

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

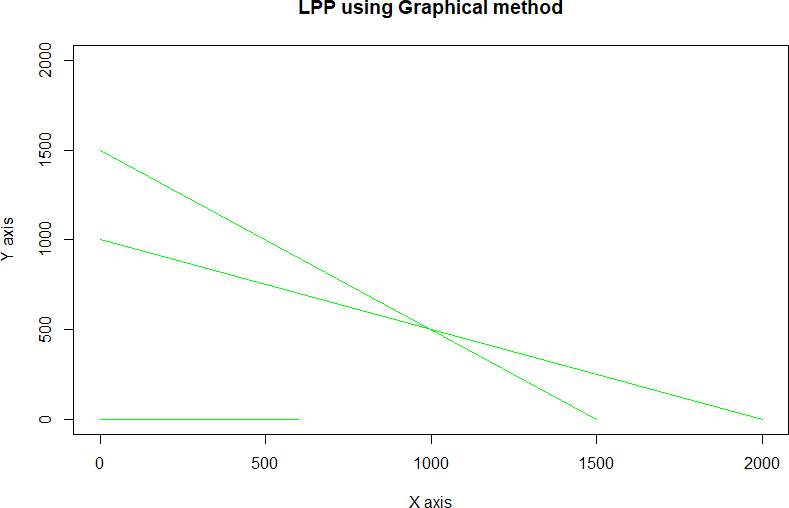
* # Display the optimum values for x1,x2
* best\_sol <- optimum$solution
* names(best\_sol) <- c("x1", "x2")
* print(best\_sol) x1 x2

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,

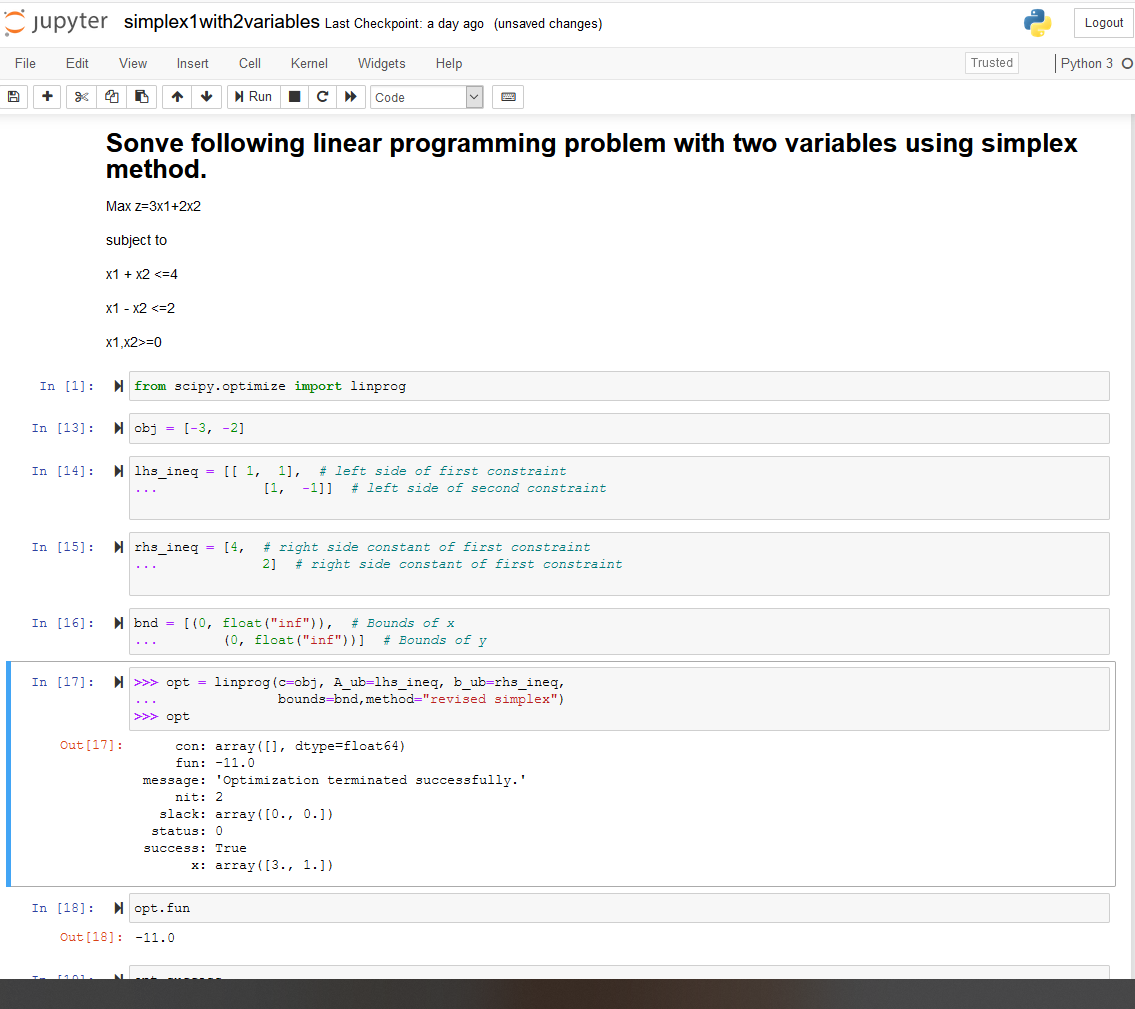
... bounds=bnd,method="revised simplex")

