Final proyect

First delivery

Santiago Alzate Cardona
Alejandro Castaño.
Esteban Sierra Patiño.
Sebatian Urrego.

October 6, 2021

Abstract

We made some of the mathematical methods for the numerical analysis project, we created the pseudocode, also two formal codes in python (main) and MathLab, that represents some of the uses you can take in those methods.

1 Introduction

Due to the fact that in the final project we will have to present many methods and have an application development in this first delivery we will deliver several methods in implemented code and its respective pseudo-code with their tests to show an advance phase and not to be left behind in later deliveries, as we rather show what we are capable of developing from the scratch.

In this document we only show the methods with their code and tests, however you can find all the files in the following Github.

2 Pseudocodes

In this parts we are going to show the pseudocodes for all the methods we are planning on presenting at the end of the project, as is well known, this pseudocodes can be implemented in any language, but for the purpose of this project will only be presented in Python and Matlab

2.1 Incremental Search

```
Input: Function f, range xin, variable delta delta, Total of iteration Nmax
Output: graph of the function with the solution
if delta <= 0 then   # Delta value cannot be negative
    break;
xant <- xin
fant <- f(xant)
xact <- xant + Delta
fact <- f(xact)

for i from 1 to Nmax Do
    if (fant * fact) < 0 then
        break
    else
        xant <- xact
        fant <- fact
        xact <- xant + Delta
        fact <- f(xact)

a = xant
b = xact

```

Figure 1: Incremental search pseudocode.

2.2 Bisection

```
Input: Function f, start left a, ends right b, Total of iteration Nmax, tolerance tol
Output: graph of the function with the solution
if a > b then   # a cannot be greater than a
    break;
fa <- f(a)
fb <- f(b)
pm <- a + (b - a)/2
fpm <- f(pm)
E = 1000    # The error
count = 1
while (count < Nmax) and (E > tol) then
    if (fa * fpm) < 0 then
        b <- pm
        fb <- fpm
    else
        a <- pm
        fa <- fpm
    p0 <- pm
    pm <- (a + b)/2
    fpm <- f(pm)
    E <- absolut_value(p0 - pm)
    count = count + 1
```

Figure 2: Bisection method pseudocode.

2.3 False rule

```
Input: Function f, start left a, ends right b, Total of iteration Nmax, tolerance tol
Output: graph of the function with the solution
if a > b then   # a cannot be greater than a
    break;
fa <- f(a)
fb <- f(b)
pm <- (fb * a - fa * b)/(fb - fa)
fpm <- f(pm)
E = 1000    # The error
count = 1
while (count < Nmax) and (E > tol) then
    if (fa * fpm) < 0 then
        b <- pm
        fb <- fpm
    else
        a <- pm
        fa <- fpm
    p0 <- pm
    pm <- (fb * a - fa * b)/(fb - fa)
    fpm <- f(pm)
    E <- absolut_value(p0 - pm)
    count = count + 1
```

Figure 3: False Rule method pseudocode.

2.4 Fixed point

```
Input: Function g, range x0, Total of iteration Nmax, tolerance tol
Output: graph of the function with the solution
xant <- x0
E <- 1000
count <- 0
while (count < Nmax) and (E > tol) Do
    xact <- g(xant)
    E <- abs(xact - xant)
    xant <- xact
    count <- count + 1
for i from 1 to Nmax Do
    if (fant * fact) < 0 then
        break
    else
        xant <- xact
        fant <- fact
        xact <- xant + Delta
        fact <- f(xact)

a = xant
b = xact
```

Figure 4: Fixed Point method pseudocode.

2.5 Newton

```
Input: Function f, derivate of Function f df,start point x ,
Total of iteration Nmax, tolerance tol
Output: graph of the function with the solution as a point
xant <- x0
fant <- f(xant)
dfant <- df(xant)
E <- 1000
i <- 0
While (i < Nmax) and (E > tol) Do
    xact <- xant - (fant/dfant)
    fact <- f(xact)
    dfact <- df(xact)
    E <- abs(xact - xant)
    xant <- xact
    fant <- fact
    dfant <- dfact
    i = i + 1
    if (i == Nmax) or (E > tol) then # It doesn't have a solution
    else
        # The method converge
return xact
```

Figure 5: Newton method pseudocode.

