} \text{minimize} & \frac{1}{2} x^\top Q_0 x + a_0^\top x \\ \text{subject to} & \frac{1}{2} x^\top Q_i x + a_i^\top x \leq b_i, \quad i = 1, \dots, m. \end{array} \tag{5}$$

A QCQP is convex if and only if all Q_i $(i = 0, \dots, m)$ are positive semidefinite (Lobo, Vandenberghe, Boyd, and Lebret 1998). Whereas convex QP or even QCQP are commonly solved by reformulations (transformations) to second-order cone programming (SOCP) or semidefinite programming (SDP) (see Section 2.3), the question how to obtain a reliable global solution for non-convex QCQP is still an active field of research. Details on the necessary transformations to cast convex QCQP into an SOCP or SDP can be found in, e.g., Lobo *et al.* (1998); Alizadeh and Goldfarb (2003); Bao, Sahinidis, and Tawarmalani (2011).

2.3. Conic programming

Conic programming (CP) refers to a class of problems designed to model convex OPs. The most prominent members of this class are LP, SOCP and SDP. In the following we will define a CP as:

$$\begin{array}{ll} \text{minimize} & a_0^\top x \\ \text{subject to} & Ax + s = b \\ & s \in \mathcal{K}, \end{array} \tag{6}$$

where the set $\mathcal{K}$ is a nonempty closed convex cone, often constructed by the Cartesian product of simpler cones $\mathcal{K} = \prod K_i$. Every cone $\mathcal{K}$ has a dual cone $\mathcal{K}^* = \{y | x^\top y \geq 0 \text{ for all } x \in \mathcal{K}\}$ for more information about the dual cone we refer the interested reader to Boyd and Vandenberghe (2004).

The standard form of CP as given in Equation 6 minimizes a linear objective over a convex cone ($b - Ax = s \in \mathcal{K}$). As Nemirovski (2006) points out, representing CPs in this form has two main advantages. First, this formulation has strong unifying abilities which means only a few cones allow to model many different types of OPs. Additionally the nonlinearities are no longer represented by general nonlinear objective and constraint functions but vectors and matrices which allows the algorithms to utilize the structure present in the convex OPs. Second, the convexity is built-in into the definition of CPs. At the same time, theoretically, any convex OP can be reformulated into the form given in Equation 6. Thereby nonlinear objective functions are expressed in *epigraph form* (see e.g., Boyd and Vandenberghe 2004):

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & f_0(x) \leq t \\ & f_i(x) \leq b_i. \end{array} \tag{7}$$

Practically the amount of OPs which can be solved via CP is limited by the number of cones supported by a given optimization solver. State of the art solvers distinguish between up to eight different types of cones. Following the definitions in Diamond and Boyd (2015) and O'Donoghue *et al.* (2016), a convex cone $\mathcal{K}$ is typically a Cartesian product from simple convex cones of the following types.

Free cone

From Equation 6 it can be immediately seen, that in the case of linear equality constraints s_i has to be zero, i.e., $s_i \in \mathcal{K}_{\text{zero}}$. Where the zero cone is defined as

$$\mathcal{K}_{\text{zero}} = \{0\} \tag{8}$$

therefore its dual, the free cone is defined as

$$\mathcal{K}_{\text{free}} = \mathbb{R}. \tag{9}$$

Nonnegative cone

Linear inequality (less than or equal) constraints are represented by requiring s_i to be non-negative, i.e., $s_i \in \mathcal{K}_{\text{nneg}}$.

$$\mathcal{K}_{\text{nneg}} = \{x \in \mathbb{R} \mid x \geq 0\} \tag{10}$$

From the definition of the free cone and nonnegative cone, it is apparent that any LP can be written as a CP where $\mathcal{K}$ is a product of free and nonnegative cones.

Second-order cone

$$\mathcal{K}_{\text{soc}}^n = \{(t,x) \in \mathbb{R}^n \mid x \in \mathbb{R}^{n-1},\ t \in \mathbb{R},\ ||x||_2 \leq t\} \tag{11}$$

The second-order cone is commonly used to model sums of norms as well as convex QP and QCQP (Lobo *et al.* 1998; Alizadeh and Goldfarb 2003). CPs where $\mathcal{K}$ is a product of free, nonnegative and second-order cones are commonly referred to as SOCP.

Positive semidefinite cone

$$\mathcal{K}_{\text{psd}}^n = \{X \mid X \in \mathcal{S}^n,\ z^\top X z \geq 0 \text{ for all } z \in \mathbb{R}^n\} \tag{12}$$

Here $\mathcal{S}^n$ refers to the space of real-symmetric $n \times n$ matrices. SDPs are commonly used for solving combinatorial problems (e.g., maximum cut problem) and for solving convex QPs and QCQPs (Vandenberghe and Boyd 1996a; Helmberg 2000; Freund 2009; Bao *et al.* 2011). Lobo *et al.* (1998) show that each SOCP can be rewritten into a SDP.

Exponential cone

The primal exponential cone can be defined as

$$\mathcal{K}_{\text{expp}} \quad = \quad \{(x,y,z) \in \mathbb{R}^3 \mid y > 0,\ ye^{\frac{x}{y}} \leq z\} \cup \{(x,0,z) \in \mathbb{R}^3 \mid x \leq 0,\ z \geq 0\}, \tag{13}$$

therefore the dual exponential cone is given by

$$\mathcal{K}_{\text{expd}} \quad = \quad \{(u,v,w) \in \mathbb{R}^3 \mid u < 0, -ue^{\frac{v}{u}} \leq ew\} \cup \{(0,v,w) \in \mathbb{R}^3 \mid v,\ y \geq 0\}. \tag{14}$$

As can be inferred from Equation 13, the exponential cone can be used to model exponential functions and logarithms. More details about the exponential cone and functions representable by the exponential cone can be found in Chares (2009) and Serrano (2015).

Power cone

The 3-dimensional primal power cone has been already investigated in Koecher (1957) and is defined as

$$\mathcal{K}_{\text{powp}}^\alpha = \{(x,y,z) \in \mathbb{R}^3 \mid x,\ y \geq 0,\ x^\alpha y^{1-\alpha} \geq |z|\},\ \text{where } \alpha \in [0,1], \tag{15}$$

therefore the dual power cone is given by

$$\mathcal{K}_{\text{powd}}^\alpha = \left\{(u,v,w) \in \mathbb{R}^3 \mid u,\ v \geq 0,\ \left(\frac{u}{\alpha}\right)^\alpha \left(\frac{v}{1-a}\right)^{1-\alpha} \geq |w|\right\},\ \text{where } \alpha \in [0,1]. \tag{16}$$

The power cone can be used to model powers and p-norms. For more information about the power cone and its modeling capabilities we refer to Chares (2009).

Putting the hierarchies described above all together we get the following ordering among OPs

$$\text{LP} \subset \text{convex QP} \subset \text{convex QCQP} \subset \text{SOCP} \subset \text{SDP} \subset \text{CP}.$$

2.4. Nonlinear optimization

The most general problem class is nonlinear optimization or nonlinear programming (NLP). Considering Equation 2, this is the problem where at least one f_i, $i = 0, \ldots, m$ is not linear. NLPs are not required to be convex, which makes it in general hard to obtain a reliable global solution. Contrary to the convex case, in a non-convex setting most optimization algorithms only find the extrema of f_0 in the neighborhood of the starting value (local optimum).

2.5. Mixed integer programming

A mixed integer program (MIP) adds additional requirements to the optimization problem, namely it requires that some of the objective variables can only take integer values. Considering Equation 2, a problem is called a mixed integer problem if the (type) constraint $x_k \in \mathbb{Z}$ for at least one k is added. In the case where all n objective variables are integral we speak of a pure integer programming (IP) problem. An IP where all variables are bounded between zero and one, i.e., $x \in \{0, 1\}^n$, is called a binary (integer) program.

Since MIPs are non-convex, even mixed integer linear programs (MILP) are in general NP-Hard. Typically they are solved via branch-and-bound (Land and Doig 1960) and the cutting plane (Gomory 1960) algorithms or a combination of both. Both algorithms avoid to solve the problem directly, but instead solve multiple relaxations where the integrality constraint is dropped. MILP problems are known to be very difficult to solve, nevertheless an increase in quantity and quality of free and nonfree solvers was observed in the last decade (Linderoth and Ralphs 2005; Bixby 2012).

3. Software

Recently, an increase of the available packages handling many different OPs in R could be observed. The CRAN optimization and mathematical programming task view (Theußl and Borchers 2017) currently lists around 100 different optimization related packages. These packages reach from solvers which can solve a wide range of optimization problems (e.g., **Rcplex** (Theußl and Bravo 2016), **Rmosek** (Friberg 2014), **optimx** (Nash and Varadhan 2011; Nash 2014a)) to very specialized solvers which are created to solve a specific problem type very fast. This section provides an overview of the solver landscape in R. The insights gained in this section will be used to derive a consistent solver infrastructure. First we divide the solver landscape into commercial and non-commercial solvers. Second, in accordance with Section 2, we split the group of non-commercial solvers into linear solvers, quadratic solvers, conic solvers and general purpose solvers.

3.1. Overview

As pointed out in Section 2, in the field of optimization we are typically facing different problem classes. The possibly three most important distinctions are between linear versus nonlinear problems, integer versus continuous and convex versus non-convex problems.

Ordered based on increasing complexity, an objective function might be of type linear, (convex) quadratic, conic (i.e., any objective expressible as a CP) or functional (i.e., any objective expressible as function). Similarly constraints are typically of type box, linear, (convex) quadratic, conic or functional. Box constraints (or variable bounds) are a special type of lin-

	Objective			
Constraints	linear	quadratic	conic	functional
no				02, 33
box				07, 08, 10, 13, 16, 19, 22 25, 26, 29, 31, 32, 34
linear	$\textcircled{04}$, $\textcircled{\boxed{11}}$, $\textcircled{\boxed{15}}$ $\textcircled{21}$, $\textcircled{\boxed{28}}$	12, $\textcircled{14}$, $\textcircled{20}$		
quadratic				
conic			$\textcircled{03}$, $\textcircled{05}$, $\textcircled{\boxed{09}}$ $\textcircled{23}$, $\textcircled{24}$, $\textcircled{30}$	
functional				01, 06, 17 18, 27

Table 1: Overview on optimization problems and solvers.[1]

ear constraints which enforce lower and upper bounds on the objective variables. The terms conic objective/constraints are used in a general way and refer to any linear and nonlinear objective/constraints that can be reformulated as a conic problem. Therefore this also includes problems with linear and convex quadratic objective/constraints. The most general form are functional objective/constraints which includes all linear and nonlinear objective/constraints.

Table 1 gives an overview on which solver can be used to solve which types of OPs. Solvers allowing mixed integer constraints are highlighted with a rectangular frame (e.g., $\boxed{00}$) and solvers restricted to convex problems are highlighted with a circular frame (e.g., $\textcircled{00}$). Consequently solvers which are restricted to convex problems and can handle integer constraints are marked with both a circular and rectangular frame (e.g., $\textcircled{\boxed{00}}$). For completeness, we note that the linear solvers are also marked to be restricted to convex problems although this is no restriction since all linear problems are convex.

Therefore the position and the frame of a particular solver in the table indicates its ability to solve a given problem. Each problem class to the left and above of the current position can be handled by the solver including its current position. For instance the **ECOS** (Domahidi, Chu, and Boyd 2013) solver provided in package **ECOSolveR** (Fu and Narasimhan 2017, $\textcircled{\boxed{09}}$) is a convex optimization solver, which can solve conic problems restricted to combinations of the zero, nonnegative, second-oder and primal exponential cone. Since **ECOS** is equipped with a branch-and-bound algorithm, it can also be used to solve mixed integer conic problems.

[1]The corresponding identifiers can be found in Table 5 in Appendix A.

3.2. Commercial solvers

Since commercial solver packages often bundle a variety of solvers, it is often not possible to assign them to a certain problem class, therefore we will treat them separately. At the time of this writing R interfaces are available to the commercial solver software **CPLEX** (ILOG 2015), **MOSEK** (ApS 2017), **Gurobi** (Gurobi Optimization 2016), **Lindo** (Lindo Systems 2003) and **localsolver** (Benoist, Estellon, Gardi, Megel, and Nouioua 2011).

3.3. Non-commercial solvers

The non-commercial solvers landscape can be split into two parts. First, solvers where the functional form is fixed and only the coefficients are provided, this includes all LP, QP, QCQP and CP solvers currently available in R. Second, solvers which can optimize any functional form expressible as an R function. This includes most NLP solvers, sometimes summarized as general purpose solvers.

Linear solvers

Interfaces to several open source LP and MILP solvers are available in R. Most of these packages provide a high-level access to the solver, those explicitly designed to provide a low-level access are commonly marked with the suffix **API**.

The Computational Infrastructure for Operations Research (COIN-OR) project (`https://www.coin-or.org/`) provides open-source software for the operations research community. Among this software there are the COIN-OR linear programming (**Clp**, Forrest, de la Nuez, and Lougee-Heimer 2004) solver and the **SYMPHONY** (Ralphs and Güzelsoy 2005, 2011) solver. **Clp** is mainly used as library and provides methods for solving LPs via interior point methods or the simplex algorithm. In R **Clp** is available through **clpAPI** (Fritzemeier and Gelius-Dietrich 2016) which provides a low level interface to **Clp**. **SYMPHONY** is a flexible MILP solver written in C++, that transforms the MILP into LP relaxations to be solved by any LP solver callable through the Open Solver Interface (OSI). **Rsymphony** (Hornik, Harter, and Theußl 2017a) provides an interface to the **SYMPHONY** solver, where by default the LP relaxations are solved by the **Clp** solver.

GNU Linear Programming Kit (**GLPK**, Makhorin 2011) is a solver library written in ANSI C, for solving LP and MILP. In R the low level interface **glpkAPI** (Fritzemeier, Gelius-Dietrich, and Luangkesorn 2015) and the high level interface **Rglpk** (Theußl and Hornik 2017) are available.

lp_solve (Berkelaar, Eikland, and Notebaert 2016) uses the simplex algorithm combined with branch-and-bound to solve LPs and MILPs. It furthermore allows to model semi-continuous and special ordered sets problems. Packages **lpSolve** (Berkelaar 2015) and **lpSolveAPI** (Konis 2016) provide access to the **lp_solve** solver in R.

Additionally the function `lpcdd()` from package **rcdd** (Geyer and Meeden 2017) and the function `simplex()` from package **boot** (Canty and Ripley 2017) can be used to solve LPs via the simplex algorithm.

By taking a closer look at the elements needed by packages capable of solving LPs and MILPs[2] we can conclude that the following elements should be present in a consistent and convenient

[2]This includes commercial and non-commercial solvers.

optimization infrastructure for modeling LPs and MILPs.

`objective`: A numeric vector giving the coefficients of the linear objective.

`constraints`:

- A constraint matrix A (see Equation 3).
- A vector giving the direction of the constraints (i.e., `==`, `<=` or `>=`).
- A vector giving the right hand side b (see Equation 3).

`bounds`: Two vectors giving the lower and upper bounds.

`types`: A vector storing the type information, i.e., binary, integer and numeric.

`maximum`: A boolean indicating if the objective function should be maximized or minimized.

Although the elements `bounds` and `maximum`, as well as the constraint directions and the binary types are not strictly necessary. Their inclusion is motivated by the fact that they are supported by many solvers and simplify the problem specification.

Quadratic solvers

At the time of this writing there exist two non-commercial packages specialized on solving quadratic OPs in R, namely **quadprog** and **LowRankQP**. Both are able to solve stricly convex QPs but can not be used to solve QCQPs. The **quadprog** (Turlach and Weingessel 2013) package uses the dual method described in Goldfarb and Idnani (1983), whereas **LowRankQP** (Ormerod and Wand 2014) is based on an interior point algorithm described in Fine and Scheinberg (2001).

Additionally, package **kernlab** (Karatzoglou, Smola, Hornik, and Zeileis 2004; Karatzoglou, Smola, and Hornik 2016) contains the function `ipop()` which implements an interior point solver capable of solving QPs.

QP solver generally take the same arguments as LP solver plus an additional matrix parameter storing the coefficients of the quadratic term Q_0.

Conic solvers

Most of the conic solvers use a standard form similar to Equation 6, where the objective function is assumed to be linear and the vector $b - Ax$ is restricted to a certain cone $\mathcal{K}$. Nevertheless, in Table 1 they are shown to have a conic objective function and conic constraints to express that they are able to solve any LP and convex NLP expressible by a CP. Therefore, which types of NLPs a given solver can solve, depends on the types of cones the solver can model. Table 2 shows the conic solvers available in R and the types of cones they support.

