Homework 2-Question-1

Stigler's diet. True story! In 1945, American economist (and future Nobel laureate) George Stigler published a paper investigating the composition of an optimal diet; minimizing total cost while meeting the recommended daily allowance (RDA) of several nutrients. To answer this question, Stigler tabulated a list of 77 foods and their nutrient content for 9 nutrients: calories, protein, calcium, iron, vitamin A, thiamine, riboflavin, niacin, and ascorbic acid.

Formulate Stigler's diet problem as an LP and solve it. To get you started, Stigler's original data is provided in stigler.csv, and the IJulia notebook stigler.ipynb imports the data and puts it into a convenient array format. How does your cheapest diet compare in annual cost to Stigler's? What foods make up your optimal diet?

Load Data

In [1]:
```
using NamedArrays      # make sure you run Pkg.add("NamedArrays") first!

# import Stigler's data set
raw = readcsv("stigler.csv")
(m,n) = size(raw)

n_nutrients = 2:n      # columns containing nutrients
n_foods = 3:m          # rows containing food names

nutrients = raw[1,n_nutrients][:]   # the list of nutrients (convert to 1-D array)
foods = raw[n_foods,1][:]           # the list of foods (convert to 1-D array)

# lower[i] is the minimum daily requirement of nutrient i.
lower = Dict( zip(nutrients,raw[2,n_nutrients]) )

# data[f,i] is the amount of nutrient i contained in food f.
data = NamedArray( raw[n_foods,n_nutrients], (foods,nutrients), ("foods","nutrients") );
```

Problem Model

In [2]:
```
using JuMP

m = Model()

# Variable vector for quantities of food
@variable(m, quantities[foods] >= 0)

# expression for cost of total food. Since the quantites are per unit dollar, summation of quantities
# is sufficient to calculate the total cost.
@expression(m, cost, sum(quantities[f] for f in foods))

# constraint to lower bound the quantity of nutrients required
@constraint(m, min_req[n in nutrients],sum(data[f,n]*quantities[f] for f in foods) >= lower[n])

# Objective to minimize the total cost
@objective(m, Min, cost)

solve(m)

[println(getvalue(quantities[f]), "\t\t", f) for f in foods if getvalue(quantities)[f] > 0]
println("Minimum Cost yearly: \$", getobjectivevalue(m)*365.25)
```

```
0.02951906167648827            Wheat Flour (Enriched)
0.0018925572907052643           Liver (Beef)
0.011214435246144865            Cabbage
0.005007660466725203            Spinach
0.061028563526693246            Navy Beans, Dried
Minimum Cost yearly: $39.68889711501794
```

Vegan and Gluten Free Diet

I will add the following constraints to the model created above. This constraints are only applied seeing the minimum cost diet solution given by the solver above.

- Liver(Beef) : Comes from animals so it's not vegan.
- Wheat Flour (Enriched) : Contains gluten.
- Lard : Animal fat

So the quantities of these are constrained to be zero.

```
In [9]: non_vegan = ["Wheat Flour (Enriched)", "Liver (Beef)", "Lard"]
        for nv in non_vegan
            @constraint(m, [nv], quantities[nv] == 0)
        end
        solve(m)

        [println(getvalue(quantities[f]), "\t\t", f) for f in foods if getvalue(quantities)[f] > 0]
        println("Minimum Cost yearly: \$", getobjectivevalue(m)*365.25)
```

```
0.005344246335991787            Corn Meal
0.01131324508827593             Cabbage
0.00517574850128731             Spinach
0.10306689112726254             Navy Beans, Dried
Minimum Cost yearly: $45.61977286704162
```

So the minimum cost increases from $39.69 to $45.62 annualy for vegan and gluten-free diet

Homework 2-Question-2 : Construction with constraints

During the next 4 months, a construction firm must complete three projects. Each project has a deadline as well as labor requirements. Project 1 must be completed no later than 3 months from now and requires 8 worker-months of labor. Project 2 must be completed no later than 4 months from now and requires 10 worker-months of labor. Project 3 must be completed no later than 2 months from now and requires 12 worker-months of labor. Each month, 8 workers are available. During a given month, no more than 6 workers can work on a single job. Determine whether all three projects can be completed on time.

