Name: - Ravina yadav Roll no:-513

Paper III (Operational Research)

MSC (Computer Science) Semester- III 2022-23.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INDEX** | | | | |
| **No** | **Date** | **Title** | **Page no** | **Sign** |
| 1 | 1/08/22 | Graphical method using R programming. |  |  |
| 2 | 1/08/22 | Simplex method using two variables using python. |  |  |
| 3 | 1/08/22 | Simplex method using three variables using python. |  |  |
| 4 | 1/08/22 | Simplex method with equality constraints using python. |  |  |
| 5 | 1/08/22 | Big M Simplex method using python. |  |  |
| 6 | 3/08/22 | Resource allocation problems using simplex method. |  |  |
| 7 | 3/08/22 | Infeasibility in simplex method. |  |  |
| 8 | 3/08/22 | Dual simplex method. |  |  |
| 9 | 3/08/22 | Transportation problems. |  |  |
| 10 | 3/08/22 | Assignment problems. |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

To solve linear programming using R studio, we need to install lpsolve package Install.packages(“lpsolve”)

## PRACTICAL 1

**GRAPHICAL METHOD USING R PROGRAMMING**

*# R Program*

*#Find a geometrical interpretation and solution as well for the following LP problem #Max z= 3x1 + 5x2*

*#subject to constraints:*

*#x1+2x2<=2000 #x1+x2<=1500 #x2<=600 #x1,x2>=0*

# Load lpSolve require(lpSolve)

## Set the coefficients of the decision variables -> C of objective function C <- c(3,5)

# Create constraint martix B A <- matrix(c(1, 2,

1, 1,

0, 1

), nrow=3, byrow=TRUE)

# Right hand side for the constraints B <- c(2000,1500,600)

# Direction of the constraints constranints\_direction <- c("<=", "<=", "<=")

# Create empty example plot plot.new()

plot.window(xlim=c(0,2000), ylim=c(0,2000)) axis(1)

axis(2)

title(main="LPP using Graphical method") title(xlab="X axis")

title(ylab="Y axis") box()

# Draw one line

segments(x0 = 2000, y0 = 0, x1 = 0, y1 = 1000, col = "green") segments(x0 = 1500, y0 = 0, x1 = 0, y1 = 1500, col = "green") segments(x0 = 0, y0 = 0, x1 = 600, y1 = 0, col = "green")

# Find the optimal solution optimum <- lp(direction="max",

objective.in = C, const.mat = A,

const.dir = constranints\_direction, const.rhs = B,

all.int = T)

# Print status: 0 = success, 2 = no feasible solution print(optimum$status)

# Display the optimum values for x1,x2 best\_sol <- optimum$solution names(best\_sol) <- c("x1", "x2") print(best\_sol)

# Check the value of objective function at optimal point print(paste("Total cost: ", optimum$objval, sep=""))

OUTPUT:

[Workspace loaded from ~/.RData]

* # Right hand side for the constraints
* B <- c(2000,1500,600)
* # R Program
* # Load lpSolve
* require(lpSolve)

Loading required package: lpSolve

* ## Set the coefficients of the decision variables -> C
* C <- c(3,5)
* # Create constraint martix B
* A <- matrix(c(1, 2,

+ 1, 1,

+ 0, 1

+ ), nrow=3, byrow=TRUE)

* # Right hand side for the constraints
* B <- c(2000,1500,600)
* # Direction of the constraints
* constranints\_direction <- c("<=", "<=", "<=")
* # Create empty example plot
* #plot(2000, 2000, col = "white", xlab = "", ylab = "")
* plot.new()
* plot.window(xlim=c(0,2000), ylim=c(0,2000))
* axis(1)
* axis(2)
* title(main="LPP using Graphical method")
* title(xlab="X axis")
* title(ylab="Y axis")
* box()
* # Draw one line
* segments(x0 = 2000, y0 = 0, x1 = 0, y1 = 1000, col = "green")
* segments(x0 = 1500, y0 = 0, x1 = 0, y1 = 1500, col = "green")
* segments(x0 = 0, y0 = 0, x1 = 600, y1 = 0, col = "green")
* # Find the optimal solution
* optimum <- lp(direction="max",

