ELEC-E5431- Osman Manzoor Ahmed - 721347 - Assignment 1

January 24, 2019

1 ELEC-E5431 – Large-Scale Data Analysis (LSD Analysis)

Home Work 1 Osman Manzoor Ahmed 721347

In every algorithm the output is as following: 1. X Matrix computed by the algorithm 2. X Optimal Matrix 3. Difference of the above mentioned matrices 4. Convergence rate plot 5. Plot of the X optimal and X created by the algorithm

Kindly scroll down to the ends of each algorithm output in order to see the plots.

```
In [13]: # All Libraries

         import pandas as pd
         from matplotlib import pyplot as plt
         from IPython.display import display, HTML
         import numpy as np
         from scipy import random, linalg
         from sklearn.datasets import make_spd_matrix
         import math
         from random import randint
```

```
In [33]: # ********************************** All Functions********************************

         # Gradient  Ax-b
         def gradient(A,b,x):
             #gradient = np.subtract(np.dot(np.transpose(A),x),b)
             gradient = np.subtract(np.dot(A,x),b)
             #gradient = np.subtract(b,np.dot(np.transpose(A),x))
             return gradient

         # Negative Gradient  b-Ax
         def neg_gradient(A,b,x):
             #gradient = np.subtract(np.dot(np.transpose(A),x),b)
             gradient = np.subtract(b,np.dot(np.transpose(A),x))
             return gradient

         #Error Function log || x_grad - x_opt ||
         def logerror(x_grad, x_opt):
```

```
    diff = x_grad - x_opt
    norm = np.linalg.norm(diff)
    log = np.log(norm)
    return log

# Gradient Descent Method
def gradient_descent(A, b, a, k,x_opt):
    #x = np.zeros((100,1))
    n = len(b)
    x = np.zeros((n,1))
    errors = []

    #for i in range(k):
    while True:
        # Calculate gradient
        grad = gradient(A,b,x)
        # Gradient descent formula
        x = x - a * grad
        #print("In")
        if(np.linalg.norm(grad) <= pow(10,-5)):
           break
        #empirical_errors.append(empirical_risk(X,Y,w))
        errors.append(logerror(x,x_opt))
    return errors,x

# Conjugate Gradient Descent Method
def conjugate_gradient_descent(A, b, a, k,x_opt):
    #x = np.zeros((100,1))
    n = len(b)
    x = np.zeros((n,1))
    p = b
    p_prev = np.zeros((n,1))
    errors = []
    checker = 1
    gamma = 0
    grad = np.ones((n,1))
    i = 0
    r = b
    r_new = b
    #for i in range(k):
    while True:
        # Calculate gradient
        i = i+1
        if(i == 1):
            p = r
            p_prev = p
        else:
            beta = np.divide(np.power(np.linalg.norm(r_new),2),np.power(np.linalg.norm
```

```
            p = np.add(r_new,np.dot(beta,p_prev))
            p_prev = p

        # conjugate Gradient formula
        x = x + a * p
        r = r_new
        r_new = neg_gradient(A,b,x)
        if(np.linalg.norm(r_new) <= pow(10,-5)):
            break
        # Calculate the gamma now
        #print("In")
        errors.append(logerror(x,x_opt))
    return errors,x

# Nesterovs Algorithm
def nesterov_algorithm(A, b, a, L, k,x_opt):
    #x = np.zeros((100,1))
    n = len(b)
    x = np.zeros((n,1))
    errors = []
    y = x
    alpha = a

    #for i in range(k):
    while True:
        # Calculate gradient
        x_prev = x
        grad = gradient(A,b,y)
        x = (np.subtract(y,(1/L)*grad))
        a_prev = alpha
        alpha = (1 + math.sqrt(1 + 4 * a_prev * a_prev))/2

        beta = (a_prev * (1-a_prev)) / (a_prev * a_prev + alpha)

        y = x + (beta * (np.subtract(x,x_prev)))

        if(np.linalg.norm(grad) <= pow(10,-5)):
            break
        # Calculate the gamma now
        #print("In")
        errors.append(logerror(x,x_opt))
    return errors,x

# Stochastic Coordinate Descent Method
def stochastic_coordinate_descent(A, b, a, k,x_opt):
    #x = np.zeros((100,1))
    n = len(b)
    x = np.zeros((n,1))
```

```
    errors = []

    for i in range(k):
    #while True:
        #for j in range(len(x)):
        random = randint(0, n-1)
        # Calculate gradient

        grad = gradient(A[random,random].reshape(1,1),b[random].reshape(1,1),x[random]
        #print(grad.shape)
        # Gradient descent formula
        x[random] = x[random] - a * grad
        #print("In")
        if(np.linalg.norm(grad) <= pow(10,-5)):
            break
        #empirical_errors.append(empirical_risk(X,Y,w))
        errors.append(logerror(x,x_opt))
    return errors
```

In [15]:
```
# Creation of Matrices A, b and X_opt(Optimal x)

# Generate a random symmetric, positive-definite matrix. Size 100*100
matrixSize = 100
A = make_spd_matrix(matrixSize, random_state=None)

# Check if the newly created matrix A is positive- definite. Check if all its Eigneva
if(np.all(np.linalg.eigvals(A) > 0)):
    print("Success, you have a positive definite matrix")

# Generate matrix b size 100*1 that should be in range of Matrix A
x = np.ptp(A,axis = 0)
b = np.reshape(x, (100,1))

# Generate X_Opt as X_opt = A(inverse)b
A_inverse = np.linalg.inv(A)
X_opt = np.dot(A_inverse,b)

#Calculate value of alpha as 1/trace(A)
alpha = 1/A.trace()
```

```
Success, you have a positive definite matrix
```