Package **CLSOCP** (Rudy 2011) is specialized in solving SOCPs, it is a pure R implementation of the one-step smoothing Newton method based on the algorithm described in Tang, He, Dong, and Fang (2012). For solving SDP there exist the specialized packages **Rcsdp** and **Rdsdp**. Since any SOCP can be transformed into an SDP they can also be used for solving SOCPs. **Rcsdp** (Bravo 2016) is an interface to the **CSDP** (Borchers 1999) library which is part of the COIN-OR project. **Rdsdp** (Zhisu Zhu 2016) is an interface to the **DSDP** (Benson

	zero	linear	second order	semidefinite	exponential	dual exponential	power	dual power
CLSOCP	✓	✓	✓					
cccp	✓	✓	✓	✓				
ECOSolveR	✓	✓	✓		✓			
Rcsdp	✓	✓	✓	✓				
Rdsdp	✓	✓	✓	✓				
scs	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Conic packages and the supported cones.

and Ye 2008) library. Both packages can read and **Rcsdp** can also write `"sdpa"`-files, which is a file format commonly used to store SDPs. The **cccp** (Pfaff 2015) package provides functions to solve LPs, QPs, SOCPs and SDPs, the algorithms are reported to be similar to those in **CVXOPT** (Andersen *et al.* 2016). **CVXOPT** is a Python package for solving convex OPs via interior-point methods (more information about the algorithms can be found in Andersen, Dahl, Liu, and Vandenberghe 2012). **ECOSolveR** (Fu and Narasimhan 2017) is an interface to the embedded conic solver **ECOS** (Domahidi *et al.* 2013). A special feature of **ECOS** is that it combines convex optimization with branch-and-bound techniques, therefore it can be used to solve CPs where some variables are required to be integer. The **scs** (O'Donoghue and Schwendinger 2016) package is an interface to the Splitting Conic Solver (**SCS**, O'Donoghue 2015) library, which uses a version of the alternating direction method of multipliers (ADMM) for solving CPs. **SCS** is designed to solve large cone problems faster than standard interior-point methods. More information about the algorithm and a comparison to other solvers can be found in O'Donoghue *et al.* (2016).

General purpose solvers

Solvers capable of handling nonlinear objective functions without further restrictions are called general purpose solvers (GPS). These solvers can minimize (or maximize) any functional form representable as an R function with different types of constraints.

Dependent on the solver different types of constraints can be used, where the most general form of constraint is the functional constraint (i.e., any constraint expressible as an R function). The generality of GPS comes at the price of performance and that there is usually no guarantee that a global optimum is reached.

	Global GPS		Local GPS	
	Gradient free	Gradient	Gradient free	Gradient
No Constraint	5	0	7	8
Box Constraint	16	4	7	10
Functional Constraint	2	0	7	7

Table 3: Overview of general purpose solvers.

Important properties of GPS are whether they are designed to search for a local or global optima, if gradient information has to be provided or the method is gradient free and which type of constraints can be set. Table 3 shows the number of GPS methods grouped by these properties (the counts are based on Table 6 where additional details can be found) and reveals some interesting details about the R GPS landscape. There exist almost twice as many GPS for local optimization than for global optimization and, even though most of the local solvers utilize gradient information, only four of the global solvers use gradient information. The difference in distribution of gradient based and gradient free optimization algorithms between global and local GPS can be explained by the fact that in global optimization, metaheuristics like evolutionary methods or particle swarm optimization are commonly used. In a recent study Mullen (2014a) surveys the continuous global optimization packages available in R and compares their performance on a set of tests bundled in the **globalOptTests** (Mullen 2014b) package. Table 6 gives an extensive listing of which methods are applicable to global OPs. For more information about the methods we refer to Mullen (2014a).

Based on the type of constraints, the GPS can be divided into *no constraints*, *box constraints*, *linear constraints*, *quadratic constraints* and *functional constraints*. As Table 3 shows, most of the GPS support no constraint or box constraints. Fortunately, package **optimx** (Nash and Varadhan 2011) provides a unified interface to many of these solvers, consolidating methods from packages **stats**, **ucminf** (Nielsen and Mortensen 2016), **minqa** (Bates, Mullen, Nash, and Varadhan 2014), **Rcgmin** (Nash 2014b), **Rvmmin** (Nash 2017) and **BB** (Varadhan and Gilbert 2009). It was designed as a possible successor of `optim` which is part of the **stats** package and can be used to solve OPs with box constraints. Another package which incorporates many different algorithms is **nloptr** (Ypma, Borchers, and Eddelbuettel 2017). It is an R interface to the **NLopt** (Johnson 2016) library, which bundles several global and local optimization algorithms. Depending on the algorithm it can solve NLPs with box-constraints or functional-constraints.

Most of the GPS able to handle functional constraints allow to specify functional equality and/or functional inequality constraints.

To model functional equality constraints the following two forms are most commonly used

- $h_i(x) = 0, \; i = 1, \ldots, k$ (e.g., **alabama** (Varadhan 2015), **DEoptimR** (Conceicao 2016), **nloptr::auglag**, **nloptr::isres**, **nloptr::slsqp**, **NlcOptim** and **Rnlminb2** (Wuertz 2014))
- $h_i(x) = b_i, \; i = 1, \ldots, k$ (e.g., **Rsolnp** (Ghalanos and Theussl 2015))

where h is a function and $b \in \mathbb{R}^k$ gives the right hand side. Similarly, functional inequality constraints are commonly given in one of the following three forms:

- $g_j(x) \leq 0, \; j = k+1, \ldots, m$ (e.g., **DEoptimR**, **nloptr::nloptr**, **NlcOptim** (Chen and Yin 2017), **Rnlminb2**, **csr::snomadr** (Racine and Nie 2017))
- $g_j(x) \geq 0, \; j = k+1, \ldots, m$ (e.g., **alabama**, **nloptr::auglang**, **nloptr::auglang**, **nloptr::cobyla**, **nloptr::ires**, **nloptr::mma**, **neldermead** (Bihorel and Baudin 2015) and **nloptr::slsqp**)
- $l_j \leq g_j(x) \leq u_j, \; j = k+1, \ldots, m$ (e.g., **ipoptr** (Ypma 2011), **Rdonlp2** (Wuertz 2007), **Rsolnp**).

where g is a function and $l \in \mathbb{R}^{m-k}$, $u \in \mathbb{R}^{m-k}$ are the lower and upper bounds of the constraints. A general optimization infrastructure should be designed in a way that the functional form employed can be transformed into the commonly used forms shown above.

An analysis of the above solver spectrum reveals that the critical arguments to GPS are:

`start`: The initial values for the (numeric) parameter vector.

`objective`: The function to be optimized.

`constraints`: Depending on the GPS the constraints can be, none, linear, quadratic, functional equality or functional inequality constraints. To model functional constraints consistently with linear and quadratic constraints the following elements are needed.

- A function representing the constraints.
- A vector giving the direction of the constraints.
- A vector giving the right hand side.

`bounds`: Variable bounds, commonly given as lower and upper bounds.

Additionally some GPS make use of the gradient and/or hessian of the objective and the jacobian of the constraints. The optional elements can be summarized by:

`gradient`: A function that evaluates the gradient of the argument `objective`.

`hessian`: A function that evaluates the hessian of the argument `objective`.

`jacobian`: A function that evaluates the jacobian of the argument `constraints`.

`maximum`: A boolean indicating whether the objective function should be maximized or minimized.

`control`: Further control arguments specific to the solver.

Return values include:

`par`: The “solution” (parameters) found.

`value/objective`: The value of the objective function evaluated at the “solution”.

`convergence, status`: An integer information about the convergence and exit status of the optimization task.

`gradient`: The gradient evaluated at the solution found.

`hessian`: The hessian evaluated at the solution found.

`message`: A text message giving additional information about the optimization / exit status.

`iterations/evaluations`: The number of iterations and / or evaluations.

4. A general optimization infrastructure for R

After reviewing the optimization resources available in R, it is apparent that the main function of a general optimization infrastructure package should take at least three arguments:

problem representing an object containing the description of the corresponding OP,

solver specifying the solver to be used (e.g., `"glpk"`, `"nlminb"`, `"scs"`),

control containing a list of additional control arguments to the corresponding solver.

The arguments `solver` and `control` are easily understood, since from the available solver spectrum we only have to choose those which are capable to handle the corresponding OP and (optionally) supply appropriate control parameters. However, building the `problem` object, in a general and intuitive way, seems to be a very challenging task which leads to several design issues.

Based on the review in Sections 2 and 3 it seems natural to instantiate OPs based on an *objective* function, one or several *constraints*, *types* and *bounds* of the objective variables, as well as the direction of optimization (whether a minimum or a *maximum* is sought).

4.1. Optimization problem

A new optimization problem is created by calling

```
OP(objective, constraints, types, bounds, maximum)
```

where the arguments are explained in detail below.

Alternatively, an OP can be formulated piece by piece, by creating an empty OP

```
OP()
```

and using the setter functions to assign the values. The setter and getter functions have the same names as the arguments of `OP` and can be used to manage specific parts of the OP. For instance, by replacing the linear objective of an LP with a quadratic objective the LP is altered into a QP. An extensive set of examples showing the creation and modification of OPs can be found at the end of this sections.

Function `OP()` always returns an S3 object of class `OP` which stores the entire OP. Storing the OP in a single R object has many advantages, among others:

- the OP can be checked for consistency during the creation of the problem,
- the different elements of the OP can easily accessed after the creation of the problem,
- and the OP can be easily altered, e.g., a constraint can be added, bounds can be changed, without the need to redefining the entire OP.

The consistency checks verify that the dimensions of the arguments fit together.

4.2. Objective function

The survey of optimization solvers in Section 3 reveals that the way the objective function is stored depends primarily on its functional form. If the objective function is linear (L), i.e., $a_0^\top x$, then it is common practice to only supply a coefficient vector $a_0 \in \mathbb{R}^n$. For quadratic objective functions (Q) of the form $\frac{1}{2}x^\top Q_0 x + a_0^\top x$ most solvers take a vector $a_0 \in \mathbb{R}^n$ and a matrix $Q_0 \in \mathbb{R}^{n \times n}$ as input. General nonlinear objective functions (i.e., nonlinear functions which can not be represented as an QP or CP), are represented as an R function (F) which takes the vector of objective variables as argument and returns the objective value. Depending on the type of the objective function, i.e., `F`, `Q`, or `L` only a subset of the solver spectrum can be used.

Objective function types and corresponding constructors implemented in **ROI** are:

F The most general form of an objective function is created with the `F_objective(F, n, G, H, names)` constructor by simply supplying `F`, an R function representing $f_0(x)$, and n the length of x. Optionally, information about the gradient and the hessian can be provided via the arguments `G` and `H`. Is no gradient provided it will be calculated numerically if needed. The optional `names` argument is propagated to the solution object to make the solution more readable.

Q Objective functions representing a quadratic form as outlined above can be easily created with the `Q_objective(Q, L, names)` constructor taking `Q`, the quadratic part Q_0, and optionally `L`, the linear part a_0, as arguments. The `names` argument is again optional.

L If the objective to be optimized is a linear function then one should use the `L_objective(L, names)` constructor supplying `L` (the coefficients to the objective variables) as a numeric vector. The `names` argument is again optional.

All three constructors return an object inheriting from class '`objective`'.

4.3. Constraints

To model all the problem classes introduced in Section 2 four different types of constraints are sufficient. Thereby arguments with the same name have the same functionality irrespective of the constraint type hence they are only explained once.

F The most general form of constraints can express any constraint representable by an R function. They are created via `F_constraint(F, dir, rhs, J, names)`, here `F` is either a function or a list of functions. `dir` is a character vector giving the direction of the constraint and `rhs` is a numeric vector giving the right hand side of the constraint. The optional arguments `J` and `names` can be used to provide the Jacobian and the variable names of the constraints.

C Conic constraints are constructed via the function `C_constraint(L, cones, rhs, names)`, thereby `L` can be either a numeric vector of length n or a matrix of dimension $m \times n$. In accordance with Equation 6 the `cones` impose a restriction on the slack variable s.

Q Quadratic constraints as defined in Equation 5 can be easily created with the constructor `Q_constraint(Q, L, dir, rhs, names)`. The quadratic constraints `Q` are given as a list of length m where the entries are either of $n \times n$ matrices or `NULL`.

L Linear constraints are constructed via the function `L_constraint(L, dir, rhs, names)`.

All constructors return an object inheriting from class '`constraint`'.

A conic constraint can be comprised of several cones, where each cone type can occur multiple times. The cone constructors all start with `K_` followed by an short cut of the cone name, as defined in Section 2.3. Currently **ROI** implements constructors for the cones `K_zero`, `K_nneg`, `K_soc`, `K_psd`, `K_expp`, `K_expd`, `K_powp`, and `K_powd`. To combine different cones the generic combine function `c()` can be used.

Since in many situations it is desirable to optimize a given objective function subject to a constraint object composed out of different constraints (which may be of different type), **ROI** can combine multiple constraints into a single constraint using the generic functions `c()` or `rbind()`. Therefore, the following constraints

$$\begin{array}{ccccc} L_{11}\ x & + & L_{12}\ y & = & \text{rhs}_1 \\ L_{21}\ x & + & L_{22}\ y & \leq & \text{rhs}_2 \\ L_{31}\ x^2 & + & L_{32}\ y^2 & \leq & \text{rhs}_3 \end{array} \tag{17}$$

could be formulated in several equivalent ways. First as combination of linear and quadratic constraints

```
R> library("ROI")
```

```
c(L_constraint(L = rbind(c(L11, L12), c(L21, L22)),
    dir = c("==", "<="), rhs = c(rhs1, rhs2)),
  Q_constraint(Q = rbind(c(2 * L31, 0), c(0, 2 * L32)), L = c(0, 0),
    dir = "<=", rhs = rhs3))
```

second as combination of linear and conic constraints

```
c(L_constraint(L = rbind(c(L11, L12), c(L21, L22)),
    dir = c("==", "<="), rhs = c(rhs1, rhs2)),
  C_constraint(L = rbind(c(0, 0), c(-L31, 0), c(0, -L32)),
    cones = K_soc(3), rhs = c(rhs3, 0, 0)))
```

or entirely as conic constraints.

```
C_constraint(L = rbind(c(L11, L12), c(L21, L22),
    c(0, 0), c(-L31, 0), c(0, -L32)),
  cones = c(K_zero(1), K_nneg(1), K_soc(3)),
  rhs = c(rhs1, rhs2, rhs3, 0, 0))
```

Additionally the constraints could also be formulated entirely as `Q_constraint` or `F_constraint` or other constraint combinations.

4.4. Objective variable types

As it is common practice in mixed-integer solvers to distinguish between the variable types continuous, integer and binary we follow this practice. To encode the variable choice characters are used, `"C"` for continuous, `"I"` for integer and `"B"` for binary, where by default all the variables are assumed to be of continuous type.

4.5. Bounds

Variable bounds are a special type of constraints typically used to restrict the objective variable between real lower and upper bounds, these are often referred to as "box bounds" or "box constraints". Although variable bounds could be easily modeled as constraints, most solvers which support any type of constraint also support variable bounds directly. Furthermore, many GPS only support variable bounds as can be seen in Table 3. Thus, it is reasonable but also convenient to consider them separately.

Typically, implementations of optimization algorithms differentiate between five types of objective variable bounds: free $(-\infty, \infty)$, upper $(-\infty, ub]$, lower $[lb, \infty)$, double bounded $[lb, ub]$, and fixed bounds. In **ROI** variable bounds are represented as a list with two elements—`upper` and `lower`, where only the non-default values are stored in a simple sparse format. In this sparse format only indices and the values of the non-default values are stored. For the lower bounds the default value is zero and for the upper bounds the default value is infinity. Thus for OPs where all the variables are required to take values in the interval $[0, \infty)$ no bounds have to be specified. Upper and/or lower bounds are specified by providing the index i of the corresponding variable (arguments `li`, `ui`) and its lower (`lb`) or upper (`ub`) bound, respectively. Therefore the box constraints $-\infty \leq x_1 \leq 4$, $0 \leq x_2 \leq 100$, $2 \leq x_3 \leq \infty$ and $0 \leq x_4 \leq \infty$ are constructed in **ROI** as follows,

```
R> V_bound(li = 1:4,  ui = 1:4, lb = c(-Inf, 0, 2, 0),
+    ub = c(4, 100, Inf, Inf))
```

```
ROI Variable Bounds:

2 lower and 2 upper non-standard variable bounds.
```

in the case all the upper and lower values are provided (default values are not omitted) the indices can be left out

```
R> V_bound(lb = c(-Inf, 0, 2, 0), ub = c(4, 100, Inf, Inf))
```

```
ROI Variable Bounds:

2 lower and 2 upper non-standard variable bounds.
```

in the case default values are omitted the number of objective variables has to be provided.

```
R> V_bound(li = c(1L, 3L),  ui = c(1L, 2L), lb = c(-Inf, 2), ub = c(4, 100),
+    nobj = 4L)
```

```
ROI Variable Bounds:

2 lower and 2 upper non-standard variable bounds.
```

4.6. Examples

Here we show how the different types of OPs can be formulated in **ROI**.

