Análisis Numérico Entrega Final

Jhon Jairo Chavarría Gaviria

Samuel Cadavid Pérez

José Alejandro Díaz Urrego

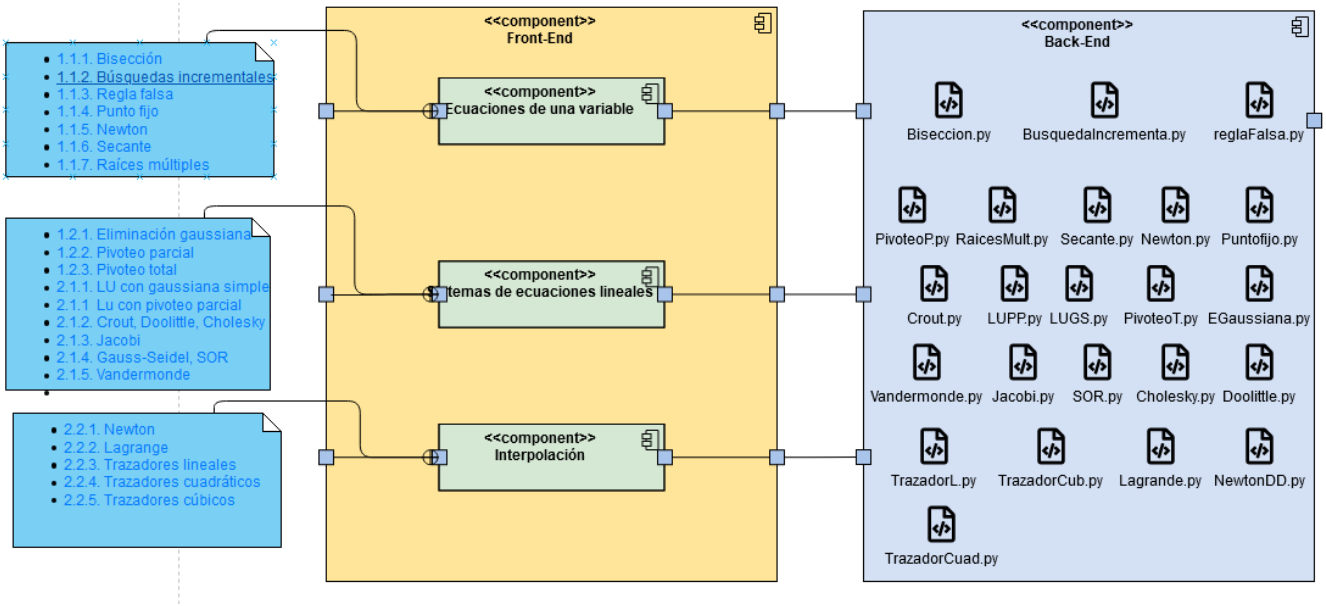
Noviembre 25, 2020

Contenidos

Acerca de

Esta aplicación se usa para solucionar problemas de manera numérica, está desarrollada en Python con el Flask, no tiene ejecutable, se debe ejecutar por consola, pero cuenta con una interfaz grafica bastante intuitiva para los usuarios.

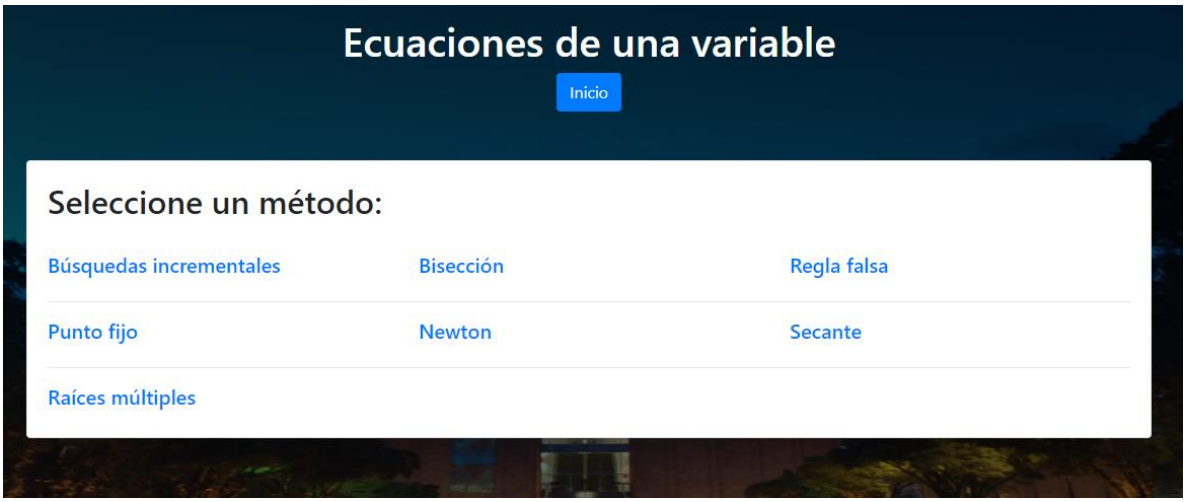
Diseño de módulos



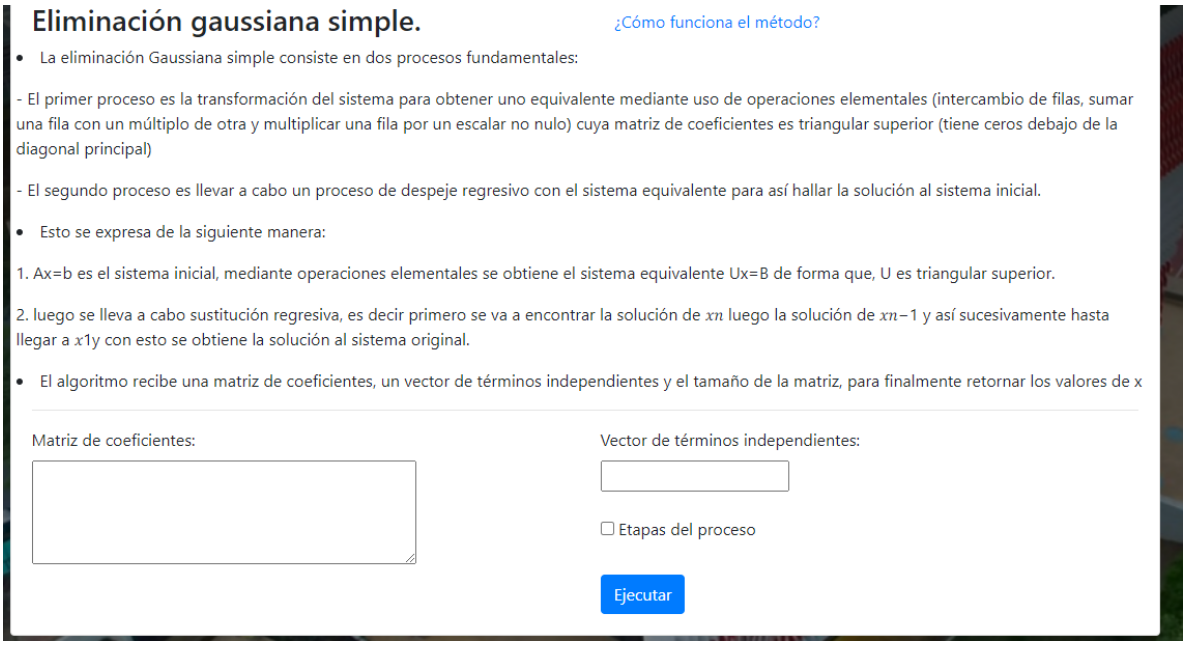
Diseño de interfaces

Para el diseño de interfaces se trato de mantener una interfaz limpia y fácil de utilizar para los usuarios, para esto se separaron los diferentes métodos por la utilidad que tenían, básicamente en los tres módulos del curso, solución de ecuaciones de una sola variable, solución de sistemas de ecuaciones e interpolación.

Dentro de cada módulo están los métodos correspondientes



Y dentro de cada método encontramos una breve descripción y los campos requeridos para poder ejecutar el método


Eliminación gaussiana simple.
¿Cómo funciona el método?
La eliminación Gaussiana simple consiste en dos procesos fundamentales:
- El primer proceso es la transformación del sistema para obtener uno equivalente mediante uso de operaciones elementales (intercambio de filas, sumar una fila con un múltiplo de otra y multiplicar una fila por un escalar no nulo) cuya matriz de coeficientes es triangular superior (tiene ceros debajo de la diagonal principal)
- El segundo proceso es llevar a cabo un proceso de despeje regresivo con el sistema equivalente para así hallar la solución al sistema inicial.
Esto se expresa de la siguiente manera:
1. Ax=b es el sistema inicial, mediante operaciones elementales se obtiene el sistema equivalente Ux=B de forma que, U es triangular superior.
2. luego se lleva a cabo sustitución regresiva, es decir primero se va a encontrar la solución de xn luego la solución de $xn-1$ y así sucesivamente hasta llegar a $x1$y con esto se obtiene la solución al sistema original.
El algoritmo recibe una matriz de coeficientes, un vector de términos independientes y el tamaño de la matriz, para finalmente retornar los valores de x
Matriz de coeficientes:
Vector de términos independientes:
Etapas del proceso
Ejecutar

Documentación del sistema

Diagrama de actividades del sistema, indica el flujo normal de la aplicación:

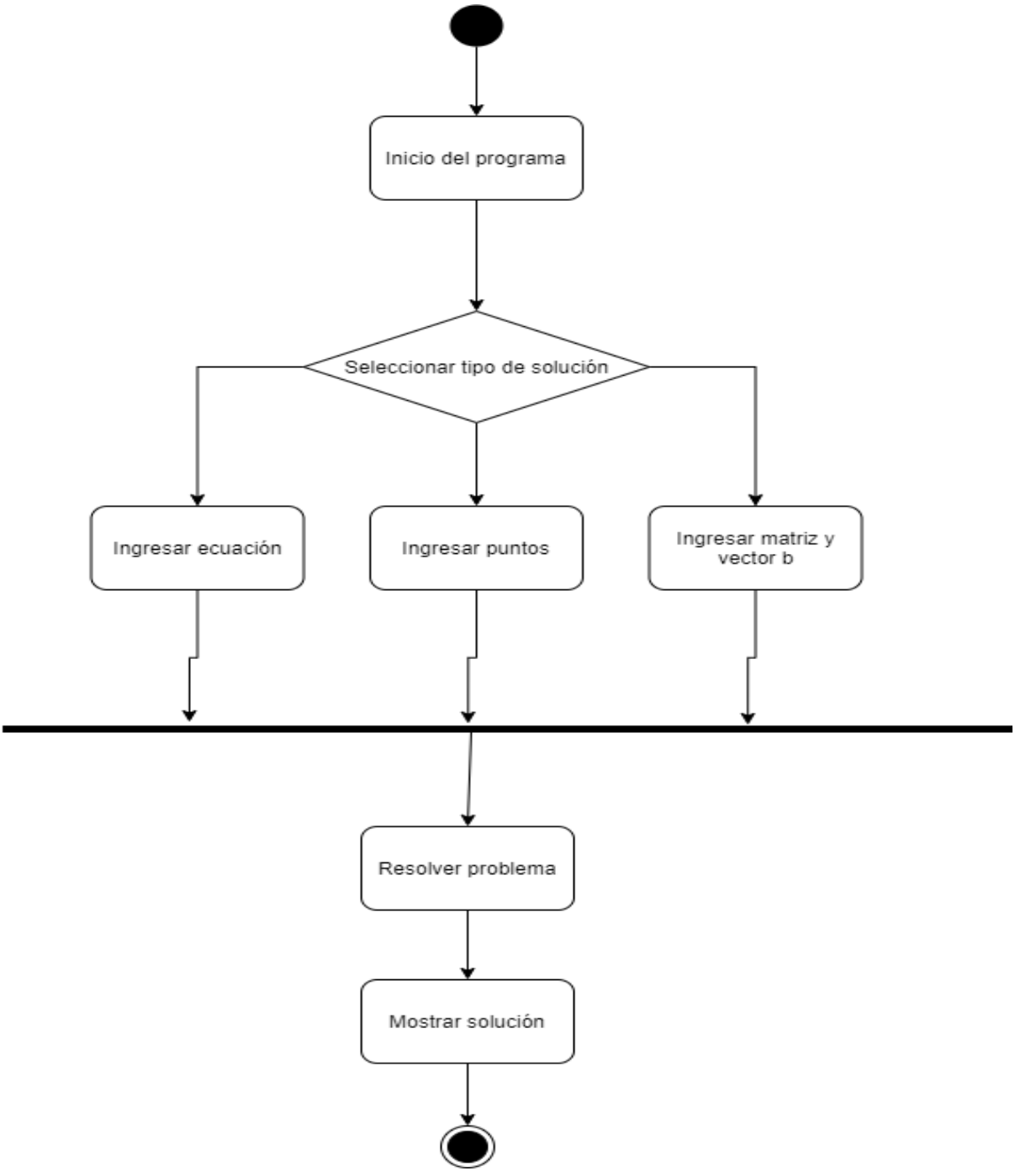
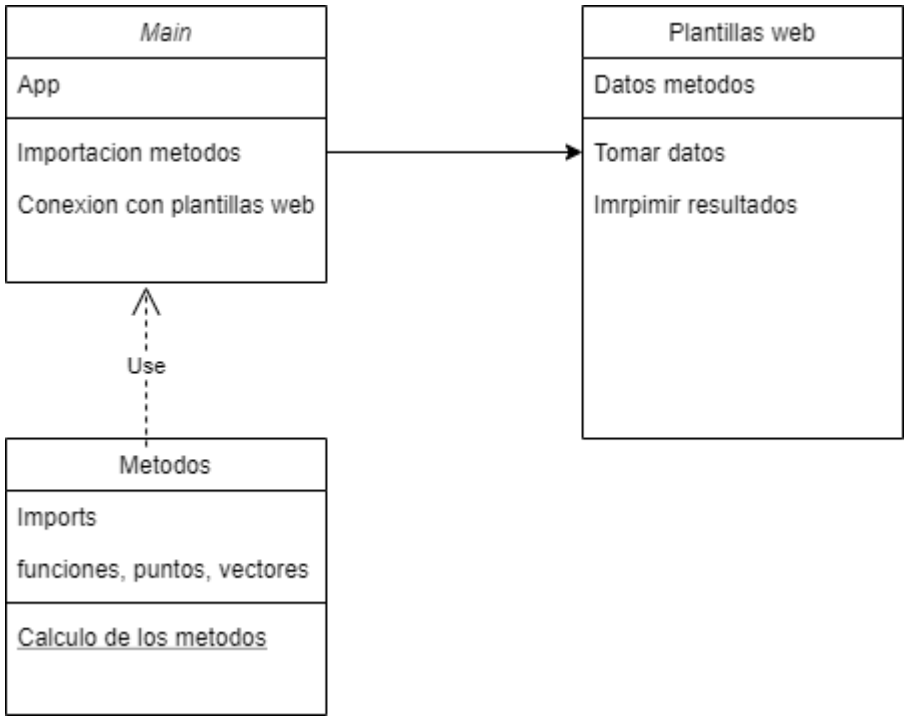


Diagrama de clases del sistema



Conclusiones

El uso de los métodos numéricos para la solución de problemas de análisis matemático nos da a grandes rasgos una aproximación o simulación de ésta al hacer uso de un ordenador para su ejecución. La gran cantidad de algoritmos ya desarrollados en esta área en busca de un mismo fin nos da a entender la gran cantidad de posibilidades de las que disponemos hoy en día para la solución numérica de un problema. Dado esto, se evidencia la cantidad de métodos y de variantes presentes para trabajar la misma aproximación de la solución.