In [22]:
```
# **************************** Gradient Descent *******************

loggError, x_grad_hat = gradient_descent(A,b,alpha,1000,X_opt)
print("*******************************************************")
print("X Matrix computed by the gradient descent")
```

```
print(x_grad_hat)
print("*****************************************************")
print("X Optimal ")
print(X_opt)
print("*****************************************************")
diff_gradient_descent = np.subtract(x_grad_hat,X_opt)
print("Gradient Descent, Difference between x and x_optimal")
print(diff_gradient_descent)

plt.plot(loggError)
plt.xlabel('Number of Iterations')
plt.ylabel('Log Error')
plt.title('Plot of error according to Iterations(Gradient Descent)')
plt.show()

fig, axes = plt.subplots(1, 2,figsize=(12, 4))
axes[0].plot(X_opt)
axes[0].set_title("X optimal")
axes[1].plot(x_grad_hat,color="red")
axes[1].set_title("X Created by Gradient Descent")
```

```
*****************************************************
X Matrix computed by the gradient descent
[[ 369.85567513]
 [ 347.52930768]
 [ 136.57170757]
 [ 356.56884788]
 [ 182.62946928]
 [ 324.9442824 ]
 [ 336.91040735]
 [ 146.8814646 ]
 [ 149.54641457]
 [ 265.2008635 ]
 [ 296.30396844]
 [ 196.94285936]
 [ 238.76143669]
 [ 245.74076526]
 [ 260.33718727]
 [ 149.88885633]
 [ 251.66869172]
 [ 252.97598286]
 [ 165.60318685]
 [ 201.12797511]
 [ 359.22615672]
 [ 233.59337221]
 [ 298.68877415]
 [ 263.51816722]
 [ 304.47635469]
```

```
[ 128.762478  ]
[ 400.0650543 ]
[ 251.4819183 ]
[ 315.40094681]
[ 258.33566112]
[ 248.87602114]
[ 229.45760242]
[ 254.82573674]
[ 182.66535639]
[ 286.18313018]
[ 220.62734832]
[ 155.46716268]
[ 263.00827141]
[ 416.08642763]
[ 393.30587325]
[ 179.49759759]
[ 191.3877822 ]
[ 208.3530741 ]
[ 294.35507129]
[ 205.79417736]
[ 119.27445813]
[ 140.07654337]
[ 260.29020516]
[ 232.48418375]
[ 290.12131115]
[ 272.84196638]
[ 331.71504429]
[ 118.21582202]
[ 208.09063269]
[  69.65789716]
[ 373.98026167]
[ 262.73466051]
[ 245.69804421]
[ 248.9715473 ]
[ 255.64950244]
[ 382.59159265]
[ 226.58587274]
[ 224.49243135]
[ 301.55041196]
[ 260.21281628]
[ 348.06374709]
[ 234.92285771]
[ 200.07346343]
[ 243.29980187]
[ 265.06142408]
[ 212.87269534]
[ 275.81240103]
[ 434.9951508 ]
```

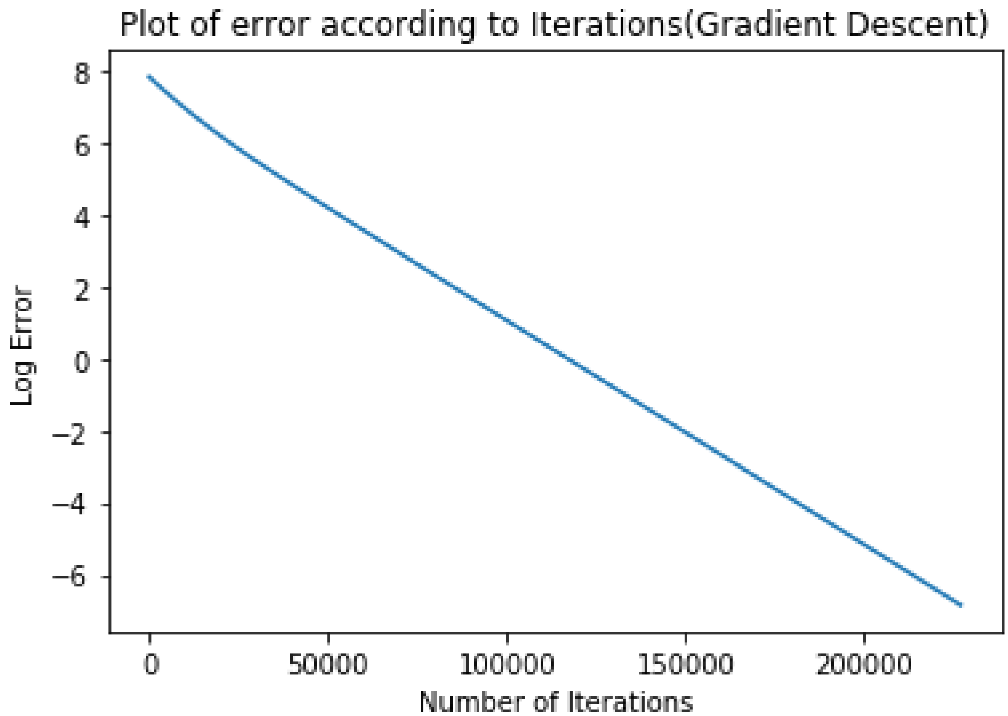
```
 [ 276.08307149]
 [ 174.0378495 ]
 [ 115.60791464]
 [ 316.65695918]
 [ 377.40613234]
 [ 246.29480932]
 [  66.22641317]
 [ 211.64928287]
 [ 274.89661135]
 [ 109.06310882]
 [ 253.37626987]
 [ 111.13063145]
 [ 245.42343474]
 [ 248.70690208]
 [ 405.74373867]
 [ 174.96807642]
 [ 169.88798413]
 [ 340.93058158]
 [ 140.73008856]
 [ 292.47183355]
 [ 170.04097507]
 [ 216.11839236]
 [ 136.5295712 ]
 [ 221.46488959]
 [ 181.15791618]
 [ 183.0397369 ]
 [ 132.62015931]]
*******************************************************
X Optimal
[[ 369.85581495]
 [ 347.52950865]
 [ 136.57162674]
 [ 356.56905223]
 [ 182.62952925]
 [ 324.94440216]
 [ 336.91052161]
 [ 146.8814103 ]
 [ 149.54627688]
 [ 265.20097177]
 [ 296.30412266]
 [ 196.94275281]
 [ 238.7614395 ]
 [ 245.74075895]
 [ 260.33719987]
 [ 149.88884021]
 [ 251.6686562 ]
 [ 252.97604413]
 [ 165.60304565]
```

```
[ 201.12811188]
[ 359.22639321]
[ 233.59343212]
[ 298.68888429]
[ 263.51823862]
[ 304.47647401]
[ 128.7623713 ]
[ 400.06517907]
[ 251.48198638]
[ 315.40107588]
[ 258.33580537]
[ 248.87600913]
[ 229.4576329 ]
[ 254.82576885]
[ 182.66527341]
[ 286.18329361]
[ 220.62736537]
[ 155.4671711 ]
[ 263.00829586]
[ 416.08668276]
[ 393.30606556]
[ 179.49753147]
[ 191.38766643]
[ 208.35303824]
[ 294.35515373]
[ 205.7941958 ]
[ 119.27434083]
[ 140.07653215]
[ 260.2903277 ]
[ 232.48427151]
[ 290.12139389]
[ 272.84202705]
[ 331.71523883]
[ 118.21571852]
[ 208.09059764]
[  69.65781271]
[ 373.98040621]
[ 262.73477905]
[ 245.69802444]
[ 248.97156119]
[ 255.64956392]
[ 382.59180357]
[ 226.58595732]
[ 224.49246644]
[ 301.55041851]
[ 260.21281363]
[ 348.06400507]
[ 234.92297486]
```

