ROI:An Extensible R Optimization Infrastructure

Buse BALTACIOĞLU ~ 2019900540

25 01 2021

Contents

1 ROI Nedir?

R optimizasyon altyapısı paketi, doğrusal, ikinci dereceden, konik ve genel doğrusal olmayan optimizasyon problemlerini tutarlı bir şekilde modellemek için genişletilebilir bir altyapı sağlar.

Ayrıca altyapı, optimizasyon problemlerini çeşitli formatlarda okumak ve yazmak için birçok farklı çözücüyü, reformülasyonu, problem koleksiyonunu ve işlevi yönetir.

Optimizasyon; istatistik, makine öğrenimmi ve veri biliminde rutin olarak kullanılan birçok yöntemde önemli bir rol oynar.

Genellikle, bu yöntemlerin uygulamaları, yalnızca belirli bir uygulamada uygulanabilecek şekilde tasarlanmıştır, oldukça özelleştirilmiş optimizasyon algoritmalarına dayanır.

2 R optimizasyon altyapısının avantajları

- Optimizasyon problemi tek bir nesnede saklanır.
- Sorunun çözümü sırasında tutarlılık açısından kontrol edilebilir.
- Optimizasyon probleminin farklı elementlerine, problem oluşturulduktan sonra kolayca erişilebilir.
- Tüm optimizasyon problemini yeniden tanımlamaya gerek kalmadan kolayca düzenlenebilir.
- Son gelişmeleri R çözücü ortamına eklemek kolaydır.
- ROI ve 23 tanımlayıcı paketi ile genişletilebilir olarak tasarlanmıştır.

3 ROI'de kullanılan optimizasyon modeli sınıfları

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

4 ROI'nin Kullanım Alanları

Kullanım Alanları	Konular
Kombinatoryal Problemler	Sudoku
Doğrusal Regresyon	En küçük kareler
Doğrusal Regresyon	En az mutlak sapma problemi
Genelleştirilmiş Doğrusal Modeller	Lojistik regresyon

Kullanım Alanları	Konular
Genelleştirilmiş Doğrusal Modeller	Poisson regresyon
Model Seçimi	Ridge regresyonu
Model Seçimi	Kement
Portföy Optimizasyonu	Portföy optimizasyonu
Ağ Modelleri	MILP ile finansal risk ağı optimizasyonu

5 ROI'ye Giriş

İlgili bir optimizasyon problemini tanımlayan bir model kurduktan sonra problemi bir algoritmanın anlayabileceği şekilde kodlamaktır. En temel biçim bir vektörler kümesi ve bir kısıtlama matrisidir.

5.1 Amaç Fonksiyonu

- Ulaşılmak istenen amacın matematiksel ifadesidir.
- Karar değişkenlerinin bir fonksiyonudur.
- Optimizasyon kriterini verir.

OP(**objective**, constraints, types, bounds, maximum = FALSE)

- *Genel doğrusal olmayan amaç fonksiyonu*

F_objective(F, n, G = NULL, H = NULL, names = NULL)

- *Doğrusal amaç fonksiyonu*

L_objective(L, names = NULL)

- *İkinci dereceden amaç fonksiyonu*

Q_objective(Q, L = NULL, names = NULL)

5.2 Kısıtlar

- İşletme kısıtlarını ifade eden matematiksel fonksiyonlardır.
- Eşitlik veya eşitsizlik biçiminde ifade edilir.

OP(objective, **constraints**, types, bounds, maximum = FALSE)

- *Genel doğrusal olmayan kısıtlar*

F_constraint(F, dir, rhs, J = NULL, names = NULL)

- *Konik kısıtlar*

C_constraint(L, cones, rhs, names = NULL)

- *İkinci dereceden kısıtlar*

Q_constraint(Q, L, dir, rhs, names = NULL)

- *Doğrusal kısıtlar*

L_constraint(L, dir, rhs, names = NULL)

5.3 Karar Değişkenleri

- Değerini bizim belirlemek istediğimiz değişkenlerdir.
- Verilecek kararı ifade eder.

OP(objective, constraints, **types**, bounds, maximum = FALSE)

Karar değişken tipleri;

- C : Sürekli
- I : Tamsayılı
- B : İkili

5.4 Hedef Değişken Sınırları

OP(objective, constraints, types, **bounds**, maximum = FALSE)

V_bound(li, ui, lb, ub, nobj, ld = 0, ud = Inf, names = NULL)

Doğrusal programlama ile ilgili örnek

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

```
con<-rbind(c(5,7,2),
           c(3,2,-9),
           c(1,3,1))

con_dir<-c("<=","<=","<=")

rhs<-c(61,35,31)
```

```
lp<-OP(objective = L_objective(c(3,7,-12)),
       constraints = L_constraint(con,
                                  dir = con_dir,
                                  rhs = rhs),
       bounds = V_bound(li = 3,
                        ui = 3,
                        lb = -10,
                        ub = 10,
                        nobj = 3),
       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.
## - 1 lower and 1 upper non-standard variable bound.
```

