Journal of Statistical Software

August 2020, Volume 94, Issue 15. doi: 10.18637/jss.v094.i15

ROI: An Extensible R Optimization Infrastructure

Stefan Theußl
Raiffeisen Bank International AG

Florian Schwendinger
WU Wirtschaftsuniversität Wien

Kurt Hornik
WU Wirtschaftsuniversität 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 optimization algorithms, designed to be only applicable within a specific application. However, in many instances recent advances, in particular in the field of convex optimization, make it possible to conveniently and straightforwardly use modern solvers instead with the advantage of enabling broader usage scenarios and thus promoting reusability. This paper introduces the R optimization infrastructure **ROI** 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, thanks 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 2020), 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 already has 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 (AMLs; e.g., GAMS, Bisschop and Meeraus 1982, and AMPL, Fourer, Gay, and Kernighan 1989) became available. AMLs are domain specific languages (DSLs) 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. 2019), Python (Python Software Foundation 2017) or R. Among the modern optimization systems, many 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, Narasimhan, and Boyd 2020) 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, Borchers, and Schwendinger 2020), it is still commonplace to develop highly sophisticated special purpose code (SPC) for many statistical problems. The reasons are many. To name but a few: 1) availability, i.e., many solvers have not been easily available in R, 2) capability, i.e., problems could not be solved due to a lack of adequate solvers, and 3) efficiency, i.e., SPC tends to be faster.

This paper introduces an extensible object oriented R optimization infrastructure (**ROI**) promoting the usage of optimization in R and R as a tool for optimization. In doing so it strives to enable users to formulate problems and experiment with different solvers in a straightforward way, help researchers to find the appropriate solver for their particular problem, or assist package developers to streamline their package dependencies. The framework is composed of package **ROI** (Hornik, Meyer, Schwendinger, and Theußl 2020) and its (at the time of this writing) 23 companion packages. Package **ROI** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=ROI`.

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. Optimization problems are constructed in a consistent way and stored in a single object. This makes it possible that problems are easily altered (reused) and shared before they are

passed to a unified solve function. Such problems are then formulated and manipulated by using the provided R functions instead of special syntax from DSLs for which highly specialized knowledge would be required. Moreover, we believe that this approach makes it more attractive to add new solvers to the R solver landscape, e.g., to take advantage of recent advances in conic optimization (increase availability). Another key feature of **ROI** is that it is designed to be extensible. 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). Furthermore, allowing package developers to plug-in new solvers quite effortlessly not only makes it easy to use their highly efficient code for a given problem but possibly also in many other applications (eliminate efficiency detriments). 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 covers many optimization problems encountered in statistics, machine learning and data science (see, e.g., Koenker and Mizera 2014, for a survey of convex problems in statistics).

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 available R packages concerned with solving these problem classes is given in Section 3. Sections 4 and 5 show, respectively, how to formulate and solve optimization problems with the **ROI** package. Based on the tools presented in the previous sections, Section 6 provides basic examples. Section 7 is dedicated to the extension of **ROI**. Applications in the field of statistics are presented in Section 8. Section 9 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 $\mathcal{C} \subseteq \mathbb{R}^n$ we are interested in finding an $x^* \in \mathbb{R}^n$ that solves

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

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 subsequently will mainly deal with minimization problems.

An OP is called bounded if there exists a finite constant u such that $f_0(x) > u,\ \forall x \in \mathcal{C}$, if such constant does not exist, the problem is unbounded. Thus, a problem like in Equation 1 may or may not have a solution. Given a feasible and bounded problem a vector $x^* \in \mathcal{C}$ that satisfies

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

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:

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

where $b \in \mathbb{R}^m$ is the so-called right-hand-side (Nocedal and Wright 2006). 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).

Several interconnected characteristics exist 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 (i.e., one as good or better than all other feasible solutions) was obtained, the same is mostly not true for non-convex problems (several local optima may exist). More information about convex programming can be found in, e.g., Boyd and Vandenberghe (2004); Ben-Tal and Nemirovski (2019).

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

2.1. Linear programming

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

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

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 $a_i^\top x \leq b_i,\ i = 1, \ldots, m$ (here $a_i^\top$ refers to the ith row of the coefficient matrix A). All LPs are convex and usually 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).

A typical statistical problem which falls into this problem class is solving the least absolute deviations (LAD) or L_1 regression problem. Following, Wagner (1959) the objective function

$$\text{minimize} \quad \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

where $\hat{y}_i = \beta_0 + \mathbf{x}_i^\top \beta$ and $\mathbf{x}_i \in \mathbb{R}^k$, can be expressed as

$$\begin{array}{ll} \underset{\beta_0,\beta,\mathbf{e}^+,\mathbf{e}^-}{\text{minimize}} & \sum_{i=1}^n e_i^+ + e_i^- \\ \text{subject to} & \beta_0 + \beta^\top \mathbf{x}_i + e_i^+ - e_i^- = y_i, \quad i = 1, \ldots, n \\ & e_i^+, e_i^- \geq 0, \quad i = 1, \ldots, n. \end{array} \tag{4}$$

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:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^\top Q_0 x + a_0^\top x \\ \text{subject to} & Ax \leq b. \end{array} \tag{5}$$

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):

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} x^\top Q_0 x + a_0^\top x \\ \text{subject to} & \frac{1}{2} x^\top Q_i x + a_i^\top x \leq b_i, \quad i = 1, \ldots, m. \end{array} \tag{6}$$

A QCQP is convex if and only if all Q_i $(i = 0, \ldots, m)$ are positive semidefinite (Lobo, Vandenberghe, Boyd, and Lebret 1998). Whereas convex QPs or even QCQPs 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 a non-convex QCQP is still an active field of research. Details on the necessary transformations to cast a 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 refers to a class of problems designed to model convex OPs. The most prominent members of this class are LP, SOCP and SDP. We follow the common practice to define a conic program (CP) as:

$$\begin{array}{ll} \text{minimize} & a_0^\top x \\ \text{subject to} & Ax + s = b \\ & s \in \mathcal{K}, \end{array} \tag{7}$$

where the set $\mathcal{K}$ is a nonempty closed convex cone.

The standard form of a CP as given in Equation 7 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 modeling of 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 7. Thereby nonlinear objective functions are expressed in epigraph form (see, e.g., Boyd and Vandenberghe 2004):

$$\begin{array}{rl} \text{minimize} & t \\ \text{subject to} & f_0(x) \leq t \\ & f_i(x) \leq b_i. \end{array} \tag{8}$$

Practically the number 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.

Zero cone and free cone

The zero and free cones are, respectively, given by

$$\mathcal{K}_{\text{zero}} = \{0\}, \mathcal{K}_{\text{free}} = \mathbb{R} = \mathcal{K}^*_{\text{zero}},$$

where for a cone $\mathcal{K}$ we write $\mathcal{K}^* = \{y | x^\top y \geq 0 \text{ for all } x \in \mathcal{K}\}$ for the dual cone (see, e.g., Boyd and Vandenberghe 2004 for more information about dual cones). From Equation 7 it can be immediately seen, that in the case of linear equality constraints s_i has to be zero, i.e., $s_i \in \mathcal{K}_{\text{zero}} \iff s_i = b_i - a_i^\top x = 0 \iff a_i^\top x = b_i$.

Linear cone (non-negative orthant)

The linear cone is given by

$$\mathcal{K}_{\text{lin}} = \{x \in \mathbb{R} \mid x \geq 0\}. \tag{9}$$

This cone is used to represent linear inequality (less than or equal) constraints, by requiring s_i to be non-negative, i.e., $s_i \in \mathcal{K}_{\text{lin}} \iff s_i = b_i - a_i^\top x \geq 0 \iff a_i^\top x \leq b_i$.

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

Second-order cone

The second-order cone is given by

$$\mathcal{K}^n_{\text{soc}} = \{(t, x) \in \mathbb{R}^n \mid x \in \mathbb{R}^{n-1},\ t \in \mathbb{R},\ \|x\|_2 \leq t\}. \tag{10}$$

This cone is used to model sums of norms as well as convex QPs and QCQPs (Lobo *et al.* 1998; Alizadeh and Goldfarb 2003). CPs where $\mathcal{K}$ is a product of free, non-negative and second-order cones are commonly referred to as SOCPs.

Positive semidefinite cone

The positive semidefinite (PSD) cone is given by

$$\mathcal{K}^n_{\text{psd}} = \{X \mid X \in \mathcal{S}^n,\ z^\top X z \geq 0 \text{ for all } z \in \mathbb{R}^n\}. \tag{11}$$

Here $\mathcal{S}^n$ refers to the space of real-symmetric $n \times n$ matrices. CPs restricted to the positive semidefinite cone are referred to as SDPs. They are commonly used for solving combinatorial

problems (e.g., maximum cut problem) and for solving convex QPs and QCQPs (Vandenberghe and Boyd 1996; Helmberg 2000; Freund 2009; Bao *et al.* 2011). Lobo *et al.* (1998) show that each SOCP can be rewritten as an SDP.

Exponential cone

The primal exponential cone is defined as

$$\mathcal{K}_{\text{expp}} = \{(x,y,z) \in \mathbb{R}^3 \mid y > 0,\ ye^{\frac{x}{y}} \leq z\} \cup \{(x,0,z) \in \mathbb{R}^3 \mid x \leq 0,\ z \geq 0\}. \quad (12)$$

Its dual is given by

$$\mathcal{K}_{\text{expd}} = \{(u,v,w) \in \mathbb{R}^3 \mid u < 0, -ue^{\frac{v}{u}} \leq ew\} \cup \{(0,v,w) \in \mathbb{R}^3 \mid v,\ w \geq 0\}. \quad (13)$$

As can be inferred from Equation 12, 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 already been investigated in Koecher (1957) and is defined as

$$\mathcal{K}^{\alpha}_{\text{powp}} = \{(x,y,z) \in \mathbb{R}^3 \mid x,\ y \geq 0,\ x^{\alpha}y^{1-\alpha} \geq |z|\},\ \text{where } \alpha \in [0,1]. \quad (14)$$

Its dual is given by

$$\mathcal{K}^{\alpha}_{\text{powd}} = \left\{(u,v,w) \in \mathbb{R}^3 \mid u,\ v \geq 0,\ \left(\frac{u}{\alpha}\right)^{\alpha}\left(\frac{v}{1-a}\right)^{1-\alpha} \geq |w|\right\},\ \text{where } \alpha \in [0,1]. \quad (15)$$

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). This is the problem where at least one f_i, $i = 0, \ldots, m$ in Equation 2 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 extremum of f_0 in the neighborhood of the starting value (local optimum).