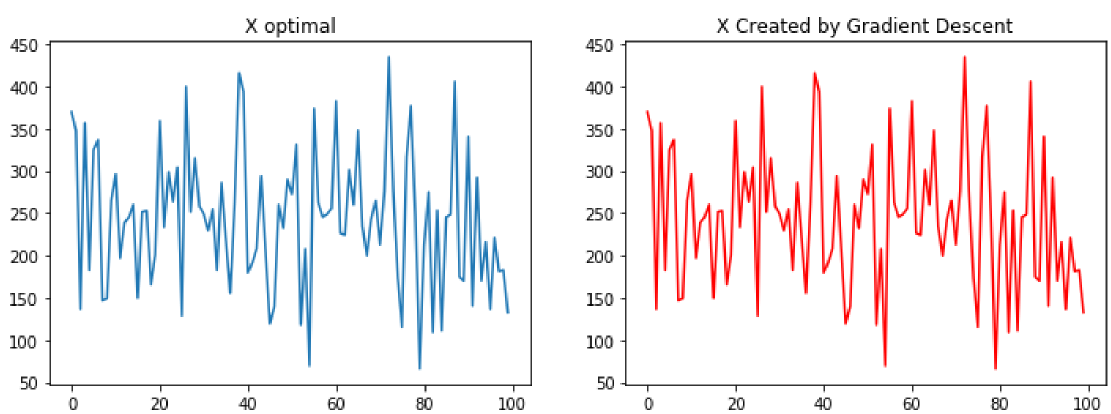
```
 [ 200.07341814]
 [ 243.29985356]
 [ 265.06143092]
 [ 212.87271981]
 [ 275.81254831]
 [ 434.99527478]
 [ 276.08308824]
 [ 174.03785089]
 [ 115.60776748]
 [ 316.65703   ]
 [ 377.40630082]
 [ 246.29490277]
 [  66.22623571]
 [ 211.64923808]
 [ 274.89664475]
 [ 109.06304026]
 [ 253.37631658]
 [ 111.13055017]
 [ 245.42350498]
 [ 248.70698019]
 [ 405.74388454]
 [ 174.96808497]
 [ 169.88801461]
 [ 340.93071288]
 [ 140.7299729 ]
 [ 292.47192549]
 [ 170.04092855]
 [ 216.11842525]
 [ 136.52941403]
 [ 221.46491147]
 [ 181.15777176]
 [ 183.03972682]
 [ 132.62009973]]
*****************************************************
Gradient Descent, Difference between x and x_optimal
[[ -1.39825909e-04]
 [ -2.00970874e-04]
 [  8.08232216e-05]
 [ -2.04343045e-04]
 [ -5.99715419e-05]
 [ -1.19762725e-04]
 [ -1.14256468e-04]
 [  5.43058867e-05]
 [  1.37690480e-04]
 [ -1.08269831e-04]
 [ -1.54220086e-04]
 [  1.06552806e-04]
 [ -2.81164117e-06]
```

```
[  6.31585988e-06]
[ -1.26050077e-05]
[  1.61159846e-05]
[  3.55188249e-05]
[ -6.12698785e-05]
[  1.41195418e-04]
[ -1.36771963e-04]
[ -2.36485385e-04]
[ -5.99151101e-05]
[ -1.10139683e-04]
[ -7.14059266e-05]
[ -1.19321766e-04]
[  1.06702162e-04]
[ -1.24768547e-04]
[ -6.80853684e-05]
[ -1.29061876e-04]
[ -1.44254470e-04]
[  1.20058944e-05]
[ -3.04726633e-05]
[ -3.21043506e-05]
[  8.29822380e-05]
[ -1.63429805e-04]
[ -1.70520302e-05]
[ -8.41348182e-06]
[ -2.44448347e-05]
[ -2.55134729e-04]
[ -1.92310498e-04]
[  6.61157455e-05]
[  1.15766105e-04]
[  3.58614275e-05]
[ -8.24373026e-05]
[ -1.84437452e-05]
[  1.17295098e-04]
[  1.12259495e-05]
[ -1.22537866e-04]
[ -8.77670906e-05]
[ -8.27447987e-05]
[ -6.06701343e-05]
[ -1.94540928e-04]
[  1.03500299e-04]
[  3.50509432e-05]
[  8.44474770e-05]
[ -1.44541636e-04]
[ -1.18531812e-04]
[  1.97703876e-05]
[ -1.38875372e-05]
[ -6.14771909e-05]
[ -2.10916721e-04]
```

```
[ -8.45846605e-05]
[ -3.50899426e-05]
[ -6.55055260e-06]
[  2.64584730e-06]
[ -2.57979155e-04]
[ -1.17154805e-04]
[  4.52892141e-05]
[ -5.16840965e-05]
[ -6.83995961e-06]
[ -2.44683323e-05]
[ -1.47274214e-04]
[ -1.23982026e-04]
[ -1.67549165e-05]
[ -1.38273734e-06]
[  1.47162065e-04]
[ -7.08230914e-05]
[ -1.68483116e-04]
[ -9.34561160e-05]
[  1.77452403e-04]
[  4.47919521e-05]
[ -3.34021680e-05]
[  6.85622598e-05]
[ -4.67112910e-05]
[  8.12809905e-05]
[ -7.02390475e-05]
[ -7.81142984e-05]
[ -1.45871825e-04]
[ -8.55273097e-06]
[ -3.04802504e-05]
[ -1.31300500e-04]
[  1.15655419e-04]
[ -9.19348385e-05]
[  4.65203871e-05]
[ -3.28896191e-05]
[  1.57165716e-04]
[ -2.18737046e-05]
[  1.44414470e-04]
[  1.00795854e-05]
[  5.95730237e-05]]
```



