MTH 408/522**: Numerical Analysis**

Assignment 1

Instructions:

- Submit the answer to all the questions in one pdf file. Give your code in a pdf file as well.
- Please write your answer neatly. The submission deadline is 13 Feb 2023 before 4 PM.

Max mark: 20

---

1. Explain why the sequence of iterates $x_{n+1} = 1 - 0.9x_n^2$, with initial guess $x_0 = 0$, does not converge to any solution of the quadratic equation $0.9x^2 + x - 1 = 0$? **(1 Marks)**

2. Use Newton's method to calculate the unique root of

$$x + e^{-Bx^2}\cos(x) = 0$$

for, $B = 1, 5, 10, 25, 50, 100$ up-to twelve digit of accuracy. Theoretically, the Newton method will converge for any value of x_0 and B. Among the choices of x_0 used, choose $x_0 = 0$ and explain any anomalous behavior. **(3 Marks)**

3. The function $f(x) = \tan \pi x - 6$ has a zero at $(1/\pi)\arctan 6 \approx 0.447431543$. Let $p_0 = 0$ and $p_1 = 0.48$, and use ten iterations of each of the following methods to approximate this root. Which method is most successful and why?

   a. Bisection method

   b. Method of False Position

   c. Secant method

(3 Marks)

4. (a) Use the following iteration formula for the computation of $\sqrt{a}$

$$x_{n+1} = \frac{x_n\left(x_n^2 + 3a\right)}{3x_n^2 + a} \quad n \geq 0$$

Assuming initial approximation x_0 has been chosen sufficiently close to actual root, find its theoretical order of convergence. Give a table of computational results (results of your code) that illustrate the theoretical order of convergence. **(2 Marks)**

(b) Define an iteration formula by

$$x_{n+1} = z_{n+1} - \frac{f\left(z_{n+1}\right)}{f'\left(x_n\right)}, \quad z_{n+1} = x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)}$$

Show that the order of convergence of $\{x_n\}$ to α is at least 3 . **(1 Marks)**

5. The function $\text{erf}(x)$ is the normal distribution error function defined by

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

a. Use the Maclaurin series to construct a table for $\text{erf}(x)$ that is accurate to within 10^{-4} for $\text{erf}(x_i)$, where $x_i = 0.2i$, for $i = 0, 1, \ldots, 5$.

b. Use both linear interpolation and quadratic interpolation to obtain an approximation to erf $\left(\frac{1}{3}\right)$. Which approach seems most feasible and why? **(3 Marks)**

6. Consider the functions e^x and $\sin(x)$ on $[-1, 1]$ and its approximation by an interpolating polynomial.

(a) Prove that for both the function $\text{Max}_{0 \le |x| \le 1} |f(x) - p_n(x)| \to 0$ as $n \to \infty$.

(b) For both of the functions, write a program for Newton interpolating polynomial of degree $n = 10, 50, 100, 200$, and compute the error at 1000 random points. Make a table and graph of the maximum error as a function of n, that is, define E_n as

$$E_n = \max_{0 \le k \le 1000} |f(t_k) - p_n(t_k)|$$

where $t_k = -1 + 2k/1000$. Comment on the nature of E_n with increase in n for both of the functions.

(c) Repeat the experiment in (b) for Chebyshev nodes and make observations on the error. **(5 Marks)**

7. The following data are taken from a polynomial $p(x)$ of degree ≤ 5. What is the polynomial and what is its degree? **(1 Marks)**

x	-2	-1	0	1	2	3
$p(x)$	-5	1	1	1	7	25

8. Let $f(x) = e^x$. Show that $f[x_0, x_1, \ldots, x_m] > 0$ for all values of m and all distinct nodes $\{x_0, x_1, \ldots, x_m\}$ **(1 Marks)**

SHUBHAJIT DEY
shubhajit19@iiserb.ac.in
19299

NUMERICAL ANALYSIS
Assignment 1

2023-02-14

Answer 1.

Let us first state and prove an important result which will be crucial for completing this answer.

