EAFIT UNIVERSITY
IT AND SYSTEMS DEPARTMENT
CHOICE OF THE PROJECT
Third report

Objective: Implementation of the numerical methods seen until this point in the semester, including the additional ones.

Course: Numerical analysis.
Teacher: Edwar Samir Posada Murillo.
Semester: 2020-2.
Due date for the Third report:Wednesday November 11.
Project name: Numerical views.
Project Repository: Link Repo

Members:

Mariana Ramírez Duque (marami21@eafit.edu.co)

Nicolás Roldán Ramírez (nroldanr@eafit.edu.co)

Mateo Sánchez Toro (msanchezt@eafit.edu.co)

Maria Cristina Castrillon (Mcastri6@eafit.edu.co)

Report description:We will write the pseudocode of each method followed by an example execution. The code can be found on the repository link cited above

1 Methods:

Gaussian Elimination with partial pivot
Gaussian Elimination with total pivot

Fixed Point

Algorithm 1: Incremental search

```
input  : f: function to find root of,
         X0: first root approximation,
         Delta: delta(x),
         N: maximum number of iterations,

if (y=0) then
|   x0 is root;
else
|   x1 ← [x0 delta]
|   y1 ← [f(x1)]
└   counter ← [1]
while y1 * y0 > 0 and counter < n do do
|   x0 ← [x1]
|   y0 ← [y1]
|   x1 ← [x0 + delta]
|   y1 ← [f(x1)]
└   counter ← counter + 1
if (y1==0) then
|   x1 is root;
else
|   if (y1 * y0 < 0) then
|   |   [x0,x1] define an interval;
|   else
|   └   print("didn't find interval in N iterations")
└
```

Algorithm 2: Bisection

```
input : xi: lower limit of the interval,
        xs: the upper range,
        f: function to find,
        N: maximum number of iterations,
        tol: tolerance,
yi ← f(xi)
ys ← f(xs)
if (yi==0) then
|  print("xi is root")
else
|  if (ys==0) then
|  |  print("xs is root")
else
|  if (yi * ys > 0) then
|  |  print("Inappropriate range")
else
|  xm ← (xi + xs)/2
|  ym ← f(xm)
|  error ← tol + 1
|  counter ← [1]
while ymf = 0 and (error > tol)and(contador < n do do
(yi * ym > 0)  xi ← xm, yi ← ym
else
|  xs ← xm, ys ← ym
|  xaux ← xm
|  xm ← xi + xs/2
|  ym ← f(xm)
|  error ← xm − xaux
|  counter ← +1
if (ym == 0) then
|  print("xm is root")
else
|  if (error < tol) then
|  |  print("Xm is root with an error equal to error")
else
|  print("The method failed")
```

Algorithm 3: False Rule

```
input  :  a: lower limit of the interval,
          b: the upper range,
          f: function to find,
          N: maximum number of iterations,
          tol: tolerance,

ya ← f(a)
yb ← f(b)
if (ya==0) then
|  print("a is root")
else
|  if (yb==0) then
|  |  print("b is root")
else
|  if (ya * yb > 0) then
|  |  print("Inappropriate range")
else
|  p ← a − f(a) * (b − a)/f(b) − f(a)
|  yp ← f(p)
|  error ← tol + 1
|  counter ← [1]
while  ypf = 0 and (error > tol)and(contador < n do do
(ya * yp > 0)  a ← p, ya ← yp
else
|  b ← p, yb ← yp
|  paux ← p
|  p ← a − f(a) * (b − a)/f(b) − f(a)
|  yp ← f(p)
|  error ← p − paux
|  counter ← +1
if (yp == 0) then
|  print("p is root")
else
|  if (error < tol) then
|  |  print("p is root with an error equal to error")
else
|  print("The method failed")
```

Algorithm 4: Multiple roots

input : f: function to find root of,
fprime: derivative of f,
f2prime: second derivative of f,
tol: error tolerance,
N: maximum number of iterations,
X0: first root approximation
output: root approximation x

x $\leftarrow x0$
fun $\leftarrow (x)$
funprime $\leftarrow fprime(x)$
fun2prime $\leftarrow f2prime(x)$
error $\leftarrow infinity$

for $i \leftarrow 0$ **to** N **do**
    if *error* $\leq$ *tol(i,i) = 0* **then**
        break;
    error $\leftarrow xx \leftarrow x0$
    fun $\leftarrow f(x)$
    funprime $\leftarrow fprime(x)$
    fun2prime $\leftarrow f2prime(x)$
    error $\leftarrow infinity$
    error $\leftarrow abs(error - x)$