Out[22]: Text(0.5,1,'X Created by Gradient Descent')



In [21]:
```
#***************************** Conjugate Gradient Descent ************

logeError,x_conj_grad_hat = conjugate_gradient_descent(A,b,alpha,10000,X_opt)
print("*****************************************************")
print("X Matrix computed by the conjugate gradient descent")
print(x_conj_grad_hat)
```

```
print("****************************************************")
print("X Optimal ")
print(X_opt)
print("****************************************************")
diff_conj_gradient_descent = np.subtract(x_conj_grad_hat,X_opt)
print("Conjugate Gradient Descent, Difference between x and x_optimal")
print(diff_conj_gradient_descent)

plt.plot(logeError)
plt.xlabel('Number of Iterations')
plt.ylabel('Log Error')
plt.title('Plot of error according to Iterations(Conjugate Gradient Descent)')
plt.show()


fig, axes = plt.subplots(1, 2,figsize=(12, 4))
axes[0].plot(X_opt)
axes[0].set_title("X optimal")
axes[1].plot(x_conj_grad_hat,color="red")
axes[1].set_title("X Created by Conjugate Gradient Descent")
```

```
****************************************************
X Matrix computed by the conjugate gradient descent
[[ 369.8557663 ]
 [ 347.5293881 ]
 [ 136.57171832]
 [ 356.56893369]
 [ 182.6294939 ]
 [ 324.94436463]
 [ 336.91047443]
 [ 146.88149111]
 [ 149.54643371]
 [ 265.20092065]
 [ 296.30404067]
 [ 196.94288516]
 [ 238.76149388]
 [ 245.74081314]
 [ 260.33723697]
 [ 149.88887617]
 [ 251.66872988]
 [ 252.97603756]
 [ 165.60321468]
 [ 201.12800814]
 [ 359.22625092]
 [ 233.59342716]
 [ 298.68884118]
 [ 263.51821643]
 [ 304.47641876]
```

```
[ 128.7624859 ]
[ 400.0651427 ]
[ 251.48197567]
[ 315.40101215]
[ 258.33572562]
[ 248.87607518]
[ 229.45765191]
[ 254.82579783]
[ 182.66537858]
[ 286.18318651]
[ 220.62738647]
[ 155.46718181]
[ 263.00832238]
[ 416.08652611]
[ 393.30596519]
[ 179.49762724]
[ 191.3878051 ]
[ 208.35310981]
[ 294.3551371 ]
[ 205.79420876]
[ 119.27447747]
[ 140.07655498]
[ 260.29025494]
[ 232.4842334 ]
[ 290.12138266]
[ 272.84201922]
[ 331.71512619]
[ 118.21581754]
[ 208.09067189]
[  69.65788448]
[ 373.98033914]
[ 262.73470448]
[ 245.69809598]
[ 248.9715964 ]
[ 255.64955969]
[ 382.59168718]
[ 226.58591913]
[ 224.49246872]
[ 301.55046311]
[ 260.21286446]
[ 348.0638271 ]
[ 234.92290429]
[ 200.07350332]
[ 243.29985805]
[ 265.06148432]
[ 212.8727228 ]
[ 275.81246286]
[ 434.99524465]
```