Alternatif kodlama

```
lp<-OP()

objective(lp)<-L_objective(c(3,7,-12))

constraints(lp)<-L_constraint(con,
                              dir = con_dir,
                              rhs = rhs)

bounds(lp)<-V_bound(li = 3,
                    ui = 3,
                    lb = -10,
                    ub = 10,
                    nobj = 3)

maximum(lp)<-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.
## - 1 lower and 1 upper non-standard variable bound.
```

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

objective(lp)(param)
```

```
## [1] -2
```

Ayarlayıcı işlevleri, önceden oluşturulmuş OP'leri değiştirmeyi kolaylaştırır.

```
terms(objective(lp))
```

```
## $L
## A 1x3 simple triplet matrix.
##
## $names
## NULL
```

Amaç fonksiyonunun verilerine erişmek için genel işlev terms() kullanılmalıdır.

OP() işlevi her zaman tüm OP'yi depolayan 'OP' sınıfına ait bir S3 nesnesi döndürür. Saklama tek bir R nesnesindeki OP, diğerleri arasında birçok avantaja sahiptir:

- Sorunun çözümü sırasında tutarlılık açısından kontrol edilebilir.
- OP'nin farklı elementlerine, problem oluşturulduktan sonra kolayca erişilebilir.
- Tüm OP'yi yeniden tanımlamaya gerek kalmadan kolayca düzenlenebilir, örneğin bir kısıt eklenebilir, sınırlar değiştirilebilir.
- Tutarlılık kontrolleri, karar değişkenlerin boyutlarının birbirine uyduğunu doğrular.

```
(ROI_solve(lp))
```

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

6 ROI Çözücüleri

ROI_solve(x, solver, control = list(), ...)

ROI, birden çok optimizasyon çözücüsü için tek bir arayüz oluşturan bir R paketidir. Optimizasyon problemleri için farklı çözücüleri kolayca değiştirmek isteyen R kullanıcılarına büyük kolaylık sağlar.

```
ROI_installed_solvers()
```

```
##                nlminb               alabama                   clp
##   "ROI.plugin.nlminb"  "ROI.plugin.alabama"     "ROI.plugin.clp"
##               deoptim                  ecos                  glpk
##  "ROI.plugin.deoptim"     "ROI.plugin.ecos"    "ROI.plugin.glpk"
##                  ipop               lpsolve                 mosek
##     "ROI.plugin.ipop"  "ROI.plugin.lpsolve"   "ROI.plugin.mosek"
##               msbinlp                  neos               nloptr
```

```
##  "ROI.plugin.msbinlp"     "ROI.plugin.neos"   "ROI.plugin.nloptr"
##                optimx                  osqp                qpoases
##   "ROI.plugin.optimx"     "ROI.plugin.osqp"  "ROI.plugin.qpoases"
##              quadprog                   scs              symphony
## "ROI.plugin.quadprog"      "ROI.plugin.scs" "ROI.plugin.symphony"
```

```
ROI_applicable_solvers(lp)
```

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

```
(lp_sol<-ROI_solve(lp, solver = "alabama", start=double(3)))
```

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

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

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

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

```
## $primal
## [1] 61.0000 35.0000 25.8806
##
## $dual
## [1] 0.5820896 1.4626866 0.0000000
```

```
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
```

7 Amaç Fonksiyonu ve Kısıtların Durumuna Göre Çözücüler

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**$^{\star+}$, **lpSolve**$^{\star+}$, **rcdd***, **Rsymphony**$^{\star+}$	**coneproj***, **Dykstra***, **kernlab**, **LowRankQP***, **osqp***, **quadprog***, **ROI.plugin.qpoases**		
Quadratic				
Conic			**cccp***, **CLSOCP***, **ECOSolveR**$^{\star+}$, **Rcsdp***, **Rdsdp***, **scs***	
Functional				**alabama**, **deoptimr**, **clue**, **NlcOptim**, **nloptr**, **Rsolnp**

İkili programlama ile ilgili örnek

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

```
blp<-OP(objective = L_objective(c(-1,-1,-1,-1,-99)),
        constraints = L_constraint(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))
```

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

```
blp_sol
```

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

```
solution(blp_sol)
```

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

```
lp_inaccurate_sol<-ROI_solve(lp,
                             solver = "glpk",
                             tol = 1e-32)
```

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

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

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

```
## [1] -4.298507  0.000000  0.000000
```

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

```
## $primal
## [1] 61.0000 35.0000 25.8806
##
## $dual
## [1] 0.5820896 1.4626866 0.0000000
```

