NumCalc

A numerical calculator app

Students:

Felipe Henao Gómez
Isabella Echeverri Villa
Juan Andrés Montoya Galeano
Thomas Martinod Saldarriaga

Professor: Edwar Samir Posada Murillo

EAFIT University

Informatics and Systems Department

Numerical Analysis

2022

FINAL PROJECT
Second report

Objective: To specify the methods used for solving roots of polynomials and linear system of equations through their pseudocodes, and to show the preliminary results for the test functions.

Numerical Methods

1. Incremental search

```
read function, x0, dx, n
xprev=x0
xact=x0+dx
rootCount=0
for i=1 to n do
  if f(xact)*f(xprev)<0
    There is a root for the function  in xprev, xact
  end
end
if rootcount==0
  No roots were found for the given number of n iterations and step size
end
end
```

Results

There's a root for the function in [-2.5, -2]
There's a root for the function in [-1, -0.5]
There's a root for the function in [0.5, 1]
There's a root for the function in [2, 2.5]
There's a root for the function in [4, 4.5]
There's a root for the function in [5, 5.5]
There's a root for the function in [7, 7.5]
There's a root for the function in [8, 8.5]
There's a root for the function in [10, 10.5]
There's a root for the function in [11.5, 12]
There's a root for the function in [13.5, 14]
There's a root for the function in [14.5, 15]
There's a root for the function in [16.5, 17]
There's a root for the function in [17.5, 18]
There's a root for the function in [19.5, 20]

There's a root for the function in [21, 21.5]
There's a root for the function in [22.5, 23]
There's a root for the function in [24, 24.5]
There's a root for the function in [26, 26.5]
There's a root for the function in [27, 27.5]
There's a root for the function in [29, 29.5]
There's a root for the function in [30, 30.5]
There's a root for the function in [32, 32.5]
There's a root for the function in [33.5, 34]
There's a root for the function in [35, 35.5]
There's a root for the function in [36.5, 37]
There's a root for the function in [38.5, 39]
There's a root for the function in [39.5, 40]
There's a root for the function in [41.5, 42]
There's a root for the function in [43, 43.5]
There's a root for the function in [44.5, 45]
There's a root for the function in [46, 46.5]

2. Bisection Method

```
read function,xi,xs,tolerance, niter
i <- 1
xm <- (xi + xs)/2
fxm <- f(xm)
error = absolute value of xm
while error > tolerance and i<niter and fxm different to 0
   if f(a)*fxm<0
      b=xm
      xm=(a+b)/2
      error=absolute value of xm-a
    else if f(b)*fxm<0
      a=xm
      xm=(a+b)/2
      error=absolute value of xm-a
    end
    fxm=f(xm)
    i=i+1
end
if fxm=0 then
    the root was found with a value of xm
end
else if error<=tolerance then
    an approximation of the root was found
    with a value of xm
```

```
end
if i=n
   The root was not found in the number
   of iterations given
end
```

Results

Iteration	a	xn	b	f(xn)	E
22	0.936404	0.936404466	0.9364047	-6.616005e-08	2.3841857e-07
23	0.9364044	0.936404585	0.9364047	2.8715108e-09	1.192092e-07
24	0.9364044	0.93640452	0.93640458	-3.1644283e-08	5.9604644e-08

An aproximation of the root was found with a value of 0.9364 and an error of 5.9605e-08

3. False position

```
read function, a, b, tolerance,n
i = 1
E = inifnite
fxn = 1
while E>tolerance and i<n and fxn different from 0
   xn=b-f(b)*((b-a)/(f(b)-f(a)))
   fxn=f(xn)
   if f(a)*fxn>0
     E=absolute value of xn-a
     a=xn
   else if f(b)*fxn>0
     E=absolute value of xn-b
     b=xn
   end
   i=i+1
end
if fxn==0
  The root was found with a value of xn
end
if E<= tolerance
  An aproximation of the root was found with a value of xn
end
if i==n
  The root was not found in the number of iterations given
```

```
end
end
```

Results

Iteration	a	xn	b	f(xn)	E
3	0.933940380	0.9364047	0.9365060	8.6782541e-08	0.000101320922984094
4	0.933940	0.9364045	0.936404	1.2815393e-10	1.49641e-07
5	0.933940	0.936404	0.9364045	1.8918200e-13	2.209796e-10

An aproximation of the root was found with a value of 0.9364 and an error of 2.2098e-10

4. Newton method

```
read function, df, x0, tolerance, n
if f(x0)=0
  The inital point given is root
end
xn=x0
fxn=f(xn)
i=1
E=infinite
while E>tolerance and i<n and fxn different from 0
  xprev = xn
  xn=xprev-(f(xprev))/df(xprev))
  E=absolute value of xn-xprev
  fxn=f(xn)
  i++
end
if fxn=0
  The root was found with a value of xn
end
if E<= tolerance
  An aproximation of the root was found with a value of xn
end
if i=n
  The root was not found in the number of iterations given
end
end
```

Results

Iteration	xn	f(xn)	E
2	0.936366741267331	-2.19126198827135e-05	0.00797475135475945
3	0.93640458001899	-4.98339092214195e-10	3.78387516588585e-05
4	0.936404580879562	-1.11022302462516e-16	8.60571947036703e-10