Theorem : If $g \in C^1[a,b]$ and p be in (a,b) with $g(p) = p$ and $|g'(p)| > 1$, then $\exists \delta > 0$ such that if $0 < |p_0 - p| < \delta$ then $|p_0 - p| < |p_1 - p|$.

Proof.
$\because g \in C^1[a,b]$, $p \in (a,b)$ and $|g'(p)| > 1$, we know $\exists \delta > 0$ such that $\forall x \in [p-\delta, p+\delta],\ |g'(x)| > 1$.
Now, using Mean Value Theorem, we get that $\exists\, \xi$ between p_0 and $p\ \ni$

$$|p_1 - p| = |g(p_0) - g(p)| = |g'(\xi)||p_0 - p| > |p_0 - p|.$$

Given that $p_0 \neq p$ and $p_0 \in [p-\delta, p+\delta]$.

Thus, this completes the proof.

$QED.$

Here we have, $g(x) = 1 - 0.9x^2 \in C^1[0,1]$. Also, the points for which $g(p) = p$ are exactly the points where $0.9p^2 + p - 1 = 0$ i.e., using Quadratic Formula,

$$p = \frac{-5 \pm \sqrt{115}}{9}.$$

We can observe that, $p = (-5 + \sqrt{115})/9 \in [0,1]$ and $g'(x) = -1.8x$. Hence,

$$|g'(p)| = \left| -\frac{9}{5}\frac{(-5+\sqrt{115})}{9} \right| > \left| \frac{-5+10}{5} \right| = 1.$$

Thus by using the above theorem, we get that $\exists \delta > 0$ such that for $p \in [p-\delta, p+\delta]$ then $|p_0 - p| < |p_1 - p|$. Given that we take $p_0 \in [p-\delta, p+\delta]$. Since, here we have a recursive sequence, we can re-scale the result to $|p_n - p| < |p_{n+1} - p|$.

$QED.$

Answer 2.

1. Function : $f(x) = x + e^{-Bx^2} \cos x = 0$
2. B = 1, 5, 10, 25, 50, 100.
3. Tolerance = 12-digit accuracy.

B	x_0	p	Rel. Error	Time Elapsed (in sec.)	Iterations
1	-0.5	-5.884017765009962e-01	0.00	0.345868	5
	-0.6	-5.884017765009962e-01	5.84922e-15	0.130487	4
	0	-5.884017765009962e-01	0.00	0.076974	7
5	-0.5	-4.049115482093092e-01	0.00	0.162168	6
	-0.4	-4.049115482093092e-01	0.00	0.202822	5
	0	-4.049115482093092e-01	0.00	0.157056	10
10	-0.4	-3.264020100974987e-01	1.36055e-15	0.161750	6
	-0.3	-3.264020100974987e-01	2.53608e-12	0.080657	5
	0	-9.846907977479555e-01	9.99248e-01	0.128982	10 (crossing maxit)
25	-0.3	-2.374362439062778e-01	0.00	0.090215	7
	-0.2	-2.374362439062778e-01	0.00	0.094977	6
	0	-9.999999798868863e-01	9.99999e-01	0.126050	10
50	-0.2	-1.832913332944849e-01	0.00	0.077944	6
	-0.1	-1.832913332944849e-01	0.00	0.084386	7
	0	-1.00	1.00	0.122131	10
100	-0.15	-1.398945646133183e-01	0.00	0.077056	6
	-0.1	-1.398945646133183e-01	0.00	0.086996	7
	0	-1.00	1.00	0.158643	10