2.6 Secant

```
Input: Function f, start point x0 ,end point x1 ,
Total of iteration Nmax, tolerance tol
Output: graph of the function with the solution
f0 <- f(x0)
f1 <- f(x1)
E <- 1000
i = 1
while (i < Nmax) and (E > tol) Do
    xact <- x1 -f1((x1 - x0)/(f1 - f0))
    fact <- f(xact)
    E <- absolute_value(xact - x1)
    x0 <- x1
    f0 <- f1
    x1 <- xact
    f1 <- fact
    i = i + 1
    if (i == Nmax) or (E > tol) then
        # It doesn't have a solution
    else
        # The method converge

```

Figure 6: Secant method pseudocode.

2.7 Multiples roots

```
Input: Function f, first derivate df , second derivate df2 , start point x0
Total of iteration Nmax, tolerance tol
Output: graph of the function with the solution
xant <- x0
fant <- f(xant)
dfant <- df(xant)
E = 1000
i = 1
while (i < Nmax) and (E > tol) Do
    xact <- x1 -f1((x1 - x0)/(f1 - f0))
    xact = (xant - fant*df(xant)/((df(xant)**2 - fant*d2f(xant))))
    fact <- f(xact)
    E <- absolute_value(xact - xant)
    xant <- xact
    fant <- fact
    i = i + 1
    if (i == Nmax) or (E > tol) then
        # It doesn't have a solution
    else
        # The method converge

```

Figure 7: IMultiple roots method pseudocode.

2.8 Simple Gaussian Elimination

```
Input: square n x n 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 <- [A b]
for i from 1 to n-1 do
    if A(i,i) = 0 then
        find 1 such that A(l,i) != 0
        switch Ai and Al
    for j from i + 1 to n do
        multiplier Mji <- A(j,i)/A(i,i)
        Aj <- Aj - Mji * Ai
x <- susreg(A)
```

Figure 8: Simple gaussian elimination pseudocode.

2.9 Gaussian elimination with partial pivot

```
Input: n x n Matrix A, n x 1 vector b
Output: regresive substitution answer
(n, m) = len(A)
A = MakeAugmentedMatrix(A, b)
if n == m then
    for k in range 1 to n - 1 Do
        bigger = 0
        row = k
        for p in range k to n Do
            if bigger < |apk| then
                bigger = |apk|
                row = p
        if bigger == 0 then
            break
        else
            if row is not k then
                for j in range l until n + l Do
                    Aux = a(k, j)
                    a(k, j) = a(row, j)
                    a(row, j) = Aux
        for i in range k + l until n Do
```

Figure 9: Pseudocode of the first part of a gaussian elimination with partial pivot.

```
            mik = aik / akk
            for j in range k until n + l Do
                aij = aij - mik * akj
    for i in range n until l Do
        sum = 0
        for p in range i + l until n Do
            sum = sum + aip * xp
        xi = (bi - sum) / au
else
    # The matrix i not cuadratic

print(a)
print(x)
```

Figure 10: Pseudocode of the second part of a gaussian elimination with partial pivot.

2.10 Gaussian elimination with total pivot

```
Input: n x n Matrix A, n x 1 vector b
Output: regresive substitution answer
(n, m) = len(A)
A = MakeAugmentedMatrix(A, b)
if n == m then
    for i in range 1 until n Do
        mark(i) = i
    for k in range 1 until n - 1 Do
        bigger = 0
        rowm = k
        colm = k
        for p in range k until n Do
            for r in range k until n Do
                if bigger < apr then
                    bigger = apr
                    rowm = p
                    colm = r

        if bigger == 0 then
            # Suspended
        else
```

Figure 11: Pseudocode of the first part of a gaussian elimination with total pivot.

```
            if rowm is not k then
                for j un range 1 until  n + 1 Do
                    Aux = a(k, j)
                    a(k, j) = a(i, colmnam)
                    a(i, columnam) = Aux
                Aux = mark(k)
                mark(k) = mark(columnam)
                mark(columnam) = Aux
        for i range k + 1 until n Do
            ma = aik / akk
            for j in range k until n + 1 Do
                aij = aij - mik * akj
        for i in range n until 1 Do
            sum = 0
            for p in range i + 1 until n Do
                sum = sum + aip * xp
            xi = (bi - sum) / aii
        for i in range n until 1 Do
            for j in range n until n Do
                if mark(j) == i then
                    k = j
```

Figure 12: Psudocode of the second part of a gaussian elimination with total pivot.

```
                Aux = x(k)
                x(k) = x(i)
                x(i) = Aux
                Aux = mark(k)
                mark(k) = mark(i)
                mark(i) = Aux
else
    # The matrix is not cuadratic
print(a)
print(x)
```

Figure 13: Pseudocode of the third part of a gaussian elimination with total pivot.