En cada uno de los métodos implementados, se logra apreciar claramente cada una de las ventajas o desventajas entre ellos, donde por lo general, no hay un método o algoritmo definitivo para el cual se obtengan resultados completamente superiores a los demás en términos de exactitud, costo computacional o rendimiento.

De aquí, se presenta un producto de software que le brinda al usuario la mayor cantidad de recursos tales como los métodos, las ayudas, la documentación, entre otras, las cuales le permitan no solo evaluar problemas con un solo algoritmo, sino que cuente con todas las herramientas a disposición y muy importante, la manera más amigable y sencilla de operación para la solución de gran variedad de problemas matemáticos sin descartar los buenos atributos de calidad.

El entendimiento tanto del problema, como su modelado por parte del usuario es fundamental a la hora del uso del aplicativo ya que el usuario debe tener conocimientos básicos de los métodos allí tratados, además del objetivo al cual desea llegar.

En cuanto al lenguaje de programación Python (versión 3) es una excelente alternativa para los problemas matemáticos, en este caso numéricos, pues cuenta con una cantidad de librerías de uso científico como lo es por ejemplo NumPy, herramientas que facilitan el análisis y desarrollo de los algortimos.

Por otro lado, la desventaja de Python es que su enfoque no es tanto el desarrollo de aplicaciones web por lo que esa parte del desarrollo hace que las cosas sean un poco más complicadas.

Metodos de la primera entrega.

Solución de ecuaciones de una variable

Bisección

Como ejecutar:

biseccion(0,1,0.0000001,100,"log(sin(x)**2 + 1) - 1/2")

Algoritmo y resultados.

```
import math
from py_expression_eval import Parser parser = Parser() def
biseccion(xInferior, xSuperior, tolerancia, maximoIteraciones, f):
    tabla = [] #
    #fxInferior = f(xInferior)
    fxInferior = parser.parse(f).evaluate({"x": xInferior})
    #fxSuperior = f(xSuperior)
    fxSuperior = parser.parse(f).evaluate({"x":
    xSuperior}) if fxInferior == 0: mensaje =
    [str(xInferior) + " es una raiz.", True] # elif
    fxSuperior == 0:
        mensaje = [str(xSuperior) + " es una raiz.", True] #
    elif fxInferior * fxSuperior < 0:
        contadorIteraciones = 1
        xMedio = (xInferior + xSuperior) / 2
        #fxMedio = f(xMedio)
      fxMedio = parser.parse(f).evaluate({"x": xMedio})
  tabla.append([contadorIteraciones, xInferior, xSuperior,
```

```
 xMedio, fxMedio, error = tolerancia + 1 while error >
tolerancia and fxMedio != 0 and contadorIteraciones <
            maximo if fxInferior * fxMedio < 0:
                xSuperior =
            xMedio fxSuperior =
            fxMedio else:
                xInferior = xMedio
            fxInferior = fxMedio
            auxiliar = xMedio
            xMedio = (xInferior + xSuperior) / 2
            #fxMedio = f(xMedio)
            fxMedio = parser.parse(f).evaluate({"x": xMedio}) error =
        abs(xMedio - auxiliar) contadorIteraciones += 1
        tabla.append([contadorIteraciones, xInferior, xSuperior, xMedio,
        fxMed if fxMedio == 0:
            mensaje = [str(xMedio) + " es una raiz.", True]
        # elif error < tolerancia:
            mensaje = [str(xMedio) + " es una aproximacion a una raiz con
            una tole
        else:
            mensaje = ["Fracaso en " + str(maximoIteraciones) + "
            iteraciones.", F else:
        mensaje = ["El intervalo es inadecuado.", False] #
 return [tabla, mensaje]
```

```
def main(): a =
    biseccion(0,1,0.0000001,100,"log(sin(x)**2 + 1) - 1/2")
    print('\n') print(a)

if __name__ == "__main__":
      main()


 #+
RE
SUL
TS:
:

: [[[1, 0, 1, 0.5, -0.2931087267313766, 0], [2, 0.5, 1, 0.75, -
0.11839639385347844 :

: ['0.9364045262336731 es una aproximacion a una raiz con una tolerancia =
1e-07',
```

Búsquedas incrementales

Como ejecutar:

```
busquedasIncrementales(-3, 0.5, 100, "log(sin(x)**2 + 1) - 1/2")
```

Algoritmo y resultados.

```
#Algoritmo de busquedas
incrementales import math

from py_expression_eval
import Parser parser =
Parser() #Datos de entrada: #f
(funcion)

#x (x inicial, x0)

#deltax (variacion en x)
#N (Numero maximo de
iteraciones) raices = []

def busquedasIncrementales(f, x, delta, maximoIteraciones):

      tabla = [] #

      #fx = f(x)
```

```
fx =

parser.parse(f).evaluate({"x": x})

tabla.append([x, fx]) # if fx == 0:

    mensaje = [str(x) + " es una raiz.", True] #

else:

    xNuevo = x + delta
    #fxNuevo =
    f(xNuevo)

    fxNuevo = parser.parse(f).evaluate({"x": xNuevo})

    tabla.append([xNuevo, fxNuevo]) # contadorIteraciones = 1

    while fx * fxNuevo > 0 and contadorIteraciones <

    maximoIteraciones:

        x =
        xNu
        evo
        fx =
        fxN
        uev
        o

        xNuevo = x + delta
        #fxNuevo =
        f(xNuevo)

        fxNuevo = parser.parse(f).evaluate({"x":

    xNuevo}) tabla.append([xNuevo, fxNuevo]) #

    contadorIteraciones += 1 if fxNuevo == 0:

        mensaje = [str(xNuevo) + " es una raiz.", True] #

    elif fx * fxNuevo < 0:
```

```
                mensaje = ["Hay una raiz entre " + str(x) + " y " + str(xNuevo),
                True] raices.append([mensaje[0]])

                busquedasIncrementales(f, xNuevo, delta, (maximoIteraciones-

            contadorIt else:

              mensaje = ["Fracaso en " + str(maximoIteraciones), False] #


        #print(tabla)

        #print(mensaje)

        #return [tabla, mensaje] #


def
    mai
    n():
    x0 =
    -3
    delt
    ax =
    0.5
    N =
    100

    f = 'log(sin(x)**2+1) - 1/2'

    #print(busquedasIncrementales(f, x0,
    deltax, N)) busquedasIncrementales(f, x0,
    deltax, N) for fila in raices: print(fila)

if __name__ == "__main__":

    main()


          #+RESULTS:

['Hay una raiz entre -2.5 y -2.0']

['Hay una raiz entre -1.0 y -0.5']

['Hay una raiz entre 0.5 y 1.0']

['Hay una raiz entre 2.0 y 2.5']
```

['Hay una raiz entre 4.0 y 4.5']

['Hay una raiz entre 5.0 y 5.5']

['Hay una raiz entre 7.0 y 7.5']

['Hay una raiz entre 8.0 y 8.5']

['Hay una raiz entre 10.0 y 10.5']

['Hay una raiz entre 11.5 y 12.0']

['Hay una raiz entre 13.5 y 14.0']

['Hay una raiz entre 14.5 y 15.0']

['Hay una raiz entre 16.5 y 17.0']

['Hay una raiz entre 17.5 y 18.0']

['Hay una raiz entre 19.5 y 20.0']

['Hay una raiz entre 21.0 y 21.5']

['Hay una raiz entre 22.5 y 23.0']

['Hay una raiz entre 24.0 y 24.5']

['Hay una raiz entre 26.0 y 26.5']

['Hay una raiz entre 27.0 y 27.5']

['Hay una raiz entre 29.0 y 29.5']

['Hay una raiz entre 30.0 y 30.5']

['Hay una raiz entre 32.0 y 32.5']

['Hay una raiz entre 33.5 y 34.0']

['Hay una raiz entre 35.0 y 35.5']

['Hay una raiz entre 36.5 y 37.0']

['Hay una raiz entre 38.5 y 39.0']

['Hay una raiz entre 39.5 y 40.0']

['Hay una raiz entre 41.5 y 42.0']

['Hay una raiz entre 43.0 y 43.5']

['Hay una raiz entre 44.5 y 45.0']

['Hay una raiz entre 46.0 y 46.5']

Regla falsa

Como

ejecutar:

reglaFalsa(0,1,0.0000001,100,"log(sin(x)**2 + 1) - 1/2")

Algoritmo y resultados.

```
import math
from py_expression_eval import Parser parser = Parser() def
reglaFalsa(xInferior, xSuperior, tolerancia, maximoIteraciones, f):
tabla = [] #

    #fxInferior = f(xInferior)
    fxInferior = parser.parse(f).evaluate({"x": xInferior})
    #fxSuperior = f(xSuperior)
    fxSuperior = parser.parse(f).evaluate({"x":
    xSuperior}) if fxInferior == 0: mensaje =
    [str(xInferior) + " es una raiz.", True] # elif
    fxSuperior == 0:

        mensaje = [str(xSuperior) + " es una raiz.", True] #
    elif fxInferior * fxSuperior < 0:

        contadorIteraciones = 1

            xMedio = xInferior - ((fxInferior * (xSuperior - xInferior)) / (fxSuperior
        #fxMedio = f(xMedio)
        fxMedio = parser.parse(f).evaluate({"x": xMedio})
        tabla.append([contadorIteraciones, xInferior, xSuperior, xMedio,
```

```
        fxMedio, error = tolerancia + 1 while error > tolerancia and fxMedio !=
        0 and contadorIteraciones < maximo if fxInferior * fxMedio < 0:
            xSuperior =
        xMedio fxSuperior =
        fxMedio else:
            xInferior = xMedio
        fxInferior = fxMedio
        auxiliar = xMedio
          xMedio = xInferior - ((fxInferior * (xSuperior - xInferior)) / (fxSupe
        #fxMedio = f(xMedio)
        fxMedio = parser.parse(f).evaluate({"x": xMedio}) error =
      abs(xMedio - auxiliar) contadorIteraciones += 1
      tabla.append([contadorIteraciones, xInferior, xSuperior, xMedio,
      fxMed if fxMedio == 0:
          mensaje = [str(xMedio) + " es una raiz.", True]
      # elif error < tolerancia:
     mensaje = [str(xMedio) + " es una aproximacion a una raiz con una tole
      else: mensaje = ["Fracaso en " + str(maximoIteraciones),
  False] # else:
      mensaje = ["El intervalo es inadecuado.", False] #
  return [tabla, mensaje] #

def main(): a = reglaFalsa(0,1,0.0000001,100,"log(sin(x)**2
    + 1) - 1/2") print('\n') print(a)
```

if __name__ == "__main__":

main()

#+RESULTS:

:

: [[[1, 0, 1, 0.9339403807182157, -0.0014290767036854723, 0], [2, 0.93394038071821

: [’0.936404580879889 es una aproximacion a una raiz con una tolerancia = 1e-07’,

Punto fijo

Como ejecutar:

puntoFijo(-0.5,0.0000001,100,"log(sin(x)**2 + 1) - 1/2","log(sin(x)**2 + 1) - 1/2")

Algoritmo y resultados.

import math

from py_expression_eval import

Parser parser = Parser()

def puntoFijo(x, tolerancia, maximoIteraciones, f, g):

tabla = [] #

#fx = f(x)

fx = parser.parse(f).evaluate({"x": x})

contadorIteraciones = 0 error = tolerancia + 1

tabla.append([contadorIteraciones, x,fx, 0]) # while error > tolerancia and

contadorIteraciones < maximoIteraciones and fx !=

#xNuevo = g(x)

xNuevo = parser.parse(g).evaluate({"x": x})

#fx = f(xNuevo)

```
            fx = parser.parse(f).evaluate({"x":
            xNuevo}) error = abs(xNuevo - x) x =
            xNuevo contadorIteraciones += 1
            tabla.append([contadorIteraciones, x,
            fx, error]) #

        if fx == 0:
            mensaje = [str(x) + " es una raiz.", True] #
        elif error <= tolerancia:

            mensaje = [str(x) + " es una aproximacion con tolerancia " +
        str(toleranci else:

            mensaje = ["Fracaso en " + str(maximoIteraciones) + " iteraciones.",
        False return [tabla, mensaje]

def main(): a = puntoFijo(-0.5,0.0000001,100,"log(sin(x)**2 + 1) -
    1/2","log(sin(x)**2 + 1 print('\n') print(a)

if __name__ == "__main__":
    main()

        #+
RESULTS
: :
: [[[0, -0.5, -0.2931087267313766, 0], [1, -0.2931087267313766, -
0.419821543606257 :
: ['-0.37444505296105535 es una aproximacion con tolerancia 1e-07', True]
```

Newton

Como ejecutar:

newton(0.5,0.0000001,100,"log(sin(x)**2 + 1) - 1/2","(2*sin(x)*cos(x))/(sin(x)**2 + 1)"

Algoritmo y resultados.

```
import math
from py_expression_eval import Parser parser
= Parser() def newton(x, tolerancia,
maximoIteraciones, f, df):
    tabla = [] #
    #fx = f(x)
    fx = parser.parse(f).evaluate({"x": x})
    #derivada = fd(x)
    derivada = parser.parse(df).evaluate({"x": x})
    contadorIteraciones = 0 error = tolerancia + 1
    tabla.append([contadorIteraciones, x, fx, derivada, 0]) # while error >
    tolerancia and contadorIteraciones < maximoIteraciones and fx !=
        xNuevo = x - (fx / derivada)
        #fx = f(xNuevo)
        fx = parser.parse(f).evaluate({"x": xNuevo})
        #derivada = fd(xNuevo)
        derivada = parser.parse(df).evaluate({"x": xNuevo})
        error = abs(xNuevo - x) x = xNuevo contadorIteraciones
        += 1 tabla.append([contadorIteraciones, x, fx, derivada,
        error]) #
    if fx == 0:
```

```
            mensaje = [str(x) + " es una raiz.", True] #
        elif error <= tolerancia:
            mensaje = [str(x) + " es una aproximacion con tolerancia " +
        str(toleranci elif derivada == 0:
            mensaje = [str(xNuevo) + "es una posible raiz multiple", True] #
        else:
            mensaje = ["Fracaso en " + str(maximoIteraciones) + " iteraciones.",
        False return [tabla, mensaje]

def main(): a = newton(0.5,0.0000001,100,"log(sin(x)**2 + 1) -
    1/2","(2*sin(x)*cos(x))/(si print('\n') print(a)

if __name__ == "__main__":
    main()

#+
RE
SU
LT
S:
:

: [[[0, 0.5, -0.2931087267313766, 0.6842068330717285, 0], [1,
0.9283919899125719, :

: ['0.9364045808795621 es una aproximacion con tolerancia 1e-07', True]
```

Secante

Como ejecutar:

secante(0.5,1,0.0000001,100,"log(sin(x)**2 + 1) - 1/2")

