Homework-1 Question-1 : Warm Up

$$\begin{aligned} \underset{x_1,x_2,x_3}{\text{maximize}} \quad & 5x_1 - x_2 + 11x_3 \\ \text{subject to:} \quad & 2x_1 \geq x_2 + x_3 \\ & 0 \leq x_j \leq 3, j \in 1,2,3. \end{aligned}$$

In [1]:
```
# Print Status helper function
function printStatus(status, x_1, x_2, x_3)
    println(status)
    println("x1: ", getvalue(x_1))
    println("x2: ", getvalue(x_2))
    println("x3: ", getvalue(x_3))
    println("Max: ", getobjectivevalue(m))
end;
```

Problem Model

In [2]:
```
using JuMP
m = Model()

@variable(m, 0 <= x_1 <= 3)
@variable(m, 0 <= x_2 <= 3)
@variable(m, 0 <= x_3 <= 3)

@constraint(m, 2x_1 >= x_2 + x_3)

status = solve(m)

@objective(m, Max, 5x_1 - x_2 + 11x_3)  # Maximize function

m
```

Out[2]:

$$\begin{aligned} \max \quad & 5x_1 - x_2 + 11x_3 \\ \text{Subject to} \quad & 2x_1 - x_2 - x_3 \geq 0 \\ & 0 \leq x_1 \leq 3 \\ & 0 \leq x_2 \leq 3 \\ & 0 \leq x_3 \leq 3 \end{aligned}$$

In [3]:
```
# Using Clp as solver
using Clp
setsolver(m, ClpSolver(LogLevel=0))
best_time = minimum( [ @elapsed solve(m) for i in 1:10 ] )

status = solve(m)
println("Best time:", best_time)
printStatus(status, x_1, x_2, x_3)
```

```
Best time:0.000129084
Optimal
x1: 3.0
x2: 0.0
x3: 3.0
Max: 48.0
```

In [4]:
```
using ECOS

setsolver(m, ECOSSolver(verbose=0))
best_time = minimum( [ @elapsed solve(m) for i in 1:10 ] )

status = solve(m)
println("Best time:", best_time)
printStatus(status, x_1, x_2, x_3)
```

```
Best time:0.000393042
Optimal
x1: 2.999999998571697
x2: 8.223270011736391e-9
x3: 3.0000000001977236
Max: 47.999999986810174
```

In [5]:
```
using SCS

setsolver(m, SCSSolver(verbose=0))
best_time = minimum( [ @elapsed solve(m) for i in 1:10 ] )

status = solve(m)
println("Best time:", best_time)
printStatus(status, x_1, x_2, x_3)
```

```
Best time:0.000449825
Optimal
x1: 2.999985652990818
x2: 4.149724928776938e-6
x3: 3.0000130627112176
Max: 48.00006780505256
```

Accuracy

Considering the objective of maximizing the provided function, SCS solver achieves the highest objective value. But this is only higher than the other solver's objective value by a order of 10^{-5}. All of the three solvers converge to the same maximum of 48.0 with deviation of a order of 10^{-6}.

Performance

Clp Solver is the fastest among the three solvers. Clp is specifically designed for a very specific type of problem which is to solve Linear Programming problems. Whereas the other solvers are more general and can solve other more complex problems. This is very common trade-off seen between functionality and performance. More the functionality the performance usually go down.

Homework-1 Question-2: Standard form with equality constraints

Convert the following optimization problem to standard form

$$
\begin{aligned}
&\underset{z_1,z_2,z_3,z_4}{\text{maximize}} && 3z_1 - z_2 \\
&\text{subject to:} && -z_1 + 6z_2 - z_3 + z_4 \geq -3 \\
& && 7z_2 + z_4 = 5 \\
& && z_3 + z_4 \leq 2 \\
& -1 \leq z_2 \leq 5, \quad -1 \leq z_3 \leq 5, \quad -2 \leq z_4 \leq 2.
\end{aligned}
$$

The problem is solved in three formats below. First, the problem is modelled using the same equations as mentioned in the question. Second, the variables are transformed to convert the equations to the standard form. Finally, the transformed equations are written in compact form to show the matrices A, b, c and x.

The optimized value and the variables which optimize the model are same in all the three formats.

