Optimization in R

Computational Economics Practice
Winter Term 2015/16
ISR
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
## Today’s Lecture

### Objectives

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Being able to characterize different optimization problems</td>
</tr>
<tr>
<td>2</td>
<td>Learn how to solve optimization problems in R</td>
</tr>
<tr>
<td>3</td>
<td>Understand the idea behind common optimization algorithms</td>
</tr>
</tbody>
</table>
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
Mathematical Optimization

- Optimization uses a rigorous mathematical model to determine the most efficient solution to a described problem
- One must first identify an objective
  - Objective is a quantitative measure of the performance
  - Examples: profit, time, cost, potential energy
  - In general, any quantity (or combination thereof) represented as a single number

![Graph showing optimization with variables and function evaluations]
Classification of Optimization Problems

Common groups

1. Linear Programming (LP)
   - Objective function and constraints are both linear
   - \( \min_{x} c^T x \) s.t. \( Ax \leq b \) and \( x \geq 0 \)

2. Quadratic Programming (QP)
   - Objective function is quadratic and constraints are linear
   - \( \min_{x} x^T Qx + c^T x \) s.t. \( Ax \leq b \) and \( x \geq 0 \)

3. Non-Linear Programming (NLP): objective function or at least one constraint is non-linear

Solution strategy

- Each problem class requires its own algorithms
  - \( \rightarrow \) R has different packages for each class
- Often, one distinguishes further, e.g. constrained vs. unconstrained
  - Constrained optimization refers to problems with equality or inequality constraints in place
Optimization in R

- **Common R packages** for optimization

<table>
<thead>
<tr>
<th>Problem type</th>
<th>Package</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>General purpose (1-dim.)</td>
<td>Built-in</td>
<td>optimize(...)</td>
</tr>
<tr>
<td>General purpose (n-dim.)</td>
<td>Built-in</td>
<td>optim(...)</td>
</tr>
<tr>
<td>Linear Programming</td>
<td>lpSolve</td>
<td>lp(...)</td>
</tr>
<tr>
<td>Quadratic Programming</td>
<td>quadprog</td>
<td>solve.QP(...)</td>
</tr>
<tr>
<td>Non-Linear Programming</td>
<td>optimize</td>
<td>optimize(...)</td>
</tr>
<tr>
<td></td>
<td>optimx</td>
<td>optimx(...)</td>
</tr>
<tr>
<td>General interface</td>
<td>ROI</td>
<td>ROI_solve(...)</td>
</tr>
</tbody>
</table>

- **All available packages** are listed in the CRAN task view “Optimization and mathematical programming”

  **URL:** https://cran.r-project.org/web/views/Optimization.html
Optimization in R

- Basic argument structure of a solver is always the same
- Format of such a generic call

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

- Routines usually provide an interface, which allows to switch between different algorithms

**Built-in optimization routines**

- `optimize(...)` is for **1-dimensional** optimization
- `optim(...)` is for **n-dimensional** optimization
  - Golden section search (with successive 2nd degree polynomial interpolation)
  - Aimed at continuous functions
  - Switching to dedicated routines usually achieves a better convergence
Built-In Optimization in R

- `optim(x0, fun, ...)` is for \( n \)-dimensional general purpose optimization
  - Argument \( x0 \) sets the initial values of the search
  - `fun` specifies a function to optimize over
  - Optional, named argument `method` chooses an algorithm

Example

- Define objective function

```r
f <- function(x) 2*(x[1]-1)^2 + 5*(x[2]-3)^2 + 10
```
Built-In Optimization in R

- Call optimization routine
  
  ```r
  r <- optim(c(1, 1), f)
  ```

- Check if the optimization converged to a minimum
  
  ```r
  r$convergence == 0  # TRUE if converged
  ```

- Optimal input arguments
  
  ```r
  r$par
  ```

- Objective at the minimum
  
  ```r
  r$value
  ```
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
Linear Programming