Algoritmo y resultados

```
import math
```

```
from py_expression_eval import

Parser parser = Parser()

def secante(x, xNuevo, tolerancia, maximoIteraciones, f):
    tabla = [] #
    #fx = f(x)
    fx = parser.parse(f).evaluate({"x": x})
    if fx == 0:
        mensaje = [str(x) + " es una raiz.", True]
    else:
        #fxNuevo = f(xNuevo)
        fxNuevo = parser.parse(f).evaluate({"x": xNuevo})
        contadorIteraciones
        = 0 error = tolerancia
        + 1 denominador =
        fxNuevo - fx
        tabla.append([contadorIteraciones, x, fx, 0])
        tabla.append([contadorIteraciones + 1, xNuevo, fxNuevo, 0]) while
        error > tolerancia and fxNuevo != 0 and denominador != 0 and
        contado xAuxiliar = xNuevo - fxNuevo * (xNuevo - x) / denominador
        error = abs(xAuxiliar - xNuevo)
            x =
            xNu
            evo
            fx =
            fxN
            uev
            o
```

```
                xNuevo = xAuxiliar
                #fxNuevo =
                f(xNuevo)
                fxNuevo = parser.parse(f).evaluate({"x":
                xNuevo}) denominador = fxNuevo - fx
                contadorIteraciones += 1
                tabla.append([contadorIteraciones + 1,
                xNuevo, fxNuevo, error]) if fxNuevo == 0:
                mensaje = [str(xNuevo) + " es una raiz.", True] elif error <
            tolerancia: mensaje = [str(xNuevo) + " es una aproximacion a una raiz
            con toleranc elif denominador == 0:
                mensaje = ["Hay una posible raiz multiple", True]
            else:
                mensaje = ["Fracaso en " + str(maximoIteraciones) + "
        iteraciones.", F return [tabla, mensaje]

def main(): a = secante(0.5,1,0.0000001,100,"log(sin(x)**2
    + 1) - 1/2") print('\n') print(a)

if __name__ == "__main__":
    main()

        #+
RESULTS
: :
: [[[0, 0.5, -0.2931087267313766, 0], [1, 1, 0.03536607938024017, 0], [2,
0.946166 :
: ['0.9364045808795615 es una aproximacion a una raiz con tolerancia=1e-
07', True
```

Raices multiples

Como ejecutar:

raicesMultiples(1,0.0000001,100,"exp(x) - x - 1","exp(x) - x","exp(x)")

Algoritmo y resultado

```
import math
from py_expression_eval import Parser parser = Parser() def
raicesMultiples(x, tolerancia, maximoIteraciones, f, df, ddf):
    tabla = []
    #fx = f(x)
    fx = parser.parse(f).evaluate({"x": x})
    #fdx = fd(x)
    fdx = parser.parse(df).evaluate({"x": x})
    #fddx = fdd(x)
    fddx = parser.parse(ddf).evaluate({"x": x})
    contadorIteraciones = 0 error = tolerancia + 1
    tabla.append([contadorIteraciones, x, fx, fdx, fddx, 0]) while error > tolerancia
    and contadorIteraciones < maximoIteraciones and fx != 0 an xNuevo = x - ((fx *
    fdx) / (math.pow(fdx, 2) - fx * fddx)) #Habia un error, est
        #fx = f(xNuevo)
        fx = parser.parse(f).evaluate({"x": xNuevo})
        #fdx = fd(xNuevo)
        fdx = parser.parse(df).evaluate({"x": xNuevo})
        #fddx = fdd(xNuevo)
        fddx = parser.parse(ddf).evaluate({"x":
        xNuevo}) error = abs(xNuevo - x) x =
        xNuevo contadorIteraciones += 1
```

```
            tabla.append([contadorIteraciones, x, fx,

            fdx, fddx, error]) if fx == 0:

            mensaje = [str(x) + " es una raiz.", True]

        elif error <= tolerancia:

            mensaje = [str(x) + " es una aproximacion a una raiz con tolerancia=" +

        str(tol elif fdx == 0:

            mensaje = ["Hay una posible raiz multiple", True]

        else:

            mensaje = ["Fracaso en " + str(maximoIteraciones) + " iteraciones.", False]

        return [tabla, mensaje]

def main(): hx
        =
        "exp(x)-
        x-1" dhx
        =
        "exp(x)-
        1" ddhx =
        "exp(x)"
        TOL =
        0.000000
        1 N =
        100

        x0 = 1

        resultado = raicesMultiples(1, TOL, N, hx, dhx, ddhx)
        print(resultado)

if __name__ == "__main__":

        main()

  #+RESULTS:
```

: [[[0, 1, 0.7182818284590451, 1.718281828459045, 2.718281828459045, 0], [1, -0.2342106 :

: ['-4.218590698935789e-11 es una raiz.', True]

Solución de sistemas de ecuaciones lineales

De nir A, b y la matriz aumentada Ab

True imprime las etapas.

A =[

[2, -1, 0, 3],

[1, 0.5, 3, 8],

[0, 13, -2, 11],

[14, 5, -1, 3]

]

b = [1, 1, 1, 1]

Ab = matrizAumentada(A, b)

Eliminación gaussiana

Como ejecutar:

gaussianaSimple(Ab,

len(Ab), True)

Algoritmo y resultado def valorMatriz(Matriz): nuevaMatriz = [] for i in range(len(Matriz)): fila = [] for j in range(len(Matriz[0])): fila.append(Matriz[i][j]) nuevaMatriz.append(fila) return nuevaMatriz

```
def matrizAumentada(A, b):
    matrizAumentada = []
    copiaDeA =
    valorMatriz(A) for i in
    range(len(copiaDeA)):
        copiaDeA[i].append(b[i])
    matrizAumentada.append(copiaDeA[i])
    return matrizAumentada

def imprimirMatriz(Matriz):
    for fila in Matriz:
    print(fila)

def sustitucionRegresiva(Ab):
    x = []
    for pivotIndex in range((len(Ab) - 1), -1, -1):
        summation = i =    0 for column in
        range((len(Ab[0]) - 2), pivotIndex, -1):
            summation +=   Ab[pivotIndex][column] * x[i]
             i += 1
        x.append( ( Ab[pivotIndex][len(Ab[0]) - 1] - summation ) /
    Ab[pivotIndex][ return x[::-1]

def gaussianaSimple(AbParam, n, printEtapas):
    #AbParam es la matriz aumentada Ab
```

```
#n =
len(AbParam)
etapas = []

Ab = valorMatriz(AbParam)

etapas.append(valorMatriz(Ab))


# Eliminaci n

for k in

range(n - 1):

    multiplicadoresDeEtapa = [] for i

    in range(k + 1, n): multiplicator =

    Ab[i][k] / Ab[k][k] for j in range(k,

    n + 1):

            Ab[i][j] = Ab[i][j] - ( multiplicator * Ab[k][j] )

    etapas.append(valorMatriz(Ab))


if printEtapas == True:

    for i in range (len(etapas)):
        print("Etapa",i,":")
        imprimirMatriz(etapas[i])
        print(" ")

#print("Etapa ", len(etapas), ": ")

#imprimirMatriz(etapas[-1]) #etapas[-1] es lo mismo que decir etapas en
la ult

#Sustitucion regresiva:

x =
sustitucionRegresiva
(Ab)
```

```
    #imprimirMatriz(x)
    return[x]

def main():
    A = [
        [2, -1, 0, 3],
        [1, 0.5, 3, 8],
        [0, 13, -2, 11],
        [14, 5, -2, 3]
    ]
    b = [1, 1, 1, 1]
    Ab = matrizAumentada(A, b)

    print("Matriz A: ") imprimirMatriz(A) print("Vector b: ")
    imprimirMatriz(b) print("Matriz Ab: ")
    imprimirMatriz(Ab) imprimirEtapas = True
    print("Despues de aplicar sustitucion Gaussiana
    simple: ") resultado = gaussianaSimple(Ab, len(Ab),
    imprimirEtapas) print("resultados x: ")
    imprimirMatriz(resultado)



if __name__ ==
    "__main__": main()

      #+RESULTS:

#+begin_exa
mple Matriz
A:

[2, -1, 0, 3]

[1, 0.5, 3, 8]

[0, 13, -2,
11] [14, 5, -
2, 3] Vector
b:
```

1

1

1

1
M
at
ri
z
A
b:

[2, -1, 0, 3, 1]

[1, 0.5, 3, 8, 1]

[0, 13, -2, 11, 1]

[14, 5, -2, 3, 1]

Despues de aplicar sustitucion 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]

[0.0, 1.0, 3.0, 6.5, 0.5]

[0.0, 13.0, -2.0, 11.0, 1.0]
[0.0, 12.0, -2.0, -18.0, -
6.0]

Etapa 2 :

[2, -1, 0, 3, 1]

[0.0, 1.0, 3.0, 6.5, 0.5]

[0.0, 0.0, -41.0, -73.5, -5.5]
[0.0, 0.0, -38.0, -96.0, -12.0]

Etapa 3 :

[2, -1, 0, 3, 1]

[0.0, 1.0, 3.0, 6.5, 0.5]

[0.0, 0.0, -41.0, -73.5, -5.5]

[0.0, 0.0, 0.0, -27.878048780487802, -6.902439024390244]

resultados x:

[0.03849518810148722, -0.18022747156605434, -0.30971128608923887, 0.24759405074365 #+end_example

Pivoteo parcial

Como ejecutar:

gaussianaPivoteoParcial(Ab, len(Ab), True)

Algoritmo y

resultado import

math

```
def valorMatriz(Matriz):
    nuevaMatriz = [] for i in
    range(len(Matriz)): fila = []
    for j in range(len(Matriz[0])):
            fila.append(Matriz[i][j])
nuevaMatriz.append(fila) return
nuevaMatriz def matrizAumentada(A,
b): matrizAumentada = [] copiaDeA =
valorMatriz(A) for i in
range(len(copiaDeA)):
```

```
copiaDeA[i].append(b[i])
matrizAumentada.append(copiaDeA[
i]) return matrizAumentada

def imprimirMatriz(Matriz):
    for fila in Matriz: print(fila) def
intercambiarFilas(M, indexA,
indexB):

    aux = M[indexA]
M[indexA] =
M[indexB] M[indexB]
= aux def
pivoteoParcial(Ab, k,
n): mayor =
math.fabs(Ab[k][k])
filaMayor = k for i in
range(k + 1, n):
        if math.fabs(Ab[i][k]) > mayor:
            mayor =
    Ab[i][k] filaMayor
    = i if mayor == 0:
        print("El sistema no tiene soluci n
œnica.") elif filaMayor != k:
intercambiarFilas(Ab, k, filaMayor) return
filaMayor def sustitucionRegresiva(Ab):

    x = []
    for pivotIndex in range((len(Ab) - 1), -1, -1):
```

```
            summation = i =      0 for column in
            range((len(Ab[0]) - 2), pivotIndex, -1):
                summation +=     Ab[pivotIndex][column] * x[i]
                 i += 1
            x.append( ( Ab[pivotIndex][len(Ab[0]) - 1] - summation ) /
Ab[pivotIndex][ return x[::-1] def gaussianaPivoteoParcial(AbParam, n,
imprimirEtapas):
    etapas = []
    etapasPrevias =
    [] filaMayorList =
    []
    Ab =
    valorMatriz(AbPar
    am) # Eliminaci n
    for k in range(n -
    1):
        etapasPrevias.append(valorMatr
        iz(Ab)) filaMayor =
        pivoteoParcial(Ab, k, n)
        filaMayorList.append(filaMayor)
        etapas.append(valorMatriz(Ab))
        multiplicadoresDeEtapa = [] for i
        in range(k + 1, n): multiplicator =
        Ab[i][k] / Ab[k][k] for j in range(k,
        n + 1):
```

```
                    Ab[i][j] = Ab[i][j] - ( multiplicator * Ab[k][j] )


        etapasPrevias.append(valorM
        atriz(Ab))
        etapas.append(valorMatriz(Ab
        )) if imprimirEtapas == True:

            for i in range (len(etapas)):
                print("Etapa",i,":")
                imprimirMatriz(etapas[i])

                print(" ")
        #Sustitucion
        regresiva x =
        sustitucionRegresiva
        (Ab)

        return [etapas, x, filaMayorList, etapasPrevias]

def main():

        A = [

            [2, -1, 0, 3],

            [1, 0.5, 3, 8],

            [0, 13, -2, 11],

            [14, 5, -2, 3]

        ]

        b = [1, 1, 1, 1]

        Ab = matrizAumentada(A, b)

        print("Matriz A: ") imprimirMatriz(A) print("Vector b: ")
        imprimirMatriz(b) print("Matriz Ab: ") imprimirMatriz(Ab)
        imprimirEtapas = True print("Despues de aplicar sustitucion
        Gaussiana con pivoteo parcial: ") resultado =
        gaussianaPivoteoParcial(Ab, len(Ab), imprimirEtapas) x =
        resultado[1] imprimirMatriz(x)


if __name__ ==
        "__main__": main()
```

#+RESULTS:

#+begin_exa
mple Matriz
A:

[2, -1, 0, 3]

[1, 0.5, 3, 8]

[0, 13, -2,
11] [14, 5, -
2, 3] Vector
b:

1

1

1

1

Matriz Ab:

[2, -1, 0, 3, 1]

[1, 0.5, 3, 8, 1]

[0, 13, -2, 11, 1]

[14, 5, -2, 3, 1]

Despues de aplicar sustitucion Gaussiana con pivoteo parcial:

Etapa 0 :

[14, 5, -2, 3, 1]

[1, 0.5, 3, 8, 1]

[0, 13, -2, 11,
1] [2, -1, 0, 3,
1]

Etapa 1 :

[14, 5, -2, 3, 1]

[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, 0.8571428571428

Etapa 2 :

[14, 5, -2, 3, 1]

[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, 0.9890109890

Etapa 3 :

[14, 5, -2, 3, 1]

[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]

0.0384951881014873

-0.18022747156605426

-0.30971128608923887

0.24759405074365706

#+end_example

Pivoteo total

Como ejecutar:

gaussianaPivoteoTotal(Ab, len(Ab), True)

