Scientific Computing
Homework-3

Pankaj Kumar
2019262

Question 1:

1

$$x_{k+1} = x_k - f(x_k/d)$$

As it is already given in the Question itself

Now since for convergence

$$(x_{k+1})' < 1$$

Now, let $h(x) = x_{k+1} = x_k - f(x_k/d)$

$$\therefore \quad |h'(x_k)| < 1$$

$$\therefore \quad h'(x_k) = 1 - \frac{f'(x_k)}{d}$$

Since we have replace the true derivative with a constant value d.

$$\therefore \quad |h'(x_k)| < 1$$

$$= \left| 1 - \frac{f'(x_k)}{d} \right| < 1$$

$$= -1 < 1 - \frac{f'(x)}{d} < 1$$

~~Adding both side~~

Subtracting -1 from both side

$$-2 < -\frac{f'(x)}{d} < 0$$

$$\therefore \quad 0 < \frac{f'(x)}{d} < 2$$

Now for convergent, $d > \frac{f'(x)}{2}$

(ii) Now for Convergence rate

$$\lim_{k \to \infty} \frac{|e_{k+1}|}{|e_k|^r} = c$$

Now since $c = |h'(x)| < 1$

$$\therefore \quad \lim_{k \to \infty} \left| \frac{e_{k+1}}{(e_k)^r} \right| < 1$$

$$\therefore \quad r = 1$$

(iii) For Quadratic Convergence.

$$|h'(x)| = 0$$

$$\left| 1 - \frac{f'(x)}{d} \right| = 0$$

$$\therefore \quad \boxed{f'(x) = d}$$

b)

```python
import numpy as np
def newton(func,func_no ,x_0, epsilon):
    x_n = x_0
    while func(x_n,'function',func_no) > epsilon:
        y = func(x_n,'function',func_no)
        y_ = func(x_n,'derivative',func_no)
        x_n = x_n-y/y_
    return x_n

def func(x,choice,function):
    if function==1:
        if choice=='function':
            return x**2 - 1
        elif choice=='derivative':
            return 2*x
    elif function ==2:
        if choice=='function':
            return (x - 1)**4
        elif choice=='derivative':
            return 4*(x - 1)**3
    elif function==3:
        if choice=='function':
            return x - np.cos(x)
        elif choice=='derivative':
            return 1 + np.sin(x)




print(newton(func,1,10**6, 1e-100))
print(newton(func,2,10, 1e-60))
print(newton(func,3,1, 1e-100))
```

Output

```
1.0000000000000053
1.000028637697589
0.7390851332151607
```

Root: 1. 0000000000000053
Convergence rate 1.9998915768819285
Since this is a Quadratic rate and we know that convergence rate is quadratic therefore a multiplicity of the root is 1 (By the help of Newton's Law) In polynomial x**2 -1=0 convergence root 1 has a multiplicity of 1.

Root: 1.000028637697589
Convergence rate 1.000000000030321
Since we observe that it has a lower quadratic convergence rate and thus it is almost linear in nature. In polynomial (x-1)**4=0 which has root 1, but a multiplicity of 4.
Thus Convergence rate is very very close to linear.

Root: 0.7390851332151607
Convergence rate: 1.998848784777601
Since we observe that rate is Quadratic and it converges at root 0.73. Having a multiplicity of 1, Thus it results in a Quadratic Convergence rate

Question 2:

```python
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

max_iteration= 100000
value=1e-12
steps = []

def newton_method(function, Jacobian, x0,max_iteration=500):
    prev = x0
    jth_point = Jacobian(prev)
    ith_point = function(prev)
    store=[]
    y = solve(jth_point, ith_point)
    for i in range(1,max_iteration+1):
        store.append(la.norm(next-previous_main))
        if la.norm(next-prev) > value:
            steps.append(prev)
            store.append(la.norm(next-previous_main))
            y = solve(jth_point, ith_point)

    print('Spherical Roots:', steps[-1])
    return steps[-1]
```

b)

```python
def newton_method_2(function, Jacobian, x0,max_iteration=500):
    previous_main = x0
    jth_point = Jacobian(previous_main)
    ith_point = function(previous_main)

    y = solve(jth_point, ith_point)
```

```
    next = previous_main-y

    for i in range(1,max_iteration+1):
        store.append(la.norm(next-previous_main))
        if la.norm(next-previous_main) > value:
            ith_steps.append(previous_main)
            previous_main = next
            y = solve(jth_point, ith_point)
            store.append(la.norm(next-previous_main))
            next = previous_main-y
    r=ith_steps[-1]
    return r
```