+ objective.in = C,

+ const.mat = A,

+ const.dir = constranints\_direction,

+ const.rhs = B,

+ all.int = T)

* # Print status: 0 = success, 2 = no feasible solution
* print(optimum$status)

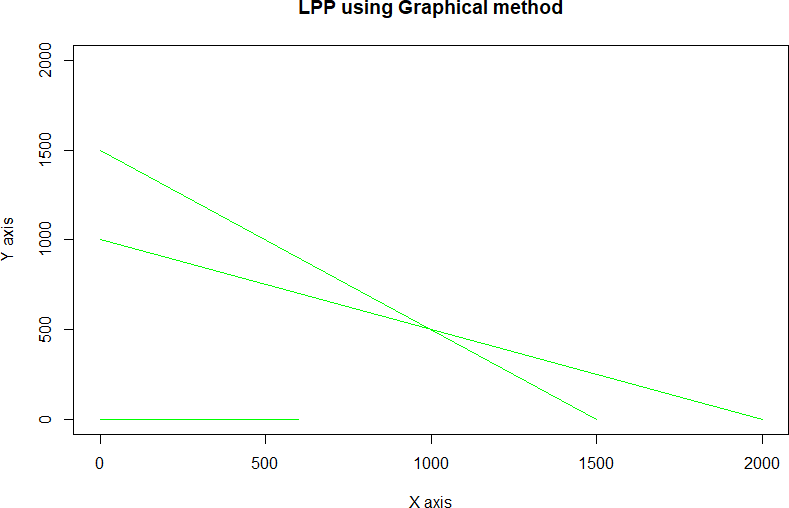
[1] 0

* # Display the optimum values for x1,x2
* best\_sol <- optimum$solution
* names(best\_sol) <- c("x1", "x2")
* print(best\_sol) x1 x2

1000 500

* # Check the value of objective function at optimal point
* print(paste("Total cost: ", optimum$objval, sep=""))

[1] "Total cost: 5500"



**PRACTICAL 2**

# Simplex Method with 2 variables using Python

from scipy.optimize import linprog #Max z=3x1+2x2

#subject to #x1 + x2 <=4 #x1 - x2 <=2 #x1,x2>=0 obj = [-3, -2]

lhs\_ineq = [[ 1, 1], # Red constraint left side

... [1, -1]] # Blue constraint left side

rhs\_ineq = [4, # Red constraint right side

... 2] # Blue constraint right side

bnd = [(0, float("inf")), # Bounds of x

... (0, float("inf"))] # Bounds of y

>>> opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... bounds=bnd,method="revised simplex")

>>> opt

opt.fun

opt.success

opt.x

; Jupyter simplex1with2variables Last Checkpoint: a day ago (unsaved changes) **;a** Logout

File Edit View Insert Cell Kernel Widgets Help Trusted I Python 3 0

**ll'l** + **:lo< *fl)*** ... + + H Run ■ **C** \* Code \_Jv **C'l**

##### Sonve following linear programming problem with two variables using simplex method.

Max z=3x1+2x2 subject to

x1 + *x2* <=4

x1 - x2 <=2

X1,x2>=0

In {1]: **H** fi:omcipy.optimize impoi:t linprog In [13], **H** ocj - [-3, -2]

In [14]: **H** lh.s\_ineq = [ ( 1, l], *I left sids of first constraint*

( 1, -1] J *I lt!ft sid!!!!= oi st!cond constraint*

In [15]: **H** rh.s\_ineq = [4, *: right side constant of first constraint*

2] *: right side constant of first constraint*

In [16]: **H** bnd= 1(0, float("inf")), *IBoundsotx*

(0, flaat("inf"))] *: Bounds of y*

In [17]: H >>> opt - linprog(e=cobj, A\_ub=,.lh.9\_ineq, b\_ub-rh.9\_ineq,

tound.9=bnd,method-"revi.9ed .9implex")

>>> opt

Out[l7]: array ( [], dtype=float64) fun: -11.0

me9.sage: 'Optimization terminated succe9.sfully.' n:i..t: 2

!5lack: array( [O., 0.]) status: 0

True

array([3., 1.])

