Polynomial Regression program to solve for the spring mass system

```
import numpy as np
import matplotlib.pyplot as plt
```

Data generation for values of x and y

Displacement is calculated using A*sin(wt) as it represents a harmonic motion where omega(w) = (k/m)^0.5 and calculated for different values of t

Stiffness(k) = 12 N/m

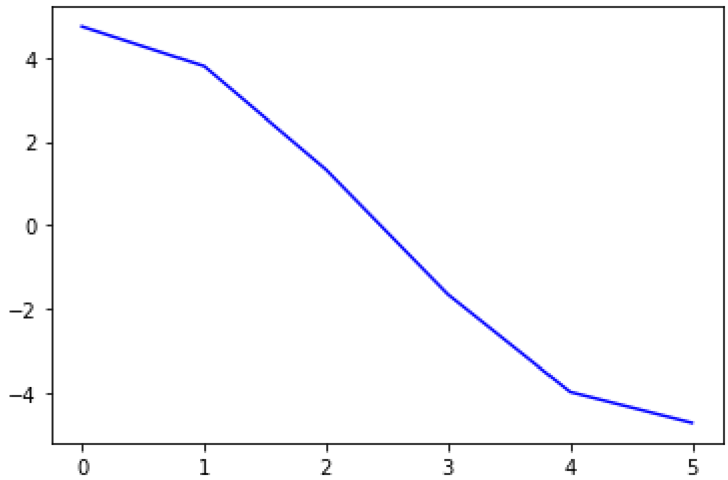
Mass(m) = 29 Kg

Amplitude(A) = 5 m

```
# Plotting the actual values

x = np.array( [[0], [1], [2], [3], [4], [5]] ) #x denotes the values of time
y_actual = np.array([5.000, 4.001, 1.404, -1.752, -4.210, -4.986]) #y_actual denotes th
y = np.array( [4.75, 3.80, 1.33, -1.66, -3.99, -4.73] ) #y denotes the values of displa
plt.plot(x,y, color="blue")
```

```
[<matplotlib.lines.Line2D at 0x7f50c7f5df60>]
```




```
class function():

 def __init__(self, exp, alpha, it):   #self is an instance for the class regression
        self.exp = exp  #Number of exponents
        self.it = it #Number of iterations
        self.alpha = alpha #learning rate

 def function_a(self, x):  #polynomial basis function
        x_matrix = np.ones((self.m, 1))
        for k in range(1, self.exp):
                x_exp = np.power(x, k) #transforming x in the form of polynomial for th
                x_matrix = np.append( x_matrix, x_exp.reshape( -1, 1 ), axis = 1 ) #app
                x_matrix[:,1:] = ( x_matrix[:,1:] - np.mean( x_matrix[:,1:], axis = 0 )
```

```
        return x_matrix

    def function_b(self, x, y): #gradient descent function
        self.x = x
        self.m, self.n = self.x.shape
        self.y = y
        x_matrix = self.function_a(self.x) #matrix transformation
        self.w = np.zeros(self.exp)
        for i in range(self.it):
            y1 = self.function_c(self.x) #predicted value of y
            E = y1 - self.y #error between the actual and predicted values of y
            self.w = self.w - self.alpha * (1 /self.m) * np.dot(x_matrix.T, E) #gradien
        return self

    def function_c(self, x): #value prediction
        x_matrix = self.function_a(x)
        return np.dot(x_matrix, self.w)
```

```
result = function(4, 0.1, 50) #No.of exponents in the polynomial taken = 4
result.function_b(x, y)
y_new = result.function_c(x)
y_new
```

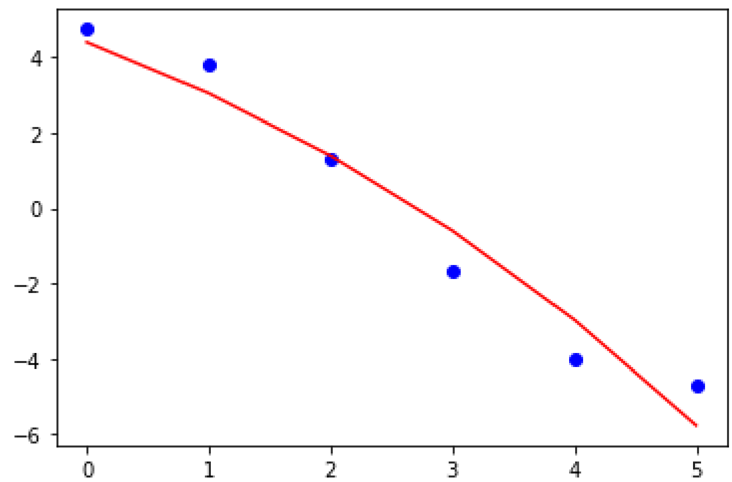
```
array([ 4.3983998 ,  3.04193734,  1.39160719, -0.59682607, -2.96759784,
       -5.76494354])
```