Question 3:

Q3 Since it is already given that a function $F_n(t)$ is said to be Chebyshev three-term recurrence

iff $F_0(t) = 1$, $F_1(t) = t$, $F_{n+1}(t) = 2tF_n(t) - F_{n-1}(t)$

To prove: $F_n(t) = \cos(n\arccos(t))$ satisfies the chebyshev three-term recurrence

Proof

$F_n(t) = \cos(n\cos^{-1}(t))$

(i) putting $n=0$.

$F_0(t) = \cos(0 \times \cos^{-1}(t))$

$= \cos 0$

$= 1$

$\therefore F_0(t) = 1$

(ii) putting $n=1$

$F_1(t) = \cos(1 \times \cos^{-1} t)$

$= \cos(\cos^{-1} t)$ $\quad (\cos(\cos^{-1} a) = a)$

$= t$

$\therefore F_1(t) = t$.

(iii) Now putting $n+1$

$F_{n+1}(t) = \cos((n+1)\cos^{-1} t)$

$= \cos(\cos^{-1} t + n\cos^{-1} t)$

$= \cos(\cos^{-1} t)\cos(n\cos^{-1} t) - \sin(n\cos^{-1} t)\sin(\cos^{-1} t)$

Now putting $n-1$ in the function

$$F_{n-1}(t) = \cos((n-1)\cos^{-1} t)$$

$$= \cos(n\cos^{-1} t - \cos^{-1} t)$$

$$= \cos(n\cos^{-1} t) \cdot \cos(\cos^{-1} t) + \sin(n\cos^{-1} t)\sin(\cos^{-1} t)$$

Now, adding $F_{n-1}(t)$ and $F_{n+1}(t)$

we get

$$F_{(n+1)}t + F_{n-1}(t) = \cos(n\cos^{-1} t)\cdot\cos(\cos^{-1} t)$$

$$+ \sin(n\cos^{-1} t)\sin(\cos^{-1} t) - \sin(n\cos^{-1} t)\sin(\cos^{-1} t)$$

$$+ \cos(n\cos^{-1} t)\cdot\cos(\cos^{-1} t)$$

$$= 2\cos(n\cos^{-1} t)\cos(\cos^{-1} t) = 2\cos(n\cos^{-1} t) \times t$$

but we know that

$$F_n(t) = \cos(n\cos^{-1} t)$$

$\therefore$ we get

$$= 2\cos(n\cos^{-1} t)\, t$$

$$= 2F_n(t)\, t$$

$\therefore F_{(n+1)}t + F_{n-1}t = 2F_n(t)\, t$

Since it satisfies all three condition $\therefore$ we can say that chebyshev three term recurrence.

$\therefore F_n(t)$ is Chebyshev Polynomial

(b) Since, we prove that $F_n(t)$ satisfies all three condition of Chebyshev polynomial i.e.