>>> opt

opt.fun

opt.success

opt.x



**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")),

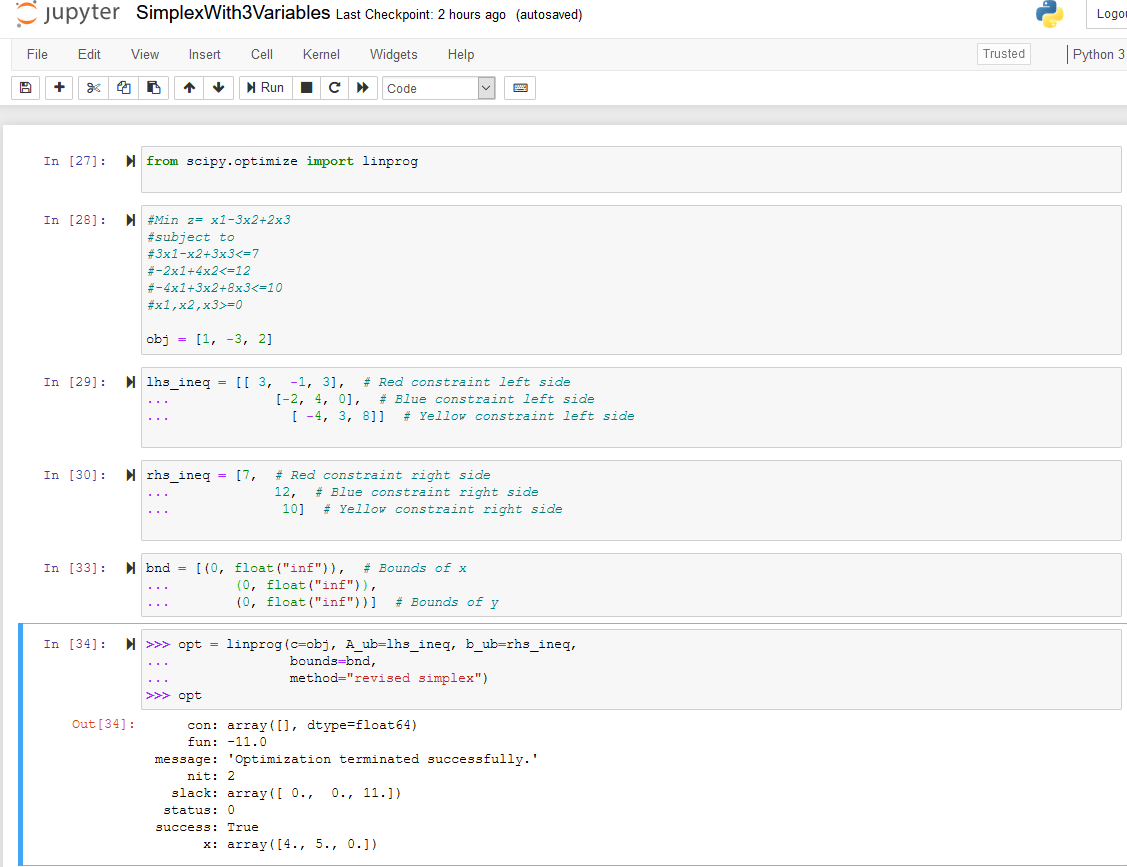
... (0, float("inf"))] # Bounds of y

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

... bounds=bnd,

... method="revised simplex")