Problem Data

In [5]:
```
n_months = 4
n_projects = 3
tasks = range(1,n_projects)

deadlines = Dict(zip(tasks,[3, 4, 2]))   # deadlines of each project
work_required = Dict(zip(tasks,[8, 10, 12]))   # Total worker-months required for each project.
;
```

Problem Model

In [6]:
```
using JuMP

m = Model()
# variables for work required for a project to complete at the start of each month
# (Project, month)
@variable(m, workRequired[1:n_projects,1:n_months+1] >= 0)
#Initial condition to put total worker months required to be equal to work required for a project to
#complete at the start of 1st month.
for p in 1:n_projects
    @constraint(m, workRequired[p,1] == work_required[p])
end
#variable to provide labor to each project each month
@variable(m, 0 <= laborProvider[1:n_projects,1:n_months] <= 6)
#constraint to limit the total availability of workers to 8 per month.
@constraint(m, worker_constr[i in 1:n_months], sum(laborProvider[:,i]) <= 8)
#constraints for flow of work left from month to next month
for p in 1:n_projects
    for mo in 2:n_months+1
        @constraint(m, workRequired[p,mo] + laborProvider[p,mo-1] == workRequired[p,mo-1])
    end
end
# objective to minimize the work required for completion of any project after its deadline.
# A min value of Zero would tell us that its possible to deliver on all of these projects.
@objective(m, Min, (sum(sum(workRequired[p, deadlines[p]+1:n_months+1]) for p in 1:n_projects)))
```

Out[6]: $workRequired_{1,4} + workRequired_{1,5} + workRequired_{2,5} + workRequired_{3,3} + workRequired_{3,4} + workRequired_{3,5}$

In [7]:
```
solve(m)
delayInCompletion = getobjectivevalue(m)
println("Delay in completion of projects: ",delayInCompletion)
laborProvided = getvalue(laborProvider)
println("Work done monthly for Projects")
for p in 1:n_projects
    println("Project ",p," (",sum(laborProvided[p,:]),"): ", laborProvided[p,:])
end
```

```
Delay in completion of projects: 0.0
Work done monthly for Projects
Project 1 (8.0): [2.0,0.0,6.0,0.0]
Project 2 (10.0): [0.0,2.0,2.0,6.0]
Project 3 (12.0): [6.0,6.0,0.0,0.0]
```

The delay in completion of projects is equal to Zero, which means all the projects met their individual deadlines and were completed in the required 4 months.

Homework 2- Question 3: Museum site planning

A site is being investigated as a potential location for a new museum. An aerial plan of the site is shown in the figure below (in feet). The museum will have a circular footprint and law mandates that there be at least 50 feet of clearance between the building and any road. If we want the largest possible museum, where should it be located? What is its optimal radius? Re-plot the figure below along with the optimally designed museum.