LP

Putting all this together, the LP

$$\begin{array}{rrcrcrl} \text{maximize} & 3x_1 & + & 7x_2 & - & 12x_3 & \\ \text{subject to} & 5x_1 & + & 7x_2 & + & 2x_3 & \leq 61 \\ & 3x_1 & + & 2x_2 & - & 9x_3 & \leq 35 \\ & x_1 & + & 3x_2 & + & x_3 & \leq 31 \end{array}$$

$$x_1, x_2 \geq 0,\ x_3 \in [-10, 10]$$

can be created by

```
R> lp <- OP(objective = L_objective(c(3, 7, -12)),
+    constraints = L_constraint(
+      L = rbind(c(5, 7,  2), c(3, 2, -9), c(1, 3,  1)),
+      dir = c("<=", "<=", "<="), rhs = c(61, 35, 31)),
+    bounds = V_bound(li = 3, ui = 3, lb = -10, ub = 10, nobj = 3),
+    maximum = TRUE)
R> lp

ROI Optimization Problem:

Maximize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 3 constraints of type linear.
- 1 lower and 1 upper non-standard variable bound.
```

Once an OP is constructed, the functions `objective()`, `constraints()`, `bounds()`, `types()` and `maximum()` can be used to access/alter the corresponding element. The function `objective()` returns the objective as function, which can be directly used to evaluate parameters. The number of parameters required, can be obtained by the generic function `length()`.

```
R> param <- rep.int(1, length(objective(lp)))
R> objective(lp)(param)

[1] -2
```

To access the data of the objective, the generic function `terms()` should be used.

```
R> terms(objective(lp))

$L
A 1x3 simple triplet matrix.

$names
NULL
```

For all the other elements the corresponding getter returns directly the underlying data representation.

MILP

To extend the LP from above to an MILP, we add the additional requirements $x_2, x_3 \in \mathbb{Z}$, which results in the following OP:

```
R> milp <- lp
R> types(milp) <- c("C", "I", "I")
R> milp

ROI Optimization Problem:

Maximize a linear objective function of length 3 with
- 1 continuous objective variable,
- 2 integer objective variables,

subject to
- 3 constraints of type linear.
- 1 lower and 1 upper non-standard variable bound.
```

BLP

The following example of a binary linear programming (BLP) problem is based on Fischetti and Salvagnin (2010) and will be used later to illustrate how multiple solutions can be obtained.

$$\begin{array}{rr} \text{minimize} & -x_1 - x_2 - x_3 - x_4 - 99x_5 \\ \text{subject to} & x_1 + x_2 \leq 1 \\ & x_3 + x_4 \leq 1 \\ & x_4 + x_5 \leq 1 \\ & x_i \in \{0, 1\} \end{array} \tag{18}$$

```
R> blp <- OP(objective = L_objective(c(-1, -1, -1, -1, -99)),
+    constraints = L_constraint(L = rbind(c(1, 1, 0, 0, 0), c(0, 0, 1, 1, 0),
+      c(0, 0, 0, 1, 1)),  dir = c("<=", "<=", "<="), rhs = rep.int(1, 3)),
+    types = rep("B", 5L))
```

QCQP

Following the definition from Equation 5, the quadratic terms are multiplied by one-half, therefore the QCQP

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}(x_1^2 + x_2^2) \\ \text{subject to} & \frac{1}{2}x_1^2 \geq \frac{1}{2} \\ & x_1, x_2 \geq 0 \end{array} \tag{19}$$

can be constructed by

```
R> qcqp <- OP(objective = Q_objective(Q = diag(2), L =  c(0, 0)),
+    constraints = Q_constraint(Q = rbind(c(1, 0), c(0, 0)), L = c(0, 0),
+      dir = ">=", rhs = 0.5))
R> qcqp

ROI Optimization Problem:

Minimize a quadratic objective function of length 2 with
- 2 continuous objective variables,

subject to
- 1 constraint of type quadratic.
- 0 lower and 0 upper non-standard variable bounds.
```

SOCP

For formulating SOCPs it can be advantageous to consider the following alternative standard form (Lobo *et al.* 1998; Andersen *et al.* 2012),

$$\begin{array}{rl} \text{minimize} & a_0^\top x \\ \text{subject to} & \|B_i x + w_i\|_2 \leq u_i^\top x + v_i, \quad i = 1, \ldots, k \end{array} \tag{20}$$

here k is the number of second-order cones, $B_i \in \mathbb{R}^{(d_i - 1) \times n}$, $w_i \in \mathbb{R}^{d_i - 1}$, $u_i \in \mathbb{R}^n$, $v_i \in \mathbb{R}$ and the dimension of each cone is given by d_i. Starting from Equation 20 the transformation into the standard form given in Equation 6 can be accomplished by

$$A = -\begin{bmatrix} u_1^\top \\ B_1 \\ \vdots \\ u_k^\top \\ B_k \end{bmatrix}, \quad b = \begin{bmatrix} v_1 \\ w_1 \\ \vdots \\ v_k \\ w_k \end{bmatrix}, \quad s \in \prod_{i=1,\ldots,k} \mathcal{K}_{\text{soc}}^{d_i}. \tag{21}$$

Therefore the OP

$$\begin{array}{rl} \text{maximize} & x + y + z \\ \text{subject to} & \sqrt{x^2 + z^2} \leq \sqrt{2} \\ & x, y, z \geq 0, \ y \leq 5 \end{array} \tag{22}$$

can be easily solved in **ROI**.

```
R> socp <- OP(objective = L_objective(c(1, 1, 1), names = c("x", "y", "z")),
+     constraints = C_constraint(rbind(c(0, 0, 0), c(-1, 0, 0), c(0, 0, -1)),
+     cones = K_soc(3), rhs = c(sqrt(2), 0, 0)),
+     bounds = V_bound(ui = 2, ub = 5, nobj = 3L), maximum = TRUE)
R> socp

ROI Optimization Problem:

Maximize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 3 constraints of type conic.
  |- 3 conic constraints of type 'soc'
- 0 lower and 1 upper non-standard variable bound.
```

Similarly by making use of the epigraph form (see Equation 7, the convex QP

$$\begin{array}{rl} \text{minimize} & \sqrt{x_1^2 + x_2^2} \\ \text{subject to} & x_1 + x_2 = 2 \\ & x_1, x_2 \geq 0 \end{array} \tag{23}$$

can be formulated as a SOCP.

```
R> A <- rbind(c(0, 0, -1), c(-1, 0, 0), c(0, -1, 0), c(1, 1, 0))
R> b <- c(1, 0, 0, 2)
R> cp <- OP(objective = L_objective(c(0, 0, 1)),
+     constraints = C_constraint(A, c(K_soc(3), K_zero(1)), b))
R> cp

ROI Optimization Problem:

Minimize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 4 constraints of type conic.
  |- 3 conic constraints of type 'soc'
  |- 1 conic constraint of type 'zero'
- 0 lower and 0 upper non-standard variable bounds.
```

SDP

Another standard form commonly used for SDPs (e.g., Vandenberghe and Boyd (1996b); Nemirovski (2004); Andersen *et al.* (2012)) is

$$\begin{array}{rl} \text{minimize} & a_0^\top x \\ \text{subject to} & \sum_{i=1}^n x_i F_i \preceq F_0, \end{array} \tag{24}$$

here $a_0 \in R^n$ and $F_i \in R^{d \times d}$ are symmetric matrices and $\preceq$ is the generalized inequality. Therefore $\sum_{i=1}^n x_i F_i \preceq F_0$ is equivalent to $F_0 - \sum_{i=1}^n x_i F_i \in \mathcal{K}_{psd}^d$. In order to transform an SDP problem given in the form of Equation 24 into the form shown in Equation 6, a half-vectorization should be performed. Half-vectorization is a special kind of matrix vectorization for symmetric matrices, which transforms a symmetric matrix

```
R> (A <- matrix(c(1, 2, 3, 2, 4, 5, 3, 5, 6), nrow = 3))
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    5
[3,]    3    5    6
```

into a vector, alike `vech` transforms n symmetric $d \times d$ matrices into a $(d(d+1)/2) \times n$ matrix:

```
R> vech(A)
```

```
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

Specifically, the following problem

$$\begin{array}{rl} \text{minimize} & x_1 + x_2 - x_3 \\ \text{subject to} & x_1 \begin{pmatrix} 10 & 3 \\ 3 & 10 \end{pmatrix} + x_2 \begin{pmatrix} 6 & -4 \\ -4 & 10 \end{pmatrix} + x_3 \begin{pmatrix} 8 & 1 \\ 1 & 6 \end{pmatrix} \preceq \begin{pmatrix} 16 & -13 \\ -13 & 60 \end{pmatrix} \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

can be modeled as follows:

```
R> F1 <- rbind(c(10, 3), c(3, 10))
R> F2 <- rbind(c(6, -4), c(-4, 10))
R> F3 <- rbind(c(8, 1), c(1, 6))
R> F0 <- rbind(c(16, -13), c(-13, 60))
R> psd <- OP(objective = L_objective(c(1, 1, -1)),
+    constraints = C_constraint( L = vech(F1, F2, F3), cone = K_psd(3),
+      rhs = vech(F0)))
R> psd
```

```
ROI Optimization Problem:

Minimize a linear objective function of length 3 with
- 3 continuous objective variables,
subject to
- 3 constraints of type conic.
  |- 3 conic constraints of type 'psd'
- 0 lower and 0 upper non-standard variable bounds.
```

NLP

The following example from Rosenbrock (1960) is some times referred to as Rosenbrock's post office problem.

$$\begin{array}{rr} \text{maximize} & x_1 x_2 x_3 \\ \text{subject to} & x_1 + 2x_2 + 2x_3 \leq 72 \\ & x_1, x_2, x_3 \in [0, 42] \end{array} \tag{25}$$

```
R> nlp_1 <- OP()
R> gradient <- function(x) c(prod(x[-1]), prod(x[-2]), prod(x[-3]))
R> objective(nlp_1) <- F_objective(F = function(x) prod(x), n = 3, G = gradient)
R> rosenbrock <- function(x) x[1] + 2 * x[2] + 2 * x[3]
R> constraints(nlp_1) <- F_constraint(F = rosenbrock, dir = "<=", rhs = 72,
+    J = function(x) c(1, 2, 2))
R> bounds(nlp_1) <- V_bound(ud = 42, nobj = 3L)
R> maximum(nlp_1) <- TRUE
R> nlp_1

ROI Optimization Problem:

Maximize a nonlinear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 1 constraint of type nonlinear.
- 0 lower and 3 upper non-standard variable bounds.
```

Alternatively the linear constraint $x_1 + 2x_2 + 2x_3 \leq 72$ could and should be modeled directly as a linear constraint,

```
R> nlp_2 <- nlp_1
R> constraints(nlp_2) <- L_constraint(L = c(1, 2, 3), "<=", 72)
R> nlp_2

ROI Optimization Problem:

Maximize a nonlinear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 1 constraint of type linear.
- 0 lower and 3 upper non-standard variable bounds.
```

using `L` and `Q` constraints rather than `F_constraint` has the advantage that for `L` and `Q` constraints the Jacobian is derived analytically if needed and not provided.

5. Package ROI

The R optimization infrastructure can be structured into the package **ROI** and its accompanying extensions. Package **ROI** provides all the necessary classes, methods and manages the extensions. The extension packages add optimization solvers, read/writing functions and additional resources (e.g., model collections). Currently **ROI** distinguishes between two different types of extensions, namely, plug-ins and models. Here plug-ins play a special role, hence all plug-ins are loaded automatically when **ROI** is loaded. When a plug-in is loaded it provides data about its capabilities. This data is stored in an in-memory database and includes information about to which problems the plug-in is applicable, which formats it can read/write and the control arguments available from the solver and how the solver specific control arguments relate to arguments commonly used.

This mechanism makes it possible that **ROI** is aware of all the installed plug-ins, without the need to change **ROI** when a new plug-in is added. To make the automatic loading possible the plug-ins have to follow the name convention `ROI.plugin.`<name>, where <name> is typically the name of an optimization solver (e.g., **ROI.plugin.glpk** (Theussl 2017)). The prefix `ROI.models` (e.g., **ROI.models.netlib** (Schwendinger 2016)) is used for data packages with predefined OPs. In Section 5.6 we give an overview about the data packages available in the **ROI** format.

5.1. Solving optimization problems

After formulating an OP as described in Section 4, it can be solved by calling the function `ROI_solve(x, solver, control, ...)`. This function takes an R object of class `OP` containing the formulation of the OP, the name of the solver to be used and a list containing solver-specific parameters as arguments. The `solver` and `control` arguments are optional, if no `solver` argument is provided **ROI** will choose an applicable solver automatically (see Section 5.7.1). Alternatively the solver-specific parameters can be specified via the dots arguments.

```
R> lp_sol <- ROI_solve(lp, solver = "glpk")
R> lp_sol

Optimal solution found.
The objective value is: 8.670149e+01

R> milp_sol <- ROI_solve(milp, solver = "symphony")
R> milp_sol

Optimal solution found.
The objective value is: 8.100000e+01

R> blp_sol <- ROI_solve(blp, solver = "glpk")
R> blp_sol
```

```
Optimal solution found.
The objective value is: -1.010000e+02

R> socp_sol <- ROI_solve(cp, solver = "ecos")
R> socp_sol

Optimal solution found.
The objective value is: 4.142136e-01

R> psd_sol <- ROI_solve(psd, solver = "scs")
R> psd_sol

Optimal solution found.
The objective value is: -1.486461e+00

R> nlp_1_sol <- ROI_solve(nlp_1, solver = "alabama", start = c(10, 10, 10))
R> nlp_1_sol

Optimal solution found.
The objective value is: 3.456000e+03
```

Some OPs have multiple solutions, in the case of BLP (MILP) some solver can retrieve all (multiple) solutions. For MILPs it is in general not possible to obtain all the solutions but only multiple solutions, since even this simple MILP

$$\begin{array}{rr} \text{minimize} & x_1 - x_2 \\ \text{subject to} & x_1 - x_2 = 0 \\ & x_1, x_2 \in \mathbb{Z} \end{array} \tag{26}$$

has an infinite number of solutions. In the following we use the `"msbinlp"` solver to retrieve all the solutions to the OP defined in Equation 18, here `method` gives the solver used within the inner loop and `nsol_max` the maximal number of solutions to be returned. Since we have a pure binary problem and five objective variables, it is clear that there can be at most ten solutions,

```
R> blp_sol <- ROI_solve(blp, solver = "msbinlp", method = "glpk", nsol_max = 10)
R> blp_sol

2 optimal solutions found.
The objective value is: -1.010000e+02
```

alternatively it is also possible to set `nsol_max` to `Inf`. Then **ROI** tries to retrieve all the solutions.

5.2. Solution and status code

To make the solutions of the various solvers easy to understand, all the solutions are canonicalized within the plug-ins. After the canonicalization each solution contains the following components:

`solution` the solution of the OP,

`objval` the optimal objective value,

`status` the canonicalized status code,

`message` the original solver message

and a meta attribute containing the solver name and additional optional arguments.

Solver status codes are used to inform the user about the exit status of the solver. Despite the common usage of status codes in optimization solvers there is no widely used standard. Nevertheless, we believe it is desirable to provide unified status codes. The status codes used in `ROI_solve` are simple and consistent with the common practice, to return `0` on success (if a "solution" meeting the solver specific requirements was found) `1` otherwise. To obtain the (primal) "solution" the generic function `solution(x, type)` should be used,

```
R> solution(lp_sol)
```

```
[1]  0.000000  9.238806 -1.835821
```

in the case of multiple solutions a `"list"` of solutions is returned.