>>> opt



**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.])

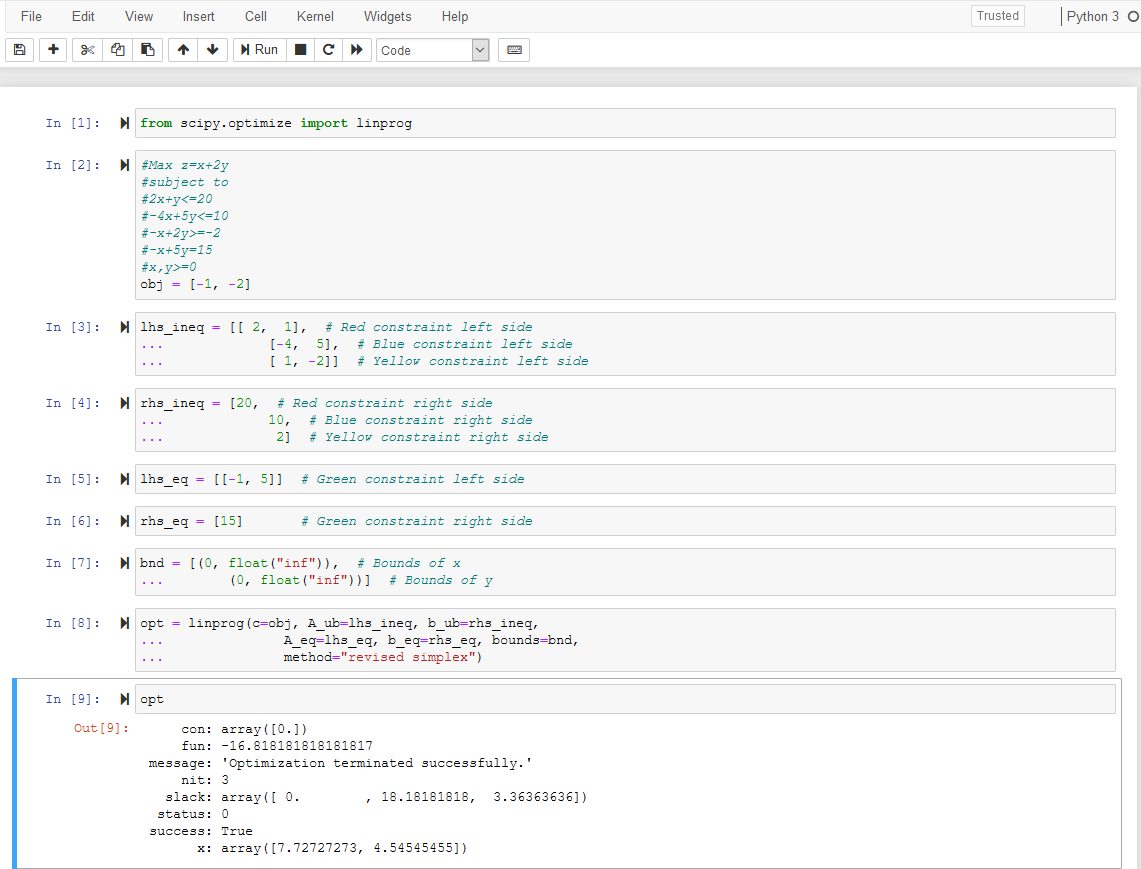
fun: -16.818181818181817

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

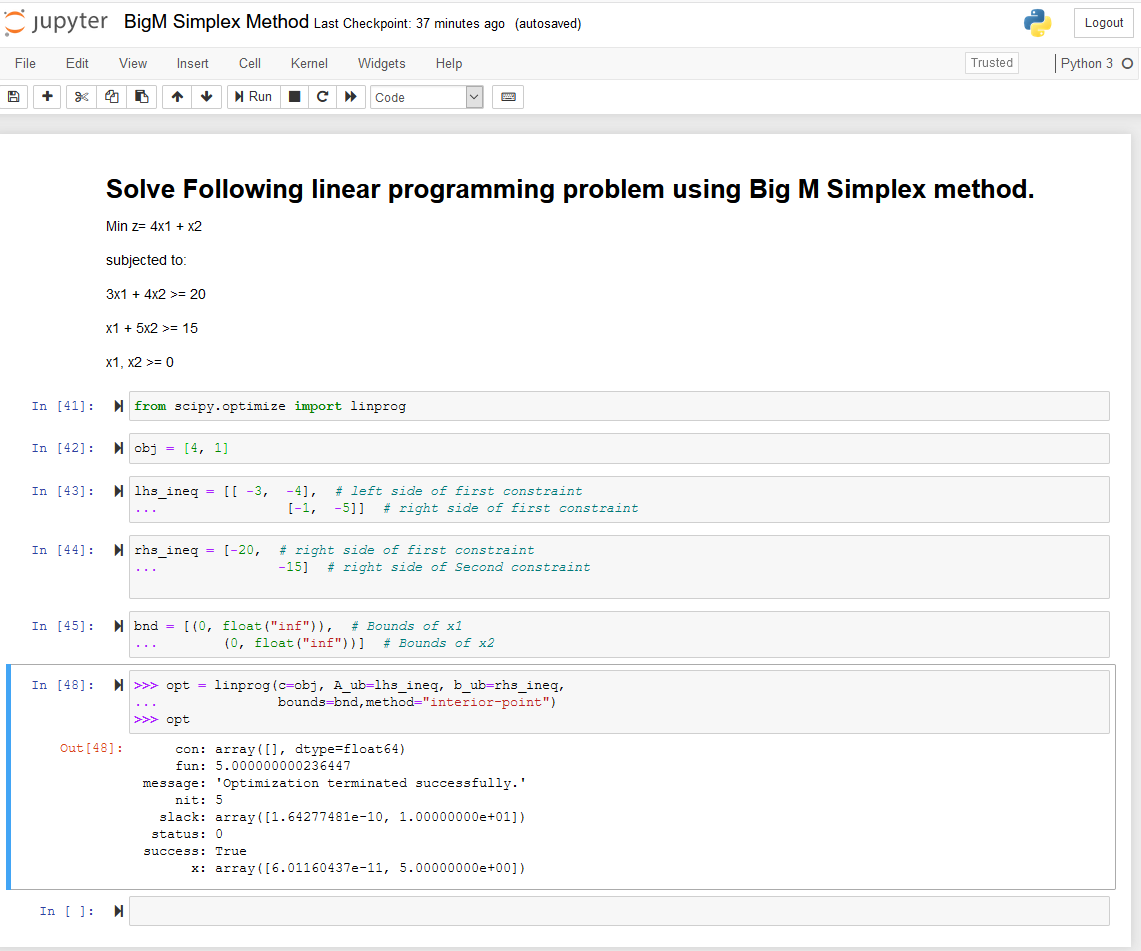
... (0, float("inf"))] # Bounds of x2

>>> 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.**



## 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**



## 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 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) 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 USING R PR

>

* # "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 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   [,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 |
| > |

PRACTICAL 1A

AIM: WRITE A PROGRAM TO CREATE A ROBOT WITH GEAR AND MOVE IT FORWARD, LEFT, RIGHT.

Description:

1] NxtRobot() –

Class that represents a simulated NXT robot brick. Parts (e.g. motors, sensors) may be assembled into the robot to make it doing the desired job.

2] Gear() –

Creates a gear instance with right motor plugged into port A, left motor plugged into port B.

3] addPart(Part) –

Assembles the given part into the robot.

4] setSpeed(int) –

Sets the speed to the given value (arbitrary units).

5] forward() –

Starts the forward movement.

6] left() –

Starts to rotate left (center of rotation at middle of the wheel axes).

7] right() –

Starts to rotate right (center of rotation at middle of the wheel axes).

