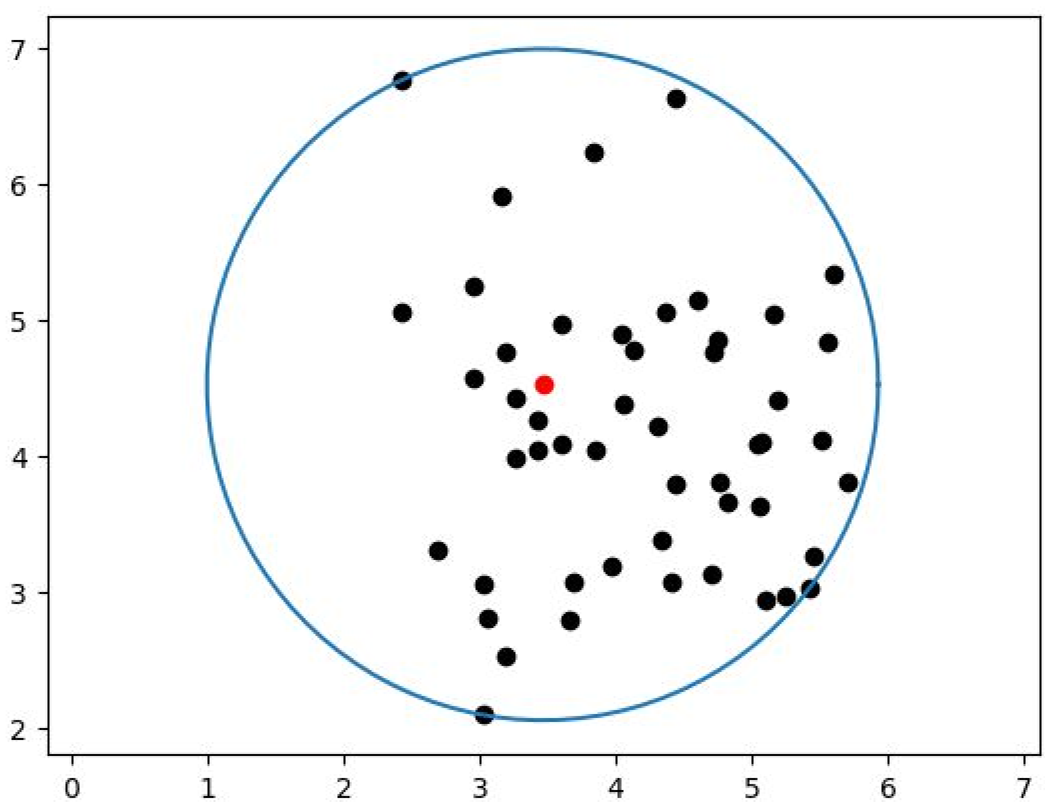
In [1]:

```
#Problem 1

using PyPlot, JuMP, Gurobi
X = 4 + randn(2,50) # generate 50 random points
t = linspace(0,2pi,100) # parameter that traverses the circle
# model solves for radius squared due to needing PSD
# and center point
m = Model(solver=GurobiSolver(OutputFlag=0))
@variable(m, r2 >= 0)
@variable(m, x[1:2])
@constraint(m, pos[i = 1:50], dot((X[:, i] - x), (X[:, i] - x)) <= r2)
@objective(m, Min, r2)
solve(m)
# model solves for radius squared due to needing PSD
r = sqrt(getobjectivevalue(m))
x1 = getvalue(x[1])
x2 = getvalue(x[2])
println("Radius is: ", r)
println("Center point coordinates are: (", x1, ",", x2, ")")
# plot circle radius r with center (x1,x2)
plot( x1 + r*cos(t), x2 + r*sin(t))
scatter( X[1,:], X[2,:], color="black") # plot the 50 points
axis("equal") # make x and y scales equal
# plot the center
scatter(x1,x2,color="red");
```

```
Academic license - for non-commercial use only
Radius is: 2.465731708804874
Center point coordinates are: (3.462036947827601,4.522762785474279)
```



```
#Problem 2a
Q = [2 4 -3; 4 2 -3; -3 -3 9;]
#v = [x y z]
3×3 Array{Int64,2}:
  2   4  -3
  4   2  -3
 -3  -3   9


#Problem 2b
(L, U) = eig(Q)
#Here we see that one of the eigenvalues of Q is -2, so Q is not pisitive definite, meaning it
#is not an elipsoid.
```