```
R> solution(blp_sol)
```

```
[[1]]
[1] 0 1 1 0 1

[[2]]
[1] 1 0 1 0 1
```

If the status code is 1 `solution` will return `NA`, to prevent the user from using solutions with a status code different from 0.

```
R> lp_inaccurate_sol <- ROI_solve(lp, solver = "scs", tol = 1e-32)
R> solution(lp_inaccurate_sol)
```

```
[1] NA NA NA
```

However in a few situations it can be desirable to obtain solutions even if the solver signals no success. In these cases **ROI** can be forced to return the solution provided by the solver regardless of the status code.

```
R> solution(lp_inaccurate_sol, force = TRUE)
```

```
[1]  8.142725e-16  9.238806e+00 -1.835821e+00
```

The "solution" to the dual problem can be retrieved by.

```
R> solution(lp_sol, type = "dual")

[1] -4.298507  0.000000  0.000000
```

Furthermore, auxiliary variables

```
R> solution(lp_sol, type = "aux")

$primal
[1] 61.0000 35.0000 25.8806

$dual
[1] 0.5820896 1.4626866 0.0000000
```

the solution matrices of a PSD problem

```
R> lapply(solution(psd_sol, type = "psd"), as.matrix)

$`4`
           [,1]        [,2]
[1,] 0.11050022 0.031337481
[2,] 0.03133748 0.008887201
```

the original solver message

```
R> solution(lp_sol, type = "msg")

$optimum
[1] 86.70149

$solution
[1]  0.000000  9.238806 -1.835821

$status
[1] 5

$solution_dual
[1] -4.298507  0.000000  0.000000

$auxiliary
$auxiliary$primal
[1] 61.0000 35.0000 25.8806

$auxiliary$dual
[1] 0.5820896 1.4626866 0.0000000
```

the objective value

```
R> solution(lp_sol, type = "objval")

[1] 86.70149
```

the status

```
R> solution(lp_sol, type = "status")

$code
[1] 0

$msg
  solver glpk
    code 5
  symbol GLP_OPT
 message Solution is optimal.
roi_code 0
```

and the status code

```
R> solution(lp_sol, type = "status_code")

[1] 0
```

of the OP can be retrieved by the function `solution()`.

5.3. Reformulations

Reformulations are often used to transform a problem of class A into a problem of class B, where the solution of the original problem can be derived from the solution of the reformulation (which is typically easier to solve). Although reformulation techniques are commonly used in optimization the functions performing these reformulations are generally hidden within the optimization software. To facilitate the comparison of different reformulation algorithms **ROI** provides functions for managing reformulations. Function `ROI_registered_reformulations()` lists the available reformulations and `ROI_reformulate(x, to, method)` performs the reformulation. Following Boros and Hammer (2002) we illustrate the transformation of a binary QP into a MILP, the code for the reformulation is based on the implementation in the **relations** (Meyer and Hornik 2017) package.

$$\begin{aligned} \text{minimize} \quad & 6 - x - 4y - z + 3xy + yz \\ & x, y, z \in \{0, 1\} \end{aligned} \tag{27}$$

```
R> Q <- rbind(c(0, 3, 0), c(0, 0, 1), c(0, 0, 0))
R> bqp <- OP(Q_objective(Q = Q + t(Q), L = c(-1, -4, -1)), types = rep("B", 3))
R> glpk_signature <- ROI_solver_signature("glpk")
R> head(glpk_signature, 3)
```

```
  objective constraints bounds cones maximum     C     I     B
1         L           X      X     X    TRUE  TRUE FALSE FALSE
2         L           L      X     X    TRUE  TRUE FALSE FALSE
3         L           X      X     X    TRUE FALSE  TRUE FALSE
```

```
R> milp <- ROI_reformulate(x = bqp, to = glpk_signature)
R> ROI_solve(milp, solver = "glpk")
```

```
Optimal solution found.
The objective value is: -4.000000e+00
```

Here **ROI** selects the applicable reformulations based on the provided signatures. A method is considered to be applicable if it can transform the given OP into a new OP, where the signature of the new OP is a subset of the signature provided in the argument `to`. Since it is possible that several methods are applicable, the argument `method` can be used to select a specific reformulation method.

5.4. ROI solvers

ROI currently can make use of eighteen different solvers, applicable to a wide range of OPs. Inspired by R's `available.packages()` function, **ROI** can return a listing of the solver plug-ins available at CRAN (`https://CRAN.R-project.org`), R-Forge (`https://r-forge.r-project.org/`, Theußl and Zeileis 2009) and GitHub (`https://github.com/`). `ROI_available_solvers()` without an argument lists all the available solvers. If an OP is provided as argument, only the available solvers applicable will be returned.

```
R> ROI_available_solvers(cp)[, c("Package", "Repository")]
```

```
           Package                             Repository
4  ROI.plugin.ecos https://cran.r-project.org/src/contrib
12  ROI.plugin.scs https://cran.r-project.org/src/contrib
17 ROI.plugin.ecos           http://R-Forge.R-project.org
27  ROI.plugin.scs           http://R-Forge.R-project.org
```

A listing of all the available plug-ins on CRAN and R-Forge could be easily compiled by just using the `available.packages()` function. But to be able to find all the solvers available and applicable to a given OP also the solver signature is needed. Therefore a database containing the solver signatures and the information provided by `available.packages` was compiled and is queried whenever `ROI_available_solvers` is called.

A vector of all solvers installed and loaded (registered) can be obtained by,

```
R> ROI_registered_solvers()
```

```
               nlminb                alabama                   cbc
  "ROI.plugin.nlminb"   "ROI.plugin.alabama"      "ROI.plugin.cbc"
                  clp                  cplex               deoptim
     "ROI.plugin.clp"     "ROI.plugin.cplex"  "ROI.plugin.deoptim"
```

```
                  ecos                   glpk                 gurobi
    "ROI.plugin.ecos"      "ROI.plugin.glpk"    "ROI.plugin.gurobi"
                  ipop                lpsolve                  mosek
    "ROI.plugin.ipop"   "ROI.plugin.lpsolve"     "ROI.plugin.mosek"
               msbinlp                 nloptr                 optimx
 "ROI.plugin.msbinlp"    "ROI.plugin.nloptr"    "ROI.plugin.optimx"
              quadprog                    scs               symphony
"ROI.plugin.quadprog"       "ROI.plugin.scs"  "ROI.plugin.symphony"
```

similarly

```
R> ROI_applicable_solvers(cp)
```

```
[1] "ecos" "scs"
```

returns a vector giving the names of the registered solvers applicable to a given problem. Both return values are based on the solver registry, which stores the solver method and information about the solver registered by the plug-ins. The solver registry is an in-memory database based on the **registry** (Meyer 2015) package.

`ROI_installed_solvers` gives a listing of all the installed plug-ins (not necessarily loaded) delivered directly with **ROI** and found

```
R> ROI_installed_solvers()
```

```
                nlminb                alabama                    cbc
  "ROI.plugin.nlminb"   "ROI.plugin.alabama"       "ROI.plugin.cbc"
                   clp                  cplex                deoptim
     "ROI.plugin.clp"     "ROI.plugin.cplex"   "ROI.plugin.deoptim"
                  ecos                   glpk                 gurobi
    "ROI.plugin.ecos"      "ROI.plugin.glpk"    "ROI.plugin.gurobi"
                  ipop                lpsolve                  mosek
    "ROI.plugin.ipop"   "ROI.plugin.lpsolve"     "ROI.plugin.mosek"
               msbinlp                 nloptr                 optimx
 "ROI.plugin.msbinlp"    "ROI.plugin.nloptr"    "ROI.plugin.optimx"
              quadprog                    scs               symphony
"ROI.plugin.quadprog"       "ROI.plugin.scs"  "ROI.plugin.symphony"
```

by searching for the prefix '`ROI.plugin`' in the R library trees.

An overview on the currently available solver plug-ins based on the problem types is given in Table 4. Please note that the functionality provided in a plug-in does not necessarily have to be the same as the functionality of the solver, e.g., **ROI.plugin.nlminb** can take functional constraints **nlminb** can only take box constraints. Furthermore we want to emphasize that **ROI** was built to be extended, as shown in Section 6.

5.5. ROI read/write

OPs are commonly stored in flat file formats, different solvers allow to read/write different types of this file formats. **ROI** manages the reader/writer registered in the plug-ins, thus allows to write

	Objective			Constraints				Types			Bounds
	linear	*quadratic*	*nonlinear*	*linear*	*quadratic*	*conic*	*nonlinear*	*binary*	*integer*	*continuous*	*box*
alabama			✓				✓			✓	✓
cbc	✓			✓				✓	✓	✓	✓
clp	✓			✓						✓	✓
cplex	✓	✓		✓	✓			✓	✓	✓	✓
deoptim			✓				✓			✓	✓
ecos	✓			✓		✓		✓	✓	✓	✓
glpk	✓			✓				✓	✓	✓	✓
gurobi	✓	✓		✓	✓			✓	✓	✓	✓
ipop	✓	✓		✓						✓	✓
lpsolve	✓			✓				✓	✓	✓	✓
mosek	✓	✓		✓				✓	✓	✓	✓
msbinlp	✓			✓				✓	✓	✓	✓
nlminb			✓				✓			✓	✓
nloptr			✓				✓			✓	✓
optimx			✓				✓			✓	✓
quadprog		✓		✓						✓	✓
scs	✓			✓		✓				✓	✓
symphony	✓			✓				✓	✓	✓	✓

Table 4: Currently available **ROI** plugins.

```
R> lp_file <- tempfile()
R> write.op(lp, lp_file, "lp_lpsolve")
R> writeLines(readLines(lp_file))

/* Objective function */
max: +3 C1 +7 C2 -12 C3;

/* Constraints */
+5 C1 +7 C2 +2 C3 <= 61;
+3 C1 +2 C2 -9 C3 <= 35;
+C1 +3 C2 +C3 <= 31;

/* Variable bounds */
-10 <= C3 <= 10;
```

and read

```
R> read.op(lp_file, "lp_lpsolve")

ROI Optimization Problem:
```

```
Maximize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 3 constraints of type linear.
- 1 lower and 1 upper non-standard variable bound.
```

OPs in various formats. Information about the available reader/writer can be obtained via the functions `ROI_registered_reader()` and `ROI_registered_writer()`.

5.6. ROI models

In optimization test problem collections are commonly used to evaluate and compare the performance of solvers. Thereby each class of optimization problems has its own test sets stored in various formats, **ROI** currently provides access to NETLIB-LP, MIPLIB and the **globalOptTests** package. The NETLIB-LP (Gay 1985) is a collection of linear programming problems, which, even though the main part was created more than 30 years ago is still used today. Mixed integer optimization problems are commonly evaluated using MIPLIB (Koch, Achterberg, Andersen, Bastert, Berthold, Bixby, Danna, Gamrath, Gleixner, Heinz, Lodi, Mittelmann, Ralphs, Salvagnin, Steffy, and Wolter 2011), an extensive collection of academic and industrial MILP applications. The **globalOptTests** (Mullen 2014a) package contains 50 box constrained nonlinear global OPs for benchmarking purposes. These libraries were transformed into **ROI** optimization problems and can be accessed through packages **ROI.models.netlib**, **ROI.models.miplib** (Schwendinger and Theussl 2017) and **ROI.models.globalOptTests** (Schwendinger 2017). Since MIPLIB provides no license file, the OPs are not included in the package but can be easily obtained with the function `miplib_download()`.

```
R> library("ROI.models.miplib")
R> if ( length(miplib()) == 0L ) {
+    miplib_download_benchmark(quiet = TRUE)
+    miplib_download_metainfo()
+ }
R> ops <- miplib("ns1766074")
R> ops

ROI Optimization Problem:

Minimize a linear objective function of length 100 with
- 10 continuous objective variables,
- 90 integer objective variables,

subject to
- 182 constraints of type linear.
- 0 lower and 0 upper non-standard variable bounds.
```

Since the problems are stored as objects of class `'OP'` they can be directly entered into `ROI_solve`. In the following we show a benchmark example, to make the example easily

```
R> boxplot(benchmark, ylab = "Time in Milliseconds", xlab = NULL, unit = "ms",
+    outline = FALSE)
```

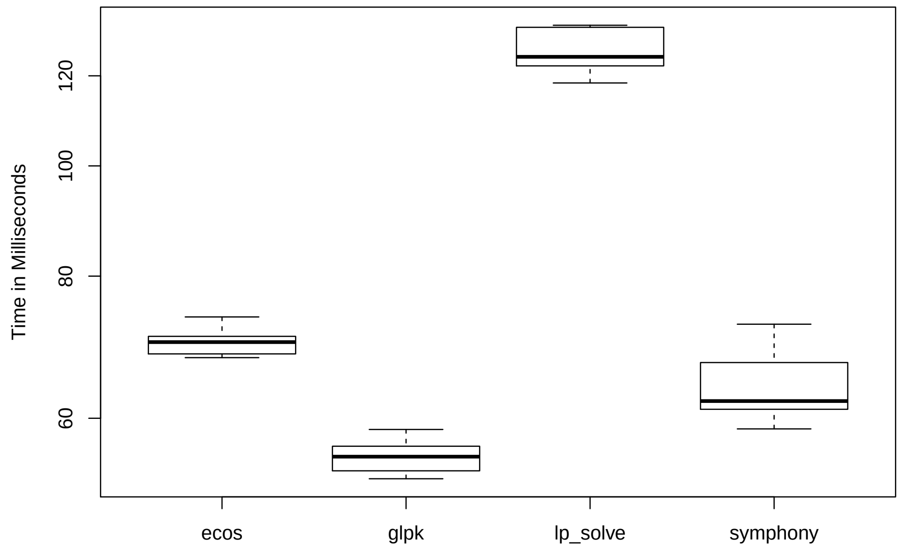

Figure 1: LP solver benchmark.

reproducible, we only considered open source solvers and a small OP. However this can be easily extended by adding solvers and iterating over more problems.

```
R> library("ROI.models.netlib")
R> library("microbenchmark")
R> agg <- netlib("agg")
R> x <- list()
R> benchmark <- microbenchmark(
+    ecos = (x$ecos <- ROI_solve(agg, "ecos")),
+    glpk = (x$glpk <- ROI_solve(agg, "glpk")),
+    lp_solve = (x$lp_solve <- ROI_solve(agg, "lpsolve")),
+    symphony = (x$symphony <- ROI_solve(agg, "symphony")),
+    times = 30)
```

ROI makes these data collections (test problem sets) available in a common format, so users can easily compare the different solvers and developers interested in creating optimization software can use them to test their packages. Furthermore, the intuitive structure of **ROI** objects and its use of sparse data structures makes it possible to directly derive a new format for the exchange of linear, quadratic and conic optimization problems.

```
R> library("jsonlite")
R> nested_unclass <- function(x) {
```

```
+    x <- unclass(x)
+    if ( is.list(x) )
+      x <- lapply(x, nested_unclass)
+    x
+  }
R> agg_json <- toJSON(nested_unclass(agg))
R> tmp_file <- tempfile()
R> writeLines(agg_json, tmp_file)
```

The resulting text file can be easily imported into any programming language supporting JavaScript Object Notation (JSON). JSON is an open-standard file format that can be parsed by almost all programming languages. For historic reasons, OP collections are commonly provided in flat file formats (e.g., `MPS`, `QPS`). We believe, that today it would be advantageous to store them in general data exchange formats like JSON or XML.

5.7. ROI settings

Solver selection

In the case no solver is provided in `ROI_solve`, the default solver set in `ROI_options` will be used.

```
R> ROI_options("default_solver")
```

```
[1] "auto"
```

By default the option `"default_solver"` is set to `"auto"` which enables automatic solver selection, if any other solver name (e.g., `"glpk"`) is provided the automatic solver selection is discarded in favor of the specified solver.

```
R> ROI_options("default_solver", "glpk")
```

Load plug-ins

The plug-ins are loaded automatically, in some situations it is desirable to deactivate the automatic loading and require plug-in packages one at a time. This can be accomplished by setting the environment variable `"ROI_LOAD_PLUGINS"` to `FALSE`.