2.5. Mixed integer programming

A mixed integer program (MIP) adds the additional requirement to the optimization problem 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) can already be hard to solve. Nevertheless an increase in quantity and quality of free and nonfree solvers was observed in the last decades (Linderoth and Ralphs 2005; Bixby 2012). Typically solvers use branch-and-bound (Land and Doig 1960) and the cutting plane (Gomory 1960) algorithms or a combination of both. The algorithms avoid solving the problem directly, but instead solve multiple relaxations where the integer constraint is dropped.

3. Software

Recently, an increase of the available packages handling many different OPs in R has been observed. The CRAN task view *Optimization and Mathematical Programming* (Theußl *et al.* 2020) currently lists around 100 different optimization related packages. The capability these packages provide range from solvers which can solve a wide range of optimization problems (e.g., **optimx**, Nash and Varadhan 2011; Nash 2014a) to very specialized solvers which are created to solve a specific problem type very fast (e.g., nonlinear regression solvers). 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 investigate the available (open source) solvers, splitting these into linear solvers, quadratic solvers, conic solvers and general purpose solvers. We then discuss commercial solvers (i.e., any solver developed for sale) and the NEOS server.

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 a function). Similarly constraints are typically of type box, linear, (convex) quadratic, conic or functional. Box constraints (or variable bounds) are a special type of linear 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. Note that, solvers that take as input values a linear objective and conic constraints are also applicable to OPs with conic objective and conic constraints by making use of the epigraph form transformation. The most general form are functional objective/constraints which includes all linear and nonlinear objective/constraints.

Table 1 gives an overview on optimization packages available on CRAN (`https://CRAN.R-project.org`) with a focus on open source solvers. The position of a particular package 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 package including its current position. For instance, the **ECOSolveR** package (Fu and Narasimhan 2019) which provides an interface to the **ECOS** library (Domahidi, Chu, and Boyd 2013) can solve conic problems restricted

Constraints	Objective			
	Linear	Quadratic	Conic	Functional
No				**BB**, **mize**, **trustOptim**
Box				**DEoptim**, **dfoptim**, **GenSA**, **lbfgsb3**, **metaheuristicOpt**, **minqa**, **optimx**, **Rcgmin**, **rgenoud**, **Rmalschains**, **Rvmmin**, **soma**, **stats**, **ucminf**
Linear	**clpAPI***, **Rglpk**$^{*+}$, **lpSolve**$^{*+}$, **rcdd***, **Rsymphony**$^{*+}$	**coneproj***, **Dykstra***, **kernlab**, **LowRankQP***, **osqp***, **quadprog***, **ROI.plugin.qpoases**		
Quadratic				
Conic			**cccp***, **CLSOCP***, **ECOSolveR**$^{*+}$, **Rcsdp***, **Rdsdp***, **scs***	
Functional				**alabama**, **deoptimr**, **clue**, **NlcOptim**, **nloptr**, **Rsolnp**

Table 1: Selected R packages displayed based on the types of optimization problems they are applicable to. Here * indicates that the solver is restricted to convex problems and $^{+}$ indicates that the solver can model integer constraints.

to combinations of the zero, non-negative, second-order 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.

3.2. The R solver landscape

The solver landscape can be split into two parts. First, solvers where the functional form is fixed and only the coefficients are provided, which 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 an open source software framework for the operations research community including the COIN-OR linear programming (**Clp**, Forrest, de la Nuez, and Lougee-Heimer 2004) 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** (Roettger and Gelius-Dietrich 2020a) 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** (Roettger and Gelius-Dietrich 2020b) and the high level interface **Rglpk** (Theußl and Hornik 2019) 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 modeling of semi-continuous and special ordered sets problems. Packages **lpSolve** (Berkelaar 2020) and **lpSolveAPI** (Konis and Schwendinger 2020) provide access to the **lp_solve** solver in R.

Additionally the function `lpcdd()` from package **rcdd** (Geyer, Meeden, and Fukuda 2019) and the function `simplex()` from package **boot** (Canty and Ripley 2020) 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[1] we can conclude that the following elements should be present in a consistent and convenient optimization infrastructure for modeling LPs and MILPs.

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

`constraints`:

- Includes a constraint matrix A (see Equation 3),
- a vector giving the direction of the constraints (i.e., `==`, `<=` or `>=`), and
- 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.

Note that 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.

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

	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.

Quadratic solvers

As Table 1 shows, most of the quadratic solvers are designed to solve convex quadratic problems with linear constraints. The **quadprog** (Turlach and Weingessel 2019) package uses the dual method described in Goldfarb and Idnani (1983). **LowRankQP** (Ormerod and Wand 2020) is based on an interior-point algorithm described in Fine and Scheinberg (2001). **Dykstra** (Helwig 2018) implements Dykstra's cyclic projection algorithm (Dykstra 1983), **coneproj** (Meyer and Liao 2018) transforms the original QP problem into a cone projection problem (Liao and Meyer 2014) and **osqp** (Stellato, Banjac, Goulart, and Boyd 2019) uses the alternating direction method of multipliers described in Stellato, Banjac, Goulart, Bemporad, and Boyd (2017).

Additionally, the package **ROI.plugin.qpoases** (Schwendinger 2020) and the `ipop()` function from **kernlab** (Karatzoglou, Smola, Hornik, and Zeileis 2004; Karatzoglou, Smola, and Hornik 2019) can be used to obtain solutions for non-convex quadratic problems with linear constraints. However, for the non-convex case there is no guarantee that the returned solution is a global optimum. **ROI.plugin.qpoases** is an interface to the **qpOASES** (Ferreau, Kirches, Potschka, Bock, and Diehl 2014; Ferreau, Potschka, and Kirches 2018) library, which uses an online active set strategy to solve quadratic optimization problems.

QP solvers generally take the same arguments as LP solvers 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 7, 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 SDPs there exist the specialized packages **Rcsdp** and

	Global GPS		Local GPS	
	Gradient free	Gradient	Gradient free	Gradient
No constraint	6	0	7	19
Box constraint	28	4	8	12
Functional constraint	2	0	7	7

Table 3: Overview of general purpose solvers.

Rdsdp. Since any SOCP can be transformed into an SDP they can also be used for solving SOCPs. **Rcsdp** (Corrada Bravo and Borchers 2020) is an interface to the **CSDP** (Borchers 1999) library which is part of the COIN-OR project. **Rdsdp** (Zhu and Ye 2020) is an interface to the **DSDP** (Benson 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 2019) 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 2019) 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 (GPSs). These solvers can minimize (or maximize) any functional form representable as an R function with – depending on solver capabilities – different types of constraints, where again the most general form of constraint is the functional constraint (i.e., an R function). The generality of GPSs comes at the price of performance and that there is usually no guarantee that a global optimum is reached.

Important properties of GPSs are whether they are designed to search for a local or global optimum, 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 5 where additional details can be found) and reveals some interesting details about the R GPS landscape. Around 60 percent of the GPS are designed for local optimization, 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 GPSs 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 5 gives an extensive listing of which methods are designed to search for a

global solution. For more information about the methods we refer to Mullen (2014a).

Based on the type of constraints, the GPSs can be divided into no constraints, box constraints, linear constraints, quadratic constraints and functional constraints. As Table 3 shows, most of the GPSs 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 2018) 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 and Johnson 2020). 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 GPSs 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[2]),
- $h_i(x) = b_i,\ i = 1, \ldots, k$ (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 2019; **Rnlminb2**, `csr::snomadr`, Racine and Nie 2019),
- $g_j(x) \geq 0,\ j = k+1, \ldots, m$ (e.g., **alabama**, `nloptr::auglag`, `nloptr::cobyla`, `nloptr::ires`, `nloptr::mma`, **neldermead**, Bihorel and Baudin 2018; and `nloptr::slsqp`),
- $l_j \leq g_j(x) \leq u_j,\ j = k+1, \ldots, m$ (e.g., **ipoptr**, Ypma 2011[2]; **Rdonlp2**, Wuertz 2007[2]; **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 GPSs 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:

[2]Note the packages **ipoptr**, **Rnlminb2** and **Rdonlp2** are not available on CRAN but on R-Forge. Information about the installation of **ipoptr** can be found at https://coin-or.github.io/Ipopt/INSTALL.html#INSTALL_R.

- a function representing the constraints;
- a vector giving the direction of the constraints; and
- a vector giving the right-hand-side.

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

Additionally some GPSs 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.

3.3. Other optimization back-ends

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. At the time of this writing R interfaces are available to the commercial solver software **CPLEX** (IBM ILOG 2019), **MOSEK** (**MOSEK** ApS 2017), **Gurobi** (**Gurobi** Optimization, Inc. 2016), **Lindo** (Lindo Systems 2003) and **localsolver** (Benoist, Estellon, Gardi, Megel, and Nouioua 2011). To cover also the commercial side, the interface packages **Rcplex** (Theußl and Bravo 2016) and **Rmosek** (**MOSEK** ApS 2019) were included in defining the requirements for a consistent solver infrastructure.

Network-enabled optimization system (NEOS)

The NEOS (Czyzyk, Mesnier, and Moré 1998; Dolan 2001; Gropp and Moré 1997) server (`https://neos-server.org`) provides free access to more than 60 numerical optimization solvers. It can be accessed via the internet by submitting OPs via the homepage, email, the XML-RPC application programming interface or Kestrel. Depending on the solver the OPs have to be formulated in different ways, overall most solvers support input from AMPL and GAMS. The R packages **rneos** (Pfaff 2020) and **ROI.plugin.neos** (Hochreiter and Schwendinger 2020) use the XML-RPC API to communicate with the NEOS server. In order to upload OPs with **rneos**, the OPs have to be formulated in the input format supported by the solver (e.g., AMPL, GAMS, `MPS`). In contrast to that, **ROI.plugin.neos** supports OPs generated with **ROI**, internally each OP is transformed to GAMS before they are submitted to the server. The result is again converted back into a suitable solution format.