Algoritmo y

resultado import

math

```
def valorMatriz(Matriz):
    nuevaMatriz = [] for i in
```

```
        range(len(Matriz)): fila = []
        for j in range(len(Matriz[0])):
                    fila.append(Matriz[i][j])
nuevaMatriz.append(fila) return
nuevaMatriz def matrizAumentada(A,
b): matrizAumentada = [] copiaDeA =
valorMatriz(A) for i in
range(len(copiaDeA)):
copiaDeA[i].append(b[i])
matrizAumentada.append(copiaDeA[
i]) return matrizAumentada

def imprimirMatriz(Matriz):
for fila in Matriz: print(fila)
def
sustitucionRegresiva(Ab):
        x = []
    for pivotIndex in range((len(Ab) - 1), -1, -1):
                summation = i =       0 for column in
            range((len(Ab[0]) - 2), pivotIndex, -1):
                    summation +=     Ab[pivotIndex][column] * x[i]
                     i += 1
            x.append( ( Ab[pivotIndex][len(Ab[0]) - 1] - summation ) /
Ab[pivotIndex][ return x[::-1] def intercambiarFilas(M, indexA, indexB):
```

```
    aux = M[indexA] M[indexA] = M[indexB]
M[indexB] = aux def intercambiarColumnas(M,
indexA, indexB, marcas):

    auxMarcas =
    marcas[indexA]
    marcas[indexA] =
    marcas[indexB]
    marcas[indexB] =
    auxMarcas for i in
    range(len(M)):

        aux = M[i][indexA]
        M[i][indexA] =
        M[i][indexB]
        M[i][indexB] = aux


def pivoteoTotal(Ab, k, n, marcas):

    mayor = 0

    filaMayor = k
    columnaMayor =
    k

    for i in range(k,
        n): for j in
        range(k, n):

            if math.fabs(Ab[i][j]) > mayor:

                mayor =
    math.fabs(Ab[i][j]) filaMayor = i
    columnaMayor = j if mayor == 0:

        print("El sistema no tiene soluci n œnica.")

    else:
```

```
        if filaMayor != k:
        intercambiarFilas(Ab, k,
        filaMayor) if columnaMayor != k:
            intercambiarColumnas(Ab, k, columnaMayor,
    marcas) return [filaMayor, columnaMayor]

def gaussianaPivoteoTotal(AbParam, n, imprimirEtapas):
    etapas = [] etapasPrevias =
    [] mayorList = [] marcas = []
    for i in range(n):
    marcas.append(i)
    Ab =
    valorMatriz(AbPar
    am) # Eliminaci n
    for k in range(n -
    1):
    etapasPrevias.ap
    pend(valorMatriz(
    Ab)) mayorIndex
    =
    pivoteoTotal(Ab,
    k, n, marcas)
```

```
            filaMayor = mayorIndex[0]
            columnaMayor =
            mayorIndex[1]

            mayorList.append([filaMayor,

            columnaMayor])

            etapas.append(valorMatriz(Ab))

            multiplicadoresDeEtapa = [] for i in

            range(k + 1, n): multiplicator =

            Ab[i][k] / Ab[k][k] for j in range(k, n +

            1):

                        Ab[i][j] = Ab[i][j] - ( multiplicator * Ab[k][j] )


      etapasPrevias.append(valorM
      atriz(Ab))
      etapas.append(valorMatriz(Ab
      )) if imprimirEtapas == True:

            for i in range (len(etapas)):
                  print("Etapa",i,":")
                  imprimirMatriz(etapas[i])

                  print(" ") #
      Sustituci n regresiva
      x =
      sustitucionRegresiva
      (Ab)


      return [etapas, x, mayorList, etapasPrevias, marcas]

def main():

      A = [

            [2, -1, 0, 3],

            [1, 0.5, 3, 8],

            [0, 13, -2, 11],
```

```
        [14, 5, -2, 3]
    ]
    b = [1, 1, 1, 1]
    Ab = matrizAumentada(A, b)

    print("Matriz A: ")
    imprimirMatriz(A)
    print("Vector b: ")
    imprimirMatriz(b)
    print("Matriz Ab: ")
    imprimirMatriz(Ab)
    imprimirEtapas = True
    print("**************************************************************************
    print("Despues de aplicar sustitucion Gaussiana con pivoteo total: ")
    resultado = gaussianaPivoteoTotal(Ab, len(Ab), imprimirEtapas)
    print("Resultados x: ") x = resultado[1] imprimirMatriz(x)

if __name__ == "__main__":
    main()

      #+RESULTS:

#+begin_exa
mple Matriz
A:

[2, -1, 0, 3]

[1, 0.5, 3, 8]

[0, 13, -2,
11] [14, 5, -
2, 3] Vector
b:

1

1

1

1

Matriz Ab:

[2, -1, 0, 3, 1]
```

[1, 0.5, 3, 8, 1]

[0, 13, -2, 11, 1]

[14, 5, -2, 3, 1]

****

Despues de aplicar sustitucion Gaussiana con pivoteo total:

Etapa 0 :

[14, 5, -2, 3, 1]

[1, 0.5, 3, 8, 1]

[0, 13, -2, 11,
1] [2, -1, 0, 3,
1]

Etapa 1 :

[14, 5, -2, 3, 1]

[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, 0.8571428571428

Etapa 2 :

[14, 5, 3, -2, 1]

[0.0, 13.0, 11.0, -2.0, 1.0]

[0.0, 0.0, 7.664835164835164, 3.1648351648351647, 0.9175824175824177]

[0.0, 2.220446049250313e-16, 4.021978021978022,
0.021978021978021955, 0.9890109890

Etapa 3 :

[14, 5, 3, -2, 1]

[0.0, 13.0, 11.0, -2.0, 1.0]

[0.0, 0.0, 7.664835164835164, 3.1648351648351647,
0.9175824175824177] [0.0, 2.220446049250313e-16, 0.0, -
1.638709677419355, 0.5075268817204301]

Resultados x:

0.038495188101487325 - 0.18022747156605423

0.24759405074365703 - 0.3097112860892388

#+end_example

Métodos de la segunda entrega

Datos a usar. Nmax = 100; Tol = 1e-7 Matriz A =

4	-1	0	3
1	15.5	3	8
0	-1.3	-4	1.1
14	5	-2	30

Vector b =

1
1
1
1

X0 =

0
0
0
0

Tabla =

x	-1 0 3 4 y 15. 3 8 1
5	

Solución de sistemas de ecuaciones lineales

LU con gaussiana simple, Lu con pivoteo parcial

Algoritmo

```
import numpy as np
from math import sqrt
from tabulate import
tabulate

def getOption(message, options):
    while True:
        ans =
        input(message)
        for i in options:
            if i == ans:
                return i
def interRow(array, row1, row2):
    if row1 != row2:
        aux = array[row1].copy()
        array[row1] = array[row2]
        array[row2] = aux

def interCol(array, col1, col2):
    if col1 != col2:
        aux = array[:,col1].copy()
        array[:,col1] = array[:,col2]
        array[:,col2] = aux


def solNumpy(array):
    A =
    array[:,0:-
    1] B =
    array[:,-1]
    solution = np.linalg.solve(A, B)
    return solution
def sumMultRow(array, rowAct, rowMult, mult, init):
```

```
        array[rowAct,init:] = array[rowAct,init:] + mult * array[rowMult,init:]

def checkSquare(array):

        rows = len(array)
        for i in range
        (0,rows):

                if len(array[i]) != rows:

                        raise Exception('La matriz debe ser cuadrada')

        return rows

def checkDet(array):

        det = np.linalg.det(array)
        print('Determinante: ' + str(det))

        if det == 0: raise Exception('La determinante de la matriz debe ser
        diferente a tol = 10e-4 if abs(det) < tol:

                option = getOption('La determinante es menor a ' + str(tol)

                        + ' y puede presentar problemas de evaluaci n       Desea
continuar?

                if option == 'n': raise Exception('Operaci n abortada por el usuario')


def pivoteoParcial(array, col, dimension):

        maxRow = np.argmax(abs(array[col:,col])) + col

        interRow(array,col,maxR
        ow) return

def progSustitution(array):

        print('---Sustituci n progresiva-
        --') dimension = len(array)
        solution = []

        #np.dot(A,B)

        for row in range(0, dimension):

                variable = (array[row,dimension] -
                np.dot(array[row,0:row],solution)) / solution.append(variable)
```

```
        #solNumpy(array)
        return
        np.array(solution)

def LUGauss(matriz):

        dimension = checkSquare(matriz)
        checkDet(matriz)

        cont = 0

        print('---Etapa 0, matriz original---') print(tabulate(matriz,floatfmt='.6f'))

        U =
        np.zeros((dimension,dimens
        ion)) L =
        np.identity(dimension) for
        col in range(0,dimension):

                for row in range(col+1,dimension):

                        mult = -(matriz[row,col]/matriz[col,col])
                        L[row,col] = -mult

                    sumMultRow(matriz,row,col,mult,col)

                U[col] =
                matriz[col] cont
                += 1

                print('---Etapa ' + str(cont) + '---') print('L')

                print(tabulate(L,floatfmt='.6f'))
                print('U')

                print(tabulate(U,floatfmt='.6f'))

        return L,U

def LUGaussVector(matriz, vector):

        dimension = len(matriz)

        array =
        np.zeros((dimension,dimension + 1))
        array[:,:-1] = matriz array[:,-1:] =
        vector numpySol = solNumpy(array)

        L, U = LUGauss(matriz)
```

```
        Larr = np.zeros((dimension,dimension + 1))

        Larr[:,:-1] = L

        Larr[:,-1:] = vector print('Dado el sistema Lz =
        b') print(tabulate(Larr,floatfmt='.6f'))
        print('Aplicamos sustituci n progresiva y
        obtenemos') pSus = progSustitution(Larr)

        print(pSus)

        Uarr = np.zeros((dimension,dimension + 1))

        Uarr[:,:-1] = U Uarr[:,-1] = pSus print('Dado el
        sistema Ux = z')
        print(tabulate(Uarr,floatfmt='.6f'))
        print('Aplicamos sustituci n regresiva y
        obtenemos')

        print(sustitution(Uarr))

        print('---Opcional---') print('La soluci n
        de    es: ' + str(numpySol))


def LUGaussPivoteoParcial(matriz):

        dimension = checkSquare(matriz)
        checkDet(matriz)

        cont = 0

        print('---Etapa 0, matriz original---') print(tabulate(matriz,floatfmt='.6f'))

        A = matriz

        P = np.identity(dimension) L
        =
        np.zeros((dimension,dimens
        ion)) for col in
        range(0,dimension):

         maxRow = np.argmax(abs(A[col:,col])) + col

                interRow(A,col,maxRow)
                interRow(P,col,maxRow)
                interRow(L,col,maxRow)
```

```
                for row in
                        range(col+1,dimension):
                        mult = -
                        (A[row,col]/A[col,col])
                        L[row,col] = -mult
                        sumMultRow(A,row,col,
                        mult,col)

                #U[col] =     matriz[col]

                cont += 1

                print('---Etapa ' + str(cont) + '---') print('P')

                print(tabulate(P,floatfmt='.6f'))
                print('A')

                print(tabulate(A,floatfmt='.6f'))

        print('--- ltima etapa--
        -') U = np.array(A,
        copy=True) print('U
        = A =')

        print(tabulate(U,floatfmt='.6f'))

        print('Se construye la matriz L con los multiplicadores
        almacenados') np.fill_diagonal(L, 1) print('L')

        print(tabulate(L,floatfmt='.6f'))
        print('P')

        print(tabulate(P,floatfmt='.6f'))
        return L,U,P

def LUGaussPivoteoParcialVector(matriz, vector):

        dimension = len(matriz)

        array =
        np.zeros((dimension,dimension + 1))
        array[:,:-1] = matriz array[:,-1:] =
        vector numpySol = solNumpy(array)

        L, U, P =
        LUGaussPivoteoParcial(matriz) Bn
        = np.matmul(P,vector)
        print('Calculamos el producto Pb =
        Bn') print(tabulate(Bn,floatfmt='.6f'))
```

```
        Larr =
        np.zeros((dimension,dimension +
        1))

        Larr[:,:-1] = L Larr[:,-1:] = Bn
        print('Dado el sistema Lz =
        Bn')
        print(tabulate(Larr,floatfmt='.6
        f'))

        print('Aplicamos sustituci n progresiva y
        obtenemos') pSus = progSustitution(Larr)

        print(pSus)

        Uarr = np.zeros((dimension,dimension + 1))

        Uarr[:,:-1] = U

        Uarr[:,-1] = pSus

        print('Dado el sistema Ux = z')
        print(tabulate(Uarr,floatfmt='.6f'))

        print('Aplicamos sustituci n regresiva y obtenemos')

        print(sustitution(Uarr))

        print('---Opcional---')

          print('La soluci n de       es: ' + str(numpySol))


def sustitution(array):

        print('---Sustituci n regresiva---')
        dimension = len(array) solution =
        [] #np.dot(A,B) for row in
        range(dimension - 1, -1, -1):

            variable = (array[row,dimension] - np.dot(array[row,row + 1:dimension],

                #print('X' + str(row) + ' = ' + str(variable))
                solution.insert(0,variable)

        #solNumpy(array)
        return
        np.array(solution)
```

```
A = np.array([[4,-1,0,3],[1,15.5,3,8],[0,-1.3,-4,1.1], [14,5,-2,30]],dtype=np.float64) x
= np.array([[1],[1],[1],[1]],dtype=np.float64)


try:

        LUGaussVector(A,x) # LU con gauss simple si dan matriz y vector

        LUGaussPivoteoParcialVector(A,x) # LU con pivoteo parcial si dan matriz
        y vecto

except Exception as msg:

        print(msg)
```

1. LU con gaussiana simple

    Como ejecutarlo

```
    A = np.array([[4,-1,0,3],[1,15.5,3,8],[0,-1.3,-4,1.1], [14,5,-2,30]],dtype=np.floa
x = np.array([[1],[1],[1],[1]],dtype=np.float64)

LUGaussVector(A,x) # LU con gauss simple si dan matriz y vector
```

    Resultado

```
Determinante: -3297.5999999999976

---Etapa 0, matriz original---

---------          ---------        ---------          ---------
 4.000000   -1.000000   0.000000   3.000000
 1.000000 15.500000 3.000000 8.000000
 0.000000  -1.300000  -4.000000  1.100000
14.000000 5.000000 -2.000000 30.000000 -
-------- --------- --------- ---------
---Etapa 1---
L
--------         --------     --------        --------
1.000000 0.000000   0.000000  0.000000
0.250000 1.000000   0.000000  0.000000
0.000000 0.000000   1.000000  0.000000
3.500000 0.000000   0.000000  1.000000
```

--- --- --- ---

U

--- --- --- ---

4.000000 -1.000000 0.000000 3.000000

0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000

--- --- --- ---

Etapa 2--L

--- --- --- ---

1.000000 0.000000 0.000000 0.000000

0.250000 1.000000 0.000000 0.000000

0.000000 -0.082540 1.000000 0.000000

3.500000 0.539683 0.000000 1.000000

--- --- --- ---

U

--- --- --- ---

4.000000 -1.0000000.000000 3.000000

0.000000 15.7500003.000000 7.250000

0.000000 0.0000000.000000 0.000000

0.000000 0.0000000.000000 0.000000

--- --- --- ---

Etapa 3--L

--- --- ---

1.000000 0.0000000.000000 0.000000

0.250000 1.0000000.000000 0.000000

0.000000 -0.0825401.000000 0.000000

3.500000 0.5396830.964467 1.000000

--- --- ---

U

--- --- ---

4.000000 -1.000000 0.000000 3.000000

0.000000 15.750000 3.000000 7.250000

0.000000 0.000000-3.752381 1.698413

0.000000 0.000000 0.000000 0.000000

--- --- --- ---

Etapa 4--L

--- --- ---

1.000000 0.0000000.000000 0.000000

0.250000 1.0000000.000000 0.000000

0.000000 -0.0825401.000000 0.000000
3.500000 0.5396830.964467 1.000000
-------- ---------- --------
U
-------- ---------- ---------
4.000000 -1.000000 0.000000 3.000000
0.000000 15.750000 3.000000 7.250000
0.000000 0.000000-3.752381 1.698413
0.000000 0.000000 0.000000 13.949239
-------- ---------- ---------
Dado el sistema Lz = b
-------- --------- ---------- --------
1.000000 0.000000 0.0000000.000000 1.000000
0.250000 1.000000 0.0000000.000000 1.000000
0.000000 -0.082540 1.0000000.000000 1.000000
3.500000 0.539683 0.964467 1.000000
1.000000 -------- --------- -------- -------- -------
Aplicamos sustituci n progresiva y obtenemos

---Sustituci n progresiva---

[1. 0.75 1.06190476 -3.92893401]