```
R> Sys.setenv(ROI_LOAD_PLUGINS = FALSE)
```

Afterwards the default load behavior of **ROI** is altered and only the `"nlminb"` solver (which is included in **ROI**) gets registered when `library("ROI")` is called. Therefore all the other plug-ins have to be loaded manually if needed (e.g., `library("ROI.plugin.glpk")`).

6. Extending ROI

To stay abreast of changes and further the availability of different solvers in the **ROI** ecosystem, **ROI** allows developers to integrate their own extensions, so called plug-ins. This can be seen as one of the key features, since it allows the use of new solvers with no or minimal code changes.

Extending **ROI** with a new solver method can be split into three parts. First, a function to be called by **ROI** has to be written. Second, the function plus information about the function are added into the **ROI** solver registry. Third, a mapping from the solver specific arguments and the status codes to their **ROI** counterpart has to be provided.

6.1. Signatures

In order to establish a connection between the OP and the solvers provided via plug-ins, both are equipped with a signature. The signature captures all the information necessary to determine which solver is applicable to a given problem.

```
R> OP_signature(lp)
```

```
  objective constraints bounds cones maximum    C     I     B
1         L           L      V     X    TRUE TRUE FALSE FALSE
```

New signatures are created by the function `ROI_plugin_make_signature()`. The following shows how to create the signature for the glpk solver,

```
R> glpk_signature <- ROI_plugin_make_signature(objective = "L",
+    constraints = "L", types = c("C", "I", "B", "CI", "CB", "IB", "CIB"),
+    bounds = c("X", "V"), maximum = c(TRUE, FALSE))
```

where the objective and the constraints have to be linear. Furthermore this signature indicates that, the variable types are allowed to be binary (`"B"`), integer (`"I"`), continuous (`"C"`) or any combinations of them. The bounds have to be variable bounds (`"V"`) or no bounds at all encoded by `"X"`. The last argument maximum specifies that, GLPK can find both maxima and minima.

6.2. Writing a new solver method

Any function supposed to add a solver to **ROI** has to take the arguments `x` and `control`, where `x` is of class '`OP`' and `control` a list containing the additional arguments. Additionally the solution has to be canonicalized before it is returned. The following shows the code from **ROI.plugin.glpk** for solving linear problems.

```
R> glpk_solve_OP <- function(x, control = list()) {
+    control$canonicalize_status <- FALSE
+    glpk <- list(Rglpk_solve_LP, obj = terms(objective(x))[["L"]],
+      mat = constraints(x)$L, dir = constraints(x)$dir,
+      rhs = constraints(x)$rhs, bounds = bounds(x),
+      types = types(x), max = maximum(x), control = control)
+    mode(glpk) <- "call"
+    if ( isTRUE(control$dry_run) )
+      return(glpk)
+
+    out <- eval(glpk)
+    ROI_plugin_canonicalize_solution(solution = out$solution,
+      optimum = out$optimum, status = out$status, solver = "glpk",
+      message = out)
+  }
```

As can be seen from this example, most plug-ins support the optional control argument `dry_run`, which returns the solver call. This is especially useful for debugging wrapper functions with more transformation steps, so the data used in the solver call can be easily shared and inspected.

6.3. Register solver methods

Registering a solver method can be seen as telling **ROI** which function it should use when `ROI_solve` with argument solver set to the name of the plug-in (e.g., `"glpk"`) is called. In order to avoid ambiguity, each plug-in should at most provide one method for each problem type. Solver methods are registered via the function `ROI_plugin_register_solver_method`, which takes as arguments the problem types (as signatures), the solver name and a wrapper function `ROI_solve()` is dispatched to.

```
ROI_plugin_register_solver_method(glpk_signature, "glpk", glpk_solve_OP)
```

After the solver registration the name of the solver will appear among the registered solvers.

6.4. Adding additional information

To be able to provide a consistent interface, each plug-in has to define a mapping between the solver specific status codes and the status codes used by **ROI**, as well as a mapping between solver specific control variables and **ROI** control variables.

Status codes

Status codes can be added via the function `ROI_plugin_add_status_code_to_db()`:

```
ROI_plugin_add_status_code_to_db(solver = "glpk", code = 5L,
  symbol = "GLP_OPT",
  message = "Solution is optimal.",
  roi_code = 0L)
```

Here, the `"glpk"` specific status code `5L` is mapped to the canonicalized **ROI** status code `0L`, which signals that the solution has been optimal as indicated by the status message.

Control variables

Plug-ins are contracted to provide a mapping between the names of the control variables used by **ROI** and the names of the control variables used by the plug-in. The following maps the **glpk** argument `tm_limit` to the **ROI** equivalent `max_time`.

```
ROI_plugin_register_solver_control(solver = "glpk", args = "tm_limit",
  roi_control = "max_time")
```

6.5. Register reformulations

While in Section 5.3 we showed how to use reformulations, here we explain how new reformulations can be added through plug-ins. Again, the signature is used to define which transformations can be performed by a given method.

```
R> bqp_signature <- ROI_plugin_make_signature(objective = "Q",
+    constraints = c("X", "L"), types = c("B"), bounds = c("X", "V"),
+    cones = c("X"), maximum = c(TRUE, FALSE))
R> milp_signature <- ROI_plugin_make_signature(objective = "L",
+    constraints = c("X", "L"),
+    types = c("C", "I", "B", "CI", "CB", "IB", "CIB"), bounds = c("X", "V"),
+    maximum = c(TRUE, FALSE), cones = c("X"))
```

```
ROI_plugin_register_reformulation(
  from = bqp_signature, to = milp_signature, method_name = "bqp_to_lp",
  method = bqp_to_lp, description = "", cite = "", author = "")
```

The code above registers the function `bqp_to_lp()`, which is based on the function `.linearize_BQP()` from the **relations** package, as a new reformulation named `"bqp_to_lp"`. The parameter `from` defines which signatures the original problem is allowed to have and `to` defines all possible signatures the reformulation could have.

6.6. Register reader/writer

Plug-ins can also add new read and write functions. Any method to be registered as read function has to take as arguments `file` the file name and `...` for optional additional arguments.

```
R> library("slam")
R> json_reader_lp <- function(file, ...) {
+    stopifnot(is.character(file))
+    y <- read_json(file, simplifyVector = TRUE)
+    to_slam <- function(x) do.call(simple_triplet_matrix, x)
+    x <- OP()
+    objective(x) <- L_objective(to_slam(y[["objective"]][["L"]]),
+      y[["objective"]][["names"]])
+    constraints(x) <- L_constraint(to_slam(y[["constraints"]][["L"]]),
+      y[["constraints"]][["dir"]], y[["constraints"]][["rhs"]],
+      y[["constraints"]][["names"]])
+    types(x) <- y[["types"]]
+    bounds(x) <- structure(y[["bounds"]], class = c("V_bound", "bound"))
+    maximum(x) <- as.logical(y[["maximum"]])
+    x
+ }
```

The write functions need the additional argument `x`, which is the OP to be written out.

```
R> json_writer_lp <- function(x, file, ...) {
+    writeLines(toJSON(nested_unclass(x), null = "null"), con = file)
+  }
```

Using the JSON based exchange format propagated in Section 5.6, we illustrate how to register simple JSON read and write functions for linear problems.

```
R> plugin_name <- "io"
R> ROI_plugin_register_writer("json", plugin_name, milp_signature,
+    json_writer_lp)
R> ROI_plugin_register_reader("json", plugin_name, json_reader_lp)
```

After the registration of the functions they can be used in the typical way.

```
R> fname <- tempfile()
R> file <- write.op(lp, file = fname, type = "json")
R> (lp_json <- read.op(file = fname, type = "json"))

ROI Optimization Problem:

Maximize a linear objective function of length 3 with
- 3 continuous objective variables,

subject to
- 3 constraints of type linear.
- 1 lower and 1 upper non-standard variable bound.
```

6.7. ROI tests

Writing tests is an important task in software development. The **ROI.tests** package provides a collection of tests which should be applied to any **ROI** plug-in during development. Since **ROI** knows the signature of each solver, **ROI.tests** can select the appropriate tests based on the solver name.

```
R> library("ROI.tests")
R> test_solver("glpk")