3 Tests

3.1 Python

3.1.1 Incremental Search

```python
def busqueda(func, xin, delta, Nmax):
    if delta<=0:
        print('Delta value can not be negative')
        return

    xant = xin
    fant = func(xant)
    xact = xant + delta
    fact = func(xact)

    i = 1
    for i in range(1, Nmax):
        print(i, "- (a =",xant,"b =",xact,")")
        if (fant * fact) < 0:
            break
        else:
            xant = xact
            fant = fact
            xact = xant + delta
            fact = func(xact)

    a = xant
    b = xact
    count = i
```

Figure 14: Code of incremental search method in python.

```
1 - (a = -3 b = -2.5 )
2 - (a = -2.5 b = -2.0 )
```

Figure 15: Incremental search test.

3.1.2 Bisection

```python
def biseccion(func, a, b, Nmax, tol):
    if a > b:
        print("'A' cannot be greather than 'B'")
        return

    fa = func(a)
    fb = func(b)
    pm = a + (b-a)/2
    fpm = func(pm)

    E = 1000
    count = 1
    while (count < Nmax) and (E > tol):
        print(count, " - (a =",a,"b =",b,")")
        if (fa * fpm) < 0:
            b = pm
            fb = fpm
        else:
            a = pm
            fa = fpm
        p0 = pm
        pm = (a+b)/2
        fpm = func(pm)
        E = abs(p0-pm)

        count += 1
```

Figure 16: Code of bisection method in python.

```
1  - (a = 0 , b = 1 )
2  - (a = 0.5 , b = 1 )
3  - (a = 0.75 , b = 1 )
4  - (a = 0.875 , b = 1 )
5  - (a = 0.875 , b = 0.9375 )
6  - (a = 0.90625 , b = 0.9375 )
7  - (a = 0.921875 , b = 0.9375 )
8  - (a = 0.9296875 , b = 0.9375 )
9  - (a = 0.93359375 , b = 0.9375 )
10  - (a = 0.935546875 , b = 0.9375 )
11  - (a = 0.935546875 , b = 0.9365234375 )
12  - (a = 0.93603515625 , b = 0.9365234375 )
13  - (a = 0.936279296875 , b = 0.9365234375 )
14  - (a = 0.9364013671875 , b = 0.9365234375 )
15  - (a = 0.9364013671875 , b = 0.93646240234375 )
16  - (a = 0.9364013671875 , b = 0.936431884765625 )
17  - (a = 0.9364013671875 , b = 0.9364166259765625 )
18  - (a = 0.9364013671875 , b = 0.9364089965820312 )
19  - (a = 0.9364013671875 , b = 0.9364051818847656 )
20  - (a = 0.9364032745361328 , b = 0.9364051818847656 )
21  - (a = 0.9364042282104492 , b = 0.9364051818847656 )
22  - (a = 0.9364042282104492 , b = 0.9364047050476074 )
23  - (a = 0.9364044666290283 , b = 0.9364047050476074 )
```

Figure 17: Bisection test.

3.1.3 False rule

```python
def reglafalsa(func, a, b, Nmax, tol):
    if a > b:
        print("'A' cannot be greather than 'B'")
        return

    fa = func(a)
    fb = func(b)
    pm = (fb*a-fa*b)/(fb-fa)
    fpm = func(pm)

    E = 1000
    count = 1
    while (count < Nmax) and (E > tol):
        print(count, " - (a =",a,"b =",b,")")
        if (fa * fpm) < 0:
            b = pm
            fb = fpm
        else:
            a = pm
            fa = fpm

        p0 = pm
        pm = (fb*a - fa*b)/(fb-fa)
        fpm = func(pm)
        E = abs(pm-p0)
        count += 1
```

Figure 18: Code of false rule in python.

```
1  - (a = 0 b = 1 )
2  - (a = 0.9339403807182157 b = 1 )
3  - (a = 0.9339403807182157 b = 0.9365060516656253 )
4  - (a = 0.9339403807182157 b = 0.9364047307426411 )
```

Figure 19: False rule test.