Dado el sistema Ux = z

-------- --------- --------- --------- ---------
4.000000 -1.000000 0.000000 3.000000 1.000000
0.000000 15.750000 3.000000 7.250000 0.750000
0.000000 0.000000 -3.752381 1.698413 1.061905
0.000000 0.000000 0.000000 13.949239 -3.928934
-------- --------- --------- --------- ---------
Aplicamos sustituci n regresiva y obtenemos

---Sustituci n regresiva---

[0.52510917 0.25545852 -0.41048035 -0.28165939]

---Opcional---

La soluci n de es: [0.52510917 0.25545852 -0.41048035 -0.28165939]

2. LU con pivoteo parcial

Como ejecurlo

A = np.array([[4,-1,0,3],[1,15.5,3,8],[0,-1.3,-4,1.1], [14,5,-2,30]],dtype=np.floa x = np.array([[1],[1],[1],[1]],dtype=np.float64)

LUGaussPivoteoParcialVector(A,x) # LU con pivoteo parcial si dan matriz y vector

Resultado

Determinante: -3297.5999999999976

---Etapa 0, matriz original---

4.000000	-1.000000	0.000000	3.000000
1.000000	15.500000	3.000000	8.000000
0.000000	-1.300000	-4.000000	1.100000
14.000000	5.000000	-2.000000	30.000000

---Etapa 1---

P

0.000000	0.000000	0.000000	1.000000
0.000000	1.000000	0.000000	0.000000
0.000000	0.000000	1.000000	0.000000
1.000000	0.000000	0.000000	0.000000

A

14.000000	5.000000	-2.000000	30.000000
0.000000	15.142857	3.142857	5.857143
0.000000	-1.300000	-4.000000	1.100000
0.000000	-2.428571	0.571429	-5.571429

---Etapa 2---

P

0.000000	0.000000	0.000000	1.000000
0.000000	1.000000	0.000000	0.000000
0.000000	0.000000	1.000000	0.000000
1.000000	0.000000	0.000000	0.000000

A

14.000000 5.000000 -2.000000 30.000000
0.000000 15.142857 3.142857 5.857143
0.000000 0.000000 -3.730189 1.602830
0.000000 0.000000 1.075472 -4.632075
--- --- --- ---
---Etapa 3---
P

--- --- --- ---
0.000000 0.000000 0.000000 1.000000
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
1.000000 0.000000 0.000000 0.000000
--- --- --- ---
A
--- --- --- ---
14.000000 5.000000 -2.000000 30.000000
0.000000 15.142857 3.142857 5.857143
0.000000 0.000000 -3.730189 1.602830
0.000000 0.000000 0.000000 -
4.169954 --- --- ---
----------Etapa 4---

P

--- --- --- ---
0.000000 0.000000 0.000000
1.000000 0.000000 1.000000
0.000000 0.000000 0.000000
0.000000 1.000000 0.000000
1.000000 0.000000 0.000000
0.000000
--- --- --- ---
A

--- --- ---
14.000000 5.000000 -2.000000 30.000000
0.000000 15.142857 3.142857 5.857143
0.000000 0.000000 -3.730189 1.602830
0.000000 0.000000 0.000000 -4.169954
--- --- ---
--- ltima etapa---
U = A =
--- --- ---
14.000000 5.000000 -2.000000 30.000000

0.000000 15.142857 3.142857 5.857143
0.000000 0.000000 -3.730189 1.602830
0.000000 0.000000 0.000000 -4.169954
-------- ---------- --------

Se construye la matriz L con los multiplicadores almacenados L

-------- -------- -------- --------
1.000000 0.000000 0.000000 0.000000
0.071429 1.000000 0.000000 0.000000
0.000000 -0.085849 1.000000 0.000000
0.285714 -0.160377 -0.288316 1.000000
-------- -------- -------- --------

P

-------- -------- -------- --------
0.000000 0.000000 0.000000 1.000000
0.000000 1.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
1.000000 0.000000 0.000000 0.000000
-------- -------- -------- --------

Calculamos el producto Pb = Bn

1.00000
0
1.00000
0
1.00000
0
1.00000
0 --------

Dado el sistema Lz = Bn

-------- -------- -------- -------- --------
1.000000 0.000000 0.000000 0.000000 1.000000
0.071429 1.000000 0.000000 0.000000 1.000000
0.000000 -0.085849 1.000000 0.000000 1.000000
0.285714 -0.160377 -0.288316 1.000000 1.000000
-------- -------- -------- -------- --------

Aplicamos sustituci n progresiva y obtenemos

---Sustituci n progresiva---

[1. 0.92857143 1.07971698 1.17450683]

Dado el sistema Ux = z

```
---------  ---------  ---------  ---------  --------
14.000000   5.000000  -2.000000  30.000000 1.000000
 0.000000  15.142857   3.142857   5.857143 0.928571
 0.000000   0.000000  -3.730189   1.602830 1.079717
 0.000000   0.000000   0.000000  -4.169954 1.174507
---------  ---------  ---------  ---------  --------
```

Aplicamos sustituci n regresiva y obtenemos

---Sustituci n regresiva---

[0.52510917 0.25545852 -0.41048035 -0.28165939]

---Opcional---

La soluci n de es: [0.52510917 0.25545852 -0.41048035 -0.28165939]

Crout, Doolittle, Cholesky

Algorit

mo import

math

```
def matrizIdentidad(n):
        M = []
        for i in range(n):
                row = [] for
                j in
                range(n):
                        if i == j:
                        row.append(1) else:
                        row.append(0)
                M.append(ro
w) return M def
```

```
valorMatriz(Matriz):
nuevaMatriz = [] for i in
range(len(Matriz)):
                    fila = [] for j in
range(len(Matriz[0])):
fila.append(Matriz[i][j])
nuevaMatriz.append(fila) return
nuevaMatriz def matrizAumentada(A, b):
matrizAumentada = [] copiaDeA =
valorMatriz(A) for i in
range(len(copiaDeA)):
copiaDeA[i].append(b[i])
matrizAumentada.append(copiaDeA[i])
return matrizAumentada

def sustitucionRegresiva(Ab):
        x = []
        for pivotIndex in range((len(Ab) - 1), -1, -1):
                summation = i =0 for column in range((len(Ab[0])
                - 2), pivotIndex, -1):
                        summation +=    Ab[pivotIndex][column] * x[i]
                         i += 1
```

```
                x.append( ( Ab[pivotIndex][len(Ab[0]) - 1] - summation ) /
        Ab[pivotInde return x[::-1]

# ********************************************************************************** def
sustitucionProgresiva(Ab):

        x = []
        for pivotIndex in range(len(Ab)):
                summation = i = 0

                for column in range(pivotIndex):
                    summation += Ab[pivotIndex][column] * x[i]
                        i += 1
                x.append( (Ab[pivotIndex][len(Ab[0]) - 1] - summation) /
        Ab[pivotIndex] return x

def imprimirMatriz(Matriz): for fila in
Matriz: print(fila) def
factorizacionDirecta(A, n, b,
metodo):

        etapas = []

        L =
        matrizIdentidad
        (n) U =
        matrizIdentidad
        (n)

        if metodo == "Crout": for i in
                range(n): L[i][i] = 0
        elif metodo == "Doolittle":
```

```
            for i in range(n): U[i][i] = 0

for k in range(n):

            suma1 = 0 for p in

            range(k): suma1 += L[k][p]

            * U[p][k]


            if metodo == "Crout":

                        L[k][k] = A[k][k] -
            suma1 elif metodo ==
            "Doolittle":

                        U[k][k] = A[k][k] -
            suma1 elif metodo ==
            "Cholesky":

                        try:

                                    L[k][k] = math.sqrt(A[k][k] - suma1)

                                       U[k][k] = L[k][k]
                        except:

                          print("No se puede realizar el metodo porque hay un num

            for i in range(k + 1, n):

                        suma2 = 0

                        for p in

                        range(k):


                                       suma2 += L[i][p] * U[p][k]

                        L[i][k] = (A[i][k] - suma2) / float(U[k][k])

            for j in range(k + 1, n): suma3 = 0 for p in

            range(k):


                                       suma3 += L[k][p] * U[p][j]
```

```
                    U[k][j] = (A[k][j] - suma3) / float(L[k][k])

        etapas.append([valorMatriz(L), valorMatriz(U)]) Lb =

        matrizAumentada(L, b) z = sustitucionProgresiva(Lb)

        Uz = matrizAumentada(U, z) x =

        sustitucionRegresiva(Uz) return [etapas, x, z, Lb, Uz]


def imprimirResultados(etapas, x): contEtapa = 0 for etapa in
        etapas: print("Resumen de la etapa " + str(contEtapa) +
        ": ")

                L =
                etapa[0]
                U =
                etapa[1]
                print("L:
                ") for
                element
                in L:

                    print(element)

                print("U:
                ") for
                element
                in U:

                        print(element)

                contEtapa+=1


        #print(etapas)
        print("Resultado: ")
        print(x)

if __name__ == "__main__":

        A = [[4.0,-1.0,0.0,3.0], [1.0,15.5,3.0,8.0], [0.0,-1.3,-4.0,1.1], [14.0,5.0,-2.

        b =
        [1,1,1,
        1] n =
        len(A)
```

```
etapas, x, z, Lb, Uz = factorizacionDirecta(A,n,b,"Crout")

print("**************************************************************************
print("Resultados Crout: ") imprimirResultados(etapas, x)


etapas, x, z, Lb, Uz = factorizacionDirecta(A,n,b,"Doolittle")

print("**************************************************************************
print("Resultados Doolittle: ") imprimirResultados(etapas, x)

etapas, x, z, Lb, Uz = factorizacionDirecta(A,n,b,"Cholesky")

print("**************************************************************************
print("Resultados Cholesky: ") imprimirResultados(etapas, x)
```

1. Crout

    Como ejecutar Crout

```
A = [[4.0,-1.0,0.0,3.0], [1.0,15.5,3.0,8.0], [0.0,-1.3,-4.0,1.1], [14.0,5.0,-2.0,3 b =
[1,1,1,1] n = len(A)

etapas, x, z, Lb, Uz = factorizacionDirecta(A,n,b,"Crout")

print("**************************************************************************
print("Resultados Crout: ")

imprimirResultados(etapas, x)
```

    Resultado

```
Resultados Crout:

Resumen de la etapa 0:

L:

[4.0, 0, 0, 0]

[1.0, 0, 0, 0]

[0.0, 0, 0, 0]
[14.0, 0, 0,
0]

U:

[1, -0.25, 0.0, 0.75]
```

[0, 1, 0, 0]

[0, 0, 1, 0]

[0, 0, 0, 1]
Resumen de
la etapa 1:

L:

[4.0, 0, 0, 0]

[1.0, 15.75, 0, 0]

[0.0, -1.3, 0, 0]

[14.0, 8.5,
0, 0] U:

[1, -0.25, 0.0, 0.75]

[0, 1, 0.19047619047619047, 0.4603174603174603]

[0, 0, 1, 0]

[0, 0, 0, 1]
Resumen de
la etapa 2:

L:

[4.0, 0, 0, 0]

[1.0, 15.75, 0, 0]

[0.0, -1.3, -
3.7523809523809524, 0]
[14.0, 8.5, -
3.619047619047619, 0] U:

[1, -0.25, 0.0, 0.75]

[0, 1, 0.19047619047619047, 0.4603174603174603]

[0, 0, 1, -0.45262267343485624]

[0, 0, 0, 1]

Resumen de la etapa 3:

L:

[4.0, 0, 0, 0]

[1.0, 15.75, 0, 0]

[0.0, -1.3, -3.7523809523809524, 0]

[14.0, 8.5, -3.619047619047619,
13.949238578680202] U:

[1, -0.25, 0.0, 0.75]

[0, 1, 0.19047619047619047, 0.4603174603174603]

[0, 0, 1, -0.45262267343485624]

[0, 0, 0, 1]

Resultado:

[0.5251091703056769, 0.2554585152838428, -0.4104803493449782, -0.28165938864628826

2. Doolittle

Como ejecutar Doolittle

```
etapas, x, z, Lb, Uz = factorizacionDirecta(A,n,b,"Doolittle")
        print("**********************************************************************
        print("Resultados Doolittle: ") imprimirResultados(etapas, x)
```

Resultado

Resultados Doolittle:

Resumen de la etapa 0:

L:

[1, 0, 0, 0]

[0.25, 1, 0, 0]

[0.0, 0, 1, 0]

[3.5, 0, 0, 1]

U:

[4.0, -1.0, 0.0, 3.0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]
Resumen de
la etapa 1:

L:

[1, 0, 0, 0]

[0.25, 1, 0, 0]

[0.0, -
0.08253968253968254, 1,
0] [3.5,
0.5396825396825397, 0,
1] U:

[4.0, -1.0, 0.0, 3.0]

[0, 15.75, 3.0, 7.25]

[0, 0, 0, 0]

[0, 0, 0, 0]
Resumen de
la etapa 2:

L:

[1, 0, 0, 0]

[0.25, 1, 0, 0]

[0.0, -0.08253968253968254, 1, 0]

[3.5, 0.5396825396825397,
0.9644670050761421, 1] U:

[4.0, -1.0, 0.0, 3.0]

[0, 15.75, 3.0, 7.25]

[0, 0, -3.7523809523809524, 1.6984126984126986]

[0, 0, 0, 0]
Resumen de
la etapa 3:

L:

[1, 0, 0, 0]

[0.25, 1, 0, 0]

[0.0, -0.08253968253968254, 1, 0]

[3.5, 0.5396825396825397,
0.9644670050761421, 1] U:

[4.0, -1.0, 0.0, 3.0]

[0, 15.75, 3.0, 7.25]

[0, 0, -3.7523809523809524,
1.6984126984126986] [0, 0, 0,
13.949238578680202] Resultado:

[0.5251091703056769, 0.2554585152838428, -0.41048034934497823, -0.2816593886462882

No se puede realizar el metodo porque hay un numero negativo en la raiz, no se tra

3. Cholesky

Como ejecutar Cholesky etapas, x, z, Lb, Uz =

factorizacionDirecta(A,n,b,"Cholesky")

print("**********

********** print("Resultados Cholesky: ")

imprimirResultados(etapas, x)

Resultado

Resultados Cholesky:

Resumen de la etapa 0:

L:

[2.0, 0, 0, 0]

[0.5, 1, 0, 0]

[0.0, 0, 1, 0]

[7.0, 0, 0, 1]

U:

[2.0, -0.5, 0.0, 1.5]

[0, 1, 0, 0]

[0, 0, 1, 0]

[0, 0, 0, 1]
Resumen de
la etapa 1:

L:

[2.0, 0, 0, 0]

[0.5, 3.968626966596886, 0, 0]

[0.0, -
0.32756920994133026, 1,
0] [7.0,
2.141798680385621, 0, 1]
U:

[2.0, -0.5, 0.0, 1.5]

[0, 3.968626966596886, 0.7559289460184544, 1.8268282862112648]

[0, 0, 1, 0]

[0, 0, 0, 1]
Resumen de
la etapa 2:

L:

[2.0, 0, 0, 0]

[0.5, 3.968626966596886, 0, 0]

[0.0, -0.32756920994133026, 1, 0]

[7.0, 2.141798680385621, -
3.6190476190476186, 1] U:

[2.0, -0.5, 0.0, 1.5]

[0, 3.968626966596886, 0.7559289460184544, 1.8268282862112648]

[0, 0, 1, 1.6984126984126986]