LP-01:  OK!
LP-02:  OK!
LP-03:  OK!
MILP-01:  OK!
MILP-02:  OK!
```

7. Applications

In the following we demonstrate how **ROI** can be used to solve selected problems from statistics.

7.1. Best subset selection

Recently Bertsimas *et al.* (2016) reported a bewildering 450 billion factor speedup from 1991 to 2015 for solving MIP, which is partly due to algorithmic improvements and partly through hardware speedups. They show how this speed gain can be utilized to solve the best subset selection problem in regression (see for example Miller 2002), which is an NP hard combinatorial OP. The best subset selection problem is a variable selection scheme which extends linear least-squares by adding a constraint on the number of predictor variables.

$$\underset{\beta}{\text{minimize}} \ \frac{1}{2}||y - X\beta||_2^2 \quad \text{subject to} \quad \sum_{i=1}^{p} \mathbb{I}_{\{\beta_i \neq 0\}} \leq k \tag{28}$$

As Equation 28 suggests the best subset selection is in spirit similar to ridge regression and lasso. However instead of the l_2 and l_1 norm best subset selection uses the l_0 norm which makes it non-convex and therefore hard to solve. The **leaps** (Lumley 2017) package implements an efficient branch-and-bound algorithm which is significantly faster than exhaustive search. Using a optimization solver has the additional advantage that it is possible to impose additional restrictions, like if the quadratic term of a covariate is selected to be in the equation the linear term has also to be selected. In **ROI** best subset selection can be either implemented as mixed integer quadratic problem or as mixed integer second order cone problem. An implementation of the second order cone version can be found in the Appendix B.

7.2. Sum-of-norms clustering

Borrowing ideas from regularization, sum-of-norms (SON) clustering (convex clustering) is an interesting alternative to established clustering approaches like hierarchical or k-means clustering, which has attracted a lot of research in recent years (Pelckmans, De Brabanter, De Moor, and Suykens 2005; Lindsten, Ohlsson, and Ljung 2011; Hocking, Joulin, Bach, and Vert 2011; Zhu, Xu, Leng, and Yan 2014; Chi and Lange 2015; Tan, Witten *et al.* 2015). Pelckmans *et al.* (2005); Hocking *et al.* (2011) describe SON clustering as a convexification of hierarchical clustering and Lindsten *et al.* (2011) establish that SON clustering can be seen as a convex relaxation of k-means clustering. Due to its convexity, SON clustering is not dependent on the starting values, which is a clear advantage over the non-convex k-means and hierarchical clustering.

SON clustering solves the following convex OP,

$$\underset{M_i}{\text{minimize}} \ \frac{1}{2}\sum_{i=1}^{m} ||X_{i*} - M_{i*}||_2^2 + \lambda \sum_{i<j} ||M_{i*} - M_{j*}||_q \tag{29}$$

where $q \in \{1, 2, \infty\}$, $X \in \mathbb{R}^{m \times n}$ is the data matrix and M_{i*} the i-th row of the optimization variable M. The regularization term $\lambda \sum_{i<j} ||M_{i*} - M_{j*}||_q$ induces equal rows M_{i*}. For $\lambda = 0$ all rows are unique and M is equal to X, when λ increases the number of unique rows of M will decrease. This gives a clustering where all equal rows belong to the same cluster. By solving Equation 29 for different λ_i, where $\lambda_1 < \lambda_2 < \cdots < \lambda_{k-1} < \lambda_k$, one can obtain a hierarchical clustering tree (Pelckmans *et al.* 2005).

There exist at least two implementations of SON clustering in R, Hocking *et al.* (2011) provide their code on R-Forge (`https://r-forge.r-project.org/projects/clusterpath/`) and Chi and Lange (2015) provide a fast implementation of SON clustering on CRAN (`https://cran.r-project.org/package=cvxclustr`).

A **ROI** formulation as SOCP of SON clustering can be found in the Appendix C.

7.3. Graphical lasso

Obtaining good estimates of the covariance matrix Σ is important in modern statistics. Often Σ is not estimated directly but its inverse, the precision matrix $\Theta = \Sigma^{-1}$ (e.g., Meinshausen and Bühlmann (2006)). Estimating the precision matrix instead of the covariance matrix has the advantage that there is a direct connection between the precision matrix and Gaussian graphical models, in the sense that the precision matrix defines the structure of the Gaussian graphical model. Since the elements of the precision matrix are the partial correlations, Θ_{ij} is zero if and only if i and j are conditionally independent. Translated to Gaussian graphical models, two edges A and B are only connected if the corresponding entry in the precision matrix is non zero (Lauritzen 1996).

Several authors proposed an algorithm connected to the lasso (Tibshirani 1996), to obtain a sparse estimate of the precision matrix, the so-called graphical lasso (glasso) (Friedman, Hastie, and Tibshirani 2008). The glasso solves the following convex OP,

$$\text{minimize }_{\Theta \succ 0} \; f(\Theta) = -\log(\det(\Theta)) + \operatorname{tr}(S\ \Theta) + \lambda\ ||X||_1 \tag{30}$$

here the data matrix $X \in \mathbb{R}^{m \times p}$ is assumed to be generated from a p-dimensional multivariate normal distribution $N_p(\mu, \Sigma)$ and S is the sample covariance matrix of X. Making use of the exponential and semidefinite cone, this can be brought into the CP standard form and solved by **ROI** using **SCS**. The corresponding R code can be found in the Appendix D.

8. Conclusions

In this paper we presented the **ROI** package and its extensions (plug-ins). **ROI** provides a consistent way to model OPs in R. Thereby it makes strong use of R's generic functions, such that users already familiar with R are not obligated to learn a new language. The plug-in packages equip **ROI** with optimization solvers and predefined optimization models. **ROI** is currently applicable to linear, quadratic, conic and general nonlinear OPs and provides access to eighteen solvers and three model plug-ins.

We illustrated how **ROI** can be used to solve OPs from many different problem classes. Furthermore, we have shown how **ROI** can be used to solve challenging statistical problems like best subset, convex clustering and glasso. The plug-in package **ROI.plugin.msbinlp** (Hornik, Meyer, and Schwendinger 2017b) serves as an example to highlight the benefit of the development of new packages based on **ROI**. The main benefit is that there is no need to have a dependency on a specific solver which could be also interesting for the implementation of nonlinear optimization algorithms (e.g., sequential quadratic programming). Another benefit is that package authors can reuse test cases from other packages based on **ROI** and the plug-in package **ROI.tests** provides a standardized way to test new solver plug-ins.

Although **ROI** already is able to cope with a wide range of optimization problems, there are still many possibilities for extensions. These include extending the supported functional forms of the objective functions and constraints as well as adding additional solvers or model collections through plug-ins. The following gives an overview of the planned extensions.

- Add solver for non-convex QPs (prototype for COIN-OR **qpOASES** (Ferreau, Kirches, Potschka, Bock, and Diehl 2014) solver interface is already on R-Forge).
- Add solver for mixed integer nonlinear problems, since **Couenne** (Belotti, Lee, Liberti, Margot, and Wächter 2009) is built on top of **Ipopt** (Wächter and Biegler 2006) and **Bonmin** (Bonami and Lee 2013) these solvers are also included in the interface.
- Add reader for the QPLIB file format (Furini, Traversi, Belotti, Frangioni, Gleixner, Gould, Liberti, Lodi, Misener, Mittelmann, Sahinidis, Vigerske, and Wiegele 2017).
- Add neos solver plug-in, at least for linear and quadratic problems the transformation into formats acceptable by the neos server (`https://neos-guide.org/`) should be possible.
- Add plotting methods.
- Explore possibilities of supervised solver recommendation.

We are working on extending the amount of solvers and resources available through **ROI**.

References

Adragni KP, Cook RD, Wu S (2012). "**GrassmannOptim**: An R Package for Grassmann Manifold Optimization." *Journal of Statistical Software*, **50**(5), 1–18. ISSN 1548-7660. `doi:10.18637/jss.v050.i05`. URL `http://www.jstatsoft.org/v50/i05/`.

Alizadeh F, Goldfarb D (2003). "Second-Order Cone Programming." *Mathematical Programming*, **95**(1), 3–51. ISSN 1436-4646. `doi:10.1007/s10107-002-0339-5`. URL `http://dx.doi.org/10.1007/s10107-002-0339-5`.

Andersen M, Dahl J, Liu Z, Vandenberghe L (2012). *Interior-Point Methods for Large-Scale Cone Programming*, chapter 3, pp. 55–83. MIT Press. ISBN 9780262016469. URL `https://www.seas.ucla.edu/~vandenbe/publications/mlbook.pdf`.

Andersen MS, Dahl J, Vandenberghe L (2016). *CVXOPT: A Python Package for Convex Optimization, Version 1.1.8.* URL `http://cvxopt.org/`.

ApS M (2017). *Introducing the* ***MOSEK*** *Optimization Suite. Version 8.1 (Revision 27).* URL `http://docs.mosek.com/8.1/intro/index.html`.

Ardia D, Mullen KM, Peterson BG, Ulrich J (2016). ***DEoptim****: Differential Evolution in R.* R version 2.2-4, URL `https://CRAN.R-project.org/package=DEoptim`.

Bao X, Sahinidis NV, Tawarmalani M (2011). "Semidefinite Relaxations for Quadratically Constrained Quadratic Programming: A Review and Comparisons." *Mathematical Programming*, **129**(1), 129–157. ISSN 1436-4646. doi:10.1007/s10107-011-0462-2. URL http://dx.doi.org/10.1007/s10107-011-0462-2.

based on the work of Ivan Zelinka JC (2014). ***soma**: General-Purpose Optimisation With the Self-Organising Migrating Algorithm.* R package version 1.1.1, URL https://CRAN.R-project.org/package=soma.

Bates D, Mullen KM, Nash JC, Varadhan R (2014). ***minqa**: Derivative-Free Optimization Algorithms by Quadratic Approximation.* R package version 1.2.4, URL https://CRAN.R-project.org/package=minqa.

Belotti P, Lee J, Liberti L, Margot F, Wächter A (2009). "Branching and Bounds Tightening Techniques for Non-Convex MINLP." *Optimization Methods and Software*, **24**(4-5), 597–634. doi:10.1080/10556780903087124. http://dx.doi.org/10.1080/10556780903087124, URL http://dx.doi.org/10.1080/10556780903087124.

Ben-Tal A, Nemirovski A (2001). *Lectures on Modern Convex Optimization.* Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898718829. http://epubs.siam.org/doi/pdf/10.1137/1.9780898718829, URL http://epubs.siam.org/doi/abs/10.1137/1.9780898718829.

Ben-Tal A, Nemirovski A (2015). "Lectures on Modern Convex Optimization." URL http://www2.isye.gatech.edu/~nemirovs/LMCO_LN.pdf.

Bendtsen C (2012). ***pso**: Particle Swarm Optimization.* R package version 1.0.3, URL https://CRAN.R-project.org/package=pso.

Benoist T, Estellon B, Gardi F, Megel R, Nouioua K (2011). "**LocalSolver** 1.x: A Black-Box Local-Search Solver for 0-1 Programming." *4OR*, **9**(3), 299. ISSN 1614-2411. doi:10.1007/s10288-011-0165-9. URL http://www.localsolver.com.

Benson SJ, Ye Y (2008). "Algorithm 875: **DSDP5**-Software for Semidefinite Programming." *ACM Transactions on Mathematical Software*, **34**(3), 16:1–16:20. ISSN 0098-3500. doi:10.1145/1356052.1356057. URL http://doi.acm.org/10.1145/1356052.1356057.

Bergmeir C, Molina D, Benítez JM (2016). "Memetic Algorithms with Local Search Chains in R: The **Rmalschains** Package." *Journal of Statistical Software*, **75**(4), 1–33. doi:10.18637/jss.v075.i04.

Berkelaar M (2015). ***lpSolve**: Interface to **lp_solve** v. 5.5 to Solve Linear/Integer Programs.* R package version 5.6.13, URL https://CRAN.R-project.org/package=lpSolve.

Berkelaar M, Eikland K, Notebaert P (2016). ***lp_solve** 5.5, Open Source (Mixed-Integer) Linear Programming System.* Version 5.5.2.5, URL http://lpsolve.sourceforge.net/5.5/.

Bertsimas D, King A, Mazumder R (2016). "Best Subset Selection via a Modern Optimization Lens." *The Annals of Statistics*, **44**(2), 813–852. ISSN 0090-5364, 2168-8966. doi:10.1214/15-AOS1388. URL http://projecteuclid.org/euclid.aos/1458245736.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017). "Julia: A Fresh Approach to Numerical Computing." *SIAM Review*, **59**(1), 65–98. doi:10.1137/141000671. https://doi.org/10.1137/141000671, URL https://doi.org/10.1137/141000671.

Bihorel S, Baudin M (2015). ***neldermead**: R Port of the Scilab **neldermead** Module.* R package version 1.0-10, URL https://CRAN.R-project.org/package=neldermead.

Bisschop J, Meeraus A (1982). "On the Development of a General Algebraic Modeling System in a Strategic Planning Environment." In *Applications*, volume 20 of *Mathematical Programming Studies*, pp. 1–29. Springer Berlin Heidelberg. URL http://link.springer.com/chapter/10.1007/BFb0121223.

Bixby RE (2012). "A Brief History of Linear and Mixed-Integer Programming Computation." *Documenta Mathematica*, pp. 107–121. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.592.1021&rep=rep1&type=pdf.

Bonami P, Lee J (2013). ***Bonmin** Users Manual.* Version 1.8, URL https://projects.coin-or.org/Bonmin/browser/stable/1.8/Bonmin/doc/BONMIN_UsersManual.pdf?format=raw.

Borchers B (1999). "**CSDP**, A C Library for Semidefinite Programming." *Optimization Methods and Software*, **11**(1-4), 613–623. doi:10.1080/10556789908805765. http://dx.doi.org/10.1080/10556789908805765, URL http://dx.doi.org/10.1080/10556789908805765.

Borchers HW (2016). ***adagio**: Discrete and Global Optimization Routines.* R package version 0.6.5, URL https://CRAN.R-project.org/package=adagio.

Boros E, Hammer PL (2002). "Pseudo-Boolean Optimization." *Discrete Applied Mathematics*, **123**(1–3), 155–225. doi:10.1016/S0166-218X(01)00341-9.

Bossek J (2016). ***cmaesr**: Covariance Matrix Adaptation Evolution Strategy.* R package version 1.0.3, URL https://CRAN.R-project.org/package=cmaesr.

Boyd S, Vandenberghe L (2004). *Convex Optimization.* Cambridge University Press, New York, NY, USA. ISBN 0521833787.

Braun M (2014). "**trustOptim**: An R Package for Trust Region Optimization with Sparse Hessians." *Journal of Statistical Software*, **60**(4), 1–16. doi:10.18637/jss.v060.i04. URL http://www.jstatsoft.org/v60/i04/.

Bravo HC (2016). ***Rcsdp**: R Interface to the **CSDP** Semidefinite Programming Library.* R package version 0.1.55, URL https://CRAN.R-project.org/package=Rcsdp.

Canty A, Ripley BD (2017). ***boot**: Bootstrap R (S-PLUS) Functions.* R package version 1.3-20, URL https://CRAN.R-project.org/package=boot.

Chares PR (2009). *Cones and Interior-Point Algorithms for Structured Convex Optimization Involving Powers and Exponentials.* Ph.D. thesis, Université Catholique de Louvain. URL http://dial.uclouvain.be/pr/boreal/en/object/boreal%3A28538/datastream/PDF_01/view.

Chen X, Yin X (2017). ***NlcOptim**: Solve Nonlinear Optimization with Nonlinear Constraints.* R package version 0.5, URL https://CRAN.R-project.org/package=NlcOptim.

Chi EC, Lange K (2015). "Splitting Methods for Convex Clustering." *Journal of Computational and Graphical Statistics*, **24**(4), 994–1013. ISSN 1061-8600. doi:10.1080/10618600.2014.948181. URL http://dx.doi.org/10.1080/10618600.2014.948181.

Ciupke K (2016). ***psoptim**: Particle Swarm Optimization.* R package version 1.0, URL https://CRAN.R-project.org/package=psoptim.

Conceicao ELT (2016). ***DEoptimR**: Differential Evolution Optimization in Pure R.* R package version 1.0-8, URL https://CRAN.R-project.org/package=DEoptimR.

Coppola A, Stewart B, Okazaki N (2014). ***lbfgs**: Limited-Memory BFGS Optimization.* R package version 1.2.1, URL https://CRAN.R-project.org/package=lbfgs.

Diamond S, Boyd S (2015). "Convex Optimization with Abstract Linear Operators." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 675–683. IEEE Computer Society. doi:10.1109/ICCV.2015.84. URL http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Diamond_Convex_Optimization_With_ICCV_2015_paper.pdf.

Diamond S, Boyd S (2016). "**CVXPY**: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research*, **17**(83), 1–5. URL http://jmlr.org/papers/volume17/15-408/15-408.pdf.

Domahidi A, Chu E, Boyd S (2013). "**ECOS**: An SOCP Solver for Embedded Systems." In *European Control Conference (ECC)*, pp. 3071–3076. URL https://web.stanford.edu/~boyd/papers/pdf/ecos_ecc.pdf.

Dorai-Raj S, Powell M (2006). ***powell**: Powell's UObyQA Algorithm.* R package version 1.0-0, URL https://CRAN.R-project.org/package=powell.

Eddelbuettel D (2016). ***RcppDE**: Global Optimization by Differential Evolution in C++.* R package version 0.1.5, URL https://CRAN.R-project.org/package=RcppDE.

Ferreau H, Kirches C, Potschka A, Bock H, Diehl M (2014). "**qpOASES**: A Parametric Active-Set Algorithm for Quadratic Programming." *Mathematical Programming Computation*, **6**(4), 327–363. doi:10.1007/s12532-014-0071-1.

Fine S, Scheinberg K (2001). "Efficient SVM Training Using Low-Rank Kernel Representations." *Journal of Machine Learning Research*, **2**(Dec), 243–264. URL http://www.jmlr.org/papers/volume2/fine01a/fine01a.pdf.

Fischetti M, Salvagnin D (2010). "Pruning Moves." *INFORMS Journal on Computing*, **22**(1), 108–119. doi:10.1287/ijoc.1090.0329. URL http://dx.doi.org/10.1287/ijoc.1090.0329.

Forrest J, de la Nuez D, Lougee-Heimer R (2004). ***Clp** User Guide.* URL http://www.coin-or.org/Clp/userguide/index.html.

Fourer R, Gay DM, Kernighan B (1989). "Algorithms and Model Formulations in Mathematical Programming." chapter AMPL: A Mathematical Programming Language, pp. 150–151. Springer-Verlag, New York, NY, USA. doi:https://doi.org/10.1007/978-3-642-83724-1.

Freund RM (2009). "Introduction to Semidefinite Programming (SDP)." URL http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-251j-introduction-to-mathematical-programming-fall-2009/readings/MIT6_251JF09_SDP.pdf.

Friberg HA (2014). ***Rmosek**: The R-to-**MOSEK** Optimization Interface.* R package version 1.2.5.1, URL https://CRAN.R-project.org/package=Rmosek.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441. doi:10.1093/biostatistics/kxm045.

Fritzemeier CJ, Gelius-Dietrich G (2016). ***clpAPI**: R Interface to C API of COIN-OR **Clp***. R package version 1.2.7, URL https://CRAN.R-project.org/package=clpAPI.

Fritzemeier CJ, Gelius-Dietrich G, Luangkesorn L (2015). ***glpkAPI**: R Interface to C API of **GLPK***. R package version 1.3.0, URL https://CRAN.R-project.org/package=glpkAPI.

Fu A (2017). "**cvxr**: Disciplined Convex Programming." https://github.com/anqif/cvxr.git.

Fu A, Narasimhan B (2017). ***ECOSolveR**: Embedded Conic Solver in R.* R package version 0.3-2, URL https://CRAN.R-project.org/package=ECOSolveR.

Furini F, Traversi E, Belotti P, Frangioni A, Gleixner A, Gould N, Liberti L, Lodi A, Misener R, Mittelmann H, Sahinidis N, Vigerske S, Wiegele A (2017). "QPLIB: A Library of Quadratic Programming Instances." *Technical report*, Optimization Online. Available at Optimization Online, URL http://www.optimization-online.org/DB_HTML/2017/02/5846.html.

Gay DM (1985). "Electronic Mail Distribution of Linear Programming Test Problems." *Mathematical Programming Society COAL Newsletter*, **13**, 10–12. URL ftp://lab.unb.br/pub/math/netlib/lp/data/nams.ps.gz.

Geyer CJ (2015). ***trust**: Trust Region Optimization.* R package version 0.1-7, URL https://CRAN.R-project.org/package=trust.

Geyer CJ, Meeden GD (2017). ***rcdd**: R Interface to (Some of) **cddlib***. R package version 1.2, URL https://CRAN.R-project.org/package=rcdd.

Ghalanos A, Pfaff B (2016). ***parma**: Portfolio Allocation and Risk Management Applications.* R package version 1.5-3, URL https://CRAN.R-project.org/package=parma.

Ghalanos A, Theussl S (2015). ***Rsolnp**: General Non-linear Optimization Using Augmented Lagrange Multiplier Method.* R package version 1.16.

Goldfarb D, Idnani A (1983). "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs." *Mathematical Programming*, **27**(1), 1–33. ISSN 1436-4646. doi:10.1007/BF02591962. URL http://dx.doi.org/10.1007/BF02591962.

Gomory RE (1960). "An Algorithm for the Mixed-Integer Problem." *Technical report*, The RAND Corporation. oai: AD0616505.

Grant M, Boyd S (2014). "**CVX**: MATLAB Software for Disciplined Convex Programming, Version 2.1." `http://cvxr.com/cvx`.

Gurobi Optimization I (2016). "**Gurobi** Optimizer Reference Manual." URL `https://www.gurobi.com`.

Hankin RKS (2006). "Special Functions in R: Introducing the **gsl** Package." *R News*, **6**(4), 24–26. URL `https://cran.r-project.org/doc/Rnews/Rnews_2006-4.pdf`.

Helmberg C (2000). "Semidefinite Programming for Combinatorial Optimization." URL `https://opus4.kobv.de/opus4-zib/files/602/ZR-00-34.pdf`.

Hocking TD, Joulin A, Bach F, Vert JP (2011). "Clusterpath: An Algorithm for Clustering Using Convex Fusion Penalties." In L Getoor, T Scheffer (eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pp. 745–752. ACM, New York, NY, USA. ISBN 978-1-4503-0619-5. URL `http://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.228.7220`.

Hornik K (2017). ***clue**: Cluster Ensembles.* R package version 0.3-53, URL `https://CRAN.R-project.org/package=clue`.

Hornik K, Harter R, Theußl S (2017a). ***Rsymphony**: Symphony in R.* R package version 0.1-27, URL `https://CRAN.R-project.org/package=Rsymphony`.

Hornik K, Meyer D, Buchta C (2016). ***slam**: Sparse Lightweight Arrays and Matrices.* R package version 0.1-40, URL `https://CRAN.R-project.org/package=slam`.

Hornik K, Meyer D, Schwendinger F (2017b). ***ROI.plugin.msbinlp**: Multi-Solution Binary Linear Problem Plug-in for the R Optimization Infrastructure.* R package version 0.3-0, URL `https://CRAN.R-project.org/package=ROI.plugin.msbinlp`.

ILOG I (2015). "**CPLEX** Optimization Studio **CPLEX** User's Manual." URL `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

Johnson SG (2016). "The **NLopt** Nonlinear-Optimization Package." URL `http://ab-initio.mit.edu/nlopt`.

Kallrath J (2004). *Mathematical Optimization and the Role of Modeling Languages*, chapter 1, pp. 3–24. Springer-Verlag, Boston, MA. ISBN 978-1-4613-0215-5. `doi:10.1007/978-1-4613-0215-5_1`. URL `http://dx.doi.org/10.1007/978-1-4613-0215-5_1`.

Karatzoglou A, Smola A, Hornik K (2016). ***kernlab**: Kernel-Based Machine Learning Lab.* R package version 0.9-25, URL `https://CRAN.R-project.org/package=kernlab`.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "**kernlab** - An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(1), 1–20. ISSN 1548-7660. `doi:10.18637/jss.v011.i09`. URL `https://www.jstatsoft.org/index.php/jss/article/view/v011i09`.

King AA (2017). ***subplex**: Unconstrained Optimization Using the Subplex Algorithm.* R package version 1.4-1, URL https://CRAN.R-project.org/package=subplex.

Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2011). "MIPLIB 2010." *Mathematical Programming Computation*, **3**(2), 103–163. doi:10.1007/s12532-011-0025-9. URL http://mpc.zib.de/index.php/MPC/article/view/56/28.

Koecher M (1957). "Positivitätsbereiche im R^n." *American Journal of Mathematics*, **79**(3), 575–596. ISSN 00029327, 10806377. doi:10.2307/2372563. URL http://www.jstor.org/stable/2372563.

Konen W, Hansen N (2015). ***rCMA**: R-to-Java Interface for 'CMA-ES'.* R package version 1.1, URL https://CRAN.R-project.org/package=rCMA.

Konis K (2016). ***lpSolveAPI**: R Interface to lp_solve 5.5.2.0-17.* R package version 5.5.2.0-17, URL https://CRAN.R-project.org/package=lpSolveAPI.

Krone-Martins A (2017). ***RCEIM**: R Cross Entropy Inspired Method for Optimization.* R package version 0.3, URL https://CRAN.R-project.org/package=RCEIM.

Land AH, Doig AG (1960). "An Automatic Method for Solving Discrete Programming Problems." *Econometrica*, **28**(3), 497–520. ISSN 00129682, 14680262. doi:10.2307/1910129. URL http://www.jstor.org/stable/1910129.

Lauritzen SL (1996). *Graphical Models*, volume 17. Clarendon Press.

Linderoth JT, Ralphs TK (2005). *Noncommercial Software for Mixed-Integer Linear Programming*, chapter 10, pp. 253–303. Operations Research Series. CRC Press. ISBN 978-0-8493-1914-3. doi:10.1201/9781420039597.ch10. URL http://dx.doi.org/10.1201/9781420039597.ch10.

Lindo Systems I (2003). "**Lindo** User's Manual." URL http://www.lindo.com/.

Lindsten F, Ohlsson H, Ljung L (2011). "Just Relax and Come Clustering!: A Convexification of K-Means Clustering." *Technical Report 2992*, Linköping University, The Institute of Technology. diva: diva2:650707.

Lobo MS, Vandenberghe L, Boyd S, Lebret H (1998). "Applications of Second-Order Cone Programming." *Linear Algebra and its Applications*, **284**(1–3), 193–228. ISSN 0024-3795. doi:http://dx.doi.org/10.1016/S0024-3795(98)10032-0.

Löfberg J (2004). "**YALMIP**: A Toolbox for Modeling and Optimization in MATLAB." In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pp. 284–289. IEEE. doi:10.1109/CACSD.2004.1393890.

Lubin M, Dunning I (2015). "Computing in Operations Research Using Julia." *INFORMS Journal on Computing*, **27**(2), 238–248. doi:10.1287/ijoc.2014.0623. URL http://dx.doi.org/10.1287/ijoc.2014.0623.

Lumley T (2017). ***leaps**: Regression Subset Selection.* R package version 3.0, URL https://CRAN.R-project.org/package=leaps.

Makhorin A (2011). *GNU Linear Programming Kit Reference Manual Version 4.47.* URL http://www.gnu.org/software/glpk.

Mebane, Jr WR, Sekhon JS (2011). "Genetic Optimization Using Derivatives: The **rgenoud** Package for R." *Journal of Statistical Software*, **42**(11), 1–26. ISSN 1548-7660. doi:10.18637/jss.v042.i11. URL https://www.jstatsoft.org/v042/i11.

Meinshausen N, Bühlmann P (2006). "High-Dimensional Graphs and Variable Selection with the Lasso." *The Annals of Statistics*, **34**(3), 1436–1462. doi:doi:10.1214/009053606000000281.

Meyer D (2015). ***registry**: Infrastructure for R Package Registries.* R package version 0.3, URL https://CRAN.R-project.org/package=registry.

Meyer D, Hornik K (2017). ***relations**: Data Structures and Algorithms for Relations.* R package version 0.6-7, URL https://CRAN.R-project.org/package=relations.

Miller A (2002). *Subset Selection in Regression.* CRC Press. doi:10.1201/9781420035933. URL https://www.crcpress.com/Subset-Selection-in-Regression/Miller/p/book/9781584881711.

Mullen K (2014a). "Continuous Global Optimization in R." *Journal of Statistical Software*, **60**(1), 1–45. ISSN 1548-7660. doi:10.18637/jss.v060.i06. URL http://www.jstatsoft.org/index.php/jss/article/view/v060i06.

Mullen K (2014b). ***globalOptTests**: Objective Functions for Benchmarking the Performance of Global Optimization Algorithms.* R package version 1.1, URL https://CRAN.R-project.org/package=globalOptTests.

Nash JC (2014a). "On Best Practice Optimization Methods in R." *Journal of Statistical Software*, **60**(1), 1–14. ISSN 1548-7660. doi:10.18637/jss.v060.i02. URL https://www.jstatsoft.org/index.php/jss/article/view/v060i02.

Nash JC (2014b). ***Rcgmin**: Conjugate Gradient Minimization of Nonlinear Functions.* R package version 2013-2.21, URL https://CRAN.R-project.org/package=Rcgmin.

Nash JC (2017). ***Rvmmin**: Variable Metric Nonlinear Function Minimization.* R package version 2017-7.18, URL https://CRAN.R-project.org/package=Rvmmin.

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: **optimx** for R." *Journal of Statistical Software*, **43**(9), 1–14. ISSN 1548-7660. doi:10.18637/jss.v043.i09. URL http://www.jstatsoft.org/v43/i09.

Nash JC, Zhu C, Byrd R, Nocedal J, Morales JL (2015). ***lbfgsb3**: Limited Memory BFGS Minimizer with Bounds on Parameters.* R package version 2015-2.13, URL https://CRAN.R-project.org/package=lbfgsb3.

Nemirovski A (2004). "Interior Point Polynomial Time Methods in Convex Programming." URL http://www2.isye.gatech.edu/~nemirovs/Lect_IPM.pdf.

Nemirovski A (2006). "Advances in Convex Optimization: Conic Programming." In *In Proceedings of International Congress of Mathematicians*, pp. 413–444. ISSN 978-3-03719-522-2. doi:10.4171/022. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.539.6900.

Nesterov Y (2004). *Introductory Lectures on Convex Optimization.* Springer-Verlag. doi:10.1007/978-1-4419-8853-9. URL https://doi.org/10.1007/978-1-4419-8853-9.

Nielsen HB, Mortensen SB (2016). ***ucminf**: General-Purpose Unconstrained Non-Linear Optimization.* R package version 1.1-4, URL https://CRAN.R-project.org/package=ucminf.

Nocedal J, Wright SJ (2006). *Numerical Optimization.* Springer Series in Operations Research and Financial Engineering. Springer-Verlag. ISBN 978-0387-30303-1. doi:10.1007/978-0-387-40065-5.

O'Donoghue B (2015). "**SCS** - (Splitting Conic Solver)." https://github.com/cvxgrp/scs.git.

O'Donoghue B, Chu E, Parikh N, Boyd S (2016). "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding." *Journal of Optimization Theory and Applications*, pp. 1–27. ISSN 1573-2878. doi:10.1007/s10957-016-0892-3. URL http://dx.doi.org/10.1007/s10957-016-0892-3.

O'Donoghue B, Schwendinger F (2016). ***scs**: Splitting Conic Solver.* R package version 1.1-1, URL https://CRAN.R-project.org/package=scs.

Ormerod JT, Wand MP (2014). ***LowRankQP**: Low Rank Quadratic Programming.* R package version 1.0.2, URL https://CRAN.R-project.org/package=LowRankQP.

Pelckmans K, De Brabanter J, De Moor B, Suykens J (2005). "Convex Clustering Shrinkage." In *Workshop on Statistics and optimization of clustering Workshop (PASCAL)*. URL https://lirias.kuleuven.be/handle/123456789/181608.

Perez RE, Jansen PW, Martins JRRA (2012). "**pyOpt**: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization." *Structures and Multidisciplinary Optimization*, **45**(1), 101–118. doi:10.1007/s00158-011-0666-3.

Pfaff B (2015). ***cccp**: Cone Constrained Convex Problems.* R package version 0.2-4, URL https://CRAN.R-project.org/package=cccp.

Python Software Foundation (2017). *Python Language Reference, Version 2.7.* Wilmington, DE. URL http://www.python.org/.

Racine JS, Nie Z (2017). ***crs**: Categorical Regression Splines.* R package version 0.15-29, URL https://CRAN.R-project.org/package=crs.

Ralphs TK, Güzelsoy M (2005). *The **SYMPHONY** Callable Library for Mixed Integer Programming*, chapter 2, pp. 61–76. Springer-Verlag. ISBN 978-0-387-23529-5. doi:10.1007/0-387-23529-9_5. URL http://dx.doi.org/10.1007/0-387-23529-9_5.

Ralphs TK, Güzelsoy M (2011). ***SYMPHONY** 5.4.0 User's Manual.* Department of Industrial and Systems Engineering, Lehigh University. URL http://www.coin-or.org/SYMPHONY/man-5.4/.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org`.

Rosenbrock HH (1960). "An Automatic Method for Finding the Greatest or Least Value of a Function." *The Computer Journal*, **3**(3), 175. `doi:10.1093/comjnl/3.3.175`. `/oup/backfile/Content_public/Journal/comjnl/3/3/10.1093/comjnl/3.3.175/2/030175.pdf`, URL `http://dx.doi.org/10.1093/comjnl/3.3.175`.

Rudy J (2011). ***CLSOCP**: A Smoothing Newton Method SOCP Solver*. R package version 1.0, URL `https://CRAN.R-project.org/package=CLSOCP`.

Satman MH (2013). "Machine Coded Genetic Algorithms For Real Parameter Optimization Problems." *Gazi University Journal of Science*, **26**(1), 85–95. ISSN 2147-1762. URL `http://gujs.gazi.edu.tr/article/view/1060000982`.

Schumann E (2017). ***NMOF**: Numerical Methods and Optimization in Finance*. R package version 1.2-2, URL `https://CRAN.R-project.org/package=NMOF`.

Schwendinger F (2016). ***ROI.models.netlib**: **ROI** Optimization Problems Based on NETLIB-LP*. R package version 1.0, URL `https://CRAN.R-project.org/package=ROI.models.netlib`.

Schwendinger F (2017). ***ROI.models.globalOptTests**: **ROI** Optimization Problems based on **globalOptTests***. R package version 1.0, URL `https://CRAN.R-project.org/package=ROI.models.globalOptTests`.

Schwendinger F, Theussl S (2017). ***ROI.models.miplib**: R Optimization Infrastructure: MIPLIB 2010 Benchmark Instances*. R package version 0.0-1, URL `https://CRAN.R-project.org/package=ROI.models.miplib`.

Scrucca L (2013). "**GA**: A Package for Genetic Algorithms in R." *Journal of Statistical Software*, **53**(4), 1–37. ISSN 1548-7660. `doi:10.18637/jss.v053.i04`. URL `http://www.jstatsoft.org/v53/i04/`.

Serrano SA (2015). *Algorithms for Unsymmetric Cone Optimization and an Implementation for Problems with the Exponential Cone*. Ph.D. thesis, Stanford University. URL `https://web.stanford.edu/group/SOL/dissertations/ThesisAkleAdobe-augmented.pdf`.

Tan KM, Witten D, *et al.* (2015). "Statistical Properties of Convex Clustering." *Electronic Journal of Statistics*, **9**(2), 2324–2347. `doi:10.1214/15-EJS1074`.

Tang J, He G, Dong L, Fang L (2012). "A New One-Step Smoothing Newton Method for Second-Order Cone Programming." *Applications of Mathematics*, **57**(4), 311–331. `doi:10.1007/s10492-012-0019-6`. URL `http://dx.doi.org/10.1007/s10492-012-0019-6`.

The MathWorks Inc (2017). *MATLAB - The Language of Technical Computing*. Version R2017b, URL `https://www.mathworks.com/products/matlab/`.

Theussl S (2017). ***ROI.plugin.glpk**: **ROI** Plug-in **GLPK***. R package version 0.3-0, URL `https://CRAN.R-project.org/package=ROI.plugin.glpk`.

Theußl S, Borchers HW (2017). "CRAN Task View: Optimization and Mathematical Programming." URL `https://CRAN.R-project.org/view=Optimization`.

Theußl S, Bravo HC (2016). ***Rcplex**: R Interface to **CPLEX***. R package version 0.3-3, URL `https://R-Forge.R-project.org/projects/rcplex`.

Theußl S, Hornik K (2017). ***Rglpk**: R/GNU Linear Programming Kit Interface.* R package version 0.6-3, URL `https://CRAN.R-project.org/package=Rglpk,http://www.gnu.org/software/glpk/`.

Theußl S, Schwendinger F, Hornik K, Meyer D (2017). ***ROI**: R Optimization Infrastructure.* R package version 0.3-0, URL `https://CRAN.R-project.org/package=ROI,https://R-Forge.R-project.org/projects/roi/`.

Theußl S, Zeileis A (2009). "Collaborative Software Development Using R-Forge." *The R Journal*, **1**(1), 9–14. URL `https://journal.R-project.org/archive/2009-1/RJournal_2009-1_Theussl+Zeileis.pdf`.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B*, **58**(1), 267–288. URL `http://www.jstor.org/stable/2346178`.

Trautmann H, Mersmann O, Arnu D (2011). ***cmaes**: Covariance Matrix Adapting Evolutionary Strategy.* R package version 1.0-11, URL `https://CRAN.R-project.org/package=cmaes`.

Turlach BA, Weingessel A (2013). ***quadprog**: Functions to Solve Quadratic Programming Problems.* R package version 1.5-5, URL `https://CRAN.R-project.org/package=quadprog`.

Udell M, Mohan K, Zeng D, Hong J, Diamond S, Boyd S (2014). "Convex Optimization in Julia." In *Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages*, HPTCDL '14, pp. 18–28. IEEE Press, Piscataway, NJ, USA. ISBN 978-1-4799-7020-9. `doi:10.1109/HPTCDL.2014.5`. URL `http://dx.doi.org/10.1109/HPTCDL.2014.5`.

Vandenberghe L, Boyd S (1996a). "Semidefinite Programming." *SIAM Review*, **38**(1), 49–95. `doi:10.1137/1038003`. `https://doi.org/10.1137/1038003`, URL `https://doi.org/10.1137/1038003`.

Vandenberghe L, Boyd S (1996b). "Semidefinite Programming." *SIAM review*, **38**(1), 49–95. ISSN 0036-1445. `doi:10.1137/1038003`.

Varadhan R (2015). ***alabama**: Constrained Nonlinear Optimization.* R package version 2015.3-1, URL `https://CRAN.R-project.org/package=alabama`.

Varadhan R, Gilbert P (2009). "**BB**: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function." *Journal of Statistical Software, Articles*, **32**(4), 1–26. ISSN 1548-7660. `doi:10.18637/jss.v032.i04`. URL `https://www.jstatsoft.org/v032/i04`.

Varadhan R, University JH, Borchers HW, Research AC (2016). ***dfoptim**: Derivative-Free Optimization*. R package version 2016.7-1, URL https://CRAN.R-project.org/package=dfoptim.

Wächter A, Biegler LT (2006). "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical Programming*, **106**(1), 25–57. ISSN 1436-4646. doi:10.1007/s10107-004-0559-y. URL https://doi.org/10.1007/s10107-004-0559-y.

Wuertz D (2007). ***Rdonlp2**: An R Extension Library to use Peter Spelluci's **DONLP2** from R*. R package version 3002.10, URL https://r-forge.r-project.org/projects/rmetrics/.

Wuertz D (2014). ***Rnlminb2**: An R Extension Library for Constrained Optimization with nlminb*. R package version 3002.10, URL https://r-forge.r-project.org/projects/rmetrics/.

Xiang Y, Gubian S, Suomela B, Hoeng J (2013). "Generalized Simulated Annealing for Global Optimization: The **GenSA** Package." *The R Journal Volume 5/1, June 2013*, **5**(1), 13–28. URL https://journal.r-project.org/archive/2013/RJ-2013-002/index.html.

Ypma J (2011). ***ipoptr**: R Interface to **Ipopt***. R package version 0.8.4, URL http://r-forge.r-project.org/projects/ipoptr/.

Ypma J, Borchers HW, Eddelbuettel D (2017). ***nloptr**: R Interface to **NLopt***. R package version 1.0.4, URL https://CRAN.R-project.org/package=nloptr.

Zambrano-Bigiarini M, Rojas R (2013). "A Model-Independent Particle Swarm Optimisation Software for Model Calibration." *Environmental Modelling & Software*, **43**(Supplement C), 5–25. ISSN 1364-8152. doi:https://doi.org/10.1016/j.envsoft.2013.01.004. URL http://www.sciencedirect.com/science/article/pii/S1364815213000133.

Zhisu Zhu YY (2016). ***Rdsdp**: R Interface to the **DSDP** Semidefinite Programming Library.* R package version 1.0.4-2, URL https://CRAN.R-project.org/package=Rdsdp.

Zhu C, Xu H, Leng C, Yan S (2014). "Convex Optimization Procedure for Clustering: Theoretical Revisit." In Z Ghahramani, M Welling, C Cortes, N Lawrence, K Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 1619–1627. Curran Associates, Inc. URL http://papers.nips.cc/paper/5307-convex-optimization-procedure-for-clustering-theoretical-revisit.pdf.

A. Tables

	Package	Library	Objective	Constraints	Mixed Integer
1	**alabama**		functional	nonlinear	No
2	**BB**		functional	no	No
3	**cccp**		conic	conic	No
4	**clpAPI**	clp	linear	linear	No
5	**CLSOCP**		conic	conic	No
6	**clue:::sumt**		functional	nonlinear	No
7	**DEoptim**		functional	box	No
8	**dfoptim**		functional	box	No
9	**ECOSolveR**	ECOS	conic	conic	Yes
10	**GenSA**		functional	box	No
11	**glpkAPI, Rglpk**	GLPK	linear	linear	Yes
12	**kernlab:::ipop**		quadratic	linear	No
13	**lbfgsb3**		functional	box	No
14	**LowRankQP**		quadratic	linear	No
15	**lpSolve, lpSolveAPI**	lp_solve	linear	linear	Yes
16	**minqa**		functional	box	No
17	**NlcOptim**		functional	nonlinear	No
18	**nloptr**	NLopt	functional	nonlinear	No
19	**optimx**		functional	box	No
20	**quadprog**		quadratic	linear	No
21	**rcdd**	cddlib	linear	linear	No
22	**Rcgmin**		functional	box	No
23	**Rcsdp**	CSDP	conic	conic	No
24	**Rdsdp**	DSDP	conic	conic	No
25	**rgenoud**		functional	box	No
26	**Rmalschains**		functional	box	No
27	**Rsolnp**		functional	nonlinear	No
28	**Rsymphony**	symphony	linear	linear	Yes
29	**Rvmmin**		functional	box	No
30	**scs**	scs	conic	conic	No
31	**soma**		functional	box	No
32	**stats**		functional	box	No
33	**trustOptim**		functional	no	No
34	**ucminf**		functional	box	No

Table 5: Overview optimization packages in R.

	Method	Package	Type	Constraint	G	H	J
1	auglag	**alabama**	local	nonlinear	Yes	No	Yes
2	dfsane	**BB**	local	no	No	No	No
3	sumt	**clue**	local	nonlinear	Yes	No	No

4	DEoptim	**DEoptim**	global	box	No	No	No
5	hjkb	**dfoptim**	local	box	No	No	No
6	nmk	**dfoptim**	local	box	No	No	No
7	GenSA	**GenSA**	global	box	No	No	No
8	LBFGSB3	**lbfgsb3**	local	box	Yes	No	No
9	SANN	**stats**	global	no	No	No	No
10	Nelder-Mead	**stats / optimx**	local	no	No	No	No
11	BFGS	**stats / optimx**	local	no	Yes	No	No
12	L-BFGS-B	**stats / optimx**	local	box	Yes	No	No
13	CG	**stats / optimx**	local	no	Yes	No	No
14	nlminb	**stats / optimx**	local	box	Yes	Yes	No
15	nlm	**stats / optimx**	local	no	Yes	Yes	No
16	ucminf	**ucminf / optimx**	local	box	Yes	No	No
17	uobyqa	**minqa / optimx**	local	no	No	No	No
18	newuoa	**minqa / optimx**	local	no	No	No	No
19	bobyqa	**minqa / optimx**	local	box	No	No	No
20	Rcgmin	**Rcgmin / optimx**	local	box	Yes	No	No
21	Rvmmin	**Rvmmin / optimx**	local	box	Yes	No	No
22	spg	**BB / optimx**	local	box	Yes	No	No
23	NlcOptim	**NlcOptim**	local	nonlinear	No	No	No
24	auglag	**nloptr**	local	nonlinear	Yes	No	Yes
25	bobyqa	**nloptr**	local	box	No	No	No
26	cobyla	**nloptr**	local	nonlinear	No	No	No
27	DIRECT	**nloptr**	global	box	No	No	No
28	isres	**nloptr**	global	nonlinear	No	No	No
29	lbfgs	**nloptr**	local	box	Yes	No	No
30	mlsl	**nloptr**	global	box	Yes	No	No
31	mma	**nloptr**	local	nonlinear	Yes	No	Yes
32	nedlermead	**nloptr**	local	box	No	No	No
33	newuoa	**nloptr**	local	no	No	No	No
34	sbplx	**nloptr**	local	box	No	No	No
35	slsqp	**nloptr**	local	nonlinear	Yes	No	Yes
36	stogo	**nloptr**	global	box	Yes	No	No
37	tnewton	**nloptr**	local	box	Yes	No	No
38	varmetric	**nloptr**	local	box	Yes	No	No
39	genoud	**rgenoud**	global	box	Yes	No	No
40	solnp	**Rsolnp**	local	nonlinear	No	No	No
41	MA-LS-Chains	**Rmalschains**	global	box	No	No	No
42	soma	**soma**	global	box	No	No	No
43	trustOptim	**trustOptim**	local	no	Yes	Yes	No
44	DEoptim	**RcppDE**	global	box	No	No	No
45	JDEoptim	**DEoptimR**	global	nonlinear	No	No	No
46	ga	**GA**	global	box	No	No	No
47	mcga	**mcga**	global	box	No	No	No
48	mcga2	**mcga**	global	box	No	No	No
49	psoptim	**pso**	global	box	Yes	No	No
50	psoptim	**psoptim**	global	box	No	No	No

51	cma_es	**cmaes**	global	box	No	No	No
52	cmaes	**cmaesr**	global	no	No	No	No
53	cmaes	**parma**	global	box	No	No	No
54	GAopt	**NMOF**	global	no	No	No	No
55	DEopt	**NMOF**	global	box	No	No	No
56	LSopt	**NMOF**	global	no	No	No	No
57	PSopt	**NMOF**	global	box	No	No	No
58	TAopt	**NMOF**	global	no	No	No	No
59	GrassmannOptim	**GrassmannOptim**	local	no	Yes	No	No
60	lbfgs	**lbfgs**	local	no	Yes	No	No
61	powell	**powell**	local	no	No	No	No
62	ceimOpt	**RCEIM**	local	box	No	No	No
63	subplex	**subplex**	local	no	No	No	No
64	ipoptr	**ipoptr**	local	nonlinear	Yes	Yes	Yes
65	Rdonlp2	**Rdonlp2**	local	nonlinear	No	No	No
66	Rnlminb2	**Rnlminb2**	local	nonlinear	Yes	Yes	No
67	CMAES	**adagio**	global	box	No	No	No
68	snomadr	**crs**	local	nonlinear	No	No	No
69	multimin	**gsl**	local	no	Yes	No	No
70	hydroPSO	**hydroPSO**	global	box	No	No	No
71	neldermead	**neldermead**	local	nonlinear	No	No	No
72	cmaOptimDP	**rCMA**	local	nonlinear	No	No	No
73	trust	**trust**	local	no	Yes	Yes	No

Table 6: GPS in R and their capability to handle type, constraint, gradient (G), Hessian (H), Jacobian (J) information.

B. Best subset selection