3.1.4 Fixed point

```python
def puntofijo(g, x0, Nmax, tol):
    xant = x0
    E = 1000
    count = 0
    while (count < Nmax) and (E > tol):
        print(count, "-", xant)
        xact = g(xant)
        E = abs(xact-xant)
        xant = xact
        count += 1

    x = xact
    err = E
```

Figure 20: Code of the fixed point method in python.

```
 i |  X
----------------
00 | -0.5
01 | -0.2931087267313766
02 | -0.41982154360625734
03 | -0.3463045191776651
04 | -0.3909584565423095
05 | -0.3644050348941392
06 | -0.3804263031679563
07 | -0.37083679528020885
08 | -0.3766056453635812
09 | -0.373145417607189
10 | -0.3752246411870562
11 | -0.37397658604830963
12 | -0.3747262157084321
13 | -0.37427613331045395
14 | -0.3745464284580923
15 | -0.3743841264348447
16 | -0.3744815908319551
17 | -0.37442306518389706
18 | -0.37445820986270584
19 | -0.3744371058494556
20 | -0.37444977872741303
21 | -0.37444216876320036
22 | -0.3744467385052047
23 | -0.37444399440652526
24 | -0.37444564222126353
25 | -0.37444465271927385
26 | -0.3744452469090602
27 | -0.37444489010190096
28 | -0.37444510436235334
29 | -0.3744449757003151
```

Figure 21: Fixed point test.

3.1.5 Newton

```python
def newton(f, df, x0, Nmax, tol):
    xant = x0;
    fant = f(xant)
    dfant = df(xant)
    E = 1000
    i = 0

    while i < Nmax and E > tol:
        print("X{i} =".format(i=i), xant,"|","f(x{i}) =".format(i=i), fant,"|","Error =", E)
        xact = xant - (fant/dfant)
        fact = f(xact)
        dfact = df(xact)
        E = abs(xact-xant)
        xant = xact
        fant = fact
        dfant = dfact
        i += 1

    if i == Nmax or E > tol:
        print("El metodo no converge con los datos dados")
    else:
        print("X{i} =".format(i=i), xact,"|","f(x{i}) =".format(i=i), fact,"|","Error =", E)
        print("El metodo converge a x:", xact, "en la iteracion:", i)

    return xact
```

Figure 22: Code of the newton method in python.

```
X0 = 0.5 | f(x0) = -0.2931087267313766 | Error = 1000
X1 = 0.9283919899125719 | f(x1) = -0.004662157097372055 | Error = 0.4283919899125719
X2 = 0.9363667412673313 | f(x2) = -2.1912619882713535e-05 | Error = 0.007974751354759446
X3 = 0.9364045800189902 | f(x3) = -4.98339092214195e-10 | Error = 3.783875165885853e-05
X4 = 0.9364045808795621 | f(x4) = -1.1102230246251565e-16 | Error = 8.605719470367035e-10
El metodo converge a x: 0.9364045808795621 en la iteracion: 4
```

Figure 23: Newton method test.

3.1.6 Secant

```python
def secante(func, x0, x1, Nmax, tol):
    x = Symbol('x')
    f = lambdify(x, func)
    fDer1 = lambdify(x, func.diff(x))

    f0 = f(x0)
    f1 = f(x1)
    E = 1000
    i = 1

    while i < Nmax and E > tol:
        xact = x1 -f1*((x1-x0)/(f1-f0))
        fact = f(xact)
        print("X{i} =".format(i=i), xact)
        print("f(x{i}) =".format(i=i), fact)
        print("Error =", E)
        print()
        E = abs(xact-x1)
        x0 = x1
        f0 = f1
        x1 = xact
        f1 = fact
        i += 1

    if i == Nmax or E > tol:
        print("El metodo no converge con los datos dados")
    else:
        print("X{i} =".format(i=i), xact)
        print("f(x{i}) =".format(i=i), fact)
        print("Error =", E)
        print()

        print("El metodo converge a x:", xact)
        print("En la iteracion:", i)
```

Figure 24: Code of the secant method in python.

```
X1 = 0.946166222306525 | f(x1) = 0.005619392737863826 | Error = 1000
X2 = 0.9359965807911726 | f(x2) = -0.00023632217470054284 | Error = 0.05383377769347497
X3 = 0.9364070023767038 | f(x3) = 1.4022358909571153e-06 | Error = 0.010169641515352379
X4 = 0.9364045814731196 | f(x4) = 3.4371649970665885e-10 | Error = 0.00041042158553117325
X5 = 0.9364045808795615 | f(x5) = -4.996003610813204e-16 | Error = 2.420903584265943e-06
X6 = 0.9364045808795615 | f(x6) = -4.996003610813204e-16 | Error = 5.935580915661376e-10
El metodo converge a x: 0.9364045808795615 en la iteracion 6
```

Figure 25: Code of a simple gaussian elimination method in matlab.