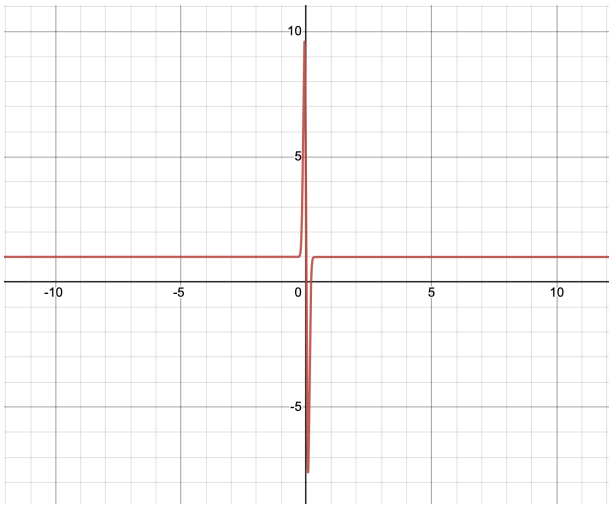
After conducting the experiment and running the relevant codes, we get the roots for the function $f(x) = x + e^{-Bx^2} \cdot \cos x$ as :

B	Root
1	-5.884017765009962e-01
5	-4.049115482093092e-01
10	-3.264020100974987e-01
25	-2.374362439062778e-01
50	-1.832913332944849e-01
100	-1.398945646133183e-01

Now to answer for the anomaly happening near the point $x = 0$, we have the crux of Newton-Raphson Method as :

$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)}.$$

We can observe that, the derivative of the given function becomes 0 near the point $x = 0$ (somewhere near $x = 0.005$). This possibly causes the anomaly we can see when the initial value is given is 0. The graph of the derivative is as follows:



Answer 3.

- Function : $f(x) = tan(\pi x) - 6$.

Interval	Approx. Root	Rel. Err	Time Elapsed	Iterations
[-0.5,0.5]	4.492187500000000e-01	7.812500000000000e-03	0.000613	7

Tabelle 1: Bisection Method

P_0	P_1	Root	Rel. Err	Time Elapsed	Iterations
0.00	0.48	4.420669490817012e-01	7.531756160959136e-03	0.023609	10

Tabelle 2: Regula Falsi Method

P_0	P_1	Root	Rel. Err	Time Elapsed	Iterations
0.00	0.48	-1.139444050480790e+02	1.031056883786811e+00	0.023609	9

Tabelle 3: Secant Method

Here, we need to provide the name of a better method amongst this specific to the problem. Thus, we need to compare the performance on the basis of the iterations done and time elapsed to provide comparable outputs. On that basis, **the Bisection method** is a better method in the given plot. Secant method did not even converge anywhere near to the root.

This can be explained by the fact that the slope of $f(x)$ reaches large values near its root i.e., $(1/\pi)\arctan 6$. Since the Regula Falsi method depends upon slopes, they tend to fail whenever we have a situation like this, that is, when there is a function which changes drastically or when there is a function that changes too slowly. Basically anomaly when, $|f'(x)| >> 1$ or $|f'(x)| \sim 0$.

ANSWER 4

a. From the question, we take the liberty to assume that $\{x_n\} \to \sqrt{a}$.
Now,

$$\begin{aligned}\lim_{n\to\infty} \frac{|x_{n+1} - \sqrt{a}|}{|x_n - \sqrt{a}|^\alpha} &= \lim_{n\to\infty} \frac{\left|\frac{x_n(x_n^2+3a)}{3x_n^2+a} - \sqrt{a}\right|}{|x_n - \sqrt{a}|^\alpha} \\ &= \lim_{n\to\infty} \frac{|x_n^3 + 3ax_n - 3\sqrt{a}x_n^2 - a\sqrt{a}|}{|3x_n^2 + a||x_n - \sqrt{a}|^\alpha} \\ &= \lim_{n\to\infty} \frac{|x_n - \sqrt{a}|^3}{|3x_n^2 + a||x_n - \sqrt{a}|^\alpha}.\end{aligned}$$

If the above limit exists (finite) and is non-zero, then we observe that for $\alpha = 3$,

$$\lim_{n\to\infty} \frac{|x_{n+1} - \sqrt{a}|}{|x_n - \sqrt{a}|^\alpha} = \lim_{n\to\infty} \frac{1}{|3x_n^2 + a|} = \frac{1}{4a} \neq 0.$$

Hence, the theoretical order of convergence is $\alpha = 3$.