Equations for lines enclosing the museum site are as follows

- $y \le 500$
- $3y + 2x \le 2100$
- $3x - y \le 1500$
- $x \ge 0$
- $y \ge 0$

Problem Model and Data

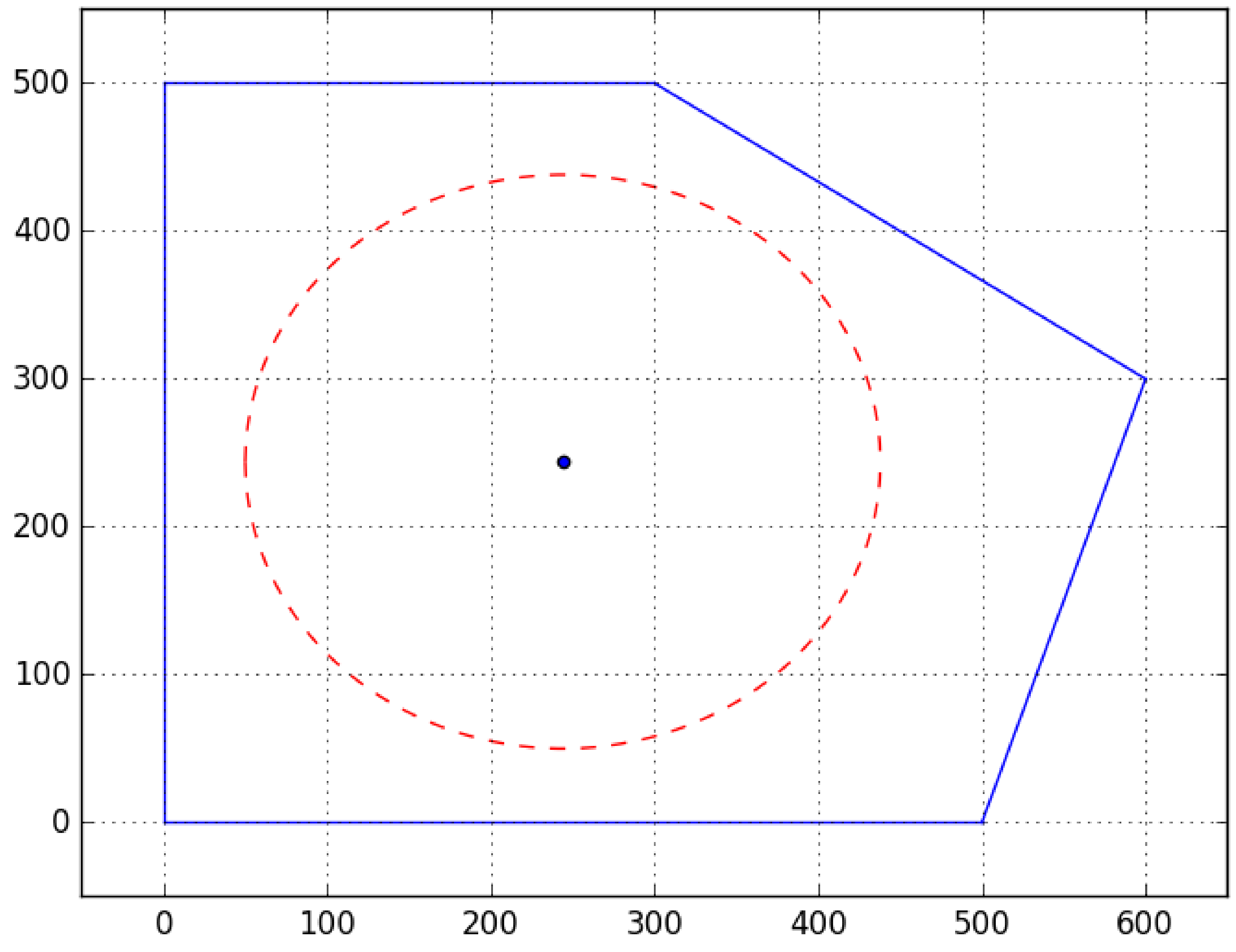
The key idea here is to solve the 2D version of problem discussed in class. The radius as found by the model would be reduced by 50 feet, to give us the required clearance from all the roads

In [1]:
```
A = [0  1; 2 3; 3 -1; -1 0; 0 -1];
b = [500; 2100; 1500; 0; 0]

using JuMP

m = Model()
@variable(m, r >= 0)           # radius
@variable(m, x[1:2])           # coordinates of center
for i = 1:size(A,1)
    @constraint(m, A[i,:]'*x + r*norm(A[i,:]) .<= b[i])
end
@objective(m, Max, r)     # maximize radius

status = solve(m)
center = getvalue(x)
radius = getvalue(r) - 50

println(status)
println("The coordinates of the Chebyshev center are: ", center)
println("The largest possible radius is: ", radius)
```

```
Optimal
The coordinates of the Chebyshev center are: [244.029,244.029]
The largest possible radius is: 194.0285267938019
```

Graph showing the roads and museum site outline

In [2]:
```
using PyPlot
radius = 244.0285267938019
x0 = linspace(0,500,1000)
x1 = linspace(0,300,1000)
x2 = linspace(500,600,1000)
x3 = linspace(300,600,100)
y1 = 0*x1 + 500
y2 = (-2x3 + 2100)/3
y3 = (3x2 - 1500)
grid("on")
xlim(-50,650)
ylim(-50,550)
radius -= 50
plot(x1, y1, color="blue",linewidth=1.0, linestyle="-")
plot(x2, y3, color="blue",linewidth=1.0, linestyle="-")
plot(x3, y2, color="blue",linewidth=1.0, linestyle="-")
plot(x0, 0*x0, color="blue",linewidth=1.0, linestyle="-")
plot(0*x0, x0, color="blue",linewidth=1.0, linestyle="-")
x = linspace(center[1]-radius, center[1]+radius, 1000)
y_upper = sqrt(abs(radius^2 - (x.-center[1]).^2)) + center[2]
y_lower = -sqrt(abs(radius^2 - (x.-center[1]).^2)) + center[2]
plot(x, y_upper, color="red", linewidth=1.0, linestyle="--")
plot(x, y_lower, color="red", linewidth=1.0, linestyle="--")
scatter(center[1], center[2]);
```



Alternative Solution

To confirm that the center or the radius doesn't change if we bring the roads inwards, the following model is contructed by translating all the lines inwards by 50 feet.