4. Data structures

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 object for the `problem` argument, in a general and intuitive way, is a 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).

In the remainder of this section we discuss the conceptual design of **ROI** and how to use it to formulate optimization problems. For illustrative purposes we already use functionality of package **ROI**.

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

4.1. 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 (F, i.e., nonlinear functions which cannot be represented as an QP or CP), are represented as an R function 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`. If no gradient is 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 of the objective variables) as a numeric vector. The `names` argument is again optional.

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

4.2. 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 and will therefore be explained only 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)`, where `L` can be either a numeric vector of length n or a matrix of dimension $m \times n$. In accordance with Equation 7 the `cones` argument imposes a restriction on the slack variables s. 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 a short cut of the cone name, as defined in Section 2.3. Currently **ROI** implements constructors for the cones `K_zero`, `K_lin`, `K_soc`, `K_psd`, `K_expp`, `K_expd`, `K_powp`, and `K_powd`. To combine different cones the generic combine `c()` can be used.

`Q` Quadratic constraints as defined in Equation 6 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 $n \times n$ 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`'. Since in many situations it is desirable to optimize a given objective function subject to composite constraints of different kinds, **ROI** can combine multiple constraints into a single constraint using the generic functions `c()` or `rbind()`. Since the matrices `L` and `Q` can become huge but are typically sparse we use the simple triplet matrix format from the **slam** (Hornik, Meyer, and Buchta 2019) package to store them internally. In the simple triplet matrix format (sometimes referred to as coordinate list format) only the row indices, the column indices and the values of the non-zero elements are stored in a list. We choose the **slam** package not only for efficiency reasons but also due to the fact that many solver APIs demand such a format.

4.3. Objective variable types

As it is common practice in mixed-integer solvers to distinguish between the variable types continuous, integer and binary we encode the variable choice with the following characters: `"C"` for continuous, `"I"` for integer and `"B"` for binary. By default all the variables are assumed to be of continuous type.

4.4. Bounds

A variable bound is a special type of constraint used to restrict an objective variable between a real lower and upper bound. Therefore, variable bounds are often also called "box bounds" or "box constraints". Although variable bounds could be easily modeled as linear constraints, many GPSs only support variable bounds (see Table 3). Furthermore, most solvers that support any type of constraint allow to specify variable bounds directly. 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. Variables with fixed bounds are a special case of double bounded variables where the lower bound is equal to the upper bound. 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 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. The default lower and upper bound can be changed by the arguments `ld` and `ud`. 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.
```

If all 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.
```

If the 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.
```

Consider the box constrains $0 \leq x_3 \leq 20$ and $-20 \leq x_i \leq 20$ for all $i \in I = \{1, 2, 4\}$. These variable bounds can be constructed by the following **ROI** code:

```
R> V_bound(li = 3, lb = 0, ld = -20, ud = 20, nobj = 4L)

ROI Variable Bounds:

3 lower and 4 upper non-standard variable bounds.
```

4.5. Optimization problem

In **ROI**, a new optimization problem is created by calling

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

As an example consider the LP

$$\begin{array}{rrcrcrl} \underset{x}{\text{maximize}} & 3x_1 & + & 7x_2 & - & 12x_3 & \\ \text{subject to} & 5x_1 & + & 7x_2 & + & 2x_3 & \leq 61 \\ & 3x_1 & + & 2x_2 & - & 9x_3 & \leq 35 \\ & x_1 & + & 3x_2 & + & x_3 & \leq 31 \\ & x_1, x_2 \geq 0, \; x_3 \in [-10, 10]. & & & & & \end{array} \tag{16}$$

This problem can be constructed by the following **ROI** code:

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

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

```
R> lp <- OP()
```

and using the setter/getter functions `objective()`, `constraints()`, `bounds()`, `types()` and `maximum()` to set/get the corresponding element.

```
R> objective(lp) <- L_objective(c(3, 7, -12))
R> constraints(lp) <- L_constraint(A, dir = dir, rhs = rhs)
R> bounds(lp) <- V_bound(li = 3, ui = 3, lb = -10, ub = 10, nobj = 3)
R> maximum(lp) <- 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.
```

The setter functions make it easy to alter previously created OPs. The getter 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 function returns directly the underlying data representation.

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 be easily accessed after the creation of the problem.
- 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.

5. Functionality

The R optimization infrastructure is structured into the package **ROI** and its accompanying extensions (plug-ins and models). Package **ROI** provides all the necessary classes and methods and manages the extensions (i.e., automatically loads plug-ins and manages meta data about the plug-ins). The extension packages add optimization solvers, read/write functions and additional resources (e.g., model collections). The plug-in extensions 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, the control arguments available for 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 2019a) is used for data packages with predefined OPs. In Section 5.6 we give an overview on 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). Alternatively the solver specific parameters can be specified via the dots arguments.

The problem stated in Equation 16 can be solved by the following **ROI** code:

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

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

5.2. Solution and status code

Status code

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) and `1` otherwise. For optimization solvers specialized on solving convex LPs, QPs and CPs it can be assumed that a global solution was found, if the status code is `0`. Unfortunately, the same is not true for non-convex QPs and GPS, here status codes are less informative. Some packages like **optimx** and **alabama** check the Karush-Kuhn-Tucker (KKT) conditions to verify that the solution found meets the criteria of a local minimum.

Solution

To make the solutions of the various solvers easy to understand, all the solutions are canonicalized. 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.

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
```

Some OPs have multiple solutions, in the case of BLP (MILP) some solvers 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

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x_1 - x_2 \\ \text{subject to} \quad & x_1 - x_2 = 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned} \tag{17}$$

has an infinite number of solutions. The following example is based on Fischetti and Salvagnin (2010) and will be used later to illustrate how multiple solutions can be obtained.

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

The `"msbinlp"` solver allows 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, we set `nsol_max` to 32,

```
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))
R> (blp_sol <- ROI_solve(blp, solver = "msbinlp", method = "glpk",
+    nsol_max = 32))

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

R> solution(blp_sol)

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

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

alternatively it is also possible to set `nsol_max` to `Inf`. This is advantageous if an upper bound on the number of solutions is hard to guess and all the solutions should be retrieved.

If the status code is not zero, `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] -8.035136e-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, `solution()` can be employed to retrieve 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 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


$sensitivity_report
[1] NA
```

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.

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. The available reformulations are listed by calling function `ROI_registered_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 2019) package.

The binary QP is given by:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & -x - 4y - z + 3xy + yz \\ & x, y, z \in \{0, 1\} \end{aligned} \tag{19}$$

In the following the **ROI** code is given to define the binary QP, transform the problem into a MILP and solve it:

```
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 can currently make use of more than thirty different solver methods, 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, 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 which are also applicable will be returned.

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

```
             Package                                Repository
3   ROI.plugin.cplex                https://CRAN.R-project.org
9    ROI.plugin.neos                https://CRAN.R-project.org
18  ROI.plugin.cplex              http://R-Forge.R-project.org
22 ROI.plugin.gurobi              http://R-Forge.R-project.org
25  ROI.plugin.mosek              http://R-Forge.R-project.org
27   ROI.plugin.neos              http://R-Forge.R-project.org
36 ROI.plugin.gurobi https://github.com/FlorianSchwendinger
37  ROI.plugin.mosek https://github.com/FlorianSchwendinger
```

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 (an rds file on R-Forge) 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 calling function `ROI_registered_solvers()`,

```
R> head(ROI_registered_solvers(), 3)
```

```
              nlminb               alabama                   cbc
 "ROI.plugin.nlminb" "ROI.plugin.alabama"     "ROI.plugin.cbc"
```

similarly

```
R> ROI_applicable_solvers(lp)
```

```
 [1] "alabama"       "cbc"           "clp"           "cplex"
 [5] "ecos"          "glpk"          "gurobi"        "ipop"
 [9] "lpsolve"       "mosek"         "neos"          "nloptr.cobyla"
[13] "nloptr.mma"    "nloptr.auglag" "nloptr.isres"  "nloptr.slsqp"
[17] "qpoases"       "scs"           "symphony"
```

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 2019) package.

`ROI_installed_solvers()` gives a listing of all the installed plug-ins (not necessarily loaded) delivered directly with **ROI** and found by searching for the prefix '`ROI.plugin`' in the R library trees.

```
R> head(ROI_installed_solvers(), 3)
```

```
             nlminb              alabama                  cbc
"ROI.plugin.nlminb" "ROI.plugin.alabama"     "ROI.plugin.cbc"
```

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, while **nlminb** can only take box constraints.

Furthermore we want to emphasize that **ROI** was built to be extended, as shown in Section 7.

5.5. ROI read/write

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

```
R> lp_file <- tempfile()
R> ROI_write(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

Constraints	Objective Linear	Quadratic	Conic	Functional
No				
Box				**optimx**
Linear	**clp***, **cbc**$^{*+}$, **glpk**$^{*+}$, **lpsolve**$^{*+}$, **msbinlp**$^{*+}$, **symphony**$^{*+}$	**ipop**, **quadprog***, **qpoases**		
Quadratic		**cplex**$^{+}$, **gurobi**$^{*+}$, **mosek**$^{*+}$, **neos**$^{+}$		
Conic			**ecos**$^{*+}$, **scs***	
Functional				**alabama**, **deoptim**, **nlminb**, **nloptr**

Table 4: Currently available **ROI** plug-ins displayed based on the types of optimization problems they are applicable to. Here * indicates that the solver is restricted to convex problems and $^{+}$ indicates that the solver can model integer constraints. Note all the plug-ins have the prefix **ROI.plugin** and the modeling capabilities of the plug-ins do not necessarily represent the modeling capabilities of the underlying solvers.

```
R> ROI_read(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

Test problem collections are commonly used in optimization to evaluate and compare the performance of solvers. As each class of optimization problems has its own test sets stored in various formats, **ROI** currently provides access to the NETLIB-LP and MIPLIB collections

and the **globalOptTests** package. The NETLIB-LP collection (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 the MIPLIB collection (Koch *et al.* 2011), an extensive collection of academic and industrial MILP applications. The **globalOptTests** package contains 50 box constrained nonlinear global OPs for benchmarking purposes.

The problems contained in these collections and packages were transformed into **ROI** optimization problems and can be accessed through the **ROI.models.netlib**, **ROI.models.miplib** (Schwendinger and Theußl 2019) and **ROI.models.globalOptTests** (Schwendinger 2017) packages. Since MIPLIB provides no license file, the OPs are not included in the package but can be easily obtained with the `miplib_download_*()` functions.