Out[147]:

```
([-2.0, 3.0, 12.0], [0.707107 -0.57735 -0.408248; -0.707107 -0.57735 -0.408248; 0.0 -0.57735 0.816497])
```

In [148]:

```
#Problem 2c
#Q equals U*diagm(L)*U' so diagm(L) can be written as the differece of the matrices

M = [[0 0 0]; [0 3 0]; [0 0 12]]
N = [[2 0 0]; [0 0 0]; [0 0 0]];
println("U*M*U' - U*N*U': ", U*M*U' - U*N*U')
println("U: ", U)
U*M*U' - U*N*U': [2.0 4.0 -3.0; 4.0 2.0 -3.0; -3.0 -3.0 9.0]
U: [0.707107 -0.57735 -0.408248; -0.707107 -0.57735 -0.408248; 0.0 -0.57735 0.816497]

println("U*U': ", U*U')
U*U': [1.0 -8.32667e-17 1.11022e-16; -8.32667e-17 1.0 1.11022e-16; 1.11022e-16 1.11022e-16 1.0]
```

$$A = U\begin{bmatrix} 0 & 0 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & \sqrt{12} \end{bmatrix}U^T$$

$$B = U\begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}U^T$$

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

```
A = U * [[0 0 0]; [0 sqrt(3) 0]; [0 0 sqrt(12)]] * U'
B = U * [[sqrt(2) 0 0]; [0 0 0]; [0 0 0]] * U'
println("A'*A + B'*B: ", A'*A + B'*B)
A'*A + B'*B: [4.0 2.0 -3.0; 2.0 4.0 -3.0; -3.0 -3.0 9.0]
```

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Now we have:

$$\begin{aligned}
\|Av\|^2 - \|Bv\|^2 &= (Av)^T(Av) - (Bv)^T(Bv) \\
&= v^T A^T A v - v^T B^T B v \\
&= v^T (A^T A - B^T B) v \\
&= v^T \left(U \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 12 \end{bmatrix} U^T - U \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T \right) v \\
&= v^T Q v
\end{aligned}$$

In [ ]:

```
#Problem 2d
```

Contraint(1) can be written as: $v^T Q v \le 1$

Since $Q = U \begin{bmatrix} -2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 12 \end{bmatrix} U^T$, let $z = U^T v$

U is orthogonal so:

$z = U^T v = U^{-1} v$

$v = Uz$

$x^2 + y^2 + z^2 = \|v\|^2 = v^T v = (Uz)^T Uz = z^T U^T U z = z^T z = \|z\|^2$

the contraint, then, can be written as $z^T \begin{bmatrix} -2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 12 \end{bmatrix} z \le 1$

Since $\|v\|^2 = \|z\|^2$, assume we want $\|v\|^2 = k$ where k is an arbitrary large number. Then the problem can be converted to:

$z_1^2 + z_2^2 + z_3^2 = k$ (1)

