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. Metodo de la biseccion

```
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++
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:

0.0384951881014587	−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 aproximation of the root was found with a value of xn and an error of E
        return
    end
    if i==N
        disp("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"
```

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

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

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 != 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
```

14. Trisection Method

```
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
                print: "The given matrix is not tridiagonal"
                return
        else
            if aij != 0
                print: "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
```