```
R> subset_selection <- function(A, b, k, beta_lb = -1000, beta_ub = 1000,
+                              count_intercept = FALSE, solver = "auto",
+                              ...) {
+   control <- list(...)
+   Q <- 2 * t(A) %*% A
+   L <- -2 * t(b) %*% A
+   x <- OP(objective = Q_objective(Q=Q, L=L),
+           bounds = V_bound(li = seq_len(nrow(Q)),
+                            lb = rep.int(-Inf, nrow(Q))))
+   y <- ROI_reformulate(x, "socp")
+   n <- length(objective(x))
+   x <- OP(objective = c(terms(objective(y))$L, double(n)))
+   L <- constraints(y)$L
+   L <- cbind(L, simple_triplet_zero_matrix(nrow(L), n))
+
+   if ( length(beta_lb) == 1L ) beta_lb <- rep.int(beta_lb, n)
+   if ( length(beta_lb) == 1L ) beta_ub <- rep.int(beta_ub, n)
+
+   LB <- cbind(simple_triplet_diag_matrix(-1, n),
+               simple_triplet_zero_matrix(n, 1),
+               simple_triplet_diag_matrix(beta_lb, n))
+   UB <- cbind(simple_triplet_diag_matrix(1, n),
+               simple_triplet_zero_matrix(n, 1),
+               simple_triplet_diag_matrix(-beta_ub, n))
+
+   if (count_intercept) {
+     SUM <- cbind(simple_triplet_zero_matrix(1, n+1), matrix(1, 1, n))
+   } else {
+     SUM <- cbind(simple_triplet_zero_matrix(1, n+2), matrix(1, 1, n-1))
+   }
+
+   constraints(x) <- C_constraint(rbind(L, LB, UB, SUM),
+                                  c(constraints(y)$cones,
+                                    K_lin(n), K_lin(n), K_lin(1L)),
+                                  c(constraints(y)$rhs, double(n),
+                                    double(n), k))
+
+   types(x) <- c(rep.int("C", length(objective(y))), rep.int("B", n))
+   len <- length(objective(x))
+   bounds(x) <- V_bound(li = seq_len(len), lb = rep.int(-Inf, len))
+   maximum(x) <- maximum(y)
+
+   if ( isTRUE(control$dry_run) )
+     return(x)
+    z <- ROI_solve(x)
+    head(solution(z), n)
+  }
```

C. SON clustering