```
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
```

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

```
## [1] 86.70149
```

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

```
## $code
## [1] 0
##
## $msg
##   solver glpk
##     code 5
##   symbol GLP_OPT
##  message Solution is optimal.
## roi_code 0
```

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

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

8 Optimizasyon Problemlerini Okuma ve Yazma

Optimizasyon sorunları genellikle özel düz metin dosyalarında depolanır ve paylaşılır. LP ve MIP dosya formatları için “mps”,“lp”,“sif” ve “freemps” yaygın olarak kullanılmaktadır.

- Örnek 1

```
lp_file<-tempfile()

ROI_write(lp, lp_file, "lp_lpsolve")

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;
```

```
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.
```

- Örnek 2

```
tmpfile <- tempfile()
con <- gzcon(url("http://www.zib.de/koch/perplex/data/netlib/mps/capri.mps.gz"))
writeLines(readLines(con), tmpfile)
close(con)
(capri <- ROI_read(tmpfile, type="mps_fixed", "lpsolve"))
```

```
## ROI Optimization Problem:
##
## Minimize a linear objective function of length 353 with
## - 353 continuous objective variables,
##
## subject to
## - 271 constraints of type linear.
## - 30 lower and 147 upper non-standard variable bounds.
```

```
(sol <- ROI_solve(capri))
```

```
## Optimal solution found.
## The objective value is: 2.690013e+03
```

9 Test Koleksiyonları

Yazılım deposu NETLIB, diğer birçok yazılımın yanı sıra bir doğrusal programlama test koleksiyonunu da içerir.

```
library(ROI.models.netlib)
netlib()
```

```
##  [1] "adlittle" "afiro"    "agg"      "agg2"     "agg3"     "bandm"
##  [7] "beaconfd" "blend"    "bnl1"     "bnl2"     "boeing1"  "boeing2"
## [13] "bore3d"   "brandy"   "capri"    "cycle"    "czprob"   "d2q06c"
## [19] "d6cube"   "degen2"   "degen3"   "dfl001"   "e226"     "etamacro"
## [25] "fffff800" "finnis"   "fit1d"    "fit1p"    "fit2d"    "fit2p"
## [31] "forplan"  "ganges"   "gfrd.pnc" "greenbea" "greenbeb" "grow15"
## [37] "grow22"   "grow7"    "israel"   "kb2"      "lotfi"    "maros.r7"
## [43] "maros"    "modszk1"  "nesm"     "perold"   "pilot.ja" "pilot"
## [49] "pilot.we" "pilot4"   "pilot87"  "pilotnov" "recipe"   "sc105"
## [55] "sc205"    "sc50a"    "sc50b"    "scagr25"  "scagr7"   "scfxm1"
## [61] "scfxm2"   "scfxm3"   "scorpion" "scrs8"    "scsd1"    "scsd6"
## [67] "scsd8"    "sctap1"   "sctap2"   "sctap3"   "seba"     "share1b"
## [73] "share2b"  "shell"    "ship04l"  "ship04s"  "ship08l"  "ship08s"
## [79] "ship12l"  "ship12s"  "sierra"   "stair"    "standata" "standmps"
## [85] "stocfor1" "stocfor2" "stocfor3" "truss"    "tuff"     "vtp.base"
## [91] "wood1p"   "woodw"    "x25fv47"  "x80bau3b"
```

```
netlib("boeing1")
```

```
## ROI Optimization Problem:
##
## Minimize a linear objective function of length 384 with
## - 384 continuous objective variables,
##
## subject to
## - 440 constraints of type linear.
## - 6 lower and 156 upper non-standard variable bounds.
```

```
problem_name<-"boeing1"

model<-netlib(problem_name)

ROI_solve(model)
```

```
## Optimal solution found.
## The objective value is: -3.352136e+02
```

İkinci dereceden programlama ile ilgili örnek

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

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

```
qp_sol<-ROI_solve(qp, "qpoases")

qp_sol
```

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

```
solution(qp_sol)
```

```
## [1] 1.4615385 0.6923077
```

Ek kısıt eklememiz gerekirse

$7x_1 + 11x_2 - x_1^2 - x_2^2 \geq 40$

```
qcqp<-qp

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

```
ROI_applicable_solvers(qcqp)
```

```
## [1] "alabama"       "neos"          "nloptr.cobyla" "nloptr.mma"
## [5] "nloptr.auglag" "nloptr.isres"  "nloptr.slsqp"
```