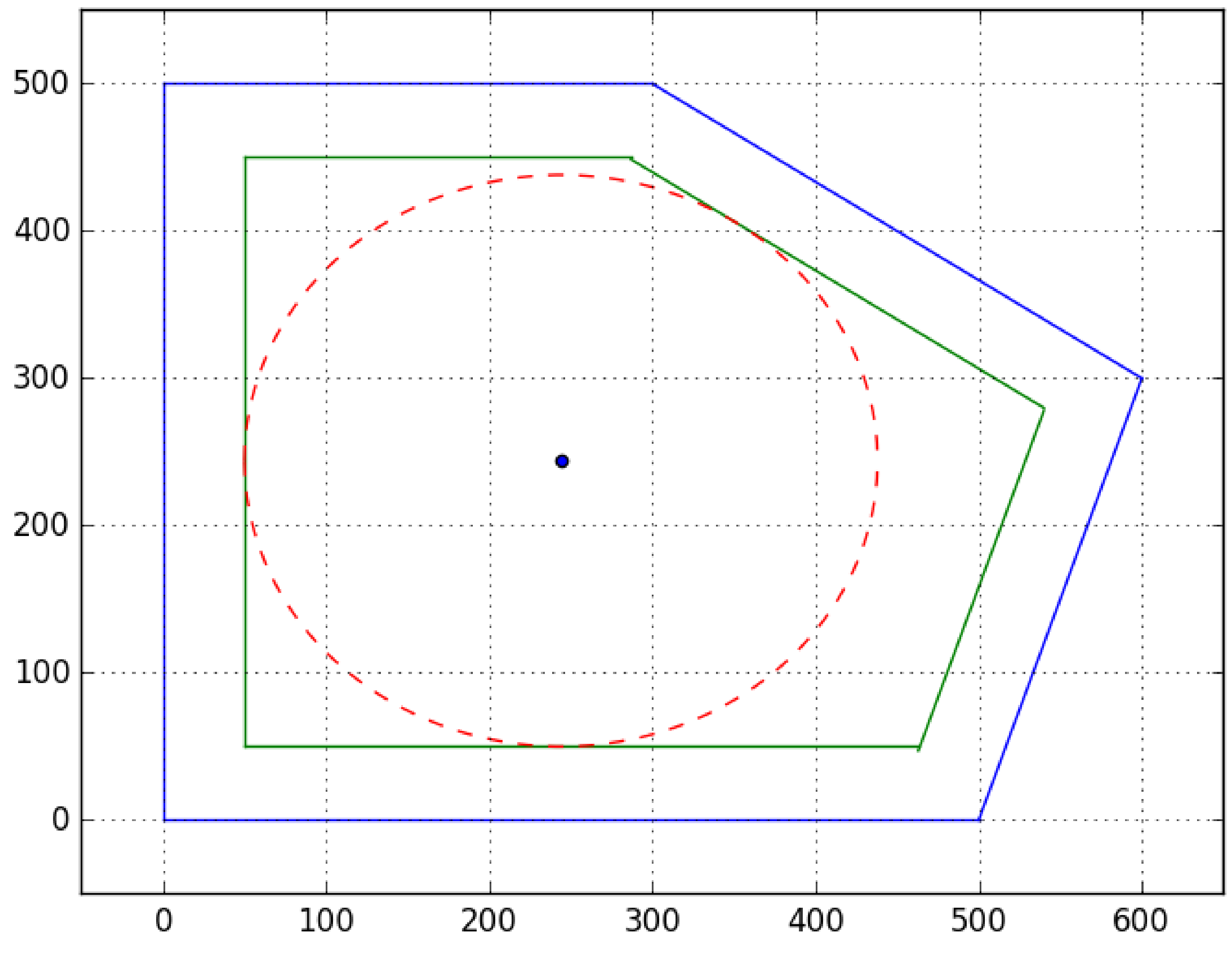
In [3]:
```julia
Af = [0  1; 2 3; 3 -1; -1 0; 0 -1];
# The projection of 50 feet is taken on the y-axis and is added/subtracted to the y-intercept of the
# original line to give the equation of the new line
bf = [450; (2100-3*(50/cos(pi - atan2(2,-3)))); (1500 - 50/cos(atan2(3,1))); -50; -50]

using JuMP

mf = Model()
@variable(mf, rf >= 0)           # radius
@variable(mf, xf[1:2])           # coordinates of center
for i = 1:size(Af,1)
    @constraint(mf, Af[i,:]'*xf + rf*norm(Af[i,:]) .<= bf[i])
end
@objective(mf, Max, rf)     # maximize radius

status = solve(mf)
center = getvalue(xf)
radius = getvalue(rf)

println(status)
println("The coordinates of the Chebyshev center are: ", center)
println("The largest possible radius is: ", radius)
```

```
Optimal
The coordinates of the Chebyshev center are: [244.029,244.029]
The largest possible radius is: 194.02852679380186
```

