

Instituto Tecnológico de Oaxaca

Ingeniería en sistemas computacionales

Métodos numéricos

SCC1017 - 4SC

Códigos programados en Python

Alumno:

García San Juan Diego Zahid

Contenido

Introducción.....	3
Sistemas de ecuaciones lineales	4
Método de Gauss	4
Factorización LU	7
Inversa de una matriz.....	10
Determinantes	15
Gauss Seidel	17
Ecuaciones no lineales.....	19
Método de bisección.....	19
Método de falsa posición	21
Método de Newton/Raphson una variable	23
Método de Newton/Raphson varias variables.....	24
Interpolación.....	27
Método de Lagrange	27
Método de Newton.....	29
Ajuste de un polinomio por mínimos cuadrados	30
Interpoladores cúbicos.....	33
Calculo numérico	34
Derivación de datos tabulados.....	35
Derivación de funciones.....	37
Integración de funciones.....	39
Integrador en cuadraturas Gaussianas.	42
Ecuaciones diferenciales	44
Euler	44
Métodos de Runge/Kutta 3o orden	46
Métodos de Runge/Kutta 4o orden	48

Introducción

Los métodos numéricos es una técnica con la cual es posible formular problemas matemáticos usando operaciones aritméticas.

Los métodos numéricos son una herramienta muy poderosa ya que se usan en la formulación de problemas complejos que requieren de un conocimiento básico de ciencias matemáticas e ingeniería adaptando un sinnúmero de cálculos que de una manera lógica nos ayudan a resolver problemas de alta complejidad manejando sistemas de ecuaciones grandes, no lineales, esto lo conseguimos con ayuda del apoyo computacional podemos emplear aplicaciones y desarrollar software que contenga estos métodos numéricos.

Con este proyecto se busca computar en lenguaje Python los diferentes métodos numéricos que nos podemos encontrar a lo largo de nuestra formación profesional y obtener las soluciones aproximadas de dichos problemas planteados, esto se logra en su mayoría haciendo uso de la librería Numpy que Python nos ofrece, esto para poder manipular de una mejor manera las matrices ya que usar los arreglos convencionales sería un poco más difícil ya que constantemente se debe hacer seguimiento de los tipos de objetos que se están manejando y puede que los valores de retorno no sean los esperados.

Sistemas de ecuaciones lineales

Método de Gauss

Código:

```
# Método de Gauss
# Solución a Sistemas de ecuaciones
# De la forma Ax = B

import numpy as np

# INGRESO DE LAS MATRICES
A = np.array([[4, 2, 5],
              [2, 5, 8],
              [5, 4, 3]])

B = np.array([[60.70],
              [92.90],
              [56.30]])

# PROCEDIMIENTO
casicero = 1e-15 # Considerar como 0

# Evitar truncamiento en operaciones
A = np.array(A, dtype=float)

# Matriz aumentada
AB = np.concatenate((A, B), axis=1)
AB0 = np.copy(AB)

# Pivoteo parcial por filas
tamano = np.shape(AB)
n = tamano[0]
m = tamano[1]
```

```

# Para cada fila en AB
for i in range(0, n - 1, 1):
    # columna desde diagonal i en adelante
    columna = abs(AB[i:, i])
    dondemax = np.argmax(columna)

    # dondemax no está en diagonal
    if (dondemax != 0):
        # intercambia filas
        temporal = np.copy(AB[i, :])
        AB[i, :] = AB[dondemax + i, :]
        AB[dondemax + i, :] = temporal
AB1 = np.copy(AB)

# eliminación hacia adelante
for i in range(0, n - 1, 1):
    pivote = AB[i, i]
    adelante = i + 1
    for k in range(adelante, n, 1):
        factor = AB[k, i] / pivote
        AB[k, :] = AB[k, :] - AB[i, :] * factor

# sustitución hacia atrás
ultfila = n - 1
ultcolumna = m - 1
X = np.zeros(n, dtype=float)

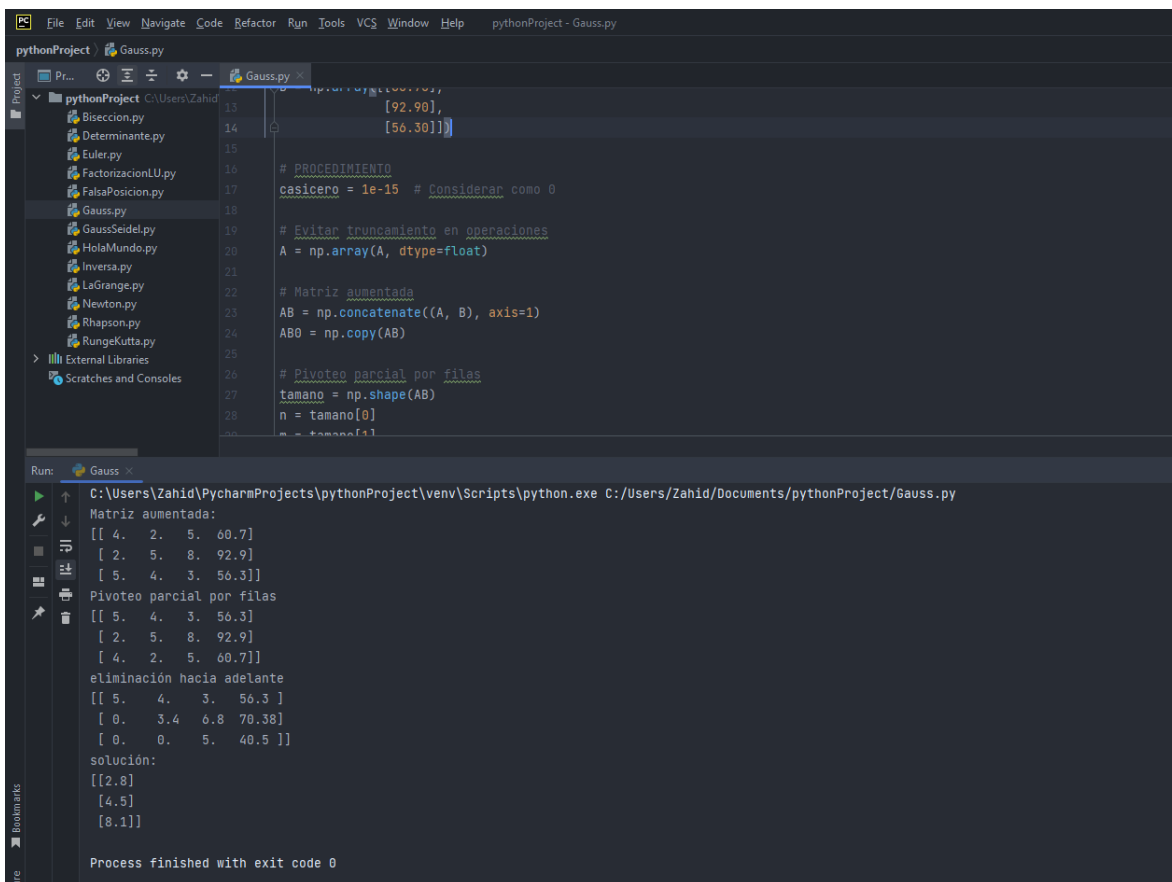
for i in range(ultfila, 0 - 1, -1):
    suma = 0
    for j in range(i + 1, ultcolumna, 1):
        suma = suma + AB[i, j] * X[j]
    b = AB[i, ultcolumna]
    X[i] = (b - suma) / AB[i, i]

X = np.transpose([X])

# SALIDA
print('Matriz aumentada:')
print(AB0)
print('Pivoteo parcial por filas')
print(AB1)
print('eliminación hacia adelante')
print(AB)
print('solución: ')
print(X)

```

Ejecución:



```
pythonProject - Gauss.py
pythonProject CAUsers\Zahid\
  Biseccion.py
  Determinante.py
  Euler.py
  FactorizacionLU.py
  FalsaPosicion.py
  Gauss.py
  GaussSeidel.py
  HolaMundo.py
  Inversa.py
  LaGrange.py
  Newton.py
  Rhapsody.py
  RungeKutta.py
  External Libraries
  Scratches and Consoles

Gauss.py
13     [92.90],
14     [56.30]]
15
16 # PROCEDIMIENTO
17 casicero = 1e-15 # Considerar como 0
18
19 # Evitar truncamiento en operaciones
20 A = np.array(A, dtype=float)
21
22 # Matriz aumentada
23 AB = np.concatenate((A, B), axis=1)
24 AB0 = np.copy(AB)
25
26 # Pivoteo parcial por filas
27 tamano = np.shape(AB)
28 n = tamano[0]
29 m = tamano[1]
```

Run: Gauss

C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/Gauss.py

Matriz aumentada:

```
[[ 4.  2.  5. 60.7]
 [ 2.  5.  8. 92.9]
 [ 5.  4.  3. 56.3]]
```