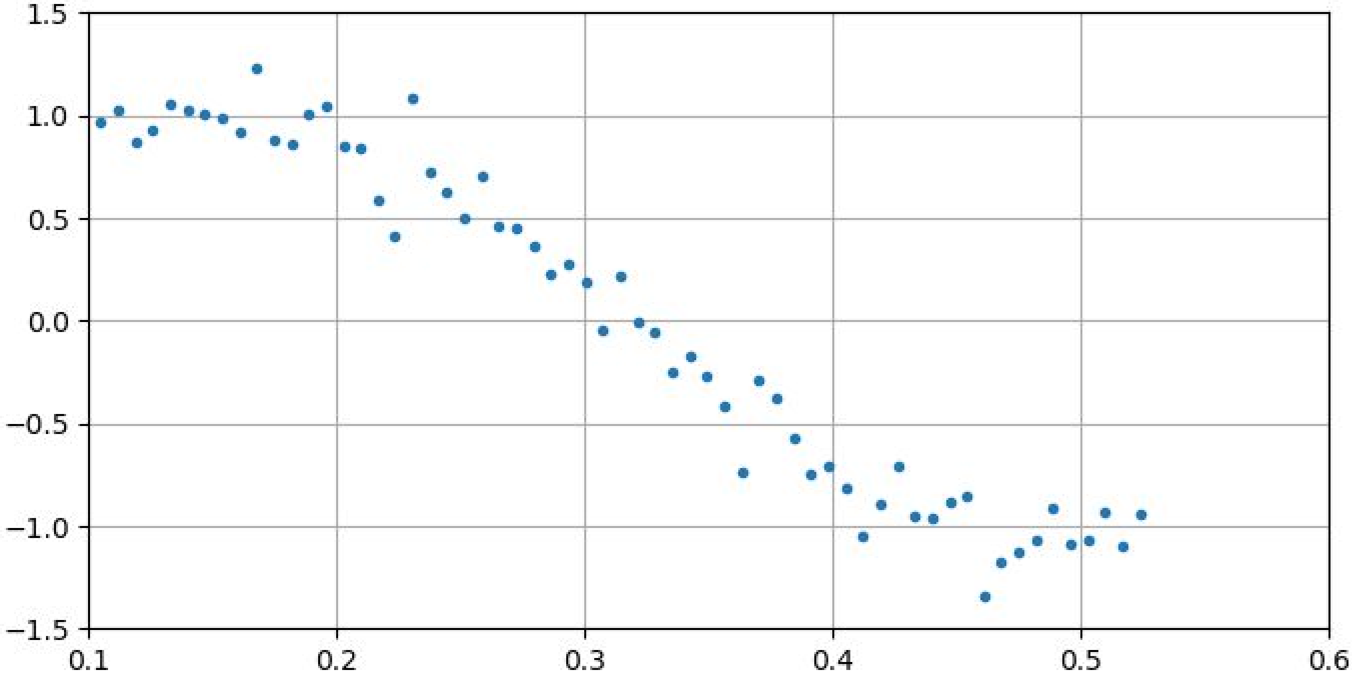
$-2z_1^2 + 3z_2^2 + 12z_3^2 \le 1$ (2)

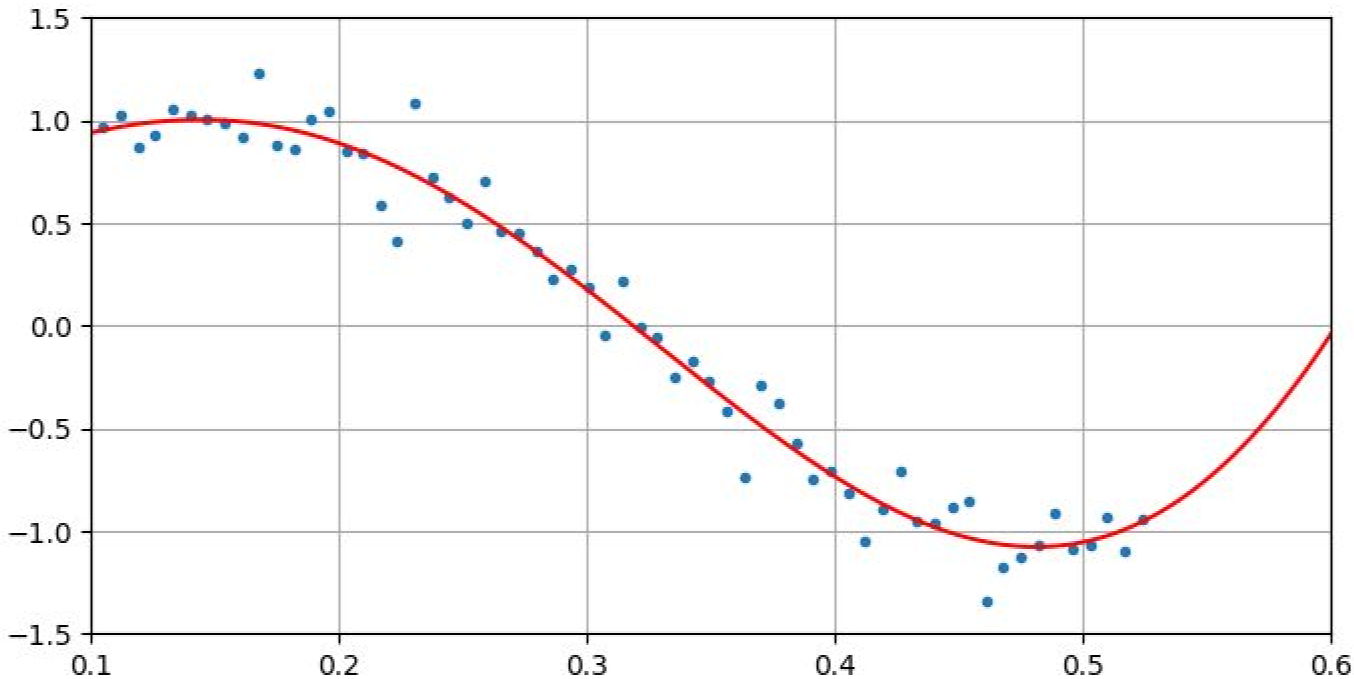
To solve, set one element as fixed and solve for the other two. Then use $v = Uz$ to find (x,y,z) that satisfies (1) with arbitrarily large k.