The solution shows that both the radius and the location of the center remains exactly same.

Graph with original roads and clearance boundaries

```
In [4]: using PyPlot
        x0 = linspace(0,500,1000)
        x1 = linspace(0,300,1000)
        x2 = linspace(500,600,1000)
        x3 = linspace(300,600,100)
        y1 = 0*x1 + 500
        y2 = (-2x3 + 2100)/3
        y3 = (3x2 - 1500)
        grid("on")
        xlim(-50,650)
        ylim(-50,550)
        plot(x1, y1, color="blue",linewidth=1.0, linestyle="-")
        plot(x2, y3, color="blue",linewidth=1.0, linestyle="-")
        plot(x3, y2, color="blue",linewidth=1.0, linestyle="-")
        plot(x0, 0*x0, color="blue",linewidth=1.0, linestyle="-")
        plot(0*x0, x0, color="blue",linewidth=1.0, linestyle="-")

        x0f = linspace(50,450,1000)
        x0f_2 = linspace(50,463,1000)
        x1f = linspace(50,287,1000)
        x2f = linspace(463,540,1000)
        x3f = linspace(287,540,100)
        y1f = 0*x1f + 450
        y2f = (-2x3f + (2100 - 3*(50/cos(pi - atan2(2,-3)))))/3
        y3f = (3x2f - 1500 + 50/cos(atan2(3,1)))
        plot(x1f, y1f, color="green",linewidth=1.0, linestyle="-")
        plot(x2f, y3f, color="green",linewidth=1.0, linestyle="-")
        plot(x3f, y2f, color="green",linewidth=1.0, linestyle="-")
        plot(x0f_2, 0*x0f .+ 50, color="green",linewidth=1.0, linestyle="-")
        plot(0*x0f.+50, x0f, color="green",linewidth=1.0, linestyle="-")

        x = linspace(center[1]-radius, center[1]+radius, 1000)
        y_upper = sqrt(abs(radius^2 - (x.-center[1]).^2)) + center[2]
        y_lower = -sqrt(abs(radius^2 - (x.-center[1]).^2)) + center[2]
        plot(x, y_upper, color="red", linewidth=1.0, linestyle="--")
        plot(x, y_lower, color="red", linewidth=1.0, linestyle="--")
        scatter(center[1], center[2]);
```



Homework-2 Question 4: Electricity grid with storage ¶

The town of Hamilton buys its electricity from the Powerco utility, which charges for electricity on an hourly basis. If less than 50 MWh is used during a given hour, then the cost is $100 per MWh. Any excess beyond 50 MWh used during the hour is charged at the higher rate of $400 per MWh. The maximum power that Powerco can provide in any given hour is 75 MWh. Here is what the average daily electricity demand looks like for Hamilton during the month of January

Hour of day (AM)	1	2	3	4	5	6	7	8	9	10	11	12
Demand (MWh)	43	40	36	36	35	38	41	46	49	48	47	47

Hour of day (PM)	1	2	3	4	5	6	7	8	9	10	11	12
Demand (MWh)	48	46	45	47	50	63	75	75	72	66	57	50

The mayor of Hamilton is concerned because the high electricity use during evening hours is costing the city a lot of money. There is also risk of black-outs at around 7pm because the average demand is dangerously close to Powerco's 75 MW limit. To address these issues, the mayor purchased a large battery with a storage capacity of 30 MWh. The idea is that extra electricity could be purchased early in the day (at the lower rate), stored in the battery, and used later in the day when demand (and prices) are high.

a) How much money can the town of Hamilton save per day thanks to the battery? Assume that the battery begins the day completely drained. Also, to be safe from possible black-outs, limit the amount of electricity purchased every hour to a maximum of 65 MWh.