An aproximation of the root was found with a value of 0.9364 and an error of 8.6057e-10

5. Fixed point

```
read function, g, x0, tolerance, n
if f(x0)=0
 The initial point given is the root
end
xn=x0
fxn=f(xn)
gxn=g(xn)
i=1
E=infinite
while E>tolerance and i<n and fxn different from 0
  xprev=xn
  xn=gxn
  E=absolute value of xn-xprev
  fxn=f(xn)
  gxn=g(xn)
  i++
end
if fxn=0
  The root was found with a value of xn
end
if E<= tolerance
  An aproximation of the root was found with a value of xn
end
if i=n
  The root was not found in the number of iterations given
end
end
```

Results

Iteration	xn	g(xn)	f(xn)	E
28	-0.3744451043623	-0.3744449757003	1.286620382457e-07	2.142604523803e-07
29	-0.3744449757003	-0.3744450529611	-7.726074024994e-08	1.286620382456e-07
30	-0.3744450529611	-0.3744450065665	4.639458395239e-08	7.726074024994e-08

An aproximation of the root was found with a value of -0.37445 and an error of 7.7261e-08

6. Secant method

```
read function,x0,x1, tolerance,
if f(x0)=0
The initial point x0 given is the root
end
if f(x1)=0
The initial point x1 given is the root
end
xn=x0
xnext=x1
fxn=f(xnext)
i=2
e=infinite
while e>tolerance and i<n and fxn different from 0
  xprev=xn
  xn=xnext
  xnext=xn-(f(xn)/((f(xn)-f(xprev))/(xn-xprev)))
  e=absolute value of xnext-xn
  fxn=f(xnext)
  i++
end
if fxn=0
  The root was found with a value of xn
end
if E<= tolerance
  An aproximation of the root was found with a value of xn
end
if i=n
  The root was not found in the number of iterations given
end
end
```

Results

Iteration	xn	f(xn)	E
4	0.936407002376704	1.40223589106814e-06	0.000410421585531284
5	0.93640458147312	3.43716499706659e-10	2.42090358437697e-06
6	0.936404580879561	-4.9960036108132e-16	5.93558091566138e-10

An aproximation of the root was found with a value of 0.9364 and an error of 5.9356e-10

7. Simple Gauss Elimination

```
read A,b
Ab=[A b]
[f,c]=size of Ab
for j=1 to c-2
for i=j to f-1
Ab(i+1,j to c)=Ab(i+1,j to c)-(Ab(i+1,j)/Ab(j,j))*Ab(j,j to c)
end
end
end
```

Regresive Sustitution function

```
read A,b
[f,c]<-size of A
x(f)<-b(f)/A(f,f)
for i=f-1 reducing 1 each step to 1
   sum=0
   for j=i+1 to f
      sum=sum+A(i,j)*x(j)
   end
   x(i)=(b(i)-sum)/A(i,i)
end
end
\\
\\
```

Main function

```
read A,b
[U,B]<-Elimination with parameters A,b
x<-Regresive sustitution function with parameters U,B
The solution of the equation is x
```

Results

Stage 2:

2.0000	−1.0000	0	3.0000	1.0000
0	1.0000	3.0000	6.5000	0.5000
0	0	−41.0000	−73.5000	−5.5000
0	0	−38.000	−96.0000	−12.0000

Stage 3:

2.0000	−1.0000	0	3.0000	1.0000
0	1.0000	3.0000	6.5000	0.5000
0	0	−41.0000	−73.5000	−5.5000
0	0	0	−27.878048780487802	−6.902439024390244

Solution:

0.038495188101487	−0.180227471566054	−0.309711286089239	0.247594050743657

8. Gauss elimination with partial pivot

Function of Gaussian elimination with partial pivot (ElimPivPar)

```
read A,b
[f,c]<-size of Ab
for j=1 to c-2
  col<-absolute value of j to f, j
  m<- find maximum in col
  temp<- Ab in row j
  Ab in row j <- Ab in row m+j-1
  Ab in rom m+j-1<- temp
  for i=j to f-1
    Ab in row i+1 and column j to c<-Ab(i+1,j:c)-(Ab(i+1,j)/Ab(j,j))*Ab(j,j:c)
  end
end
end
```

Regresive Sustitution function

```
read A,b
[f,c]<-size of A
x(f)<-b(f)/A(f,f)
for i=f-1 reducing 1 each step to 1
   sum=0
   for j=i+1 to f
      sum=sum+A(i,j)*x(j)
   end
```

```
      x(i)=(b(i)-sum)/A(i,i)
   end
   end
```

Main function

```
read A,b
[U,B]<-function ElimPivPar with parameters A,b
x<-Regresive sustitution function with parameters U,B
The solution of the equation is x
```

Results

Stage 2:

14.000	5.000	−2.000	3.000	1.000
0	13.000	−2.000	11.000	1.000
0	0	3.164835164835165	7.664835164835164	0.917582417582418
0	0	0.021978021978022	4.021978021978022	0.989010989010989

Stage 3:

14.000	5.000	−2.000	3.000	1.000
0	13.000	−2.000	11.000	1.000
0	0	3.164835164835165	7.664835164835164	0.917582417582418
0	0	0	3.96875000	0.982638888888889

Solution:

0.038495188101487 −0.180227471566054 −0.309711286089239 0.247594050743657

9. **Gauss method with total pivot**

Elimination Gauss method with total pivot (ElimPivTot)

```
read A,b
A,b=[A b]
[f c]= size of Ab
tags<-1 to c-1
for j=1 to c-2
   subm <- submatrix of Ab(j to f,j to c-1)
   [mi,mj]<-find maximum between subm,[]
   temp<-Ab(j,j to end)
   Ab(j, j to end)<-Ab(mi+j,j to end)
   Ab(mi+j-1,j to end)<-temp
   temp<-Ab in column j
   Ab in column mj+j-1<-temp
   temp<-tags(j)
   tags(j)<-tags(mj+j-1)
```

```
          tags(mj+j-1)=temp
          for i=j to f-1
              Ab(i+1,j to c)=Ab(i+!,j to c)-(Ab(i+1,j)/Ab(j,j))*Ab(j,j to c)
          end
       end
```

Regresive Sustitution function (solve)

```
read A,b
[f,c]<-size of A
x(f)<-b(f)/A(f,f)
for i=f-1 reducing 1 each step to 1
   sum=0
   for j=i+1 to f
      sum=sum+A(i,j)*x(j)
   end
   x(i)=(b(i)-sum)/A(i,i)
end
end
\\
\\
```

Main function

```
 read A,b
    [U,B,tags]<-ElimPivTot(A,b)
    x<-solve(U,B)
    f<-size of x,2
    xtemp<-[]
    for i=1 to f
      ind<-tags in position i
      xtemp(ind)<-x(i)
    end
    x=xtemp
end
```

Results

Stage 2:

14.0000	5.0000	−2.0000	3.0000	1.0000
0	13.0000	−2.0000	11.0000	1.0000
0	0	3.1648	7.6648	0.9176
0	0.0000	0.0220	4.0220	0.9890

Stage 3:

14.0000	5.0000	3.0000	−2.0000	1.0000
0	13.0000	11.0000	−2.0000	1.0000
0	0	7.6648	3.1648	0.9176
0	0.0000	0	−1.6387	0.5075

Solution:

0.0385 −0.1802 −0.3097 0.2476

10. Multiple roots

```
    read f, df, d2f, x0, tolerance, n
    if f(x0)=0
    The initial point given is the root
    end
    xn=x0
    Fxn=f(xn)
    i=1;
    E=infinite
    while E>tolerance and i<N and Fxn different from 0
        xprev<-xn
        F<-f(xprev);
        dF<-df(xprev)
        d2F<-d2f(xprev)
        xn<-xprev-(F*dF)/((dF^2)-F*d2F)
        Fxn<-f(xn)
        E=absolute value of xn-xant
        i=i++;
    end
    if Fxn=0
        The root was found with a value of xn
        return
    end
    if E<=Tol
        An approximation of the root was found with a value of xn and an error of E
        return
    end
    if i=N
        The root was not found in the number of iterations given
        return
    end
end
```

Results

Iteration	xn	f(xn)	E
2	-0.0084583	3.5671e-05	0.22575
3	-1.189e-05	7.0688e-11	0.0084464
4	-4.2186e-11	0	1.189e-05

The root was found with a value of -4.2186e-11

11. Müller's algorithm

```
read f, x0, x1, x2, tolerance, N

h1 = x1 - x0
h2 = x2 - x1
d1 = (f(x1) - f(x0))/h1
d2 = (f(x2) - f(x1))/h2
d = (d2 - d1)/(h2 + h1)
i = 2

while i < N:
    b = d2 + h2*d
    D = (b^2 - 4*f(x2)*d)^1/2 ----------- from the cuadratic formula

    if |b-D| < |b+D|:
        E = b + d
    else:
        E = b - d

    h = -2*f(x2)/E

    if |h| < tolerance:
        return p, E, i  -------p is the x coordinate for the root and E the
                               i-th iteration error
        break
    else:
        x0 = x1
        x1 = x2
        x2 = p
        h1 = x1 - x0
        h2 = x2 - x1
        d1 = ((f(x1) - f(x2))/h1
        d2 = ((f(x2) - f(x1))/h2
        d = (d2 - d1)/(h2 + h1)
```

```
        i = i + 1
end

print: "The method failed after " + N + " iterations"
```

Results

Iteration	xn	f(xn)	E
6	1.8393	-1.3324e-05	0.001417
7	1.8393	2.0229e-10	2.4357e-06
8	1.8393	2.2204e-16	3.6978e-11

An aproximation of the root was found with a value of 1.8393 and an error of 3.6978e-11

12. Steffensen's algorithm

```
read g (function), p0 (initial value), tolerance, N

i = 1

while i < N:
    p1 = g(p0)
    p2 = g(p1)
    p = p0 - (p1-p0)^2/(p2-2*p1+p0)

    if |p-p0| < tolerance:
        return p ------- p is the x coordinate for the root and E the
                         i-th iteration error
    else:
        i = i + 1
        p0 = p
end

The method failed after N iterations
```