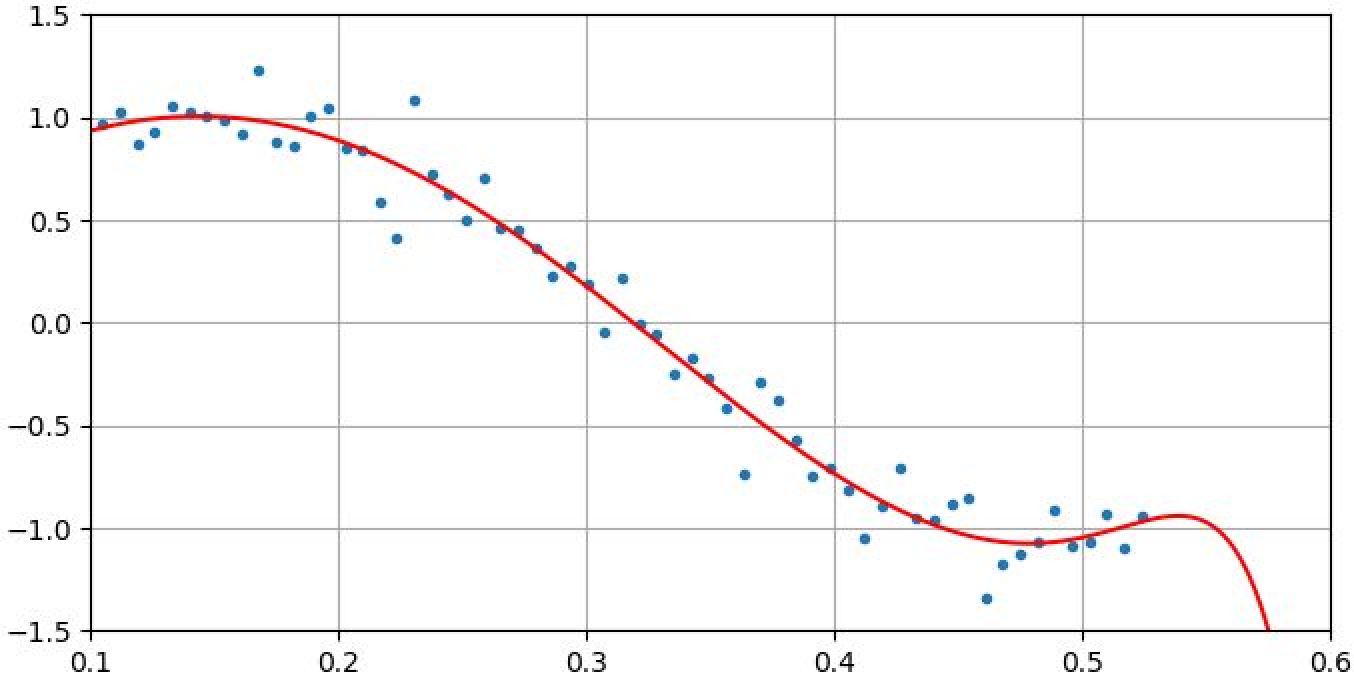
In [136]:

```
#problem 3
using PyPlot, JuMP, Gurobi
raw = readcsv("lasso_data.csv");
x = raw[:,1]
y = raw[:,2]
figure(figsize=(8,4))
p = plot(x, y, ".")
axis([.1,.6,-1.5,1.5])
grid()

function f(k)
k = k
```

```
n = length(x)
A = zeros(n,k+1)
for i in 1:n
    for j = 1:k+1
        A[i,j] = x[i]^(k+1-j)
    end
end

m = Model(solver = GurobiSolver(OutputFlag=0))

@variable(m, u[1:k+1])
@objective(m, Min, sum((y-A*u).^2))

status = solve(m)
uopt = getvalue(u)
println(status)
println(getobjectivevalue(m))
println(getvalue(u))

npts = 10000
xfine = linspace(0,10,npts)
ffine = ones(npts)

for j = 1:k
    ffine = [ffine .*xfine ones(npts)]
end
yfine = ffine * uopt
figure(figsize=(8,4))
plot(x,y, ".")
plot(xfine,yfine, "r-")
axis([.1,.6,-1.5,1.5])
grid()
end
f(5)
f(15)
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.1
0.2
0.3
0.4
0.5
0.6


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.1
0.2
0.3
0.4
0.5
0.6




