Question 1

For the optimizing values of x1, the three solvers found values close to 3, with Clp finding 3.0, while ECOS and SCS found values marginally below 3.0.

For the optimizing values of x2, the three solvers found values close to 0, with Clp finding 0.0, while ECOS and SCS found values marginally above 0.0.

For the optimizing values of x3, the three solvers found values close to 3, with Clp finding 3.0, while ECOS and SCS found values marginally above 3.0.

For the maximum value of the objective function, the three solvers found values close to 48, with Clp finding 48.0. ECOS found a value marginally below 48.0 and SCS found a value marginally above 48.0.

In [21]:

```
# Solving using Clp

using JuMP
using Clp

m = Model(solver = ClpSolver())

@defVar(m, 0 <= x[1:3] <= 3)                         #Variables x1,
 x2, x3
@addConstraint(m, 2*x[1] >= x[2] +x[3])             #Constraint
@setObjective(m, Max, 5*x[1] - x[2] + 11*x[3])      #Objective fun
ction to maximize

status = solve(m)

println(m)
println(status)
println()

println("x1 = ", getValue(x[1]))
println("x2 = ", getValue(x[2]))
println("x3 = ", getValue(x[3]))
println("objective function =  ", getObjectiveValue(m))
```

```
Max 5 x[1] - x[2] + 11 x[3]
Subject to
 2 x[1] - x[2] - x[3] ≥ 0
 0 ≤ x[i] ≤ 3 ∀ i ∈ {1,2,3}

Optimal

x1 = 3.0
x2 = 0.0
x3 = 3.0
objective function =  48.0
```

In [6]:
```
# Solving using ECOS

using JuMP
using ECOS

m = Model(solver = ECOSSolver())

@defVar(m, 0 <= x[1:3] <= 3)                        #Variables x1, x2, x3
@addConstraint(m, 2*x[1] >= x[2] +x[3])             #Constraint
@setObjective(m, Max, 5*x[1] - x[2] + 11*x[3])      #Objective function to maximize

status = solve(m)

println(m)
println(status)
println()

println("x1 = ", getValue(x[1]))
println("x2 = ", getValue(x[2]))
println("x3 = ", getValue(x[3]))
println("objective function =  ", getObjectiveValue(m))
```

```
Max 5 x[1] - x[2] + 11 x[3]
Subject to
 2 x[1] - x[2] - x[3] ≥ 0
 0 ≤ x[i] ≤ 3 ∀ i ∈ {1,2,3}

Optimal

x1 = 2.999999998571697
x2 = 8.223270011736391e-9
x3 = 3.0000000001977236
objective function =  47.999999986810174

ECOS 2.0.4 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.c
om/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: 0.000080 seconds.
```

In [ ]:

In [7]:
```
# Solving using SCS

using JuMP
using SCS

m = Model(solver = SCSSolver())

@defVar(m, 0 <= x[1:3] <= 3)                         #Variables x1, x2, x3
@addConstraint(m, 2*x[1] >= x[2] +x[3])              #Constraint
@setObjective(m, Max, 5*x[1] - x[2] + 11*x[3])    #Objective function to maximize

status = solve(m)

println(m)
println(status)
println()

println("x1 = ", getValue(x[1]))
println("x2 = ", getValue(x[2]))
println("x3 = ", getValue(x[3]))
println("objective function =  ", getObjectiveValue(m))
```

```
Max 5 x[1] - x[2] + 11 x[3]
Subject to
 2 x[1] - x[2] - x[3] ≥ 0
 0 ≤ x[i] ≤ 3 ∀ i ∈ {1,2,3}

Optimal

x1 = 2.999985652990818
x2 = 4.149724928776938e-6
x3 = 3.0000130627112176
objective function =  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: 1.53e-04s
----------------------------------------------------------------------------
 Iter | pri res | dua res | rel gap | pri obj | dua obj | kap/tau | time (s)
----------------------------------------------------------------------------
     0|      inf       inf       nan      -inf       nan       inf  2.57e-05
   100| 8.00e-05  1.91e-04  8.48e-06 -4.80e+01 -4.80e+01  2.69e-15  1.50e-04
   140| 4.49e-06  2.70e-06  1.09e-07 -4.80e+01 -4.80e+01  0.00e+00  2.06e-04
----------------------------------------------------------------------------
Status: Solved
Timing: Solve time: 2.08e-04s
        Lin-sys: nnz in L factor: 19, avg solve time: 2.01e-07s
        Cones: avg projection time: 5.10e-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 [ ]:

Question 2

a)
By the following substitutions:

$z2 = x2 – 1$,
$z3 = x3 – 1$,
$z4 = x4 – 2$

the bounds on z2, z3, z4 becomes:

$0 <= x2, x3 <= 6, 0<= x4 <= 4$

Introduce slack variables:

$x2 + x2' = 6, x3 + x3' = 6, x4 + x4' = 4, x2', x3', x4' => 0.$ (1)

These equations are added as constraints

z1 is transformed to:

$z1 = x1 – x1'$, $x1, x1' => 0$

The inequality constraints are transformed into equality constraints by adding slack variables, x5 and x6, on the first and second original constraints:

$-z1 + 6*z2 – z3 + z4 – x5 = -3$, $x5 => 0$
$7z2 + z4 = 5$
$z3 + z4 + x6 = 2$, $x6 => 0$

With substitution to x-variables the constraints become:

$-x1 + x1' + 6*x2 – x3 + x4 – x5 = 4$
$7x2 + x4 = 14$
$x3 + x4 + x6 = 5$

Add constraints from (1):
$x2 + x2' = 6$
$x3 + x3' = 6$
$x4 + x4' = 4$

In conclusion:

```
A = [-1 1 6 0 -1 0 1 0 -1 0;
      0 0 7 0 0 0 1 0 0 0;
      0 0 0 0 1 0 1 0 0 1;
      0 0 1 1 0 0 0 0 0 0;
      0 0 0 0 1 1 0 0 0 0;
      0 0 0 0 0 0 1 1 0 0 ]
b = [4; 14; 5; 6; 6; 4]
c = [-3; 3; 1; 0; 0; 0; 0; 0; 0; 0]
x = [x1; x1'; x2; x2'; x3; x3'; x4; x4'; x5; x6]
```

Original LP

In [3]:

```
using JuMP

m = Model()

#Variables z1, z2, z3, z4
@defVar(m, z1)
@defVar(m, -1 <= z2 <= 5)
@defVar(m, -1 <= z3 <= 5)
@defVar(m, -2 <= z4 <= 2)

#Constraints
@addConstraint(m, - z1 + 6*z2 -z3 + z4 >= -3)
@addConstraint(m, 7*z2 + z4 == 5)
@addConstraint(m, z3 + z4 <= 2)

#Objective
@setObjective(m, Max, 3*z1 - z2)

#Display results
status = solve(m)

println(m)
println(status)
println()
println("z1 = ", getValue(z1) )
println("z2 = ", getValue(z2) )
println("z3 = ", getValue(z3) )
println("z4 = ", getValue(z4) )

println("objective = ", getObjectiveValue(m) )
```

```
Max 3 z1 - z2
Subject to
 -z1 + 6 z2 - z3 + z4 ≥ -3
 7 z2 + z4 = 5
 z3 + z4 ≤ 2
 z1 free
 -1 ≤ z2 ≤ 5
 -1 ≤ z3 ≤ 5
 -2 ≤ z4 ≤ 2

Optimal

z1 = 8.571428571428571
z2 = 0.42857142857142855
z3 = -1.0
z4 = 2.0
objective = 25.28571428571429
```

In [ ]:

Transformed LP

In [12]:

```
#c, A, b vectors
c = [-3; 3; 1; 0; 0; 0; 0; 0; 0; 0]
A = [-1 1 6 0 -1 0 1 0 -1 0;
     0 0 7 0 0 0 1 0 0 0;
     0 0 0 0 1 0 1 0 0 1;
     0 0 1 1 0 0 0 0 0 0;
     0 0 0 0 1 1 0 0 0 0;
     0 0 0 0 0 0 1 1 0 0]
b = [4; 14; 5; 6; 6; 4]

using JuMP
m = Model()