Problem Data

In [1]:

```
n_hours = 24
demand_arr = [43 40 36 36 35 38 41 46 49 48 47 47 48 46 45 47 50 63 75 75 72 66 57 50]
demand = Dict(zip(1:n_hours,demand_arr))

curr_cost = 0
for h in 1:n_hours
    curr_cost += (demand[h]*100)*(demand[h]<=50) + (demand[h]>50)*((demand[h] - 50)*400 + 50*100)
end

println("Currently the town spends \$",curr_cost," daily for its electricty needs")
```

```
Currently the town spends $152400 daily for its electricty needs
```

Problem Model

In [2]:

```
using JuMP

m = Model()

batteryCapacity = 30

# Variable vector for power taken at lower rate of $100 per hour
@variable(m, 0 <= lRatePower[1:n_hours] <= 50)
# Variable vector for power taken at higher rate of $400 per hour
@variable(m, hRatePower[1:n_hours] >= 0)
# Variable vector for battery charge per hour
@variable(m, battery[1:n_hours+1] >= 0)

# Constraint to limit the total intake of power to be 65MWh
@constraint(m, [lRatePower[h]+hRatePower[h] for h in 1:n_hours] .<= 65)

# Initial constraint to start battery empty at the start of the day
@constraint(m, battery[1] == 0)

# Power flow constraint, which means power taken is either used or
# stored in the battery for the next hour
@constraint(m, [battery[h] + lRatePower[h] + hRatePower[h] - battery[h+1] - demand[h]
        for h in 1:n_hours] .== 0)

# Constraint for battery capcacity
@constraint(m, [battery[h] for h in 1:n_hours+1] .<= batteryCapacity)

# Objective function to minimize the total power intake.
@objective(m, Min, sum(lRatePower[h]*100 + hRatePower[h]*400 for h in 1:n_hours));
```

In [3]:
```
status = solve(m)
println("Status: ",status)
println("After battery of 30MWh the town spends \$",getobjectivevalue(m),
    " daily for its electricty needs")
println("The savings are equal to \$",curr_cost-getobjectivevalue(m))
println()
lRate_a = getvalue(lRatePower)
hRate_a = getvalue(hRatePower)
Battery_a = getvalue(battery)
println("Battery + lRate + hRate = Battery(for next hour) + demand(now) ")
for h in 1:n_hours
    println(Battery_a[h],"\t  ", lRate_a[h],"\t  ", hRate_a[h],"\t=\t",
        Battery_a[h+1],"\t\t \t", demand[h])
end
```

```
Status: Optimal
After battery of 30MWh the town spends $143400.0 daily for its electricty needs
The savings are equal to $9000.0

Battery + lRate + hRate = Battery(for next hour) + demand(now)
0.0       50.0    0.0   =      7.0                     43
7.0       50.0    0.0   =      17.0                    40
17.0      19.0    0.0   =      0.0                     36
0.0       36.0    0.0   =      0.0                     36
0.0       35.0    0.0   =      0.0                     35
0.0       38.0    0.0   =      0.0                     38
0.0       44.0    0.0   =      3.0                     41
3.0       50.0    0.0   =      7.0                     46
7.0       50.0    0.0   =      8.0                     49
8.0       50.0    0.0   =      10.0                    48
10.0      50.0    0.0   =      13.0                    47
13.0      50.0    0.0   =      16.0                    47
16.0      50.0    0.0   =      18.0                    48
18.0      50.0    0.0   =      22.0                    46
22.0      50.0    0.0   =      27.0                    45
27.0      50.0    0.0   =      30.0                    47
30.0      50.0    0.0   =      30.0                    50
30.0      50.0    13.0  =      30.0                    63
30.0      50.0    15.0  =      20.0                    75
20.0      50.0    13.0  =      8.0                     75
8.0       50.0    15.0  =      1.0                     72
1.0       50.0    15.0  =      0.0                     66
0.0       50.0    7.0   =      0.0                     57
0.0       50.0    0.0   =      0.0                     50
```

b) How much money would be saved if the battery had an infinite capacity? In this scenario, how much of the battery's capacity is actually used?