[0, 0, 0, 1]

Resumen de la etapa 3:

L:

[2.0, 0, 0, 0]

[0.5, 3.968626966596886, 0, 0]

[0.0, -0.32756920994133026, 1, 0]

[7.0, 2.141798680385621, -3.6190476190476186, 4.661967183459393] U:

[2.0, -0.5, 0.0, 1.5]

[0, 3.968626966596886, 0.7559289460184544, 1.8268282862112648]

[0, 0, 1, 1.6984126984126986]
[0, 0, 0, 4.661967183459393]
Resultado:

[0.1774813938930236, -0.16055505320998817, 0.9885789803157822, 0.04317312373930581

Jacobi

Algorit

mo import

math def

jacobi(x, A,

b):

newX = [] n =

len(x) for i in

range(n):

suma = 0 for j

in range(n):

```
                if j != i: suma += A[i][j] * x[j]
newX.append( (b[i] - suma) / A[i][i] )
return newX def norma2Euclidiana(x):
    sumator
    ia = 0.0
    n =
    len(x)
    for j in range(n): sumatoria +=
math.pow(math.fabs(x[j]), 2) return
math.sqrt(sumatoria) def disp(x, y): mayor
= 0 for i in range(len(x)):
        if math.fabs(x[i]-y[i]) > mayor: mayor = math.fabs(x[i]-
y[i]) return mayor def insertar(tabla, xInicial, contador,
dispersion):
    fila = []
    fila.append(contador) for i in xInicial: fila.append(i)
fila.append(dispersion) tabla.append(fila) def
iterativo(previus, tolerancia, maximoIteraciones, A, b):
    tabla = [] contador =
    0 dispersion =
    tolerancia + 1
    insertar(tabla,
```

```
        previus, contador,
        0) while contador <
        maximoIteraciones
        and dispersion >
        tolerancia :

            current = jacobi(previus, A, b)
            #elif metodo == "Seidel": current = seidel(previus, A, b)
            #dispersion = normaInfinito(current) -
        normaInfinito(previus) dispersion = disp(current, previus)
        previus = current contador += 1 insertar(tabla, previus,
        contador, dispersion) if dispersion < tolerancia:
            mensaje = [str(current) + " Es una aprox con tolerancia: "+str(tolerancia),
        Tru else:

            mensaje = ["Fracaso en " + str(maximoIteraciones) + " iteraciones", False]
        return [tabla, mensaje]

if __name__ == "__main__":
        A = [[4.0,-1.0,0.0,3.0], [1.0,15.5,3.0,8.0], [0.0,-1.3,-4.0,1.1], [14.0,5.0,-2.0,30 b =
        [1,1,1,1] x0 = [0,0,0,0] tol = 1e-7 nMax = 100
        resultado = iterativo(x0, tol,nMax,
        A, b) tabla = resultado[0] mensaje
        = resultado[1] for row in tabla:
        print(row) print("Solucion: " +
        mensaje[0])
```

Como ejecutar

```
if __name__ == "__main__":
```

A = [[4.0,-1.0,0.0,3.0], [1.0,15.5,3.0,8.0], [0.0,-1.3,-4.0,1.1], [14.0,5.0,-2.0,30 b = [1,1,1,1] x0 = [0,0,0,0] tol = 1e-7 nMax = 100

resultado = iterativo(x0, tol,nMax,
A, b) tabla = resultado[0] mensaje
= resultado[1] for row in tabla:
print(row)

print("Solucion: " + mensaje[0])

Resultado

[0, 0, 0, 0, 0, 0]

[1, 0.25, 0.06451612903225806, -0.25, 0.03333333333333333, 0.25]

[2, 0.2411290322580645, 0.07956989247311828, -0.2618010752688172, -0.110752688172043,

[3, 0.35295698924731184, 0.15679327089836975, -0.3063172043010753, -0.10990860215053762

[4, 0.37162976933749564, 0.15775893166840096, -0.33118267863336803, -0.1779332870852121

[5, 0.4228896982310093, 0.1964764234351165, -0.3502033067406637, -0.18846589287778936,

[6, 0.4404685255171211, 0.20228692677505142, -0.36568295815780494, -0.22010815019636798

[7, 0.4656528443410388, 0.22048035809853145, -0.3762729925058929, -0.23031199691435209,

[8, 0.4778540872103969, 0.2261717489349327, -0.38499191553346956, -0.24580291987596622,

[9, 0.4908951271407078, 0.23506742054178978, -0.39110162136974386, -0.25302665988957196

[10, 0.4985368500526264, 0.23913696877968385, -0.395979243145714, -0.26100240418061155

[11, 0.5055360453303797, 0.24370452340834892, -0.39949517600306544, -0.2655719743642206

[12, 0.5101051116252526, 0.24629195339306978, -0.4022362630578741, -0.2698339201224397

[13, 0.5139484284400972, 0.24872742184050886, -0.4042492128864186, -0.27258012852782115

[14, 0.5166169518559931, 0.25028646699624046, -0.4057959474433162, -0.27491378443789144

[15, 0.5187569550774787, 0.25161813974045727, -0.40694439249419834, -0.2765220518617245

[16, 0.5202960738314077, 0.2525324282128331, -0.4078194596776229, -0.27781922849251284

[17, 0.5214975284225929, 0.2532720085897783, -0.4084733270046118, -0.278748203135304, 0 [18, 0.5223791544989226, 0.2538005211402371, -0.40896915865388656, -0.27947573649581386

[19, 0.5230569326569197, 0.25421511054382245, -0.4093409969069259, -0.2800083028664625

[20, 0.5235600047858024, 0.25451822458035855, -0.4096221942150195, -0.2804184867909946

[21, 0.5239434212383356, 0.2547519014314331, -0.40983350685614006, -0.28072251927776887

[22, 0.5242298648161849, 0.25492498410014425, -0.4099930607666022, -0.28095448060687145 [23, 0.5244471064801897, 0.2550571072476512, -0.4101131019994365, -0.2811276383153505,

[24, 0.5246100055484256, 0.25515569716393055, -0.410203660392208, -0.2812590410319928,

[25, 0.5247332050649772, 0.2552305357344607, -0.41027183786207544, -0.28135752947606757

[26, 0.5248257810406658, 0.2552866157632122, -0.41032324471961834, -0.28143204084353785

[27, 0.5248956845734565, 0.25532905031396724, -0.4103619613550169, -0.28148801676082064 [28, 0.5249482751491072, 0.255360924746978, -0.410391145961265, -0.2815302919436087, 5.

[29, 0.524987950144451, 0.2553849998892618, -0.4104131308272603, -0.28156209225816403,

[30, 0.5250178191659385, 0.2554031084130737, -0.4104297003350052, -0.2815860854374381,

[31, 0.525040341181347, 0.25541677195732787, -0.41044218372954444, -0.2816041470352839

[32, 0.5250573032657949, 0.25542705717997144, -0.41045159132083464, -0.2816177667928195

[33, 0.5250700893896075, 0.25543481322833966, -0.4104586794515161, -0.2816280238087552

[34, 0.5250797211636513, 0.2554406541570956, -0.4104640208466181, -0.2816357558833078,

[35, 0.5250869804517547, 0.2554450573188816, -0.41046804546896576, -0.2816415802923277

[36, 0.5250924495489662, 0.25544837408346865, -0.4104710782090267, -0.2816459701285635

[37, 0.5250965711172898, 0.2554508739423627, -0.4104733633624823, -0.2816492773506975,

[38, 0.5250996764986138, 0.255452757275854, -0.4104750853027097, -0.2816517697359612, 3 [39, 0.5251020166209344, 0.25545417659981967, -0.4104763827920419, -0.28165364759884276

[40, 0.525103779849087, 0.25545524597070535, -0.41047736048462313, -0.28165506270920876

[41, 0.5251051085245829, 0.2554560518244163, -0.41047809718551165, -0.28165612895699965

[42, 0.5251061096738538, 0.2554566590121258, -0.4104786523061102, -0.28165693242790885

[43, 0.5251068640739631, 0.25545711655920955, -0.4104790705966158, -0.2816575378368934

[44, 0.5251074325174725, 0.2554574613168408, -0.4104793857868888, -0.2816579940341588,

[45, 0.5251078608548293, 0.255457721104288, -0.4104796232873669, -0.2816583377800866, 4

[46, 0.525108183611137, 0.25545791685470737, -0.4104798022484174, -0.2816585968021261,

[47, 0.5251084268152715, 0.25545806435813706, -0.4104799370983646, -0.281658791977543,

[48, 0.5251086100726915, 0.2554581755032365, -0.4104800387102189, -0.2816589390467072,

[49, 0.5251087481608395, 0.25545825925365306, -0.4104801152763964, -0.2816590498651434

[50, 0.5251088522122708, 0.25545832236061267, -0.41048017297035166, -0.2816591333690937

[51, 0.5251089306169734, 0.2554583699130022, -0.4104802164436999, -0.28165919629051855

Solucion: [0.5251089306169734, 0.2554583699130022, -0.4104802164436999, -0.281659196290

Gauss-Seidel, SOR

Algoritmo

```
import numpy as np
from tabulate import
tabulate from sympy
import * import
matplotlib.pyplot as plt
def solNumpy(array):
        A =
        array[:,0:-
        1] B =
        array[:,-1]
        solution = np.linalg.solve(A, B)
        return solution

def checkSquare(array):
        rows = len(array)
        for i in range
        (0,rows):
                if len(array[i]) != rows:
                        raise Exception('La matriz debe ser cuadrada')
        return rows

def checkDet(array):
        det = np.linalg.det(array)
        print('Determinante: ' + str(det))
        if det == 0: raise Exception('La determinante de la matriz debe ser
        diferente a tol = 10e-4 if abs(det) < tol:
                option = getOption('La determinante es menor a ' + str(tol)
```

```
                                + ' y puede presentar problemas de evaluaci n       Desea
continuar?
                    if option == 'n': raise Exception('Operaci n abortada por el usuario')


def gauss_seidelMatricial(matriz, vector, x0, tol, nMax):
        dimension = checkSquare(matriz)
        checkDet(matriz)
        array =
        np.zeros((dimension,dimension + 1))
        array[:,:-1] = matriz array[:,-1:] =
        vector print(tabulate(array))
        numpySol = solNumpy(array)
        D = np.triu(np.tril(matriz))
        L = -np.tril(matriz,-1)
        U = -np.triu(matriz,+1)
        Tj = np.dot(np.linalg.inv(D-L),U)
        print('---Tj---')
        print(tabulate(Tj, floatfmt='.8f')) radioS
        = np.amax(abs(np.linalg.eigvals(Tj)))
        print('Radio espectral: ' + str(radioS))
        Cg = np.dot(np.linalg.inv(D-L),vector) print('---Cg-
        --')
        print(tabulate(Cg, floatfmt='.8f'))

        table = [[0,x0]]
        for n in range(1,nMax):
                table.append([n])
                table[n].append(np.dot(Tj,table[n-1][1])+Cg)
                errorAbs = abs(np.linalg.norm(table[n][1])-np.linalg.norm(table[n-
                1][1] table[n].append(errorAbs) if errorAbs < tol: break
```

```
        print(tabulate(table, headers=['i','b','E'], floatfmt=['i','.8f','.1E'])) print('La

        soluci n es: ' + str(numpySol))

def SORMatricial(matriz, vector, x0, tol, nMax, w):
        dimension = checkSquare(matriz)

        checkDet(matriz)

        array =
        np.zeros((dimension,dimension + 1))
        array[:,:-1] = matriz array[:,-1:] =
        vector print(tabulate(array))
        numpySol = solNumpy(array)

        D = np.triu(np.tril(matriz))

        L = -np.tril(matriz,-1)

        U = -np.triu(matriz,+1)

        Tw = np.dot(np.linalg.inv(D - w*L),((1-w)*D + w*U)) print('---Tw-
        --')

        print(tabulate(Tw, floatfmt='.8f'))
        radioS =
        np.amax(abs(np.linalg.eigvals(Tw)))
        print('Radio espectral: ' + str(radioS))

        Cw = w * np.dot((np.linalg.inv(D - w*L)),vector) print('---Cw--
        -')

        print(tabulate(Cw, floatfmt='.8f'))

        table = [[0,x0]]


        for n in range(1,nMax):
                table.append([n])

                table[n].append(np.dot(Tw,table[n-1][1])+Cw)

                errorAbs = abs(np.linalg.norm(table[n][1])-np.linalg.norm(table[n-
                1][1] table[n].append(errorAbs) if errorAbs < tol: break
```

print(tabulate(table, headers=['i','b','E'], floatfmt=['i','.8f','.1E'])) print('La

soluci n es: ' + str(numpySol))

A = np.array([[4,-1,0,3],[1,15.5,3,8],[0,-1.3,-4,1.1], [14,5,-2,30]],dtype=np.float64) x = np.array([[1],[1],[1],[1]],dtype=np.float64)

x0 = np.array([[0],[0],[0],[0]],dtype=np.float64) tol = 1e-7 nMax = 100

w = 1.5

gauss_seidelMatricial(A,x,x0,tol,nMax)

SORMatricial(A,x,x0,tol,nMax,1.5)

1. Gauss-Seidel

Como ejecutar

A = np.array([[4,-1,0,3],[1,15.5,3,8],[0,-1.3,-4,1.1], [14,5,-2,30]],dtype=np.floa x = np.array([[1],[1],[1],[1]],dtype=np.float64)

x0 = np.array([[0],[0],[0],[0]],dtype=np.float64) tol = 1e-7 nMax = 100

gauss_seidelMatricial(A,x,x0,tol,nMax)

Resultado

Determinante: -3297.5999999999976

4	-1	0	3	1
1	15.5	3	8	1
0	-1.3	-4	1.1	1
14	5	-2	30	1

---Tj---

0.00000000 0.25000000 0.00000000 -0.75000000

0.00000000 -0.01612903 -0.19354839 -0.46774194

0.00000000 0.00524194 0.06290323
0.42701613 0.00000000 -0.11362903
0.03645161 0.45642473 ---------- ---------
---------- ----------Radio espectral:
0.5994876461601171

---Cg---

0.25000000

0.04838710 -
0.26572581

-0.10911290

----

i b E

--- ---------- --
----0 [[0.]

[0.]

[0.]

[0.]]