In [18]: **H** opt.fun OUt[lS]: -11.0



**PRACTICAL 3**

# Simplex Method with 3 variables using Python

from scipy.optimize import linprog #Min z= x1-3x2+2x3

#subject to #3x1-x2+3x3<=7 #-2x1+4x2<=12

#-4x1+3x2+8x3<=10 #x1,x2,x3>=0

obj = [1, -3, 2]

lhs\_ineq = [[ 3, -1, 3], # Red constraint left side

... [-2, 4, 0], # Blue constraint left side

... [ -4, 3, 8]] # Yellow constraint left side

rhs\_ineq = [7, # Red constraint right side

... 12, # Blue constraint right side

... 10] # Yellow constraint right side

bnd = [(0, float("inf")), # Bounds of x

... (0, float("inf")),

... (0, float("inf"))] # Bounds of y

>>> opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... bounds=bnd,

... method="revised simplex")

>>> opt

;: **JUpyter SimplexWith3Variables** Last Checkpoint: 2 hours ago (autosa•,ed) LO!iJO

File Edit View Insert Cell Kernel Widgets

+ :I< I',') **l',** + + Run ■ **C** Code

Help

I:::. **Cl**

Trusted I Python 3

In 127]: **M** from !5cipy.optimiz import linprog

In {28]: **H** *#Min* 2= *x1-3x2+2x3*

*tsubp::ct to #3xl-x.2+3x3<-7 f-2x1+4x2<=12*

*f-4xl+Jx2+8x3<=10 #xl,x.2, x3.>=0*

obJ = 11, -3, 2]

In 129]: **M** lh.9\_ineq = I [ 3, -1, 3], *: Red constraint lsft side*

(-2, 4, 0], *I Bl11e constraint left side*

[ -4, 3, 8]] *i Yellov constraint le.fr. side*

In JJO]: **M** rh!5\_J.n q = 17, *i Red constraint* right *side*

12, *I Blue constraint* right *side*

10] f *YslloJo'constraint right side*

In {33]: **H** bnd = I {0, float {"inf")), *i Bo11nds of x*

(0, float("inf")),

(0, float ("1.nf"))] *# Bounds of* y

In {34]: **M** >>>opt= linprog(c=obJ, A\_ub=ln.5\_J.n q, b\_ub=rh.5\_J.n q,

bound.9=bnd, method="revised simplex")

>>> opt

0Ut[34]: array([], dtyp-e::float64) fun: -11.0

me5.5age: 'Optimization terminated .5ucce.5.5fully.' nit: 2

slack: array{[ 0., 0., 11.])

.5tatu.5: 0

True

array([4,, 5., 0,])

**PRACTICAL 4**

# Simplex Method with Equality Constraints Using Python

from scipy.optimize import linprog #Max z=x+2y

#subject to #2x+y<=20

#-4x+5y<=10 #-x+2y>=-2

#-x+5y=15 #x,y>=0

obj = [-1, -2]

lhs\_ineq = [[ 2, 1], # Red constraint left side

... [-4, 5], # Blue constraint left side

... [ 1, -2]] # Yellow constraint left side

rhs\_ineq = [20, # Red constraint right side

... 10, # Blue constraint right side

... 2] # Yellow constraint right side

lhs\_eq = [[-1, 5]] # Green constraint left side rhs\_eq = [15] # Green constraint right side

bnd = [(0, float("inf")), # Bounds of x

... (0, float("inf"))] # Bounds of y

opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... A\_eq=lhs\_eq, b\_eq=rhs\_eq, bounds=bnd,

... method="revised simplex")

Opt

#### ## method =”revised simplex” solves linear programming problem using two phase simplex method.

:

con: array([0.])