Algorithm 5: Gaussian elimination

input : nxn matrix A, column vector b
output: solution vector x

if *(A is not square) or (size of A and size of b are not computable)* **then**
    break ;
if *det(A) = 0* **then**
    break ;
$A \leftarrow [A\ b]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if *$\bar{A}$(i,i) = 0* **then**
        fund l such that $\bar{A}$(l,i) $\neq 0$;
        swith $\bar{A}$(i) and $\bar{A}$(l)
    for $j \leftarrow i+1$ **to** n **do**
        multiplier $M_{j,i} \leftarrow \frac{\bar{A}(j,i)}{\bar{A}(i,i)}$
        $\bar{A}_j \leftarrow \bar{A}_J - M_{j,i} * \bar{A}_i$

$x \leftarrow$ susreg($\bar{(}A)$)

Algorithm 6: Gaussian elimination with partial pivot

```
input  : square n x n matrix A, n vector b
output: solution vector x
if det(A) == 0 then
    break ;
for i ← 1 to n − 1 do
    champion ← i
    for j ← champion + 1 to n − 1 do
        if abs(A(j,i)) == abs(A(champion,i)) then
            champion ← i
    Swap row in A(champion) with A(i)
    Swap value in b(champion) with b(i)
    for j ← i + 1 to n − 1 do
        multiplicand ← A(j,i) / A(i,i)
        A_j ← A_J − M_{j,i} ∗ A_i
x ← susreg(A)
```

Algorithm 7: Gaussian elimination with total pivot

```
input  : square n x n matrix A, n vector b
output: solution vector x
if det(A) == 0 then
    break ;
posStamp ← [from0ton − 1]for i ← 1 to n − 1 do
    row ← i
    col ← i
    for j ← row to n − 1 do
        for k ← col to n − 1 do
            if abs(A(i,j)) == abs(A(row,col)) then
                row ← i col ← j
    if col != i then
        Swap column A(,col) with A(,i) Swap value in posStamp(col) with
         posStamp(i)
    if col != i then
        Swap row A(row) with A(i) Swap value in b(row) with b(i)
    for j ← i + 1 to n − 1 do
        multiplicand ← A(j,i) / A(i,i)
        A_j ← A_J − M_{j,i} ∗ A_i
x ← susreg(A)
Sort x using posStamp
```

Algorithm 8: Fixed Point

input : f(x), g(x), x0, tol, N
output: The approximate root

```
Define variables
int i = 0;
double x = 0;
Start While (AbsoluteValue (f (x))¿ = tol  i ¡= N)
x = g(x);
i ++;
End While
Start if (AbsoluteValue (f (x)) ¡1E-8)
print "Root:" + x;
else
print "Not possible to obtain root"
End if
```

Algorithm 9: Secant

input : iter, xi, f(xi),E
output: The approximate root

```
I = f (Xo)
If I = 0 Then
Show: "Xo is Root"
Y1 = f (X1)
Counter = 0
Error = Tolerance + 1
While
X2 = X1 - ((Y1 * (X1 Xo)) / Den)
Error = Abs ((X2 - X1) / X2)
Xo = X1
I = Y1
X1 = X2
Y1 = f (X1)
Counter = Counter + 1
End while
If Y1 = 0 Then
Show: "X1 is Root"
Otherwise If Error ¡Tolerance Then
Show: " X1 'is an approximate root with a tolerance' Tolerance "
Otherwise If Den = 0 Then
Show: 'There is possibly a multiple root'
```

Secant
Newton

Algorithm 10: Newton

```
input  : f(x), g(x), x0, tol, N
output: The approximate root
if fx == 0:
return the root
end if
if dfun == 0:
return Error, derivative cannot be 0.
end if
counter = 0
error = tolerance + 1
while error¿ tol and fx! = 0 and dfx! = 0 and counter ¡iterations:
xn = xi - (fx / dfx)
fx = fun.evaluate2 (xn)
dfx = dfun.evaluate2 (xn)
end while
if type_error == 0 :
error = — xn-xi —
else:
error = — (xn-xi) / xn —
end if
xi = xn
counter +1
if fx == 0:
return xi is root
elif error ¡tol:
return xn is an approximation to a root with a tolerance ”
elif dfx == 0:
return xn is a possible multiple root
else:
return The method failed in n iterations
end if
```

Extra method Aikten

Extra method Steffensen

Algorithm 11: Extra method Aikten

input : function f, float tolerance, integer maxIterations
output: solution vector x

```
x ← f(1)
x1 ← f(2)
x2 ← f(3)
aikten1 ← 1 aikten2 ← 0
for i ← 1 to maxIterations do
    if abs(aikten1 - aikten0) ¡= tolerance then
        break ;
    aikten1 ← aikten2 aikten2 ← aiktkenEcuation(x, x1, x2)
    x ← x1 x1 ← x2 x2 ← f(i + 3)
```