Mathematical specification

1 Matrix notation

\[
\begin{align*}
\min_x & \quad c_1^T x_1 + c_2^T x_2 + \cdots + c_n^T x_n \\
\text{subject to} & \quad \sum_{i=1}^m a_{ij} x_j \geq b_j, \quad x_j \geq 0
\end{align*}
\]

2 Alternative formulation in a more compact form

\[
\begin{align*}
\min_x & \quad c^T x = \min x_1 c_1 + \cdots + x_n c_n \\
\text{subject to} & \quad Ax \geq b, \quad x \geq 0
\end{align*}
\]
Linear Programming

Example

1. Objective function
   - Goal is to maximize the total profit
   - Products A and B are sold at €25 and €20 respectively

2. Resource constraints
   - Product A requires 20 resource units, product B needs 12
   - Only 1800 resource units are available per day

3. Time constraints
   - Both products require a production time of $\frac{1}{15}$ hour
   - A working day has a total of 8 hour
Linear Programming

Problem formulation

- Variables: let $x_1$ denote the number of produced items of A and $x_2$ of B
- **Objective function** maximizes the total sales

\[
Sales_{\text{max}} = \max_{x_1, x_2} 25x_1 + 20x_2 = \max_{x_1, x_2} \begin{bmatrix} 25 \\ 20 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]

Constraints

- Constraints for resources and production time are given by

\[
20x_1 + 12x_2 \leq 1800
\]
\[
\frac{1}{15}x_1 + \frac{1}{15}x_2 \leq 8
\]

- Both constraints can also be rewritten in matrix form

\[
\begin{bmatrix} 20 & 12 \\ \frac{1}{15} & \frac{1}{15} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \preceq \begin{bmatrix} 1800 \\ 8 \end{bmatrix}
\]

Optimization in R: LP
Visualization of **objective function** and both constraints
Linear Programming in R

- Package \texttt{lpSolve} contains routine \texttt{lp(...)} to solve linear optimization problems

- General syntax

\begin{verbatim}
lp(direction="min", objective.in, const.mat, const.dir, const.rhs)
\end{verbatim}

- \texttt{direction} controls whether to minimize or maximize
- Coefficients $c$ are encoded a vector \texttt{objective.in}
- Constraints $A$ are given as a matrix \texttt{const.mat} with directions \texttt{const.dir}
- Constraints $b$ are inserted as a vector \texttt{const.rhs}
Linear Programming in R

- Loading the package

```r
library(lpSolve)
```

- Encoding and executing the previous example

```r
objective.in <- c(25, 20)
const.mat <- matrix(c(20, 12, 1/15, 1/15), nrow=2, byrow=TRUE)
const.rhs <- c(1800, 8)
const.dir <- c("<=", "<=")
optimum <- lp(direction="max", objective.in, const.mat, const.dir, const.rhs)
```

- Optimal values of $x_1$ and $x_2$

```r
optimum$solution
## [1] 45 75
```

- Objective at minimum

```r
optimum$objval
## [1] 2625
```
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
Quadratic Programming

Mathematical specification

1 Compact form

$$\min_x \frac{1}{2} x^T D x - d^T x \text{ subject to } A^T x \geq b$$

2 Matrix notation

$$\min_x \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^T \begin{bmatrix} d_{11} & d_{12} & \ldots & d_{1n} \\ d_{21} & d_{22} & \ldots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \ldots & d_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{ s.t. } \begin{bmatrix} a_{11} & a_{12} & \ldots \\ a_{21} & a_{22} & \ldots \\ \vdots & \vdots & \ddots \\ a_{m1} & a_{m2} & \ldots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \geq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$
Quadratic Programming

- Parameter mapping in R
  - Quadratic coefficients $D$ are mapped to $D_{\text{mat}}$
  - Linear coefficients $d$ are mapped to $d_{\text{vec}}$
  - Constraints matrix $A$ is mapped to $A_{\text{mat}}$
  - Constraint equalities or inequalities $b$ are provided in $b_{\text{vec}}$
  - Parameter $\text{meq} = n$ sets the first $n$ entries as equality constraints; all further constraints are inequality