$QED.$

Coding Results : When we consider $\frac{|x_{n+1}-\sqrt{a}|}{|x_n-\sqrt{a}|^3} = \lambda$, we get for $a = 4$

n	$\sqrt{a}$	λ
1	1.000000000000000e-01	2.481389578163772e-01
2	2.980148883374690e-01	2.343875275915045e-01
3	1.998794387124115e+00	1.628900128730090e-01
4	1.999999999890378e+00	7.591167805741734e-02
5	2.000000000000000e+00	6.255654718542600e-02
6	2.000000000000000e+00	6.250000000513853e-02
7	2.000000000000000e+00	6.249999999999997e-02
8	2.000000000000000e+00	6.249999999999997e-02
9	2.000000000000000e+00	6.249999999999997e-02

b. We have,

$$x_{n+1} = z_{n+1} - \frac{f(z_{n+1})}{f'(x_n)}$$
$$z_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Now applying Newton's method for convergence to any root, say, a: $z_{n+1} - r \approx \frac{f"(a)}{f'(a)}(x_n - a)^2$ and following from above we get :

$$x_{n+1} - a = \frac{z_{n+1} - a - f'(a)(z_{n+1} - a) + \frac{1}{2}f"(a)(\cdots)}{f'(a) + f"(a)(x_n - a)} \approx \frac{f"(a)^2(x_n - a)^3}{f'(a)^2}.$$

Thus, one can see that the order of convergence is at least 3. $\boxed{|x_{n+1}| \leq c \cdot |x_n - a|^3}$.

$QED.$

ANSWER 5

a. We have the error function: $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$. Now we need to use the Maclaurin Series, which by definition is the Taylor Expansion around the point 0, to deduce the values of $erf(x)$. Using the Maclaurin Expansion, we have:

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots$$

Now here, we have :

$$erf(0) = 0;\ erf'(0) = 0;\ erf''(0) = 0;\ erf'''(0) = \frac{-4}{\sqrt{\pi}}.$$

i	x_i	$erf(x_i)$
0	0.2i = 0	0.00
1	0.2i = 0.2	-3.009011112254701e-03
2	0.2i = 0.4	-2.407208889803761e-02
3	0.2i = 0.6	-8.124330003087690e-02
4	0.2i = 0.8	-1.925767111843009e-01
5	0.2i = 1.0	-3.761263890318375e-01

b. Now we use both linear and quadratic interpolation to approximate the value of $f(x) = erf(x)$ at $x = 1/3$.

Interpolation	x nodes	$erf(1/3)$
Linear	0, 0.4	-2.006007408169801e-02
Quadratic	0, 0.2, 0.4	-2.006007408169801e-02

From feasibility point of view, calculating 1 degree polynomial is a bit better than calculating 2 degree polynomial. The time complexity involved as well as the memory under usage in doing the former is lesser than the that of the latter. Moreover this conclusion is recommended in the current problem because the results obtained for $erf(1/3)$ are almost the same.

ANSWER 6

b. Equispaced Nodes.

For $f(x) = e^x$:

n	E_n
10	$8.648223e-11$
50	$3.684868e-04$
100	$2.257212e+11$
200	$9.420729e+40$

For $f(x) = \sin(x)$:

n	E_n
10	$2.489963e-10$
50	$1.690538e-04$
100	$5.404206e+10$
200	$7.943306e+39$

c. Chebyshev Nodes.

For $f(x) = e^x$:

n	E_n
10	$9.983946e-12$
50	$1.797084e-15$
100	$2.613941e-15$
200	$4.084282e-15$

For $f(x) = \sin(x)$:

n	E_n
10	$2.856070e-11$
50	$2.111014e-15$
100	$2.242952e-15$
200	$3.430397e-15$

Answer 7.

The task in hand is to interpolate a polynomial p(x) of degree ≤ 5.
The data for the same is given as:

x	-2	-1	0	1	2	3
p(x)	-5	1	1	1	7	25

Results :
1. The resultant and required polynomial is: $p(x) = x^3 - x + 1$.
2. The code for the same is given in the codes section of this document.
3. Time elapsed by the code is 0.246497 seconds.