@defVar(m, x[1:10] >= 0)   #Variables: [x1; x1'; x2; x2'; x3; x3
'; x4; x4'; x5; x6]
@addConstraint(m, A*x .== b)  #Constraint
@setObjective(m, Min, dot(c, x) -1)

#Calculate and display result
println(solve(m))
println()
#Display original z-variables by reversing transformations on x-
variables
println("z1 = ", getValue(x[1] - x[2]))
println("z2 = ", getValue(x[3] - 1))
println("z3 = ", getValue(x[5] - 1))
println("z4 = ", getValue(x[7] - 2))
println("objective = ", -(getObjectiveValue(m)) ) #negated to gi
ve maximum value
```

```
Optimal

z1 = 8.571428571428571
z2 = 0.4285714285714286
z3 = -1.0
z4 = 2.0
objective = 25.28571428571429
```

In [ ]:

Question3

The cost-minimizing steel has the following composition:

Material	Quantity(tons
Iron alloy 1	400.0
Iron alloy 2	0.0
Iron alloy 3	39.7763019923104
Copper 1	0.0
Copper 2	2.761272282418734
Aluminium 1	57.462425725270855
Aluminium 2	0.0

In [10]:
```
#----------Data----------#

#Chemical elements
element = [:carbon, :copper, :manganese]

#Minimum grade of chemical elements required
minGrade = Dict(:carbon => 0.02, :copper => 0.004, :manganese => 0.012)

#Maximum grade of chemical elements required
maxGrade = Dict(:carbon => 0.03, :copper => 0.006, :manganese => 0.0165)

#Raw materials
mat = [:ironAlloy1, :ironAlloy2, :ironAlloy3, :copper1, :copper2, :aluminium1, :a
luminium2]

#Carbon grade of raw materials
carbGrade = Dict(:ironAlloy1 => 0.025, :ironAlloy2 => 0.03, :ironAlloy3 => 0,
          :copper1 => 0, :copper2 => 0, :aluminium1 => 0, :aluminium2 => 0)

#Copper grade of raw materials
copGrade = Dict(:ironAlloy1 => 0, :ironAlloy2 => 0, :ironAlloy3 => 0.003,
          :copper1 => 0.9, :copper2 => 0.96, :aluminium1 => 0.004, :aluminium2 =>
 0.006)

#Manganese grade of raw materials
mangGrade = Dict(:ironAlloy1 => 0.013, :ironAlloy2 => 0.008, :ironAlloy3 => 0,
          :copper1 => 0, :copper2 => 0.04, :aluminium1 => 0.012, :aluminium2 => 0
)

#Amount of raw materials available in tons
matQuant = Dict(:ironAlloy1 => 400, :ironAlloy2 => 300, :ironAlloy3 => 600,
          :copper1 => 500, :copper2 => 200, :aluminium1 => 300, :aluminium2 => 25
0)

#Cost per ton of raw materials in dollars
matCost = Dict(:ironAlloy1 => 200, :ironAlloy2 => 250, :ironAlloy3 => 150,
          :copper1 => 220, :copper2 => 240, :aluminium1 => 200, :aluminium2 => 16
5)

#Amount of steel ordered in tons
orderedQuant = 500



#----------Model----------#

using JuMP
m = Model()

#Variables
@defVar(m, matUsed[mat] >= 0)                                        #Amount o
f a material used in tons

#Expressions
@defExpr(prodQuant, sum{matUsed[i], i in mat})                                 #Tot
al material amount used in production
@defExpr(totalCost, sum{matUsed[i]*matCost[i], i in mat})                      #Tot
al cost
@defExpr(prodCarbGrade, sum{matUsed[i]*carbGrade[i], i in mat} / orderedQuant)
#Carbon grade of the steel produced
@defExpr(prodCopGrade, sum{matUsed[i]*copGrade[i], i in mat} / orderedQuant)
```

```
Materials used:
matUsed: 1 dimensions, 7 entries:
 [aluminium1] = 57.462425725270904
 [aluminium2] = 0.0
 [   copper1] = 0.0
 [   copper2] = 2.761272282418734
 [ironAlloy1] = 400.0
 [ironAlloy2] = 0.0
 [ironAlloy3] = 39.77630199231035
Total cost: 98121.63579168123 dollars
```

In [ ]:

In [61]:
```
#-----Data-----#