- Function call with package `quadprog`

  ```r
  require(quadprog)
  solve.QP(Qmat, dvec, Amat, bvec, meq)
  ```

- Many problems can formulated in quadratic form, e.g., portfolio optimization, circus tent problem, demand response, ...
Example Circus Tent

Question
How to bring this into quadratic form?
Example Circus Tent

- How to calculate the height of the tent at every point?
- Tent height at each grid point \((x, y)\) is given by \(u(x, y)\)
- Tent sheet settles into minimal energy state \(E[u]\) for each height \(u\)
- Use the Dirichlet energy to estimate \(E[u]\) of \(u\)
- We discretize the energy and ultimately come up with

\[
E[u] \approx \frac{h_x h_y}{2} u^T L u
\]

which is quadratic

Full description

### Overview: Non-Linear Optimization

<table>
<thead>
<tr>
<th>Dimensionality</th>
<th>One-dimensional</th>
<th>Multi-dimensional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Non-gradient based</td>
<td>Gradient based</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Golden Section Search</td>
<td>Gradient descent methods</td>
</tr>
<tr>
<td>Package</td>
<td>stats</td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td>optimize()</td>
<td>CG</td>
</tr>
</tbody>
</table>
One-Dimensional Non-linear Programming

- Golden Section Search can be used to solve one-dimensional non-linear problems
- Basic steps:
  1. Golden Ratio defined as $\varphi = \frac{\sqrt{5}-1}{2} = 0.618$
  2. Pick an interval $[a, b]$ containing the optimum
  3. Evaluate $f(x_1)$ at $x_1 = a + (1 - \varphi)(b - a)$ and compare with $f(x_2)$ at $x_2 = a + \varphi(b - a)$
  4. If $f(x_1) < f(x_2)$, continue the search in the interval $[a, x_1]$, else $[x_2, b]$
- Implementation in R with built-in packages

```r
optimize(f = , interval = , ..., tol = .Machine$double.eps^0.25)
```
Golden Section Search Iterations

- Minimize $f(x) = (x - \frac{1}{3})^2$ with `optimize`
- Use `print` to show steps of $x$

```r
f <- function(x) (print(x) - 1/3)^2
xmin <- optimize(f,
                 interval = c(0, 1),
                 tol = 0.0001)
## [1] 0.381966
## [1] 0.618034
## [1] 0.236068
## [1] 0.333333
## [1] 0.3333
## [1] 0.3333667
## [1] 0.3333333

xmin
## $minimum
## [1] 0.3333333
##
## $objective
## [1] 0
```

Optimization in R: NLP
Example: Non-differentiable function with optimx()

- Does not require differentiability, e.g., \( f(x) = |x - 2| + 2|x - 1| \)

\[
f \left( x \right) = |x - 2| + 2|x - 1|
\]

```r
f <- function(x) return(abs(x-2) + 2*abs(x-1))
xmin <- optimize(f, interval = c(0, 3), tol = 0.0001)
xmin

# $minimum
# [1] 1.000009
#
# $objective
# [1] 1.000009
```

```r
plot(f, 0, 3)
```
Non-Linear Multi-Dimensional Programming

- Collection of non-linear methods in package \texttt{optimx}

\begin{verbatim}
require(optimx)
optimx(par, fn, gr=Null, Hess=Null, lower=inf,
       upper=inf, method='', itnmax=Null, ...)
\end{verbatim}

- Multiple optimization algorithms possible
  - Gradient based: Gradient descent methods (’CG’)
  - Hessian based: Newton and quasi-Newton methods (’BFGS’, ’L-BFGS-B’)
  - Non-gradient based: Golden section search, Nelder-Mead, ... (’Nelder-Mead’)

- The default method of \texttt{optimx} is "Nelder-Mead"; if constraints are provided, "L-BFGS-B" is used
Optimx parameters

- **Important input parameters**
  - `par`: Initial values for the parameters (vector)
  - `fn`: Objective function with minimization parameters as input
  - `control`: List of control parameters

- **Important output parameters**
  - `pn`: Optimal set of parameters
  - `value`: Minimum value of `fn`
  - `fevals`: Number of calls to `fn`
  - `gevals`: Number of calls to the gradient calculation
  - `xtimes`: Execution time in seconds
Himmelblau’s function

Definition

\[ f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \] (2)

Himmelblau’s function (Zimmermann 2007) is a popular multi-modal function to benchmark optimization algorithms.