Pivoteo parcial por filas

```
[[ 5.  4.  3. 56.3]
 [ 2.  5.  8. 92.9]
 [ 4.  2.  5. 60.7]]
```

eliminación hacia adelante

```
[[ 5.  4.  3. 56.3]
 [ 0.  3.4 6.8 70.38]
 [ 0.  0.  5. 40.5 ]]
```

solución:

```
[[2.8]
 [4.5]
 [8.1]]
```

Process finished with exit code 0

Figura 1: Salida del código de Gauss

Factorización LU

Código:

```
import numpy as np

#Matrices a trabajar
a = np.array([[2, 4, 2, 6], [4, 9, 6, 15], [2, 6, 9, 18], [6, 15, 18, 40]])
b = np.array([9, 23, 22, 47])

def LU(a, b):
    m, n = a.shape
    l = np.zeros((n, n))
    u = np.zeros((n, n))
    s1 = 0
    s2 = 0

    for i in range(n):
        l[i][0] = a[i][0]
        u[i][i] = 1

    for j in range(1, n):
        u[0][j] = a[0][j] / l[0][0]

    for k in range(1, n):
        for i in range(k, n):
            for r in range(k): s1 += l[i][r] * u[r][k]
            l[i][k] = a[i][k] - s1
            s1 = 0
        for j in range(k + 1, n):
            for r in range(k): s2 += l[k][r] * u[r][j]
            u[k][j] = (a[k][j] - s2) / l[k][k]
            s2 = 0

    y = np.zeros(n)
    s3 = 0
    y[0] = b[0] / l[0][0]
    for k in range(1, n):
        for r in range(k):
            s3 += l[k][r] * y[r]
        y[k] = (b[k] - s3) / l[k][k]
        s3 = 0
```

```

x = np.zeros(n)
s4 = 0
x[n - 1] = y[n - 1]
for k in range(n - 2, -1, -1):
    for r in range(k + 1, n):
        s4 += u[k][r] * x[r]
    x[k] = y[k] - s4
    s4 = 0

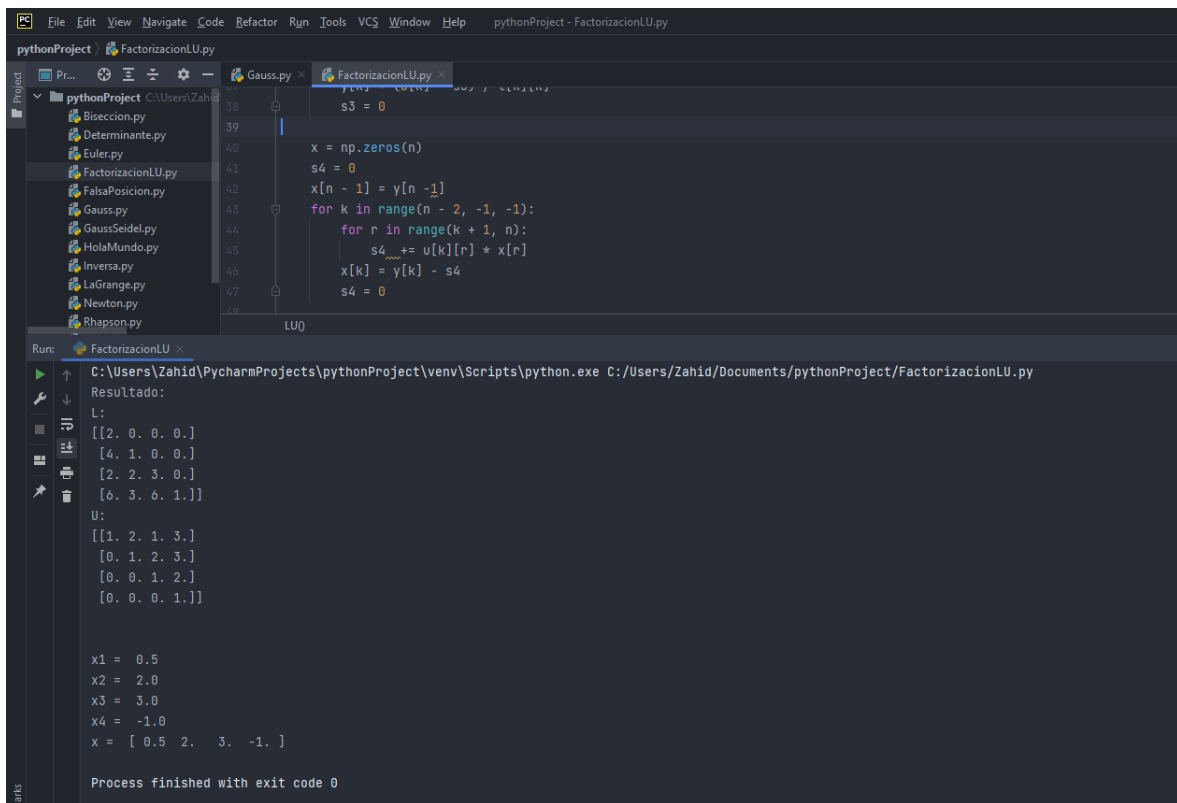
print("Resultado: ")
print("L: ")
print(l)
print("U: ")
print(u)
print("\n")

for i in range(n):
    print("x" + str(i + 1) + " = ", x[i])
print("x" " = ", x)

LU(a, b)

```


Ejecución:



```
pythonProject - FactorizacionLU.py
pythonProject C:\Users\Zahid\Documents\pythonProject
Biseccion.py
Determinante.py
Euler.py
FactorizacionLU.py
FalsaPosicion.py
Gauss.py
GaussSeidel.py
HolaMundo.py
Inversa.py
LaGrange.py
Newton.py
Rhapson.py
FactorizacionLU.py
38 s3 = 0
39
40 x = np.zeros(n)
41 s4 = 0
42 x[n - 1] = y[n - 1]
43 for k in range(n - 2, -1, -1):
44     for r in range(k + 1, n):
45         s4 += u[k][r] * x[r]
46     x[k] = y[k] - s4
47     s4 = 0
48
LUQ

Run: FactorizacionLU
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/FactorizacionLU.py
Resultado:
L:
[[2. 0. 0. 0.]
 [4. 1. 0. 0.]
 [2. 2. 3. 0.]
 [6. 3. 6. 1.]]
U:
[[1. 2. 1. 3.]
 [0. 1. 2. 3.]
 [0. 0. 1. 2.]
 [0. 0. 0. 1.]]

x1 = 0.5
x2 = 2.0
x3 = 3.0
x4 = -1.0
x = [ 0.5  2.   3.  -1.]

Process finished with exit code 0
```

Figura 2: Salida del código de Factorización LU

Inversa de una matriz

Código:

```
# Inversa de una matriz

def imprimir(A, titulo):
    print(titulo)
    for fila in A:
        for value in fila:
            print(f"{value:6.2f} ", end="")
        print()
    print("")

def inversion(A):
    num_filas = len(A)
    num_cols = len(A[0])

    m1 = - 1
    n2 = num_cols * 2

    if num_filas != num_cols:
        print("Error: La matriz no es cuadrada. Por tanto, no es
invertible.")
        return None

    # Construcción de la matriz A | I
    for idx_fila in range(num_filas):
        A[idx_fila] += [1 if idx_fila == j else 0 for j in
range(num_filas)]

    imprimir(A, "Matriz ampliada inicial:")
```

```

# Algoritmo - Triangularización superior

for idx_col in range(num_cols):
    # Búsqueda de pivote
    print(f"Procesando columna {idx_col}")
    l = [(abs(A[idx_fila][idx_col]), idx_fila) for idx_fila in
range(idx_col, num_filas) if A[idx_fila][idx_col] != 0]
    if len(l) == 0:
        print("Error: La matriz no es invertible.")
        return None

    idx_fila = min(l)[1]
    if idx_fila != idx_col:
        print(f"Intercambiar fila {idx_fila} con {idx_col}")
        A[idx_col], A[idx_fila] = A[idx_fila], A[idx_col]
        imprimir(A, "Matriz intercambiada")

    # Triangularización superior
    for idx_fila in [idx for idx in range(idx_col + 1, num_filas)
if A[idx][idx_col] != 0]:
        alpha = -A[idx_fila][idx_col] / A[idx_col][idx_col]
        print(f"Ajuste para fila {idx_fila} es {alpha}")
        for k in range(n2):
            A[idx_fila][k] += A[idx_col][k] * alpha
        imprimir(A, "Matriz ajustada")

    imprimir(A, "Matriz triangulación superior")

# Algoritmo - Triangularización inferior

for idx_col in range(1, num_cols):
    print(f"Procesando columna {idx_col}")
    for idx_fila in range(idx_col):
        alpha = -A[idx_fila][idx_col] / A[idx_col][idx_col]
        print(f"Fila {idx_fila}, factor {alpha}")
        for k in range(idx_col, n2):
            A[idx_fila][k] += A[idx_col][k] * alpha
        imprimir(A, "Ajustada")

    imprimir(A, "Matriz triangulación inferior")

```

```

# Algoritmo - Triangularización inferior

for idx_col in range(1, num_cols):
    print(f"Procesando columna {idx_col}")
    for idx_fila in range(idx_col):
        alpha = -A[idx_fila][idx_col] / A[idx_col][idx_col]
        print(f"Fila {idx_fila}, factor {alpha}")
        for k in range(idx_col, n2):
            A[idx_fila][k] += A[idx_col][k] * alpha
        imprimir(A, "Ajustada")

imprimir(A, "Matriz triangulación inferior")
# Algoritmo - Transformación a la matriz identidad

for idx_fila in range(num_filas):
    alpha = A[idx_fila][idx_fila]
    for idx_col in range(idx_fila, n2):
        A[idx_fila][idx_col] /= alpha

imprimir(A, "Matriz identidad")

inversa = []
for fila in A:
    inversa.append(fila[num_cols:])

return inversa

A = [[5, 8, 9], [2, 6, 6], [3, 1, 1]]
x = inversion(A)
imprimir(x, "Inversa")

```

Ejecución:

```
pythonProject C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/Inversa.py
Matriz ampliada inicial:
5.00  8.00  9.00  1.00  0.00  0.00
2.00  6.00  6.00  0.00  1.00  0.00
3.00  1.00  1.00  0.00  0.00  1.00

Procesando columna 0
Intercambiar fila 1 con 0
Matriz intercambiada
2.00  6.00  6.00  0.00  1.00  0.00
5.00  8.00  9.00  1.00  0.00  0.00
3.00  1.00  1.00  0.00  0.00  1.00

Ajuste para fila 1 es -2.5
Matriz ajustada
2.00  6.00  6.00  0.00  1.00  0.00
0.00 -7.00 -6.00  1.00 -2.50  0.00
3.00  1.00  1.00  0.00  0.00  1.00

Ajuste para fila 2 es -1.5
Matriz ajustada
2.00  6.00  6.00  0.00  1.00  0.00
0.00 -7.00 -6.00  1.00 -2.50  0.00
0.00 -8.00 -8.00  0.00 -1.50  1.00
```

