```
Author: John Wroblewski
netID:  907 287 3103
```

In [39]:

```
m = Model(solver = SCSSolver())
@variable(m, 0 <= x1 <= 3)
@variable(m, 0 <= x2 <= 3)
@variable(m, 0 <= x3 <= 3)
@constraint(m, 2 * x1 >= x2 + x3)
@objective(m, Max, 5 * x1 - x2 + 11 * x3)

@time solve(m)

println("       x1: ", getvalue(x1))
println("       x2: ", getvalue(x2))
println("       x3: ", getvalue(x3))
println("max value: ", getobjectivevalue(m))
```

```
  0.000538 seconds (613 allocations: 36.141 KB)
       x1: 2.999985652990818
       x2: 4.149724928776938e-6
       x3: 3.0000130627112176
max value: 48.00006780505256
----------------------------------------------------------------------------
        SCS v1.1.8 - Splitting Conic Solver
        (c) Brendan O'Donoghue, Stanford University, 2012-2015
----------------------------------------------------------------------------
Lin-sys: sparse-direct, nnz in A = 9
eps = 1.00e-04, alpha = 1.80, max_iters = 20000, normalize = 1, scale = 5.00
Variables n = 3, constraints m = 7
Cones:  linear vars: 7
Setup time: 3.23e-05s
----------------------------------------------------------------------------
 Iter | pri res | dua res | rel gap | pri obj | dua obj | kap/tau | time (s)
----------------------------------------------------------------------------
     0|      inf       inf      -nan      -inf      -nan       inf  1.04e-05
   100| 8.00e-05  1.91e-04  8.48e-06 -4.80e+01 -4.80e+01  2.69e-15  4.47e-05
   140| 4.49e-06  2.70e-06  1.09e-07 -4.80e+01 -4.80e+01  0.00e+00  5.96e-05
----------------------------------------------------------------------------
Status: Solved
Timing: Solve time: 6.09e-05s
        Lin-sys: nnz in L factor: 19, avg solve time: 1.00e-07s
        Cones: avg projection time: 3.15e-08s
----------------------------------------------------------------------------
Error metrics:
dist(s, K) = 1.3565e-17, dist(y, K*) = 0.0000e+00, s'y/m = -9.2050e-18
|Ax + s - b|_2 / (1 + |b|_2) = 4.4865e-06
|A'y + c|_2 / (1 + |c|_2) = 2.7040e-06
|c'x + b'y| / (1 + |c'x| + |b'y|) = 1.0946e-07
----------------------------------------------------------------------------
c'x = -48.0001, -b'y = -48.0001
============================================================================
```

In [40]:

```
m = Model(solver = ECOSSolver())
@variable(m, 0 <= x1 <= 3)
@variable(m, 0 <= x2 <= 3)
@variable(m, 0 <= x3 <= 3)
@constraint(m, 2 * x1 >= x2 + x3)
@objective(m, Max, 5 * x1 - x2 + 11 * x3)

@time solve(m)

println("       x1: ", getvalue(x1))
println("       x2: ", getvalue(x2))
println("       x3: ", getvalue(x3))
println("max value: ", getobjectivevalue(m))
```

```
  0.000498 seconds (755 allocations: 44.969 KB)
       x1: 2.999999998571697
       x2: 8.223270011736391e-9
       x3: 3.0000000001977236
max value: 47.999999986810174

ECOS 2.0.5 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t    mu     step   sigma     IR    |   BT
 0  -2.250e+01  -8.440e+01  +1e+02  2e-01  3e-01  1e+00  1e+01    ---    ---    1  1  - |  -  -
 1  -4.615e+01  -5.603e+01  +2e+01  2e-02  6e-02  7e-01  3e+00  0.8410  6e-02   0  0  0 |  0  0
 2  -4.726e+01  -4.850e+01  +3e+00  3e-03  8e-03  2e-01  4e-01  0.9283  7e-02   0  0  0 |  0  0
 3  -4.799e+01  -4.803e+01  +8e-02  1e-04  2e-04  7e-03  1e-02  0.9798  9e-03   1  0  0 |  0  0
 4  -4.800e+01  -4.800e+01  +9e-04  1e-06  3e-06  8e-05  1e-04  0.9890  1e-04   1  0  0 |  0  0
 5  -4.800e+01  -4.800e+01  +9e-06  1e-08  3e-08  9e-07  1e-06  0.9890  1e-04   1  0  0 |  0  0
 6  -4.800e+01  -4.800e+01  +1e-07  1e-10  3e-10  1e-08  1e-08  0.9890  1e-04   1  0  0 |  0  0

OPTIMAL (within feastol=3.3e-10, reltol=2.2e-09, abstol=1.0e-07).
Runtime: 0.000101 seconds.
```