```
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`.

```
R> library("ROI.models.netlib")
R> agg <- netlib("agg")
R> ROI_solve(agg, "glpk")
```

```
Optimal solution found.
The objective value is: -3.599177e+07
```

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 make 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 (Extensible Markup Language).

5.7. ROI settings

Many general and/or solver-related settings can be set or modified using the `ROI_options()` function. Apart from that **ROI** recognizes some environment variables.

Gradient and Jacobian

When creating a plug-in, the function `G` should be used to derive the gradient and `J` should be used to derive the Jacobian. In **ROI** the gradient and Jacobian are derived analytically for linear and quadratic terms. For the derivation of nonlinear terms, by default, the **numDeriv** (Gilbert and Varadhan 2019) package with the Richardson extrapolation is used. However, this can be easily changed by providing customized functions to derive the gradient or Jacobian function.

```
R> simple_gradient <- function(func, x, ...) {
+    numDeriv::grad(func, x, method = "simple", ...)
+  }
R> ROI_options("gradient", simple_gradient)
R> simple_jacobian <- function(func, x, ...) {
+    numDeriv::jacobian(func, x, method = "simple", ...)
+  }
R> ROI_options("jacobian", simple_jacobian)
```

Solver selection

If 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")
R> ROI_options("default_solver", "auto")
```

Load plug-ins

The plug-ins are loaded automatically. However, 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. Examples

In this section we provide small examples to introduce the reader into the modeling capabilities of **ROI**.

6.1. Linear optimization problems

Consider the LP

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & x_1 + 2x_2 \\ \text{subject to} \quad & x_1 + 8x_2 = 9 \\ & 5x_1 + \ x_2 \leq 6 \\ & x_1 \in [-9, 9], \ x_2 \in [-7, 7]. \end{aligned} \tag{20}$$

To solve this LP we use the following **ROI** code:

```
R> lp <- OP(c(1, 2), maximum = TRUE,
+    L_constraint(L = rbind(c(1, 8), c(5, 1)), dir = c("==", "<="),
+    rhs = c(9, 6)), bounds = V_bound(lb = c(-9, -7), ub = c(9, 7)))
R> (lp_sol <- ROI_solve(lp, "glpk"))

Optimal solution found.
The objective value is: 3.000000e+00

R> solution(lp_sol)

[1] 1 1
```

6.2. Quadratic optimization problems

Consider the QP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}(x_1^2 + x_2^2) - x_1 \\ \text{subject to} \quad & 4x_1 + 6x_2 \geq 10 \\ & x_1, x_2 \geq 0. \end{aligned} \tag{21}$$

Recall that for quadratic terms **ROI** uses the standard form $\frac{1}{2}x^\top Qx + a^\top x$ (see Equation 6). Therefore, this problem can be solved by the following **ROI** code:

```
R> qp <- OP(Q_objective(Q = diag(2), L = c(-1, 0)),
+    L_constraint(c(4, 6), ">=", 10))
R> (qp_sol <- ROI_solve(qp, "qpoases"))

Optimal solution found.
The objective value is: -1.538462e-01

R> solution(qp_sol)

[1] 1.4615385 0.6923077
```

To add the constraint $7x_1 + 11x_2 - x_1^2 - x_2^2 \geq 40$ we combine the new constraint with the existing constraint.

```
R> qcqp <- qp
R> constraints(qcqp) <- rbind(constraints(qp),
+    Q_constraint(-diag(c(2, 2)), L = c(7, 11), ">=", 40))
```

In **ROI** a QCQP can be solved by the solvers

```
R> ROI_applicable_solvers(qcqp)

[1] "alabama"       "cplex"         "gurobi"        "neos"
[5] "nloptr.cobyla" "nloptr.mma"    "nloptr.auglag" "nloptr.isres"
[9] "nloptr.slsqp"
```

where among these solvers `"cplex"` and `"gurobi"` are best suited for this type of problem. However for reproducibility we use the open source solver `"alabama"`:

```
R> (qcqp_sol <- ROI_solve(qcqp, "alabama", start = c(5, 5)))

Optimal solution found.
The objective value is: 9.447513e+00

R> solution(qcqp_sol)

[1] 2.845720 4.060584
```

and the NEOS server:

```
R> (qcqp_sol <- ROI_solve(qcqp, "neos", method = "mosek"))

Optimal solution found.
The objective value is: 9.447513e+00

R> solution(qcqp_sol)

[1] 2.845695 4.060596
```

6.3. Conic optimization problems

Conic optimization problems in standard form (see Equation 7) are comprised of a linear objective function and conic constraints. The requirement of a linear objective function is not restrictive, since by making use of the epigraph form (see Equation 8) any CP can be transformed into the standard form. In conic optimization conic constraints are used to express predefined linear and nonlinear constraints by the equation $Ax + s = b$, where the slack variables s are required to lie in a specific cone $b - Ax = s \in \mathcal{K}$.

Zero cone

The zero cone is used to model linear equality constraints, since $Ax = b$ is equivalent to $Ax + s = b$, $s \in \mathcal{K}_{\text{zero}}$. The linear constraint $x_1 + 8x_2 = 9 \iff 9 - (1 \;\; 8)x = s \in \mathcal{K}_{\text{zero}}$ can be expressed as follows:

```
R> cpeq <- C_constraint(c(1, 8), K_zero(1), 9)
```

Linear cone

The linear cone is used to model linear less than equal constraints, since $Ax \leq b$ is equivalent to $Ax + s = b$, $s \in \mathcal{K}_{\text{lin}}$. The linear constraint $5x_1 + x_2 \leq 6 \iff 6 - (5 \;\; 1)x = s \in \mathcal{K}_{\text{lin}}$ can be expressed as follows:

```
R> cpleq <- C_constraint(c(5, 1), K_lin(1), 6)
```

By combining the zero cone constraint with the linear cone constraint the LP stated in Equation 20 can also be formulated as a CP.

```
R> zlcp <- lp
R> constraints(zlcp) <- c(cpeq, cpleq)
R> (zlcp_sol <- ROI_solve(zlcp, solver = "ecos"))

Optimal solution found.
The objective value is: 3.000000e+00

R> solution(zlcp_sol)
```

```
[1] 1 1
```

Note that since in the definition of $\mathcal{K}_{\text{zero}}$ and $\mathcal{K}_{\text{lin}}$ only one variable is involved, a single zero cone or linear cone constraint can be expressed by a single row of the constraint matrix A. For all the other cones at least two rows of the constraint matrix A will be needed to express a single conic constraint.

Second-order cone

The second-order cone is used to express constraints of the type $\|u\|_2 \leq w$ (see Equation 10), where the variables $u \in \mathbb{R}^{n-1}$ and $w \in \mathbb{R}$ are expressed by $b - Ax \in \mathcal{K}^n_{\text{soc}}$. Specifically, w is expressed by $b_1 - a_1^\top x$ and u_i is expressed by $b_i - a_i^\top x,\ i = 2, \ldots, n$.

Consider the SOCP

$$\begin{array}{ll} \underset{(y,t)}{\text{maximize}} & y_1 + y_2 \\ \text{subject to} & \sqrt{(2+3y_1)^2 + (4+5y_2)^2} \leq 6 + 7t \\ & y_1,\ y_2 \in \mathbb{R},\ t \in (-\infty, 9] \end{array} \tag{22}$$

for $x = (y_1, y_2, t)^\top$, the constraint

$$\sqrt{(2+3y_1)^2 + (4+5y_2)^2} \leq 6 + 7t$$

is equivalent to

$$\sqrt{(b_2 - a_2^\top x)^2 + (b_3 - a_3^\top x)^2} \leq b_1 - a_1^\top x,$$

where

$$A = \begin{pmatrix} 0 & 0 & -7 \\ -3 & 0 & 0 \\ 0 & -5 & 0 \end{pmatrix},\ b = \begin{pmatrix} 6 \\ 2 \\ 4 \end{pmatrix}. \tag{23}$$

Given the constraint matrix A and the right-hand-side b this can be directly translated into the following OP:

```
R> soc1 <- OP(c(1, 1, 0),
+    C_constraint(L = rbind(c(0, 0, -7), c(-3, 0, 0), c(0, -5, 0)),
+    cone = K_soc(3), rhs = c(6, 2, 4)), maximum = TRUE,
+    bounds = V_bound(ld = -Inf, ui = 3, ub = 9, nobj = 3))
R> (soc1_sol <- ROI_solve(soc1, solver = "ecos"))

Optimal solution found.
The objective value is: 2.535571e+01

R> solution(soc1_sol)

[1] 19.055671  6.300041  9.000000
```

Consider the SOCP

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \sqrt{x_1^2 + x_2^2} \\ \text{subject to} & x_1 + x_2 = 2 \\ & x_1,\ x_2 \geq 0 \end{array} \tag{24}$$

by making use of the epigraph form the OP can be rewritten into the standard form.

$$\begin{array}{ll} \underset{(x,t)}{\text{minimize}} & t \\ \text{subject to} & \sqrt{x_1^2 + x_2^2} \leq t \\ & x_1 + x_2 = 2 \\ & x_1,\ x_2 \geq 0 \end{array} \tag{25}$$

This problem can be solved by the following **ROI** code:

```
R> A <- rbind(c(0, 0, -1), c(-1, 0, 0), c(0, -1, 0))
R> soc2 <- OP(objective = L_objective(c(0, 0, 1)),
+    constraints = c(C_constraint(A, K_soc(3), c(0, 0, 0)),
+    L_constraint(c(1, 1, 0), "==", 2)))
R> (soc2_sol <- ROI_solve(soc2, solver = "ecos"))

Optimal solution found.
The objective value is: 1.414214e+00

R> solution(soc2_sol)