Results

Iteration	xn	f(xn)	E
3	0.93634	-3.6044e-05	0.0081341
4	0.9364	-2.1289e-09	6.2238e-05
5	0.9364	-2.2204e-16	3.6764e-09

An aproximation of the root was found with a value of 0.9364 and an error of 3.6764e-09

13. Aitken's process for accelerating convergence

```
read f, g, x0, tolerance, N

% Initial assignments
xn=x0
Fxn=f(xn)
Gxn=g(xn)
i=1;
E=inf;

while E > tolerance and i < N and Fxn different to 0
    AitkenMod = false
    xant = xn
    xn = Gxn

    % Check mod3 families until we obtain a multiple of 3
    if mod(i,3) == 1
        x1 = xn;
    else if mod(i,3) = 2
        x2 = xn;
    else if mod(i,3) = 0
        xn = xo - ((x1-xo)^2/(x2-2*x1+xo))
        xo = xn;
        AitkenMod = true

    E = abs(xn-xant)
    Fxn = f(xn)
    Gxn = g(xn)
    i=i+1
end


if Fxn == 0
    print: "The root was found with a value of " + xn
    return
if E <= tolerance
    print: "An aproximation of the root was found with a value of " + xn + " and
            an error of " + E
    return
if i == N
    print: "The root was not found in the number of iterations given"
```

```
return
```

Results

Iteration	xn	Aitken	g(xn)	f(xn)	E
8	-0.37444	0	-0.37445	-7.641e-07	1.2724e-06
9	-0.37445	1	-0.37445	-4.9033e-13	4.7741e-07
10	-0.37445	0	-0.37445	2.9454e-13	4.9033e-13

An aproximation of the root was found with a value of -0.37445 and an error of 4.9033e-13

14. Tridiagonal Gaussian Elimination

```
read A,b ---------- % Ax = b system
n = length(b)

% Check if matrix is tridiagonal
for i=1 : n
    for j=1 : n
        aij = A(i,j);
        if i = j or i-1 = j or i+1 = j
            if aij = 0
                "The given matrix is not tridiagonal"
                return
        else
            if aij != 0
                "The given matrix is not tridiagonal"
                return
    end
end

Ab = [A b] --------------- % Augmented A|b matrix
diagp = zeros(n,1)
diagu = zeros(n-1,1)
diagl = zeros(n-1,1)
diagp(1) = A(1,1)

for i=2 : n
    % Extract the elmenents from each diagonal
    diagl(i-1) = Ab(i,i-1)
    diagp(i) = Ab(i,i)
    diagu(i-1) = Ab(i-1,i)
```

```
    % Making the diagonal below zeros
    M = diagl(i-1)/diagp(i-1) ------------ % Multiplier
    diagp(i) = diagp(i)-M*diagu(i-1)
    diagl(i-1) = diagl(i-1)-M*diagp(i-1)
    b(i) =b(i)-M*b(i-1)
    Ab(i,i-1:i+1) = [diagl(i-1) diagp(i) diagu(i-1)]
    Ab(i,end) = b(i)
end

% Substitution
x(n) = b(n)/diagp(n)
for i=n-1 : -1 : 1
    x(i) = (b(i)-diagu(i)*x(i+1))/diagp(i);
end
print: x
```

Results

Stage 0:

$$\begin{matrix} 2.0400 & -1.0000 & 0 & 48.8000 \\ -1.0000 & 2.0400 & -1.0000 & 0.8000 \\ 0 & -1.0000 & 2.0400 & 0.8000 \end{matrix}$$

Stage 2:

$$\begin{matrix} 2.0400 & -1.0000 & 0 & 48.8000 \\ 0 & 1.5498 & -1.0000 & 24.7216 \\ 0 & -1.0000 & 2.0400 & 0.8000 \end{matrix}$$

Stage 3:

$$\begin{matrix} 2.0400 & -1.0000 & 0 & 48.8000 \\ 0 & 1.5498 & -1.0000 & 24.7216 \\ 0 & 0 & 1.3948 & 16.7514 \end{matrix}$$

Solution:

$$\begin{matrix} 35.5397 & 23.7010 & 12.0103 \end{matrix}$$

The matrix given in Microsoft Teams to test Elimination Methods came out as not Tridiagonal.

15. Trisection

```
    read f,a,b,Tol,N
```

```
if a>=b
    The given interval is not valid, a is greater or equal to b
    return
end
if f(a) is 0
    The root is the given value for a
    return
end
if f(b) is 0
    The root is the given value for b
    return
end
if f(a)*f(b)>0
    There is no root in the given interval
    return
end
i=1;
xm1=(2*a+b)/3; %mid point 1
xm2=(2*b+a)/3; %mid point 2
Fxm1=f(xm1);
Fxm2=f(xm2);
if absolute value of Fxm1<= absolute value of Fxm2
    xm=xm1;
    Fxm=Fxm1;
else
    xm=xm2;
    Fxm=Fxm2;
end
E=absolute value of xm;
while E>Tol and i<N and Fxm1 different to 0
    if f(a)*Fxm1<0
        b=xm1;
    elseif Fxm1*Fxm2<0
        a=xm1;
        b=xm2;
    else
        a=xm2;
    end
    xm1=(2*a+b)/3
    xm2=(2*b+a)/3
    Fxm1=f(xm1)
    Fxm2=f(xm2)
    xant=xm
```

```
        if absolute value of Fxm1<= absolute value of Fxm2
            xm=xm1
            Fxm=Fxm1
        else
            xm=xm2
            Fxm=Fxm2
        end
        E=absolute value of xm-xant
        i=i+1
    end
    if Fxm==0
        The root was found with a value of xm
        return
    end
    if E<=Tol
        An approximation of the root was found with a value of xm and an error of E
        return
    end
    if i==N
        The root was not found in the number of iterations given
        return
    end
end
```