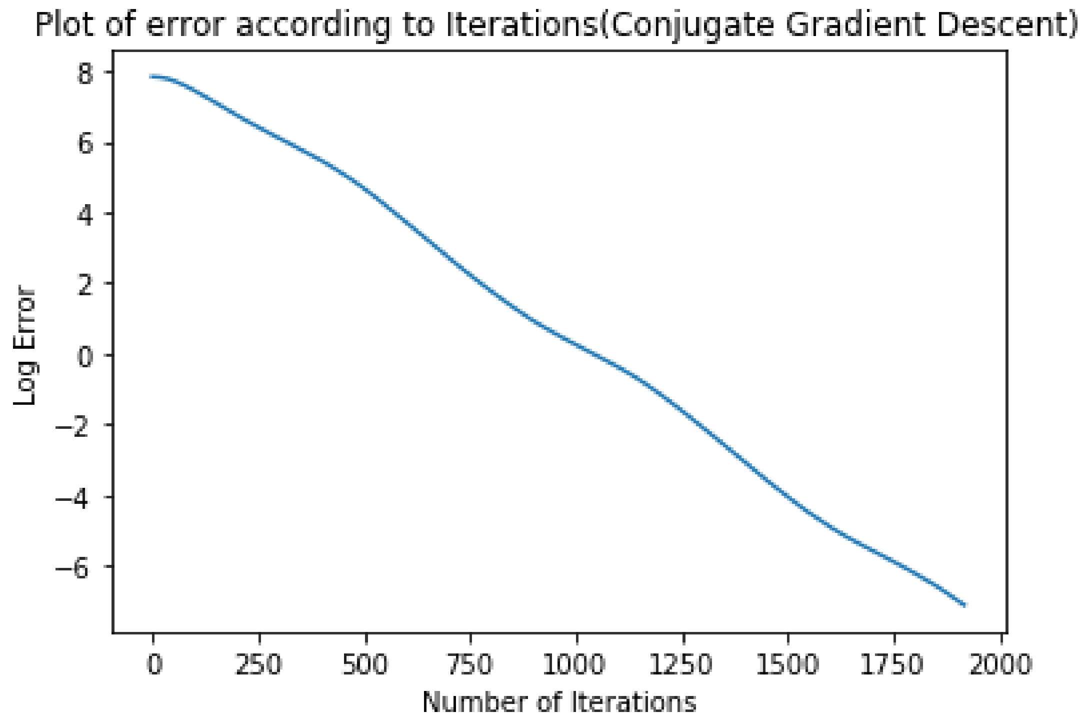
```
 [ 276.08313168]
 [ 174.03787743]
 [ 115.60792818]
 [ 316.65703202]
 [ 377.40622025]
 [ 246.29486033]
 [  66.22640707]
 [ 211.64932752]
 [ 274.89665632]
 [ 109.06311258]
 [ 253.37632304]
 [ 111.13062988]
 [ 245.42348768]
 [ 248.70694681]
 [ 405.74383365]
 [ 174.96809866]
 [ 169.88801723]
 [ 340.93065344]
 [ 140.73010361]
 [ 292.47189662]
 [ 170.04100266]
 [ 216.11843486]
 [ 136.52960224]
 [ 221.46492791]
 [ 181.15794183]
 [ 183.03976737]
 [ 132.62016994]]
*****************************************************
X Optimal
[[ 369.85581495]
 [ 347.52950865]
 [ 136.57162674]
 [ 356.56905223]
 [ 182.62952925]
 [ 324.94440216]
 [ 336.91052161]
 [ 146.8814103 ]
 [ 149.54627688]
 [ 265.20097177]
 [ 296.30412266]
 [ 196.94275281]
 [ 238.7614395 ]
 [ 245.74075895]
 [ 260.33719987]
 [ 149.88884021]
 [ 251.6686562 ]
 [ 252.97604413]
 [ 165.60304565]
```

```
[ 201.12811188]
[ 359.22639321]
[ 233.59343212]
[ 298.68888429]
[ 263.51823862]
[ 304.47647401]
[ 128.7623713 ]
[ 400.06517907]
[ 251.48198638]
[ 315.40107588]
[ 258.33580537]
[ 248.87600913]
[ 229.4576329 ]
[ 254.82576885]
[ 182.66527341]
[ 286.18329361]
[ 220.62736537]
[ 155.4671711 ]
[ 263.00829586]
[ 416.08668276]
[ 393.30606556]
[ 179.49753147]
[ 191.38766643]
[ 208.35303824]
[ 294.35515373]
[ 205.7941958 ]
[ 119.27434083]
[ 140.07653215]
[ 260.2903277 ]
[ 232.48427151]
[ 290.12139389]
[ 272.84202705]
[ 331.71523883]
[ 118.21571852]
[ 208.09059764]
[  69.65781271]
[ 373.98040621]
[ 262.73477905]
[ 245.69802444]
[ 248.97156119]
[ 255.64956392]
[ 382.59180357]
[ 226.58595732]
[ 224.49246644]
[ 301.55041851]
[ 260.21281363]
[ 348.06400507]
[ 234.92297486]
```

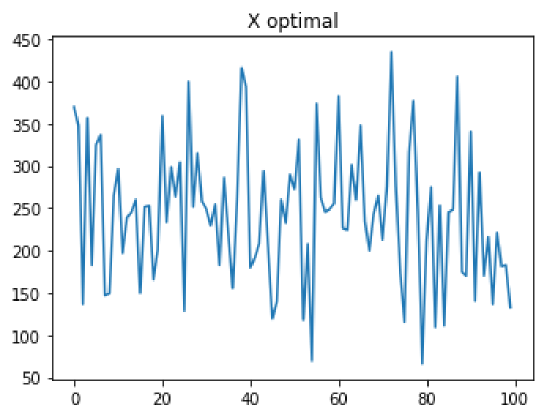
```
 [ 200.07341814]
 [ 243.29985356]
 [ 265.06143092]
 [ 212.87271981]
 [ 275.81254831]
 [ 434.99527478]
 [ 276.08308824]
 [ 174.03785089]
 [ 115.60776748]
 [ 316.65703   ]
 [ 377.40630082]
 [ 246.29490277]
 [  66.22623571]
 [ 211.64923808]
 [ 274.89664475]
 [ 109.06304026]
 [ 253.37631658]
 [ 111.13055017]
 [ 245.42350498]
 [ 248.70698019]
 [ 405.74388454]
 [ 174.96808497]
 [ 169.88801461]
 [ 340.93071288]
 [ 140.7299729 ]
 [ 292.47192549]
 [ 170.04092855]
 [ 216.11842525]
 [ 136.52941403]
 [ 221.46491147]
 [ 181.15777176]
 [ 183.03972682]
 [ 132.62009973]]
*******************************************************
Conjugate Gradient Descent, Difference between x and x_optimal
[[ -4.86553316e-05]
 [ -1.20551458e-04]
 [  9.15788362e-05]
 [ -1.18536082e-04]
 [ -3.53566042e-05]
 [ -3.75316243e-05]
 [ -4.71776352e-05]
 [  8.08123762e-05]
 [  1.56832388e-04]
 [ -5.11175271e-05]
 [ -8.19884876e-05]
 [  1.32346819e-04]
 [  5.43838688e-05]
```