$$F_0(t) = 1$$

$$F_1(t) = t$$

$$F_{n+1}^{t} + F_{n-1}^{t} = 2F_n(t)\,t$$

$\therefore$ we can say that $F_n(t)$

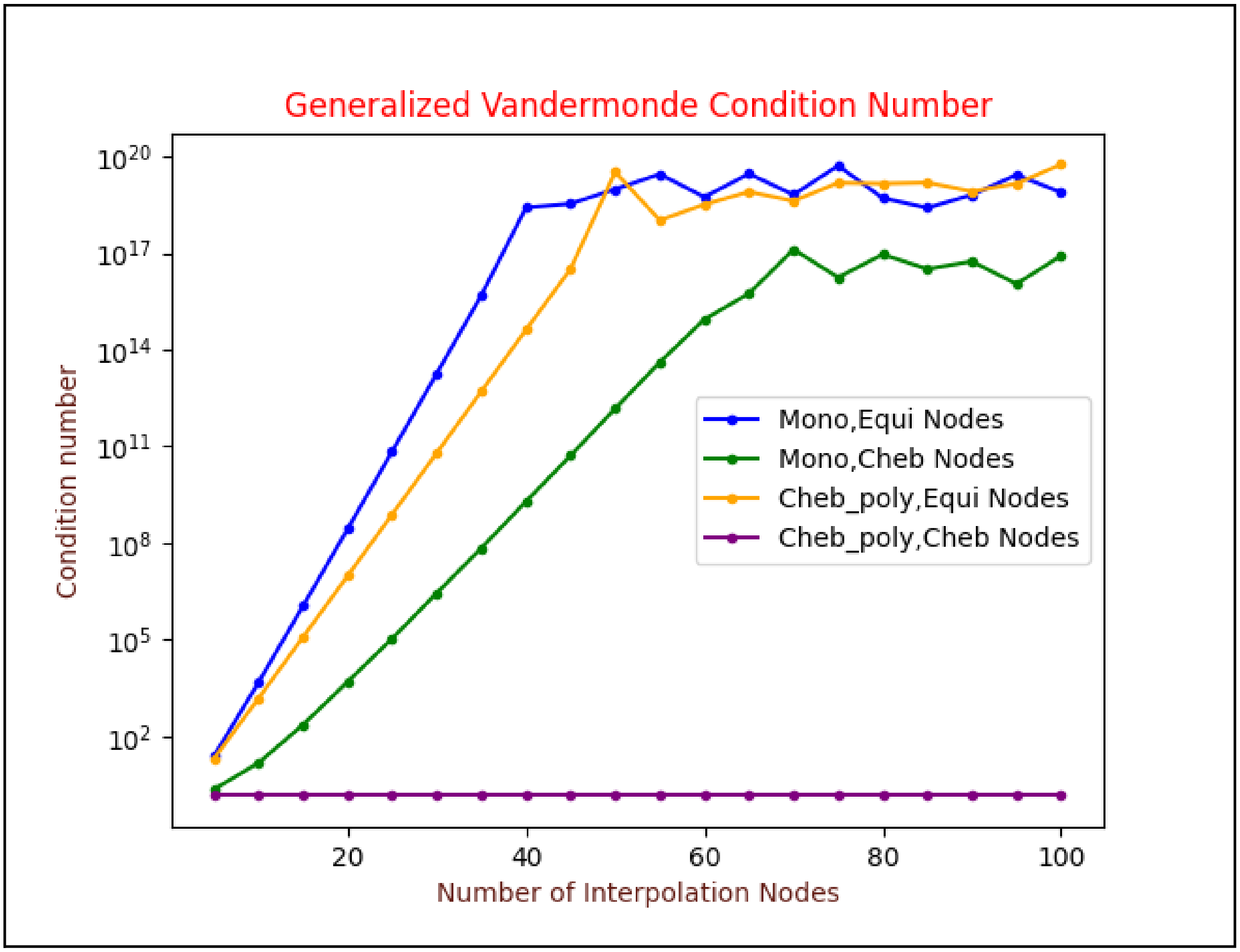
(c) For code, look in the report.

(d) By plotting the graph of Generalized Vandermode Condition no, we can say that best results are obtained when the Chebyshev polynomial is combined with chebyshev Node and for a given number of Interpolation node, it has a constant condition number.

Source Code

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
def equispaced(n):
    return np.linspace(-1, 1, n)
def chebyshev(n):
    X = np.zeros(n)
    for i in range(0, n):
        X[i] = np.cos((2*i + 1)/(2*n) * np.pi)
    return X
def size(X):
    return X.size
def chebyshev_poly(n, x):
    return np.cos(n * np.arccos(x))
def monomial(n, x):
    return x**n
def vandermonde(X, m):
    n = size(X)
    V_mono = np.zeros((size(X), m))
    V_cheb = np.zeros((size(X), m))
    for i in range(size(X)):
        for j in range(m):
            V_mono[i, j] = monomial(j, X[i])
            V_cheb[i, j] = chebyshev_poly(j, X[i])
    return V_mono, V_cheb
values={}
values["mono_equi"]=[]
values["cheb_qui"]=[]
values["mono_cheb"]=[]
values["cheb_cheb"]=[]

for n in range(5, 101, 5):
    X_equi = equispaced(n)
    X_cheb = chebyshev(n)

    mono_equi, cheb_equi = vandermonde(X_equi, n)
```

```python
    mono_cheb, cheb_cheb = vandermonde(X_cheb, n)