Figura 3: Salida del código de Inversa

```
pythonProject - Inversa.py
Run: Inversa
0.00  0.00  -1.14  -1.14  1.36  1.00

Procesando columna 1
Fila 0, factor 0.8571428571428571
Ajustada
2.00  0.00  0.86  0.86  -1.14  0.00
0.00  -7.00  -6.00  1.00  -2.50  0.00
0.00  0.00  -1.14  -1.14  1.36  1.00

Procesando columna 2
Fila 0, factor 0.7500000000000002
Ajustada
2.00  0.00  0.00  -0.00  -0.12  0.75
0.00  -7.00  -6.00  1.00  -2.50  0.00
0.00  0.00  -1.14  -1.14  1.36  1.00

Fila 1, factor -5.249999999999998
Ajustada
2.00  0.00  0.00  -0.00  -0.12  0.75
0.00  -7.00  0.00  7.00  -9.62  -5.25
0.00  0.00  -1.14  -1.14  1.36  1.00

Matriz triangulación inferior
2.00  0.00  0.00  -0.00  -0.12  0.75
0.00  -7.00  0.00  7.00  -9.62  -5.25
0.00  0.00  -1.14  -1.14  1.36  1.00

Matriz identidad
1.00  0.00  0.00  -0.00  -0.06  0.38
0.00  1.00  -0.00  -1.00  1.37  0.75
0.00  0.00  1.00  1.00  -1.19  -0.87

Inversa
-0.00  -0.06  0.38
-1.00  1.37  0.75
1.00  -1.19  -0.87

Process finished with exit code 0
```

Figura 4: Salida del código de Inversa

Determinantes

Código:

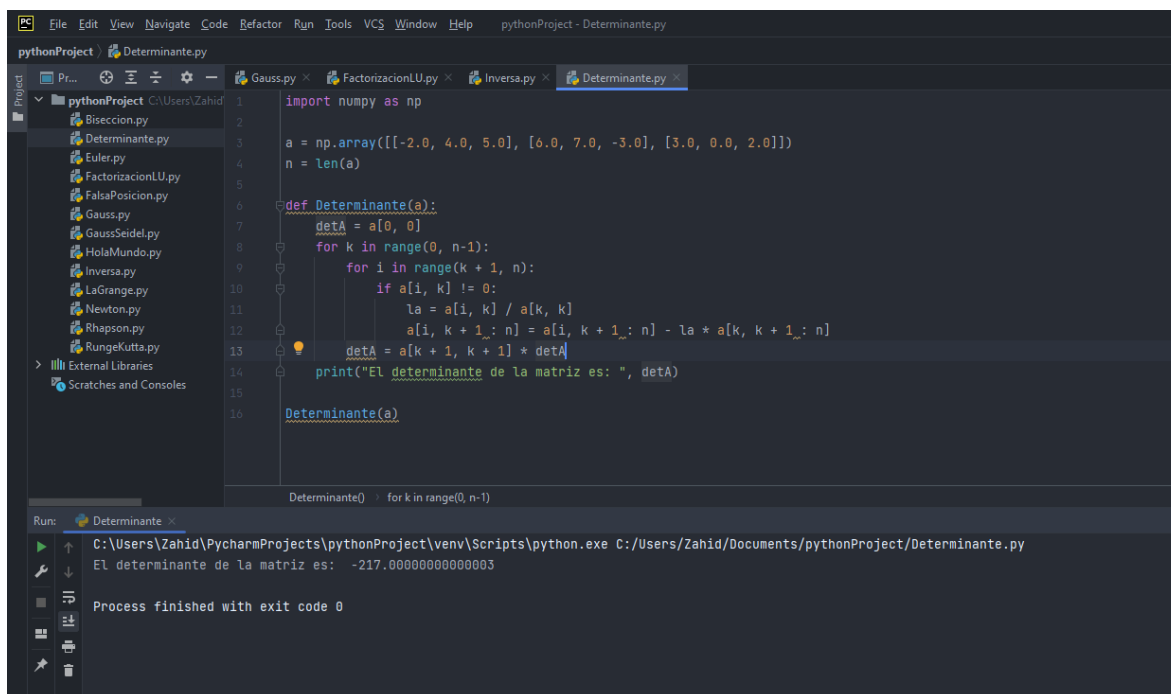
```
import numpy as np

a = np.array([[-2.0, 4.0, 5.0], [6.0, 7.0, -3.0], [3.0, 0.0, 2.0]])
n = len(a)

def Determinante(a):
    detA = a[0, 0]
    for k in range(0, n-1):
        for i in range(k + 1, n):
            if a[i, k] != 0:
                la = a[i, k] / a[k, k]
                a[i, k + 1 : n] = a[i, k + 1 : n] - la * a[k, k + 1 :
n]
                detA = a[k + 1, k + 1] * detA
    print(detA)

Determinante(a)
```

Ejecución:



```
pythonProject - Determinante.py
pythonProject C:\Users\Zahid
Biseccion.py
Determinante.py
Euler.py
FactorizacionLU.py
FalsaPosicion.py
Gauss.py
GaussSeidel.py
HolaMundo.py
Inversa.py
LaGrange.py
Newton.py
Rhapson.py
RungeKutta.py
External Libraries
Scratches and Consoles

1 import numpy as np
2
3 a = np.array([[-2.0, 4.0, 5.0], [6.0, 7.0, -3.0], [3.0, 0.0, 2.0]])
4 n = len(a)
5
6 def Determinante(a):
7     detA = a[0, 0]
8     for k in range(0, n-1):
9         for i in range(k+1, n):
10             if a[i, k] != 0:
11                 la = a[i, k] / a[k, k]
12                 a[i, k+1:n] = a[i, k+1:n] - la * a[k, k+1:n]
13     detA = a[k+1, k+1] * detA
14     print("El determinante de la matriz es: ", detA)
15
16 Determinante(a)

Run: Determinante
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Zahid\Documents\pythonProject\Determinante.py
El determinante de la matriz es: -217.00000000000003
Process finished with exit code 0
```

Figura 5: Salida del código de Determinante

Código:

```
# Método de Gauss-Seidel
# solución de sistemas de ecuaciones
# por métodos iterativos

import numpy as np

# INGRESO
A = np.array([[8, 4, -2],
              [3, 6, -1],
              [2, -2, 6]])

B = np.array([24, 13, 16])

X0 = np.array([0., 0., 0.])

tolera = 0.00001
iteramax = 100

# PROCEDIMIENTO

# Gauss-Seidel
tamano = np.shape(A)
n = tamano[0]
m = tamano[1]
# valores iniciales
X = np.copy(X0)
diferencia = np.ones(n, dtype=float)
errado = 2 * tolera

itera = 0
while not (errado <= tolera or itera > iteramax):
    # por fila
    for i in range(0, n, 1):
        # por columna
        suma = 0
        for j in range(0, m, 1):
            # excepto diagonal de A
            if (i != j):
                suma = suma - A[i, j] * X[j]

        nuevo = (B[i] + suma) / A[i, i]
        diferencia[i] = np.abs(nuevo - X[i])
        X[i] = nuevo
    errado = np.max(diferencia)
    itera = itera + 1
```

```

# Respuesta X en columna
X = np.transpose([X])

# revisa si NO converge
if (itera > iteramax):
    X = 0
# revisa respuesta
verifica = np.dot(A, X)

# SALIDA
print('respuesta X: ')
print(X)
print('verificar A.X=B: ')
print(verifica)

```

Ejecución:

The screenshot shows a Python IDE with a project named 'pythonProject'. The file explorer on the left lists several Python files, including 'GaussSeidel.py'. The main editor window displays the code for 'GaussSeidel.py', which implements the Gauss-Seidel method for solving a system of linear equations. The code includes comments in Spanish, imports numpy as np, and defines matrices A and B. It also sets a tolerance and a maximum number of iterations. The execution output at the bottom shows the results of the calculation.

```

1  # Método de Gauss-Seidel
2  # solución de sistemas de ecuaciones
3  # por métodos iterativos
4
5  import numpy as np
6
7  # INGRESO
8  A = np.array([[8, 4, -2],
9                [3, 6, -1],
10               [2, -2, 6]])
11
12  B = np.array([24, 13, 16])
13
14  X0 = np.array([0., 0., 0.])
15
16  tolera = 0.00001
17  iteramax = 100
18
19  # PROCEDIMIENTO

```

Run: GaussSeidel x

```

C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/GaussSeidel.py
respuesta X:
[[3.00000385]
 [0.99999664]
 [1.9999976 ]]
verificar A.X=B:
[[24.00002221]
 [12.99999381]
 [16.          ]]

Process finished with exit code 0