1 [[0.25] 3.8E-01
[0.0483871] [-
0.26572581]

[-0.1091129]]

2 [[0.34393145] 1.4E-01
[0.15007414] [-
0.32878014] [-
0.17409904]]

3 [[0.41809282] 9.5E-02
[0.19103484] [-
0.35996356]

[-0.21761336]]

4 [[0.46096873] 5.8E-02

[0.21676315]

[-0.3802917]

[-0.24326538]]

5 [[0.48663982] 3.5E-02
[0.23228118] [-
0.39238936]

[-0.25863807]]

6 [[0.50204885] 2.1E-02
[0.24156283] [-
0.39963339]

[-0.26785883]]

7 [[0.51128483] 1.3E-02
[0.24712813] [-
0.40397782]

[-0.27338613]]

8 [[0.51682163] 7.5E-03
[0.25046457] [-
0.40658217]

[-0.27669967]]

9 [[0.52014089] 4.5E-03
[0.25246471] [-
0.40814344]

[-0.2786861]]

10 [[0.52213075] 2.7E-03
[0.25366376]

[-0.4090794]

[-0.27987694]]

11 [[0.52332364] 1.6E-03
[0.25438258]

[-0.4096405]

[-0.28059083]]

12 [[0.52403877] 9.7E-04

[0.25481351] [-
0.40997687] [-
0.2810188]]

13 [[0.52446748] 5.8E-04
[0.25507184] [-
0.41017852]

[-0.28127536]]

14 [[0.52472448] 3.5E-04
[0.25522671] [-
0.41029941]

[-0.28142917]]

15 [[0.52487856] 2.1E-04
[0.25531955] [-
0.41037188]

[-0.28152138]]

16 [[0.52497092] 1.3E-04
[0.25537521] [-
0.41041532]

[-0.28157665]]

17 [[0.52502629] 7.5E-05
[0.25540857] [-
0.41044137]

[-0.28160979]]

18 [[0.52505948] 4.5E-05
[0.25542858] [-
0.41045698]

[-0.28162965]]

19 [[0.52507938] 2.7E-05
[0.25544057] [-
0.41046634]

[-0.28164156]]

20 [[0.52509131] 1.6E-05
[0.25544776] [-
0.41047195]

[-0.2816487]]

21 [[0.52509847] 9.7E-06
[0.25545206] [-
0.41047531]

[-0.28165298]]

22 [[0.52510275] 5.8E-06
[0.25545465] [-
0.41047733] [-
0.28165555]]

23 [[0.52510532] 3.5E-06
[0.2554562] [-
0.41047854]

[-0.28165709]]

24 [[0.52510686] 2.1E-06
[0.25545713] [-
0.41047926]

[-0.28165801]]

25 [[0.52510779] 1.3E-06
[0.25545768]

[-0.4104797]

[-0.28165856]]

26 [[0.52510834] 7.5E-07
[0.25545802] [-
0.41047996]

[-0.28165889]]

27 [[0.52510867] 4.5E-07
[0.25545822] [-
0.41048012]

[-0.28165909]]

28 [[0.52510887] 2.7E-07
[0.25545834] [-
0.41048021]

[-0.28165921]]

29 [[0.52510899] 1.6E-07
[0.25545841] [-
0.41048027]

[-0.28165928]]

30 [[0.52510906] 9.7E-08
[0.25545845]

[-0.4104803]

[-0.28165932]]

La soluci n es: [0.52510917 0.25545852 -0.41048035 -0.28165939]

2. SOR

Como ejecutar

A = np.array([[4,-1,0,3],[1,15.5,3,8],[0,-1.3,-4,1.1], [14,5,-2,30]],dtype=np.floa

x = np.array([[1],[1],[1],[1]],dtype=np.float64)

x0 = np.array([[0],[0],[0],[0]],dtype=np.float64)

tol =
1e-7
nMax
= 100

w = 1.5

gauss_seidelMatricial(A,x,x0,tol,nMax)

SORMatricial(A,x,x0,tol,nMax,w)

Resultado

Determinante: -3297.5999999999976

-- ---- -- ---- -

4 -1 0 3 1

1 15.5 3 8 1

0 -1.3 -4 1.1 1

14 5 -2 30 1
-- ---- -- ---- -

---Tw---

---------- ---------- ---------- ----------
0.50000000 0.37500000 0.00000000 -
1.12500000 0.04838710 -0.53629032 -

0.29032258 -0.66532258 -0.02358871
0.26144153 -0.35846774 0.73684476

0.33554435 -0.10228327 0.03673387
0.52751512 ---------- ---------- ----------
----------Radio espectral: 0.6312081938144991

---Cw---

0.37500000
0.06048387 -
0.40448589

-0.26806956

---

i b E

--- ---------- --
----0 [[0.]

[0.]

[0.]

[0.]]

1 [[0.375] 6.2E-01
[0.06048387] [-
0.40448589]

[-0.26806956]]

2 [[0.5117597] 2.0E-01
[0.34197623] [-
0.45004916]

[-0.30469599]]

3 [[0.59014422] 1.4E-02
[0.23522845] [-
0.39033638]

[-0.30859371]]

4 [[0.51530648] 2.2E-02
[0.28152633]

[-0.4443708]

[-0.27123634]]

5 [[0.52806002] 3.3E-02
[0.24390875] [-
0.38360511]

[-0.28336154]]

6 [[0.52121751] 1.9E-02
[0.25512531] [-
0.42445767]

[-0.27939858]]

7 [[0.52438664] 7.0E-03
[0.25800267] [-
0.40379938]

[-0.28225196]]

8 [[0.52709114] 4.7E-03
[0.25251377] [-
0.41262971]

[-0.28222923]]

9 [[0.52365497] 1.8E-03
[0.25813679] [-
0.41094639]

[-0.2810727]]

10 [[0.5261806] 3.3E-04 [0.25369679] [-0.40914648]
[-0.28212891]]

11 [[0.52444102] 8.2E-04 [0.25638029] [-0.41179033]
[-0.28131836]]

12 [[0.52540526] 8.0E-04
[0.25508527] [-
0.40950273]

[-0.28184609]]

13 [[0.5250312] 6.3E-04
[0.2555134] [-
0.41107293]

[-0.28158444]]

14 [[0.52508442] 3.9E-04

[0.25554748] [-
0.41019651]

[-0.2816734]]

15 [[0.52517067] 1.9E-04
[0.25533652] [-
0.41056857]

[-0.28167376]]

16 [[0.52504884] 6.2E-05
[0.25556209] [-
0.41049266]

[-0.2816371]]

17 [[0.5251531] 4.0E-06 [0.25538879] [-0.41043101]
[-0.28167892]]

18 [[0.52508303] 3.0E-05
[0.2554967] [-
0.41053169]

[-0.28164601]]

19 [[0.52512151] 3.2E-05
[0.25544277] [-
0.41044149]

[-0.28166689]]

20 [[0.52510554] 2.5E-05
[0.25546126] [-
0.41050422]

[-0.28165617]]

21 [[0.52510839] 1.6E-05 [0.25546165] [-0.41046862]
[-0.28166007]]

22 [[0.5251115] 7.8E-06
[0.25545384] [-
0.41048422]

[-0.2816599]]

23 [[0.52510682] 2.7E-06
[0.2554626] [-
0.41048062]

[-0.28165854]]

24 [[0.52511092] 2.1E-08
[0.25545573] [-
0.41047851]

[-0.28166015]]

La soluci n es: [0.52510917 0.25545852 -0.41048035 -0.28165939]

Vandermonde

Algoritmo

```
import numpy as np
import tabulate as tb
from tabulate import
tabulate from sympy
import *
tb.PRESERVE_WHIT
ESPACE = True

x = symbols('x')
init_printing(use_unicode=False)

def vandermonde(puntos):

    print('---puntos-
    --')
    print(puntos)
    grado =
    len(puntos) - 1
    matriz = [] for i
    in puntos:

        row = [] for j in
        range(grado,-1,-1):

            row.append(i[0]**j)

        matriz.append(row)

    matriz =
    np.array(matriz,dtype=np.float64) print('-
    --matriz A---') print(tabulate(matriz))
```

```
        vector =
        [] for i in
        puntos:
                vector.append([i[1]])
        vector =
        np.array(vector,dtype=np.float64)
        print('---vector x---')
        print(tabulate(vector)) sol =
        gauss(matriz, vector) print('---
        Coeficientes---') print(sol) expr =
        sympify('0') aux = grado for j in sol:
                expr = expr + sympify(str(j) + '*x**' + str(grado))
                grado += -1
        print('---Polinomio---')
        print(pretty(expr))


def checkSquare(array):
        rows = len(array)
        for i in range
        (0,rows):
                if len(array[i]) != rows:
                        raise Exception('La matriz debe ser cuadrada')
        return rows

def checkDet(array):
        det = np.linalg.det(array)
        #print('Determinante: ' + str(det))
        if det == 0: raise Exception('La determinante de la matriz debe ser
        diferente a tol = 10e-4 if abs(det) < tol:
                option = getOption('La determinante es menor a ' + str(tol)
                        + ' y puede presentar problemas de evaluaci n       Desea
continuar?
                if option == 'n': raise Exception('Operaci n abortada por el
                usuario')

def pivoteoTotal(array, col, dimension, orden):
```

```
        subArray = array[col:,col:-1]

        dimSubArray = len(subArray) index
        = np.argmax(abs(array[col:,col:-1]))
        rowCol =
        getRowCol(index,dimSubArray,col)
        interCol(array,col,rowCol[1])
        inter1D(orden,col,rowCol[1])
        interRow(array,col,rowCol[0]) return

def getRowCol(index, dimSub, corner):

        row = int(index / dimSub)
        col = index % dimSub
        return [row + corner,col +
        corner]

def interCol(array, col1, col2):

        if col1 != col2:

                aux = array[:,col1].copy()
                array[:,col1] = array[:,col2]
                array[:,col2] = aux

def interRow(array, row1, row2):

        if row1 != row2:

                aux = array[row1].copy()
                array[row1] = array[row2]
                array[row2] = aux

def inter1D(array, pos1, pos2):

        aux = array[pos1]
        array[pos1] =
        array[pos2]
        array[pos2] = aux


def sumMultRow(array, rowAct, rowMult, mult, init):

        array[rowAct,init:] = array[rowAct,init:] + mult * array[rowMult,init:]

def sustitution(array):

        #print('---Sustituci n regresiva---')
        dimension = len(array) solution =
        [] #np.dot(A,B) for row in
        range(dimension - 1, -1, -1):
```

```
        variable = (array[row,dimension] - np.dot(array[row,row + 1:dimension],

            #print('X' + str(row) + ' = ' + str(variable))
            solution.insert(0,variable)

#solNumpy(array)
return
np.array(solution)
def gauss(matriz,
vector):

dimension = checkSquare(matriz)
checkDet(matriz)

array =
np.zeros((dimension,dimension + 1))
array[:,:-1] = matriz array[:,-1:] =
vector

orden = np.arange(dimension)

cont = 0

#print('---Etapa 0, matriz original---
')
#print(tabulate(array,floatfmt='.6f'))
for col in range(0,dimension-1):

            pivoteoTotal(array, col, dimension,
            orden) for row in
            range(col+1,dimension):

                    mult = -(array[row,col]/array[col,col])

                sumMultRow(array,row,col,mult,col)

            cont += 1

            #print('---Etapa ' + str(cont) + ', haciendo 0 la columna '

            # + str(col) + '---')

            #print(tabulate(array,floatfmt='.
6f')) sol = sustitution(array) for i in
range(0,len(orden)):

            inter1D(sol,i,orden[i])
            inter1D(orden,i,orden[i])

np.set_printoptions(precision=6)
```

```
        #print('DespuØs de aplicar sustituci n regresiva: ' + str(sol))
        return sol
```

Como ejecutar

```
puntos = [[-1,15.5],[0,3],[3,8],[4,1]]

vandermonde(puntos)
```

Resultado

```
---puntos---
[[-1, 15.5], [0, 3], [3, 8], [4, 1]]
---matriz A---
--      -
-       -
-       -
1

1       -
1

1

 0

0

0

1 27

9

3

1

64

16
```

4

1 -- -
- -
- -

---vector
x------

1
 5
 .
 5
 3

 8

 1

---Coeficientes---

[-1.141667 5.825 -5.533333 3.]

---Polinomio---

                    3   2

- 1.1416666666666666*x+ 5.824999999999999*x- 5.533333333333332*x + 3.0

Interpolación

Newton

Algorit

mo def

calcularB(x,

y):

diferenciasGradoAnterior = y # Diferencias divididas de orden 0.

b = []

b.append(diferenciasGradoAnterior[0])

```
    # En este ciclo se buscan los
    bi. for orden in range(1,
    len(x)): diferencias = []

        i = 0
        j = i
        +
        orde
        n

        for diferencia in range(len(diferenciasGradoAnterior) - 1):

            numerador = diferenciasGradoAnterior[i] - diferenciasGradoAnterior[i
            + 1] denominador = x[i] - x[j]
            diferencias.append(numerador/denominador)

            i
            +
            =
            1
            j
            +
            =
            1

        b.append(diferencias[0])

diferenciasGradoAnterior = diferencias return b

def p(xEval, b, x): sumatoria = 0 for i in

range(len(b)): productorio = 1 for j in range(i):

productorio *= xEval - x[j] sumatoria += b[i] *

productorio return sumatoria


if __name__==("__main__"):
    x = [-1,0,3,4] y =
    [15.5,3,8,1]
    resultado =
```

```
    calcularB(x,y)
    print(resultado)
```

Como ejecutarlo

```
x = [-
1,0,3,4]
y =
[15.5,3,
8,1]

resultado = calcularB(x,y)
print(resultado)
```

Resultado

```
[15.5, -12.5, 3.5416666666666665, -1.1416666666666666]
```

Lagrange

Algortimo

```
import numpy as np
import tabulate as tb
from tabulate import
tabulate from sympy
import *
tb.PRESERVE_WHIT
ESPACE = True

x = symbols('x')
init_printing(use_unicode=False)

def lagrange(puntos):
        print('---
        puntos---')
        print(puntos)
        print() tabla =
        []

        sum = sympify(0) print('---Lk(x)
        (Opcional)---') for k in range(0,
        len(puntos)): numerador =
```

```
            sympify(1) denominador =
            sympify(1) for n in range(0,
            len(puntos)):
                        if n != k:
                                    numerador *= sympify('x - ' + str(puntos[n][0]))
                                    denominador *= sympify(str(puntos[k][0]) + ' - ' +
                                    str(
                    lk = numerador / denominador
                    print('L' + str(k) + '(x) = ' + str(lk))

                    sum += sympify(lk * puntos[k][1])
            print()
            print('---Polinomio
            expandido---')
            print(pretty(sum)) print()
            poli = simplify(sum)
            print('---Polinomio
            simplificado---')
            print(pretty(poli)) print()


puntos = [[-1,15.5],[0,3],[3,8],[4,1]]

lagrange(puntos)
```

Como ejecutarlo

```
puntos = [[-1,15.5],[0,3],[3,8],[4,1]]
lagrange(puntos)
```

Resultado

```
[[-1, 15.5], [0, 3], [3, 8], [4, 1]]

---Lk(x) (Opcional)---

L0(x) = -x*(x - 4)*(x - 3)/20

L1(x) = (x - 4)*(x - 3)*(x + 1)/12

L2(x) = -x*(x - 4)*(x + 1)/12
L3(x) = x*(x - 3)*(x + 1)/20
```

---Polinomio expandido---

$$-0.775*x*(x - 4)*(x - 3) - \frac{2*x*(x - 4)*(x + 1)}{3} + \frac{x*(x - 3)*(x + 1)}{20} + \frac{(x - 4)*(x - 3)*(x}{4}$$