    values["mono_equi"].append([n, la.cond(mono_equi)])
    values["cheb_qui"].append([n, la.cond(cheb_equi)])
    values["mono_cheb"].append([n, la.cond(mono_cheb)])
    values["cheb_cheb"].append([n, la.cond(cheb_cheb)])

np_mono_equi = np.asarray(values["mono_equi"])
np_cheb_equi = np.asarray(values["cheb_qui"])
np_mono_cheb = np.asarray(values["mono_cheb"])
np_cheb_cheb = np.asarray(values["cheb_cheb"])


plot0_mono_eqi=np_mono_equi[:, 0]
plot0_cheb_eqi=np_cheb_equi[:, 0]
plot0_mono_cheb=np_mono_cheb[:, 0]
plot0_cheb_cheb=np_cheb_cheb[:, 0]

plot1_mono_eqi=np_mono_equi[:, 1]
plot1_cheb_eqi=np_cheb_equi[:, 1]
plot1_mono_cheb=np_mono_cheb[:, 1]
plot1_cheb_cheb=np_cheb_cheb[:, 1]

plt.semilogy(plot0_mono_eqi, plot1_mono_eqi, label='Mono,Equi
Nodes',color='blue',marker='.')
plt.semilogy(plot0_cheb_eqi, plot1_cheb_eqi, label='Mono,Cheb
Nodes',color='green',marker='.')
plt.semilogy(plot0_mono_cheb, plot1_mono_cheb, label='Cheb_poly,Equi
Nodes',color='orange',marker='.')
plt.semilogy(plot0_cheb_cheb, plot1_cheb_cheb, label='Cheb_poly,Cheb
Nodes',color='purple',marker='.')
plt.xlabel("Number of Interpolation Nodes",color="#641E16")
plt.ylabel('Condition number',color="#641E16")
plt.title("Generalized Vandermonde Condition Number",color="Red")
plt.xticks(visible = True)
plt.legend(loc='best')
plt.savefig("problem_3.png")
plt.show()
```



Generalized Vandermonde Condition Number
Condition number
Number of Interpolation Nodes
Mono,Equi Nodes
Mono,Cheb Nodes
Cheb_poly,Equi Nodes
Cheb_poly,Cheb Nodes

Question 4:

Q4 Since we have the Newton's polynomials as

$$f[t_1, t_2 \dots t_k] = \frac{f[t_2, t_3, \dots t_k] - f[t_1, t_2 \dots t_{k-1}]}{t_k - t_1}$$

$$f[t_j] = f(t_j)$$

To proof: we have to prove that approach gives the coefficient of the j^{th} basis function using Newton interpolation polynomial.

Proove by Mathematical Induction.

proove:

Assuming that $P_k(t)$ interpolating f on $t_1 \dots t_k$ such that

$$P_k(t) = a_1 + a_2(t-t_1) + a_3(t-t_1)(t-t_2) + \dots + a_k(t-t_1)(t-t_2) \dots (t-t_{k-1})$$

where $a_1, a_2, \dots a_{k-1}$ are induction hypothesis and given by the divided difference

Now, we have

$$f(t_k) = p_k(t)$$

Since $f(t_k) = p_k(t)$, $\therefore$

$$f(t_k) = a_1 + a_2(t_2 - t_1) + \dots + a_k(t_k - t_1) \dots (t_k - t_{k-1})$$