Four equivalent minima are located at \( f(-3.7793; -3.2832) = 0 \), \( f(-2.8051; 3.1313) = 0 \), \( f(3; 2) = 0 \) and \( f(3.5844; -1,8481) = 0 \).
Implementation of Himmelblau’s function

```r
fn <- function(para){  # Vector of the parameters
  matrix.A <- matrix(para, ncol=2)
  x <- matrix.A[,1]
  y <- matrix.A[,2]
  f.x <- (x^2+y-11)^2+(x+y^2-7)^2
  return(f.x)
}
par <- c(1,1)
```

Optimization in R: NLP
Plot of Himmelblau’s function

```r
xy <- as.matrix(expand.grid(seq(-5,5,length = 101),
                              seq(-5,5,length = 101)))
colnames(xy) <- c("x", "y")
df <- data.frame(fnxy = fn(xy), xy)

library(lattice)
wireframe(fnxy ~ x*y, data = df, shade = TRUE, drape=FALSE,
          scales = list(arrows = FALSE),
          screen = list(z=-240, x=-70, y=0))
```
Gradient-Free Method: Nelder-Mead

- Nelder Mead solves multi-dimensional equations using function values
- Works also with non-differentiable functions
- Basic steps:
  1. Choose a simplex consisting of \( n + 1 \) points \( p_1, p_2, \ldots, p_{n+1} \) are chosen with \( n \) being the number of variables
  2. Calculate \( f(p_i) \) and sort by size, e.g., \( f(p_1) \leq f(p_2) \leq f(p_{n+1}) \)
  3. Check if the best value is good enough, if so, stop
  4. Drop the point with highest \( f(p_i) \) from the simplex
  5. Choose a new point to be added to the simplex
  6. Continue with step 2
- Different options and implementations to choose new point, often these are combined:
  - Reflection to the center of gravity of the simplex formed by the other points and further expansion in the same direction
  - Contraction of the 'worst' point towards the center of the simplex
  - Compression, e.g., contraction of all points towards the 'best' point
  - Usage of the gradient to determine direction of next point
Nelder mead search

Iteration 1

log2(y + 1)

Optimization in R: NLP
Gradient-Based: Conjugate Gradients

- Use the first derivative to obtain gradient for the search direction
- Search direction $s_n$ of the next point results from the negative gradient of the last point
- Basic steps
  1. Calculate search direction $s_n = -\Delta f(x_n)$
  2. Pick next point $x_{n+1}$ by moving with step size $a_n$ in the search direction; step size $a$ can be fixed or variable
  3. Repeat until $\Delta f(x_n) = 0$ or another stopping criterion
- Results in a "zig-zagging" movement towards the minimum
One Dimensional CG

- Find minima of the function $f(x) = -\sin(x) - (0.25x - 2)^3$
Gradient descent search path

- $a_n$ for gradient descent is fixed at 0.01
- The algorithm stops when its within 0.1 of a zero
Newton-Raphson

- Newton’s method is often used to find the zeros of a function
- Minima fulfill the conditions $f'(x^*) = 0$ and $f''(x^*) > 0$, so Newton can be used to find the zeros of the first derivative
- Basic steps
  1. Approximate the function at the starting point with a linear tangent (e.g., second order Taylor series) $t(x) \approx f'(x_0) + (x - x_0)f''(x_0)$
  2. Find the intersect $t(x_i) = 0$ as an approximation for $f'(x^*) = 0$
  3. Use the intersect as new starting point
  4. Finally, the algorithm $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$ is repeated until $f'(x_n)$ is close enough to 0.
Visualization of Newton-Raphson Search