In [38]:
```
m = Model(solver = ClpSolver())
@variable(m, 0 <= x1 <= 3)
@variable(m, 0 <= x2 <= 3)
@variable(m, 0 <= x3 <= 3)
@constraint(m, 2 * x1 >= x2 + x3)
@objective(m, Max, 5 * x1 - x2 + 11 * x3)

@time solve(m)

println("       x1: ", getvalue(x1))
println("       x2: ", getvalue(x2))
println("       x3: ", getvalue(x3))
println("max value: ", getobjectivevalue(m))
```

```
  0.000248 seconds (98 allocations: 6.109 KB)
       x1: 3.0
       x2: 0.0
       x3: 3.0
max value: 48.0
```

## Comparison of Answers

The most accurate solution was produced by the Clp solver, assuming that the solution in the integers was exact. It was also the fastest solver. I speculate that the Clp solver was the quickest because it might use some sort of "closed form" method to solve, whereas the other solvers seem to be using iterative methods to produce solutions that are within some epsilon of the actual solution. Also Clp is designed to solve linear programs, of which this problem is, where the other solvers are not.

In [71]:
```
m = Model()
@variable(m, z1)
@variable(m, -1 <= z2 <= 5)
@variable(m, -1 <= z3 <= 5)
@variable(m, -2 <= z4 <= 2)
@constraint(m, -1 * z1 + 6 * z2 - z3 + z4 >= -3)
@constraint(m, 7 * z2 + z4 == 5)
@constraint(m, z3 + z4 <= 2)
@objective(m, Max, 3 * z1 - z2)

solve(m)

println("optimal value: ", getobjectivevalue(m))
```

```
optimal value: 25.28571428571429
```

In [153]:
```
m = Model()
@variable(m, x2 >= 0)
@variable(m, x3 >= 0)
@variable(m, x4 >= 0)
@variable(m, s1 >= 0)
@variable(m, s2 >= 0)
@variable(m, s3 >= 0)
@variable(m, s4 >= 0)
@variable(m, s5 >= 0)
@variable(m, s6 >= 0)
@variable(m, s7 >= 0)

@constraint(m, x2 + s3 == 6)
@constraint(m, x3 + s4 == 6)
@constraint(m, x4 + s5 == 4)
@constraint(m, -(s6 - s7) + 6x2 - x3 + x4 - s1 == 4)
@constraint(m, 7x2 + x4 == 14)
@constraint(m, x3 + x4 + s2 == 5)
@objective(m, Min,  -(3(s6 - s7) - x2 + 1))

solve(m)

println("optimal value: ", -getobjectivevalue(m))
```

```
optimal value: 25.28571428571429
```

### Transformations

Let: z1 = x1 = s6 - s7, z2 = x2 - 1, z3 = x3 - 1, z4 = x4 - 2. These change of variables help satisfy the constraint of non-negativity of the variables. Finally we replace x1 with the difference between two variables which both range over all non-negative values. With these new variables and with the given standard form, we get the following values for A, b, c, and x.