Answer 8.

We have the following Theorem from book :

Theorem : Suppose that $f \in C^n[a, b]$ and $x_0, x_1, \cdots, x_n$ are distinct numbers in [a,b]. Then a number ξ exists in (a,b) with

$$f[x_0, x_1, \cdots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

In our context, $f(x) = e^x$. Without Loss of Generality, we can choose any closed interval $[a, b]$ such that for any given set of distinct nodes $\{x_0, x_1, \cdots, x_n\}$, all of them lies in $[a, b]$.

Here, borrowing results from Real Analysis we know that $f(x) = e^x \in C^\infty[a, b]$ and $f^{(n)}(x) = e^x$ itself, for each $n = 0, 1, \cdots$. Thus, using the above theorem, we can proceed as, for each appropriate value of m and distinct nodes $\{x_0, x_1, \cdots, x_m\}$, we have $\xi \in [a, b]$

$$\begin{aligned} f[x_0, x_1, \cdots, x_n] &= \frac{f^{(n)}(\xi)}{n!} \\ &= \frac{e^{(\xi)}}{n!} \\ &> 0 \qquad \text{since, } e^x \text{ is always positive in } \mathbb{R} \text{ and } n! \text{ is positive, obviously.} \end{aligned}$$

Thus, this completes the proof.

$QED.$

CODES – MATLAB

CODE FOR NEWTON-RAPHSON METHOD

```
% A code to implement the Newton-Raphson method to find roots of equations.
% Program : To approximate a root of the equation
% f(x) = 0 in the interval [a, b].
%--------------------------------
% The main function defined is Newton-Raphson().
% To use this function run the commands:
%           1. syms x;
%           2. [numit,p,relerr,P] = Newton_Raphson(appropriate inputs)
% in the command window.
% Note: the input 'g' must me symbolic expression.
%--------------------------------

function [numit,p,relerr,P] = Newton_Raphson(g,pzero,tol,maxit)
% Input :  - g, the iteration function.
%          - pzero, the initial guess for the root.
%          - tol, tolrence.
%          - maxit, maximum number of iteraions.
% Output : - numit, the number of iterations that were carried out.
%          - p, the approximation to the fixed point/ root
%          - relerr, the relative error in the approximation.
%          - P, it contains the sequence pn.

tic;
syms x;

format long e; %..........see the matlab format command
P(1) = pzero;
eps = 10^-12;

f = diff(g);

for k = 2:maxit
    g0 = vpa(subs(g,x,P(k-1)));
    g0_derr = vpa(subs(f,x,P(k-1)));

    P(k) = P(k-1) - (g0/g0_derr);

    err = abs(P(k) - P(k - 1));
    relerr = err/(abs(P(k) + eps)); %......machine epsilon is added in order to
    p = P(k);                       %......balance out some error for division.

    if (err < tol) || (relerr < tol )
        break;
    end
end

numit = k;
if k >= maxit
    disp('number of maximum iterations exceeded')
else
    fprintf('Five digit accurate approximaion of fix point computed in %d iterations. \n',k);
end

P = P';
toc
end
```