- Find minima of \( f(x) = \frac{1}{4}(x - 3)^4 + \frac{1}{3}x^3 + 5x + 15 \)
Newton Raphson search paths

The algorithm stops when its within 0.1 of a zero
Hessian-Based: BFGS and L-BFGS-B

- Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm builds on the idea of Newton’s method to take gradient information into account
- Gradient information comes from an approximation of the Hessian matrix
- No guaranteed conversion; especially problematic if Taylor expansion does not fit well
- L-BFGS-B stands for limited-memory-BFGS-box
  - Extension of BFGS
  - Memory efficient implementation
  - Additionally handles box constraints
Comparison Newton and Gradient Descent

Iteration 1

\[
\begin{align*}
\text{log}_2(y + 1) & \\
\text{x1} & \\
\text{x2} &
\end{align*}
\]
Comparison Newton and Gradient Descent

- $\alpha_n$ for gradient descent is fixed at 0.01.
- Both algorithms stop if they are within 0.01 of a zero.
Method Comparison with optimx()

- Optimization comparison requires `optimx` package

```r
library(optimx)
```

- Nelder-Mead

```r
optimx(par, fn, method = "Nelder-Mead")
```

```
# p1  p2   value fevals gevals niter convcode
# Nelder-Mead 2.999995 2.000183 5.56163e-07 67 NA NA 0
#           kkt1 kkt2 xtimes
# Nelder-Mead FALSE TRUE 0
```

- Conjugate gradients

```r
optimx(par, fn, method = "CG")
```

```
# p1  p2   value fevals gevals niter convcode kkt1 kkt2 xtimes
# CG   3 2 1.081231e-12 119 31 NA 0 TRUE TRUE 0
```

- BFGS

```r
optimx(par, fn, method = "BFGS")
```

```
# p1  p2   value fevals gevals niter convcode kkt1 kkt2 xtimes
# BFGS 3 2 1.354193e-12 32 11 NA 0 TRUE TRUE 0
```
Choosing Optimization Methods

- Many methods available, as problems vary in size and complexity
- Depending on the problem optimization methods have specific advantages
- optimx offers a great way to test and compare search methods

```r
optimx(par, fn, method = c("Nelder-Mead", "CG", "BFGS", "spg", "nlm"))
```

<table>
<thead>
<tr>
<th>Method</th>
<th>p1</th>
<th>p2</th>
<th>value</th>
<th>fevals</th>
<th>gevals</th>
<th>niter</th>
<th>convcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nelder-Mead</td>
<td>2.99995</td>
<td>2.000183</td>
<td>5.561630e-07</td>
<td>67</td>
<td>NA</td>
<td>NA</td>
<td>0</td>
</tr>
<tr>
<td>CG</td>
<td>3.00000</td>
<td>2.000000</td>
<td>1.081231e-12</td>
<td>119</td>
<td>31</td>
<td>NA</td>
<td>0</td>
</tr>
<tr>
<td>BFGS</td>
<td>3.00000</td>
<td>2.000000</td>
<td>1.354193e-12</td>
<td>32</td>
<td>11</td>
<td>NA</td>
<td>0</td>
</tr>
<tr>
<td>spg</td>
<td>3.00000</td>
<td>2.000000</td>
<td>2.239653e-13</td>
<td>15</td>
<td>NA</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>nlm</td>
<td>3.00000</td>
<td>2.000000</td>
<td>1.450383e-14</td>
<td>NA</td>
<td>NA</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>kkt1</th>
<th>kkt2</th>
<th>xtimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nelder-Mead</td>
<td>FALSE</td>
<td>TRUE</td>
<td>0.00</td>
</tr>
<tr>
<td>CG</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.00</td>
</tr>
<tr>
<td>BFGS</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.00</td>
</tr>
<tr>
<td>spg</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.08</td>
</tr>
<tr>
<td>nlm</td>
<td>TRUE</td>
<td>TRUE</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Control Object