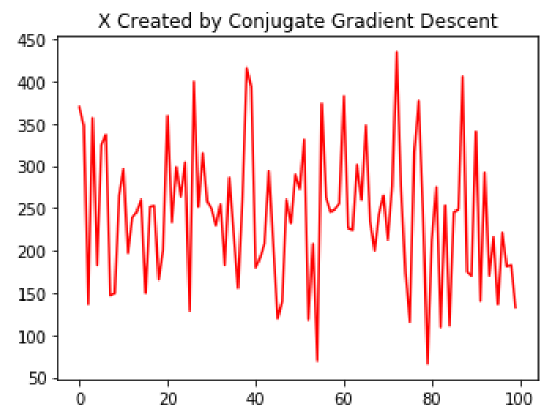
```
[  5.41922167e-05]
[  3.70984940e-05]
[  3.59629918e-05]
[  7.36777574e-05]
[ -6.56913247e-06]
[  1.69022949e-04]
[ -1.03741460e-04]
[ -1.42289934e-04]
[ -4.96676196e-06]
[ -4.31128158e-05]
[ -2.21922047e-05]
[ -5.52574916e-05]
[  1.14602511e-04]
[ -3.63736509e-05]
[ -1.07123665e-05]
[ -6.37258567e-05]
[ -7.97580397e-05]
[  6.60519044e-05]
[  1.90127979e-05]
[  2.89881725e-05]
[  1.05171925e-04]
[ -1.07091389e-04]
[  2.10966390e-05]
[  1.07190066e-05]
[  2.65231076e-05]
[ -1.56657286e-04]
[ -1.00365492e-04]
[  9.57677768e-05]
[  1.38664774e-04]
[  7.15708208e-05]
[ -1.66266809e-05]
[  1.29651053e-05]
[  1.36642407e-04]
[  2.28352164e-05]
[ -7.27580455e-05]
[ -3.81144012e-05]
[ -1.12288411e-05]
[ -7.82043617e-06]
[ -1.12644087e-04]
[  9.90206041e-05]
[  7.42449759e-05]
[  7.17654278e-05]
[ -6.70692689e-05]
[ -7.45690407e-05]
[  7.15365498e-05]
[  3.52114274e-05]
[ -4.22733231e-06]
[ -1.16384926e-04]
```

```
[ -3.81906661e-05]
[  2.28640226e-06]
[  4.45992918e-05]
[  5.08304290e-05]
[ -1.77964949e-04]
[ -7.05692823e-05]
[  8.51842744e-05]
[  4.49760711e-06]
[  5.34011656e-05]
[  2.98732513e-06]
[ -8.54501853e-05]
[ -3.01391324e-05]
[  4.34371212e-05]
[  2.65443306e-05]
[  1.60705618e-04]
[  2.02081708e-06]
[ -8.05740888e-05]
[ -4.24385064e-05]
[  1.71354294e-04]
[  8.94384019e-05]
[  1.15690636e-05]
[  7.23202314e-05]
[  6.45806966e-06]
[  7.97126707e-05]
[ -1.73036263e-05]
[ -3.33855544e-05]
[ -5.08963743e-05]
[  1.36925319e-05]
[  2.61168506e-06]
[ -5.94362040e-05]
[  1.30704754e-04]
[ -2.88679851e-05]
[  7.41100615e-05]
[  9.60964752e-06]
[  1.88204336e-04]
[  1.64378910e-05]
[  1.70068308e-04]
[  4.05499358e-05]
[  7.02089793e-05]]
```



Out[21]: Text(0.5,1,'X Created by Conjugate Gradient Descent')





In [28]:
```
#****************************** Nesterovs Algorithm ************

logError,x_nest_algo = nesterov_algorithm(A,b,0,A.trace(),10000,X_opt)
print("*******************************************************")
print("X Matrix computed by the Nesterov Algorithm")
print(x_nest_algo)
```

```python
    print("*****************************************************")
    print("X Optimal ")
    print(X_opt)
    print("*****************************************************")
    diff_nest_algo = np.subtract(x_nest_algo,X_opt)
    print("Nesterov Algorithm, Difference between x and x_optimal")
    print(diff_nest_algo)

    plt.plot(logError)
    plt.xlabel('Number of Iterations')
    plt.ylabel('Log Error')
    plt.title('Plot of error according to Iterations(Nesterov Method)')
    plt.show()

    fig, axes = plt.subplots(1, 2,figsize=(12, 4))
    axes[0].plot(X_opt)
    axes[0].set_title("X optimal")
    axes[1].plot(x_nest_algo,color="red")
    axes[1].set_title("X Created by Nesterov Algorithm")
```

```
*****************************************************
X Matrix computed by the Nesterov Algorithm
[[ 369.85567513]
 [ 347.52930769]
 [ 136.57170756]
 [ 356.56884789]
 [ 182.62946928]
 [ 324.9442824 ]
 [ 336.91040735]
 [ 146.8814646 ]
 [ 149.54641457]
 [ 265.2008635 ]
 [ 296.30396844]
 [ 196.94285936]
 [ 238.76143669]
 [ 245.74076526]
 [ 260.33718727]
 [ 149.88885633]
 [ 251.66869172]
 [ 252.97598287]
 [ 165.60318685]
 [ 201.12797511]
 [ 359.22615673]
 [ 233.59337221]
 [ 298.68877416]
 [ 263.51816722]
 [ 304.4763547 ]
 [ 128.762478  ]
```

```
[ 400.06505431]
[ 251.4819183 ]
[ 315.40094682]
[ 258.33566112]
[ 248.87602114]
[ 229.45760242]
[ 254.82573674]
[ 182.66535639]
[ 286.18313018]
[ 220.62734832]
[ 155.46716268]
[ 263.00827141]
[ 416.08642763]
[ 393.30587325]
[ 179.49759758]
[ 191.3877822 ]
[ 208.3530741 ]
[ 294.35507129]
[ 205.79417736]
[ 119.27445812]
[ 140.07654337]
[ 260.29020516]
[ 232.48418375]
[ 290.12131115]
[ 272.84196638]
[ 331.71504429]
[ 118.21582201]
[ 208.09063269]
[  69.65789716]
[ 373.98026167]
[ 262.73466052]
[ 245.69804421]
[ 248.9715473 ]
[ 255.64950244]
[ 382.59159265]
[ 226.58587274]
[ 224.49243135]
[ 301.55041196]
[ 260.21281628]
[ 348.06374709]
[ 234.92285771]
[ 200.07346342]
[ 243.29980187]
[ 265.06142408]
[ 212.87269534]
[ 275.81240103]
[ 434.99515081]
[ 276.08307149]
```