3.1.7 Multiple roots

```python
def raicesmlt(f, df, d2f, x0, Nmax, tol):
    xant = x0;
    fant = f(xant)
    dfant = df(xant)
    E = 1000
    i = 0

    while i < Nmax and E > tol:
        print("X{i} =".format(i=i), xant,"|","f(x{i}) =".format(i=i), fant,"|","Error =", E)
        xact = xant - fant*df(xant)/((df(xant)**2 - fant*d2f(xant)))
        fact = f(xact)
        E = abs(xact-xant)
        xant = xact
        fant = fact
        i += 1

    if i == Nmax or E > tol:
        print("El metodo no converge con los datos dados")
    else:
        print("El metodo converge a x:", xact, "en la iteracion", i)
```

Figure 26: Code of the multiple roots method in python.

```
X0 = 1 | f(x0) = 0.7182818284590451 | Error = 1000
X1 = -0.23421061355351425 | f(x1) = 0.025405775475345838 | Error = 1.2342106135535142
X2 = -0.00845827991076109 | f(x2) = 3.567060801401567e-05 | Error = 0.22575233364275316
X3 = -1.1890183808588653e-05 | f(x3) = 7.068789997788372e-11 | Error = 0.008446389726952502
X4 = -4.218590698935789e-11 | f(x4) = 0.0 | Error = 1.1890141622681664e-05
El metodo converge a x: -4.218590698935789e-11 en la iteracion 5
```

Figure 27: Multiple roots method test.

3.1.8 Simple Gaussian elimination

```python
def gausspl(A, b):
    A
    M = np.column_stack((A, b))
    n = A.shape[0]
    #print("Rows: ", M.shape[0], ",", "Columns:", M.shape[1])
    M = M.astype('float')
    #print(M.dtype)

    etapa = 1
    # Todas las columnas de A - 1
    for i in range(0, n - 1):
        print("Etapa", etapa)
        print(M)
        etapa += 1
        if M[i][i] == 0:
            change = False
            for c in range(i+1, n):
                if M[c][i] != 0:
                    M[[c, i],:] = M[[i,c],:]
                    change = True
                    break
            if change:
                print("Etapa", etapa)
                print(M)
                etapa += 1
            else:
                print('This equation system can not be resolved by this method')
                return


        # Por cada columna iterar por cada fila debajo de la diagonal
        for j in range(i+1, n):
            if M[j][i] != 0:
                # Operacion de subfila
                M[j][i:n+1] = M[j][i:] - (M[j][i]/M[i][i])*M[i][i:]

    print("Etapa", etapa)
    print(M)
    # Sustitucion regresiva
    x = sustreg(M)
    print("X:")
    print(x)
    return x
```

Figure 28: Code of a simple gaussian elimination method in python.

```
Etapa 1
[[ 2.  -1.   0.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [14.   5.  -2.   3.   1. ]]
Etapa 2
[[  2.   -1.    0.    3.    1. ]
 [  0.    1.    3.    6.5   0.5]
 [  0.   13.   -2.   11.    1. ]
 [  0.   12.   -2.  -18.   -6. ]]
Etapa 3
[[  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 4
[[  2.          -1.           0.           3.           1.        ]
 [  0.           1.           3.           6.5          0.5       ]
 [  0.           0.         -41.         -73.5         -5.5       ]
 [  0.           0.           0.         -27.87804878  -6.90243902]]
X:
[ 0.03849519 -0.18022747 -0.30971129  0.24759405]
```

Figure 29: Simple gaussian elimination test

3.1.9 Gaussian elimination with partial pivot

```python
def gausspar(A, b):
    A
    M = np.column_stack((A, b))
    n = A.shape[0]
    #print("Rows: ", M.shape[0], ",", "Columns:", M.shape[1])
    M = M.astype('float')
    #print(M.dtype)

    etapa = 1
    # Todas las columnas de A - 1
    for i in range(0, n - 1):
        print("Etapa", etapa)
        print(M)
        etapa += 1
        col = np.abs(M[i+1:, i])
        aux0 = np.max(col) # Maximo en columna
        aux1 = np.argmax(col) # Indice de subcolumna
        if aux0 > abs(M[i][i]):
            #aux2=M[i+aux1+1][i:] # Temporal de fila con valor mayor
            # aux2 cambiaba al cambiar la matrix (Funciona como una variable referencia)
            M[[i, i+aux1+1],i:] = M[[i+aux1+1, i],i:] # Asi se intercambian filas en python
            #M[i+aux1+1][i:] = M[i][i:]
            #M[i][i:]=aux2
            print("Etapa", etapa)
            print(M)
            etapa += 1
        # Por cada columna iterar por cada fila debajo de la diagonal
        for j in range(i+1, n):
            if M[j][i] != 0:
                # Operacion de subfila
                M[j][i:n+1] = M[j][i:] - (M[j][i]/M[i][i])*M[i][i:]

    print("Etapa", etapa)
    print(M)
    # Sustitucion regresiva
    x = sustreg(M)
    print("X:")
    print(x)
    return x
```