n = 4

Underfitting

```
plt.scatter( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

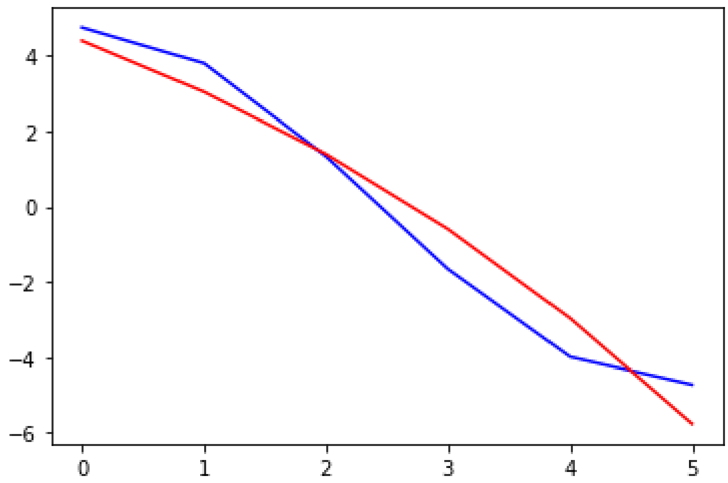
```
[<matplotlib.lines.Line2D at 0x7f50c85c0470>]
```



Plot between the predicted values and analytical values

```
plt.plot( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

```
[<matplotlib.lines.Line2D at 0x7f50c85c0da0>]
```



```
#Mean squared error

MSE_one = np.square(np.subtract(y, y_new)).mean()
MSE_one
```

```
0.6581383735498948
```

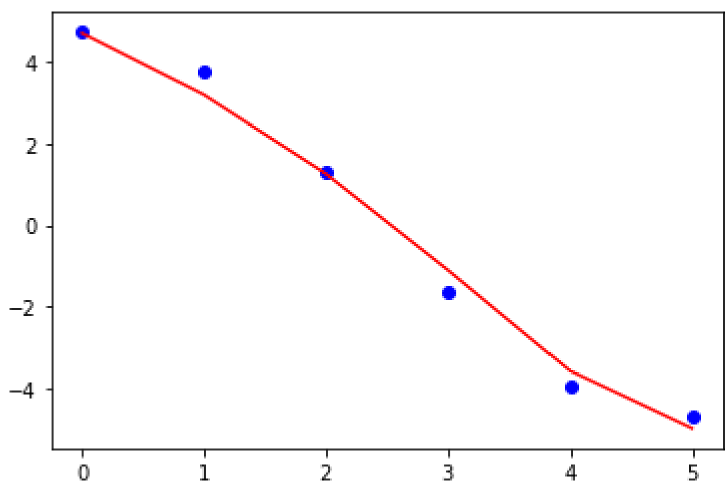
n = 8

```
result = function(8, 0.1, 50) #No.of exponents in the polynomial taken = 8
result.function_b(x, y)
y_new = result.function_c(x)
y_new
```

```
array([ 4.72310394,  3.2025117 ,  1.26679347, -1.10227794, -3.58916952,
       -4.99838477])
```

```
plt.scatter( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

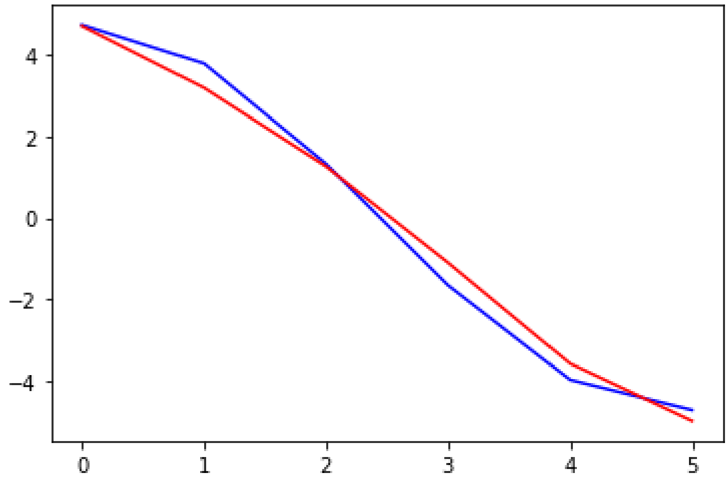
```
[<matplotlib.lines.Line2D at 0x7f50c7ff7dd8>]
```



Plot between the predicted values and analytical values

```
plt.plot( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

```
[<matplotlib.lines.Line2D at 0x7f50c7f225f8>]
```




```
#Mean squared error

MSE_two = np.square(np.subtract(y, y_new)).mean()
MSE_two
```

```
0.1509100146118269
```

n = 20

Overfitting

```
result = function(20, 0.1, 50) #No.of exponents in the polynomial taken = 20
result.function_b(x, y)
y_new = result.function_c(x)
y_new
```

```
array([ 4.69505819,  3.34826291,  1.53912644, -0.97980624, -4.40245124,
       -4.69761317])
```

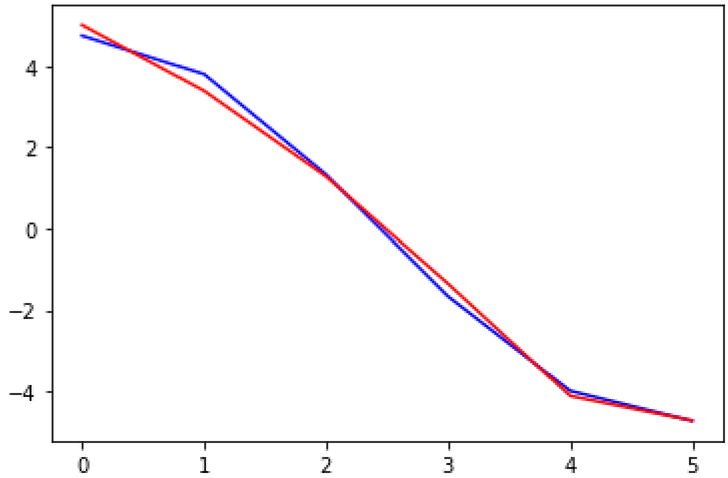
```
plt.scatter( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