#Manufacturing methods
method = [:m1, :m2]
#Germanium grades
grade = [:a, :b, :c, :d, :def]
#Cost per transistor of the two methods and refiring in dollars
cost = Dict(:m1 => 50, :m2 => 70)
refCost = 25

#Grade yields of the two methods in percentage
yield = Dict(
:m1 => Dict(:a => 0.05, :b => 0.15, :c => 0.2, :d => 0.3, :def => 0.3),
:m2 => Dict(:a => 0.15, :b => 0.20, :c => 0.25, :d => 0.2, :def => 0.2)   )

#Grade yields of the refiring process
yieldR = Dict(
:a => Dict(:a => 0.0, :b => 0.0, :c => 0.0, :d => 0.0, :def => 0.0),
:b => Dict(:a => 0.5, :b => 0.5, :c => 0.0, :d => 0.0, :def => 0.0),
:c => Dict(:a => 0.3, :b => 0.3, :c => 0.4, :d => 0.0, :def => 0.0),
:d => Dict(:a => 0.2, :b => 0.2, :c => 0.3, :d => 0.3, :def => 0.0),
:def => Dict(:a => 0.1, :b => 0.2, :c => 0.15, :d => 0.25, :def => 0.3),    )

#Per month furnace capacity
capacity = 20000
#Demand for each grade of transistor
demand = Dict(:a => 1000, :b => 2000, :c => 3000, :d => 3000, :def => 0)
```

Out[61]:
```
Dict{Symbol,Int64} with 5 entries:
  :c   => 3000
  :a   => 1000
  :b   => 2000
  :d   => 3000
  :def => 0
```

In [60]:
```
# -----Model-----#

using JuMP
m= Model()

#Method output
@defVar(m, q[method] >= 0)
#Refiring input
@defVar(m, r[grade] >= 0)


#Method grade output
@defExpr(m1Out[g in grade], q[:m1]*(yield[:m1])[g])
@defExpr(m2Out[g in grade], q[:m2]*(yield[:m2])[g])
#Refiring grade output
@defExpr(rOut[g in grade], sum{r[h]*yieldR[h][g], h in grade} )
#Total transistors produced of each grade
@defExpr(finalOut[g in grade], m1Out[g] + m2Out[g] + rOut[g] - r[g])
#Total cost
@defExpr(totalCost, sum{q[m]*cost[m], m in method} + sum{r[g], g in grade}*refCos
t)

#Constraints
#Input into refiring is no more than output of method 1, 2
@addConstraint(m, rIn[g in grade], r[g] <= m1Out[g] + m2Out[g])
#Furnace capacity constraint
@addConstraint(m, sum{q[m], m in method} + sum{r[g], g in grade} <= capacity)
#Demand fulfillment
@addConstraint(m, meetDem[g in grade], finalOut[g] >= demand[g])

#Minimize cost
@setObjective(m, Min, totalCost)

solve(m)
println("Minimum cost required: ", getValue(totalCost), " dollars")
```

```
Minimum cost required: 641725.352112676 dollars
```

In [ ]:

In [78]:

```
#-----Data-----#
#Electricity demand (MWH). Index corresponds to hour
demand = [43, 40, 36, 36, 35, 38, 41, 46, 49, 48, 47, 47,
          48, 46, 45, 47, 50, 63, 75, 75, 72, 66, 57, 50]

purchaseLimit = 75         #Maximum power can be purchased

#Cost schedule
cost1 = 100
cost2 = 400
cutOff = 50               #Cut off point between first and second cost level

batCap = 30               #Battery power capacity

#Calculate daily cost without battery
totalCostWOBat = 0

for i = 1:24
    if(demand[i] <= cutOff)
        totalCostWOBat += demand[i]*cost1
    else
        totalCostWOBat += cutOff*cost1 + (demand[i] - 50)*cost2
    end
end
```

In [79]:

```
#-----Model-----#
using JuMP
m = Model()