Original Problem

In [1]:
```
using JuMP

m = Model()

@variable(m, z_1)
@variable(m, -1 <= z_2 <= 5)
@variable(m, -1 <= z_3 <= 5)
@variable(m, -2 <= z_4 <= 2)

@constraint(m, -z_1 + 6z_2 - z_3 + z_4 >= -3)
@constraint(m, 7z_2 + z_4 == 5)
@constraint(m, z_3 + z_4 <= 2)

@objective(m, Max, 3z_1 - z_2)

status = solve(m)
println(status)
println()
println("z1: ", getvalue(z_1))
println("z2: ", getvalue(z_2))
println("z3: ", getvalue(z_3))
println("z4: ", getvalue(z_4))
println("Max: ", getobjectivevalue(m))
m
```

```
Optimal

z1: 8.571428571428571
z2: 0.42857142857142855
z3: -1.0
z4: 2.0
Max: 25.28571428571429
```

Out[1]:

$$
\begin{aligned}
\max \quad & 3z_1 - z_2 \\
\text{Subject to} \quad & -z_1 + 6z_2 - z_3 + z_4 \geq -3 \\
& 7z_2 + z_4 = 5 \\
& z_3 + z_4 \leq 2 \\
& z_1 free \\
& -1 \leq z_2 \leq 5 \\
& -1 \leq z_3 \leq 5 \\
& -2 \leq z_4 \leq 2
\end{aligned}
$$

Standard Form

should look like:

$$
\begin{aligned}
\text{minimize} \quad & c^T x \\
\text{subject to:} \quad & Ax = b \\
& x \geq 0.
\end{aligned}
$$

In [3]:
```
sf_m = Model()

@variable(sf_m, x_1 >= 0)
@variable(sf_m, x_2 >= 0)
@variable(sf_m, x_3 >= 0)
@variable(sf_m, x_4 >= 0)
@variable(sf_m, x_5 >= 0)
@variable(sf_m, s_1 >= 0)
@variable(sf_m, s_2 >= 0)
@variable(sf_m, s_3 >= 0)
@variable(sf_m, s_4 >= 0)
@variable(sf_m, s_5 >= 0)

@constraint(sf_m, (x_1 - 1) + s_1 == 5)
@constraint(sf_m, (x_4 - 1) + s_2 == 5)
@constraint(sf_m, (x_5 - 2) + s_3 == 2)
@constraint(sf_m, (x_2-x_3) - 6(x_1 - 1) + (x_4 - 1) - (x_5 - 2) + s_4 == 3)
@constraint(sf_m, 7(x_1 - 1) + (x_5 - 2) == 5)
@constraint(sf_m, (x_4 - 1) + (x_5 - 2) + s_5 == 2)

@objective(sf_m, Min, -3(x_2-x_3) + (x_1 - 1))

status = solve(sf_m)

println(status)
println()
println("z1: ", getvalue((x_2 - x_3)))
println("z2: ", getvalue((x_1 - 1)))
println("z3: ", getvalue((x_4 - 1)))
println("z4: ", getvalue((x_5 - 2)))
println("Max: ", -getobjectivevalue(sf_m))
sf_m
```

```
Optimal

z1: 8.571428571428571
z2: 0.4285714285714286
z3: -1.0
z4: 2.0
Max: 25.28571428571429
```

Out[3]:

$$
\begin{aligned}
\min \quad & -3x_2 + 3x_3 + x_1 - 1 \\
\text{Subject to} \quad & x_1 + s_1 = 6 \\
& x_4 + s_2 = 6 \\
& x_5 + s_3 = 4 \\
& x_2 - x_3 - 6x_1 + x_4 - x_5 + s_4 = -4 \\
& 7x_1 + x_5 = 14 \\
& x_4 + x_5 + s_5 = 5 \\
& x_1 \geq 0 \\
& x_2 \geq 0 \\
& x_3 \geq 0 \\
& x_4 \geq 0 \\
& x_5 \geq 0 \\
& s_1 \geq 0 \\
& s_2 \geq 0 \\
& s_3 \geq 0 \\
& s_4 \geq 0 \\
& s_5 \geq 0
\end{aligned}
$$

There were four variables in the original problem, namely z_1, z_2, z_3 and z_4. Looking at the lower and upper bounds of each variable, the following transformations were applied to these variables.

- z_1 => $x_2 - x_3$
- z_2 => $x_1 - 1$
- z_3 => $x_4 - 1$
- z_4 => $x_5 - 2$
- There were five slack variables s_1 to s_5 added to convert inequalities to equalities.

Finally, the objective in the original problem was to maximize a function, but the standard form contains a minimization objective. For this the requried transformation was applied by negating the function and then negating again the returned minimized value.

The required matrices A, b ,c and x are mentioned in the code block below. The equations in the code block above have been written in a compact form as per the standard form.

Standard Form (compact)

In [4]:
```
# Mapping of x to variable in above standard format
# x = [x_1; x_2; x_3; x_4; x_5; s_1; s_2; s_3; s_4; s_5]
A = [ 1  0  0  0  0  1  0  0  0  0;
      0  0  0  1  0  0  1  0  0  0;
      0  0  0  0  1  0  0  1  0  0;
     -6  1 -1  1 -1  0  0  0  1  0;
      7  0  0  0  1  0  0  0  0  0;
      0  0  0  1  1  0  0  0  0  1;]    # Required Matrix A
b = [ 6; 6; 4; -4; 14; 5]               # Required Matrix b
c = [ 1; -3; 3; 0; 0; 0; 0; 0; 0; 0]    # Required Matrix c

sfc_m = Model()

@variable(sfc_m, x[1:10] >= 0)      # specify a vector variable, which is the required Matrix x
@constraint(sfc_m, A*x .== b )      # element-wise (vector) inequalities
@objective(sfc_m, Min, dot(c,x) )   # must use dot(c,x) or (c'*x)[1] to return a scalar

status = solve(sfc_m)

println(status)
println()
println("z1: ", getvalue((x[2]-x[3])))
println("z2: ", getvalue((x[1] - 1)))
println("z3: ", getvalue((x[4] - 1)))
println("z4: ", getvalue((x[5] - 2)))
println("Max: ", -(getobjectivevalue(sfc_m)-1))
sfc_m
```

```
Optimal

z1: 8.571428571428571
z2: 0.4285714285714286
z3: -1.0
z4: 2.0
Max: 25.28571428571429
```

Out[4]:

$$
\begin{aligned}
\min \quad & x_1 - 3x_2 + 3x_3 \\
\text{Subject to} \quad & x_1 + x_6 = 6 \\
& x_4 + x_7 = 6 \\
& x_5 + x_8 = 4 \\
& -6x_1 + x_2 - x_3 + x_4 - x_5 + x_9 = -4 \\
& 7x_1 + x_5 = 14 \\
& x_4 + x_5 + x_{10} = 5 \\
& x_i \geq 0 \quad \forall i \in \{1, 2, \ldots, 9, 10\}
\end{aligned}
$$

Homework-1 Question-3

Farmer Jane owns 45 acres of land. She is going to plant each with wheat or corn. Each acre planted with wheat yields $200 profit; each with corn yields $300 profit. The labor and fertilizer used for each acre are given in the table below. One hundred workers and 120 tons of fertilizer are available.

	Wheat	**Corn**
Labor	3 workers	2 workers
Fertilizer	2 tons	4 tons

Problem Data

In [1]:
```
grain = [:wheat, :corn]  # Types of grain

profit = Dict(zip(grain, [200, 300]))  # Profit for each grain.

requirements = [:land, :labor, :fertilizer]  # Requirements to plant grain.

# Quantities available for the requirements
quantities_available = Dict(zip(requirements, [45, 100, 120]))

# recipes (requirements, grain)
using NamedArrays
recipe_mat = [ 1 1
               3 2
               2 4 ]
recipe = NamedArray(recipe_mat, (requirements, grain), ("requirements", "grain"))
```

Out[1]:
```
3×2 Named Array{Int64,2}
requirements \ grain │ wheat   corn
─────────────────────┼─────────────
land                 │     1      1
labor                │     3      2
fertilizer           │     2      4
```

Problem Model

In [2]:
```
using JuMP

m = Model()

# Vector variable for land alloted to each grain.
@variable(m, land[grain] >= 0)

# Expression to calculate total profit, by multiplying land with profit with each grain.
@expression(m, total_profit, sum( profit[g]*land[g] for g in grain) )

# Constraint to limit the recipe requirements to the quantities available.
@constraint(m, constr[r in requirements], sum( recipe[r,g]*land[g] for g in grain )
                        <= quantities_available[r] )

# Objective to maximize the total profit.
@objective(m, Max, total_profit )

solve(m)
println(getvalue(land))
println("Profit: \$", getobjectivevalue(m))
m
```

```
land: 1 dimensions:
[wheat] = 19.99999999999999
[ corn] = 20.000000000000007

Profit: $10000.0
```

Out[2]:

$$
\begin{aligned}
\max \quad & 200land_{wheat} + 300land_{corn} \\
\text{Subject to} \quad & land_{wheat} + land_{corn} \leq 45 \\
& 3land_{wheat} + 2land_{corn} \leq 100 \\
& 2land_{wheat} + 4land_{corn} \leq 120 \\
& land_i \geq 0 \quad \forall i \in \{wheat, corn\}
\end{aligned}
$$

```
Farmer Jane should plant equal amount of corn and wheat grain on 20 acres of the land each to
maximize profit to \$10000.
```

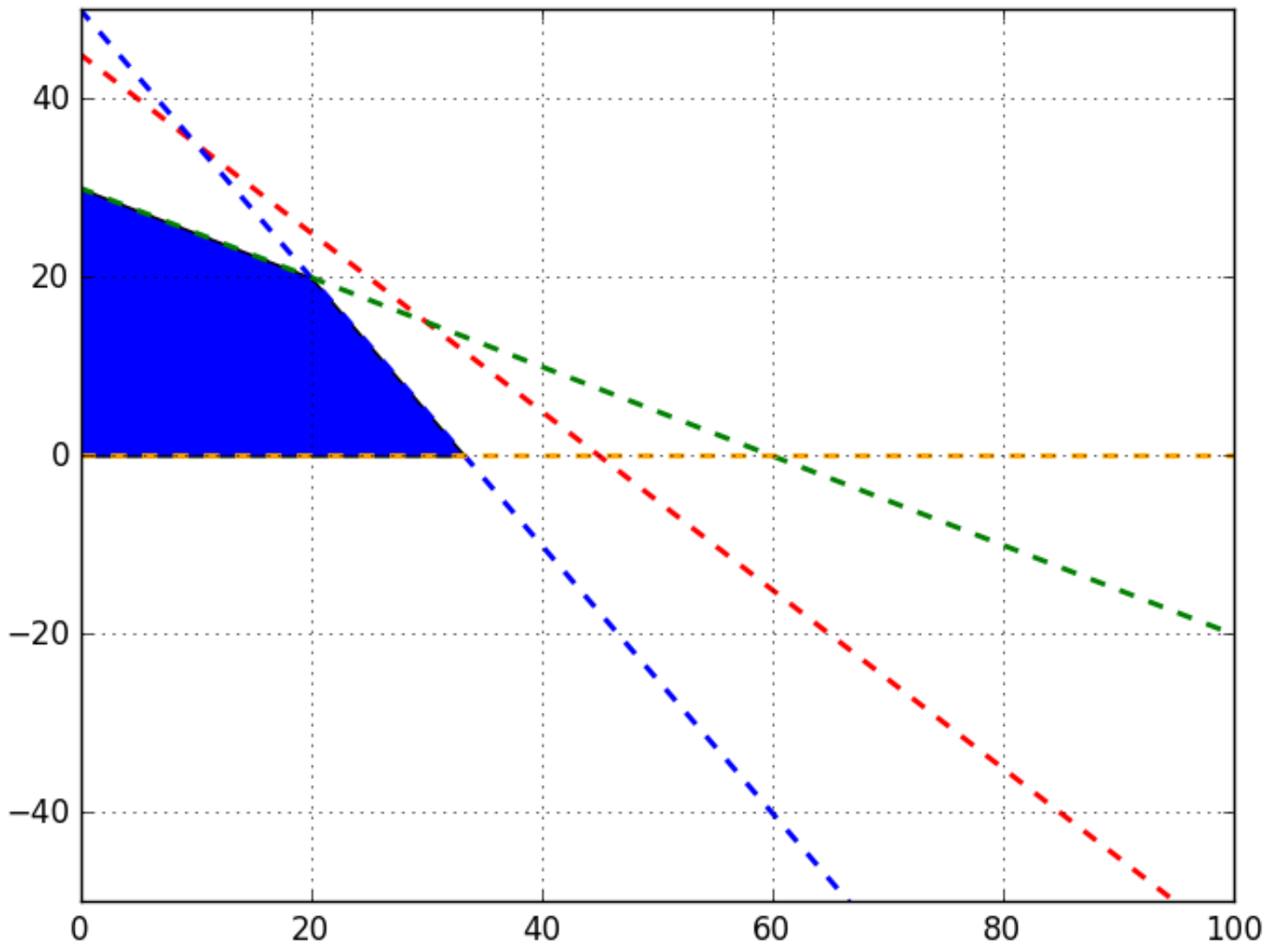
Graphical Solution

As shown by the output of above cell the constraints of the model are represented by five equations. The graph created by code block below draws four of those constraints. The bounded region of the linear program has been filled in as shown below. The bounded region has three vertices which need to be checked, namely (20,20), (100/3, 0) and (0,30). This is found out by inspecting the color of lines intersecting and solving them to get the points. (0,0) is ignored as the solution to that would be no profit. The objective function value at these three points is as follows

- (20,20) => $10000
- (100/3,0) => $6666.67
- (0,30) => $9000

Choosing the maximum value at (20,20) gives us the optimal solution which is verified by the graph below.

In [11]:
```
using PyPlot
land_w = linspace(-10,1000,10000)
land_constraint_y = -(land_w - 45)
labor_constraint_y = -(3land_w - 100)/2
fert_constraint_y = -(2land_w - 120)/4
lower_bound_yw = 0*land_w
grid("on")
xlim(0,100)
ylim(-50,50)
plot(land_w, land_constraint_y, color="red", linewidth=2.0, linestyle="--")
plot(land_w, labor_constraint_y, color="blue", linewidth=2.0, linestyle="--")
plot(land_w, fert_constraint_y, color="green", linewidth=2.0, linestyle="--")
plot(land_w, lower_bound_yw, color="orange", linewidth=2.0, linestyle="--")
fill_between(land_w, lower_bound_yw, min(labor_constraint_y,fert_constraint_y),
    where=min(labor_constraint_y,fert_constraint_y) .>= lower_bound_yw)
```



Out[11]: PyObject <matplotlib.collections.PolyCollection object at 0x7fc927f3b8d0>

In [ ]:

Homework-1 Question-4

Alloy Bending

The company Steelco has received an order for 500 tons of steel to be used in shipbuilding. The steel must have the following characteristics:

Chemical Element	Minimum Grade (%)	Maximum Grade (%)
Carbon (C)	2	3
Copper (Cu)	0.4	0.6
Manganese (Mn)	1.2	1.65

The company has seven different raw materials in stock that may be used for the production of this steel. The following table lists the grades, available amounts and prices for all materials:

Raw Material	C%	Cu%	Mn%	Availability in tons	Cost in $/ton
Iron alloy 1	2.5		1.3	400	200
Iron alloy 2	3		0.8	300	250
Iron alloy 3		0.3		600	150
Copper 1		90		500	220
Copper 2		96	4	200	240
Aluminum 1		0.4	1.2	300	200
Aluminum 2		0.6		250	165

Problem Data

In [1]:
```
# raw materials list, Iron Alloy1,2,3, Copper 1,2, Aluminium 1,2
raw_materials = [:IronAlloy1, :IronAlloy2, :IronAlloy3, :Copper1, :Copper2, :Aluminium1, :Aluminium2]

# elements in the steel
elements = [:Carbon, :Copper, :Manganese]

# cost of each raw material
cost = Dict(zip(raw_materials, [200, 250, 150, 220, 240, 200, 165]))

# availability of each raw material
availability = Dict(zip(raw_materials, [400, 300, 600, 500, 200, 300, 250]))

# weight of elements as per 500 tons of final steel, respecting min and max grades required
min_grade = Dict(zip(elements, [0.02, 0.004, 0.012].*500))
max_grade = Dict(zip(elements, [0.03, 0.006, 0.0165].*500))

# (raw_materials, elements)
using NamedArrays
# Composition matrix raw_materials vs elements
composition_mat = [0.025 0      0.013
                   0.03  0      0.008
                   0     0.003  0
                   0     0.9    0
                   0     0.96   0.04
                   0     0.004  0.012
                   0     0.006  0]

composition = NamedArray( composition_mat, (raw_materials,elements), ("raw_materials","elements") )
println(composition)
```

```
7×3 Named Array{Float64,2}
raw_materials \ elements │    Carbon     Copper  Manganese
─────────────────────────┼──────────────────────────────
IronAlloy1               │     0.025        0.0      0.013
IronAlloy2               │      0.03        0.0      0.008
IronAlloy3               │       0.0      0.003        0.0
Copper1                  │       0.0        0.9        0.0
Copper2                  │       0.0       0.96       0.04
Aluminium1               │       0.0      0.004      0.012
Aluminium2               │       0.0      0.006        0.0
```

Problem Model

In [3]:
```
using JuMP

m = Model()

#Vectored variable to store quantities of all raw materials
@variable(m, quantities[raw_materials] >=0)

#Expressions for total cost of steel created and total quantity after mixing chosen raw materials
@expression(m, total_cost, sum(cost[r]*quantities[r] for r in raw_materials))
@expression(m, total_quantity, sum(quantities[r] for r in raw_materials))

@constraint(m, total_quantity == 500)  # Total quantity should meet the demand of 500 tons

# Constraint on quantities of raw materials to be less than the available raw materials
@constraint(m, avail_constr[r in raw_materials], quantities[r] <= availability[r])

# Constraint on min and max grade of elements required in the final 500 ton steel
@constraint(m, min_grade_constr[e in elements],
               sum(composition[r,e]*quantities[r] for r in raw_materials)
               >= min_grade[e] )
@constraint(m, max_grade_constr[e in elements],
               sum(composition[r,e]*quantities[r] for r in raw_materials)
               <= max_grade[e] )

@objective(m, Min, total_cost)  # Objective to minimize the final cost
solve(m)
println("Minimum Cost: \$", getobjectivevalue(m))
println(getvalue(quantities))
println(m)
```

```
Minimum Cost: $98121.63579168124
quantities: 1 dimensions:
[IronAlloy1] = 400.0
[IronAlloy2] = 0.0
[IronAlloy3] = 39.77630199231039
[   Copper1] = 0.0
[   Copper2] = 2.761272282418735
[Aluminium1] = 57.462425725270876
[Aluminium2] = 0.0

Min 200 quantities[IronAlloy1] + 250 quantities[IronAlloy2] + 150 quantities[IronAlloy3] + 220 quant
ities[Copper1] + 240 quantities[Copper2] + 200 quantities[Aluminium1] + 165 quantities[Aluminium2]
Subject to
 quantities[IronAlloy1] + quantities[IronAlloy2] + quantities[IronAlloy3] + quantities[Copper1] + qu
antities[Copper2] + quantities[Aluminium1] + quantities[Aluminium2] = 500
 quantities[IronAlloy1] ≤ 400
 quantities[IronAlloy2] ≤ 300
 quantities[IronAlloy3] ≤ 600
 quantities[Copper1] ≤ 500
 quantities[Copper2] ≤ 200
 quantities[Aluminium1] ≤ 300
 quantities[Aluminium2] ≤ 250
 0.025 quantities[IronAlloy1] + 0.03 quantities[IronAlloy2] ≥ 10
 0.003 quantities[IronAlloy3] + 0.9 quantities[Copper1] + 0.96 quantities[Copper2] + 0.004 quantitie
s[Aluminium1] + 0.006 quantities[Aluminium2] ≥ 2
 0.013 quantities[IronAlloy1] + 0.008 quantities[IronAlloy2] + 0.04 quantities[Copper2] + 0.012 quan
tities[Aluminium1] ≥ 6
 0.025 quantities[IronAlloy1] + 0.03 quantities[IronAlloy2] ≤ 15
 0.003 quantities[IronAlloy3] + 0.9 quantities[Copper1] + 0.96 quantities[Copper2] + 0.004 quantitie
s[Aluminium1] + 0.006 quantities[Aluminium2] ≤ 3
 0.013 quantities[IronAlloy1] + 0.008 quantities[IronAlloy2] + 0.04 quantities[Copper2] + 0.012 quan
tities[Aluminium1] ≤ 8.25
 quantities[i] ≥ 0 ∀ i ∈ {IronAlloy1,IronAlloy2,…,Aluminium1,Aluminium2}
```

The minimum cost found by the model comes out to be $98121.63. The final quantities used of all alloys is as mentioned in the output of previous cell.