[1] 1.000000 1.000000 1.414214
```

Exponential cone

The primal exponential cone is used to express constraints of the type $ve^{\frac{u}{v}} \leq w$ (see Equation 12), here $u \in \mathbb{R}$, $v \in \mathbb{R}$, $w \in \mathbb{R}$ and $v > 0$. Since three scalar variables are evolved one primal exponential cone adds three rows to the constraint matrix A. The variables u, v and w are again expressed by the corresponding elements of $b - Ax$. Specifically, $u := b_1 - a_1^\top x$, $v := b_2 - a_2^\top x$ and $w := b_3 - a_3^\top x$.

Consider the CP

$$\begin{array}{ll} \underset{(y,t)}{\text{maximize}} & y_1 + 2y_2 \\ \text{subject to} & \exp(7 + 3y_1 + 5y_2) \leq 9 + 11t_1 + 12t_2 \\ & y_1,\ y_2 \in (-\infty, 20],\ t_1,\ t_2 \in (-\infty, 50]. \end{array} \tag{26}$$

The constraint $\exp(7 + 3y_1 + 5y_2) \leq 9 + 11t_1 + 12t_2$ can be represented by the exponential cone by recognizing that $u = 7 + 3y_1 + 5y_2$, $v = 1$ and $w = 9 + 11t_1 + 12t_2$.

For $x = (y_1, y_2, t_1, t_2)^\top$ this leads to the matrices

$$A = \begin{pmatrix} -3 & -5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -11 & -12 \end{pmatrix},\ b = \begin{pmatrix} 7 \\ 1 \\ 9 \end{pmatrix}, \tag{27}$$

and the following conic constraint:

```
R> cexpp <- C_constraint(L = rbind(c(-3, -5, 0, 0), c(0, 0, 0, 0),
+    c(0, 0, -11, -12)), cone = K_expp(1), rhs = c(7, 1, 9))
```

Therefore this CP can be solved by the following **ROI** code:

```
R> expp1 <- OP(c(1, 2, 0, 0), cexpp,
+    bounds = V_bound(ld = -Inf, ub = c(20, 20, 50, 50)), maximum = TRUE)
R> (expp1_sol <- ROI_solve(expp1, solver = "ecos"))

Optimal solution found.
The objective value is: 6.685104e+00

R> solution(expp1_sol)

[1] -33.31490  20.00000  49.99996  49.99996
```

In many statistical models the objective contains logarithmic and exponential terms (e.g., logistic regression, relative risk regression, Poisson regression). As a simple example consider the CP

$$\begin{array}{ll} \underset{x}{\text{maximize}} & \log(9+7x) \\ \text{subject to} & 0 \leq x \leq 1, \end{array} \tag{28}$$

or in the epigraph form

$$\begin{array}{ll} \underset{(x,t)}{\text{maximize}} & t \\ \text{subject to} & \log(9+7x) \geq t \\ & 0 \leq x \leq 1. \end{array} \tag{29}$$

Taking the exponential of $\log(w) \geq u$ gives $w \geq \exp(u)$, therefore the constraint $\log(9+7x) \geq t$ can be represented by the exponential cone by recognizing that $u = t$, $v = 1$ and $w = 9 + 7x$, which yields the matrices

$$A = \begin{pmatrix} 0 & -1 \\ 0 & 0 \\ -7 & 0 \end{pmatrix}, \; b = \begin{pmatrix} 0 \\ 1 \\ 9 \end{pmatrix}. \tag{30}$$

Therefore this CP can be solved by the following **ROI** code:

```
R> A <- rbind(c(0, -1), c(0, 0), c(-7, 0))
R> log1 <- OP(L_objective(c(0, 1), c("x", "t")),
+    C_constraint(A, K_expp(1), rhs = c(0, 1, 9)),
+    bounds = V_bound(lb = c(0, -Inf), ub = c(1, Inf)), maximum = TRUE)
R> (log1_sol <- ROI_solve(log1, solver = "ecos"))

Optimal solution found.
The objective value is: 2.772589e+00

R> solution(log1_sol)
```

```
       x        t
1.000000 2.772589
```

Power cone

The primal power cone is used to express constraints of the type $u^\alpha v^{1-\alpha} \geq |w|$ (see Equation 14), here $u \in \mathbb{R}$, $v \in \mathbb{R}$, $w \in \mathbb{R}$, $u, v \geq 0$ and $\alpha \in [0, 1]$. Since three scalar variables are involved, one primal exponential cone adds three rows to the constraint matrix A. The variables u, v and w are again expressed by the corresponding elements of $b - Ax$. Specifically $u := b_1 - a_1^\top x$, $v := b_2 - a_2^\top x$ and $w := b_3 - a_3^\top x$.

Consider the CP

$$\begin{array}{ll} \underset{y}{\text{minimize}} & 3y_1 + 5y_2 \\ \text{subject to} & 5 + y_1 \geq (2 + y_2)^4 \\ & y_1 \geq 0,\ y_2 \geq 2. \end{array} \tag{31}$$

The constraint $5 + y_1 \geq (2 + y_2)^4$ can be represented by the power cone by recognizing that $u = 5 + y_1$, $v = 1$, $w = 2 + y_2$ and $\alpha = \frac{1}{4}$. For $x = (y_1, y_2)^\top$ this leads to the matrices

$$A = \begin{pmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix}, \tag{32}$$

and the following conic constraint:

```
R> A <- rbind(c(-1, 0), c(0, 0), c(0, -1))
R> cpowp <- C_constraint(A, K_powp(1/4), rhs = c(5, 1, 2))
```

Therefore, this CP can be solved by the following **ROI** code:

```
R> powp1 <- OP(c(3, 5), cpowp, bounds = V_bound(lb = c(0, 2)))
R> (powp1_sol <- ROI_solve(powp1, solver = "scs", max_iter = 1e6))

Optimal solution found.
The objective value is: 7.629999e+02

R> solution(powp1_sol)

[1] 251   2
```

Positive semidefinite cone

The positive semidefinite cone is used to express constraints of the type $X \in \mathcal{S}^n$ and $z^\top X z \geq 0$ for all $z \in \mathbb{R}^n$ (see Equation 11). Positive semidefinite constraints are expressed by the constraint matrix A and right-hand-side b. To express the linear matrix inequality $\sum_{i=1}^n x_i F_i \preceq F_0$ which is equivalent to $F_0 - \sum_{i=1}^n x_i F_i \in \mathcal{K}_{\text{psd}}^d$, in terms of $b - Ax \in \mathcal{K}_{\text{psd}}$ the symmetric matrices $F_i \in R^{d \times d}$ are transformed into vectors by a half-vectorization. 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. The function `vech` correspondingly 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
```

Consider the SDP

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

This problem can be solved by the following **ROI** code:

```
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_sol <- ROI_solve(psd, solver = "scs"))
```

```
Optimal solution found.
The objective value is: -1.486458e+00
```

```
R> solution(powp1_sol)
```

```
[1] 251   2
```

6.4. General nonlinear optimization problems

The following example from Rosenbrock (1960) is known as Rosenbrock's post office problem.

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

The following code can be used to solve this problem with **ROI**.

```
R> nlp_1 <- OP(maximum = TRUE, bounds = V_bound(ud = 42, nobj = 3L))
R> objective(nlp_1) <- F_objective(F = function(x) prod(x), n = 3,
+    G = function(x) c(prod(x[-1]), prod(x[-2]), prod(x[-3])))
R> constraint <- function(x) x[1] + 2 * x[2] + 2 * x[3]
R> constraints(nlp_1) <- F_constraint(F = constraint, dir = "<=", rhs = 72,
+    J = function(x) c(1, 2, 2))
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 thus the analytical Jacobian does not need to be provided.

In contrast to LP, QP and CP solvers almost all GPSs require that users specify starting values. The choice of the starting values has a big influence on how fast and to which solution the algorithm will converge.

```
R> (nlp_1_sol_1 <- ROI_solve(nlp_1, "alabama", start = c(10, 10, 10)))

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

R> solution(nlp_1_sol_1)

[1] 24.00002 11.99999 11.99999

R> (nlp_1_sol_2 <- ROI_solve(nlp_1, "alabama", start = c(20, 20, 20)))

No optimal solution found.
The solver message was: Convergence due to lack of progress in parameter
updates.
The objective value is: 1.314286e+308

R> solution(nlp_1_sol_2)

[1] NA NA NA
```

There are several possibilities to help the algorithm to find a good solution. In almost all practical applications it is possible to specify lower and upper bounds. Carefully chosen bounds can improve the quality of the solution and decrease the runtime of the algorithm. The runtime of the algorithm also strongly depends on the tolerances set; if the tolerances are set too small the algorithm will reach the maximum number of iterations before convergence.

```
R> (nlp_1_sol_3 <- ROI_solve(nlp_1, "alabama", start = c(10, 10, 10),
+    tol = 1E-24))

No optimal solution found.
The solver message was: ALABaMA ran out of iterations and did not converge.
The objective value is: 3.456000e+03

R> solution(nlp_1_sol_3, force = TRUE)

[1] 24.00002 11.99999 11.99999
```

Last but not least the chosen method has a big influence on the solution. As mentioned before some algorithms are designed to search for a global solution, others are designed to search for a local solution. **ROI.plugin.deoptim** provides access to the packages **DEoptim** (Mullen, Ardia, Gil, Windover, and Cline 2011) and **DEoptimR** which implement a differential evolution algorithm for global optimization. For more information about continuous global optimization in R we refer to Mullen (2014a).

```
R> (nlp_1_sol_4 <- ROI_solve(nlp_1, "deoptimr", start = c(20, 20, 20),
+    max_iter = 400, tol = 1E-6))
```

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

R> solution(nlp_1_sol_4)

[1] 23.99997 11.99991 12.00011
```

In general it is often hard to obtain a global optimum for non-convex optimization problems. Often the only option is to try different algorithms, parameters and starting values and hope that one of the solutions is a global optimum. **ROI** lowers the burden to compare different algorithms and therefore can assist in finding a global solution.

6.5. Mixed integer problems

Consider the MIP

$$\begin{array}{rl} \underset{x}{\text{minimize}} & 5x_1 + 7x_2 \\ \text{subject to} & 5x_1 + 3x_2 \geq 7 \\ & 7x_1 + 1x_2 \geq 9 \\ & x_1 \in \{0, 1\},\ x_2 \geq 0, x_2 \in \mathbb{Z}. \end{array} \tag{33}$$

This problem can be solved by the following **ROI** code:

```
R> A <- rbind(c(5, 3), c(7, 1))
R> milp <- OP(c(5, 7),
+    constraints = L_constraint(A, c(">=", ">="), c(7, 9)),
+    types = c("B", "I"))
R> (milp_sol <- ROI_solve(milp, solver = "glpk"))

Optimal solution found.
The objective value is: 1.900000e+01

R> solution(milp_sol)

[1] 1 2
```

7. Extending ROI

To stay abreast of changes and to 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 a key feature, 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.