Figure 30: Code of gaussian elimination with partial pivot method in python.

```
Etapa 1
[[ 2.  -1.   0.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [14.   5.  -2.   3.   1. ]]
Etapa 2
[[14.   5.  -2.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [ 2.  -1.   0.   3.   1. ]]
Etapa 3
[[14.          5.         -2.          3.          1.        ]
 [ 0.          0.14285714  3.14285714  7.78571429  0.92857143]
 [ 0.         13.         -2.         11.          1.        ]
 [ 0.         -1.71428571  0.28571429  2.57142857  0.85714286]]
Etapa 4
[[14.          5.         -2.          3.          1.        ]
 [ 0.         13.         -2.         11.          1.        ]
 [ 0.          0.14285714  3.14285714  7.78571429  0.92857143]
 [ 0.         -1.71428571  0.28571429  2.57142857  0.85714286]]
Etapa 5
[[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  2.19780220e-02  4.02197802e+00
   9.89010989e-01]]
Etapa 6
[[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  0.00000000e+00  3.96875000e+00
   9.82638889e-01]]
X:
[ 0.03849519 -0.18022747 -0.30971129  0.24759405]
```

Figure 31: Gaussian elimination with partial pivot test.

3.1.10 Gaussian elimination with total pivot

```python
def gausstot(A, b):
    M = np.column_stack((A, b))
    n = A.shape[0]
    M = M.astype('float')
    stack = []
    etapa = 1
    # Todas las columnas de A - 1
    for i in range(0, n - 1):
        print("Etapa", etapa)
        print(M)
        etapa += 1
        submatrix = abs(M[i:,i:n])
        #print(submatrix)
        a = np.amax(submatrix)
        row, col = np.where(submatrix == a)
        row = row[0]
        col = col[0]
        #print("Row:",row,"Col:",col)
        change = False
        # Cambio de fila
        if row+i != i:
            M[[row+i, i],i:] = M[[i, row+i],i:]
            change = True
        # Cambio de columna
        if col+i != i:
            M[:,[i, col+i]] = M[:,[col+i, i]]
            stack = np.append(stack, [i, col+i])  # Agregar cambio de columna al stack
            change = True

        if change:
            print("Etapa", etapa)
            print(M)
            etapa += 1

        # Por cada columna iterar por cada fila debajo de la diagonal
        for j in range(i+1, n):
            if M[j][i] != 0:
                # Operacion de subfila
                M[j][i:n+1] = M[j][i:] - (M[j][i]/M[i][i])*M[i][i:]

    print("Etapa", etapa)
    print(M)
    stack = np.array(stack, dtype=int).reshape(-1,2)
    # Sustitucion regresiva
    x = sustreg(M)
    print("X antes del cambio de columnas:")
    print(x)
    x = cambioCols(x, stack)
    print("X despues del cambio de columnas:")
    print(x)
```

Figure 32: Code of gaussian elimination with total pivot method in python.

```
Etapa 1
[[ 2.  -1.   0.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [14.   5.  -2.   3.   1. ]]
Etapa 2
[[14.   5.  -2.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [ 2.  -1.   0.   3.   1. ]]
Etapa 3
[[14.          5.         -2.          3.          1.        ]
 [ 0.          0.14285714  3.14285714  7.78571429  0.92857143]
 [ 0.         13.         -2.         11.          1.        ]
 [ 0.         -1.71428571  0.28571429  2.57142857  0.85714286]]
Etapa 4
[[14.          5.         -2.          3.          1.        ]
 [ 0.         13.         -2.         11.          1.        ]
 [ 0.          0.14285714  3.14285714  7.78571429  0.92857143]
 [ 0.         -1.71428571  0.28571429  2.57142857  0.85714286]]
Etapa 5
[[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  2.19780220e-02  4.02197802e+00
   9.89010989e-01]]
Etapa 6
[[ 1.40000000e+01  5.00000000e+00  3.00000000e+00 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01  1.10000000e+01 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  4.02197802e+00  2.19780220e-02
   9.89010989e-01]]
Etapa 7
[[ 1.40000000e+01  5.00000000e+00  3.00000000e+00 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01  1.10000000e+01 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  0.00000000e+00 -1.63870968e+00
   5.07526882e-01]]
X antes del cambio de columnas:
[ 0.03849519 -0.18022747  0.24759405 -0.30971129]
X despues del cambio de columnas:
[ 0.03849519 -0.18022747 -0.30971129  0.24759405]
```

Figure 33: Gaussian elimination with total test.