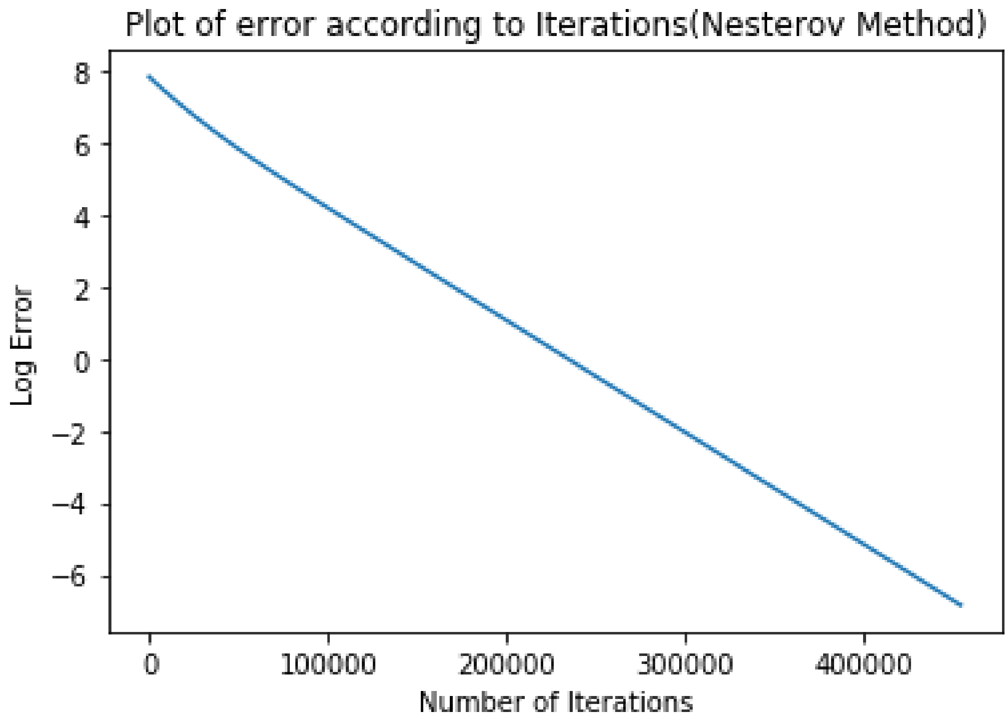
```
 [ 174.0378495 ]
 [ 115.60791464]
 [ 316.65695918]
 [ 377.40613234]
 [ 246.29480932]
 [  66.22641316]
 [ 211.64928287]
 [ 274.89661135]
 [ 109.06310882]
 [ 253.37626987]
 [ 111.13063145]
 [ 245.42343475]
 [ 248.70690208]
 [ 405.74373867]
 [ 174.96807642]
 [ 169.88798413]
 [ 340.93058158]
 [ 140.73008856]
 [ 292.47183356]
 [ 170.04097506]
 [ 216.11839236]
 [ 136.5295712 ]
 [ 221.46488959]
 [ 181.15791617]
 [ 183.0397369 ]
 [ 132.6201593 ]]
*******************************************************
X Optimal
[[ 369.85581495]
 [ 347.52950865]
 [ 136.57162674]
 [ 356.56905223]
 [ 182.62952925]
 [ 324.94440216]
 [ 336.91052161]
 [ 146.8814103 ]
 [ 149.54627688]
 [ 265.20097177]
 [ 296.30412266]
 [ 196.94275281]
 [ 238.7614395 ]
 [ 245.74075895]
 [ 260.33719987]
 [ 149.88884021]
 [ 251.6686562 ]
 [ 252.97604413]
 [ 165.60304565]
 [ 201.12811188]
```

```
[ 359.22639321]
[ 233.59343212]
[ 298.68888429]
[ 263.51823862]
[ 304.47647401]
[ 128.7623713 ]
[ 400.06517907]
[ 251.48198638]
[ 315.40107588]
[ 258.33580537]
[ 248.87600913]
[ 229.4576329 ]
[ 254.82576885]
[ 182.66527341]
[ 286.18329361]
[ 220.62736537]
[ 155.4671711 ]
[ 263.00829586]
[ 416.08668276]
[ 393.30606556]
[ 179.49753147]
[ 191.38766643]
[ 208.35303824]
[ 294.35515373]
[ 205.7941958 ]
[ 119.27434083]
[ 140.07653215]
[ 260.2903277 ]
[ 232.48427151]
[ 290.12139389]
[ 272.84202705]
[ 331.71523883]
[ 118.21571852]
[ 208.09059764]
[  69.65781271]
[ 373.98040621]
[ 262.73477905]
[ 245.69802444]
[ 248.97156119]
[ 255.64956392]
[ 382.59180357]
[ 226.58595732]
[ 224.49246644]
[ 301.55041851]
[ 260.21281363]
[ 348.06400507]
[ 234.92297486]
[ 200.07341814]
```