Results

Iteration	a	xn	b	f(xn)	E
13	0.93640310025	0.9364043547	0.9364049819	-1.3098e-07	6.2723e-07
14	0.93640435470	0.9364045637	0.9364049819	-9.9042e-09	2.0908e-07
15	0.93640456377	0.93640463346	0.9364047728	3.0453e-08	6.9692e-08

An aproximation of the root was found with a value of 0.9364 and an error of 6.9692e-08

16. Jacobi

```
function read A,b,x0,p,Tol,N
if det(A)==0
    error
D<- diagonal of A
L<- -lower triangular part of A +D
U<- -upper triangular part of A +D
T<- inverse matrix of D *(L+U)
```

```
    C<-inverse matrix of D *b
    ro<- maximum of |eigenvalues of T| %spectral radius of iteration matrix
    x<-x0
    i<-0
    E<-infinite
    while E>Tol and i<N
        xprev<-x
        x<-T*xprev+C
        E=p_norm of x-xprev
        i<-i+1
    end
end
```

Results

Iteration	Error	x1	x2	x3	x4
50	1.5846e-07	0.52511	0.25546	-0.41048	-0.28166
51	1.1941e-07	0.52511	0.25546	-0.41048	-0.28166
52	8.9974e-08	0.52511	0.25546	-0.41048	-0.28166

Answer
0.5251
0.2555
-0.4105
-0.2817

17. Crout Factorization

```
function read A
[f,c]=size of A
L=matrix of zeros with dimensions f,c
U=identity matrix  of f dimension
L(:,1)=A(:,1)
U(1,2 to end)=A(1,2 to end)/L(1,1)
for j=2 to c
    for i=2 to f
        if i>=j
            L(i,j)= A(i,j)- [(L(i,1 to i-1) x
            transpose matrix of U(1 to i-1,j))]
        else
            U(i,j)=(A(i,j)- [(L(i,1:j-1) x U(1:j-1,j)'))/L(i,i)]
        end
```

```
        end
    end
end
```

Results

Lower Triangular Matrix L

4.0000	0.0000	0.0000	0.0000
1.0000	15.7500	0.0000	0.0000
0.0000	−1.3000	−3.7524	0.0000
14.0000	8.5000	−3.6190	13.9492

Upper Triangular Matrix U

1.0000	−0.2500	0.0000	0.7500
0.0000	1.0000	0.1905	0.4603
0.0000	0.0000	1.0000	−0.4526
0.0000	0.0000	0.0000	1.0000

Progressive substitution Lz=b
0.2500 0.0476 -0.2830 -0.2817

Regresive substitution Ux=z, solution
0.5251 0.2555 -0.4105 -0.2817

18. **Gauss Seidel Method**

```
        function read A,b,x0,p,Tol,N
    if det(A)==0
        error
    D<- diagonal of A
    L<- -lower triangular part of A +D
    U<- -upper triangular part of A +D
    T<- inverse matrix of D *(L+U)
    C<-inverse matrix of D *b
    ro<- maximum of |eigenvalues of T| %spectral radius of iteration matrix
    x<-x0
    i<-0
    E<-infinite
    while E>Tol and i<N
        xprev<-x
        x<-T*xprev+C
        E<-p_norm of x-xprev
        i<-i+1
    end
```

```
end
```

Results

Iteration	Error	x1	x2	x3	x4
28	2.7736e-07	0.52511	0.25546	-0.41048	-0.28166
29	1.6628e-07	0.52511	0.25546	-0.41048	-0.28166
30	9.968e-08	0.52511	0.25546	-0.41048	-0.28166

Answer
0.5251
0.2555
-0.4105
-0.2817

19. Doolittle Factorization

```
    function read A
    [f,c]=size of A
    L= identity matrix of dimension f
    U=zero matrix of dimensions f,c
    for j=1 to c
        for i=1 to f
            if i<=j
                U(i,j)=A(i,j)
                U(i,j)=U(i,j)-[(L(i,1:i-1) x Transpose matrix of U(1:i-1,j))]
            else
                L(i,j)=A(i,j)
                L(i,j)=L(i,j)-[(L(i,1:j-1) x Transpose matrix of U(1:j-1,j)]
                L(i,j)=L(i,j)/U(j,j)
            end
        end
    end
end
```

Results

Lower Triangular Matrix L

1.0000	0.0000	0.0000	0.0000
0.2500	1.0000	0.0000	0.0000
0.0000	−0.0825	1.0000	0.0000
3.5000	0.5397	0.9645	1.0000