- Control optimize allows to specify the optimization process
  
  **trace**  Non-negative integer to show iterative search information
  
  **follow.on**  If TRUE and multiple methods, then later methods start the search where the previous method stopped (effectively a polyalgorithm implementation)
  
  **maximize**  If TRUE, maximize the function (not possible for methods "nlm","nlminb" and "ucminf")

- Example in R

```r
optimx(par, fn, method = c("BFGS", "Nelder-Mead"),
       control = list(trace = 6, follow.on=TRUE, maximize=FALSE))
```
Scaling

- Optimization treats all variables in the same way
- Sometimes, variables have strongly different scale
- Example, particle with speed $10^7 \text{ m/s}$ and mass $10^{-27} \text{ kg}$
- Step size and error will be hugely different for the two variables
- Besides manual scaling, two options in \texttt{optimx}
  - \texttt{fnscale} Overall scaling to the function and gradient values
  - \texttt{parscale} Vector scaling of parameters
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
R Optimization Infrastructure (ROI)

- **ROI** is a package which provides a standardized interface to many R optimization packages
- Setup and installation
  ```r
  install.packages("ROI")
  ```
- The latest (non-stable) versions are on R-Forge, use the `repos` option to install these
  ```r
  install.packages("ROI",
  repos="http://R-Forge.R-project.org")
  ```
- Currently supported solvers and corresponding plugins
  ```r
  require(ROI)
  ROI_available_solvers()
  glpk          quadprog          symphony
  "ROI.plugin.glpk" "ROI.plugin.quadprog" "ROI.plugin.symphony"
  ```
- Implementation of many more solvers planned, overview
  https://r-forge.r-project.org/R/?group_id=308
Installation of ROI plugins

- Solver plug-ins need to be installed separately

```r
install.packages("ROI.plugin.glpk")
install.packages("ROI.plugin.quadprog")
install.packages("ROI.plugin.symphony")
```

- Check solver plug-in installation

```r
library(ROI)

ROI_installed_solvers()
##
## glpk quadprog symphony
## "ROI.plugin.glpk" "ROI.plugin.quadprog" "ROI.plugin.symphony"

ROI_registered_solvers()
##
## nlminb glpk quadprog
## "ROI.plugin.nlminb" "ROI.plugin.glpk" "ROI.plugin.quadprog"
## symphony
## "ROI.plugin.symphony"
```
Usage of ROI

▶ Package definition and function call

```r
require(ROI)
ROI_solve(x, solver, control = NULL, ...)
```

▶ Arguments of `ROI_solve`

- `x`: object with problem and constraint description
- `solver`: solver to be used
- `control`: list of additional control arguments
Solving optimization problems with ROI)

- linear 3-dimensional example

```r
lp <- OP(objective = c(2, 4, 3),
          L_constraint =
          L = matrix(c(3, 2, 1, 4, 1, 3, 2, 2, 2), nrow = 3),
          dir = c("<=", "<=", "<="),
          rhs = c(60, 40, 80)),
          maximum = TRUE)

lp

## ROI Optimization Problem:
## Maximize a linear objective function of length 3 with
## - 3 continuous objective variables,
## subject to
## - 3 constraints of type linear.

sol <- ROI_solve(lp, solver = "glpk")

## Optimal solution found.
## The objective value is: 7.666667e+01
```
Solving optimization problems with ROI

- Quadratic problem with linear constraints

```r
qp <- OP(
  Q_objective(Q = diag(1, 3),
              L = c(0, -5, 0)),
  L_constraint(
    L = matrix(c(-4, -3, 0, 2, 1, 0, 0, -2, 1),
                ncol = 3,
                byrow = TRUE),
    dir = rep(">=", 3),
    rhs = c(-8, 2, 0))
)
```

### ROI Optimization Problem:

```
## Minimize a quadratic objective function of length 3 with
## - 3 continuous objective variables,
## subject to
## - 3 constraints of type linear.
```

```r
sol <- ROI_solve(qp, solver = "quadprog")
```

### Optimal solution found.
```
## The objective value is: -2.380952e+00
```
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
Optimization inside the LASSO