7.1. 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 thereof. 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.

7.2. 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. Furthermore, 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, as in this way the data used in the solver call can be easily shared and inspected.

7.3. Registering 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 `ROI_plugin_register_solver_method()` function, which takes as arguments the problem types (as signatures), the solver name and a wrapper function, `ROI_solve()` is dispatched to.

The following code registers the solver, but is not executed as the entry already exists in the registry.

```
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.

7.4. 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()`. The following code is not executed as the entry already exists in the registry.

```
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 is 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`. The following code is not executed as the entry already exists in the registry.

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

7.5. Registering 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.

We define the signatures for BQP and MILP:

```
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"))
```

The following code 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"`:

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

The parameter `from` defines which signatures the original problem is allowed to have and `to` defines all possible signatures the reformulation could have. The code is not executed because this reformulation is already registered.

7.6. Registering 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 proposed 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 <- ROI_write(lp, file = fname, type = "json")
R> (lp_json <- ROI_read(file = fname, type = "json"))

ROI Optimization Problem:

Maximize a linear objective function of length 2 with
- 2 continuous objective variables,

subject to
- 2 constraints of type linear.
- 2 lower and 2 upper non-standard variable bounds.
```

7.7. ROI tests

Writing tests is an important task in software development. The **ROI.tests** (Schwendinger 2019b) 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!
```

8. Applications

In the following we demonstrate how **ROI** can be used to solve selected problems from statistics, namely L_1 regression, best subset selection, relative risk regression, sum-of-norms clustering, and graphical lasso.

8.1. L_1 regression

The linear programming formulation of the L_1 regression problem as shown in Section 2.1 can be constructed using **ROI** methods via the following R function.

```
R> create_L1_problem <- function(x, y) {
+    p <- ncol(x) + 1L
+    m <- 2 * nrow(x)
+    L <-  cbind(1, x, diag(nrow(x)), -diag(nrow(x)))
+    bnds <- V_bound(li = seq_len(p), lb = rep(-Inf, p), nobj = p + m)
+    OP(objective = L_objective(c(rep(0, p), rep(1, m))),
+       constraints = L_constraint(L, dir = eq(nrow(x)), rhs = y),
+       bounds = bnds)
+  }
```

One can solve, e.g., Brownlee's stack loss plant data example from the **stats** package using the above OP and the solver **GLPK** as follows.

```
R> data("stackloss", package = "datasets")
R> l1p <- create_L1_problem(x = as.matrix(stackloss)[,-4], y = stackloss[,4])
R> L1_res <- ROI_solve(l1p, solver = "glpk")
R> solution(L1_res)[1:ncol(stackloss)]

[1] -39.68985507   0.83188406   0.57391304  -0.06086957
```

The first value corresponds to the intercept and the others to the model coefficients.

8.2. 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 because of 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.

$$\underset{\beta}{\text{minimize}} \; \frac{1}{2}\|y - X\beta\|_2^2 \quad \text{subject to} \quad \sum_{i=1}^{p} \mathbb{I}_{\{\beta_i \neq 0\}} \leq k \tag{34}$$

As Equation 34 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 2020) package implements an efficient branch-and-bound algorithm which is significantly faster than exhaustive search. Using an optimization solver has the additional advantage that it is possible to impose additional restrictions, e.g., 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. This problem can be solved with a mixed integer QP solver or a mixed integer SCOP

solver. An implementation of the second order cone version can be found in the supplementary material.

8.3. Relative risk regression

Generalized linear models (GLMs) are often the statisticians' first choice for regression analysis of binary response data. The most prominent model of the GLM family is logistic regression.[3] A GLM which is mainly used in epidemiology but closely related to logistic regression is relative risk regression. The main difference between these two models is the fact that relative risk regression uses the log link and therefore estimates relative risks instead of odds ratios. Lumley, Kronmal, and Ma (2006) reviews the algorithms proposed in the literature to perform relative risk regression. Luo, Zhang, and Sun (2014) suggest to use a log-binomial model with

$$\underset{\beta}{\text{maximize}} \sum_{i=1}^{n} y_i \; X_{i*}\beta + \sum_{i=1}^{n} (1 - y_i) \; \log(1 - \exp(X_{i*}\beta)) \quad \text{subject to} \quad X\beta \leq 0. \tag{35}$$

Here X_{i*} refers to the ith row of the data matrix X, the constraint $X\beta \leq 0$ ensures that the estimated probabilities are in the interval [0, 1]. The log-binomial regression model can be formulated as a general nonlinear optimization problem:

```
R> logbin_gps <- function(y, X) {
+    loglikelihood <- function(beta) {
+      xb <- drop(X %*% beta)
+      if (any(xb > 0)) NaN else sum(y * xb + (1 - y) * log(1 - exp(xb)))
+    }
+
+    gradient <- function(beta) {
+      exb <- exp(drop(X %*% beta))
+      drop(crossprod(X, (y - exb) / (1 - exb)))
+    }
+
+    OP(F_objective(loglikelihood, n = ncol(X), G = gradient),
+      L_constraint(L = X, dir = leq(nrow(X)), rhs = double(nrow(X))),
+      bounds = V_bound(ld = -Inf, nobj = ncol(X)), maximum = TRUE)
+  }
```

which can be solved by any GPS which allows to specify linear constraints (e.g., using package **alabama**).

Alternatively the log-binomial regression model can be solved by any CP solver which supports the linear and the primal exponential cone (e.g., using package **scs**). The latter formulation has attractive theoretical and numerical convergence guarantees without the need to find suitable starting values.

```
R> logbin_cp <- function(y, X, rhs_eps = 1e-7) {
+    y_is_0 <- y == 0L
```

[3] An example for logistic regression can be found on the **ROI** web page at https://ROI.R-forge.R-project.org/.

```
+    n_y_is_0 <- sum(y_is_0)
+    o <- OP(c(y %*% X, double(n_y_is_0), rep(1, n_y_is_0)), maximum = TRUE)
+    L1 <- cbind(X, matrix(0, nrow(X), 2 * n_y_is_0))
+    log1exp <- function(xi, j, n_y_is_0) {
+      M <- matrix(0, nrow = 6, ncol = length(xi) + 2 * n_y_is_0)
+      M[1, seq_along(xi)] <- -xi
+      M[3, length(xi) + j] <- -1
+      M[4, length(xi) + n_y_is_0 + j] <- -1
+      M[6, length(xi) + j] <- 1
+      M
+    }
+    L2 <- mapply(log1exp, split(X[y_is_0,], seq_len(n_y_is_0)),
+      seq_len(n_y_is_0), MoreArgs = list(n_y_is_0 = n_y_is_0),
+      SIMPLIFY = FALSE)
+    rhs <- c(c(0, 1, 0), c(0, 1, 1))
+    rhs <- c(rep(-rhs_eps, nrow(X)), rep(rhs, n_y_is_0))
+    cones <- c(K_lin(nrow(X)), K_expp(2 * n_y_is_0))
+    L <- do.call(rbind, c(list(L1), L2))
+    constraints(o) <- C_constraint(L, cones, rhs)
+    bounds(o) <- V_bound(ld = -Inf, nobj = length(objective(o)))
+    o
+  }
```

To illustrate the estimation of GLMs with binary responses and log link using **ROI**, we generate a data set similar to an example in the **rms** (Harrell Jr 2020) manual.

```
R> generate_data <- function(n) {
+    treat <- factor(sample(c("a", "b", "c"), n, TRUE))
+    num.diseases <- sample(0:4, n, TRUE)
+    age <- rnorm(n, 50L, 10L)
+    cholesterol <- rnorm(n, 200L, 25L)
+    weight <- rnorm(n, 150L, 20L)
+    sex <- factor(sample(c("female", "male"), n, TRUE))
+
+    # Specify population model for log probability that Y = 1
+    L <- (-1 + 0.1 * (num.diseases - 2) + 0.045 * (age - 70)
+      + (log(cholesterol - 10) - 5.2) - 2 * (treat == "a")
+      + 0.5 * (treat == "b") - 0.5 * (treat == "c") )
+    # Simulate binary y to have Prob(y = 1) = exp(L)
+    y <- as.double(runif(n) < exp(L))
+
+    A <- cbind(intercept = 1, age, sex, weight,
+      logchol = log(cholesterol - 10), num.diseases,
+      treatb = (treat == "b"), treatc = (treat == "c"))
+    return(list(y = y, A = A))
+  }
R> suppressWarnings(RNGversion("3.5"))
```

```
R> set.seed(1234)
R> dat <- generate_data(1500L)
R> start <- c(log(0.2), double(ncol(dat$A) - 1))
R> prob_login_bin_gps <- logbin_gps(dat$y, dat$A)
R> s1 <- ROI_solve(prob_login_bin_gps, "alabama", start = start)
R> solution(s1)

[1] -11.655999855   0.051756340   0.054627947  -0.005642101
[5]   1.044477583   0.066972373   2.649430289   1.735273305

R> prob_login_bin_cp <- logbin_cp(dat$y, dat$A)
R> s2 <- ROI_solve(prob_login_bin_cp, solver = "ecos")
R> head(solution(s2), ncol(dat$A))

[1] -11.665400207   0.051763544   0.054607482  -0.005639921
[5]   1.046087685   0.066980548   2.649563917   1.735398851

R> obj_fun <- objective(prob_login_bin_gps)
R> obj_fun(head(solution(s2), ncol(dat$A))) - obj_fun(solution(s1))

[1] 6.43562e-06
```

We see that both approaches yield a similar result.

8.4. 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) and 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,

$$\underset{M_i}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^{m} \|X_{i*} - M_{i*}\|_2^2 + \lambda \sum_{i<j} \|M_{i*} - M_{j*}\|_q \tag{36}$$

where $q \in \{1, 2, \infty\}$, $X \in \mathbb{R}^{m \times n}$ is the data matrix and M_{i*} the ith 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 36 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).