Upper Triangular Matrix U

4.0000	−1.0000	0.0000	3.0000
0.0000	15.7500	3.0000	7.2500
0.0000	0.0000	−3.7500	1.6984
0.0000	0.0000	0.0000	13.9492

Progressive substitution Lz=b
1.0000 0.7500 1.0619 -3.9289

Regresive substitution Ux=z, solution
0.5251 0.2555 -0.4105 -0.2817

20. Cholesky Factorization

```
    function read A
    [f,c]<-size of A
    L<-zero matrix of dimenssions f,c
    U<-zero matrix of dimenssions f,c
    L(1,1)<-square root of A(1,1)
    U(1,1)<-L(1,1);
    L(2 to end,1)<-A(2 to end,1)/L(1,1)
    U(1,2:end)<-A(1,2:end)/L(1,1)
    for j=2 to c
     for i=2 to f
      if i>j
       L(i,j)<-(A(i,j)-[(L(i,1:j-1) x transpose matrix of U(1:j-1,j)]/L(j,j)
      else if i=j
       L(i,i)<-square root of (A(i,i)-[(L(i,1:j-1) x transpose matrix of U(1:j-1,i)
       U(i,i)<-L(i,i)
      else
       U(i,j)<-(A(i,j)-[(L(i,1:j-1) x transpose matrix of U(1:j-1,j)]/L(i,i)
      end
      end
    end
end
```

Results

Lower Triangular Matrix L

$$\begin{matrix} 2.0000 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 0.0000i \\ 0.5000 + 0.0000i & 3.9686 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 0.0000i \\ 0.0000 + 0.0000i & -0.3276 + 0.0000i & 0.0000 + 1.9371i & 0.0000 + 0.0000i \\ 7.0000 + 0.0000i & 2.1418 + 0.0000i & 0.0000 + 1.8683i & 3.7349 + 0.0000i \end{matrix}$$

Upper Triangular Matrix U

$$\begin{matrix} 2.0000 + 0.0000i & -0.5000 + 0.0000i & 0.0000 + 0.0000i & 1.5000 + 0.0000i \\ 0.0000 + 0.0000i & 3.9686 + 0.0000i & 0.7559 + 0.0000i & 1.8268 + 0.0000i \\ 0.0000 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 1.9371i & 0.0000 - 0.8768i \\ 0.0000 + 0.0000i & 0.0000 + 0.0000i & 0.0000 + 0.0000i & 3.7349 + 0.0000i \end{matrix}$$

Progressive substitution Lz=b
0.5000 + 0.0000i 0.1890 + 0.0000i 0.0000 - 0.5482i -1.0520 + 0.0000i

Regresive substitution Ux=z, solution
0.4982 0.1477 0.1555 -0.2817

21. SOR

```
function read A,b,x0,p,w,Tol,N
if determinant of A=0
    error
D<-diagonal of A
L<- -lower triangular part of A +D
U<- -upper triangular part of A +D
T<-inverse matrix of (D-w*L) * ((1-w)*D+w*U)
C<-w*inverse matrix of (D-w*L)*b
ro<-spectral radius of t
x<-x0
i<-0
E<-inf
while E>Tol and i<N
    xprev<-x
    x<-T*xprev+C;
    E<-p_norm x-xprev
    i<-i+1
end
```

```
end
```

Results

Iteration	Error	x1	x2	x3	x4
33	1.8071e-07	0.52511	0.25546	-0.41048	-0.28166
34	1.106e-07	0.52511	0.25546	-0.41048	-0.28166
35	5.9459e-08	0.52511	0.25546	-0.41048	-0.28166

Answer
0.5251
0.2555
-0.4105
-0.2817

Progressive Substitution

```
    function read L,B
    f=rows of L
    x=zero matrix of dimensions 1,f
    x(1)=B(1)/L(1,1)
    for i=2 to f
        sum=0
        for j=1 to i
            sum=sum+L(i,j)*x(j)
        end
        x(i)=(B(i)-sum)/L(i,i)
    end
end
```

22. LU Factorization Partial Pivot

```
function read A
[f,c]<-size of A
L<-identity matrix of dimension f
P<- identity matrix of dimension f
for j=1 to c-1
    col<-|A(j:f,j)|
    m<- maximum of col
    m<-m(1)
    row change of A and P
    if j>1
        rows and columns change of L
```

```
        end
        for i=j to f-1
            Mi<-A(i+1,j)/A(j,j)
            A(i+1,j:c)<-A(i+1,j:c)-(Mij)*A(j,j to c);
            L(i+1,j)=Mij
        end
    end
    U<-A
end
```

Results

Lower Triangular Matrix L

1.0000	0.0000	0.0000	0.0000
0.2500	1.0000	0.0000	0.0000
0.0000	−0.0825	1.0000	0.0000
3.5000	0.5397	0.9645	1.0000

Upper Triangular Matrix U

4.0000	−1.0000	0.0000	3.0000
0.0000	15.7500	3.0000	7.2500
0.0000	0.0000	−3.7500	1.6984
0.0000	0.0000	0.0000	13.9492