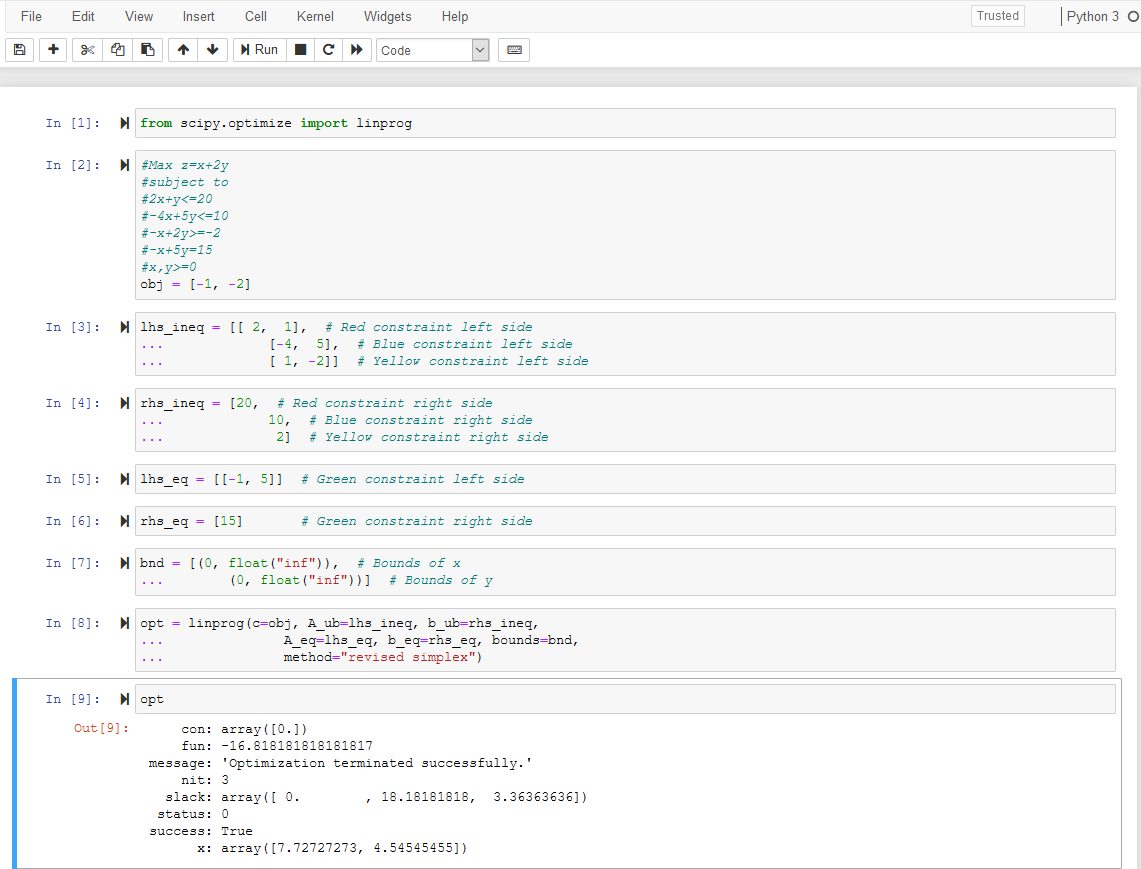
fun: -16.818181818181817

message: 'Optimization terminated successfully.' nit: 3

slack: array([ 0. , 18.18181818, 3.36363636])

status: 0 success: True

x: array([7.72727273, 4.54545455])



## PRACTICAL 5

**BigM Simplex Method using Python**

**Solve Following linear programming problem using Big M Simplex method.**

Min z= 4x1 + x2 subjected to:

3x1 + 4x2 >= 20 x1 + 5x2 >= 15 x1, x2 >= 0

from scipy.optimize import linprog obj = [4, 1]

lhs\_ineq = [[ -3, -4], # left side of first constraint

... [-1, -5]] # right side of first constraint

rhs\_ineq = [-20, # right side of first constraint

... -15] # right side of Second constraint

bnd = [(0, float("inf")), # Bounds of x1

... (0, float("inf"))] # Bounds of x2

>>> opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,

... bounds=bnd,method="interior-point")