Algorithm 12: Extra method Steffensen

input : function f, float tolerance, integer maxIterations, float approximation
output: root of f

```
x0 ← approximation
x1 ← f(x0)
x2 ← f(x1)
x3 ← aiktkenEcuation(x0, x1, x2)
for i ← 1 to maxIterations do
    if abs(x0 - x3) ¡= tolerance then
        break ;
    x0 ← x1
    x1 ← x2
    x2 ← f(i + 3)
    x3 ← aiktkenEcuation(x0, x1, x2)
x ← x3
```

Algorithm 13: Extra method Muller

input : function f, root approximation 1 x0,root approximation 2 x1 , root approximation 3 x2, tolerance tol, Maximum number of iterations N

output: root approximation

$h1 = x1 - x0$
$h2 = x2 - x1$
$t1 = (f(x1) - f(x0))/h1$
$t2 = (f(x2) - f(x1))/h2$
$d = (t2 - t1)/(h2 + h1)$
for $i \leftarrow 3$ **to** N **do**
    $b = t2 + h2 * d$
    $D = (b^2 - 4 * f(x2) * d)^{1/2}$
    if $abs(b - D) < abs(b + D)$ **then**
        $E = b + D$;
    else
        $E = b - D$
    $h = (-2 * f(x2))/E$
    $e = x$
    $x = x2 + h$
    $e = abs(e - x)$
    if $abs(e) \leq tol)$ **then**
        break;
    $x0 = x1$
    $x1 = x2$
    $x2 = x$
    $h1 = x1 - x0$
    $h2 = x2 - x1$
    $t1 = (f(x1) - f(x0))/h1$
    $t2 = (f(x2) - f(x1))/h2$
    $d = (t2 - t1)/(h2 + h1)$

Vandermonde

Algorithm 14: Vandermonde

input : x, y

output: Polynomial coefficients, Vandermonde polynom

Repeat
Repeat
A(j,i)=X(j)**(n-i)
until($j < n$)
until($i < n$)
print: (Reverse A)*y

Newton Divided difference

Algorithm 15: Newton Divided difference

input : vector x0, x1, ...xn, vector values f(x0); a point to evaluate
output: Divided differences F0,0 ...Fn,n

```
Step 1
for i=0,...,n do
    Set Fi,0= f(Xi)
Step 2
for i=1,...,n do
    for j=1,2,....i do
        Set Fi,j= ((Fi,j-1) -(Fi-1,j-1))/(Xi-Xi-j)
    End ;
Output (F0,0...,Fi,i,...Fn,n)
STOP
```

Algorithm 16: Lagrange

input : vector x0, x1, ...xn, vector values f(x0); a point to evaluate
output: P Lagrange polynomial P(x) evaluated at z

```
Step 1
Initialize variables. Set P z equal zero. Set n to the number of pairs of points (x,
 y). Set L to be the all ones vector of length n
Step 2
for i=1,2,....n do
    Step 3
    for j=1,2,....i do
        Step 4
        if i = j then Li=(z-Xj)/(Xi-Xj)*Li then
            break ;
        Step 5
Pz= (Li * yi + Pz))
Step 6 Output Pz. STOP
```

Lagrange
Neville

Algorithm 17: Neville

---

input : Numbers x0, x1, ...xn, values f(x0),f(x1)...f(xn)
output: the table Q with p(x)=Qn,n

Step 1
for *i=1,2,....n* **do**
  for *j=1,2,....i* **do**

Set Q(ij)=((x-xi-j)Qi,j-1 - (x-xi)Qi-1,j-1)/(xi-xi-j)
Step 2
Output(Q);
STOP

Simple LU

Algorithm 18: Simple LU

---

input : nxn matrix A, column vector b
output: solution vector x

if *(A is not square) or (size of A and size of b are not computable)* **then**
  break ;
if *det(A) = 0* **then**
  break ;
$A \leftarrow LU$
$z \leftarrow \ progresiveSustitution([L\ b])$
$x \leftarrow \ RegressiveSustitution([U\ z])$

Partial Pivoting LU
SOR
Gauss-Seidel
Jacobi

2 Pruebas

Incremental search

Algorithm 19: Partial Pivoting LU

input : nxn matrix A, column vector b
output: solution vector x

if *(A is not square) or (size of A and size of b are not computable)* **then**
  break ;
if *det(A) = 0* **then**
  break ;
$PA \leftarrow LU$
$z \leftarrow \ progresiveSustitution([L \ \ P * b])$
$x \leftarrow \ RegressiveSustitution([U \ z])$