```

Figura 6: Salida del código de Gauss Siedel

Ecuaciones no lineales

Método de bisección

Código:

```
# Método de Biseccion
import numpy as np

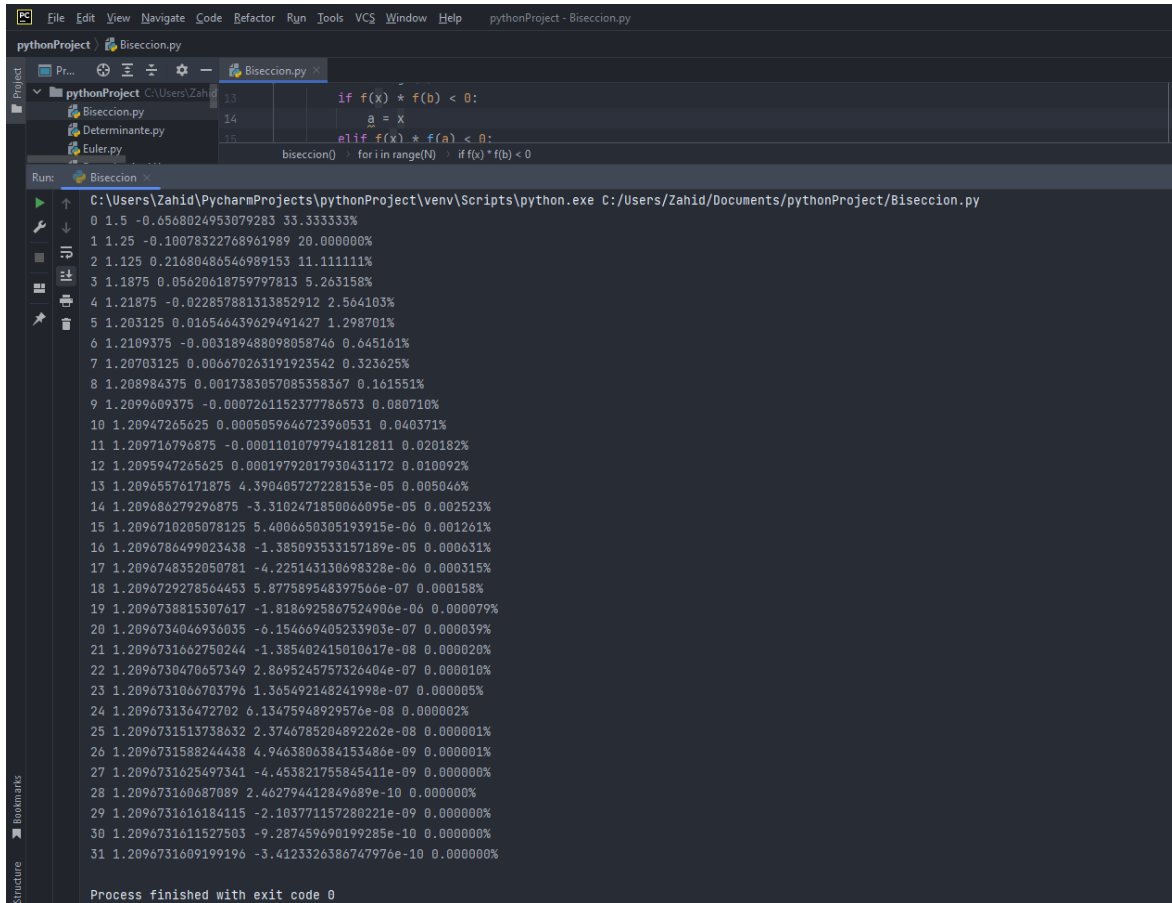
def f(x):
    return np.sin(2 *x + 1) - 3 *x /5 + 1

a = 0
b = 2

def biseccion(f,a, b, N = 100, emax = 1e-10):
    x = (a + b) / 2
    for i in range(N):
        if f(x) * f(b) < 0:
            a = x
        elif f(x) * f(a) < 0:
            b = x
        else:
            break
    xold = x
    x = (a + b) / 2
    e = abs((x - xold)/x)
    if e < emax:
        break
    print(i, x, f(x), '{:%}'.format(e))

biseccion(f, 0, 2)
```

Ejecución:



```
pythonProject Biseccion.py
pythonProject C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/Biseccion.py
0 1.5 -0.6568024953079283 33.333333%
1 1.25 -0.10078322768961989 20.000000%
2 1.125 0.21680486546989153 11.111111%
3 1.1875 0.05620618759797813 5.263158%
4 1.21875 -0.022857881313852912 2.564103%
5 1.203125 0.016546439629491427 1.298701%
6 1.2109375 -0.003189488098058746 0.645161%
7 1.20703125 0.006670263191923542 0.323625%
8 1.208984375 0.0017383057085358367 0.161551%
9 1.2099609375 -0.0007261152377786573 0.080710%
10 1.20947265625 0.0005059646723960531 0.040371%
11 1.209716796875 -0.00011010797941812811 0.020182%
12 1.2095947265625 0.00019792017930431172 0.010092%
13 1.20965576171875 4.390405727228153e-05 0.005046%
14 1.209686279296875 -3.3102471850066095e-05 0.002523%
15 1.2096710205078125 5.4006650305193915e-06 0.001261%
16 1.2096786499023438 -1.385093533157189e-05 0.000631%
17 1.2096748352050781 -4.225143130698328e-06 0.000315%
18 1.2096729278564453 5.877589548397566e-07 0.000158%
19 1.2096738815307617 -1.8186925867524906e-06 0.000079%
20 1.2096734046936035 -6.154669405233903e-07 0.000039%
21 1.2096731662750244 -1.385402415010617e-08 0.000020%
22 1.2096730470657349 2.8695245757326404e-07 0.000010%
23 1.2096731066703796 1.365492148241998e-07 0.000005%
24 1.209673136472702 6.13475948929576e-08 0.000002%
25 1.2096731513738632 2.3746785204892262e-08 0.000001%
26 1.2096731588244438 4.9463806384153486e-09 0.000001%
27 1.2096731625497341 -4.453821755845411e-09 0.000000%
28 1.209673160687089 2.462794412849689e-10 0.000000%
29 1.2096731616184115 -2.103771157280221e-09 0.000000%
30 1.2096731611527503 -9.287459690199285e-10 0.000000%
31 1.2096731609199196 -3.412332638674797e-10 0.000000%

Process finished with exit code 0
```

Figura 7: Salida del código de Método de la bisección

Método de falsa posición

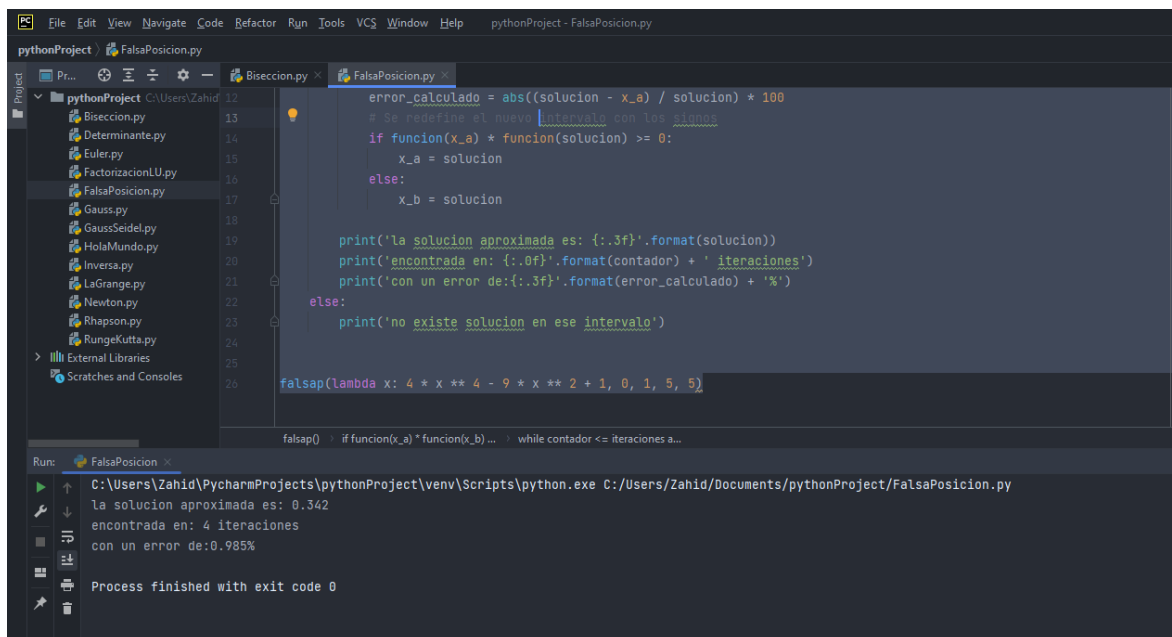
Código:

```
def falsap(funcion, x_a, x_b, iteraciones=10, error_r=0.001):
    # Se inicializan las variables
    solucion = None
    contador = 0
    error_calculado = 101
    # Se evalua si la raiz está dentro del intervalo
    if funcion(x_a) * funcion(x_b) <= 0:
        # Se procede a calcular la funcion
        while contador <= iteraciones and error_calculado >= error_r:
            contador = contador + 1
            solucion = x_b - ((funcion(x_b) * (x_b - x_a)) /
(funcion(x_b) - funcion(x_a)))
            error_calculado = abs((solucion - x_a) / solucion) * 100
            # Se redefine el nuevo intervalo con los signos
            if funcion(x_a) * funcion(solucion) >= 0:
                x_a = solucion
            else:
                x_b = solucion

        print('la solucion aproximada es: {:.3f}'.format(solucion))
        print('encontrada en: {:.0f}'.format(contador) + '
iteraciones')
        print('con un error de:{:.3f}'.format(error_calculado) + '%')
    else:
        print('no existe solucion en ese intervalo')

falsap(lambda x: 4 * x ** 4 - 9 * x ** 2 + 1, 0, 1, 5, 5)
```

Ejecución:



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject - FalsaPosicion.py
pythonProject / FalsaPosicion.py
Project
  pythonProject
    Biseccion.py
    Determinante.py
    Euler.py
    FactorizacionLU.py
    FalsaPosicion.py
    Gauss.py
    GaussSeidel.py
    HolaMundo.py
    Inversa.py
    LaGrange.py
    Newton.py
    Rhapsody.py
    RungeKutta.py
  External Libraries
  Scratches and Consoles

FalsaPosicion.py
12 error_calculado = abs((solucion - x_a) / solucion) * 100
13 # Se redefine el nuevo intervalo con los signos
14 if funcion(x_a) * funcion(solucion) >= 0:
15     x_a = solucion
16 else:
17     x_b = solucion
18
19 print('la solucion aproximada es: {:.3f}'.format(solucion))
20 print('encontrada en: {:.0f}'.format(contador) + ' iteraciones')
21 print('con un error de: {:.3f}'.format(error_calculado) + '%')
22 else:
23     print('no existe solucion en ese intervalo')
24
25 falsap(lambda x: 4 * x ** 4 - 9 * x ** 2 + 1, 0, 1, 5, 5)
26
Run: FalsaPosicion
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Zahid\Documents\pythonProject\FalsaPosicion.py
la solucion aproximada es: 0.342
encontrada en: 4 iteraciones
con un error de:0.985%
Process finished with exit code 0
```

Figura 8: Salida del código de Método de falsa posición

Método de Newton/Raphson una variable

Código:

```
from math import *

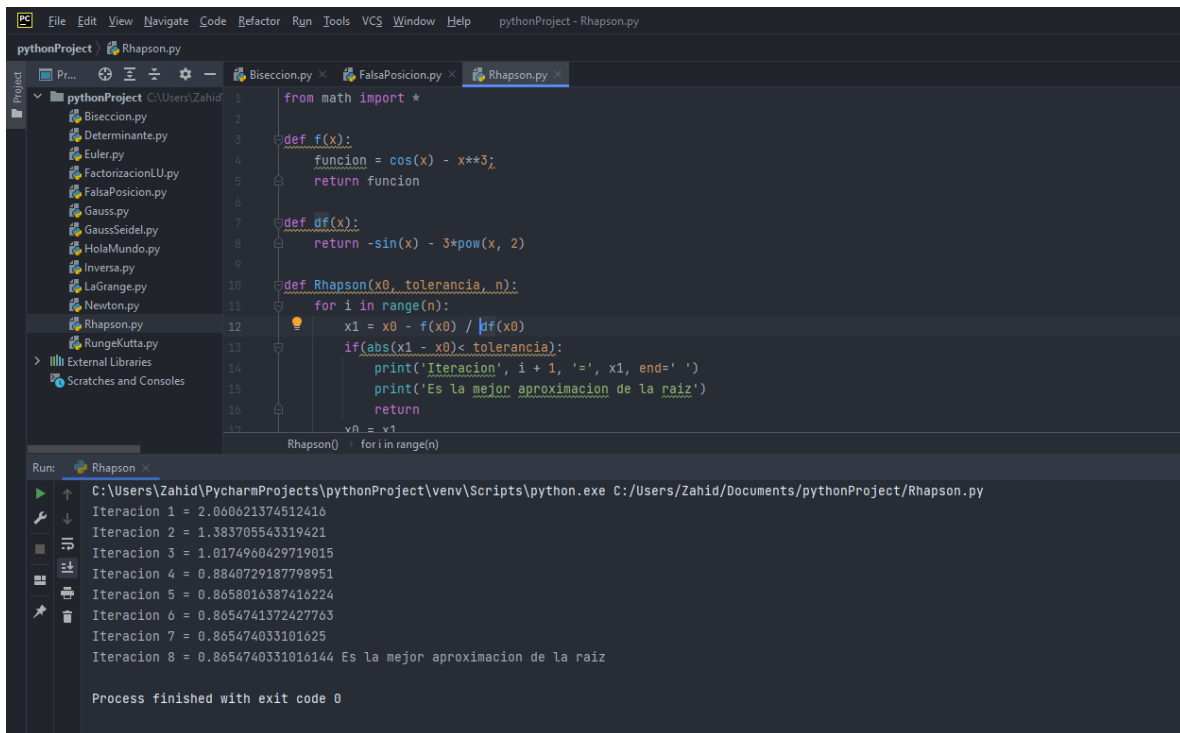
def f(x):
    funcion = cos(x) - x**3;
    return funcion

def df(x):
    return -sin(x) - 3*pow(x, 2)

def Rhapsn(x0, tolerancia, n):
    for i in range(n):
        x1 = x0 - f(x0) / df(x0)
        if(abs(x1 - x0)< tolerancia):
            print('Iteracion', i + 1, '=', x1, end=' ')
            print('Es la mejor aproximacion de la raiz')
            return
        x0 = x1
        print('Iteracion', i + 1, '=', x1)

Rhapsn(pi, 0.0000001, 10)
```

Ejecución:



```
pythonProject - Rhapson.py
pythonProject C:\Users\Zahid\
Biseccion.py
Determinante.py
Euler.py
FactorizacionLU.py
FalsaPosicion.py
Gauss.py
GaussSeidel.py
HolaMundo.py
Inversa.py
LaGrange.py
Newton.py
Rhapson.py
RungeKutta.py
External Libraries
Scratches and Consoles

1 from math import *
2
3 def f(x):
4     funcion = cos(x) - x**3
5     return funcion
6
7 def df(x):
8     return -sin(x) - 3*pow(x, 2)
9
10 def Rhapson(x0, tolerancia, n):
11     for i in range(n):
12         x1 = x0 - f(x0) / df(x0)
13         if abs(x1 - x0) < tolerancia:
14             print('Iteracion', i + 1, '=', x1, end=' ')
15             print('Es la mejor aproximacion de la raiz')
16             return
17         x0 = x1
18     Rhapson() for i in range(n)

Run: Rhapson
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Zahid\Documents\pythonProject\Rhapson.py
Iteracion 1 = 2.060621374512416
Iteracion 2 = 1.383705543319421
Iteracion 3 = 1.0174960429719015
Iteracion 4 = 0.8840729187798951
Iteracion 5 = 0.8658016387416224
Iteracion 6 = 0.8654741372427763
Iteracion 7 = 0.865474033101625
Iteracion 8 = 0.8654740331016144 Es la mejor aproximacion de la raiz

Process finished with exit code 0
```

Figura 9: Salida del código de Método de Newton/Rhapson

Método de Newton/Raphson varias variables

Código:

```
# Ejercicio tomado del libro de Burden, Análisis numérico apartado 10.2, ejercicio
3 inciso c

from math import cos, sin, pi, exp, sqrt
import numpy as np

# introducir valores que se evaluarán de la función
def Fs(x1, x2, x3):
    f1 = 3*x1-cos(x2*x3)-1/2
    f2 = x1**2-81*(x2+0.1)**2+sin(x3)+1.06
    f3 = exp(-x1*x2)+20*x3+(10*pi-3)/3
    return np.matrix([[f1], [f2], [f3]])

#Matriz Jacobiana
def JInv(x1, x2, x3):
    J = np.matrix([[3, x3*sin(x2*x3), x2*sin(x2*x3)], [2*x1, -162*(x2+0.1),
cos(x3)], [-x2*exp(-x1*x2), -x1*exp(-x1*x2), 20]])
    JV = np.linalg.inv(J)
    return [J, JV]

def RhapsomMulti(x1, x2, x3, P0, k, tolerancia):
    print("k \t x1 \t x2 \t x3 \t (x(k)-x(k-1))")
    print("{0:1d} \t {1:1.4f} \t {2:1.4f} \t {3:1.4f} \t".format(k, x1, x2, x3))

    while k < 10:
        # Calcular vector F y matriz Jacobiana
        J, JI = JInv(x1, x2, x3)
        F = Fs(x1, x2, x3)
        Y = -JI * F
        # Vector x
        X = np.matrix(P0).T + Y

        # Actualizando valores
        x1, x2, x3 = float(X[0][0]), float(X[1][0]), float(X[2][0])

        # Calculo de la magnitud del vector
        magnitud = sqrt((x1 - P0[0]) ** 2 + (x2 - P0[1]) ** 2 + (x3 - P0[2]) ** 2)
```

```

# Redefinir P0
P0 = [x1, x2, x3]
k += 1
print("{0:1d} \t {1:1.6f} \t {2:1.6f} \t {3:1.6f} \t {4:1.6f}".format(k,
x1, x2, x3, magnitud))

# Calcular magnitud del vector "Y" y aplicar la tolerancia
if sqrt(Y[0][0] ** 2 + Y[1][0] ** 2 + Y[2][0] ** 2) < Tolerancia:
    print("Cálculo exitoso:")
    break

#Aproximacion lineal
P0 = [0.1, 0.1, -0.1]

# Valores de aproximacion por separado
x1, x2, x3 = P0

# valor de k
k= 0

Tolerancia = 0.0000000001

# Llamada de la funcion con sus respectivos parametros
RhapsonMulti(x1, x2, x3, P0, k, Tolerancia)

```

```

Run: RhapsonMultiVariable
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/RhapsonMultiVariable.py

k  x1      x2      x3      (x(k)-x(k-1))
0  0.1000   0.1000   -0.1000
1  0.499870 0.019467 -0.521520 0.586567
2  0.500014 0.001589 -0.523557 0.017994
3  0.500000 0.000012 -0.523598 0.001577
4  0.500000 0.000000 -0.523599 0.000012
5  0.500000 0.000000 -0.523599 0.000000
6  0.500000 0.000000 -0.523599 0.000000
Cálculo exitoso:)

Process finished with exit code 0

```

Figura 10: Salida del código de Método de Newton/Raphson varias variables

Interpolación

Método de Lagrange

Código:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate

x = np.array([1, 4, 8, 13, 18])
y = np.array([1.1, 1.5, 12.8, 15.3, 15.5])
n = x.size
xi = 3
yi = 0
# Calcula los factores de Lagrange y hace la suma
def lagrange(x, y, xi, yi):
    for i in range(0, n):
        producto = y[i]
        for j in range(0, n):
            if i != j:
                producto = producto * (xi - x[j]) / (x[i] - x[j]);
        yi = yi + producto
    print(yi)
    f = interpolate.lagrange(x, y) # usando la funcion de Lagrange
de ←-scipy
    print(f(xi))
    plt.plot(x, y, 'o')
    plt.plot(xi, yi, 'sr ')
    plt.text(xi + 0.1, yi, ' Profundidad ' + str(yi))
    plt.title('Profundidad del agua ')
    plt.legend(['Datos ', ' Interpolacion '])
    plt.grid(True)
    plt.show()

lagrange(x, y, xi, yi)
```

Ejecución:

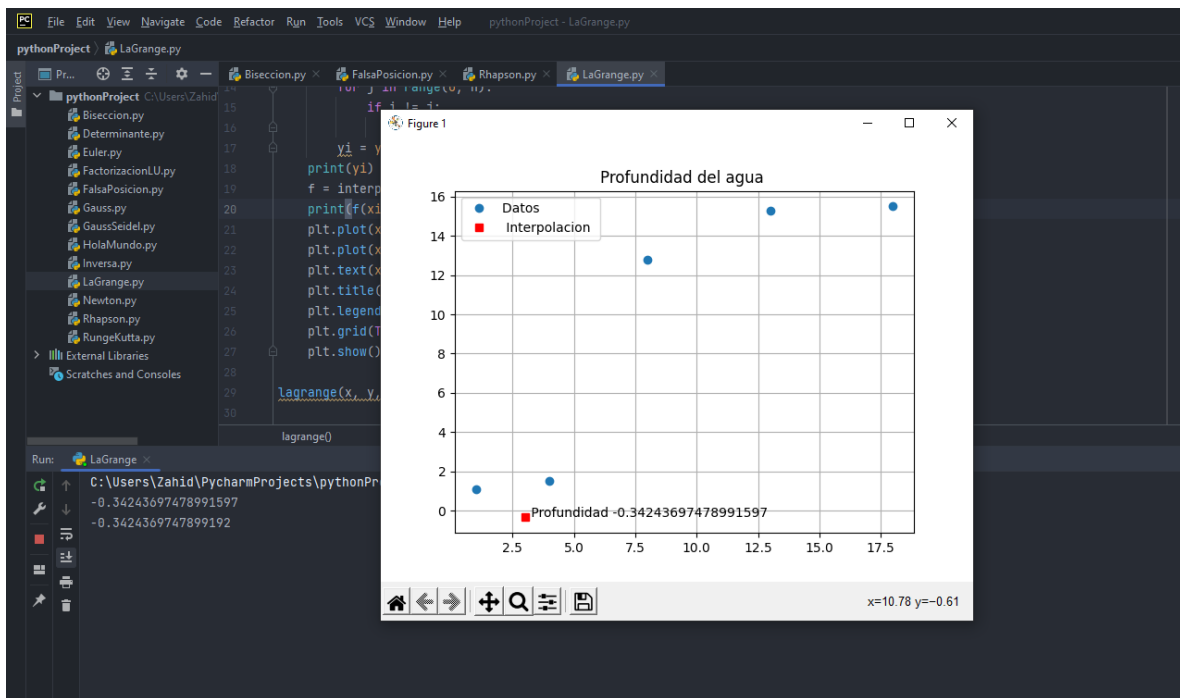


Figura 11: Salida del código de Método de Lagrange

Método de Newton

Código:

```
from pprint import pprint
def Newton(dat):
    # Implementación del interpolador de Newton
    # Entradas:
    # dat -- lista de puntos (x, y) en el plano
    #
    # Salidas:
    # F -- tabla de diferencias divididas
    # P -- función de interpolación

    n = len(dat)
    F = [[0 for x in dat] for x in dat] # crear tabla nula

    for i, p in enumerate(dat): # condiciones iniciales
        F[i][0] = p[1]

    for i in range(1, n): # tabla de diferencias divididas
        for j in range(1, i+1):
            F[i][j] = (F[i][j-1]-F[i-1][j-1])/(dat[i][0]-dat[i-j][0])

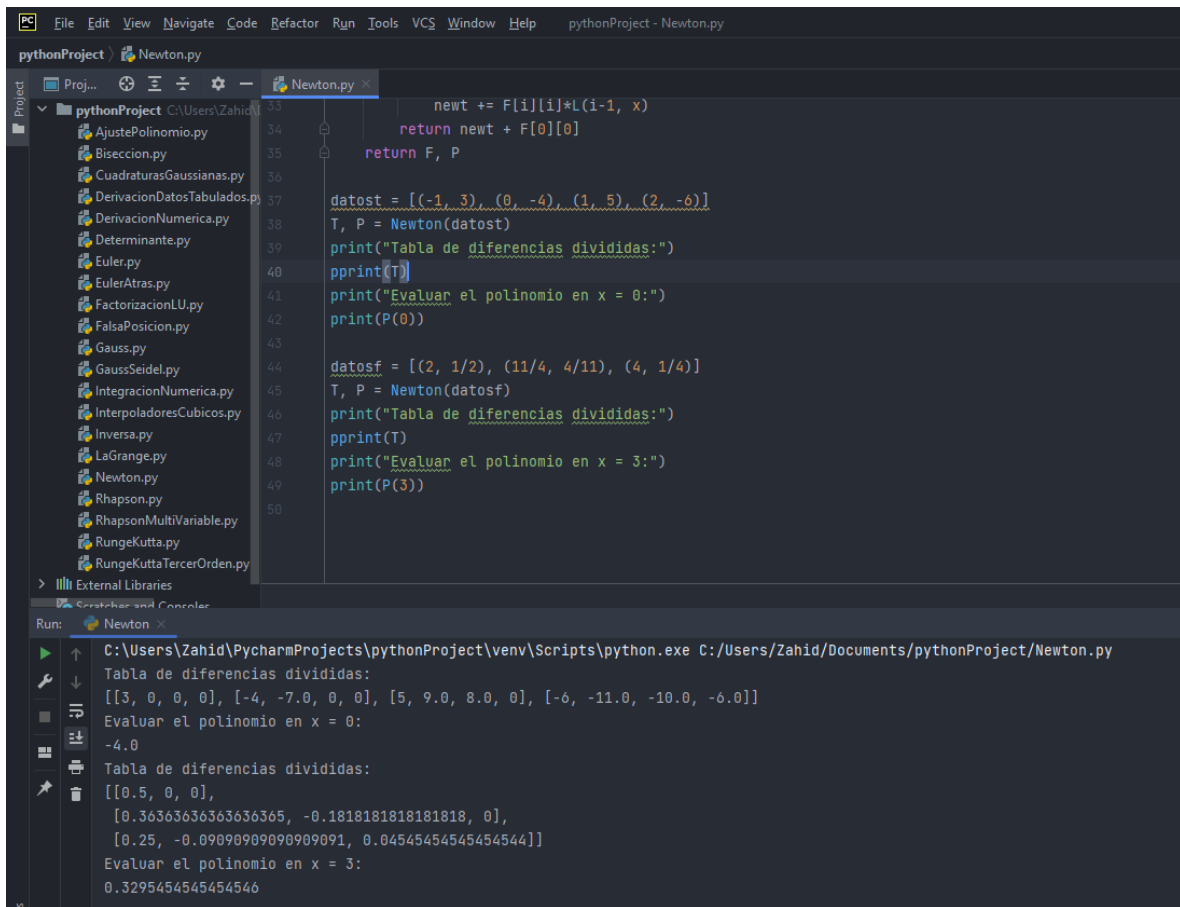
    def L(k, x):
        # Implementación funciones L_k(x)
        out = 1
        for i, p in enumerate(dat):
            if i <= k:
                out *= (x - p[0])
        return out

    def P(x):
        #Implementación polinomio P(x)
        newt = 0
        for i in range(1, n):
            newt += F[i][i]*L(i-1, x)
        return newt + F[0][0]
    return F, P

datost = [(-1, 3), (0, -4), (1, 5), (2, -6)]
T, P = Newton(datost)
print("Tabla de diferencias divididas:")
pprint(T)
print("Evaluar el polinomio en x = 0:")
print(P(0))

datosf = [(2, 1/2), (11/4, 4/11), (4, 1/4)]
T, P = Newton(datosf)
print("Tabla de diferencias divididas:")
pprint(T)
print("Evaluar el polinomio en x = 3:")
print(P(3))11
```

Ejecución:



```
pythonProject - Newton.py
pythonProject C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/Newton.py
Run: Newton X
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/Newton.py
Tabla de diferencias divididas:
[[3, 0, 0, 0], [-4, -7.0, 0, 0], [5, 9.0, 8.0, 0], [-6, -11.0, -10.0, -6.0]]
Evaluar el polinomio en x = 0:
-4.0
Tabla de diferencias divididas:
[[0.5, 0, 0],
 [0.36363636363636365, -0.1818181818181818, 0],
 [0.25, -0.09090909090909091, 0.04545454545454544]]
Evaluar el polinomio en x = 3:
0.3295454545454546
```

Figura 12: Salida del código de Método de Newton

Código:

```
import numpy as np
import matplotlib.pyplot as plt

# Arrays con los valores de "x" y "y"
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])

y = np.array([432, 439, 442, 444, 446, 447, 449, 451, 453, 458, 463,
466, 468, 470, 472, 477, 479, 491, 516, 516, 521, 524, 526, 533, 541,
548, 555, 560, 563, 564])

def AjustePolinomio(x, y):
    n = len(x)

    # Sumatorias y demas operaciones correspondientes
    SumX = sum(x)
    SumY = sum(y)
    SumXX = sum(x**2)
    SumXY = sum(x*y)

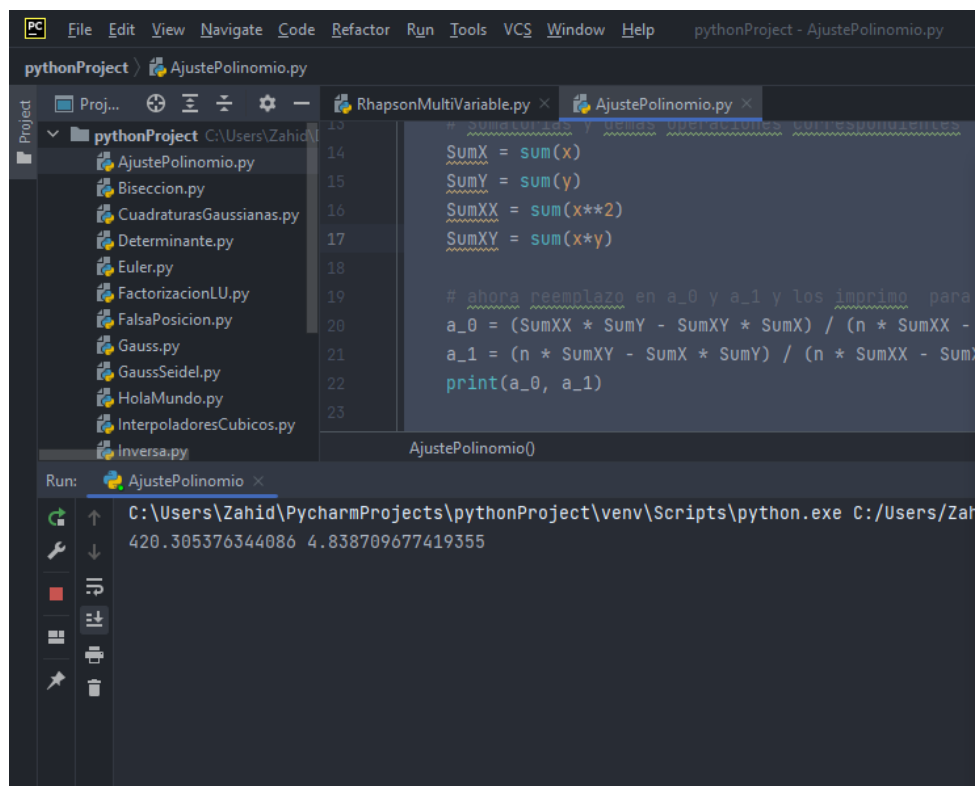
    # ahora reemplazo en a_0 y a_1 y los imprimo para visualizarlos
    a_0 = (SumXX * SumY - SumXY * SumX) / (n * SumXX - SumX ** 2)
    a_1 = (n * SumXY - SumX * SumY) / (n * SumXX - SumX ** 2)
    print(a_0, a_1)

    # Se generan valores para plotear la recta segun a_0 y a_1
    xa = np.linspace(0, 29, 100)
    ya = a_0 + a_1 * xa

    # Grafico de los puntos y la recta de ajuste
    plt.figure(1)
    plt.scatter(x, y, color='r')
    plt.grid(linestyle='dotted')
    plt.plot(xa, ya, color='b')
    plt.show()

AjustePolinomio(x, y)
```