- **Lasso** (least absolute shrinkage and selection operator) is a popular method for predictions.
- The underlying regression is solved by minimizing an error term, e.g., **RSS** (residual sum of squares) and a tuning parameter.
- In case of the Lasso

\[
\begin{align*}
\min_{\beta} & \ (y - \beta X)^2 \quad \text{subject to } \sum |\beta| \leq s \\
\end{align*}
\]  

(3)

- Regression part written out

\[
\begin{align*}
\min_{\beta} & \ y^T y - 2y^T X \beta + \beta^T X^T X \beta \\
\end{align*}
\]  

(4)

- Variables for quadratic optimization \( Dmat = X^T X \) and \( dvec = y^T X \)
Comparison of regression and minimization

```r
# Sample data
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
y <- 1 + x1 + x2 + rnorm(n)
X <- cbind(rep(1, n), x1, x2)

# Regression
r <- lm(y ~ x1 + x2)

# Optimization
library(quadprog)
s <- solve.QP(t(X) %% X, t(y) %% X, matrix(nr=3, nc=0), numeric(), 0)

# Comparison
coef(r)

## (Intercept) x1 x2
## 1.0645272 1.0802060 0.9807713

s$solution  # Identical

## [1] 1.0645272 1.0802060 0.9807713
```

Optimization in R: Applications
Optimization inside Quantile Regressions

- Basic problem, find the median such that

\[
\min_{\mu} \sum_{i=0}^{N} |x_i - \mu|
\]  

(5)

- This can be written as a linear problem

\[
\min_{\mu, a_i, b_i} \sum_{i=0}^{N} a_i + b_i
\]

(6)

subject to \(a_i \geq 0\),

(7)

\(b_i \geq 0\) and

(8)

\(x_i - \mu = a_i - b_i\)

(9)
Optimization inside Quantile Regressions

- Finding the median with a linear optimization

```r
n <- 101  # Odd number for unique median
x <- rlnorm(n)
library(lpSolve)

# One constraint per row: a[i], b[i] >= 0
A1 <- cbind(diag(2*n), 0)

# a[i] - b[i] = x[i] - mu
A2 <- cbind(diag(n), -diag(n), 1)

r <- lp("min",
        c(rep(1,2*n), 0),
        rbind(A1, A2),
        c(rep(">=", 2*n), rep("=", n)),
        c(rep(0,2*n), x)
)

# Comparison
tail(r$solution, 1)
## [1] 0.9890153

median(x)
## [1] 0.9890153
```
Introducing $\tau = .3$ allows to calculate a quantile regression

```r
require(lpSolve)

tau <- .3
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
y <- 1 + x1 + x2 + rnorm(n)
X <- cbind(rep(1,n), x1, x2)
A1 <- cbind(diag(2*n), 0, 0, 0)  # a[i], b[i] >= 0
A2 <- cbind(diag(n), -diag(n), X)  # a[i] - b[i] = (y - X %*% beta)[i]
r <- lp("min",
    c(rep(tau,n), rep(1-tau,n), 0, 0, 0),
    rbind(A1, A2),
    c(rep("\geq", 2*n), rep("=", n)),
    c(rep(0,2*n), y)
)

tail(r$solution, 3)
## [1] 0.5827969 1.2125340 0.8054628

# Compare with quantreg
rq(y~x1+x2, tau=tau)
```

# Call:
# rq(formula = y ~ x1 + x2, tau = tau)

# Coefficients:
# (Intercept)     x1     x2
# 0.5827969 1.2125340 0.8054628

# Degrees of freedom: 100 total; 97 residual
Outline

1. Introduction to Optimization in R
2. Linear Optimization
3. Quadratic Programming
4. Non-Linear Optimization
5. R Optimization Infrastructure (ROI)
6. Applications in Statistics
7. Wrap-Up
Outlook

Additional Material

- Short summary of Optimization with R → Seminar Paper
- Further exercises as homework
- R Reference Card, will also be available during exam

Future Exercises

R will be used to solve sample problems from Business Intelligence