```
Academic license - for non-commercial use only
Optimal
1.0144487799939412
[-320.872, 619.059, -333.419, 41.1237, 2.18777, 0.584456]
Academic license - for non-commercial use only
Optimal
1.0135488280265221
[-3.37904e5, 1.86649e5, 74678.9, -4481.85, -17029.3, -8090.89, -745.502, 1319.48, 774.64, 312.963,
-144.165, -31.6809, 18.4594, -40.3935, 10.9502, 0.228989]
```

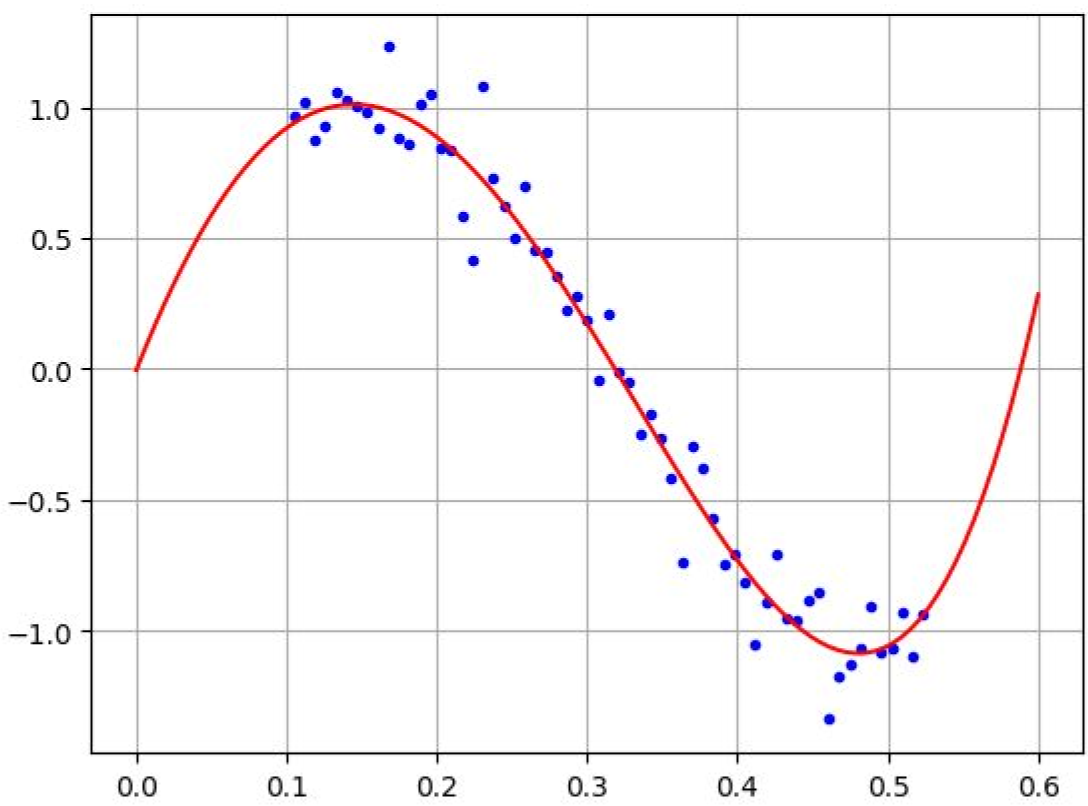
In [ ]:

```
# Here we see that the magnitudes of the coeffitients are very large.
```

In [101]:

```
#Problem 3b
k = 15

n = length(x)
A = zeros(n, k+1)
for i = 1:n
   for j = 1:k+1
      A[i,j] = x[i]^(k+1-j)
   end
end
```

```julia
using JuMP, Gurobi

λ = 1/(10^6)

m = Model(solver=GurobiSolver(OutputFlag=0))

@variable(m, u[1:k+1])
@objective(m, Min, sum((y-A*u).^2) + λ*sum(u.^2))
status = solve(m)

uopt = getvalue(u)
println(status)
println("uopt:", uopt)
println("Error:", sum((y-A*uopt).^2))

using PyPlot

npts = 100
xfine = linspace(0, 0.6, npts)
ffine = ones(npts)

for j = 1:k
    ffine = [ffine.*xfine ones(npts)]
end

yfine = ffine * uopt

plot(x,y,"b.")
plot(xfine,yfine,"r-")
grid()
```



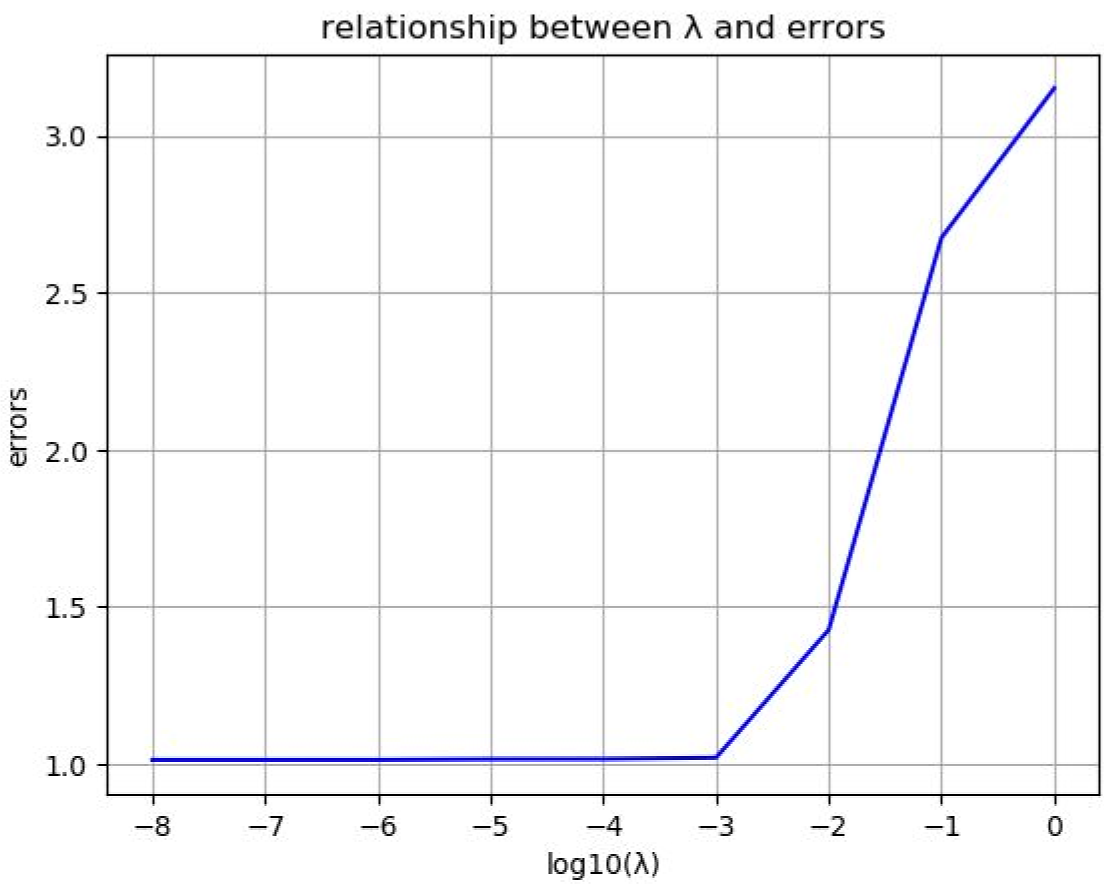
```
Academic license - for non-commercial use only
Optimal
uopt:[-0.59465, -0.975681, -1.54994, -2.35464, -3.35103, -4.29544, -4.51095, -2.58953, 3.71018, 16.501, 33.9216, 42.0711, 11.6188, -55.0281, 14.6015, -0.00540095]
Error:1.0168419826436905
```