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

```
solution(qcqp_sol, type = "msg")
```

```
## $par
## [1] 2.845720 4.060584
##
## $value
## [1] 9.447513
##
## $counts
## function gradient
##      194       43
##
## $convergence
## [1] 0
##
## $message
## NULL
##
```

```
## $outer.iterations
## [1] 7
##
## $lambda
## [1] 0.000000 1.410964 0.000000 0.000000
##
## $sigma
## [1] 1e+06
##
## $gradient
## [1] 0.006139029 0.013503808
##
## $ineq
## [1] 2.574639e+01 5.157837e-09 2.845720e+00 4.060584e+00
##
## $equal
## [1] NA
##
## $kkt1
## [1] TRUE
##
## $kkt2
## [1] FALSE
##
## $hessian
##           [,1]    [,2]
## [1,]  887629.1 1905601
## [2,] 1952716.7 4192416
```

Doğrusal olmayan programlama ile ilgili örnek

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

```
nlp1<-OP(maximum = TRUE,
         bounds = V_bound(ud = 42, nobj = 3L))

objective(nlp1)<-F_objective(F = function(x) prod(x),
                             n =3,
                             G = function(x) c(prod(x[-1]),
                                               prod(x[-2]),
                                               prod(x[-3])))

constraints(nlp1)<-F_constraint(F = function(x) x[1]+2*x[2]+2*x[3],
```

```
                                dir = "<=",
                                rhs = 72,
                                J= function(x)c(1,2,2))

nlp1
```

```
## 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.
```

```
nlp1_sol1<-ROI_solve(nlp1, "alabama",
                     start = c(10,10,10))

nlp1_sol1
```

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

```
solution(nlp1_sol1)
```

```
## [1] 24.00002 11.99999 11.99999
```

```
nlp1_sol2<-ROI_solve(nlp1, "alabama",
                     start = double(3))
nlp1_sol2
```

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

```
solution(nlp1_sol2, type = "msg")
```

```
## $par
## [1] 24.00002 11.99999 11.99999
##
## $value
## [1] -3456
##
## $counts
## function gradient
##      152       27
##
## $convergence
## [1] 0
##
## $message
```

```
## NULL
##
## $outer.iterations
## [1] 7
##
## $lambda
## [1] 144.0002   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
##
## $sigma
## [1] 1e+06
##
## $gradient
## [1] -0.01702376 -0.03442507 -0.03442507
##
## $ineq
## [1] 1.734131e-08 2.400002e+01 1.199999e+01 1.199999e+01 1.799998e+01
## [6] 3.000001e+01 3.000001e+01
##
## $equal
## [1] NA
##
## $kkt1
## [1] TRUE
##
## $kkt2
## [1] TRUE
##
## $hessian
##           [,1]    [,2]    [,3]
## [1,]  791594.6 1583178 1583178
## [2,] 1583177.3 3166380 3166356
## [3,] 1583177.3 3166356 3166380
```

Karışık tamsayılı programlama ile ilgili örnek

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

```
A<-rbind(c(5,3),c(7,1))

milp<-OP(c(5,7),
         constraints = L_constraint(A, c(">=",">="),
                                    c(7,9)),
         types = c("B", "I"))
```

```
milp_sol<-ROI_solve(milp, solver = "glpk")

milp_sol
```

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

```
solution(milp_sol)
```

```
## [1] 1 2
```

10 Uygulamalar

10.1 Doğrusal Regresyon Problemi

```
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)
 }
```

```
data("stackloss", package = "datasets")

l1p<-create_L1_problem(x = as.matrix(stackloss)[,-4],
                       y = stackloss[,4])


L1_res<-ROI_solve(l1p, solver = "glpk")

roi_sol<-solution(L1_res)[1:ncol(stackloss)]
```

Bu değerler model katsayılarına karşılık gelir.

- lmtest paketiyle regresyonu çözüp model katsayılarını karşılaştıralım

```
attach(stackloss)
lmodel<-lm(stack.loss~Air.Flow+Water.Temp+Acid.Conc., data = stackloss)
lm_sol<-lmodel$coefficients
```

```
difference<-(roi_sol-lm_sol)
```

```
cbind(roi_sol,lm_sol, difference)
```

```
##                  roi_sol      lm_sol  difference
## (Intercept) -39.68985507 -39.9196744  0.22981935
## Air.Flow      0.83188406   0.7156402  0.11624386
## Water.Temp    0.57391304   1.2952861 -0.72137308
## Acid.Conc.   -0.06086957  -0.1521225  0.09125295
```

10.2 Gezgin Satıcı Problemi

Bir şehir listesi ve her bir şehir çifti arasındaki mesafeler göz önüne alındığında, her şehri tam olarak bir kez ziyaret edip başlangıç şehrine geri dönen mümkün olan en kısa rota nedir?

```
library(knitr)
library(dplyr)
library(ggplot2)
```