Vector Pb

1.0000
1.0000
1.0000
1.0000

Progressive substitution Lz=Pb
1.0000 0.9286 1.0797 1.1745

Regresive substitution Ux=z, solution
0.5251 0.2555 -0.4105 -0.2817

23. LU Factorization

```
    function read A
    [f,c]=size of A
    L=identity matrix of dimension f
    for j=1 to c-1
        for i=j to f-1
            Mij=A(i+1,j)/A(j,j)
            A(i+1,j to c)=A(i+1,j to c)-(Mij)*A(j,j to c)
            L(i+1,j)=Mij
        end
    U=A
end
```

Results

Lower Triangular Matrix L

$$\begin{matrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.2500 & 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & -0.0825 & 1.0000 & 0.0000 \\ 3.5000 & 0.5397 & 0.9645 & 1.0000 \end{matrix}$$

Upper Triangular Matrix U

$$\begin{matrix} 4.0000 & -1.0000 & 0.0000 & 3.0000 \\ 0.0000 & 15.7500 & 3.0000 & 7.2500 \\ 0.0000 & 0.0000 & -3.7500 & 1.6984 \\ 0.0000 & 0.0000 & 0.0000 & 13.9492 \end{matrix}$$

Progressive substitution Lz=b
1.0000 0.7500 1.0619 -3.9289

Regresive substitution Ux=z, solution
0.5251 0.2555 -0.4105 -0.2817

24. Lagrange Interpolation Method

```
begin
```

```
    Read the number of points (n)

    for i = 1 to n
        Read (x_i, y_i) pairs -------------- y_i = f(x_i)
    end

    Read xp
    Initialize: yp = 0 --------------------- yp = f(xp)

    For i = 1 to n
        p = 1
        For j =1 to n
            If i  j
                p = p * (x_p - x_j)/(x_i - x_j)
            End If
        Next j
        yp = yp + p*y_i
    End

    print yp ------------------------------- Lagrange polynomial at xp
end
```

For the Lagrange polynomial, the output would be the expression,

$$\sum_{i=0}^{n} L_i(x_i) \tag{1}$$

where, for each iteration,

$$L_i(x) = \prod_{p} \frac{x - x_p}{x_i - x_p} \tag{2}$$

Results

Iteration	Li(x)
0	-0.05000*(x) (x - 3.00000) (x - 4.00000)
1	0.08333*(x + 1.00000) (x - 3.00000) (x - 4.00000)
2	-0.08333*(x + 1.00000) (x) (x - 4.00000)
3	0.05000*(x + 1.00000) (x) (x - 3.00000)

Polynomial Coefficients
-0.77500
0.25000
-0.66667
0.05000

Polynomial
-0.775*(x)*(x - 3.00000)*(x - 4.00000) + 0.25*(x + 1.00000) (x - 3.00000) (x - 4.00000) - 0.66667*(x + 1.00000) (x) (x - 4.00000)

25. Vandermonde's matrix method

```
begin
    input X, b ------ X = (x1, x2, ..., xn), and Y = (f(x1), f(x2), ..., f(xn))

    initialize deg = length(X)
    initialize a_i = zeros(deg, 1) ---- The vector of the coefficients


    For i = 0 < degree, i++
        For j = 0; j < degree; j++
            A[i][j] = x[i]^-(j+1) -------- Vandermonde's Matrix
        end
    end

    output A, b

    Use this output to solve the system of equaions A*a_1 = b with whatever
    method you prefer.

    sol = GaussianElimination (A, b)
end
```

Results

Polynomial Coefficients
-1.14167
5.82500
-5.53333
3.00000

Polynomial
-1.14167x$^3 + 5.82500x^2 - 5.53333x^1 + 3.00000$

Newton's Method
Divided differences method

26.

```
begin
    input (x0, f(x0)), (x1, f(x1)), ..., (xn, f(xn))

    For i = 0, 1, ..., n
        F_i,0 = f(xi)
    end

    For i = 1, ..., n
        For  j = 1, ..., i
            set F_i,j = (F_i,j-1 - F_i-1,j-1)/(xi - xi-j)
        end
    end

    output F_0,0 , ..., F_i,i, ..., F_n,n
end
```

The output $F_{0,0}, ..., F_{i,i}, ..., F_{n,n}$ can be translated into the expression,

$$P(x) = \sum_{x=0}^{n} F_{i,i} * \prod_{j=0}^{i-1}(x - x_j) \tag{3}$$

where $P(x)$ is the Newton's polynomial.