3.2 Matlab

3.2.1 Incremental Search

```
function xv = incSearch(func, xmin,xmax, sc)
if nargin < 3, error('At least 3 arguments are required for this function'), end
if nargin <4, sc = 50; end
x = linspace(xmin, xmac, sc);
f = func(x);
nv = 0  %Number of valuable brackets (where  there is a sign change)
xv = [] %Base of the return value, no sign changes = empty vector return.

for k = 1:length(x)-1
    if sign(f(k)) ~= sign(f(k+1))
        nv = nv + 1;
        xv(nv,1) = x(k);
        xv(nv,2) = x(k+1);
    end
end
if isempty(xv)
    disp('No brackets with sign change found')
else
    disp('Number of brackets: ')
    disp(nv)
end
```

Figure 34: Code of incremental search method in matlab.

3.2.2 Bisection

```
function [x, icount, err] = bisection(func,xmin, xmax, tol, sc)
if func(xmin) == 0
    disp('Lower endpoint of the function is one of the roots')
    return
elseif func(xmax) == 0
    disp('Upper endpoint of the function is one of the roots')
    return
end

icount = 1 ;
err = 1000;
fmin = func(xmin);
th = (xmin+xmax)/2
fth = func(th);

while fth ~= 0 && err>tol && icount<sc
    if sign(fmin) ~= sign(fth)
        tmax = th;
    else
        tmin = th;
    end
    temp = th;
    th = (tmax+tmin)/2;
    fth = func(th);
    E = abs(th-temp);
    icount = icount+1
end

if th == 0
fprintf('Solution: %t is root', th)
else
    if E<tol
        fprint('%t is an approximation of a root with a tolerance of %v',th,tol)
    else
        fprintf('Failure after %g iterations', icount)
    end
end
```

Figure 35: Code of bisection method in matlab.

3.2.3 False rule

```
function fp = falsePosition(func, xmin, xmax, tol, sc)
fmax = func(xmax);
fmin = func(xmin);
icount = 1;
error = 1000;
th = (xmin)-((func(xmin)*(xmin-xmax))/(func(xmax)-f(xmin)));
fth = func(th);
Z = [icount, xmin, xmax,th,fth,error];

while error > tol
    if sign(fth) == sign(fmax)
        xmax = th;
        fmax = fth;
    else
        xmin = th;
        fmin = fth;
    end
    temp = th;
    th = (xmin)-((func(xmin)*(xmin-xmax))/(func(xmax)-f(xmin)));
    fth = func(th);
    error = abs(th-temp)/th;
    icount = icount+1;
    Z(icount,1)= icount;
    Z(icount,2)=xmin;
    Z(icount,3)=xmax;
    Z(icount,4)=th;
    Z(icount,5)=fth;
    Z(icount,6)=error;
end
if fth==0
disp('SOLUTION:%g is root',th);
else
if Error<Tol
disp('SOLUTION:%g is an approximation to a root with a tolerance %f',th,tol);
else
disp('SOLUTION: Failure after %g iterations',icount);
end
end
end
```

Figure 36: Code of the false rule method in matlab.

3.2.4 Fixed point

```
function [x, icount, err] = fixedPoint(gunc, startp, tol)
x1=startp;
x2=gunc(x1);
icount = 1;

while abs(x2-x1) > tol
    x1 = x2; ;
    x2=gunc(x1);
    err = abs((x1-startp)/x1);
    icount = icount + 1;
end

if x2==0
disp('SOLUTION:%g is root',startp);
else
if Error<Tol
disp('SOLUTION:%g is an approximation with a tolerance of %f',startp,tol);
else
disp('SOLUTION: Failure in %g iterations',icount);
end
end
```

Figure 37: Code of fixed point method in matlab.

3.2.5 Newton

```
function [x, icount, err] = newton(func, df, startp, tol, sc)
x0 = startp;
err=1000;
cont=0;

while E>tol && cont < sc
    x1 = x0 - func(x0)/df(x0);
    err = abs(x1-x0);
    icount = icount+1;
    %Falta retornar valores de la iteracion fact ^ fant.
    x0 = x1;
end
disp(x0, err, icount)
```

Figure 38: Code of the newton method in matlab.