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_1a {

Prac\_1a(){

NxtRobot robot = new NxtRobot();

Gear g = new Gear();

Robot.addPart(g);

g.setSpeed(100);

g.forward(500);

g.left(250);

g.forward(500);

g.right(250);

g.forward(500);

}

Public static void main (String[] args) {

New Prac\_1a();

}

}

OUTPUT:

PRACTICAL 1B

AIM: WRITE A PROGRAM TO CREATE A ROBOT WITHOUT GEAR AND MOVE IT FORWARD, LEFT, RIGHT.

Description:

TurtleRobot() –

Creates a turtle robot instance.

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_1b {

Prac\_1b(){

TurtleRobot t = new TurtleRobot();

t.forward(100);

t.left(90);

t.forward(100);

t.right(90);

t.forward(100);

}

Public static void main (String[] args) {

New Prac\_1b();

}

}

OUTPUT:

PRACTICAL 2

AIM: WRITE A PROGRAM TO CREATE A ROBOT WITH 2 MOTORS AND MOVE IT FORWARD, LEFT, RIGHT.

DESC:

1] Motor() –

Creates a motor instance that is plugged into given port.

2] Tools.delay() –

Suspends execution of the current thread for the given amount of time.

3] stop() –

Stops the rotation.

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_2 {

Prac\_2(){

NxtRobot r = new NxtRobot();

Motor m1 = new Motor(MotorPort.A);

Motor m2 = new Motor(MotorPort.B);

r.addPart(m1);

r.addPart(m2);

m1.forward();

Tools.delay(1090);

M2.forward();

Tools.delay(1090);

M1.stop();

M2.forward();

Tools.delay(1090);

M1.forward();

M1.stop();

M2.stop();

}

Public static void main(String args[]){

New Prac\_2();

}

}

OUTPUT:

PRACTICAL 3

AIM: WRITE A PROGRAM TO DO A SQUARE USING A WHILE LOOP.

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_3 {

Prac\_3(){

NxtRobot robot = new NxtRobot();

Gear g = new Gear();

Robot.addPart(g);

g.setSpeed(100);

while (true){

g.forward(600);

g.left(280);

}

}

Public static void main (String[] args) {

New Prac\_3();

}

}

OUTPUT:

PRACTICAL 4

AIM: WRITE A PROGRAM TO CREATE A ROBOT WITH LIGHT SENSORS TO FOLLOW A LINE.

Description:

1] RobotContext() –

Creates a RobotContext instance.

2] setStartPosition(int, int) –

Sets the Nxt starting position (x-y-coordinates 0..500, origin at upper left).

3] useBackground(String) –

Use the given image as background (playground size 501 x 501).

4] LegoRobot() –

Creates a robot with its playground using defaults from RobotContext.

5] LightSensor(SensorPort) –

Creates a sensor instance pointing downwards connected to the given port.

6] getValue() –

For sensor ports 1, 2, 3, 4: returns the brightness of the background at the current location.

7] leftArc() –

Starts to move to the left on an arc with given radius.

8] rightArc() –

Starts to move to the right on an arc with given radius.

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_4 {

Static {

RobotContext.setStartPosition(32,495);

RobotContext.useBackground(“sprites/road.gif”);

}

Prac\_4(){

LegoRobot r=new LegoRobot();

Gear g = new Gear();

LightSensor ls= new LightSensor(SensorPort.S3);

r.addPart(g);

r.addPart(ls);

g.forward();

g.setSpeed(50);

while(true){

int v =ls.getValue();

if(v < 100)

g.forward();

if(v > 350 && v<750)

g.leftArc(0.005);

if(v > 800)

g.rightArc(0.005);

}

}

Public static void main (String args[]){

New Prac\_4();

}

}

OUTPUT:

PRACTICAL 5

AIM: WRITE A PROGRAM TO CREATE A ROBOT THAT DOES A CIRCLE USING 2 MOTORS.

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_5 {

Prac\_5() {

NxtRobot r = new NxtRobot();

Motor A = new Motor(MotorPort.A);

Motor B = new Motor(MotorPort.B);

r.addPart(B);

r.addPart(A);

A.setSpeed(100);

B.setSpeed(100);

A.forward();

B.forward();

While (true){

Tools.delay(200);

A.stop();

Tools.delay(200);

A.forward();

}

}

Public static void main(String arg[]) {

New Prac\_5();

}

}

OUTPUT:

PRACTICAL 6

AIM: WRITE A PROGRAM TO CREATE A PATH FOLLOWING ROBOT.