At least two implementations of SON clustering exist in R, Hocking *et al.* (2011) provide their code on R-Forge (Hocking 2015) and Chi and Lange (2015) provide a fast implementation of SON clustering on CRAN (Chi and Lange 2014). A **ROI** formulation as SOCP of SON clustering can be found in the supplementary material.

8.5. 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. 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,

$$\underset{\Theta \succ 0}{\text{minimize}} \quad -\log(\det(\Theta)) + \text{tr}(S\ \Theta) + \lambda\ \|X\|_1, \tag{37}$$

where the data matrix $X \in \mathbb{R}^{n \times p}$ 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 supplementary material.

9. Conclusions

In this paper we presented the **ROI** package and its extensions. **ROI** provides a consistent way to model OPs in R. **ROI** makes strong use of R's generic functions, such that users already familiar with R are not obliged 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 nineteen 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 selection, 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**. As package **ROI.plugin.msbinlp** shows, 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 and possible future work:

- Currently, **ROI** is lacking a plug-in capable of solving general nonlinear optimization problems with mixed integer constraints. Within the COIN-OR project the **Couenne** (Belotti, Lee, Liberti, Margot, and Wächter 2009) and the **Bonmin** (Bonami and Lee 2013) solvers are designed to try to obtain a global solution for non-convex MINLPs. Therefore both solvers would be a valuable extension of **ROI**.

- Another popular solver from the COIN-OR project is **Ipopt** (Wächter and Biegler 2006), which aims to solve general nonlinear optimization problems. As mentioned before, there exists already the **ipoptr** package, but since there are many steps needed for the installation we assume it is currently not accessible to many R users. Therefore a **ROI** plugin installable directly from CRAN would be preferable.

- Add a reader for the QPLIB file format (Furini *et al.* 2017).

- Add plotting methods.

- Explore possibilities of supervised solver recommendation.

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

Acknowledgments

The authors would like to thank David Meyer for his thoughtful advice on the original conceptual framework of **ROI** which is still part of the package. Furthermore, we would like to thank Hans W. Borchers as well as the two anonymous reviewers for their comments on early versions of this paper as well as pointing us to examples for showcasing the strengths of **ROI**.

References

Alizadeh F, Goldfarb D (2003). "Second-Order Cone Programming." *Mathematical Programming*, **95**(1), 3–51. doi:10.1007/s10107-002-0339-5.

Andersen MS, Dahl J, Liu Z, Vandenberghe L (2012). "Interior-Point Methods for Large-Scale Cone Programming." In *Optimization for Machine Learning*, chapter 3, pp. 55–83. MIT Press. 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 https://cvxopt.org/.

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

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.

Ben-Tal A, Nemirovski A (2001). *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898718829.

Ben-Tal A, Nemirovski A (2019). "Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications." URL http://www2.isye.gatech.edu/~nemirovs/LMCO_LN.pdf.

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), 299–316. doi:10.1007/s10288-011-0165-9.

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

Berkelaar M (2020). ***lpSolve**: Interface to **lp_solve** v. 5.5 to Solve Linear/Integer Programs*. R package version 5.6.15, 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. doi:10.1214/15-aos1388.

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.

Bihorel S, Baudin M (2018). ***neldermead**: R Port of the Scilab **neldermead** Module*. R package version 1.0-11, 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-Verlag. doi:10.1007/BFb0121223.

Bixby RE (2012). "A Brief History of Linear and Mixed-Integer Programming Computation." *Documenta Mathematica*, **ISMP**, 107–121. URL https://www.math.uni-bielefeld.de/documenta/vol-ismp/25_bixby-robert.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.

Boros E, Hammer PL (2002). "Pseudo-Boolean Optimization." *Discrete Applied Mathematics*, **123**(1–3), 155–225. doi:10.1016/s0166-218x(01)00341-9.

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

Canty A, Ripley BD (2020). ***boot**: Bootstrap Functions (Originally by Angelo Canty for S)*. R package version 1.3-25, 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 https://dial.uclouvain.be/pr/boreal/en/object/boreal%3A28538/datastream/PDF_01/view.

Chen X, Yin X (2019). ***NlcOptim**: Solve Nonlinear Optimization with Nonlinear Constraints*. R package version 0.6, URL https://CRAN.R-project.org/package=NlcOptim.

Chi EC, Lange K (2014). ***cvxclustr**: Splitting Methods for Convex Clustering*. R package version 1.1.1, URL https://CRAN.R-project.org/package=cvxclustr.

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

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

Corrada Bravo H, Borchers B (2020). ***Rcsdp**: R Interface to the **CSDP** Semidefinite Programming Library*. R package version 0.1.57.1, URL https://CRAN.R-project.org/package=Rcsdp.

Czyzyk J, Mesnier MP, Moré JJ (1998). "The NEOS Server." *IEEE Journal on Computational Science and Engineering*, **5**(3), 68–75. doi:10.1109/99.714603.

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.

Diamond S, Boyd S (2016). "**CVXPY**: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research*, **17**(83), 1–5.

Dolan ED (2001). "The NEOS Server 4.0 Administrative Guide." *Technical Memorandum ANL/MCS-TM-250*, Mathematics and Computer Science Division, Argonne National Laboratory.

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.

Dykstra RL (1983). "An Algorithm for Restricted Least Squares Regression." *Journal of the American Statistical Association*, **78**(384), 837–842. doi:10.1080/01621459.1983.10477029.

Ferreau HJ, Kirches C, Potschka A, Bock HG, 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.

Ferreau HJ, Potschka A, Kirches C (2018). ***qpOASES** Webpage*. URL https://www.qpOASES.org/.

Fine S, Scheinberg K (2001). "Efficient SVM Training Using Low-Rank Kernel Representations." *Journal of Machine Learning Research*, **2**(Dec), 243–264.

Fischetti M, Salvagnin D (2010). "Pruning Moves." *INFORMS Journal on Computing*, **22**(1), 108–119. doi: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). "AMPL: A Mathematical Programming Language." In SW Wallace (ed.), *Algorithms and Model Formulations in Mathematical Programming*, pp. 150–151. Springer-Verlag, New York. doi:10.1007/978-3-642-83724-1.

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

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.

Fu A, Narasimhan B (2019). ***ECOSolveR**: Embedded Conic Solver in R*. R package version 0.5.3, URL https://CRAN.R-project.org/package=ECOSolveR.

Fu A, Narasimhan B, Boyd S (2020). "**CVXR**: An R Package for Disciplined Convex Optimization." *Journal of Statistical Software*, **94**(14), 1–34. doi:10.18637/jss.v094.i14.

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.

Geyer CJ, Meeden GD, Fukuda K (2019). ***rcdd**: R Interface to (Some of) **cddlib***. R package version 1.2-2, URL https://CRAN.R-project.org/package=rcdd.

Ghalanos A, Theußl S (2015). ***Rsolnp**: General Non-Linear Optimization Using Augmented Lagrange Multiplier Method*. R package version 1.16, URL https://CRAN.R-project.org/package=Rsolnp.

Gilbert P, Varadhan R (2019). ***numDeriv***: *Accurate Numerical Derivatives*. R package version 2016.8-1.1, URL https://CRAN.R-project.org/package=numDeriv.

Goldfarb D, Idnani A (1983). "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs." *Mathematical Programming*, **27**(1), 1–33. doi: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, URL http://cvxr.com/cvx.

Gropp W, Moré JJ (1997). "Optimization Environments and the NEOS Server." In MD Buhman, A Iserles (eds.), *Approximation Theory and Optimization*, pp. 167–182. Cambridge University Press.

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

Harrell Jr FE (2020). ***rms***: *Regression Modeling Strategies*. R package version 6.0-0, URL https://CRAN.R-project.org/package=rms.

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

Helwig NE (2018). ***Dykstra***: *Quadratic Programming Using Cyclic Projections*. R package version 1.0-0, URL https://CRAN.R-project.org/package=Dykstra.

Hochreiter R, Schwendinger F (2020). ***ROI.plugin.neos***: *NEOS Plug-in for the R Optimization Interface*. R package version 0.3-2, URL https://CRAN.R-project.org/package=ROI.plugin.neos.

Hocking TD (2015). ***clusterpath***: *Fast Agglomerative Convex Clustering, Non-**Rcpp** Implementation*. R package version 1.2, URL https://R-Forge.R-project.org/projects/clusterpath/.

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)*, pp. 745–752. ACM, New York.

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 (2019). ***slam***: *Sparse Lightweight Arrays and Matrices*. R package version 0.1-47, 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.

Hornik K, Meyer D, Schwendinger F, Theußl S (2020). ***ROI**: R Optimization Infrastructure*. R package version 1.0-0, URL https://CRAN.R-project.org/package=ROI.

IBM ILOG (2019). "**CPLEX** Optimization Studio." URL https://www.ibm.com/analytics/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." In J Kallrath (ed.), *Modeling Languages in Mathematical Optimization*, chapter 1, pp. 3–24. Springer-Verlag, Boston, MA. doi:10.1007/978-1-4613-0215-5_1.

Karatzoglou A, Smola A, Hornik K (2019). ***kernlab**: Kernel-Based Machine Learning Lab*. R package version 0.9-29, 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**(9), 1–20. doi:10.18637/jss.v011.i09.

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: Mixed Integer Programming Library version 5." *Mathematical Programming Computation*, **3**(2), 103–163. doi:10.1007/s12532-011-0025-9.

Koecher M (1957). "Positivitätsbereiche im R^n." *American Journal of Mathematics*, **79**(3), 575–596. doi:10.2307/2372563.

Koenker R, Mizera I (2014). "Convex Optimization in R." *Journal of Statistical Software*, **60**(5), 1–23. doi:10.18637/jss.v060.i05.

Konis K, Schwendinger F (2020). ***lpSolveAPI**: R Interface to **lp_solve** 5.5.2.0-17*. R package version 5.5.2.0-17.7, URL https://CRAN.R-project.org/package=lpSolveAPI.

Land AH, Doig AG (1960). "An Automatic Method for Solving Discrete Programming Problems." *Econometrica*, **28**(3), 497–520. doi:10.2307/1910129.

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

Liao X, Meyer MC (2014). "**coneproj**: An R Package for the Primal or Dual Cone Projections with Routines for Constrained Regression." *Journal of Statistical Software*, **61**(12), 1–22. doi:10.18637/jss.v061.i12.

Linderoth JT, Ralphs TK (2005). "Noncommercial Software for Mixed-Integer Linear Programming." In *Integer Programming*, Operations Research Series, chapter 10, pp. 253–303. CRC Press. doi:10.1201/9781420039597.

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. URL http://www.diva-portal.org/smash/get/diva2:650707/FULLTEXT01.pdf.

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. doi:10.1016/s0024-3795(98)10032-0.

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

Lumley T (2020). ***leaps**: Regression Subset Selection*. R package version 3.1, URL https://CRAN.R-project.org/package=leaps.

Lumley T, Kronmal R, Ma S (2006). "Relative Risk Regression in Medical Research: Models, Contrasts, Estimators, and Algorithms." Collection of Biostatistics Research Archive.

Luo J, Zhang J, Sun H (2014). "Estimation of Relative Risk Using a Log-Binomial Model with Constraints." *Computational Statistics*, **29**(5), 981–1003. doi:10.1007/s00180-013-0476-8.

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

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

Meyer D (2019). ***registry**: Infrastructure for R Package Registries*. R package version 0.5-1, URL https://CRAN.R-project.org/package=registry.

Meyer D, Hornik K (2019). ***relations**: Data Structures and Algorithms for Relations*. R package version 0.6-9, URL https://CRAN.R-project.org/package=relations.

Meyer MC, Liao X (2018). ***coneproj**: Primal or Dual Cone Projections with Routines for Constrained Regression*. R package version 1.14, URL https://CRAN.R-project.org/package=coneproj.

Miller A (2002). *Subset Selection in Regression*. CRC Press. doi:10.1201/9781420035933.

MOSEK ApS (2017). *Introducing the **MOSEK** Optimization Suite*. Version 8.1 (Revision 27), URL https://www.mosek.com/documentation/.

MOSEK ApS (2019). ***Rmosek**: The R to **MOSEK** Optimization Interface*. R package version 1.3.5, URL https://CRAN.R-project.org/package=Rmosek.

Mullen K (2014a). "Continuous Global Optimization in R." *Journal of Statistical Software*, **60**(6), 1–45. doi:10.18637/jss.v060.i06.

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.

Mullen K, Ardia D, Gil D, Windover D, Cline J (2011). "**DEoptim**: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software*, **40**(6), 1–26. doi:10.18637/jss.v040.i06.

Nash JC (2014a). "On Best Practice Optimization Methods in R." *Journal of Statistical Software*, **60**(2), 1–14. doi:10.18637/jss.v060.i02.

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 (2018). ***Rvmmin**: Variable Metric Nonlinear Function Minimization*. R package version 2018-4.17, 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. doi:10.18637/jss.v043.i09.

Nemirovski A (2006). "Advances in Convex Optimization: Conic Programming." In *Proceedings of International Congress of Mathematicians*, pp. 413–444. doi:10.4171/022.

Nesterov Y (2004). *Introductory Lectures on Convex Optimization*. Springer-Verlag. doi: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-Verlag. 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*, **169**(3), 1042–1068. doi:10.1007/s10957-016-0892-3.

O'Donoghue B, Schwendinger F (2019). ***scs**: Splitting Conic Solver*. R package version 1.3-2, URL https://CRAN.R-project.org/package=scs.

Ormerod JT, Wand MP (2020). ***LowRankQP**: Low Rank Quadratic Programming*. R package version 1.0.4, 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.

Pfaff B (2020). **rneos**: *XML-RPC Interface to NEOS*. R package version 0.4-0, URL https://CRAN.R-project.org/package=rneos.

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

Racine JS, Nie Z (2019). **crs**: *Categorical Regression Splines*. R package version 0.15-31.1, URL https://CRAN.R-project.org/package=crs.

Ralphs TK, Güzelsoy M (2005). "The **SYMPHONY** Callable Library for Mixed Integer Programming." In *The Next Wave in Computing, Optimization, and Decision Technologies*, chapter 2, pp. 61–76. Springer-Verlag. doi: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 (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org.

Roettger M, Gelius-Dietrich G (2020a). **clpAPI**: *R Interface to C API of COIN-OR* **Clp**. R package version 1.2.13, URL https://CRAN.R-project.org/package=clpAPI.

Roettger M, Gelius-Dietrich G (2020b). **glpkAPI**: *R Interface to C API of* **GLPK**. R package version 1.3.2, URL https://CRAN.R-project.org/package=glpkAPI.

Rosenbrock HH (1960). "An Automatic Method for Finding the Greatest or Least Value of a Function." *The Computer Journal*, **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.

Schwendinger F (2017). **ROI.models.globalOptTests**: **ROI** *Optimization Problems Based on* **globalOptTests**. R package version 1.1, URL https://CRAN.R-project.org/package=ROI.models.globalOptTests.

Schwendinger F (2019a). **ROI.models.netlib**: **ROI** *Optimization Problems Based on NETLIB-LP*. R package version 1.1, URL https://CRAN.R-project.org/package=ROI.models.netlib.

Schwendinger F (2019b). **ROI.tests**: **ROI** *Tests*. R package version 0.0-2, URL https://R-Forge.R-project.org/projects/roi/.

Schwendinger F (2020). **ROI.plugin.qpoases**: **qpOASES** *Plugin for the R Optimization Infrastructure*. R package version 0.3-3, URL https://CRAN.R-project.org/package=ROI.plugin.qpoases.

Schwendinger F, Theußl S (2019). **ROI.models.miplib**: *R Optimization Infrastructure: MIPLIB 2010 Benchmark Instances*. R package version 0.0-2, URL https://CRAN.R-project.org/package=ROI.models.miplib.

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.

Stellato B, Banjac G, Goulart P, Bemporad A, Boyd S (2017). "**OSQP**: An Operator Splitting Solver for Quadratic Programs." In *2018 UKACC 12th International Conference on Control (CONTROL)*. doi:10.1109/control.2018.8516834.

Stellato B, Banjac G, Goulart P, Boyd S (2019). ***osqp**: Quadratic Programming Solver Using the **OSQP** Library*. R package version 0.6.0.3, URL https://CRAN.R-project.org/package=osqp.

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.

The MathWorks Inc (2019). *MATLAB – The Language of Technical Computing, Version R2019a*. Natick, Massachusetts. 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, Schwendinger F (2020). "CRAN Task View: Optimization and Mathematical Programming." Version 2020-05-21, 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 (2019). ***Rglpk**: R/GNU Linear Programming Kit Interface*. R package version 0.6-4, URL https://CRAN.R-project.org/package=Rglpk.

Theußl S, Zeileis A (2009). "Collaborative Software Development Using R-Forge." *The R Journal*, **1**(1), 9–14. doi:10.32614/rj-2009-007.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B*, **58**(1), 267–288. doi:10.1111/j.2517-6161.1996.tb02080.x.

Turlach BA, Weingessel A (2019). ***quadprog**: Functions to Solve Quadratic Programming Problems*. R package version 1.5-8, 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 First Workshop for High Performance Technical Computing in Dynamic Languages*, HPTCDL'14, pp. 18–28. IEEE Press, Piscataway. doi:10.1109/hptcdl.2014.5.

Vandenberghe L, Boyd S (1996). "Semidefinite Programming." *SIAM Review*, **38**(1), 49–95. 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*, **32**(4), 1–26. doi:10.18637/jss.v032.i04.

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. doi:10.1007/s10107-004-0559-y.

Wagner HM (1959). "Linear Programming Techniques for Regression Analysis." *Journal of the American Statistical Association*, **54**(285), 206–212. doi:10.2307/2282146.

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/.

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, Johnson SG (2020). ***nloptr**: R Interface to **NLopt***. R package version 1.2.2.1, URL https://CRAN.R-project.org/package=nloptr.

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

Zhu Z, Ye Y (2020). ***Rdsdp**: R Interface to the **DSDP** Semidefinite Programming Library*. R package version 1.0.5, URL https://CRAN.R-project.org/package=Rdsdp.

A. Solver overview

	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	malschains	**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	pureCMAES	**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
74	abc_optim	**ABCoptim**	Global	Box	No	No	No
75	CEoptim	**CEoptim**	Global	No	No	No	No
76	manifold.optim	**ManifoldOptim**	Local	No	Yes	Yes	No
77	ALO	**metaheuristicOpt**	Global	Box	No	No	No
78	DA	**metaheuristicOpt**	Global	Box	No	No	No
79	FFA	**metaheuristicOpt**	Global	Box	No	No	No
80	GA	**metaheuristicOpt**	Global	Box	No	No	No
81	GOA	**metaheuristicOpt**	Global	Box	No	No	No
82	GWO	**metaheuristicOpt**	Global	Box	No	No	No
83	HS	**metaheuristicOpt**	Global	Box	No	No	No
84	MFO	**metaheuristicOpt**	Global	Box	No	No	No
85	PSO	**metaheuristicOpt**	Global	Box	No	No	No
86	SCA	**metaheuristicOpt**	Global	Box	No	No	No
87	WOA	**metaheuristicOpt**	Global	Box	No	No	No
88	SD	**mize**	Local	No	Yes	Yes	No
89	BFGS	**mize**	Local	No	Yes	Yes	No
90	SR1	**mize**	Local	No	Yes	Yes	No
91	L-BFGS	**mize**	Local	No	Yes	Yes	No

92	CG	**mize**	Local	No	Yes	Yes	No
93	TN	**mize**	Local	No	Yes	Yes	No
94	NAG	**mize**	Local	No	Yes	Yes	No
95	DBD	**mize**	Local	No	Yes	Yes	No
96	Momentum	**mize**	Local	No	Yes	Yes	No
97	n1qn1	**n1qn1**	Local	No	Yes	No	No
98	qnbd	**n1qn1**	Local	Box	Yes	No	No
99	tnbc	**Rtnmin**	Local	Box	Yes	No	No
100	COBRA	**SACOBRA**	Local	Box	No	No	No

Table 5: GPSs in R and their capability to handle type, constraint, gradient (G), Hessian (H), Jacobian (J) information.

B. 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	Quadratically constraint quadratic programming
QP	Quadratic programming
SDP	Semidefinite programming
SOCP	Second order cone programming

Table 6: Abbreviations used for the classes of optimization problems.

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

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2020, Volume 94, Issue 15 *Submitted:* 2017-10-22
doi:10.18637/jss.v094.i15 *Accepted:* 2019-06-05