Algorithm 20: SOR

input : nxn matrix A, column vector b, initial aproximation X0, weighing factor w, tolerance, maximum interations Nmax
output: solution vector x, final number of iterations iter, error err

$A \leftarrow D - L - U$
$T \leftarrow (D - w * L)^{-1} * ((1 - w) * D + w * U)$
$C \leftarrow w * (D - w * L)^{-1} * b$
$xant \leftarrow x0$
$count \leftarrow 0$
$error = 100$
while $error > tolerance \ and \ counter < nmax$ **do**
  $xact = T * xant + C$
  $error = norm(xant - xact)$
  $xant = xact;$
  $cont = cont + 1;$
$x = xact$
$iter = cont$
$err = error$

Algorithm 21: Gauss-Seidel

input : nxn matrix A, column vector b, initial aproximation X0, weighing factor w, tolerance, maximum interations Nmax

output: solution vector x, final number of iterations iter, error err

$A \leftarrow D - L - U$
$T \leftarrow (D - L)^{-1} * U)$
$C \leftarrow (D - L)^{-1} * b)$
$xant \leftarrow x0$
$count \leftarrow 0$
$error = 100$
while $error > tolerance$ *and* $counter < nmax$ **do**
  $xact = T * xant + C$
  $error = norm(xant - xact)$
  $xant = xact;$
  $cont = cont + 1;$
$x = xact$
$iter = cont$
$err = error$

Algorithm 22: Jacobi

input : nxn matrix A, column vector b, initial aproximation X0, weighing factor w, tolerance, maximum interations Nmax

output: solution vector x, final number of iterations iter, error err

$A \leftarrow D - L - U$
$T \leftarrow (D)^{-1} * (L + U)$
$C \leftarrow (D)^{-1} * b$
$xant \leftarrow x0$
$count \leftarrow 0$
$error = 100$
while $error > tolerance$ *and* $counter < nmax$ **do**
  $xact = T * xant + C$
  $error = norm(xant - xact)$
  $xant = xact;$
  $cont = cont + 1;$
$x = xact$
$iter = cont$
$err = error$

```
INCREMENTAL SEARCHES

Results:
There is a root of f in [ -2.5 , -2.0 ]
There is a root of f in [ -1.0 , -0.5 ]
There is a root of f in [ 0.5 , 1.0 ]
There is a root of f in [ 2.0 , 2.5 ]
There is a root of f in [ 4.0 , 4.5 ]
There is a root of f in [ 5.0 , 5.5 ]
There is a root of f in [ 7.0 , 7.5 ]
There is a root of f in [ 8.0 , 8.5 ]
There is a root of f in [ 10.0 , 10.5 ]
There is a root of f in [ 11.5 , 12.0 ]
There is a root of f in [ 13.5 , 14.0 ]
There is a root of f in [ 14.5 , 15.0 ]
There is a root of f in [ 16.5 , 17.0 ]
There is a root of f in [ 17.5 , 18.0 ]
There is a root of f in [ 19.5 , 20.0 ]
There is a root of f in [ 21.0 , 21.5 ]
There is a root of f in [ 22.5 , 23.0 ]
There is a root of f in [ 24.0 , 24.5 ]
There is a root of f in [ 26.0 , 26.5 ]
There is a root of f in [ 27.0 , 27.5 ]
There is a root of f in [ 29.0 , 29.5 ]
There is a root of f in [ 30.0 , 30.5 ]
There is a root of f in [ 32.0 , 32.5 ]
There is a root of f in [ 33.5 , 34.0 ]
There is a root of f in [ 35.0 , 35.5 ]
There is a root of f in [ 36.5 , 37.0 ]
There is a root of f in [ 38.5 , 39.0 ]
There is a root of f in [ 39.5 , 40.0 ]
There is a root of f in [ 41.5 , 42.0 ]
There is a root of f in [ 43.0 , 43.5 ]
There is a root of f in [ 44.5 , 45.0 ]
There is a root of f in [ 46.0 , 46.5 ]
>>>
```