3.2.6 Secant

```
function [ret] = secantMethod(func,x0,x1,tol,sc)
fx0 = func(x0);
if fx0 == 0
    disp('Solution: %t',x0)
else
    icount = 1;
    err = 1000;
    fx1 = func(x1);
    delta = (fx1-fx0);
    err = Tol+1;
    ret = [icount,x1,fx1,err]
    while err > tol && icount <sc && delta ~= 0
        icount = icount + 1;
        temp = x1-f1*(x1-x0)/(fx1-fx0);
        ftemp = func(temp);
        err = abs(temp-x1);
        x0=x1;
        fx0=fx1;
        x1=temp;
        fx1 = ftemp;
        delta = (fx1-fx0)
        ret(cont,1)=icount;
        ret(cont,2)=x1;
        ret(cont,3)=fx1;
        ret(cont,4)=err;
    end
end
disp(ret)
```

Figure 39: Code of the secant method in matlab.

3.2.7 Multiple roots

```
function ret = multiroot(func, fprime, f2prime, x0, tol)
err = 10;
icount = 1
while abs(err) >tol
    temp = x0-func(x0)*fprime(x0)/((fprime(x0))^2-func(x0)*f2prime(x0));
    err = ((temp-x0)/temp*100);
    x0=temp;
    icount = icount +1;
end
disp('Root: %f in %i iterations',x0,icount);
```

Figure 40: Code of the multiple roots method in matlab.

3.2.8 Simple Gaussian elimination

```
function ret = gaussElimination(A,B)

for col = 1:n - 1
    for row = col+1:nargin
        factor = A(row,col)/A(col,col);
        A(row,:) = A(row,:) - faactor*A(col,:);
        B(row) = B(row) - factor*B(col);
        c = [num2str(A), T, num2str(B)];
        disp(c);
        disp(newline);
    end
end
```

Figure 41: Code of a simple gaussian elimination method in matlab.

3.2.9 Gaussian elimination with partial pivot

```
function x = gaussPartialPivot(A,b)
[m, n] = size(A);
x = zeros(m,1);
l = zeros(m:m-1);


for col = 1:m-1

    for p = col+1:m
        if(abs(A(col,col)) < abs(A(p,col)))
            %Partial pivoting, switching rows.
            A([col p],:) = A([p col], :);
            b([k p]) = b([p k]);
        end
    end

    for i = col+1:m
        l(i,col) = A(i,col)/A(col,col);
        for j = col+1:n
            A(i,j) = A(i,j)-l(i,col)*A(col,j);
        end
        b(i) = b(i)-l(i,col)*b(col);
    end

    disp([A B'])
end
for col = 1:m-1
    for i = col+1:m
        A(i,col) = 0;
    end
end

x(m) = b(m)/A(m,m);

for i = m-1:-1:1
    sum = 0;
    for j = i+1:m
        sum = sum + A(i,j)*x(j);
    end
    x(i) = (b(i) - sum)/A(i,i);
end
```

Figure 42: Code of gaussian elimination with partial pivot method in matlab.

3.2.10 Gaussian elimination with total pivot

```
function x = gaussCompPivot(A,b)
C =[A,b'];

if n==m
for i=1:n
    trace(i)=i;
end
for k=1:n-1
    max=0;
    RowM=k;
    ColM=k;
    for p=k:n
        for r=k:n
            if max<abs(C(p,r))
                max=abs(C(p,r));
                RowM=p;
                ColM=r;
            end
        end
    end
    if max == 0
        fprintf('This system has infinte solutions!')
        break
    else
        if RowM ~= k
            for j=1:(n+1)
                aux=C(k,j);
                C(k,j)=C(RowM,j);
                C(RowM,j)=aux;
            end

        end
    if ColM ~= k
        for i=1:n
            aux=C(i,k);
            C(i,k)=C(i,ColM);
            C(i,ColM)=aux;
        end
        aux = trace(k);
        trace(k)= trace(ColM);
        trace(ColM)=aux;
    end
    end
```

Figure 43: First part of the code of gaussian elimination with total pivot method in matlab.

```
for i=(k+1):n %scalar to set up the subtraction.
    m(i,k)=C(i,k)/C(k,k);
    for j=k:(n+1)
        C(i,j)= C(i,j) - m(i,k)*C(k,j); %Resulting Row
    end
end
disp(C)
end
trace(ColM) = aux;
for i=n:-1:1
    suma=0;
    for p=(i+1):n
        suma = suma + C(i,p)*X(p);
    end
    X(i)=(C(i,n+1)-suma)/C(i,i);
end

%Reorganization of the matrix through the trace vector created.
for i=1:n
    for j=1:n
        if trace(j)==i
            k=j;
        end
    end
    aux=X(k);
    X(k)=X(i);
    X(i)=aux;
    aux=trace(k);
    trace(k)=trace(i);
    trace(i)=aux;
end
else
    disp('Matrix is not squared: nxn');
end
```

Figure 44: Second part of the code of gaussian elimination with total pivot method in matlab.