>>> opt

#### ## method =” interior-point” solves linear programming problem using default simplex method.

*;::.* **JUpyter BigM Simplex Method** Last Checkpoint: 37 minutes ago (autosaved) **;a** Logout

File Edit View Insert Cell Kernel Widgets

**ll'l** + :l<: It) + + **H** Run ■ C .. Code

Help Trusted I Python 3 0

El m

##### Solve Following linear programming problem using Big M Simplex method.

Min z= 4x1 + x2

subjected to:

3x1 + 4x2 >= 20 X1 +5X2>= 15

x1, x2 >= o

In (41]: **H** from .9cipy.optimize import linprog In (42], **H** obJ (4, l]

In (43]: **H** lhs\_ineq = [[ -3, -4], f *left sids of first constraint*

**(-1,** -5]] # *right sid of first constrc1int*

In (44]: **H** rh!!\_ineq = {-20, *i right: sid of first const:r.aint:*

-15] *right side of Second constraint*

In (45]: **H** bnd = {(0, flaat("inf")), *I Bounds of xl*

(0, float("inf"))] *I Bounds of x2*

In (48]: **M** >>> opt = linprog(e=obJ, A\_ub=l.'1.!!\_:i..neq, b\_ub=rn!!\_:i..neq,

bound.5=bnd, method=" J. nter J.or-point")

>>> opt

Out[48]: array {I], dtype=Eloat64) fun: 5.000000000236447

mes.sage: 'Optimization terminated 3ucce33fully. '

nit: 5

!!lack: array{11. 642774Ble-10, 1. 00000000e+0l])

.5tatu.5: 0

True

array ( I 6. 01160437e- l 1, 5. 00000000e+00] )

In I ] a H

## PRACTICAL 6

### RESOURCE ALLOCATION PROBLEM BY SIMPLEX METHOD

**Use SciPy to solve the resource allocation problem stated as follows:**

Max z= 20x1 + 12x2 +40x3 + 25x4 (profit) subjected to:

x1 + x2 + x3 + x4 <= 50 (manpower)

3x1 + 2x2 + x3 <= 100 -------------(material A)

x2 + 2x3 <= 90 -------------(material B)

x1, x2, x3, x4 >= 0

**from scipy.optimize import linprog**

**obj = [-20, -12, -40, -25] #profit objective function**

**lhs\_ineq = [[1, 1, 1, 1], # Manpower**

**... [3, 2, 1, 0], # Material A**

**... [0, 1, 2, 3]] # Material B**

**rhs\_ineq = [ 50, # Manpower**

**... 100, # Material A**

**... 90] # Material B**

**opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,**

**... method="revised simplex")**

**Opt**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| File | Edit | View Insert Cell Kernel | Help |  | Trusted I Python 3 i |
|  | + :i,; | + **,I,** II Run ■ **C** » Code |  | 13 |  |

##### Use SciPy to solve the resource allocation problem stated as follows:

Max z= 20x1 + 12x2 +40x3 + 25x4 (profit)

SUbjocled to:

xl + x2 + x3 + x4 3x1 + 2x2 + K3

x.2 + 2x3

50 (manpower)

100 ------------- (m.atei::ial A)

90 ------------- (m.atei::ial 8)

x.1. x.2, x3, x.4 >·= 0

In [12]: M rrom :3cipy.optimize import linpi::og

In [13]: M obj = [-20, -12, -40, -25] *#prQfit objective function*

In [14]: M lh:3 i.neq = [ [1, 1, 1, 11, # *I'.,gnpover*

[3, 2, 1, O], #- *i'f.:teric.l* -

[0, 1, 2, 31 l # *Materia.1 B*

In [15]: M :i::h:3\_ineq = 50, # *ivr.,gnpover*

100, #- *i'1ateria.1 A*

90] # *i'1a teria.1 B*

In [161; II opt li.nprog(c=obj, A\_ub=lh:3\_1.neq, b\_ub=:t:h:3 ineq,

method=":revi:3ed :3implex")

In [l'i']: M opt

Ou.t[l?J: ai::i::ay( [], dtype=floatf.4) fun: -1900 .0

me:3:3age: 'Optimization tei:m.inated :3ucce:3:3:fully.' nit: 2

:3lack: ai::i::ay([ 0., 40., O.])