CODE FOR BISECTION METHOD

```
% A code to implement the bisection method to find roots of polynomials.
% Program : To approximate a root of the equation
% f(x) = 0 in the interval [a, b].
%--------------------------------
% The main function defined is bisect().
% To use this function run the command:
% [root,err,f_root] = Bisect(appropriate input variables)
% in the command window.
%--------------------------------

function [root,err,f_root] = Bisect(f,a,b,delta)
% Proceed with the method only if f (x) is continuous and
% f(a) and f(b) have opposite signs.
% Inputs:  - f, the function input as a string fs.
%          - a and b, the left and right endpoints of the 'compact' interval.
%          - delta, the tolerance.
% Outputs: - c is the zero.
%          - yc = f(c).
%          - err, the error estimate for c.
tic;

ya = feval(f,a);
yb = feval(f,b);
maxit = 0; %.....................................................maximum number of iterations.

if (ya * yb > 0)
    fprintf('f(a) and f(b) are not of the opposite sign.\n Please select the interval wisely!');
    return;
end

max_bdd = 1 + round((log(b-a)-log(delta))/log(2));
fprintf('This method will require at most %d
iterations to produce %e accurate computed solution.\n',max_bdd,delta );

for k = 1:max_bdd
    root = (a + b)/2;
    f_root = feval(f,root);
    if f_root == 0
       a = root;
       b = root;
    elseif yb * f_root > 0
       b = root;
       yb = f_root;
    elseif ya * f_root > 0
       a = root;
       ya = f_root;
    end

    if b - a < delta
       maxit = k;
    break,
    end
end

fprintf('Desired root is obtained in %d iterations. \n',maxit);
root = (a + b)/2;
err = abs(b - a);
f_root = feval(f,root);
toc
end
```

CODE FOR REGULA FALSI METHOD

```
% A code to implement the Secant method to find roots of equations.
% Program : To approximate a root of the equation
% f(x) = 0 in the interval [a, b].
%---------------------------------
% The main function defined is Regula().
% To use this function run the commands: [numit,p,relerr,P] = Regula(appropriate inputs)
% in the command window.
%---------------------------------

function [numit,p,relerr,P] = Regula(g,pzero,pone,tol,maxit)
% Input :  - g, the iteration function.
%          - pzero, the initial guess for the root.
%          - pone, the second initial guess for the root.
%          - tol, tolrence.
%          - maxit, maximum number of iteraions.
% Output : - numit, the number of iterations that were carried out.
%          - p, the approximation to the fixed point/ root
%          - relerr, the relative error in the approximation.
%          - P, it contains the sequence pn.

tic;
format long e; %.............see the matlab format command
eps = 10^-12;
P(1) = pzero;
P(2) = pone;
num = feval(g,P(2)) * (P(2) - P(1));
din = feval(g,P(2)) - feval(g,P(1));
P(3) = P(2) - (num/din);

for k = 3:maxit
    if (sign(feval(g,P(k-1))) * sign(feval(g,P(k))) > 0) %..signum fxn used to
        P(k-1) = P(k-2);                                 %..handle undervalue err.
    end
    numerator = feval(g,P(k)) * (P(k) - P(k-1));
    dinominator = feval(g,P(k)) - feval(g,P(k-1));
    P(k+1) = P(k) - (numerator/dinominator);

    err = abs(P(k+1) - P(k));
    relerr = err/(abs(P(k+1) + eps)); %.......machine epsilon is added in
    p = P(k+1);                       % order to balance out some error for division.

    if (err < tol) || (relerr < tol )
        break;
    end
end

numit = k + 1;
if k >= maxit
    disp('number of maximum iterations exceeded')
else
    fprintf('Five digit accurate approximaion of fix point computed in %d iterations. \n',k);
end

P = P';
toc
end
```