In [101]:
```
A = [ 1  0  0  0  0  1  0  0  0  0
      0  1  0  0  0  0  1  0  0  0
      0  0  1  0  0  0  0  1  0  0
      6 -1  1 -1  0  0  0  0 -1 -1
      7  0  1  0  0  0  0  0  0  0
      0  1  1  0  1  0  0  0  0  0 ]

b = [ 6; 6; 4; 4; 14; 5]

c = [ 1 0 0 0 0 0 0 0 -3 -3 ]   #the objective function is affine, so we must remember to subtract the con

x = [ x2; x3; x4; s1; s2; s3; s4; s5; s6; s7 ]  #the s variables are the slack variables
```

## Farmer's Dilemma

In [105]:
```
m = Model()
@variable(m, w >= 0)
@variable(m, c >= 0)
@constraint(m, w + c <= 45)
@constraint(m, 3w + 2c <= 100)
@constraint(m, 2w + 4c <= 120)
@objective(m, Max, 200w + 300c)

solve(m)

println("    wheat: ", getvalue(w))
println("     corn: ", getvalue(c))
println("max proft: ", getobjectivevalue(m))
```

```
    wheat: 19.999999999999999
     corn: 20.000000000000007
max proft: 10000.0
```

We see that maximum profit is reached when 20 acres of wheat and 20 acres of corn are planted.

In [146]:
```
m = Model()
@variable(m, 0 <= m1 <= 400)
@variable(m, 0 <= m2 <= 300)
@variable(m, 0 <= m3 <= 600)
@variable(m, 0 <= m4 <= 500)
@variable(m, 0 <= m5 <= 200)
@variable(m, 0 <= m6 <= 300)
@variable(m, 0 <= m7 <= 250)

@expression(m, ts, m1 + m2 + m3 + m4 + m5 + m6 + m7)
@expression(m, tc, 0.025m1 + 0.03m2)
@expression(m, tcu, 0.003m3 + 0.9m4 + 0.96m5 + 0.004m6 + 0.006m7)
@expression(m, tmn, 0.013m1 + 0.008m2 + 0.04m5 + 0.012m6)
@constraint(m, ts >= 500)
@constraint(m, tc <= 0.03ts)
@constraint(m, tc >= 0.02ts)
@constraint(m, tcu <= 0.006ts)
@constraint(m, tcu >= 0.004ts)
@constraint(m, tmn <= 0.0165ts)
@constraint(m, tmn >= 0.012ts)

@objective(m, Min, 200m1 + 250m2 + 150m3 + 220m4 + 240m5 + 200m6 + 165m7)

solve(m)
println("Amounts of each raw material used, in tons.")
println("Iron Alloy 1: ", getvalue(m1))
println("Iron Alloy 2: ", getvalue(m2))
println("Iron Alloy 3: ", getvalue(m3))
println("Copper 1: ", getvalue(m4))
println("Copper 2: ", getvalue(m5))
println("Aluminum 1: ", getvalue(m6))
println("Aluminum 2: ", getvalue(m7))

println("Carbon grade: ", getvalue(tc) / getvalue(ts) * 100, "%")
println("Copper grade: ", getvalue(tcu) / getvalue(ts) * 100, "%")
println("Manganese grade: ", getvalue(tmn) / getvalue(ts) * 100, "%")
println("Amount of steel produced: ", getvalue(ts))
println("Total cost: \$", getobjectivevalue(m))
```

```
Amounts of each raw material used, in tons.
Iron Alloy 1: 400.0
Iron Alloy 2: 0.0
Iron Alloy 3: 39.77630199231035
Copper 1: 0.0
Copper 2: 2.7612722824187346
Aluminum 1: 57.46242572527088
Aluminum 2: 0.0
Carbon grade: 2.0000000000000004%
Copper grade: 0.6000000000000001%
Manganese grade: 1.2000000000000002%
Amount of steel produced: 499.99999999999994
Total cost: $98121.63579168123
```

Then, by looking at the numbers above, we can determine the composition of the steel to minimize production costs, along with the grades of the finished product and the actual production cost.