:3tatu::1: 0

:3ucce:3:3: True

an:ay([ 5. 0\_, 45., 0.])

The i::e:3ult tell:3 yo,u that the m.a.xim.al pi::ofit i:3 1900 a.nd coi::re:3pc-nd:3 to x1 = 5 and ID = 45. I-c' :3 not p?:Dfitable to p::::oduce the :3econd and fo,urth p::::□duct:3 unde::: the given condition:3 You can draw :3evei::al inte:i::e:3ting conclu:3ion:3

he!:e:

The thii::d pre duct b::::ing:3 the lai::ge:3t pi::ofi t pe::: unit, :30the factor}; will produce it the mo:3"t.

The fir:3t :3lack i:3 o. whim mea.n:3 that the value:3 of the left and right :3ide:3 of the manpower (:fir:3t) on:3traint the :3ame. The factoi::y pi::oduce:3 50 unit:3pe::: day, a.nd tha-c':3 it:3 full capacity.

The :3econd :3lack i:3 40 becau:3e the factm:y con:3ume:3 60 unit:3 of r:aw mate:i:::ial A (15 unit:3 for: the :fir:3t p:i::oduct plu:3 45 for the third) out of a potential 100 unit::3.

The thii::d :3lack i:3 0, whim mea.n:3 -cha-c the :factoi::y con:3ume:3 all 90 unit:3 o,f the i::aw matei::ial B. Thi:3 entii::e amount i:3 con:3umed :fm: the thii::d pr:oduct. That' :3 why the :facto:i::y can't pr:oduce the :3econd or: fourthpr:oduct at all a.nd ca.n' t p::::oduce mo::::e than 45 unit:3 of the thi:::dproduct. It lack:3 the i::aw material B.

opt .:3tatu:3 i:3 0 and opt i:3 True, indicating that the op-cimization problem wa:3 :3ucce:3:3:fully :301·..red with the

## PRACTICAL 7

### INFEASIBILITY IN SIMPLEX METHOD

**# Solve following linear programming problem using Simplex method**

### WHILE SOLVING LINEAR PROGRAMMING PROBLEM USING SIMPLEX METHOD, IF ONE OR MORE ARTIFICIAL VARIABLES REMAIN IN THE BASIS AT POSITIVE LEVEL AT THE END OF PHASE 1 COMPUTATION , THE PROBLEM HAS NO FEASIBLE SOLUTION( INFEASIBLE SOLUTION).

**Example:**

**Max z= 200x - 300y subject to 2x+3y>=1200 x+y<=400 2x+3/2y>=900**

**x,y>=0**

**from scipy.optimize import linprog obj = [-200, 300]**

**lhs\_ineq = [[ -2, -3], # Red constraint left side**

**... [1, 1], # Blue constraint left side**

**... [ -2, -1.5]] # Yellow constraint left side**

**rhs\_ineq = [-1200, # Red constraint right side**

**... 400, # Blue constraint right side**

**... -900] # Yellow constraint right side**

**bnd = [(0, float("inf")), # Bounds of x**

**... (0, float("inf"))] # Bounds of y**

**opt = linprog(c=obj, A\_ub=lhs\_ineq, b\_ub=rhs\_ineq,**

**... bounds=bnd,**

**... method="revised simplex") opt**

## PRACTICAL 8 DUAL SIMPLEX METHOD

##SOLVE FOLLOWING LINEAR PROGRAMMING PROBLEM USING DUAL SIMPLEX METHOD USING R PROGRAMMING

# Max z=40x1+50x2 #subject to

#2x1 + 3x2 <= 3 #8x1 + 4x2 <= 5 # x1, x2>=0

# Import lpSolve package library(lpSolve)

# Set coefficients of the objective function f.obj <- c(40, 50)

# Set matrix corresponding to coefficients of constraints by rows

# Do not consider the non-negative constraint; it is automatically assumed f.con <- matrix(c(2, 3,

8, 4), nrow = 2, byrow = TRUE)