Bisection

BISECTION

Results:

iter	a	xm	b	f(Xm)	E
1.0	0.50000000000	0.75000000000	1.00000000000	-0.11839639385	0.25000000000
2.0	0.75000000000	0.87500000000	1.00000000000	-0.03681769076	0.12500000000
3.0	0.87500000000	0.93750000000	1.00000000000	0.00063391616	0.06250000000
4.0	0.87500000000	0.90625000000	0.93750000000	-0.01777228923	0.03125000000
5.0	0.90625000000	0.92187500000	0.93750000000	-0.00848658221	0.01562500000
6.0	0.92187500000	0.92968750000	0.93750000000	-0.00390535863	0.00781250000
7.0	0.92968750000	0.93359375000	0.93750000000	-0.00163043812	0.00390625000
8.0	0.93359375000	0.93554687500	0.93750000000	-0.00049693532	0.00195312500
9.0	0.93554687500	0.93652343750	0.93750000000	0.00006882244	0.00097656250
10.0	0.93554687500	0.93603515625	0.93652343750	-0.00021397351	0.00048828125
11.0	0.93603515625	0.93627929688	0.93652343750	-0.00007255479	0.00024414062
12.0	0.93627929688	0.93640136719	0.93652343750	-0.00000186098	0.00012207031
13.0	0.93640136719	0.93646240234	0.93652343750	0.00003348203	0.00006103516
14.0	0.93640136719	0.93643188477	0.93646240234	0.00001581085	0.00003051758
15.0	0.93640136719	0.93641662598	0.93643188477	0.00000697501	0.00001525879
16.0	0.93640136719	0.93640899658	0.93641662598	0.00000255703	0.00000762939
17.0	0.93640136719	0.93640518188	0.93640899658	0.00000034803	0.00000381470
18.0	0.93640136719	0.93640327454	0.93640518188	-0.00000075648	0.00000190735
19.0	0.93640327454	0.93640422821	0.93640518188	-0.00000020422	0.00000095367
20.0	0.93640422821	0.93640470505	0.93640518188	0.00000007190	0.00000047684
21.0	0.93640422821	0.93640446663	0.93640470505	-0.00000006616	0.00000023842
22.0	0.93640446663	0.93640458584	0.93640470505	0.00000000287	0.00000011921
23.0	0.93640446663	0.93640452623	0.93640458584	-0.00000003164	0.00000005960

0.9364045262336731 is root with tol: 5.960464477539063e-08

Figure 2: Proof of Bisection

False Rule

FALSE RULE

Results:

iter	a	xm	b	f(Xm)	E
1.0	0.93394038072	0.93650605167	1.00000000000	0.00005875601	0.00256567095
2.0	0.93394038072	0.93640473074	0.93650605167	0.00000008678	0.00010132092
3.0	0.93394038072	0.93640458110	0.93640473074	0.00000000013	0.00000014964
4.0	0.93394038072	0.93640458088	0.93640458110	0.00000000000	0.00000000022

0.936404580879889 is root with tol: 2.2098023411132317e-10
>>>

Figure 3: Proof of False Rule

Multiple roots

```
Raices multiples

 iter|      xi          |      f(xi)       |       E
   0 |    1.000000e+00 |    7.182818e-01 |
   1 |   -2.342106e-01 |    2.540578e-02 |    1.234211e+00
   2 |   -8.458280e-03 |    3.567061e-05 |    2.426689e-01
   3 |   -1.189018e-05 |    7.068790e-11 |    8.470170e-03
   4 |   -4.218591e-11 |    0.000000e+00 |    1.189023e-05
   5 |   -4.218591e-11 |    0.000000e+00 |    8.437181e-11

A root approximation was found at -4.218590698935789e-11
```

Figure 4: Proof of Gaussian Elimination

```
{2.0, -1.0, 0.0, 3.0, 1.0}
{1.0, 0.5, 3.0, 8.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{14.0, 5.0, -2.0, 3.0, 1.0}

Phase 1
{14.0, 5.0, -2.0, 3.0, 1.0}
{1.0, 0.5, 3.0, 8.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{2.0, -1.0, 0.0, 3.0, 1.0}

{14.0, 5.0, -2.0, 3.0, 1.0}
{0.0, 0.1428571428571429, 3.142857142857143, 7.785714285714286, 0.9285714285714286}
{0.0, 13.0, -2.0, 11.0, 1.0}
{0.0, -1.7142857142857142, 0.2857142857142857, 2.5714285714285716, 0.8571428571428572}

Phase 2
{14.0, 5.0, -2.0, 3.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{0.0, 0.1428571428571429, 3.142857142857143, 7.785714285714286, 0.9285714285714286}
{0.0, -1.7142857142857142, 0.2857142857142857, 2.5714285714285716, 0.8571428571428572}

{14.0, 5.0, -2.0, 3.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{0.0, 0.0, 3.1648351648351647, 7.664835164835164, 0.9175824175824177}
{0.0, 2.220446049250313E-16, 0.021978021978021955, 4.021978021978022, 0.989010989010989}

Phase 3
{14.0, 5.0, -2.0, 3.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{0.0, 0.0, 3.1648351648351647, 7.664835164835164, 0.9175824175824177}
{0.0, 2.220446049250313E-16, 0.021978021978021955, 4.021978021978022, 0.989010989010989}

{14.0, 5.0, -2.0, 3.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{0.0, 0.0, 3.1648351648351647, 7.664835164835164, 0.9175824175824177}
{0.0, 2.220446049250313E-16, 0.0, 3.96875, 0.982638888888889}

{14.0, 5.0, -2.0, 3.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{0.0, 0.0, 3.1648351648351647, 7.664835164835164, 0.9175824175824177}
{0.0, 2.220446049250313E-16, 0.0, 3.96875, 0.982638888888889}

Answers{0.03849518810148731, -0.18022747156605426, -0.30971128608923887, 0.24759405074365706}
```