Şehir sayısı:

```
n<-10
```

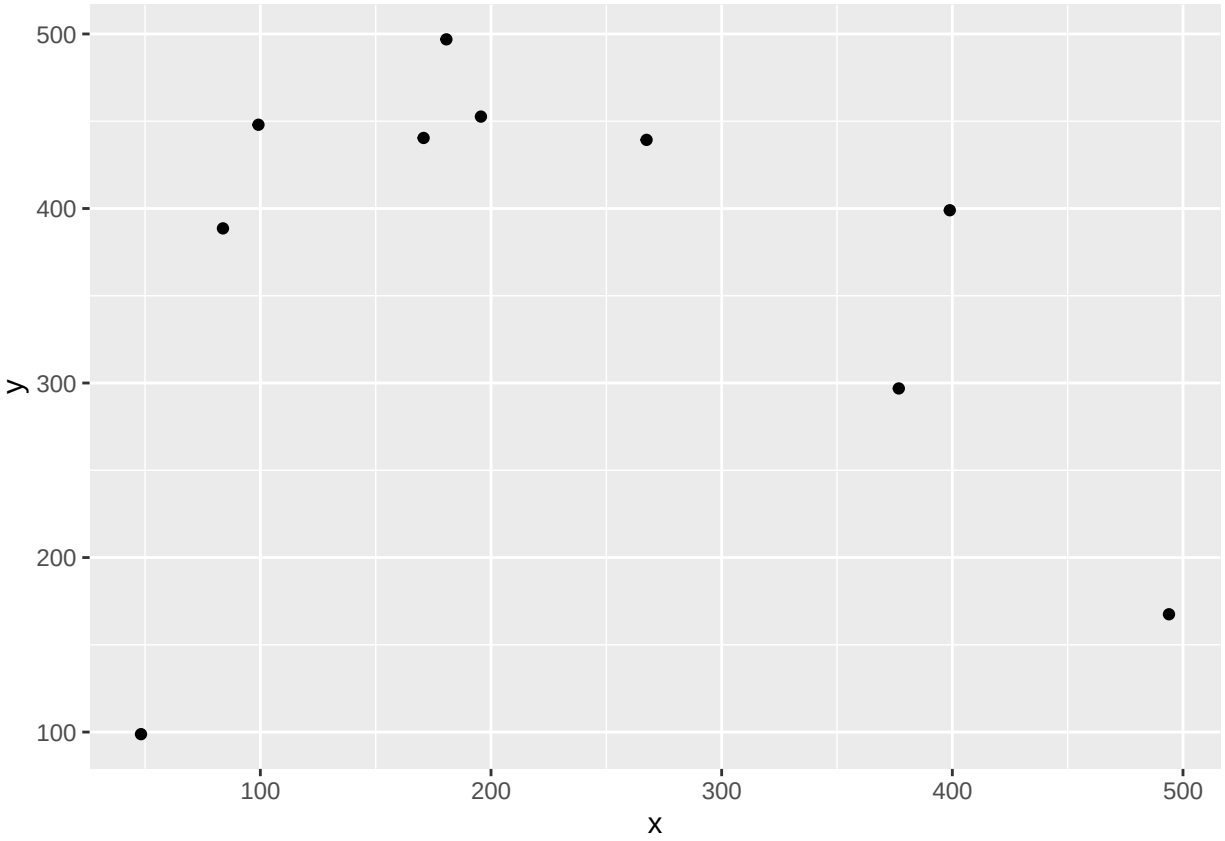
Öklid uzayının sınırı:

```
max_x<-500
max_y<-500
```

Bazı rastgele şehirler:

```
set.seed(123456)
cities <- data.frame(id = 1:n, x = runif(n, max = max_x), y = runif(n, max = max_y))
```

```
ggplot(cities, aes(x,y)) + geom_point()
```