```
[<matplotlib.lines.Line2D at 0x7f50c84c5908>]
```

Plot between the predicted values and analytical values

```
plt.plot( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

```
[<matplotlib.lines.Line2D at 0x7f50c889e400>]
```




```
#Mean squared error

MSE_three = np.square(np.subtract(y, y_new)).mean()
MSE_three
```

```
0.05894143128081794
```

Increase in the number of data points

```
#plotting for the increase in the number of data points

x = np.array( [[0], [1], [2], [3], [4], [5], [6], [7], [8], [9]] ) #x denotes the value
y_actual = np.array([5.000, 4.001, 1.404, -1.752, -4.210, -4.986, -3.770, -1.049, 2.091
y = np.array( [4.75, 3.80, 1.33, -1.66, -3.99, -4.73, -3.58, -0.99, 1.98, 4.17] ) #y de
plt.plot(x,y, color="blue")
```

```
[<matplotlib.lines.Line2D at 0x7f50c839ae48>]
```

n = 4

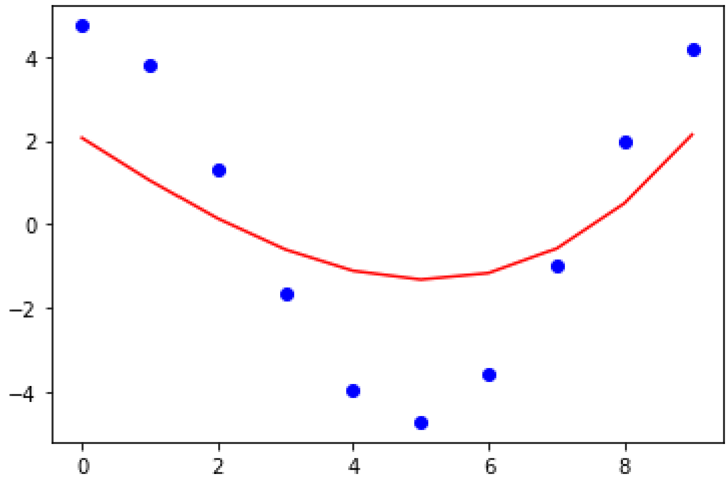
Underfitting

```
result = function(4, 0.1, 50) #No.of exponents in the polynomial taken = 4
result.function_b(x, y)
y_new = result.function_c(x)
y_new
```

```
array([ 4.3983998 ,  3.04193734,  1.39160719, -0.59682607, -2.96759784,
       -5.76494354])
```

```
plt.scatter( x, y, color = 'blue' )
plt.plot( x, y_new, color = 'red' )
```

```
[<matplotlib.lines.Line2D at 0x7f50c866e438>]
```



```
MSE_four = np.square(np.subtract(y, y_new)).mean()
MSE_four
```

```
4.933311252127409
```

n = 25

Overfitting

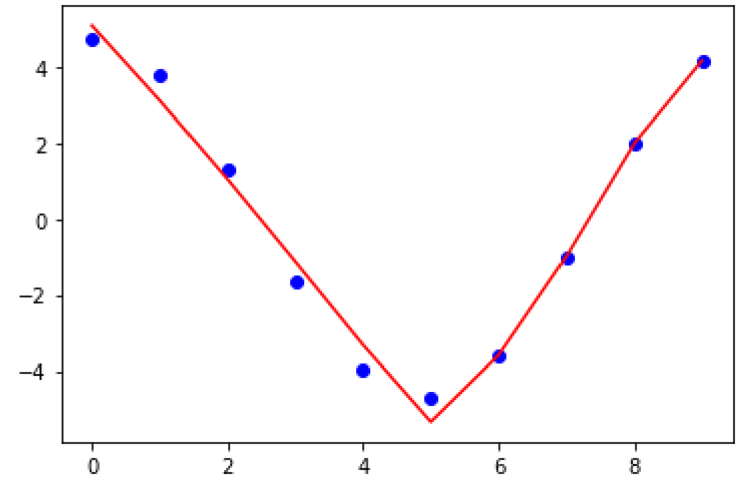
```
result = function(25, 0.1, 50) #No.of exponents in the polynomial taken = 25
result.function_b(x, y)
y_new = result.function_c(x)
y_new
```

```
array([ 5.09527984,  3.1272574 ,  1.04368458, -1.12667306, -3.31673369,
       -5.33928385, -3.58800574, -0.97832116,  1.99334719,  4.16944849])
```

```
plt.scatter( x, y, color = 'blue' )
```

```
plt.plot( x, y_new, color = 'red' )
```

```
[<matplotlib.lines.Line2D at 0x7f50c827d320>]
```




```
MSE_five = np.square(np.subtract(y, y_new)).mean()
MSE_five
```

```
0.1763108187409777
```

**Dimensionality reduction from 'n' to 'r', u
PCA approach**

Number of features(n) taken is 50

r is taken for values,

r = 1

r = 2

r = 3

Dimensionality reduction from 'n' to 'r', using a) Mode shape approach, b) PCA approach