$\therefore$

$$\frac{f(t_k) - a_1}{t_k - t_1} = a_2 + a_3(t_k - t_2) + \cdots + a_k(t_k - t_2)\cdots(t_k - t_{k-1})$$

Therefore we get the get

$a_1 = f(t_1)$ (as in hypothesis itsel)

putting the value of $a_1 = f(t_1)$ in above equation, we get.

$$\frac{f(t_k) - f(t_1)}{t_k - t_1} = a_2 + \cdots a_k(t_k - t_2)\cdots(t_k - t_{k-1})$$

similarly, if we put the value of all a_i where $i \in [0 \ldots k]$

we get.

$$\frac{f[t_1 \ldots t_{k-1}] - a_{k-1}}{(t_k - t_{k-1})} = a_k$$

$$\therefore \quad f[t_1, t_2 \ldots t_k] = a_k$$

Hence proved.

(b) Given : we have three data points

$(-1,1)$, $(0,0)$, $(1,1)$

to proove : to find the interpolating polynomial of degree using

(i) Using Monomial (iii) Newton Basis

(ii) Lagrange Basis

Proof

(i) Monomial Basis

Since we already know that monomial basis linear system is given by the following equation

$$Ax = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

such that

$$Ax = y.$$

Since we have the the data point as follow.

$(-1,1)$, $(0,0)$, $(1,1)$

$\therefore$ putting them in equation itself.

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Now applying the guassian elimination to solve the equation. we get as follow

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right]$$

applying row transformation as

$$R_1 \to R_1 - R_2, \quad R_3 \to R_3 - R_2$$

$$\left[\begin{array}{ccc|c} 0 & -1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{array}\right]$$

$$R_3 \to R_3 + R_1$$

$$\left[\begin{array}{ccc|c} 0 & -1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \end{array}\right]$$

Now, on expanding we get.

$$-x_2 + x_3 = 1, \quad x_1 = 0, \quad 2x_3 = 2.$$

$$\boxed{x_1 = 0} \quad \boxed{x_3 = 1}$$

$$\therefore \boxed{x_2 = 0}$$

$$x = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$\therefore$ Polynomial will act as

$$P(t) = \begin{bmatrix} \cancel{0} \\ \cancel{0} \\ \cancel{1} \end{bmatrix} = 0 + 0t + t^2$$

$$= 0 + t^2$$

$$= t^2$$

(ii) Using Lagrange Basis.

Since we already know that lagrange basis function is given by

$$l_j(t) = \frac{l(t)\, w_j}{t - t_j} \qquad \forall j \in [1,2,3]$$

$$\text{and } l(t) = (t - t_1)(t - t_2)(t - t_3)$$

Now first we will find the value of w_1, w_2 and w_3 in the above function where $w_j = \frac{1}{l'(t_j)}$