Figure 5: Proof Partial Pivot

```
{2.0, -1.0, 0.0, 3.0, 1.0}
{1.0, 0.5, 3.0, 8.0, 1.0}
{0.0, 13.0, -2.0, 11.0, 1.0}
{14.0, 5.0, -2.0, 3.0, 1.0}

Phase 1
{11.0, 13.0, -2.0, 0.0, 1.0}
{8.0, 0.5, 3.0, 1.0, 1.0}
{3.0, -1.0, 0.0, 2.0, 1.0}
{3.0, 5.0, -2.0, 14.0, 1.0}

{11.0, 13.0, -2.0, 0.0, 1.0}
{0.0, -8.954545454545455, 4.454545454545455, 1.0, 0.2727272727272727}
{0.0, -4.545454545454545, 0.5454545454545454, 2.0, 0.7272727272727273}
{0.0, 1.4545454545454546, -1.4545454545454546, 14.0, 0.7272727272727273}

Phase 2
{11.0, 0.0, -2.0, 13.0, 1.0}
{0.0, 14.0, -1.4545454545454546, 1.454545454545455, 0.7272727272727273}
{0.0, 2.0, 0.5454545454545454, -4.545454545454545, 0.7272727272727273}
{0.0, 1.0, 4.454545454545455, -8.954545454545455, 0.2727272727272727}

{11.0, 0.0, -2.0, 13.0, 1.0}
{0.0, 14.0, -1.4545454545454546, 1.454545454545455, 0.7272727272727273}
{0.0, 0.0, 0.7532467532467532, -4.753246753246753, 0.6233766233766234}
{0.0, 0.0, 4.558441558441559, -9.058441558441558, 0.22077922077922077}

Phase 3
{11.0, 0.0, 13.0, -2.0, 1.0}
{0.0, 14.0, 1.454545454545455, -1.4545454545454546, 0.7272727272727273}
{0.0, 0.0, -9.058441558441558, 4.558441558441559, 0.22077922077922077}
{0.0, 0.0, -4.753246753246753, 0.7532467532467532, 0.6233766233766234}

{11.0, 0.0, 13.0, -2.0, 1.0}
{0.0, 14.0, 1.454545454545455, -1.4545454545454546, 0.7272727272727273}
{0.0, 0.0, -9.058441558441558, 4.558441558441559, 0.22077922077922077}
{0.0, 0.0, 0.0, -1.6387096774193548, 0.5075268817204301}

{11.0, 0.0, 13.0, -2.0, 1.0}
{0.0, 14.0, 1.454545454545455, -1.4545454545454546, 0.7272727272727273}
{0.0, 0.0, -9.058441558441558, 4.558441558441559, 0.22077922077922077}
{0.0, 0.0, 0.0, -1.6387096774193548, 0.5075268817204301}

Answers{0.03849518810148732, -0.18022747156605426, -0.30971128608923887, 0.24759405074365706}
```

Figure 6: Proof Total Pivot

Gaussian Elimination

```
Eliminaci�n Gaussiana Simple
 Etapa 0
[[ 2.  -1.   0.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [14.   5.  -2.   3.   1. ]]

Etapa 1
[[  2.   -1.    0.    3.    1. ]
 [  0.    1.    3.    6.5   0.5]
 [  0.   13.   -2.   11.    1. ]
 [  0.   12.   -2.  -18.   -6. ]]

Etapa 2
[[  2.   -1.    0.    3.    1. ]
 [  0.    1.    3.    6.5   0.5]
 [  0.    0.  -41.  -73.5  -5.5]
 [  0.    0.  -38.  -96.  -12. ]]

Etapa 3
[[  2.         -1.          0.          3.          1.       ]
 [  0.          1.          3.          6.5         0.5      ]
 [  0.          0.        -41.        -73.5        -5.5      ]
 [  0.          0.          0.        -27.878049  -6.902439]]

Despu�s de sustituci�n regresiva:
 x:
[[ 0.03849519]
 [-0.18022747]
 [-0.30971128]
 [ 0.24759405]]
```