# Set unequality signs f.dir <- c("<=",

"<=")

# Set right hand side coefficients f.rhs <- c(3,

5)

# Final value (z)

lp("max", f.obj, f.con, f.dir, f.rhs)

# Variables final values

lp("max", f.obj, f.con, f.dir, f.rhs)$solution

# Sensitivities

lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.from lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.to

# Dual Values (first dual of the constraints and then dual of the variables) # Duals of the constraints and variables are mixed

lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals

# Duals lower and upper limits

lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.from lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.to

## OUTPUT:

##SOLVE FOLLOWING LINEAR PROGRAMMING PROBLEM USING DUAL SIMPLEX METHOD USING R PROGRAMM

* # Max z=40x1+50x2
* #subject to
* #2x1 + 3x2 <= 3
* #8x1 + 4x2 <= 5
* # x1, x2>=0
* # Import lpSolve package
* library(lpSolve)
* # Set coefficients of the objective function
* f.obj <- c(40, 50)
* # Set matrix corresponding to coefficients of constraints by rows
* # Do not consider the non-negative constraint; it is automatically assumed
* f.con <- matrix(c(2, 3,

+ 8, 4), nrow = 2, byrow = TRUE)

* # Set unequality signs
* f.dir <- c("<=",

+ "<=")

* # Set right hand side coefficients
* f.rhs <- c(3,

+ 5)

* # Final value (z)
* lp("max", f.obj, f.con, f.dir, f.rhs) Success: the objective function is 51.25
* # Variables final values
* lp("max", f.obj, f.con, f.dir, f.rhs)$solution

[1] 0.1875 0.8750

* # Sensitivities
* lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.from

[1] 33.33333 20.00000

* lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$sens.coef.to

[1] 100 60

* # Dual Values (first dual of the constraints and then dual of the variables)
* # Duals of the constraints and variables are mixed
* lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals

[1] 15.00 1.25 0.00 0.00

* # Duals lower and upper limits
* lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.from

[1] 1.25e+00 4.00e+00 -1.00e+30 -1.00e+30

* lp("max", f.obj, f.con, f.dir, f.rhs, compute.sens=TRUE)$duals.to

[1] 3.75e+00 1.20e+01 1.00e+30 1.00e+30

>

## PRACTICAL 9 TRANSPORTATION PROBLEM

##sOLVE FOLLOWING TRANSPORTATION PROBLEM IN WHICH CELL ENTRIES REPRESENT UNIT COSTS USING R PROGRAMMING.

# "Customer 1", "Customer 2", "Customer 3", "Customer 4" SUPPLY

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| #sUPPLIER 1 | 10 | 2 | 20 | 11 | 15 |
| #sUPPLIER 1 | 12 | 7 | 9 | 20 | 25 |
| #sUPPLIER 1 | 4 | 14 | 16 | 18 | 10 |

#DEMAND 5 15 15 15

# Import lpSolve package library(lpSolve)

# Set transportation costs matrix costs <- matrix(c(10, 2, 20, 11,

12, 7, 9, 20,

4, 14 , 16, 18), nrow = 3, byrow = TRUE)

# Set customers and suppliers' names

colnames(costs) <- c("Customer 1", "Customer 2", "Customer 3", "Customer 4")

rownames(costs) <- c("Supplier 1", "Supplier 2", "Supplier 3")

# Set unequality/equality signs for suppliers row.signs <- rep("<=", 3)

# Set right hand side coefficients for suppliers row.rhs <- c(15, 25, 10)

# Set unequality/equality signs for customers col.signs <- rep(">=", 4)

# Set right hand side coefficients for customers col.rhs <- c(5, 15, 15, 15)

# Final value (z)

TotalCost <- lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)