Number of features(n) taken is 50

r is taken for values,

r = 1

r = 2

r = 3

Data generation by using the normal distribution for k and F

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as la
```

```
#force matrix using normal distribution for 50 features
from numpy import random
F = random.normal(0, 4, size=(50, 50)) #mean = 0, variance = 4
print(F)
```

```
[[-0.32826066 -2.63840343 -4.06635326 ... -4.06897903 -2.71077742
```

```
  -0.83510131]
 [ 2.37388417 -2.22016266 -6.00011555 ... -0.94734617  3.21545214
   5.33280485]
 [ 2.77276998  9.23053583  1.92631164 ...  1.68280197  3.26280854
  -5.80674242]
 ...
 [-0.67023126  5.8235095   1.77784708 ...  6.37681295 -6.11885884
   9.11087499]
 [-1.30792333 -3.55473962 -4.26669268 ...  1.27209736  2.56116553
   2.80352537]
 [-0.26623149 -1.74449597 -3.88186995 ...  1.22093178 -8.58926291
   0.68744661]]
```

```
#stiffness matrix using normal distribution for 50 features
from numpy import random
K = random.normal(29, 4, size=(50, 50))#mean = 29, variance = 4
print(K)
```

```
[[34.35947412 31.7440926  29.42680266 ... 21.76170858 22.49828244
  27.99509168]
 [26.89478594 33.90701656 27.29050321 ... 26.84335184 24.73051946
  28.4701384 ]
 [37.37882673 27.64582079 37.07426448 ... 26.73940338 19.2778512
  32.94848873]
 ...
 [25.94300887 34.23502538 28.89101037 ... 31.09536982 24.29226867
  30.86694379]
 [30.54084938 34.27384457 21.55309526 ... 31.72501902 31.58124067
  19.40675585]
 [26.95323292 32.03605141 25.91631795 ... 25.66128703 24.58690371
  29.35647769]]
```

```
#inverse of K
K_inverse = np.linalg.inv(K)
print(K_inverse)
```

```
[[-0.01106066  0.01660129 -0.04223546 ... -0.06286567  0.00902258
  -0.02145127]
 [ 0.01966305 -0.00131725 -0.0159605  ... -0.0117605   0.02148333
   0.01318231]
 [-0.02878862  0.02985367 -0.03839035 ... -0.01908277 -0.01792005
  -0.01834423]
 ...
 [-0.06899119  0.01526508 -0.06243377 ... -0.00197879 -0.00824669
   0.00184661]
 [ 0.01186506  0.00965848 -0.0422376  ... -0.02569065  0.01920257
   0.00190156]
 [ 0.0340736  -0.00406116 -0.00982725 ...  0.00827243 -0.02039658
  -0.02820923]]
```

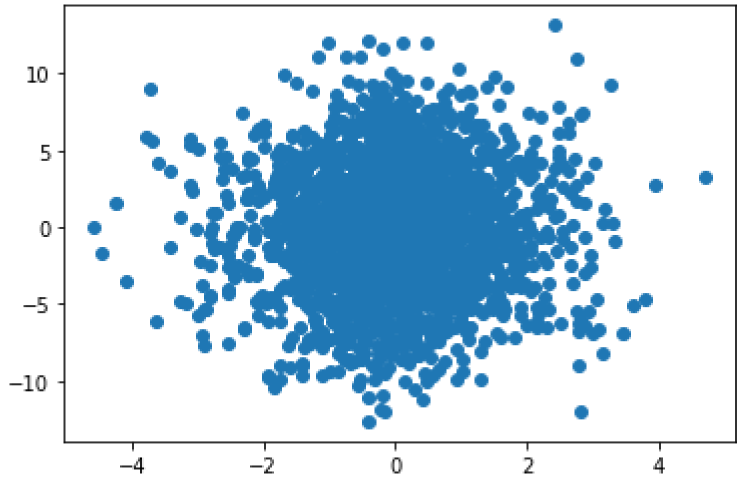
```
#displacement
X = K_inverse.dot(F)
print(X)
```

```
[[-1.61565900e-01 -1.38276099e+00 -1.60189026e+00 ... -1.34186742e+00
   2.23911834e-01 -5.65715321e-01]
 [-1.04571319e+00 -1.43260693e-01 -1.19760301e+00 ...  2.92706742e-01
```

```
   2.25683198e-01 -1.38471419e-03]
 [ 8.70026702e-01 -5.71265326e-01 -1.37076512e+00 ... -7.50654863e-01
   6.75501313e-01  1.86027320e-01]
 ...
 [-4.84940868e-01 -1.25650142e+00 -2.23100673e+00 ... -2.08222446e+00
  -5.24575107e-01 -1.95834295e-01]
 [ 2.02388098e-01 -5.93989731e-01 -7.57052845e-01 ...  3.26455915e-01
  -2.02753636e-01  5.45038155e-01]
 [ 6.37512068e-01  8.37451417e-02  1.40018191e+00 ... -5.21638152e-02
  -5.02810363e-01  2.10123481e-01]]
```

```
1 plt.scatter(X, F) #scatterplot showing 50 features
```

```
<matplotlib.collections.PathCollection at 0x7faa062aa860>
```



a) Mode Shape approach

```
1 eigen_values, eigen_vectors = la.eig(X) #computing eigenvalues and eigenvectors
2 eigen_values = eigen_values.real #taking only the real part excluding the imaginary par
3 eigen_vectors = eigen_vectors.real
4 print(eigen_values)
```