Description:

1] setStartDirection(double) –

Sets the Nxt starting direction (zero to EAST).

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_6 {

Prac\_6(){

NxtRobot robot=new NxtRobot();

Gear gear=new Gear();

LightSensor ls1=new LightSensor(SensorPort.S1);

LightSensor ls2=new LightSensor(SensorPort.S2);

Robot.addPart(gear);

Robot.addPart(ls1);

Robot.addPart(ls2);

Gear.forward();

Gear.setSpeed(100);

While(true)

{

Int rightValue=ls1.getValue();

Int leftValue=ls2.getValue();

If(leftValue < 10)

Gear.rightArc(0.05);

If(rightValue < 10)

Gear.leftArc(0.05);

If(leftValue > 10 && rightValue > 10)

Gear.forward();

}

}

Public static void main(String args[])

{

New Prac\_6();

}

Static

{

NxtContext.setStartPosition(267,232);

NxtContext.setStartDirection(-90);

NxtContext.useBackground(“sprites/path.gif”);

}

}

OUTPUT:

PRACTICAL 7

AIM: WRITE A PROGRAM TO RESIST OBSTACLES.

Description:

1] TouchSensor(SensorPort) –

Creates a sensor instance connected to the given port.

2] isPressed() –

Polls the touch sensor and returns true, if there is a collision with any of the collision obstacles.

3] backward() –

Starts moving backward and returns immediately.

4] useObstacle(Obstacle) –

Defines the given obstacle to be used as touch obstacle.

CODE:

Import ch.aplu.robotsim.\*;

Public class Prac\_7 {

Prac\_7(){

LegoRobot r=new LegoRobot();

Gear g = new Gear();

TouchSensor t1= new TouchSensor(SensorPort.S1);

TouchSensor t2 = new TouchSensor(SensorPort.S2);

r.addPart(g);

r.addPart(t1);

r.addPart(t2);

g.forward();

g.setSpeed(50);

while(true){

Boolean b1 = t1.isPressed();

Boolean b2 = t2.isPressed();

If(b1 && b2){

g.backward(150);

g.right(400);

g.forward();

}

If(b1){

g.backward(150);

g.left(200);

g.forward();

}

If(b2){

g.backward(150);

g.right(200);

g.forward();

}

}

}

Static {

RobotContext.setStartPosition(100,250);

RobotContext.useObstacle(RobotContext.channel);

}

Public static void main(String args[]){

New Prac\_7();

}

}

OUTPUT:

PRACTICAL 8

AIM: ULTRASONIC SENSOR.

DESC:

1] UltrasonicSensor(SensorPort) –

The port selection determines the position of the sensor and the direction of the beam axis.

2] setBeamAreaColor(Color) –

Sets the color of the beam area (two sector border lines and axis).

3] setProximityCircleColor(Color) –

Sets the color of the circle with center at sensor location and radius equals to the current distance value.

4] getDistance() –

Returns the distance to the nearest target object.

5] useTarget(String, Point[], int, int) –

Creates a target for the ultrasonic sensor using the given sprite image.

CODE:

Import ch.aplu.robotsim.\*;

Import java.awt.Color;

Import java.awt.Point;

Public class Prac\_8 {

Prac\_8() {

LegoRobot robot = new LegoRobot();

Gear gear = new Gear();

Robot.addPart(gear);

UltrasonicSensor us = new UltrasonicSensor(SensorPort.S1);

Robot.addPart(us);

Us.setBeamAreaColor(Color.green);

Us.setProximityCircleColor(Color.lightGray);

Double arc = 0.5;

Gear.setSpeed(50);

Gear.rightArc(arc);

Boolean isRightArc = true;

Int oldDistance = 0;

While (true)

{

Tools.delay(100);

Int distance = us.getDistance();

If (distance == -1)

Continue;

If (distance < oldDistance)

{

If (isRightArc)

{

Gear.leftArc(arc);

isRightArc = false;

}

Else

{

Gear.rightArc(arc);

isRightArc = true;

}

}

oldDistance = distance;

}

}

Static{

Point[] mesh\_bar =

{

New Point(10, 200), new Point(-10, 200),

New Point(-10, -200), new Point(10, -200)

};

RobotContext.useTarget(“sprites/bar1.gif”, mesh\_bar, 200, 250);

RobotContext.useTarget(“sprites/bar1.gif”, mesh\_bar, 300, 250);

RobotContext.setStartPosition(250, 460);

}

Public static void main(String[] args) {

New Prac\_8();

}

}

OUTPUT:

Assignment 1(A):

Aim: Write a program to create a robot to perform rectangular motion using gears

Description:

1] NxtRobot() :

Class that represents a simulated NXT robot brick. Parts (e.g. motors, sensors) may be assembled into the robot to make it doing the desired job.