@defVar(m, qBuy1[1:24] >= 0)          #Electricity bounght on the first cost leve
l
@defVar(m, qBuy2[1:24] >= 0)          #Electricity bounght on the second cost lev
el
@defVar(m, batLevel[1:25] >= 0)       #Battery level at each hour

@defExpr(totalCost, sum{qBuy1[i]*cost1 + qBuy2[i]*cost2, i in 1:24}) #Total cost

@addConstraint(m, batLevel[1] == 0)                                   #Initial batt
ery level
@addConstraint(m, batLevelLimit[i in 1:24], batLevel[i] <= batCap)  #Battery leve
l must not exceed its capacity
@addConstraint(m, priceLevel[i in 1:24], qBuy1[i] <= cutOff)        #Cut off for 
the first price level
#Electricty bought must be within the limit
@addConstraint(m, avoidBlackout[i in 1:24], qBuy1[i] + qBuy2[i] <= purchaseLimit)

#Flow constraint
@addConstraint(m, balance[i in 1:24], qBuy1[i] + qBuy2[i] + batLevel[i] == demand
[i] + batLevel[i + 1])

@setObjective(m, Min, totalCost)    #Minimize cost

solve(m)
println("Total cost without battery: ", totalCostWOBat, " dollars")
println("Total cost with battery: ", getValue(totalCost), " dollars")
println("Savings: ", totalCostWOBat - getValue(totalCost), " dollars")
```

```
Total cost without battery: 152400 dollars
Total cost with battery: 143400.0 dollars
Savings: 9000.0 dollars
```

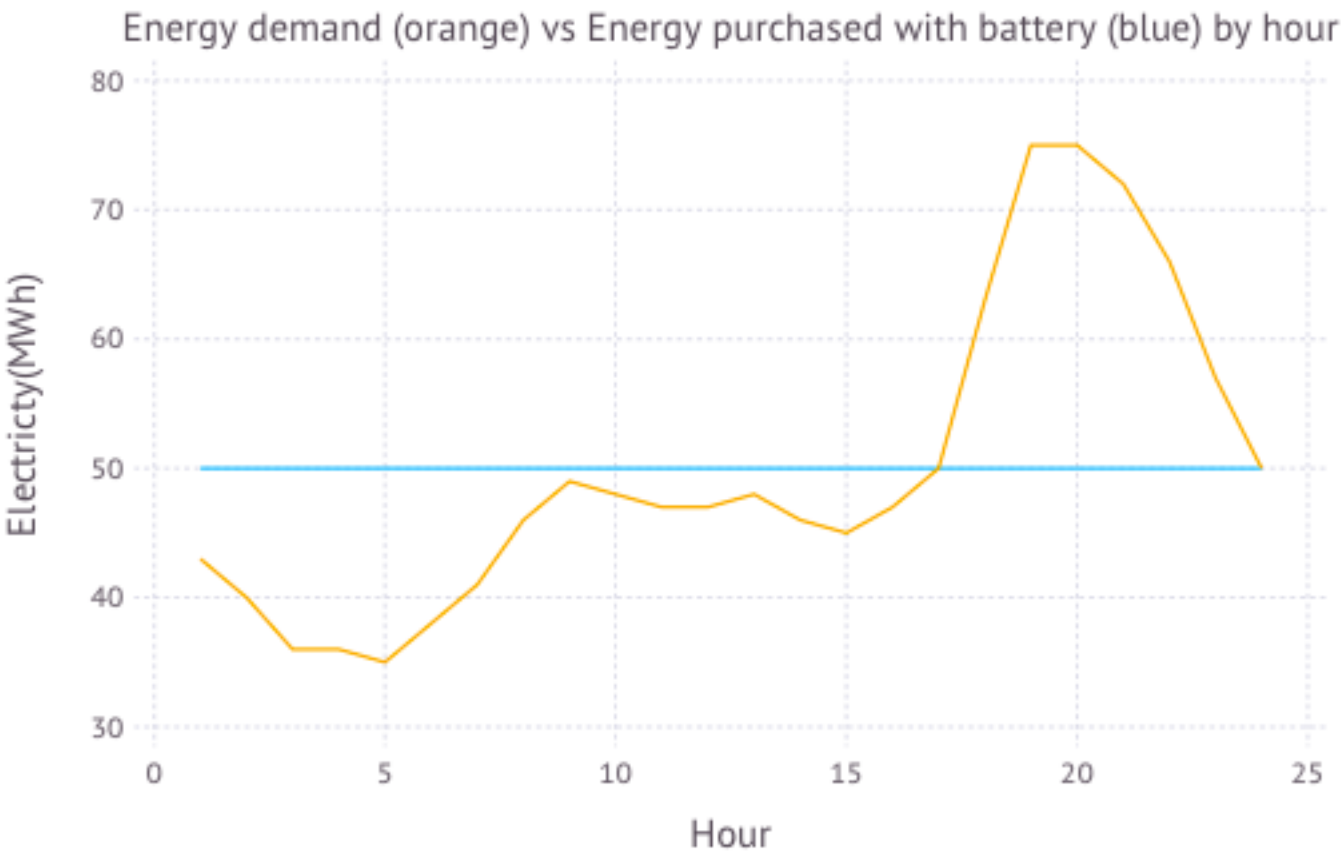
In [80]:

```
#Model with infinite battery capacity
#-----Data-----#
#Electricity demand (MWH). Index corresponds to hour
demand = [43, 40, 36, 36, 35, 38, 41, 46, 49, 48, 47, 47,
          48, 46, 45, 47, 50, 63, 75, 75, 72, 66, 57, 50]

purchaseLimit = 75         #Maximum power can be purchased

#Cost schedule
cost1 = 100
cost2 = 400
cutOff = 50                #Cut off point between first and second cost level

batCap = 30                #Battery power capacity

#Calculate daily cost without battery
totalCostWOBat = 0

for i = 1:24
    if(demand[i] <= cutOff)
        totalCostWOBat += demand[i]*cost1
    else
        totalCostWOBat += cutOff*cost1 + (demand[i] - 50)*cost2
    end
end
```

In [98]:
```
#-----Model-----#
using JuMP
m = Model()

@defVar(m, qBuy1[1:24] >= 0)          #Electricity bounght on the first cost leve
l
@defVar(m, qBuy2[1:24] >= 0)          #Electricity bounght on the second cost lev
el
@defVar(m, batLevel[1:25] >= 0)       #Battery level at each hour

@defExpr(totalCost, sum{qBuy1[i]*cost1 + qBuy2[i]*cost2, i in 1:24}) #Total cost

@addConstraint(m, batLevel[1] == 0)                                 #Initial batt
ery level
#@addConstraint(m, batLevelLimit[i in 1:24], batLevel[i] <= batCap)  #Battery lev
el must not exceed its capacity
@addConstraint(m, priceLevel[i in 1:24], qBuy1[i] <= cutOff)        #Cut off for
the first price level
#Electricty bought must be within the limit
@addConstraint(m, avoidBlackout[i in 1:24], qBuy1[i] + qBuy2[i] <= purchaseLimit)

#Flow constraint
@addConstraint(m, balance[i in 1:24], qBuy1[i] + qBuy2[i] + batLevel[i] == demand
[i] + batLevel[i + 1])

@setObjective(m, Min, totalCost)    #Minimize cost

solve(m)
println("Total cost without battery: ", totalCostWOBat, " dollars")
println("Total cost with battery: ", getValue(totalCost), " dollars")
println("Savings: ", totalCostWOBat - getValue(totalCost), " dollars")
println("Battery levels: ", getValue(batLevel'))
```

```
Total cost without battery: 152400 dollars
Total cost with battery: 120000.0 dollars
Savings: 32400.0 dollars
Battery levels: [0.0 7.0 17.0 31.0 45.0 60.0 72.0 81.0 85.0 86.0 88.0 91.0 94.0
96.0 100.0 105.0 108.0 108.0 95.0 70.0 45.0 23.0 7.0 0.0 0.0]
```

At most 108.0 MW of battery power is used



Energy demand (orange) vs Energy purchased with battery (blue) by hour
80
70
60
50
40
30
Electricty(MWh)
0
5
10
15
20
25
Hour