CODE FOR SECANT METHOD

```
% A code to implement the Secant method to find roots of equations.
% Program : To approximate a root of the equation
% f(x) = 0 in the interval [a, b].
%---------------------------------
% The main function defined is Secant().
% To use this function run the commands: [numit,p,relerr,P] = Secant(appropriate inputs)
% in the command window.
%---------------------------------

function [numit,p,relerr,P] = Secant(g,pzero,pone,tol,maxit)
% Input :  - g, the iteration function.
%          - pzero, the initial guess for the root.
%          - pone, the second initial guess for the root.
%          - tol, tolrence.
%          - maxit, maximum number of iteraions.
% Output : - numit, the number of iterations that were carried out.
%          - p, the approximation to the fixed point/ root
%          - relerr, the relative error in the approximation.
%          - P, it contains the sequence pn.

tic;
format long e; %..............see the matlab format command
P(1) = pzero;
P(2) = pone;
eps = 10^-12;

for k = 3:maxit
    numerator = feval(g,P(k-1)) * (P(k-1) - P(k-2));
    dinominator = feval(g,P(k-1)) - feval(g,P(k-2));
    P(k) = P(k-1) - (numerator/dinominator);

    err = abs(P(k) - P(k - 1));
    relerr = err/(abs(P(k) + eps));
    p = P(k);

    if (err < tol) || (relerr < tol )
        break;
    end
end

numit = k - 1 ;
if k >= maxit
    disp('number of maximum iterations exceeded')
else
    fprintf('Five digit accurate approximaion of fix point computed in %d iterations. \n',k);
end

P = P';
toc
end
```

```
%---------------------------------------
%------ GENERATING LAGRANGE POLY -------
%---------------------------------------
% A code to generate Lagrange polynomial of degree k using given x-nodes.
% Program : To generate Lagrange polynomial.
% Degree k, given x-nodes.
%---------------------------------
% The main function defined is Lagrange_basis().
% To use this function run the commands: Lagrange_basis(x,x_nodes,k)
% in the command window.
%---------------------------------

function L_k = Lagrange_basis(x,x_nodes,k)
% Input  : - x, a vector of x-values or a symbolic variable.
%          - x_nodes, a vector of size (n+1) storing the values of nodes x_k (k=0,1,2,...,n).
%          - k, a integer in the range from 0 to n.
% Output : - L_k, k_th Lagrange polynomial (of degree n) that is evaluated at x.
 n = length(x_nodes) - 1;
 xk = x_nodes(k+1);
 L_k = 1;
 for i = 0:n
     if i == k
         continue;
 end
     prod = (x - x_nodes(i + 1)) / (xk - x_nodes(i + 1));
     L_k = L_k.*prod;

 end
```

```
%---------------------------------------
%------ INTERPOLATING FUNCTIONS --------
%---------------------------------------
% A code to interpolate functions f with Lagrange polynomials.
% Program : To interpolate f.
% Pre-requisite program : Lagrange_basis.m
%---------------------------------
% The main function defined is Lagrange_Interp().
% To use this function run the commands: Lagrange_Interp(appropriate inputs)
% in the command window.
%---------------------------------

function [polyInterp] = Lagrange_Interp(fdata, x_nodes, eval_nodes)
% Input :  - fdata, a vector of function values at (n+1) nodes.
%          - x_nodes, a vector of size (n+1) storing the values of nodes x_k (k=0,1,2,..n)
%          - eval_nodes, a vector of test points.
% Output : - PX, polynomial approximation of f at the all evaluation points.

fdata = [0.0 -2.407208889803761e-02];
x_nodes = [0 0.4];
eval_nodes = [1/3 0.333];
n = length(fdata) - 1;
polyInterp = zeros(size(eval_nodes));
for k = 0:n
    yk = fdata(k + 1);
    L_k = Lagrange_basis(eval_nodes,x_nodes,k);
    polyInterp = polyInterp + (yk * L_k);
end
end
```

CODE TO GET INTERPOLATING POLYNOMIAL

```
%---------------------------------------
%---------- INTERPOLATION --------------
%---------------------------------------

% A code to provide the interpolating polynomial to
% interpolate f with Lagrange polynomials.
% Program : To interpolate f.
% Pre-requisite program : NIL
%---------------------------------
% The main function defined is Interpol().
% To use this function run the commands: Interpol(appropriate inputs)
% in the command window.
%---------------------------------

function [n,P] = Interpol(x_val,f_val)
% Input :  - fX, a vector of function values at (n+1) nodes.
%          - X, a vector of size (n+1) storing the values of nodes x_k (k=0,1,2,..n)
% Output : - n, degree of the interpolating polynomial.
%          - P, the interpolating polynomial.
tic;
syms x;

k = size(x_val);
n1 = k(2);
K = x - x_val; %..........here jth entry of K corresponds to x-X(j).

for j = 1:n1
    X1 = K;
    X2 = x_val;

    X1(j)=[];
    X2(j)=[];

    Num = prod(X1);
    Den = prod(x_val(j)-X2);
    L(j) = simplify(Num/Den); %..........jth Lagrange polynomial.
end

P = simplify(sum(f_val.*L)); %............the interpolating polynomial.
n = polynomialDegree(P,x);

toc
end
```