To model the case of infinite capacity battery, the upper bound from the battery capacity is removed

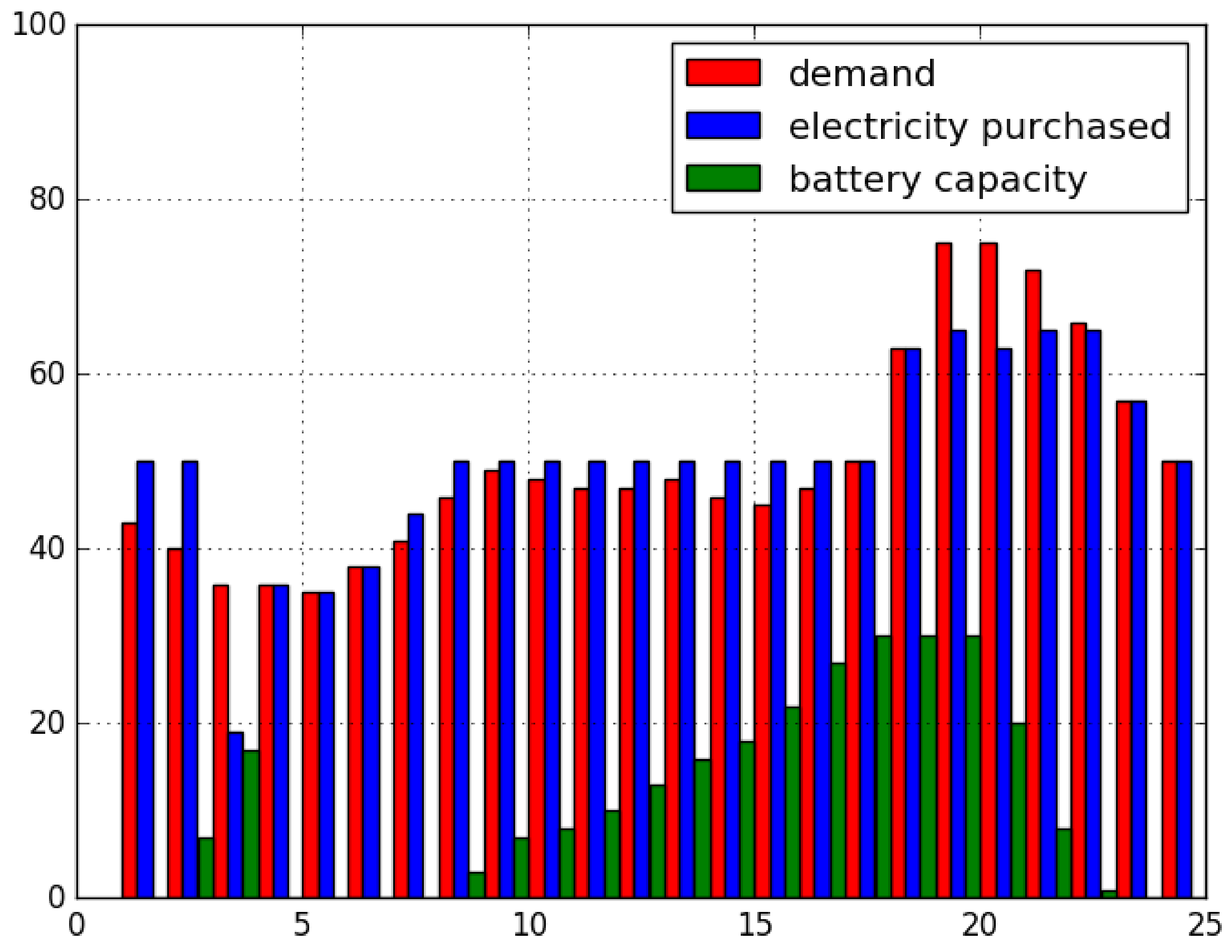
In [4]:
```
batteryCapacity = Inf64
m = Model()
# Variable vector for power taken at lower rate of $100 per hour
@variable(m, 0 <= lRatePower[1:n_hours] <= 50)
# Variable vector for power taken at higher rate of $400 per hour
@variable(m, hRatePower[1:n_hours] >= 0)
# Variable vector for battery charge per hour
@variable(m, battery[1:n_hours+1] >= 0)

# Constraint to limit the total intake of power to be 65MWh
@constraint(m, [lRatePower[h]+hRatePower[h] for h in 1:n_hours] .<= 65)
# Initial constraint to start battery empty at the start of the day
@constraint(m, battery[1] == 0)
# Power flow constraint, which means power taken is either used
# or stored in the battery for the next hour
@constraint(m, [battery[h] + lRatePower[h] + hRatePower[h] - battery[h+1] - demand[h]
        for h in 1:n_hours] .== 0)
# Constraint for battery capcacity
@constraint(m, [battery[h] for h in 1:n_hours+1] .<= batteryCapacity)

# Objective function to minimize the total power intake.
@objective(m, Min, sum(lRatePower[h]*100 + hRatePower[h]*400 for h in 1:n_hours));

status = solve(m)
println("Status: ",status)
println("After battery of ", batteryCapacity," the town spends \$",getobjectivevalue(m),
                " daily for its electricty needs")
println("The savings are equal to \$",curr_cost-getobjectivevalue(m))
println("The maximum capcaity of the battery that is actual used is ",
                maximum(getvalue(battery)))
println()
lRate = getvalue(lRatePower)
hRate = getvalue(hRatePower)
Battery = getvalue(battery)
println("Battery + lRate + hRate = Battery(for next hour) + demand(now) ")
for h in 1:n_hours
    println(Battery[h],"\t  ", lRate[h],"\t  ", hRate[h],"\t=\t",
        Battery[h+1],"\t\t \t", demand[h])
end
```

```
Status: Optimal
After battery of Inf the town spends $120000.0 daily for its electricty needs
The savings are equal to $32400.0
The maximum capcaity of the battery that is actual used is 108.0

Battery + lRate + hRate = Battery(for next hour) + demand(now)
0.0       50.0    0.0   =      7.0                      43
7.0       50.0    0.0   =      17.0                     40
17.0      50.0    0.0   =      31.0                     36
31.0      50.0    0.0   =      45.0                     36
45.0      50.0    0.0   =      60.0                     35
60.0      50.0    0.0   =      72.0                     38
72.0      50.0    0.0   =      81.0                     41
81.0      50.0    0.0   =      85.0                     46
85.0      50.0    0.0   =      86.0                     49
86.0      50.0    0.0   =      88.0                     48
88.0      50.0    0.0   =      91.0                     47
91.0      50.0    0.0   =      94.0                     47
94.0      50.0    0.0   =      96.0                     48
96.0      50.0    0.0   =      100.0                    46
100.0     50.0    0.0   =      105.0                    45
105.0     50.0    0.0   =      108.0                    47
108.0     50.0    0.0   =      108.0                    50
108.0     50.0    0.0   =      95.0                     63
95.0      50.0    0.0   =      70.0                     75
70.0      50.0    0.0   =      45.0                     75
45.0      50.0    0.0   =      23.0                     72
23.0      50.0    0.0   =      7.0                      66
7.0       50.0    0.0   =      0.0                      57
0.0       50.0    0.0   =      0.0                      50
```