```
[-0.83129577 -0.83129577 -4.07378472  3.41127446  0.82329012  0.82329012
  1.59001963  1.59001963 -1.88463101 -1.88463101 -1.78059316 -0.34806307
 -0.34806307  1.69274778  1.06851613  1.06851613 -1.21490344 -0.94750014
 -0.94750014 -0.30490656 -0.30490656  0.11349885  0.11349885 -0.99916788
 -0.49906124 -0.49906124  0.27887292  0.27887292  0.89784069  0.89784069
  0.88130346 -0.60400602 -0.60400602  0.58087593  0.58087593  0.70251159
  0.23355499  0.23355499 -0.56254668 -0.05671278 -0.05671278 -0.28796987
 -0.28796987  0.33092736  0.33092736  0.30672005 -0.14393707  0.14785467
 -0.02255561  0.02014944]
```

```
1 v1 = eigen_vectors[2].reshape(50,1) #eigen vector for the smallest eigen value is compu
2 v1
```

```
array([[ 0.03402176],
       [ 0.03402176],
       [-0.06536042],
       [ 0.05950486],
       [ 0.03767031],
       [ 0.03767031],
       [ 0.11991198],
```

```
       [ 0.11991198],
       [-0.05823974],
       [-0.05823974],
       [ 0.07035585],
       [-0.0443585 ],
       [-0.0443585 ],
       [ 0.03428137],
       [-0.0387377 ],
       [-0.0387377 ],
       [-0.26713414],
       [-0.06537709],
       [-0.06537709],
       [-0.02364068],
       [-0.02364068],
       [-0.00812224],
       [-0.00812224],
       [-0.18898985],
       [ 0.05438232],
       [ 0.05438232],
       [-0.03430873],
       [-0.03430873],
       [-0.0290203 ],
       [-0.0290203 ],
       [-0.03500054],
       [-0.01567158],
       [-0.01567158],
       [ 0.05097855],
       [ 0.05097855],
       [ 0.0054683 ],
       [ 0.10131655],
       [ 0.10131655],
       [ 0.19732178],
       [-0.04754567],
       [-0.04754567],
       [-0.06086093],
       [-0.06086093],
       [ 0.06643983],
       [ 0.06643983],
       [ 0.10467897],
       [-0.1823799 ],
       [ 0.0087131 ],
       [ 0.02910694],
       [ 0.03432984]])
```

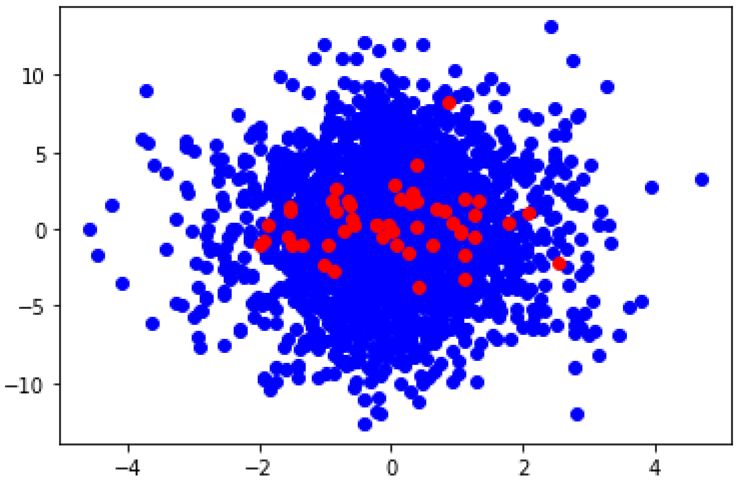
```
x_reduced = X.dot(v1) #transformed X vector w.r.t v1 direction(eigen vector)
x_reduced
```

```
array([[ 0.91634954],
       [ 0.00615527],
       [ 1.1090558 ],
       [ 0.42808408],
       [-0.59357529],
       [ 0.38230201],
       [-1.99253557],
       [-1.54163462],
       [-0.21503058],
       [-0.04365843],
       [ 1.26853569],
       [-1.93672045],
       [ 1.31130682],
```

```
       [ 1.04938607],
       [-0.12962275],
       [-1.49361668],
       [-0.63123659],
       [ 0.13016291],
       [-0.8678559 ],
       [ 1.27235369],
       [ 0.32744079],
       [ 0.85889472],
       [ 1.78394519],
       [ 0.3905208 ],
       [-0.84831129],
       [-0.2292705 ],
       [-0.89213273],
       [ 0.07332368],
       [ 0.81518734],
       [ 1.05382155],
       [-0.82301909],
       [-1.54785596],
       [ 0.68626773],
       [-1.00786181],
       [-0.55821935],
       [ 0.29072043],
       [ 0.27011441],
       [ 0.6376206 ],
       [ 1.1189785 ],
       [-0.94145904],
       [-1.34588214],
       [-0.71032592],
       [ 0.37611533],
       [-1.86798845],
       [-1.5691727 ],
       [ 2.53169053],
       [ 1.10696645],
       [ 2.0825902 ],
       [ 0.0536942 ],
       [-0.6433464 ]])
```

```
#scatter plot to show the reduced dimension
F_new = K.dot(x_reduced)
plt.scatter(X, F, color="blue")
plt.scatter(x_reduced, F_new, color="red")
```

```
<matplotlib.collections.PathCollection at 0x7faa06295668>
```



```
MSE = np.square(np.subtract(X,x_reduced)).mean() #Mean squared error
MSE
```

```
2.4716702451442147
```

b) PCA approach

i) n = 50, r = 1

```
mean = np.mean(X, axis=0) #transforming the x values w.r.t mean
X = X - mean
print(X)
```