Ejecución:



The screenshot shows the PyCharm IDE with a project named 'pythonProject'. The file explorer on the left lists several Python files, including 'AjustePolinomio.py'. The main editor window displays the code for 'AjustePolinomio.py', which calculates the coefficients of a linear polynomial using the least squares method. The code includes comments in Spanish and uses variables like 'SumX', 'SumY', 'SumXX', and 'SumXY'. The output window at the bottom shows the command 'C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Zahid\...' and the resulting output '420.305376344086 4.838709677419355'.

```
pythonProject - AjustePolinomio.py
pythonProject \ AjustePolinomio.py
# Sumatoria / DEMAS OPERATIVAS SUMESUMIENTES
14 SumX = sum(x)
15 SumY = sum(y)
16 SumXX = sum(x**2)
17 SumXY = sum(x*y)
18
19 # ahora reemplazo en a_0 y a_1 y los imprimo para
20 a_0 = (SumXX * SumY - SumXY * SumX) / (n * SumXX - SumX**2)
21 a_1 = (n * SumXY - SumX * SumY) / (n * SumXX - SumX**2)
22 print(a_0, a_1)
23
AjustePolinomio()
Run: AjustePolinomio
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\Zahid\...
420.305376344086 4.838709677419355
```

Figura 13: Salida del código de Ajuste de un polinomio por mínimos cuadrados

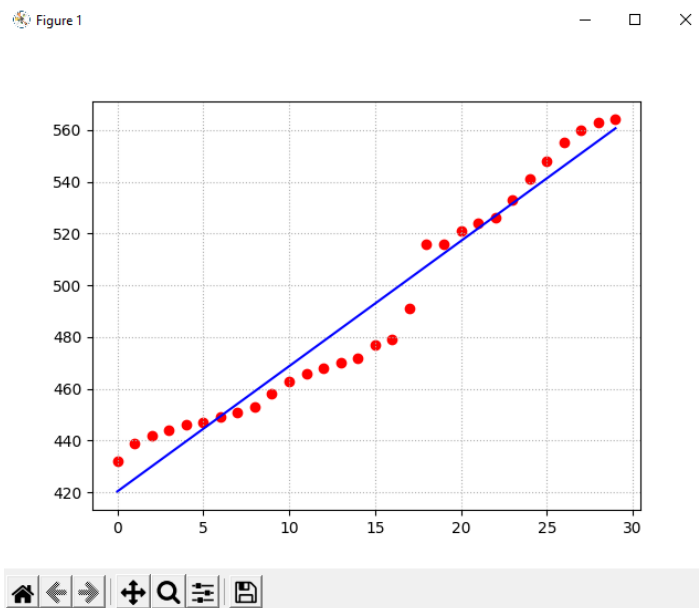


Figura 11: Grafica Ajuste de un polinomio por mínimos cuadrados

Interpoladores cúbicos

Código:

```
from math import *

def InteporladoresCubicos(datos):
    # Entrada:
    # datos == lista de puntos en "x" y "y" en el plano ordenados por
    x

    # Salida:
    # a == vector de coeficientes constantes
    # b == vector de coeficientes lineales
    # c == vector de coeficientes quadraticos
    # d == vector de coeficientes cubicos

    n = len(datos) - 1

    # Inicando los vectores aux
    A = [x[1] for x in datos]
    X = [x[0] for x in datos]
    H = [0.0 for x in range(n)]
    B = [0.0 for x in range(n + 1)]
    C = [0.0 for x in range(n + 1)]
    D = [0.0 for x in range(n + 1)]
    alpha = [0.0 for x in range(n)]
    mu = [0.0 for x in range(n + 1)]
    lo = [0.0 for x in range(n + 1)]
    z = [0.0 for x in range(n + 1)]

    # Creacion del vector 2
    for i in range(n):
        H[i] = X[i + 1] - X[i]

    # Vector n
    for i in range(1, n):
        alpha[i] = (3 / H[i] * (A[i + 1] - A[i]) - (3 / H[i - 1]) *
(A[i] - A[i - 1]))

    # Dar solucion al sistema tridiagonal
    for i in range(1, n):
        lo[i] = 2 * (X[i + 1] - X[i - 1]) - H[i - 1] * mu[i - 1]
        mu[i] = H[i] / lo[i]
        z[i] = (alpha[i] - H[i - 1] * z[i - 1]) / lo[i]
```

```

# Solucionar sistema tridiagonal
for j in range(n - 1, -1, -1):
    C[j] = z[j] - mu[j] * C[j + 1]
    B[j] = (A[j + 1] - A[j]) / (H[j]) - H[j] * (C[j + 1] + 2 *
C[j]) / 3
    D[j] = (C[j + 1] - C[j]) / (3 * H[j])

# Retorna vectores A, B, C, D
return A[:-1], B[:-1], C[:-1], D[:-1]

# Datos de prueba (1, 2), (2, 3), (3, 5)
DatosDePrueba = [[1, 2], [2, 3], [3, 5]]

# Llamada de la funcion
a, b, c, d = InteporladoresCubicos(DatosDePrueba)
print("Vectores de coeficientes:")
print("A =", a)
print("B =", b)
print("C =", c)
print("D =", d)

```

The screenshot displays a Python IDE with the following components:

- File Explorer:** Shows a project named 'pythonProject' containing various Python files, including 'InterpoladoresCubicos.py'.
- Code Editor:** Displays the source code of 'InterpoladoresCubicos.py', which includes the tridiagonal system solver and test data.
- Run Console:** Shows the execution output:


```

C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/InterpoladoresCubicos.py
Vectores de coeficientes:
A = [2, 3]
B = [0.75, 1.5]
C = [0.0, 0.75]
D = [0.25, -0.25]
Process finished with exit code 0

```

Figura 14: Salida del código de Interpoladores cúbicos

Calculo numérico

Derivación de datos tabulados.