Plot

c) Make a plot that shows (i) the typical energy demand vs time of day (ii) the electricity purchased using the strategy found in part a) vs time of day, and (iii) the battery capacity used as a function of time (draw all three plots on the same axes).

In [7]:
```
using PyPlot
hours = linspace(1,24,24)
grid("on")
xlim(0,25)
ylim(0,100)
bar(hours,demand_arr',width=0.33,color="red",align="edge", label="demand")
bar(hours.+0.33,lRate_a.+hRate_a,width=0.33,color="blue", align="edge", label="electricity purchased"
bar(hours.+0.67,Battery_a[1:24],width=0.33,color="green", align="edge", label="battery capacity");
legend();
```



d) Comment on whether the solutions you found are unique. Are other solutions possible? Why? Suggest a way of finding another optimal solution.

The solution consists of all of the possible electricty quantities which can be bought in a particular hour which do not change the total cost incurred through out the day.

The solution found above for part a) is not unique. Since the lower electricity rate applies to any quantity bought below 50MWh, a lot of solutions can exist with the same optimal cost but different combinations of quantity. For example, at 3AM the demand is 36 and the lower rate electricity bought is 19MWh, If instead the lower rate electricity of 25MWh(a sample value chosen, many other possible) was bought it could have been stored in the battery and used in the next hour. The total cost remains the same but the solution differs.