Mesafe matrisi

```
distance <- as.matrix(stats::dist(select(cities, x, y), diag = TRUE, upper = TRUE))
dist_fun <- function(i, j) {
  vapply(seq_along(i), function(k) distance[i[k], j[k]], numeric(1L))
}
```

```
library(ompr)
```

```
model <- MIPModel() %>%
  # we create a variable that is 1 iff we travel from city i to j
  add_variable(x[i, j], i = 1:n, j = 1:n,
               type = "integer", lb = 0, ub = 1) %>%

  # a helper variable for the MTZ formulation of the tsp
  add_variable(u[i], i = 1:n, lb = 1, ub = n) %>%

  # minimize travel distance
  set_objective(sum_expr(dist_fun(i, j) * x[i, j], i = 1:n, j = 1:n), "min") %>%

  # you cannot go to the same city
  set_bounds(x[i, i], ub = 0, i = 1:n) %>%

  # leave each city
  add_constraint(sum_expr(x[i, j], j = 1:n) == 1, i = 1:n) %>%
  #
```

```
  # visit each city
  add_constraint(sum_expr(x[i, j], i = 1:n) == 1, j = 1:n) %>%

  # ensure no subtours (arc constraints)
  add_constraint(u[i] >= 2, i = 2:n) %>%
  add_constraint(u[i] - u[j] + 1 <= (n - 1) * (1 - x[i, j]), i = 2:n, j = 2:n)
```

```
model
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 10
##   Integer: 100
##   Binary: 0
## Model sense: minimize
## Constraints: 110
```

```
library(ompr.roi)
library(ROI.plugin.glpk)
```

```
result <- solve_model(model, with_ROI(solver = "glpk", verbose = TRUE))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.47
## 110 rows, 110 columns, 434 non-zeros
##       0: obj =  0.000000000e+000  infeas = 2.900e+001 (20)
## *    33: obj =  2.101879949e+003  infeas = 0.000e+000 (1)
## *    57: obj =  1.450925576e+003  infeas = 0.000e+000 (1)
## OPTIMAL SOLUTION FOUND
## GLPK Integer Optimizer, v4.47
## 110 rows, 110 columns, 434 non-zeros
## 100 integer variables, 90 of which are binary
## Integer optimization begins...
## +    57: mip =     not found yet >=              -inf        (1; 0)
## +    83: >>>>>  1.469860058e+003 >=  1.450925576e+003   1.3% (3; 0)
## +   118: mip =  1.469860058e+003 >=     tree is empty   0.0% (0; 5)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```
result
```

```
## Status: optimal
## Objective value: 1469.86
```

```
head(result$solution)
```

```
## u[1] u[2] u[3] u[4] u[5] u[6]
##    1    2    9    7    8    6
```

Çözümü çıkarmak için get_solution, tidyverse paketlerle daha fazla kullanabileceğimiz bir data.frame döndüren yöntemi kullanabiliriz.

```r
solution <- get_solution(result, x[i, j]) %>%
  filter(value > 0)
kable(head(solution, 3))
```

variable	i	j	value
x	7	1	1
x	1	2	1
x	5	3	1

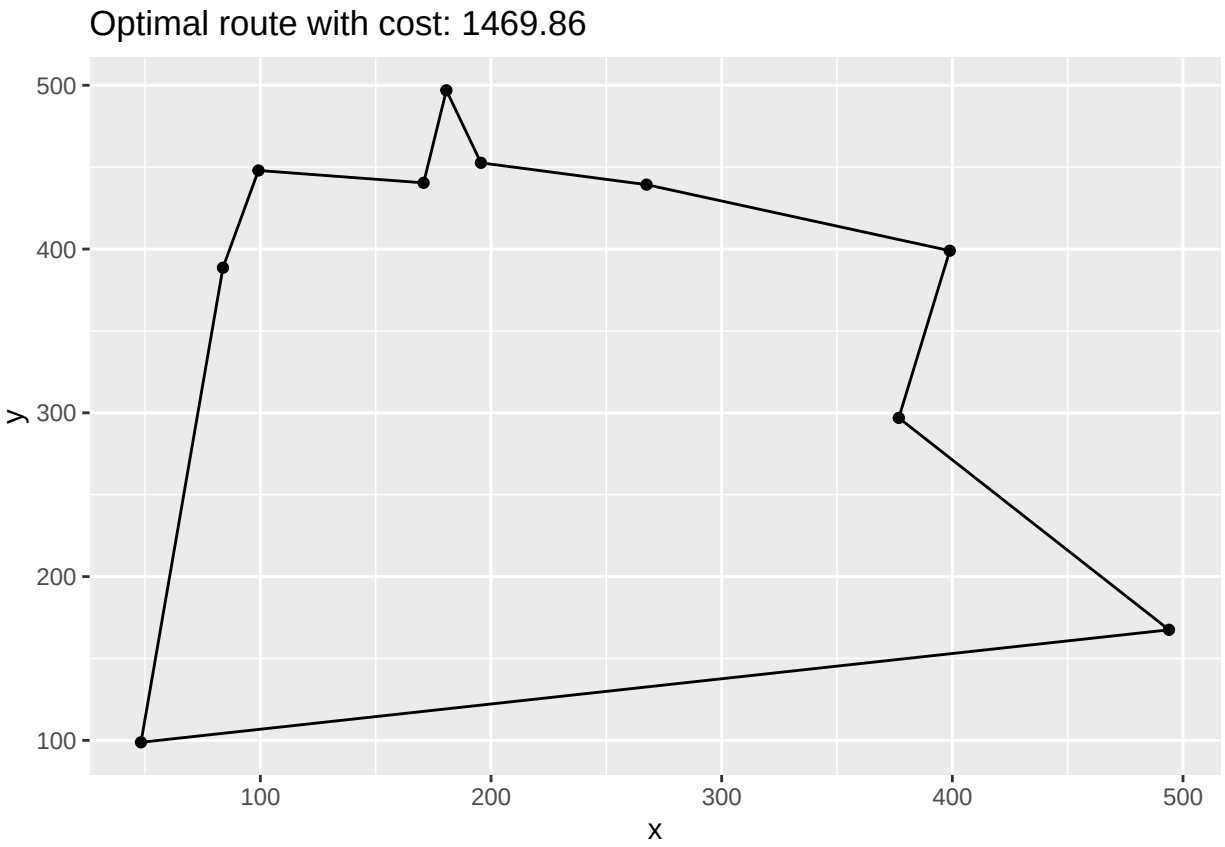
Şimdi modelimizdeki indeksleri gerçek şehirlerle tekrar ilişkilendirmemiz gerekiyor.

```r
paths <- select(solution, i, j) %>%
  rename(from = i, to = j) %>%
  mutate(trip_id = row_number()) %>%
  tidyr::gather(property, idx_val, from:to) %>%
  mutate(idx_val = as.integer(idx_val)) %>%
  inner_join(cities, by = c("idx_val" = "id"))
```

```r
kable(head(arrange(paths, trip_id), 4))
```