CODE TO VERIFY ORDER OF CONVERGENCE OF A GIVEN RECURSIVE SEQUENCE

```
%---------------------------------------
%------- ORDER OF CONVERGENCE ----------
%---------------------------------------

% A code to verify the order of convergence of an iterative sequence.
% Program : To get the order of convergence.
% Pre-requisite program : NIL
%---------------------------------
% To use this function just run the program.
%---------------------------------

function [P, y] = Conv()

n_max = 20;
x_0 = 0.1;
a = 4;

    function x_n1 = seq(x_n)
        x_n1 = x_n * (x_n^2 + 3 * a) / (3 * x_n^2 + a);
    end

P = zeros(length(n_max));
P(1) = x_0;

for i = 2:n_max
    P(i) = seq(P(i-1));
end

y=[];
y=[y,1 / abs(3 * P(1)^2 + a)];

for i = 2:n_max-1
    y = [y,1 / abs(3 * P(i)^2 + a)];
    err = abs(y(i) - y(i - 1));
    abs_err = err / (eps + abs(y(i)));
end
y = y';
end
```

CODE TO TEST INTERPOLATION OF ANY GIVEN FUNCTION

```
%---------------------------------------
%--- TESTING FUNCTION INTERPOLATION ----
%---------------------------------------
% A code to test polynomial interpolations of functions f.
% Program : To test interpolations done.
% Pre-requisite program : Lagrange_basis.m, Lagrange_Interp.m
%---------------------------------
% To use this function run the commands: PolyInterp in the command window.
%---------------------------------

clear all;
tic;
f = @(x) exp(x);  %..........function (A) to be tested.
% f = @(x) sin(x); %..........function (B) to be tested.
n = 10;           %...........degree of the Lagrange polynomial to be used.
a = -1.0;
b = 1.0;
neval = 1000 ;
xeval = linspace(a,b,neval);

 %------ for equi-distant nodes ------
    x_nodes = linspace(-1.0,1.0,n+1);

 %------ Chebyshev Nodes ------
%  for k=0:n
%      tk = ((2 * n + 1 - 2 * k) * pi) / (2 * n + 2);
%      x_nodes(k + 1) = 0.5 * (b - a) * cos(tk) + 0.5 * (a + b);
%  end

 y_nodes = f(x_nodes) ;
 polyInterp_at_xeval = zeros(size(xeval)) ;
 polyInterp_at_xeval = Lagrange_Interp(y_nodes, x_nodes, xeval);

 %------ Relative Error ------
 max_error = max(abs(polyInterp_at_xeval - f(xeval)));
 max_in_exact = max(abs(f(xeval)));
 Rel_Error = max_error/max_in_exact ;

 fprintf('Maximum error = %d , Relative error = %d. \n', Rel_Error, max_error );
 plot(xeval,f(xeval),'b',xeval,polyInterp_at_xeval,'r',x_nodes,y_nodes,...
      'ko' ,'LineWidth',1)
 % plot(xeval,polyInterp_at_xeval)
 % plot(xeval,f(xeval))
 legend('f(x)','P(x)','Nodes');
 set (gca,'FontSize',10);
 toc
 %-------------------------------------------------------------------------
 %------- Lagrange Interpolation at Chebyshev Points
 %------- Here, our grid will not be uniform
 %------- Take grid all the roots of Chebyshev polynomial
 %------- nth degree Chebyshev poly given by TN(x) = cos(N arccos(x)) for 1  x  1
 %------- Now, nodes are the roots of this polynomial
 %------- xk = cos((2k + 1) / 2N) for k=0, 1, ..., N1
 %------- compute Chebyshev nodes in [-1,1]
 %-------------------------------------------------------------------------

 % for i=0:n
 %     tk=((2*n+1-2*k)*pi)/(2*n+2);
 %     x_nodes(k+1)=0.5*(b-a)cos(tk)+0.5*(a+b);
 % end
```