Figure 7: Proof of Gaussian Elimination

Muller Method

```
Muller method
 iter|      x0          |         x1        |      x2           |          Root     |          E
   0 |   -1.000000e+00 |    0.000000e+00  |    1.000000e+00  |                   |
   1 |   -1.000000e+00 |    0.000000e+00  |    1.000000e+00  |    -2.220446e-16  |
   2 |    0.000000e+00 |    1.000000e+00  |   -2.220446e-16  |    -2.220446e-16  |    0.000000e+00
A root aproximation was found at -2.220446049250313e-16
```

Figure 8: Proof of Muller Method

Fixed Point

```
iter xi g(xi) f(xi) E
0 -0.5 -0.6842068330717285 -0.2931087267313766 0.10000010000000001
1 -0.6842068330717285 -0.6999584211588118 -0.16388637436010217 0.18420683307172847
2 -0.6999584211588118 -0.6964329939682882 -0.15288781535493412 0.015751588087083324
3 -0.6964329939682882 -0.6972792521639637 -0.15534454046026097 0.0035254271905236223
4 -0.6972792521639637 -0.697079089789521 -0.1547545467399959 0.00084625819567552
5 -0.697079089789521 -0.6971266036840224 -0.15489408050278758 0.00020016237444264728
6 -0.6971266036840224 -0.6971153345307346 -0.1548609575705832 4.751389450130539e-05
7 -0.6971153345307346 -0.6971180078402816 -0.1548688134852106 1.1269153287751799e-05
8 -0.6971180078402816 -0.6971173736982946 -0.15486694987383254 2.6733095469522183e-06
9 -0.6971173736982946 -0.6971175241262941 -0.15486739194527666 6.34141986921577e-07
10 -0.6971175241262941 -0.69711748844261 -0.15486728707928488 1.5042799950126806e-07
-0.69711748844261 is an aproximation with tolerance of 1e-07 and after 11 iterations
```

Figure 9: Proof of Fixed Point

Secant

```
Iter xi f(xi) E
0 0.5 -0.2931087267313766
1 1 0.035366079380240017
2 0.946166222306525 0.005619392737863826 0.05383377769347497
3 0.9359965807911726 -0.00023632217470054284 0.010169641515352379
4 0.9364070023767038 1.4022358909571153e-06 0.00041042158553117325
5 0.9364045814731196 3.4371649970665885e-10 2.420903584265943e-06
6 0.9364045808795615 -4.996003610813204e-16 5.935580915661376e-10
0.9364045808795624 is the root
```

Figure 10: Proof of Secant

Newton

```
Iter xi f(xi) E
0 0.5 -0.2931087267313766 1.0000001
1 0.9283919899125719 -0.004662157097372055 0.4283919899125719
2 0.9363667412673313 -2.1912619882713535e-05 0.007974751354759446
3 0.9364045800189902 -4.98339092214195e-10 3.783875165885853e-05
0.9364045808795621 its an aproximation to a root with a tolerance of 1e-07
```

Figure 11: Proof of Newton

Doolitle

```
24 [0.5251077877274448, 0.25545768216313536, -0.41047969902650905, -0.2816585612350974]
25 [0.5251083414671069, 0.25545801583827094, -0.4104799594870898, -0.281658892623501]
26 [0.5251086734271935, 0.25545821587239254, -0.4104801156299903, -0.2816590912867551]
27 [0.5251088724331645, 0.25545833579037724, -0.41048020923573025, -0.2816592103829217]
28 [0.5251089917347855, 0.25545840767972766, -0.41048026535121496, -0.28165928177960226]
29 [0.5251090632546336, 0.25545845077650514, -0.4104802989917548, -0.28165932458103016]
30 [0.5251091061298989, 0.25545847661249077, -0.41048031915884275, -0.2816593502399575]
```

Figure 12: Proof of Doolitle

Jacobi

```
| 42 |    1.5e-06 | 0.525106  0.255457  -0.410479  -0.281657
| 43 |    1.1e-06 | 0.525107  0.255457  -0.410479  -0.281658
| 44 |    8.7e-07 | 0.525107  0.255457  -0.410479  -0.281658
| 45 |    6.5e-07 | 0.525108  0.255458  -0.410480  -0.281658
| 46 |    4.9e-07 | 0.525108  0.255458  -0.410480  -0.281659
| 47 |    3.7e-07 | 0.525108  0.255458  -0.410480  -0.281659
| 48 |    2.8e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 49 |    2.1e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 50 |    1.6e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 51 |    1.2e-07 | 0.525109  0.255458  -0.410480  -0.281659
| 52 |    9.0e-08 | 0.525109  0.255458  -0.410480  -0.281659
```