```
[[-0.16032883 -1.38582083 -1.60287833 ... -1.3461429   0.22589077
  -0.56718092]
 [-1.04447612 -0.14632054 -1.19859108 ...  0.28843126  0.22766213
  -0.00285031]
 [ 0.87126377 -0.57432517 -1.37175318 ... -0.75493035  0.67748025
   0.18456172]
 ...
 [-0.4837038  -1.25956127 -2.23199479 ... -2.08649994 -0.52259617
  -0.19729989]
 [ 0.20362517 -0.59704958 -0.75804091 ...  0.32218043 -0.2007747
   0.54357256]
 [ 0.63874914  0.08068529  1.39919384 ... -0.0564393  -0.50083143
   0.20865788]]
```

```
XT = X.transpose() #computing X transpose
XT
```

```
array([[-0.16032883, -1.04447612,  0.87126377, ..., -0.4837038 ,
         0.20362517,  0.63874914],
       [-1.38582083, -0.14632054, -0.57432517, ..., -1.25956127,
        -0.59704958,  0.08068529],
       [-1.60287833, -1.19859108, -1.37175318, ..., -2.23199479,
        -0.75804091,  1.39919384],
       ...,
       [-1.3461429 ,  0.28843126, -0.75493035, ..., -2.08649994,
         0.32218043, -0.0564393 ],
       [ 0.22589077,  0.22766213,  0.67748025, ..., -0.52259617,
        -0.2007747 , -0.50083143],
       [-0.56718092, -0.00285031,  0.18456172, ..., -0.19729989,
         0.54357256,  0.20865788]])
```

```
cov = np.dot(X, XT) #computing covariance matrix
cov
```

```
array([[ 45.30076632,   0.98582499,  35.57152439, ...,  71.30277365,
          9.39303671, -10.72949307],
       [  0.98582499,  14.43823489, -10.9946391 , ...,  -5.34428528,
          3.12454977,  -7.79830639],
       [ 35.57152439, -10.9946391 ,  52.0074522 , ...,  73.65996617,
          4.18230226,  -6.1746608 ],
       ...,
```

```
       [ 71.30277365,  -5.34428528,  73.65996617, ..., 155.06874937,
          6.79329412, -23.93982314],
       [  9.39303671,   3.12454977,   4.18230226, ...,   6.79329412,
         11.64807035,   1.98625858],
       [-10.72949307,  -7.79830639,  -6.1746608 , ..., -23.93982314,
          1.98625858,  16.41848511]])
```

```
eigen_values, eigen_vectors = la.eig(cov) #computing eigenvalues and eigenvectors
eigen_values = eigen_values.real
eigen_vectors = eigen_vectors.real
print(eigen_values)
```

```
[2.17840885e+03 2.77883595e+02 1.74883586e+02 9.01072574e+01
 8.35072680e+01 2.95355578e+01 2.57014364e+01 1.83480849e+01
 1.39152275e+01 1.34211276e+01 8.83338761e+00 7.61824460e+00
 6.88534583e+00 5.43271672e+00 4.41554222e+00 4.03781836e+00
 2.95438278e+00 2.34171353e+00 2.04370011e+00 1.86516226e+00
 1.54633972e+00 1.40456212e+00 1.25156078e+00 1.14571547e+00
 1.12200620e+00 9.10371634e-01 8.08962164e-01 7.34052566e-01
 6.88914165e-01 5.35435497e-01 4.77512020e-01 3.96138136e-01
 3.53297054e-01 3.37353254e-01 2.84654279e-01 2.75772488e-01
 2.02645113e-01 1.09845588e-01 9.60125527e-02 4.29903320e-15
 4.24627934e-04 1.49449439e-03 1.06085354e-02 1.57937843e-02
 7.82256349e-02 3.57560693e-02 4.07879575e-02 5.05346462e-02
 6.11194410e-02 5.91631144e-02]
```

```
v1 = eigen_vectors[:,0].reshape(50,1) #computing the eigenvector with the highest eigen
print(v1)
```

```
[[ 0.12459615]
 [-0.01091151]
 [ 0.1361502 ]
 [ 0.07536737]
 [-0.07603867]
 [ 0.04094702]
 [-0.26190175]
 [-0.22857662]
 [-0.02686735]
 [-0.06506675]
 [ 0.19412958]
 [-0.22858025]
 [ 0.1898109 ]
 [ 0.12083483]
 [-0.02598408]
 [-0.16449669]
 [-0.06537844]
 [-0.00190484]
 [-0.07456267]
 [ 0.1811751 ]
 [ 0.06503398]
 [ 0.11790032]
 [ 0.21080275]
 [ 0.06728852]
 [-0.18399661]
 [ 0.01479172]
 [-0.14593122]
 [-0.00637049]
 [ 0.10530968]
```