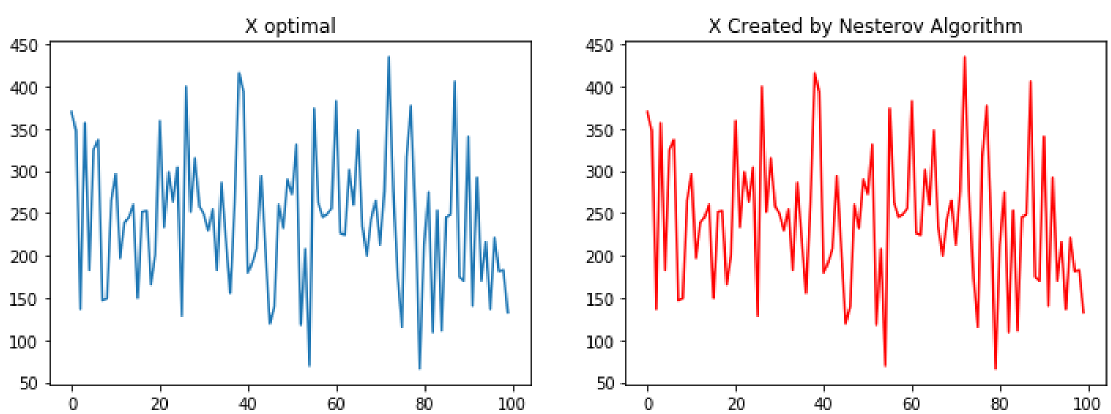
```
 [ 243.29985356]
 [ 265.06143092]
 [ 212.87271981]
 [ 275.81254831]
 [ 434.99527478]
 [ 276.08308824]
 [ 174.03785089]
 [ 115.60776748]
 [ 316.65703   ]
 [ 377.40630082]
 [ 246.29490277]
 [  66.22623571]
 [ 211.64923808]
 [ 274.89664475]
 [ 109.06304026]
 [ 253.37631658]
 [ 111.13055017]
 [ 245.42350498]
 [ 248.70698019]
 [ 405.74388454]
 [ 174.96808497]
 [ 169.88801461]
 [ 340.93071288]
 [ 140.7299729 ]
 [ 292.47192549]
 [ 170.04092855]
 [ 216.11842525]
 [ 136.52941403]
 [ 221.46491147]
 [ 181.15777176]
 [ 183.03972682]
 [ 132.62009973]]
*****************************************************
Nesterov Algorithm, Difference between x and x_optimal
[[ -1.39822990e-04]
 [ -2.00966697e-04]
 [  8.08215417e-05]
 [ -2.04338794e-04]
 [ -5.99702826e-05]
 [ -1.19760221e-04]
 [ -1.14254071e-04]
 [  5.43047510e-05]
 [  1.37687619e-04]
 [ -1.08267561e-04]
 [ -1.54216876e-04]
 [  1.06550591e-04]
 [ -2.81158785e-06]
 [  6.31573275e-06]
```

```
[ -1.26047557e-05]
[  1.61156554e-05]
[  3.55180890e-05]
[ -6.12686081e-05]
[  1.41192486e-04]
[ -1.36769117e-04]
[ -2.36480475e-04]
[ -5.99138668e-05]
[ -1.10137374e-04]
[ -7.14044448e-05]
[ -1.19319272e-04]
[  1.06699948e-04]
[ -1.24765946e-04]
[ -6.80839544e-05]
[ -1.29059184e-04]
[ -1.44251467e-04]
[  1.20056414e-05]
[ -3.04720313e-05]
[ -3.21036906e-05]
[  8.29805142e-05]
[ -1.63426397e-04]
[ -1.70516709e-05]
[ -8.41331141e-06]
[ -2.44443293e-05]
[ -2.55129430e-04]
[ -1.92306502e-04]
[  6.61143646e-05]
[  1.15763687e-04]
[  3.58606849e-05]
[ -8.24355965e-05]
[ -1.84433535e-05]
[  1.17292655e-04]
[  1.12257104e-05]
[ -1.22535307e-04]
[ -8.77652764e-05]
[ -8.27431006e-05]
[ -6.06688772e-05]
[ -1.94536880e-04]
[  1.03498154e-04]
[  3.50502208e-05]
[  8.44457070e-05]
[ -1.44538620e-04]
[ -1.18529344e-04]
[  1.97699767e-05]
[ -1.38872489e-05]
[ -6.14759028e-05]
[ -2.10912338e-04]
[ -8.45829030e-05]
```

```
[ -3.50892103e-05]
[ -6.55041021e-06]
[  2.64579046e-06]
[ -2.57973797e-04]
[ -1.17152373e-04]
[  4.52882703e-05]
[ -5.16830193e-05]
[ -6.83981682e-06]
[ -2.44678237e-05]
[ -1.47271140e-04]
[ -1.23979431e-04]
[ -1.67545652e-05]
[ -1.38270238e-06]
[  1.47159012e-04]
[ -7.08216267e-05]
[ -1.68479603e-04]
[ -9.34541804e-05]
[  1.77448707e-04]
[  4.47910286e-05]
[ -3.34014888e-05]
[  6.85608332e-05]
[ -4.67103220e-05]
[  8.12792864e-05]
[ -7.02375927e-05]
[ -7.81126704e-05]
[ -1.45868794e-04]
[ -8.55255911e-06]
[ -3.04796234e-05]
[ -1.31297762e-04]
[  1.15653014e-04]
[ -9.19329279e-05]
[  4.65194192e-05]
[ -3.28889340e-05]
[  1.57162436e-04]
[ -2.18732585e-05]
[  1.44411471e-04]
[  1.00793774e-05]
[  5.95717718e-05]]
```



Out[28]: Text(0.5,1,'X Created by Nesterov Algorithm')




```
In [34]: #***************************** Stochastic Coordinate Descent Algorithm ***********

         logcoorError = stochastic_coordinate_descent(A,b,alpha,50000,X_opt)
         #print(len(logError))
         #print("Gradient X")
         #print(x_grad)
```

```
#print(x_grad.shape)

plt.plot(logcoorError)
plt.xlabel('Number of Iterations')
plt.ylabel('Log Error')
plt.title('Plot of error according to Iterations(Stochastic Coordinate Descent Method)
plt.show()
```