```
# Here we see that using L2 regularization causes the error to get larger while the
#magnitudes of the coefficients get smaller
```

In [119]:

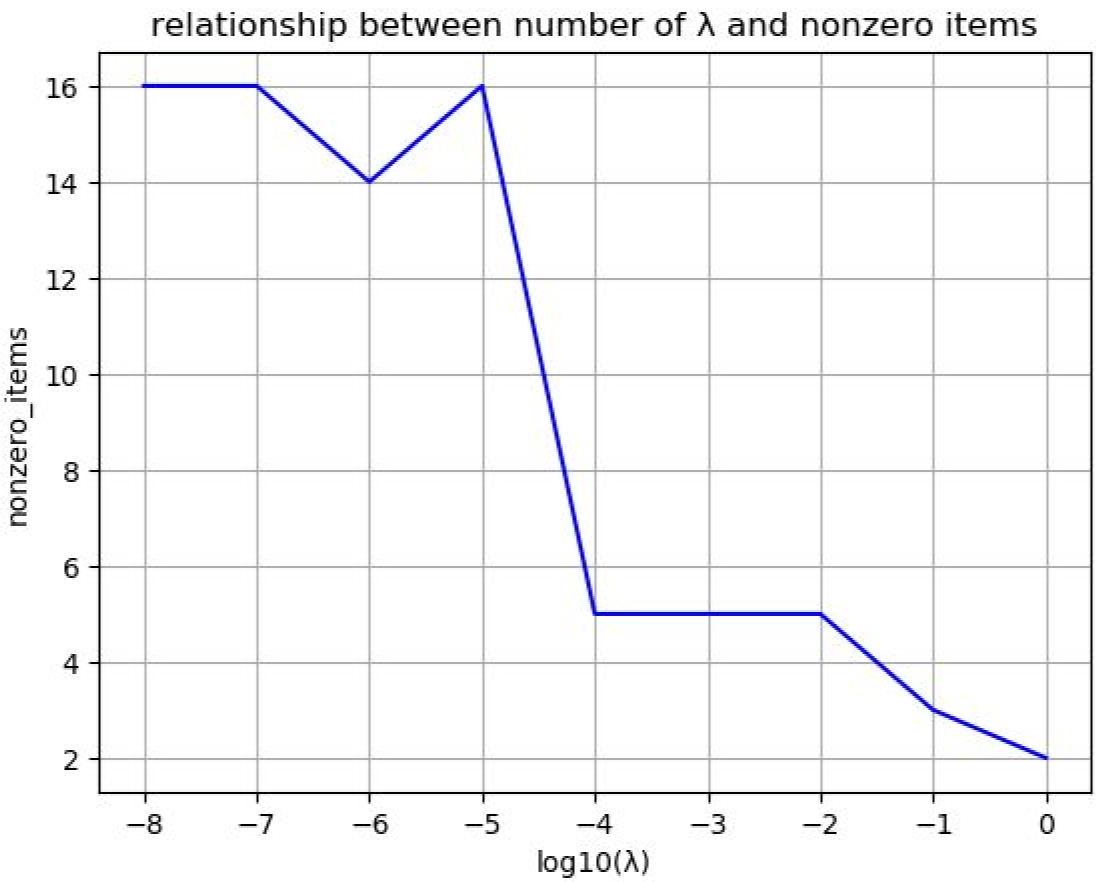
```
#Problem 3c
input = readcsv("lasso_data.csv")
x = input[:, 1]
y = input[:, 2]