```
 [ 0.13239557]
 [-0.11826925]
 [-0.24936578]
 [ 0.08301968]
 [-0.1003451 ]
 [-0.04309616]
 [ 0.04794526]
 [ 0.01010847]
 [ 0.12693966]
 [ 0.10492187]
 [-0.15801576]
 [-0.18089161]
 [-0.08517428]
 [ 0.08353734]
 [-0.21467108]
 [-0.18903492]
 [ 0.32374505]
 [ 0.11435384]
 [ 0.25893233]
 [ 0.01328912]
 [-0.03789443]]
```

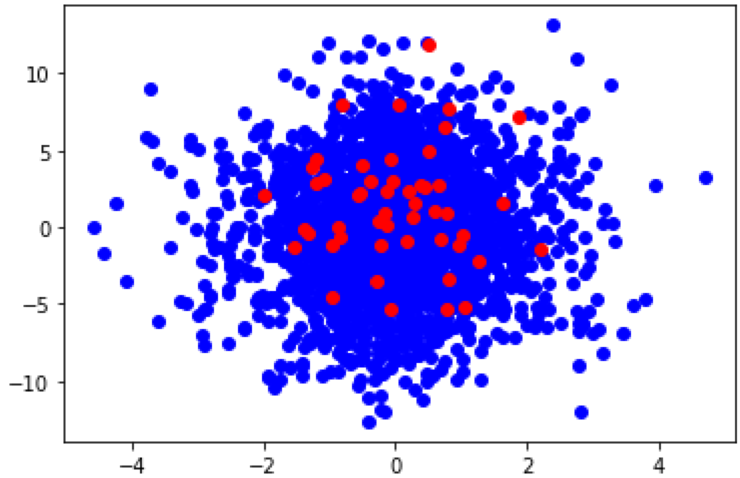
```
1 x1_reduced = X.dot(v1) #tranforming the X vector w.r.t v1 direction(eigen vector)
2 x1_reduced
```

```
array([[-0.87582018],
       [-0.96774148],
       [ 0.79376257],
       [-1.32876333],
       [-0.94760235],
       [ 0.25350127],
       [ 2.21858966],
       [ 1.26923341],
       [-0.05338239],
       [ 1.05443969],
       [-1.06919756],
       [ 0.51039619],
       [-1.98134298],
       [ 0.80573341],
       [ 0.79595816],
       [ 1.86163459],
       [-1.24685402],
       [ 0.17163936],
       [ 0.43747617],
       [ 0.28565053],
       [-1.18688813],
       [-0.29187863],
       [ 0.66913152],
       [-0.8331434 ],
       [ 0.21470875],
       [ 0.95980867],
       [ 0.49038975],
       [-0.12054406],
       [-0.12978422],
       [ 0.06091367],
       [-0.50358189],
       [ 1.61966635],
       [-0.26805089],
       [-0.16723528],
       [-0.05943281],
```

```
       [-0.81839876],
       [-1.19919213],
       [ 0.37244891],
       [-0.56037781],
       [-0.06434378],
       [ 0.76042559],
       [ 0.67994706],
       [-0.38762236],
       [ 0.78876668],
       [ 0.60966028],
       [-1.36762545],
       [-0.51957787],
       [-1.53781158],
       [-0.22383343],
       [ 1.02614454]])
```

```
# scatter plot to show the reduced dimension
F1_new = K.dot(x1_reduced)
plt.scatter(X, F, color="blue")
plt.scatter(x1_reduced, F1_new, color="red")
```

```
<matplotlib.collections.PathCollection at 0x7faa05d38fd0>
```




```
#Mean squared error
MSE_1 = np.square(np.subtract(X,x1_reduced)).mean()
MSE_1
```

```
1.9398925466898174
```

n = 50, r = 2

```
mean = np.mean(X, axis=0)
X = X - mean
XT = X.transpose()
cov = np.dot(X, XT)
eigen_values, eigen_vectors = la.eig(cov)
eigen_values = eigen_values.real
print(eigen_values)

```

```
[2.17840885e+03 2.77883595e+02 1.74883586e+02 9.01072574e+01
```

```
 8.35072680e+01 2.95355578e+01 2.57014364e+01 1.83480849e+01
 1.39152275e+01 1.34211276e+01 8.83338761e+00 7.61824460e+00
 6.88534583e+00 5.43271672e+00 4.41554222e+00 4.03781836e+00
 2.95438278e+00 2.34171353e+00 2.04370011e+00 1.86516226e+00
 1.54633972e+00 1.40456212e+00 1.25156078e+00 1.14571547e+00
 1.12200620e+00 9.10371634e-01 8.08962164e-01 7.34052566e-01
 6.88914165e-01 5.35435497e-01 4.77512020e-01 3.96138136e-01
 3.53297054e-01 3.37353254e-01 2.84654279e-01 2.75772488e-01
 2.02645113e-01 1.09845588e-01 9.60125527e-02 6.95076510e-15
 4.24627934e-04 1.49449439e-03 1.06085354e-02 1.57937843e-02
 7.82256349e-02 3.57560693e-02 4.07879575e-02 5.05346462e-02
 6.11194410e-02 5.91631144e-02]
```