Figure 13: Proof of Jacobi

Vandermonde

```
Vandermonde Matrix:
[[-1.  1. -1.  1.]
 [ 0.  0.  0.  1.]
 [27.  9.  3.  1.]
 [64. 16.  4.  1.]]
Polynomial coefficients:  [-1.1417  5.825  -5.5333  3.    ]
Vandermonde polynom:
-1.1416666666666666 *x^ 3 +
5.824999999999999 *x^ 2 +
-5.533333333333332 *x^ 1 +
3.0 *x^ 0 +
```

Figure 14: Proof of Vandermonde

Newton Divided Difference

```
Newton’s polynomial coefficients:
[ 15.5    -12.5      3.5417  -1.1417]
Newton’s Divided Difference Table
[[ 15.5      0.       0.       0.    ]
 [  3.     -12.5      0.       0.    ]
 [  8.       1.6667   3.5417   0.    ]
 [  1.      -7.      -2.1667  -1.1417]]
 Newton’s polynom:
-1.14166666666667*x*(x - 3.0)*(x + 1.0) + 3.54166666666667*x*(x + 1.0) - 12.5*x
+ 3.0
Newton’s simple polynom::
-1.14166666666667*x**3 + 5.825*x**2 - 5.53333333333333*x + 3.0
```

Figure 15: Proof of Newton Divided Difference

Lagrange

```
Lagrange interpolating polynomials:
L0 -0.05*x**3 + 0.35*x**2 - 0.6*x
L1 0.08333333333333333*x**3 - 0.5*x**2 + 0.4166666666666667*x + 1.0
L2 -0.08333333333333333*x**3 + 0.25*x**2 + 0.3333333333333333*x
L3 0.05*x**3 - 0.1*x**2 - 0.15*x
Lagrange polynom
15.5*L0+3.0*L1+8.0*L2+1.0*L3
```

Figure 16: Proof of Lagrange

Spline Linear

```
Lineal tracers coefficients

-12.5 <-> 3.0
1.66667 <-> 3.0
-7.0 <-> 29.0

Lineal tracers

3.0 - 12.5*x
1.66666666666667*x + 3.0
29.0 - 7.0*x
```

Figure 17: Proof of Spline Linear

Spline Square

```
Cuadratic tracers coefficients

0.0 <-> -12.5 <-> 3.0
4.72222 <-> -12.5 <-> 3.0
-22.83333 <-> 152.83333 <-> -245.0

Cuadratic tracers

3.0 - 12.5*x
4.72222222222222*x**2 - 12.5*x + 3.0
-22.8333333333333*x**2 + 152.833333333333*x - 245.0
```

Figure 18: Proof of Spline Square

Spline Cubic

```
Cubic tracers coefficients

2.53333 <-> 7.6 <-> -7.43333 <-> 3.0
-1.52222 <-> 7.6 <-> -7.43333 <-> 3.0
2.03333 <-> -24.4 <-> 88.56667 <-> -93.0

Cubic tracers

2.53333333333333*x**3 + 7.6*x**2 - 7.43333333333333*x + 3.0
-1.52222222222222*x**3 + 7.6*x**2 - 7.43333333333333*x + 3.0
2.03333333333333*x**3 - 24.4*x**2 + 88.5666666666667*x - 93.0
```

Figure 19: Proof of Spline Cubic

Neville

```
[[15.5, None, None, None], [3.0, 3.0 - 12.5*x, None, None], [8.0,
1.66666666666667*x + 3.0, -0.25*(3.0 - 12.5*x)*(x - 3.0) + 0.25*(x +
1.0)*(1.66666666666667*x + 3.0), None], [1.0, 29.0 - 7.0*x, 0.25*x*(29.0 -
7.0*x) - 0.25*(x - 4.0)*(1.66666666666667*x + 3.0), -0.2*(x - 4.0)*(-0.25*(3.0 -
12.5*x)*(x - 3.0) + 0.25*(x + 1.0)*(1.66666666666667*x + 3.0)) + 0.2*(x +
1.0)*(0.25*x*(29.0 - 7.0*x) - 0.25*(x - 4.0)*(1.66666666666667*x + 3.0))]]
-1.14166666666667*x**3 + 5.825*x**2 - 5.53333333333333*x + 3.0
```

Figure 20: Proof of Neville