trip_id	property	idx_val	x	y
1	from	7	267.4290	439.3217
1	to	1	398.8922	398.9946
2	from	1	398.8922	398.9946
2	to	2	376.7825	296.8970

```r
ggplot(cities, aes(x, y)) +
  geom_point() +
  geom_line(data = paths, aes(group = trip_id)) +
  ggtitle(paste0("Optimal route with cost: ", round(objective_value(result), 2)))
```



Gezgin satıcı problemimiz için ürettiğimiz şehirler arasında minimum yolu bulmayı hedeflemiştik ve optimal değerimizi 1469.86 bulmuş olduk.

10.3 Sudoku yaratma ve çözme

Sudoku'nun bir ikili optimizasyon problemi olarak formüle edilebileceği iyi bilinmektedir.

R paketleri sudoku ve sudokuAlt üretmek ve Sudoku bulmacaları çözmek için kullanılabilir.

```
library(sudokuAlt)
```

```
sudoku_puzzle <- makeGame()
plot(sudoku_puzzle)
```

7				8				
			7					
		5						
1	8			2				6
	5		6			9	8	
							1	
6	7						3	
				7	9			
3		1						

```
library(slam)
library(ROI)
library(ROI.plugin.msbinlp)
library(ROI.plugin.glpk)
```