Results

Xi	f(Xi)	1	2	3
-1	15.5	0	0	0
0	3	-12.5	0	0
3	8	1.6667	3.5417	0
4	1	-7	-2.1667	-1.1417

Polynomial Coefficients
15.50000
-12.50000
3.54167
-1.14167

Polynomial
15.50000 - 12.50000(x + 1.00000) + 3.54167(x + 1.00000)(x) - 1.14167(x + 1.00000)(x)(x - 3.00000)

Splines
1st, 2nd, and 3rd degree

```
begin
    input (X0, f(X0)), (X1, f(X1)), ..., (Xn, f(Xn))
    input degree

    if degree == 1
        for i = 0, i = n-1
            M_i = (f(X_i+1) - f(X_i))/(X_i+1 - X_i)
            return p(X) = f(X_i+1) - f(X_i) = M_i * (X - X_i)
        end
    end

    if degree == 2
        for i = 1, i = n
            p(X) = A_i*X_i^2 - B_i*X_i + C_i
        end
        for i = 2, i = n
            p(X) = 2*A_i-1*X_i-1 - B_i-1
        end
        To be natural spline both p(X) must be equal to 0 in both cases
        return p(X)
    end

    if degree == 3
        for i = 1, i < n
            p(X) = A_i*X_i^3 + B_i*X_i^2 + C_i*X_i + D_i
        end
        for i = 2, i = n
            p(X) = 3*A_i-1*X_i-1^2 + 2*B_i-1*X_i-1 + C_i-1
        end
        for i = 3, i = n
            p(X) = 6*A_i-1*X_0 + 2*B_i-1 = 0
        end
        To be natural spline, it must follow that
            - 6*A_i-1*X_0 + B_i-1 = 0
            - 6*A_i*X_n + B_i = 0
        return p(X)
    end
end
```

Results

Linear

Polynomial	Coefficient a	Coefficient b
0	-12.5	3
1	1.6667	3
2	-7	29

Polynomial	Spline
0	-12.50000x + 3.00000
1	1.66667x + 3.00000
2	-7.00000x + 29.00000

Cuadratic

Polynomial	Coefficient a	Coefficient b	Coefficient c
0	0	-12.5	3
1	4.7222	-12.5	3
2	-22.833	152.83	-245

Polynomial	Spline
0	$0.00000\text{x}^2 - 12.50000x + 3.00000$
1	$4.72222\text{x}^2 - 12.50000x + 3.00000$
2	$-22.83333\text{x}^2 + 152.83333x - 245.00000$

Cubic

Polynomial	Coefficient a	Coefficient b	Coefficient c	Coefficient d
0	2.5333	7.6	-7.4333	3
1	-1.5222	7.6	-7.4333	3
2	2.0333	-24.4	88.567	-93

Polynomial	Spline
0	$2.53333\text{x}^3 + 7.60000x^2 - 7.43333x + 3.00000$
1	$-1.52222\text{x}^3 + 7.60000x^2 - 7.43333x + 3.00000$
2	$2.03333\text{x}^3 - 24.40000x^2 + 88.56667x - 93.00000$

Composite trapezoidal rule

```
function read f,a,b,n
h=(b-a)/n
I=f(a)+2*f(a+h)+2*f(a+2h)+...+f(b)
I=(h/2)*I
return I
```

```
end
```

Results Using $f(x) = x * sin(x)$, $a = 3$, $b = 10$ and $N = 100$, the result is:
The approximated value of the integral of f(x) from a to b is 4.73310320

Composite Simpson's 1/3 rule

```
    function read f,a,b,n
    Check if n is an even number, it must be even to continue
    h=(b-a)/n
    I=f(a)+2*f(a+h)+4*f(a+2h)+2*f(a+3h)+4*f(a+4h)+...+f(b)
    I=(h/3)*I
    return I
end
```

Results Using $f(x) = x * sin(x)$, $a = 3$, $b = 10$ and $N = 100$, the result is:
The approximated value of the integral of f(x) from a to b is 4.73559768

Simple Simpson's 3/8 rule

```
    function read f,a,b
    h=(b-a)/n
    x1=(2*a+b)/3
    x2=(a+2*b)/3
    I=f(a)+3*f(x1)+3*f(x2)+f(b)
    I=(3*h/8)*I
    return I
end
```

Results Using $f(x) = x * sin(x)$, $a = 3$, $b = 10$ and $N = 100$, the result is:
The approximated value of the integral of f(x) from a to b is 3.99661303

Euler's Method

```
    function read f,t0,tn,y0,h
    n=ceiling of |tn-t0|/h
    t=t0 to tn with step size h
    y= column vector of n zeros
    y in position 1=y0
    calculate each y(i) with y(i)=y(i-1)+h*f(t(i-1),y(i-1)) until having n points
    return t, y
end
```

Results

t_i	y_i
0	0
0.25	0.25
0.5	0.54688
0.75	0.87109
1	1.1982
1.25	1.4978
1.5	1.7316
1.75	1.852
2	1.7994
2.25	1.4993
2.5	0.85847
2.75	-0.23942
3	-1.9399

Modified Euler's Method (Heun's Method)

```
    function read f,t0,tn,y0,h
    n=ceiling of |tn-t0|/h
    t=t0 to tn with step size h
    y<- column vector of n zeros
    y in position 1<-y0

    calculate each y(i) with
        yest<-y(i-1)+h*f(t(i-1),y(i-1))     (this is euler's method)
        k1=f(t(i-1),y(i-1))
        k2=f(t(i-1)+h,yest)
        y(i)=y(i-1)+(h/2)*(k1+k2)
    until having n points
    return t, y
end
```

Results

t_i	y_i
0	0
0.25	0.27344
0.5	0.59058
0.75	0.92855
1	1.2581
1.25	1.5416
1.5	1.731
1.75	1.7647
2	1.5638
2.25	1.0271
2.5	0.024903
2.75	-1.6087
3	-4.0866