---Polinomio simplificado---

$$-1.14166666666667*x^3 + 5.825*x^2 - 5.53333333333333*x + 3.0$$

Trazadores lineales

Algoritmo

```
import numpy as np

#Este metodo recive 2 listas, los X y los Y de los puntos

#Retornara una lista con las pendientes de las rectas de 1 punto a otro y una lista de def spline_lineal(X,Y):

    pendientes = []
    ecuaciones = []

    for i in range(len(X)-1):

        m = ( Y[i+1] - Y[i] ) / ( X[i+1] - X[i] )

        pendientes.append(m)

        corte = (m*(-X[i+1])) +
        Y[i+1] ecuacion =
        str(m)+"X " if corte < 0
        :

            ecuacion = ecuacion + str(corte)

        else:

            ecuacion = ecuacion +"+"+str(corte)

        ecuacion = ecuacion +"  "+str(Y[i]) + " <= X <= " + str(Y[i+1])

    ecuaciones.append( ecuacion ) return pendientes, ecuaciones
```

```
#Recive las listas de X y de Y, el valor de X a evaluar y la lista de las pendientes

#Retorna el valor de F de X a
evaluar def
evaluar_recta(X,Y,X_eval,M): j
= 0 for i in range(len(X)-1):

        a = X[i] b = X[i+1] if X[i] <=
        X_eval and X_eval <= X[i+1]:

            G = ( M[j] ) * ( X_eval-X[j] ) + ( Y[j] )
j = j+1 return G def mainLineal(xs, ys,
valor_evaluar):

    M, ecuaciones = spline_lineal(xs,ys)
    evaluacion =
    evaluar_recta(xs,ys,valor_evaluar,M)

    #imprimir
    ecuaciones for
    x in
    ecuaciones:
    print(x)

    #imprimir el valor de F de X a evaluar

    print("> Al evaluar",valor_evaluar, "se obtiene:", evaluacion)

    return evaluacion, ecuaciones


if __name__ == "__main__":

    x = [-
    1,0,3,4]
    y =
    [15.5,3,
    8,1]
    valor =
    1

    evaluacion, ecuaciones = mainLineal(x,y,valor)
    #print(ecuaciones)
```

Como ejecutarlo x = [-1,0,3,4]
y = [15.5,3,8,1]
valor = 1

```
evaluacion, ecuaciones = mainLineal(x,y,valor)

#print(ecuaciones)
```

Resultado

-12.5X +3.0 15.5 <= X <= 3

1.6666666666666667X +3.0 3 <= X <= 8

-7.0X +29.0 8 <= X <= 1

> Al evaluar 1 se obtiene: 4.666666666666667

Trazadores cuadráticos

Algoritmo

```
import numpy as np

#Este metodo recive 2 listas, los X y los Y de los puntos

#Retornara una lista con los Xs resultantes de la matriz generada y una lista de string def spline_cuadratico(X,Y):

    k = 0

    ecu1 = []
    ecu1y = []

    for i in range(2,len(X)):

        ecu11 = [] a = (X[i-1])**2

        b = (X[i-1])
        c = 1
```

```
    d  = Y[i-1]p = (3* (len(X)-1))-3

    for i in range(p):
        ecu11.append(0)

    if a ==
        ((X[1])**2
        ): a = 0

    ecu11.insert(k,c)
    ecu11.insert(k,b)
    ecu11.insert(k,a)
    ecu1.append(ecu11)

    ecu1y.append(d)

    k = k+3

k = 3

ecu2
= []
ecu2y
= []

for i in
    range(2,len(X
    )): ecu22 = []
    a = (X[i-1])**2
    b = (X[i-1])

    c  = 1
    d  = Y[i-1]p = (3* (len(X)-1))-3 for i in
       range(p):
          ecu22.append(0)
    ecu22.insert(k,c)
    ecu22.insert(k,b)
    ecu22.insert(k,a)
    ecu2.append(ecu22)
    ecu2y.append(d)

    k = k+3

ecu3 =
[0,X[0],1] p =
(3* (len(X)-1)) -
```

```
3 for i in
range(p):
    ecu3.append(0)
ecu3y=Y[0]

a = (X[len(X)-
1])**2 ecu4 = [a,
X[len(X)-1],1] p =
(3*(len(X)-1)) -3
for i in range(p):
    ecu4.insert(0,0)
ecu4y = Y[len(X)-1]

w
=
3
u
=
0
ecu5 = [] ecu5y =
[] for i in range(2,
len(X)): ecu55 = []
p = (3* (len(X)-1)) -
4
    for k in range(p):
        ecu55.append(0)
    a =( X[i-
    1]*2 ) b
    = 1
    ecu55.inser
    t(u,b) if a
    == (X[1])*2:
    a1 = 0
        ecu55.insert(u,a1)
    else:
        ecu55.insert(u,a)
    ecu55.insert(w,-b)
```

```
        ecu55.insert(w,-a)
        ecu5.append(ecu55)

        u
        =
        u
        +
        3
        w
        =
        w
        +
        3

        ecu5y.append(0)

    ecu6 = [1]

    p=(3*(len(X)-
    1))-1 for k in
    range(p):

        ecu6.appen
    d(0) ecu6y = 0

    matriz = []
    matrizy =
    []

    for i in range(len(ecu1)):
        matriz.append(ecu1[i])
        matrizy.append(ecu1y[i])
        matriz.append(ecu2[i])
        matrizy.append(ecu2y[i])

    matriz.append(ecu3)
    matrizy.append(ecu3y)

    matriz.append(ecu4)
    matrizy.append(ecu4y)

                    for i in
      range(len(ecu5)):
 matriz.append(ecu5[i])
matrizy.append(ecu5y[i
                    ])
```

```
matriz.append(ecu6)
matrizy.append(ecu6y)

# imprimir la matriz resultante del sistema de ecuaciones
"""
print(ma
triz)
print("\nEl vector con los valores de y
es:") print(matrizy) """ solucion =
np.linalg.solve(matriz, matrizy)

ecuaciones = [] for x in
range(0,len(solucion),3):
ecuacion = ""
    ecuacion = ecuacion +str(solucion[x])+"(X^2) "

    if solucion[x+1] < 0: ecuacion = ecuacion +
        str(solucion[x+1]) + "X "
    else: ecuacion = ecuacion
        +"+"+str(solucion[x+1]) + "X "

    if solucion[x+2] < 0:
        ecuacion = ecuacion + str(solucion[x+2])
    else:
        ecuacion = ecuacion

+"+"+str(solucion[x+2])

ecuaciones.append(ecuacion) return

solucion, ecuaciones


#Recive las listas de X de los puntos y la lista de la solucion de Xs de la matriz
resu

#Retorna el valor de F de X a
evaluar def
evaluar_cuadratico(X,X2,X_ev
al): j = 0 for i in range(len(X)-
1):
```

```
a
=
X[
i]
b
=
X[
i+
1]
if
X[
i]
<
=
X
_
e
v
al
a
n
d
X
_
e
v
al
<
=
X[
i+
1]
:
        G = ( X2[j] ) * ( X_eval**2 ) + ( X2[j+1] ) * X_eval + (X2[j+2])
j = j+3 return G

#Recibe un vector de Xs y uno de Ys de los puntos a interpolar, y un valor de X a
evalu #Retorna el valor de F de X a evaluar y una lista de strings con los
polinomios def mainCuadratico(xs, ys, valor_evaluar):

    X2, ecuaciones = spline_cuadratico(xs,ys)
    evaluacion =
    evaluar_cuadratico(xs,X2,valor_evaluar)
```

```
    for x in range(len(ecuaciones)): ecuaciones[x] = ecuaciones[x] +"
                "+str(xs[x]) + " <= X <= " + str(xs[x+1])

            #imprimir ecuaciones
            print(ecuaciones[x])

    #imprimir el valor de F de X a evaluar

    print("> Al evaluar",valor_evaluar, "se obtiene:", evaluacion)

    return evaluacion, ecuaciones


if __name__ == "__main__":

    x = [-
    1,0,3,4]
    y =
    [15.5,3,
    8,1]
    valor =
    1

    evaluacion, ecuaciones = mainCuadratico(x,y,valor)
```

Como ejecutarlo

```
x  = [-1,0,3,4]
y  = [15.5,3,8,1]valor = 1

 evaluacion, ecuaciones = mainCuadratico(x,y,valor)
```

Resultado

0.0(X^2) -12.499999999999972X +3.0000000000000284-1 <= X <= 0

4.7222222222222126(X^2) -12.499999999999972X +3.00 <= X <= 3

-22.83333333333332(X^2) +152.83333333333326X -244.99999999999863 <= X <= 4 > Al evaluar 1 se obtiene: -4.777777777777759

Trazadores cúbicos

Algoritmo

```
import numpy as np
```

```
#Este metodo recive 2 listas, los X y los Y de los puntos

#Retornara una lista con los Xs resultantes de la matriz generada y una lista de string def spline_cubico(X,Y):

    k = 0

    ecu1 = [] ecu1y =
    [] for i in
    range(2,len(X)):

        ecu11
        = [] a =
        (X[i-
        1])**3
        b =
        (X[i-
        1])**2

        c  = (X[i-1])
        d  = 1
        e  = Y[i-1]p = (4* (len(X)-1))-4 for i in range(p): ecu11.append(0)
           ecu11.insert(k,d) ecu11.insert(k,c) ecu11.insert(k,b) ecu11.insert(k,a)
           ecu1.append(ecu11)
        ecu1y.append(e)

        k = k+4

    k = 4

    ecu2
    = []
    ecu2y
    = []
    for i in
    range
    (2,len(
    X)):
    ecu22
    = [] a
    = (X[i-
    1])**3
    b =
    (X[i-
    1])**2
    c =
```

```
(X[i-
1])
    d = 1
    e = Y[i-1]p = (4* (len(X)-1))-4 for i in
      range(p):
        ecu22.append(0)
    ecu22.insert(k,d)
    ecu22.insert(k,c)
    ecu22.insert(k,b)
    ecu22.insert(k,a)
    ecu2.append(ecu22)
    ecu2y.append(e)

    k = k+4

ecu3 = [ X[0]**3, X[0]**2,
X[0], 1 ] p = (4* (len(X)-1)) -
len(ecu3) for i in range(p):

    ecu3.append(0)

ecu3y=Y[0]

a = X[len(X)-1]**3 b =
X[len(X)-1]**2 c =
X[len(X)-1]
ecu4=[a,b,c,1] p =
(4*(len(X)-1)) -
len(ecu4) for i in
range(p):

    ecu4.insert(0,0)

ecu4y = Y[len(X)-1]

w
=
4
u
=
0

ecu5 = [] ecu5y =
[] for i in range(2,
len(X)): ecu55 = []
```

```
p = (4* (len(X)-1)) -
6

      for k in range(p):

            ecu55.append(0)

      a =( X[i-
      1]**2 )*3 b
      = X[i-1]*2

      c = 1

      ecu55.insert(u,c)
      ecu55.insert(u,b)
      ecu55.insert(u,a)
      ecu55.insert(w,-c)
      ecu55.insert(w,-b)
      ecu55.insert(w,-a)
      ecu5.append(ecu55)

      u
      =
      u
      +
      4
      w
      =
      w
      +
      4

      ecu5y.append(0)

w
=
4
u
=
0

ecu6 = [] ecu6y =
[] for i in range(2,
len(X)): ecu66 = []
a = X[i-1]*6

      b = 2
```

```
        p = (4* (len(X)-1))
        -4 for k in
        range(p):

              ecu66.
        append(0)
        ecu66.inser
        t(u,b)
        ecu66.inser
        t(u,a)
        ecu66.inser
        t(w,-b)
        ecu66.inser
        t(w,-a)
        ecu6.appen
        d(ecu66) u
        = u+4 w =
        w+4

        ecu6y.append(0)

a = 6
*
X[0]
b = 2

ecu7 = []

p = ( 4*(len(X)-1) )
-2 for k in
range(p):

      ecu7.app
end(0)
ecu7.insert(0,
b)
ecu7.insert(0,
a) ecu7y = 0

a  = 6*X[len(X)-1]
b  = 2
w = 4 *
(len(X)-2)
ecu8 = [] p =
(4*(len(X)-1))
```

```
-2 for k in
range(p):
      ecu8.app
end(0)
ecu8.insert(w
,b)
ecu8.insert(w
,a) ecu8y = 0

matriz = []
matrizy =
[]

for i in range(len(ecu1)):
      matriz.append(ecu1[i])
      matrizy.append(ecu1y[i])
      matriz.append(ecu2[i])
      matrizy.append(ecu2y[i])

matriz.append(ecu3)
matrizy.append(ecu3y)

matriz.append(ecu4)
matrizy.append(ecu4y)

for i in range(len(ecu5)):
      matriz.append(ecu5[i])
      matrizy.append(ecu5y[i])

for i in range(len(ecu6)):
      matriz.append(ecu6[i])
      matrizy.append(ecu6y[i])
      matriz.append(ecu7)
      matrizy.append(ecu7y)

matriz.append(ecu8)
matrizy.append(ecu8y)

# imprimir la matriz resultante del sistema de ecuaciones
"""

print(matriz)

print("\nEl vector con los valores de y es:")
```

```
print(matrizy) """ solucion =
np.linalg.solve(matriz, matrizy)

ecuaciones = [] for x in
range(0,len(solucion),4):
ecuacion = ""
    ecuacion = ecuacion +str(solucion[x])+"(X^3) "

    if solucion[x+1] < 0: ecuacion = ecuacion +
        str(solucion[x+1]) + "(X^2) "
    else: ecuacion = ecuacion +"+"+str(solucion[x+1]) +
        "(X^2) "

    if solucion[x+2] < 0: ecuacion = ecuacion +
        str(solucion[x+2]) + "X "
    else: ecuacion = ecuacion
        +"+"+str(solucion[x+2]) + "X "

    if solucion[x+3] < 0:
        ecuacion = ecuacion + str(solucion[x+3])
    else: ecuacion = ecuacion
+"+"+str(solucion[x+3])
ecuaciones.append(ecuacion) return
solucion, ecuaciones

#Recive las listas de X de los puntos y la lista de la solucion de Xs de la matriz
resu

#Retorna el valor de F de X a
evaluar def
evaluar_cubico(X,X2,X_eval): j
= 0 for i in range(len(X)-1):
    a = X[i] b = X[i+1] if X[i] <=
    X_eval and X_eval <= X[i+1]:
        G = ( X2[j] ) * (X_eval**3) + (X2[j+1]) * (X_eval**2) + (X2[j+2]) * X_eval
j = j+4 return G
```

```
#Recibe un vector de Xs y uno de Ys de los puntos a interpolar, y un valor de X a
evalu #Retorna el valor de F de X a evaluar y una lista de strings con los
polinomios def mainCubico(xs, ys, valor_evaluar):

    X2, ecuaciones = spline_cubico(xs,ys)
    evaluacion =
    evaluar_cubico(xs,X2,valor_evaluar)

    for x in range(len(ecuaciones)): ecuaciones[x] = ecuaciones[x] +"
            "+str(xs[x]) + " <= X <= " + str(xs[x+1])

        #imprimir ecuaciones
        print(ecuaciones[x])

    #imprimir el valor de F de X a evaluar

    print("> Al evaluar",valor_evaluar, "se obtiene:", evaluacion)

    return evaluacion, ecuaciones


if __name__ == "__main__":
  x    = [-1,0,3,4]y = [15.5,3,8,1] valor = 1
  evaluacion, ecuaciones =
  mainCubico(x,y,valor)


     Como
ejecutarlo x
= [-1,0,3,4]

  y    = [15.5,3,8,1]valor = 1 evaluacion,
  ecuaciones = mainCubico(x,y,valor)

     Resultado

2.5333333333333337(X^3) +7.600000000000001(X^2) -7.433333333333334X
+2.9999999999999982

-1.5222222222222226(X^3) +7.600000000000001(X^2) -7.433333333333333X
+3.0    0 <= X <= 2.0333333333333323(X^3) -24.39999999999999(X^2)
+88.56666666666665X -92.99999999999999 > Al evaluar 1 se obtiene:
1.6444444444444466
```