Yardımcı fonksiyonlar

```
as.matrix.sudoku <- function(x) matrix(as.numeric(x), 9, 9)

to_col_index <- function(i, j, v) {
    (v - 1) * 81 + (i - 1) * 9 + j
}

index_to_triplet <- function(idx) {
    .index_to_triplet <- function(idx) {
        v <- (idx - 1) %/% 81 + 1
        idx <- idx - (v - 1) * 81
        i <- (idx - 1)%/% 9 + 1
        idx <- idx - (i - 1) * 9
        c(i = i, j = idx, v = v)
    }
    t(sapply(idx, .index_to_triplet))
}

solve_sudoku <- function(M, solver, solve = TRUE, ...) {
```

```
    stm <- simple_triplet_matrix
    seq9_9 <- rep(1:9, 9)
    seq9_9e <- rep(1:9, each = 9)
    seq81_9e <- rep(seq_len(81), each = 9)
    ones729 <- rep.int(1, 9^3)
    M <- as.matrix(M)
    M[is.na(M)] <- 0
    M <- as.simple_triplet_matrix(M)
    ## setup OP
    op <- OP(double(9^3))
    ## basic setting (fixed coefficients)
    j <- mapply(to_col_index, M$i, M$j, M$v)
    nfv <- length(M$i) ## number of fixed variables
    A0 <- stm(i = seq_len(nfv), j = j, v = rep.int(1, nfv), ncol = 9^3)
    LC0 <- L_constraint(A0, eq(nfv), rep.int(1, nfv))
    ## sum_{i=1:n} x_ijk = 1,  j = 1:n, k = 1:n
    only_one_k_in_each_column <- function(j, k) {
        sapply(1:9, function(i) to_col_index(i, j, k))
    }
    j <- unlist(mapply(only_one_k_in_each_column, seq9_9e, seq9_9, SIMPLIFY = FALSE))
    A1 <- stm(i = seq81_9e, j = j, v = ones729, nrow = 81, ncol = 9^3)
    ## sum_{j=1:n} x_ijk = 1
    only_one_k_in_each_row <- function(i, k) {
        sapply(1:9, function(j) to_col_index(i, j, k))
    }
    j <- unlist(mapply(only_one_k_in_each_row, seq9_9e, seq9_9, SIMPLIFY = FALSE))
    A2 <- stm(i = seq81_9e, j = j, v = ones729, nrow = 81, ncol = 9^3)
    only_one_k_in_each_submatrix <- function(blocki, blockj, k) {
        i <- (blocki - 1) * 3 + 1:3
        j <- (blockj - 1) * 3 + 1:3
        coo <- expand.grid(i = i, j = j, v = k)
        mapply(to_col_index, i = coo$i, j = coo$j, v = coo$v)
    }
    coo <- expand.grid(i = 1:3, j = 1:3, k = 1:9)
    j <- unlist(mapply(only_one_k_in_each_submatrix,
                       blocki = coo$i, blockj = coo$j, k  = coo$k, SIMPLIFY = FALSE))
    A3 <- stm(i = seq81_9e, j = j, v = ones729, ncol = 9^3)
    ## at every position in the matrix must be one value
    fill_matrix <- function(i, j) {
        sapply(1:9, function(k) to_col_index(i, j, k))
    }
    j <- unlist(mapply(fill_matrix, i = seq9_9e, j = seq9_9, SIMPLIFY = FALSE))
    A4 <- stm(i = seq81_9e, j = j, v = ones729, ncol = 9^3)
    A <- rbind(A1, A2, A3, A4)
    LC1 <- L_constraint(A, eq(nrow(A)), rep.int(1, nrow(A)))
    constraints(op) <- rbind(LC0, LC1)
    types(op) <- rep.int("B", 9^3)
    if (!solve) return(op)
    s <- ROI_solve(op, solver = solver, ...)
    sol <- solution(s)
    to_sudoku_solution <- function(sol) {
        coo <- index_to_triplet(which(as.logical(sol)))
        sudoku_solution <- as.matrix(stm(coo[,1], coo[,2], coo[,3]), nrow = 9, ncol = 9)
```

```
        structure(sudoku_solution, class = c("sudoku", "matrix"))
    }

    if ( any(lengths(sol) > 1L) & length(sol) > 1L ) {
        lapply(solution(s), to_sudoku_solution)
    } else {
        if ( length(sol) == 1L ) {
            to_sudoku_solution(sol[[1L]])
        } else {
            to_sudoku_solution(sol)
        }
    }
}

sudoku_is_valid_solution <- function(x) {
    .sudoku_is_valid_solution <- function(x) {
        stopifnot(inherits(x, "sudoku"))
        seq19 <- seq_len(9)
        for (i in seq19) {
            if ( any(sort(as.vector(x[i, ])) != seq19) ) return(FALSE)
            if ( any(sort(as.vector(x[, i])) != seq19) ) return(FALSE)
        }
        for (i in 1:3) {
            for (j in 1:3) {
                block <- x[(i-1) * 3 + 1:3, (j-1) * 3 + 1:3]
                if ( any(sort(as.vector(block)) != seq19) ) return(FALSE)
            }
        }
        return(TRUE)
    }
    if ( is.list(x) ) {
        sapply(x, .sudoku_is_valid_solution)
    } else {
        .sudoku_is_valid_solution(x)
    }
}
```

Yukarıda tanımladığımız sudokuyu çözelim.

```
sudoku_solution <- solve_sudoku(sudoku_puzzle, solver = "glpk")
sudoku_is_valid_solution(sudoku_solution)
```

```
## [1] TRUE
```

```
sudoku_solution
```

```
##
##  + - - - + - - - + - - - +
##  | 7 3 2 | 5 8 4 | 6 9 1 |
##  | 8 1 6 | 7 9 3 | 4 5 2 |
##  | 9 4 5 | 1 6 2 | 3 7 8 |
##  + - - - + - - - + - - - +
```

```
##  | 1 8 3 | 9 2 5 | 7 4 6 |
##  | 2 5 7 | 6 4 1 | 9 8 3 |
##  | 4 6 9 | 8 3 7 | 2 1 5 |
##  + - - - + - - - + - - - +
##  | 6 7 4 | 2 1 8 | 5 3 9 |
##  | 5 2 8 | 3 7 9 | 1 6 4 |
##  | 3 9 1 | 4 5 6 | 8 2 7 |
##  + - - - + - - - + - - - +
```

Ayrıca Sudoku için birden fazla çözüm olup olmadığını kontrol etmek kolaydır. Burada, tüm çözümleri elde ediyoruz.

```
has_unique_solution <- inherits(sudoku_solution, "sudoku")
```

```
sudoku_solution <- solve_sudoku(sudoku_puzzle, solver = "msbinlp", nsol_max = 10L)
```

11 Kaynaklar

- https://www.jstatsoft.org/article/view/v094i15
- https://rdrr.io/cran/ROI/man/ROI_write.html
- http://roi.r-forge.r-project.org/index.html
- https://www.r-orms.org/mixed-integer-linear-programming/practicals/problem-course-assignment/