```
R> convex_clust <- function(A, gamma, solver = "auto", control = list()) {
+    m <- nrow(A)
+    n <- ncol(A)
+    ncombn <- ( m * (m-1) / 2 )
+    k <- ncombn * (n+1)
+    obj <- c(rep.int(0, n * m), 1/2, rep.int(gamma, ncombn))
+    b <- c(1, -1, 2*as.double(A), rep.int(0, k))
+    L <- simple_triplet_zero_matrix(nrow = 2 + n * m + k,
+                                    ncol = n * m + 1 + ncombn)
+    L[1, n*m+1] <- -1
+    L[2, n*m+1] <- -1
+    L[2+seq_len(n * m), seq_len(n * m)] <- diag(2, n * m)
+
+    ko <- combn(seq_len(m), 2)
+    M <- matrix(seq_len(n*m), m, n, byrow=FALSE)
+    irow <- n * m + 3
+    cones <- K_soc(n*m+2)
+    for ( i in seq_len(ncol(ko)) ) {
+      L[irow, n * m + 1 + i] <- -1
+      cones <- c(cones, K_soc(n+1))
+      irow <- irow + 1
+      for (j in seq_len(n)) {
+        L[irow, M[ko[,i], j]] <- c(-1, 1)
+        irow <- irow + 1
+      }
+    }
+
+    op <- OP(objective=L_objective(obj),
+             constraints = C_constraint(L=L, cones=cones, rhs=b),
+             bounds = V_bound(ld = -Inf, nobj = nrow(L)))
+    if ( isTRUE(control$dry_run) )
+      return(op)
+
+    x <-  ROI_solve(op, solver, control)
+    matrix(x$solution[seq_len(n*m)], m, n)
+  }
```

D. Graphical lasso

```
R> index_to_vech <- function(i, j, n) {
+     ind_to_vech <- function(i, j, n) {
+         if (j > i) return(NA)
+         (j - 1) * n + i - (j - 1) * j / 2
+     }
+     unlist(mapply(ind_to_vech, i, j, MoreArgs = list(n = n),
+                   SIMPLIFY = FALSE, USE.NAMES = FALSE),
+            recursive = FALSE, use.names = FALSE)
+ }

R> ROI_glasso <- function(s, rho, solver = "auto", ...) {
+     stopifnot(nrow(s) > 1)
+     control <- list(...)
+     stm <- simple_triplet_matrix
+     n <- nrow(s); nv <- (n + 1) * n / 2; nij <- choose(n, 2)
+     ndzx <- (2 * n + 1) * 2 * n / 2
+     seqn <- seq_len(n); seqnv <- seq_len(nv); seqnij <- seq_len(nij)
+
+     obj <- c(as.vector(vech(s + s * lower.tri(s))), double(ndzx))
+     A <- simple_triplet_diag_matrix(-1, nv + ndzx)
+     cones <- K_psd(c(nv, ndzx))
+
+     ij <- combn(seqn, 2)
+     k <- index_to_vech(n + ij[1,], ij[2,], 2 * n)
+     Z_UPPER <- stm(seqnij, nv + k, rep.int(1, nij), nrow = nij,
+                    ncol = nv + ndzx)
+     cones <- c(cones, K_zero(nij))
+
+     k <- index_to_vech(ij[2,], ij[1,], 2 * n)
+     D_DIAG <- stm(seqnij, nv + k, rep.int(1, nij), nrow = nij,
+                   ncol = nv + ndzx)
+     cones <- c(cones, K_zero(nij))
+
+     kd <- index_to_vech(seqn, seqn, 2 * n)
+     kz <- index_to_vech(n + seqn, seqn, 2 * n)
+     EQ_DIAG <- stm(c(seqn, seqn), nv + c(kd, kz),
+                    c(rep.int(-1, n), rep.int(1, n)),
+                    nrow = n, ncol = nv + ndzx)
+     cones <- c(cones, K_zero(n))
+     A <- rbind(A, rbind(Z_UPPER, D_DIAG, EQ_DIAG))
+
+     EQ_X <- stm(c(seqnv, seqnv), c(seqnv, ndzx + seqnv),
+                 c(rep.int(-1, nv), rep.int(1, nv)),
+                 nrow = nv, ncol = ncol(A))
+      cones <- c(cones, K_zero(nrow(EQ_X)))
+      A <- rbind(A, EQ_X)
+      rhs <- double(nrow(A))
+
+      obj <- c(obj, rep.int(-1, n))
+      j <- nv + kd
+      LOG <- stm(c(3 * seqn, 3 * seqn - 2), c(j, ncol(A) + seqn),
+                 rep.int(-1, 2 * n), nrow = 3 * n, ncol = ncol(A) + n)
+      A <- rbind(cbind(A, simple_triplet_zero_matrix(nrow(A), n)), LOG)
+      cones <- c(cones, K_expp(n))
+      rhs <- c(rhs, rep.int(c(0, 1, 0), n))
+
+      rho_matrix <- matrix(rho, n, n)
+      obj <- c(obj, vech(rho_matrix + rho_matrix * lower.tri(rho_matrix)))
+      NORM <- stm(i = c(seq_len(2 * nv), seq_len(2 * nv)),
+                  j = c(seqnv, seqnv, ncol(A) + seqnv, ncol(A) + seqnv),
+                  v = c(rep.int(1, nv), rep.int(-1, 3 * nv)),
+                  nrow = 2 * nv, ncol = ncol(A) + nv)
+      A <- rbind(cbind(A, simple_triplet_zero_matrix(nrow(A), nv)), NORM)
+      cones <- c(cones, K_lin(2 * nv))
+      rhs <- c(rhs, double(2 * nv))
+
+      model <- OP(objective = L_objective(obj),
+                  constraints = C_constraint(A, cones = cones, rhs = rhs),
+                  bounds = V_bound(ld = -Inf, nobj = length(obj)))
+
+      if ( isTRUE(control$dry_run) )
+          return(model)
+
+      so <- ROI_solve(model, solver=solver, control)
+      Y <- matrix(0, n, n)
+      Y[lower.tri(Y, diag=TRUE)] <- so$solution[seq_len(n * (n+1) / 2)]
+      Y[upper.tri(Y)] <- t(Y)[upper.tri(Y)]
+      list(w=chol2inv(chol(Y)), wi=Y,
+           errflag=so$status$code, niter=so$message$info$iter)
+  }
```

E. Abbreviations

Abbreviation	Full Name
BLP	Binary Linear Programming
CP	Conic Programming
IP	Integer Programming
LP	Linear Programming
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
NLP	Nonlinear Programming
QCQP	Quadraticly Constraint Quadratic Programming
QP	Quadratic Programming
SDP	Semidefinite Programming
SOCP	Second Order Cone Programming

Affiliation:

Stefan Theußl
Raiffeisen Bank International AG
Market Strategy/Quant Research
Am Stadtpark 9
1030 Vienna, Austria
E-mail: `Stefan.Theussl@R-project.org`

Florian Schwendinger, Kurt Hornik
Department of Finance, Accounting and Statistics
Institute for Statistics and Mathematics
WU Vienna University of Economics and Business
Welthandelsplatz 1, Building D4, Level 4
1020 Vienna, Austria
E-mail: `FlorianSchwendinger@gmx.at`, `Kurt.Hornik@R-project.org`