1 Problem 3 Take Home Final Exam

```
# Importing the necessary libraries
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import randn, uniform, seed
from numpy.linalg import norm, solve, matrix_rank, inv
```

For this problem we need to solve: min f(x) = sum(xi log(xi)) over i=1,. . . ,n subject to Ax=b A in R pxn p<n

Define the new functions

```
def f(x):
        return np.dot(np.transpose(x),np.log(x)) # This dot product is equivalent to sur

def grad_f(x):
        return np.log(x) + np.ones((n,1))

def hess_f(x):
        return np.diag((1/x)[:,0])
```

Initialize the problem

```
n = 100
p = 30
```

Choose A randomly

```
seed(999)
A = randn(p,n)
```

Make sure that A is full rank

```
while matrix_rank(A)<p and matrix_rank(A)<n:
        A = randn(p,n)
```

Set the range for the distribution on random x

```
low = 0.0
high = 1.0
x = uniform(low,high,size=(n,1))
```

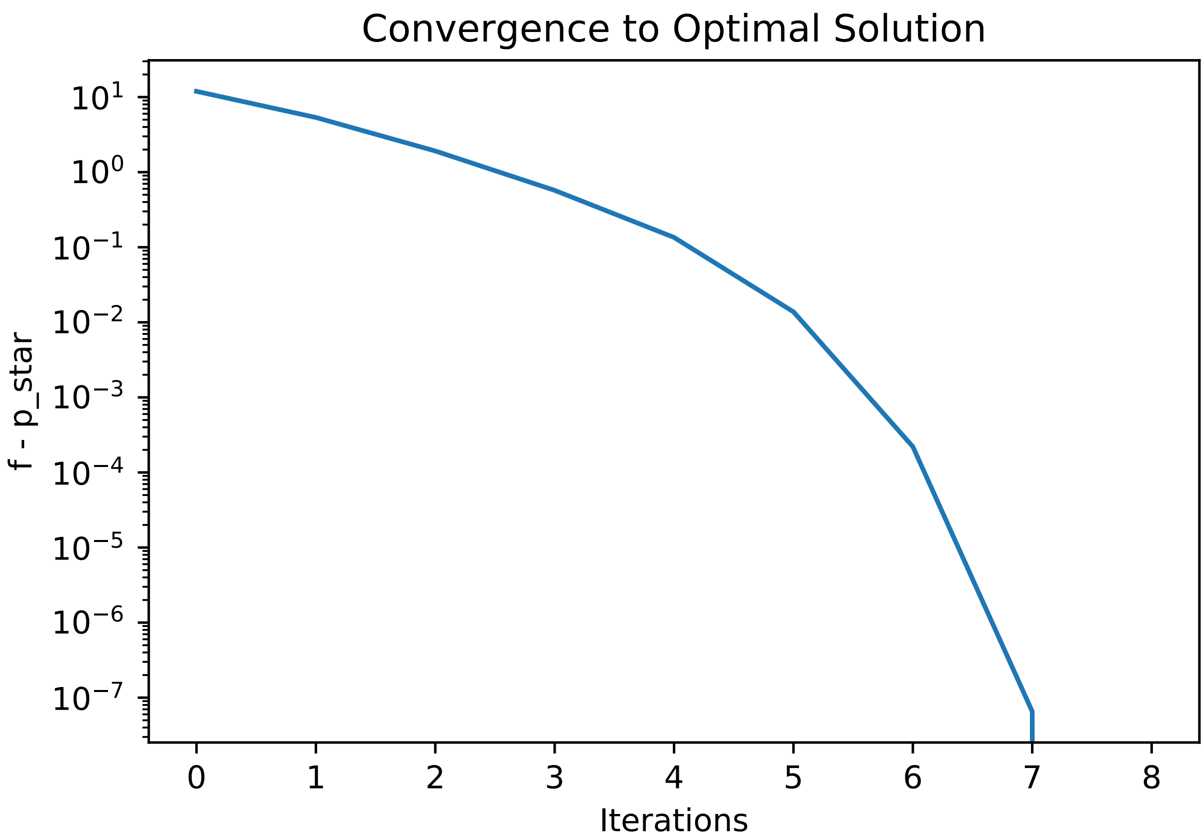
Set b=Axhat

```
b = np.dot(A,x)
```

1.1 Part A - Using the standard newton's method

```
# Backtracking method for updating t
def backtrack(x,f,grad_f,dx,alpha,beta):
        t=1.0
        y = f(x)
        gx = grad_f(x)
        while f(x+t*dx) > y+alpha*t*np.dot(np.transpose(gx),dx):
                t=beta*t
        return t

# Newton Method
# This time the function only returns the optimal point found
def newton_method(x,iterations,alpha,beta,eps):
        # Repeat
        y = np.array([])
        s = np.array([])

        hx = inv(hess_f(x))
        gx = grad_f(x)

        dnt = -np.dot(hx,gx)
        dec = -np.dot(np.transpose(gx),dnt)

        p = f(x)
        y = np.append(y,p)

        for i in range(iterations):

                if dec/2 <= eps:
                        break

                t = backtrack(x,f,grad_f,dnt,alpha,beta)
                s = np.append(s,t)

                x = x + t*dnt

                p = f(x)
                y = np.append(y,p)

                hx = inv(hess_f(x))
                gx = grad_f(x)

                dnt = -np.dot(hx,gx)
                dec = -np.dot(np.transpose(gx),dnt)

        return y,s,p

# Set the parameters for Newton's Method
max_i = 1000
alpha =0.1
beta = 0.25
eps = 1e-8

y,s,p_star = newton_method(x,max_i,alpha,beta,eps)

fig,ax = plt.subplots()
ax.semilogy(np.transpose(y-p_star))
ax.set_title('Convergence to Optimal Solution')
```

```
ax.set_xlabel('Iterations')
ax.set_ylabel('f - p_star')
plt.show()
```



I couldn't get the inseasible Newton method to work on here and debugging via python is terrible. I am going to implement b and c in matlab.