# Variables final values

lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)$solution print(TotalCost)

### OUTPUT:

* ##sOLVE FOLLOWING TRANSPORTATION PROBLEM IN WHICH CELL ENTRIES REPRESENT UNIT COSTS U
* # "Customer 1", "Customer 2", "Customer 3", "Customer 4" SUPPLY

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| * #sUPPLIER 1 | 10 | 2 | 20 | 11 | 15 |
| * #sUPPLIER 1 | 12 | 7 | 9 | 20 | 25 |
| * #sUPPLIER 1 | 4 | 14 | 16 | 18 | 10 |
| * #DEMAND | 5 | 15 | 15 | 15 |  |
|  |  |  |  |  |  |

* # Import lpSolve package
* library(lpSolve)
* # Set transportation costs matrix
* costs <- matrix(c(10, 2, 20, 11,

+ 12, 7, 9, 20,

+ 4, 14 , 16, 18), nrow = 3, byrow = TRUE)

* # Set customers and suppliers' names
* colnames(costs) <- c("Customer 1", "Customer 2", "Customer 3", "Customer 4")
* rownames(costs) <- c("Supplier 1", "Supplier 2", "Supplier 3")
* # Set unequality/equality signs for suppliers
* row.signs <- rep("<=", 3)
* # Set right hand side coefficients for suppliers
* row.rhs <- c(15, 25, 10)
* # Set unequality/equality signs for customers
* col.signs <- rep(">=", 4)
* # Set right hand side coefficients for customers
* col.rhs <- c(5, 15, 15, 15)
* # Final value (z)
* TotalCost <- lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)
* # Variables final values
* lp.transport(costs, "min", row.signs, row.rhs, col.signs, col.rhs)$solution [,1] [,2] [,3] [,4]

[1,] 0 5 0 10

[2,] 0 10 15 0

[3,] 5 0 0 5

* print(TotalCost)

Success: the objective function is 435

## PRACTICAL 10 ASSIGNMENT PROBLEM

### #SOLVE FOLLOWING ASSIGNMENT PROBLEM REPRESENTED IN FOLLOWING MATRIX USING R PROGRAMMING

**# Assignment Problem**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **JOB1** | **JOB2** | **JOB3** |
| **#W1** | **15** | **10** | **9** |
| **#W2** | **9** | **15** | **10** |
| **#W3** | **10** | **12** | **8** |

**# Import lpSolve package library(lpSolve)**

**# Set assignment costs matrix costs <- matrix(c(15, 10, 9,**

**9, 15, 10,**

**10, 12 ,8), nrow = 3, byrow = TRUE)**

**# Print assignment costs matrix costs**

**# Final value (z) lp.assign(costs)**

**# Variables final values lp.assign(costs)$solution**

### OUTPUT:

* #SOLVE FOLLOWING ASSIGNMENT PROBLEM REPRESENTED IN FOLLOWING MATRIX USING R PROGRAMMI
* # Assignment Problem
* # JOB1 JOB2 JOB3

|  |  |  |  |
| --- | --- | --- | --- |
| * #W1 | 15 | 10 | 9 |
| * #W2 | 9 | 15 | 10 |
| * #W3 | 10 | 12 | 8 |
|  |  |  |  |

* # Import lpSolve package
* library(lpSolve)
* # Set assignment costs matrix
* costs <- matrix(c(15, 10, 9,

+ 9, 15, 10,

+ 10, 12 ,8), nrow = 3, byrow = TRUE)

* # Print assignment costs matrix
* costs

|  |  |  |
| --- | --- | --- |
|  | [,1] | [,2] [,3] |
| [1,] | 15 | 10 9 |
| [2,] | 9 | 15 10 |
| [3,] | 10 | 12 8 |
|  |  |  |

* # Final value (z)
* lp.assign(costs)

Success: the objective function is 27

* # Variables final values
* lp.assign(costs)$solution [,1] [,2] [,3]

[1,] 0 1 0

[2,] 1 0 0

[3,] 0 0 1