k = 15

n = length(x)
A = zeros(n, k+1)
for i = 1:n
    for j = 1:k+1
        A[i,j] = x[i]^(k+1-j)
    end
end

λ_array = []
error_array = []
nonzero_items = []

using JuMP, Gurobi

for λ in [1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001]

    m = Model(solver=GurobiSolver(OutputFlag=0))

    # l1-norm
    @variable(m, t[1:k+1])

    @variable(m, u[1:k+1])


    @constraint(m, u .<= t)
    @constraint(m, -t .<= u)

    @objective(m, Min, sum((y-A*u).^2) + λ*sum(t))

    status = solve(m)
```

```
    uopt = getvalue(u)

    push!(λ_array, log10(λ))
    push!(error_array, sum((y-A*(getvalue(u))).^2))

    nz_items = 0
    # print non-zero items
    for i = 1:k+1
        if abs(uopt[i]) >= 1/(10^5)
            # println(i, " : ", uopt[i])
            nz_items = nz_items + 1
        end
    end

    # println("nonzero items:", nz_items)
    push!(nonzero_items, nz_items)

end

using PyPlot
Plots.default(overwrite_figure=false)
title("λ and error corelation")
plot(λ_array, error_array, "b-")
xlabel("log10(λ)")
ylabel("errors")
grid()
```



In [118]:

```
title("number of λ and nonzero items corelation")
plot(λ_array, nonzero_items, "b-")
xlabel("log10(λ)")
ylabel("nonzero_items")
grid("on")
```



In [125]:

```
# From the two graphs above, we see that when lambda = 10^{-3} we have five nonzero
#items and the error is 1. When lambda gets smaller, the error will not change but
#there will be more nonzero items. When lamba gets bigger, the error will increase
# but there will be less nonzero items. 10^(-3) is a good middle ground. I use this
#value of lambda in the following.
```

In [126]:

```
input = readcsv("lasso_data.csv")
x = input[:, 1]
y = input[:, 2]

k = 15

n = length(x)
A = zeros(n, k+1)
for i = 1:n
    for j = 1:k+1
        A[i,j] = x[i]^(k+1-j)
    end
end

using JuMP, Gurobi

λ = 0.001

m = Model(solver = GurobiSolver(OutputFlag=0))

# l1-norm
@variable(m, t[1:k+1])
@variable(m, u[1:k+1])
@constraint(m, u .<= t)
@constraint(m, -t .<= u)
@objective(m, Min, sum((y-A*u).^2) + λ*sum(t))

status = solve(m)

uopt = getvalue(u)

println(status)
println("uopt: ", uopt)
println("error: ", sum((y-A*(getvalue(u))).^2))

nz_items = 0
# print non-zero items
for i = 1:k+1
    if abs(uopt[i]) >= 1/(10^5)
        println(i, " : ", uopt[i])
        nz_items = nz_items + 1
    else
        println(i, " : ", 0)
    end
end

println("nonzero items:", nz_items)
```

Academic license - for non-commercial use only
Optimal
uopt: [1.50878e-9, 2.97466e-9, 5.82115e-9, 1.12857e-8, 2.16288e-8, 4.09081e-8, 7.65213e-8, 1.43863e-7, 2.88423e-7, 7.57818e-7, 41.1878, 50.3653, 1.51781e-7, -48.5432, 13.0343, 0.119144]
error: 1.020887943014919
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 0
9 : 0
10 : 0
11 : 41.18778432016339
12 : 50.36533671321371
13 : 0
14 : -48.54323558552761
15 : 13.034344853120968
16 : 0.11914408814181378
nonzero items:5