```
1 v1 = eigen_vectors[:,:2].reshape(50,2)
2 print(v1)
```

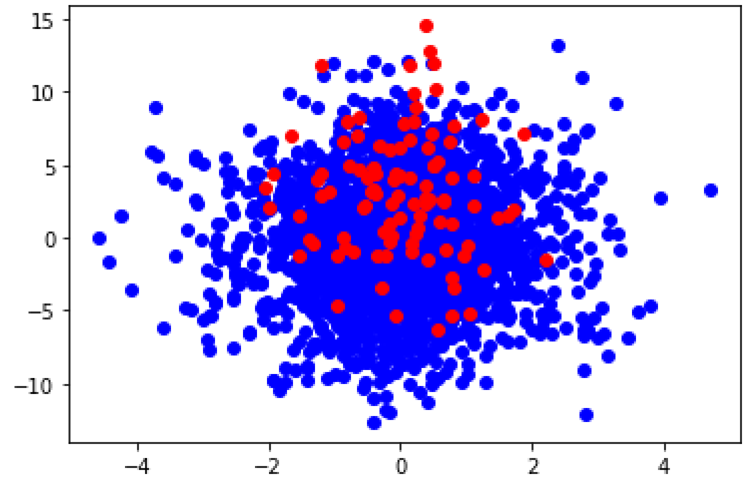
```
[[ 0.12459615  0.10401285]
 [-0.01091151  0.10459207]
 [ 0.1361502  -0.05346326]
 [ 0.07536737  0.22573629]
 [-0.07603867  0.02014435]
 [ 0.04094702 -0.15515965]
 [-0.26190175 -0.16852989]
 [-0.22857662 -0.02947437]
 [-0.02686735 -0.04968045]
 [-0.06506675 -0.37711594]
 [ 0.19412958  0.11832162]
 [-0.22858025  0.10253855]
 [ 0.1898109   0.34010815]
 [ 0.12083483 -0.13582645]
 [-0.02598408 -0.07143868]
 [-0.16449669  0.10828229]
 [-0.06537844  0.2284769 ]
 [-0.00190484  0.04058211]
 [-0.07456267  0.12293899]
 [ 0.1811751  -0.1168117 ]
 [ 0.06503398 -0.13281997]
 [ 0.11790032 -0.05272596]
 [ 0.21080275 -0.15050753]
 [ 0.06728852  0.17318047]
 [-0.18399661 -0.17279134]
 [ 0.01479172  0.00153879]
 [-0.14593122  0.08920354]
 [-0.00637049 -0.096328  ]
 [ 0.10530968 -0.04888031]
 [ 0.13239557  0.05079586]
 [-0.11826925  0.06366457]
 [-0.24936578 -0.24510664]
 [ 0.08301968 -0.04511739]
 [-0.1003451   0.24902101]
 [-0.04309616  0.02850715]
 [ 0.04794526  0.11758095]
 [ 0.01010847  0.1276262 ]
 [ 0.12693966 -0.00887085]
 [ 0.10492187 -0.25083505]
 [-0.15801576 -0.10420891]
 [-0.18089161  0.04329372]
 [-0.08517428 -0.10355696]
 [ 0.08353734  0.20288404]
 [-0.21467108  0.12269869]
```

```
 [-0.18903492  0.07487367]
 [ 0.32374505 -0.14659604]
 [ 0.11435384  0.00975085]
 [ 0.25893233 -0.06679499]
 [ 0.01328912  0.00864231]
 [-0.03789443 -0.09635568]]
```

```
x2_reduced = X.dot(v1)
F2_new = K.dot(x2_reduced)
```

```
plt.scatter(X, F, color="blue")
plt.scatter(x2_reduced, F2_new, color="red")
```

```
<matplotlib.collections.PathCollection at 0x7faa05c7ab38>
```




```
X2 = X[:,:2].reshape(50,2)
```

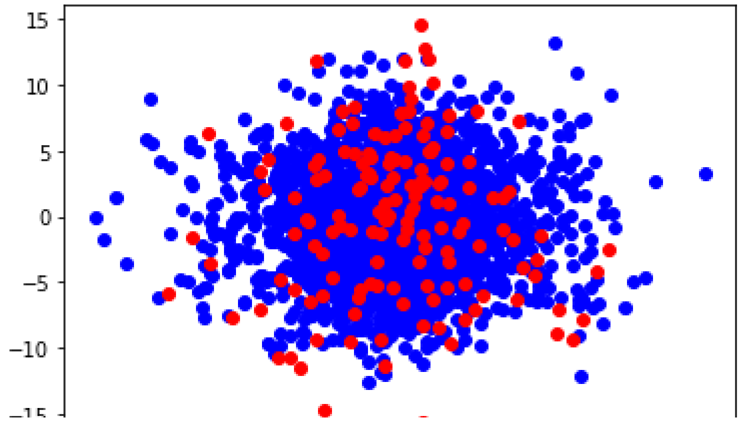
```
MSE2 = np.square(np.subtract(X2,x2_reduced)).mean()
MSE2
```

```
1.5181412250895392
```

n = 50, r = 3

```
mean = np.mean(X, axis=0)
X = X - mean
XT = X.transpose()
cov = np.dot(X, XT)
eigvals, eigvecs = la.eig(cov)
eigvals = eigvals.real
v3 = eigvecs[:,:3].reshape(50,3)
x3_reduced = X.dot(v3)
F3_new = K.dot(x3_reduced)
plt.scatter(X, F, color="blue")
plt.scatter(x3_reduced, F3_new, color="red")
```

```
<matplotlib.collections.PathCollection at 0x7faa05bcaf98>
```




```
1 X3 = X[:,:3].reshape(50,3)
```

```
1 MSE3 = np.square(np.subtract(X3,x3_reduced)).mean()
2 MSE3
```

```
1.2028941094985188
```

```
1 #Mean squared error w.r.t r
2
3 r = np.array([[1], [2], [3]])
4 MSE = np.array([3.80, 1.25, 1.61])
5 plt.plot(r, MSE, color="blue")
```

```
[<matplotlib.lines.Line2D at 0x7faa05b56438>]
```