2] Gear() :

Creates a gear instance with right motor plugged into port A, left motor plugged into port B.

3] addPart() :

Assembles the given part into the robot.

4] setSpeed() :

Sets the speed to the given value (arbitrary units).

5] forward() :

Starts the forward movement for the given duration (in ms) and stops. Method returns at the end of the given duration.

6] left() :

Starts to rotate left (center of rotation at middle of the wheel axes). Method returns immediately, while the movement continues

Code:

Import ch.aplu.robotsim.NxtRobot;

Import ch.aplu.robotsim.Gear;

Public class assignment1A {

Public assignment1A() {

NxtRobot r = new NxtRobot ();

Gear g = new Gear();

r.addPart (g);

g.setSpeed (100);

while (true){

g.forward (800);

g.left (280);

}

}

Public static void main (String [] args){

New assignment1A ();

}

}

Output:

Assignment 1(B):

Aim: Write a program to create a robot to perform circular motion using gears

Description:

1] rightArc() :

Starts to move to the right on arc with given radius. Method returns immediately, while the movement continues.

.

Code:

Import ch.aplu.robotsim.NxtRobot;

Import ch.aplu.robotsim.Gear;

Public class assignment1B {

Public assignment1B () {

NxtRobot r = new NxtRobot ();

Gear g = new Gear ();

r.addPart (g);

g.setSpeed (100);

while (true) {

g.rightArc (0.5);

}

}

Public static void main (String [] args){

New assignment1B ();

}

}

Output

Assignment 2 (A):

Aim: Write a program to create robot to perform a square motion without using gear.

Code:

Import ch.aplu.robotsim.\*;

Public class Assignment\_2a {

Assignment\_2a () {

TurtleRobot t = new TurtleRobot ();

t.setTurtleSpeed (100);

while (true){

t.forward(200);

t.left (90);

}

}

Public static void main (String [] args) {

New Assignment\_1a ();

}

}

Output:

Assignment 2 (B):

Aim: Write a program to create robot to perform a circular motion without using gear.

Code:

Import ch.aplu.robotsim.\*;

Public class Assignment\_2b {

Assignment\_2b () {

TurtleRobot t = new TurtleRobot ();

t.setTurtleSpeed (100);

while (true) {

t.forward (2);

t.left (2);

}

}

Public static void main (String [] args) {

New Assignment\_1b ();

}

}

Output:

Assignment 3:

Aim: Write a program to do a square using while or for loop, change direction based on condition and control motor movement

Description:

1] Motor() :

Creates a motor instance that is plugged into given port.

2] Tools.delay() :

Suspends execution of the current thread for the given amount of time.

Code:

Import ch.aplu.robotsim.\*;

Import java.util.\*;

Public class assignment2 {

Assignment2 () {

Scanner sc = new Scanner (System.in);

NxtRobot r = new NxtRobot ();

Motor m1 = new Motor (MotorPort.A);

Motor m2 = new Motor (MotorPort.B);

r.addPart (m1);

r.addPart (m2);

System.out.println (“Enter 1 for left and 2 for right :”);

Int direction = sc.nextInt ();

Switch (direction) {

Case 1:

For (int i=0; i<4; i++){

M1.forward ();

Tools.delay (1090);

M2.forward ();

Tools.delay (1090);

M1.stop ();

M2.stop ();

}

Break;

Case 2:

For (int i=0; i<4; i++){

M2.forward ();

Tools.delay (1090);

M1.forward ();

Tools.delay (1090);

M1.stop ();

M2.stop ();

}

Break;

}

}

Public static void main (String args[]){

New assignment2 ();

}

}

Output: