1. Warm-up [5 pts]

Compare the answers found by each solver: which solver is more accurate? Which is fastest (use the @time macro)? Can you speculate as to why?

In [9]:
```
using JuMP, Clp, ECOS, SCS

m1 = Model(solver=ClpSolver(LogLevel=0)) # using the Clp solver

@variable(m1, 1 <= x1 <= 3)
@variable(m1, 1 <= x2 <= 3)
@variable(m1, 1 <= x3 <= 3)

@constraint(m1, 2x1 >= x2 + x3)

@objective(m1, Max, 5x1 - x2 + 11x3)

@time status = solve(m1)

println(m1)
println(status)

println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
println("x3 = ", getvalue(x3))
```

```
  0.000319 seconds (106 allocations: 6.391 KB)
Max 5 x1 - x2 + 11 x3
Subject to
 2 x1 - x2 - x3 >= 0
 1 <= x1 <= 3
 1 <= x2 <= 3
 1 <= x3 <= 3

Optimal
x1 = 3.0
x2 = 1.0
x3 = 3.0
```

In [15]:
```
m2 = Model(solver=ECOSSolver(verbose=0)) # using the ECOS solver

@variable(m2, 1 <= x1 <= 3)
@variable(m2, 1 <= x2 <= 3)
@variable(m2, 1 <= x3 <= 3)

@constraint(m2, 2x1 >= x2 + x3)

@objective(m2, Max, 5x1 - x2 + 11x3)

@time status = solve(m2)

println(m2)
println(status)

println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
println("x3 = ", getvalue(x3))
```

```
  0.000511 seconds (841 allocations: 48.766 KB)
Max 5 x1 - x2 + 11 x3
Subject to
 2 x1 - x2 - x3 >= 0
 1 <= x1 <= 3
 1 <= x2 <= 3
 1 <= x3 <= 3

Optimal
x1 = 2.999999998615505
x2 = 1.0000000052158287
x3 = 3.0000000004504215
```

In [20]:
```
m3 = Model(solver=SCSSolver(verbose=0)) # using the SCS solver

@variable(m3, 1 <= x1 <= 3)
@variable(m3, 1 <= x2 <= 3)
@variable(m3, 1 <= x3 <= 3)

@constraint(m3, 2x1 >= x2 + x3)

@objective(m3, Max, 5x1 - x2 + 11x3)

@time status = solve(m3)

println(m3)
println(status)

println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
println("x3 = ", getvalue(x3))
```

```
  0.000577 seconds (638 allocations: 37.188 KB)
Max 5 x1 - x2 + 11 x3
Subject to
 2 x1 - x2 - x3 >= 0
 1 <= x1 <= 3
 1 <= x2 <= 3
 1 <= x3 <= 3

Optimal
x1 = 2.999979012563557
x2 = 1.000007765927258
x3 = 3.0000200958587935
```

On first run, Clp took the longest time to run, followed by ECOS then SCS, which was the fastest solver of the three.

After multiple runs, Clp runs the quickest, followed by ECOS then SCS which was the slowest. However, both ECOS and SCS have trouble with integer rounding, with ECOS having less error from the actual values.

In [ ]:

2. Original Problem

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

In [44]:
```
using JuMP

m = Model()

@variable(m, z1)
@variable(m, -1 <= z2 <= 5)
@variable(m, -1 <= z3 <= 5)
@variable(m, -2 <= z4 <= 2)

@constraint(m, -z1 + 6z2 - z3 + z4 >= -3)
@constraint(m, 7z2 + z4 == 5)
@constraint(m, z3 + z4 <= 2)

@objective(m, Max, 3z1 - z2)

status = solve(m)

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

```
Optimal

z_1 = 8.571428571428571
z_2 = 0.42857142857142855
z_3 = -1.0
z_4 = 2.0
objective = 25.28571428571429
```

Alternative Standard Form

Rather than using the standard LP form we saw in class, some prefer using a form where all variables are nonnegative, all constraints are equality constraints, and the cost function is a minimization.

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

Notice that the bounds on the variables z_2, z_3 and z_4 can be converted into 2 constraints each; meaning that by adding slack variables s1 through s6, we can convert these into equalities.

In [43]:
```
using JuMP

m3 = Model()

# @variable(m3, z1)
@variable(m3, z1a >= 0)
@variable(m3, z1b >= 0)

# @variable(m3, -1 <= z2 <= 5)
@variable(m3, z2 >= 0)
@variable(m3, s1 >= 0)
@variable(m3, s2 >= 0)
@constraint(m3, z2 + s1 == 5)
@constraint(m3, -(z2 - s2) == -(-1))

# @variable(m3, -1 <= z3 <= 5)
@variable(m3, z3 >= 0)
@variable(m3, s3 >= 0)
@variable(m3, s4 >= 0)
@constraint(m3, z3 + s3 == 5)
@constraint(m3, -(z3 - s4) == -(-1))

# @variable(m3, -2 <= z4 <= 2)
@variable(m3, z4 >= 0)
@variable(m3, s5 >= 0)
@variable(m3, s6 >= 0)
@constraint(m3, z4 + s5 == 2)
@constraint(m3, -(z4 - s6) == -(-2))

# @constraint(m3, 7z2 + z4 == 5)
@constraint(m3, 7z2 + z4 == 5)

# @constraint(m3, -z1 + 6z2 - z3 + z4 >= -3)
@variable(m3, s7 >= 0)
@constraint(m3, -(-(z1a - z1b) + 6z2 - z3 + z4 - s7) == -(-3))

# @constraint(m, z3 + z4 <= 2)
@variable(m3, s8 >= 0)
@constraint(m3, z3 + z4 + s8 == 2)

@objective(m3, Max, 3(z1a - z1b) - z2)

status3 = solve(m3)

println(status3)
println()
println("z1 = ", getvalue((z1a - z1b)))
println("z2 = ", getvalue(z2))
println("z3 = ", getvalue(z3))
println("z4 = ", getvalue(z4))

println()
println("s1 = ", getvalue(s1))
println("s2 = ", getvalue(s2))
println("s3 = ", getvalue(s3))
println("s4 = ", getvalue(s4))
```

```
println("s5 = ", getvalue(s5))
println("s6 = ", getvalue(s6))
println("s7 = ", getvalue(s7))
println("s8 = ", getvalue(s8))

println()
println("objective = ", getobjectivevalue(m3))
```

```
Optimal

z1 = 7.571428571428571
z2 = 0.42857142857142855
z3 = 0.0
z4 = 2.0

s1 = 4.571428571428571
s2 = 1.4285714285714286
s3 = 5.0
s4 = 1.0
s5 = 0.0
s6 = 4.0
s7 = 0.0
s8 = 0.0

objective = 22.28571428571429
```

What are A, b, c,and x? Be sure to explain how the decision variables of your transformed LP relate to those of the original LP.

I don't know, they don't match :(

In [ ]:

3. Crop planning problem [10 pts]

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

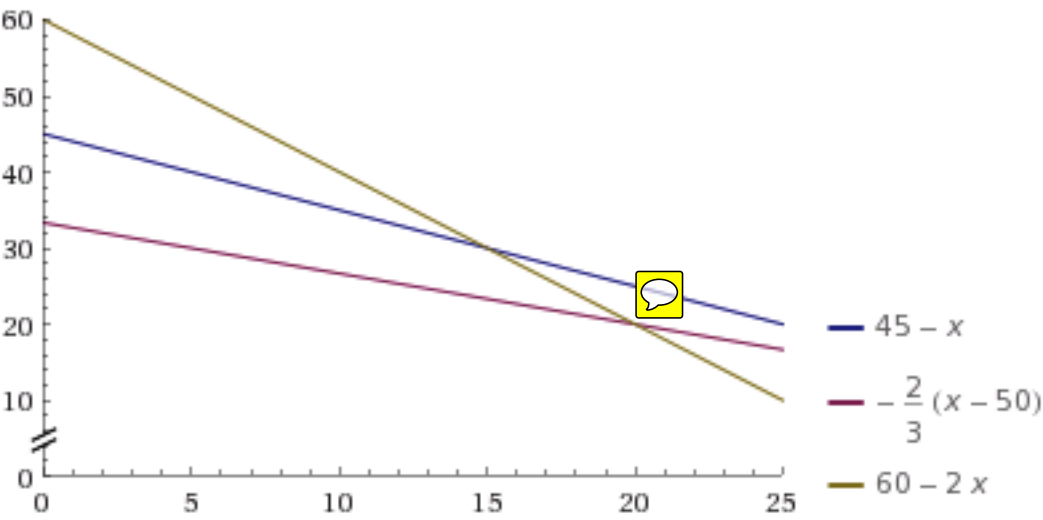
a) How should Jane plant her crops to maximize profit? Model and solve this problem using JuMP.

Solution

In [35]:
```
using JuMP

total_labor = 100
total_fertilizer = 120
total_land = 45

m = Model()
@variable(m, w >= 0)
@variable(m, c >= 0)

@constraint(m, w + c <= total_land)
@constraint(m, 3w + 2c <= total_labor)
@constraint(m, 2w + 4c <= total_fertilizer)

@objective(m, Max, 200w + 300c)

status = solve(m)
println("acres of wheat = ", getvalue(w))
println("acres of corn = ", getvalue(c))
println()
println(status)
println(getobjectivevalue(m))
```

```
acres of wheat = 19.99999999999999
acres of corn = 20.000000000000007

Optimal
10000.0
```

b) Solve the problem graphically and confirm that you obtain the same solution.


60
50
40
30
20
10
0
0
5
10
15
20
25
45 – x
– 2/3 (x – 50)
60 – 2 x

4. Alloy blending [10 pts]

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	0	1.3	400	200
Iron alloy 2	3	0	0.8	300	250
Iron alloy 3	0	0.3	0	600	150
Copper 1	0	90	0	500	220
Copper 2	0	96	4	200	240
Aluminum 1	0	0.4	1.2	300	200
Aluminum 2	0	0.6	0	250	165

Determine the composition of the steel that minimizes the production cost.

In [30]:
```
m = Model()

# define all raw materials to be unsigned
@variable(m, fe1 >= 0)
@variable(m, fe2 >= 0)
@variable(m, fe3 >= 0)
@variable(m, cu1 >= 0)
@variable(m, cu2 >= 0)
@variable(m, al1 >= 0)
@variable(m, al2 >= 0)

raw_materials = [fe1, fe2, fe3, cu1, cu2, al1, al2]
availability  = [400, 300, 600, 500, 200, 300, 250]
cost          = [200, 250, 250, 220, 240, 200, 265]

perc_c        = [2.5, 3, 0, 0, 0, 0, 0]
perc_cu       = [0, 0, 0.3, 90, 96, 0.4, 0.6]
perc_mn       = [1.3, 0.8, 0, 0, 4, 1.2, 0]

# constrain each according to their individual availability
@constraint(m, raw_materials[1] <= availability[1])
@constraint(m, raw_materials[2] <= availability[2])
@constraint(m, raw_materials[3] <= availability[3])
@constraint(m, raw_materials[4] <= availability[4])
@constraint(m, raw_materials[5] <= availability[5])
@constraint(m, raw_materials[6] <= availability[6])
@constraint(m, raw_materials[7] <= availability[7])

# set the sum of all the quantities to be == 500
@expression(m, total_mass, raw_materials[1] + raw_materials[2] + raw_materials[3]
@constraint(m, total_mass == 500)

# set the constraint for the percentage of carbon
@expression(m, total_mass_c, raw_materials[1]*perc_c[1] + raw_materials[2]*perc_c
@expression(m, total_perc_c, total_mass_c / 500)
@constraint(m, 2 <= total_perc_c <= 3)

# set the constraint for the percentage of copper
@expression(m, total_mass_cu, raw_materials[1]*perc_cu[1] + raw_materials[2]*perc
@expression(m, total_perc_cu, total_mass_cu / 500)
@constraint(m, 0.4 <= total_perc_cu <= 0.6)

# set the constraint for the percentage of manganese
@expression(m, total_mass_mn, raw_materials[1]*perc_mn[1] + raw_materials[2]*perc
@expression(m, total_perc_mn, total_mass_mn / 500)
@constraint(m, 1.2 <= total_perc_mn <= 1.65)

@expression(m, total_cost, raw_materials[1]*cost[1] + raw_materials[2]*cost[2] +
@objective(m, Min, total_cost)

solve(m)

println()
println("C % = ", getvalue(total_perc_c))
println("Cu % = ", getvalue(total_perc_cu))
println("Mn % = ", getvalue(total_perc_mn))
```

```
println()
println("Total cost = ", getvalue(total_cost))
```

```
C % = 2.0
Cu % = 0.4
Mn % = 1.2757142857142858

Total cost = 100035.71428571429
```