$$w_1 = \frac{1}{(t_1 - t_2)(t_1 - t_3)} \quad , \quad w_2 = \frac{1}{(t_2 - t_1)(t_2 - t_3)}$$

$$w_3 = \frac{1}{(t_3 - t_1)(t_3 - t_2)}$$

Now, putting the value in the function itself.

B

$$P_2(t) = l(t)\left[y_1 \frac{w_2}{t-t_1} + y_2 \frac{w_2}{t-t_2} + y_3 \frac{w_3}{t-t_3}\right]$$

Now since we have the data points as $(-1,1), (0,0)$
$(1,1)$

$\therefore$

$$w_1 = \frac{1}{(-1)(-1-1)} = 1/2$$

$$w_2 = \frac{1}{(0+1)(0-1)} = \frac{1}{-1} = -1$$

$$w_3 = \frac{1}{(1+1)(1)} = \frac{1}{2}$$

$$l(t) = (t+1)(t-0)(t-1)$$
$$= t(t+1)(t-1)$$
$$= t(t^2-1)$$
$$= t^3-t$$

Now putting all these values in the lagrange function itself we get

$$P(t) = (t^3-t)\left(\frac{1}{2(t+1)} + 0 \cdot \frac{(-1)}{t-0} + \frac{1}{2(t-1)}\right)$$

$$= (t^3-t)\left(\frac{t-\cancel{1}+t+\cancel{1}}{2(t+1)(t-1}\right)$$

$$= (t^3 - t) \times \frac{\cancel{2}t}{\cancel{2}(t-1)(t+1)}$$

$$= t(\cancel{t^2-1}) \times \frac{t}{(\cancel{t-1})(\cancel{t+1})}$$

$$= t^2$$

$\therefore$ $P(t) = t^2$

(iii) Using Newton Basis

Since we already know that newton basis, linear system is given by

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & t_2 - t_1 & 0 \\ 1 & t_3 - t_1 & (t_3-t_1)(t_3-t_2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$Ax = y.$$

Now getting the values of A, x and y using the data point

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

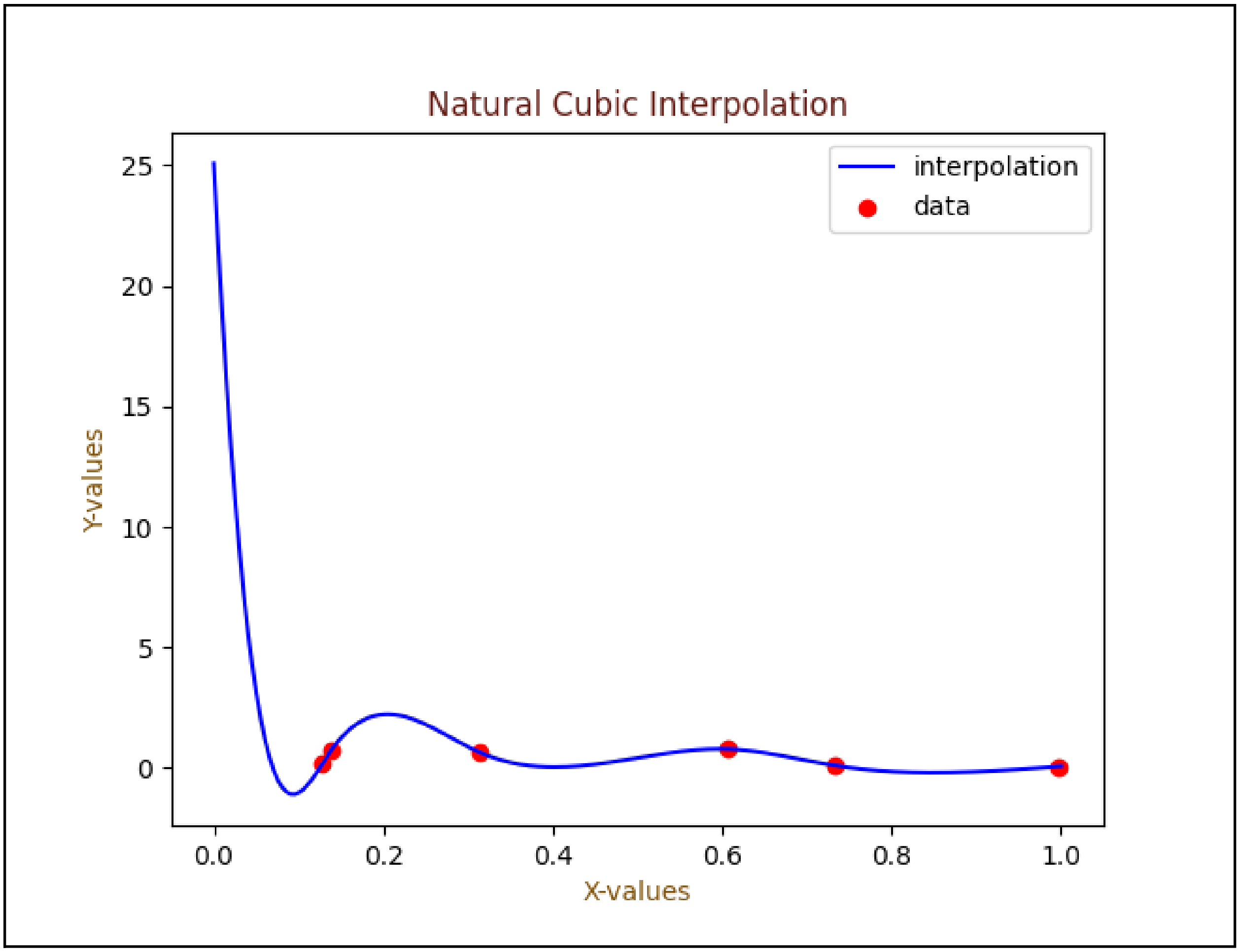
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$\therefore$ linear system become as follow

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Source Code:

```
import random
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline

np.random.seed(2021)
X = np.random.rand(6)
X = np.sort(X)
Y = np.random.rand(6)

cubic_spline = CubicSpline(X, Y, bc_type='natural', extrapolate=True)
plt.scatter(X, Y,marker='o', label='data', color='red')
plt.plot(np.linspace(0, 1-1e-20, 200), cubic_spline(np.linspace(0,
1-1e-20, 200)),  label="interpolation", color='blue')
plt.legend(loc='upper right')
plt.title('Natural Cubic Interpolation',color='#641E16')
plt.xlabel('X-values',color='#7E5109')
plt.ylabel('Y-values',color='#7E5109')
plt.show()
plt.savefig('problem_4c.png')
```


Natural Cubic Interpolation
interpolation
data
Y-values
X-values
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0