Código:

```
import numpy as np
from tabulate import tabulate

valoraevaluar = 1
h = 0.5
DR = 6**2 # La derivada de la funcion

#funcion
def f(x):
    return ((2)*(x)**3)

# Progresiva (hacia adelante)
def fpf(x0, h):
    return (f(x0 + h) - f(x0)) / h
# Centrado
def fpc(x0, h):
    return (f(x0 + h) - f(x0 - h)) / (2*h)
# Hacia regresivo (Hacia atras)
def fpb(x0, h):
    return (f(x0) - f(x0 - h)) / h

def error(DR, valoraevaluar , h):
    return (abs(DR - fpf(valoraevaluar, h)) / DR) * 100 #Ecuacion porcentual

def DerivadaTab():
    print("Progresiva: ")
    print(fpb(valoraevaluar, h))
    print("Centrado: ")
    print(fpc(valoraevaluar, h))
    print("Regresiva: ")
    print(fpb(valoraevaluar, h))
    print("\n")
```

```

print("=== Tabulacion de resultados Progresiva ===")
    Table_Porcentual = [["0.5", fpf(valoraevaluar, 0.5), error(DR,
valoraevaluar, 0.5)], ["0.05", fpf(valoraevaluar, 0.05), error(DR,
valoraevaluar, 0.05)], ["0.01", fpf(valoraevaluar, 0.01), error(DR,
valoraevaluar, 0.01)]]
    print(tabulate(Table_Porcentual, headers= ["Tamaño h", "Derivada
aproxmidad", "Error (%)"], tablefmt= "francy_grid"))

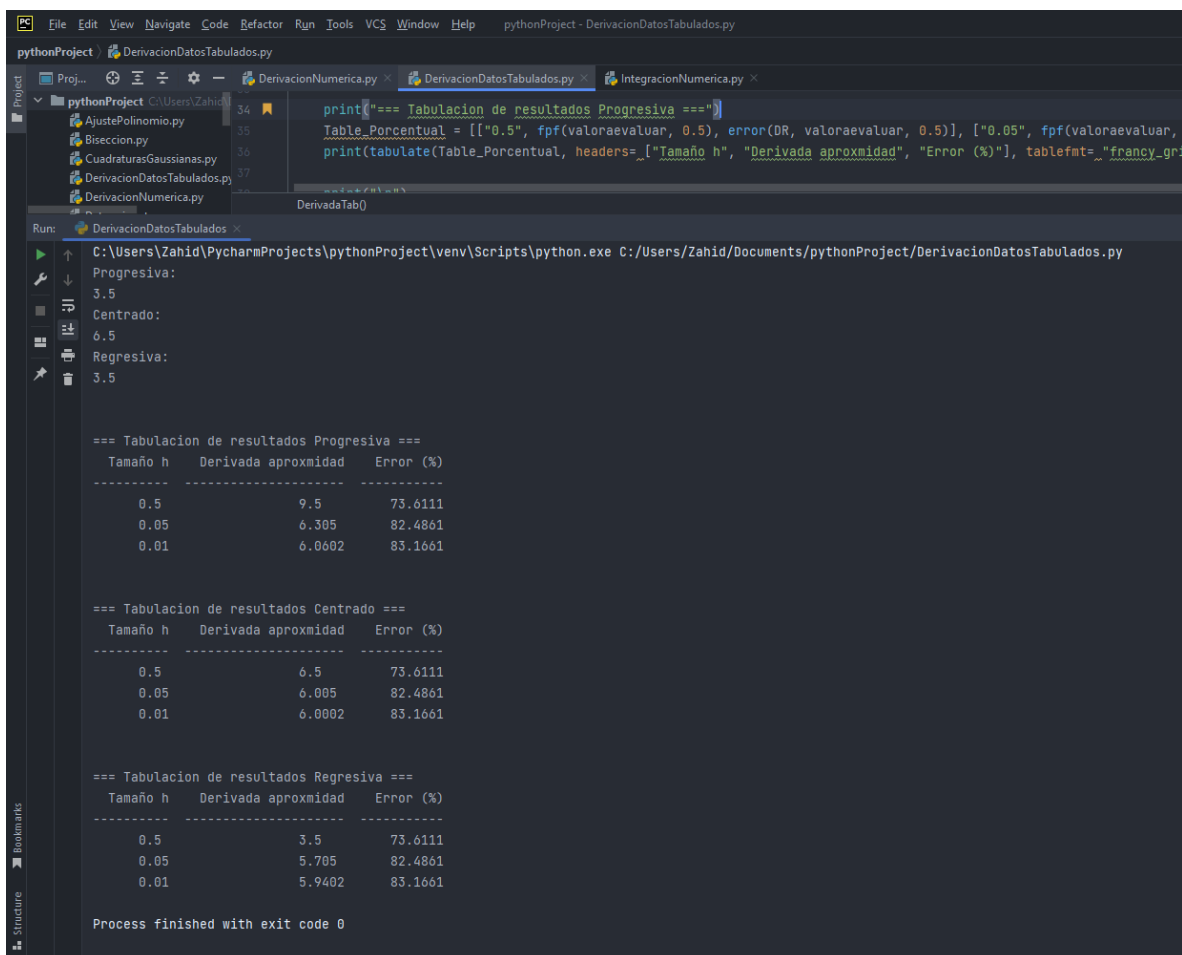
    print("\n")
    print("=== Tabulacion de resultados Centrado ===")
    Tabla_centrado = [["0.5", fpc(valoraevaluar, 0.5), error(DR, valoraevaluar,
0.5)], ["0.05", fpc(valoraevaluar, 0.05), error(DR, valoraevaluar, 0.05)],
["0.01", fpc(valoraevaluar, 0.01), error(DR, valoraevaluar, 0.01)]]
    print(tabulate(Tabla_centrado, headers= ["Tamaño h", "Derivada aproxmidad",
"Error (%)"], tablefmt= "francy_grid"))

    print("\n")
    print("=== Tabulacion de resultados Regresiva ===")
    Tabla_regresiva = [["0.5", fpb(valoraevaluar, 0.5), error(DR,
valoraevaluar, 0.5)], ["0.05", fpb(valoraevaluar, 0.05), error(DR,
valoraevaluar, 0.05)], ["0.01", fpb(valoraevaluar, 0.01), error(DR,
valoraevaluar, 0.01)]]
    print(tabulate(Tabla_regresiva, headers= ["Tamaño h", "Derivada
aproxmidad", "Error (%)"], tablefmt= "francy_grid"))

# Llamada de la funcion
DerivadaTab()

```

Ejecución:



```
pythonProject - DerivacionDatosTabulados.py
pythonProject C:\Users\Zahid\
DerivacionNumerica.py x DerivacionDatosTabulados.py x IntegracionNumerica.py x
34 print("=== Tabulacion de resultados Progresiva ===")
35 Table_Porcentual = [{"0.5", fpf(valoraevaluar, 0.5), error(DR, valoraevaluar, 0.5)], ["0.05", fpf(valoraevaluar,
36 print(tabulate(Table_Porcentual, headers=["Tamaño h", "Derivada aproximada", "Error (%)", tablefmt="francy_gri
37
DerivadaTab()
Run: DerivacionDatosTabulados x
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/DerivacionDatosTabulados.py
Progresiva:
3.5
Centrado:
6.5
Regresiva:
3.5

=== Tabulacion de resultados Progresiva ===
Tamaño h Derivada aproximada Error (%)
-----
0.5 9.5 73.6111
0.05 6.305 82.4861
0.01 6.0602 83.1661

=== Tabulacion de resultados Centrado ===
Tamaño h Derivada aproximada Error (%)
-----
0.5 6.5 73.6111
0.05 6.005 82.4861
0.01 6.0002 83.1661

=== Tabulacion de resultados Regresiva ===
Tamaño h Derivada aproximada Error (%)
-----
0.5 3.5 73.6111
0.05 5.705 82.4861
0.01 5.9402 83.1661

Process finished with exit code 0
```

Figura 15: Salida del código de Derivación de datos tabulados

Código:

```
import numpy as np

valoraevaluar = np.pi/2
h = 0.0001

def f(x):
    return np.sin(x**2) + 1

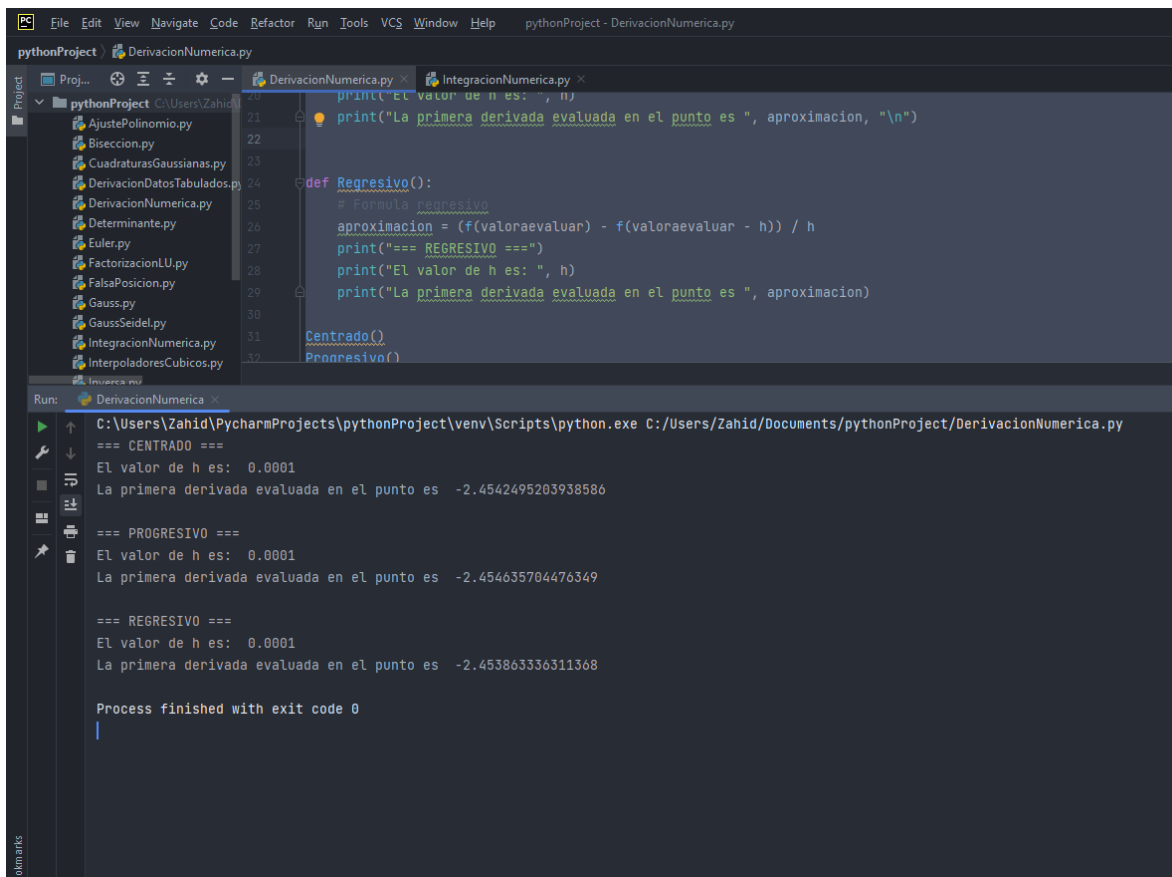
def Centrado():
    # Formula centrado
    aproximacion = (f(valoraevaluar + h) - f(valoraevaluar - h)) / (2*h)
    print("=== CENTRADO ===")
    print("El valor de h es: ", h)
    print("La primera derivada evaluada en el punto es ", aproximacion, "\n")

def Progresivo():
    # Formula progresivo
    aproximacion = (f(valoraevaluar + h) - f(valoraevaluar)) / h
    print("=== PROGRESIVO ===")
    print("El valor de h es: ", h)
    print("La primera derivada evaluada en el punto es ", aproximacion, "\n")

def Regresivo():
    # Formula regresivo
    aproximacion = (f(valoraevaluar) - f(valoraevaluar - h)) / h
    print("=== REGRESIVO ===")
    print("El valor de h es: ", h)
    print("La primera derivada evaluada en el punto es ", aproximacion)

Centrado()
Progresivo()
Regresivo()
```

Ejecución:



The screenshot displays the PyCharm IDE interface. The top toolbar includes icons for File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The main editor window shows the file `DerivacionNumerica.py` with the following Python code:

```
20 print("El valor de h es: ", h)
21 print("La primera derivada evaluada en el punto es ", aproximacion, "\n")
22
23
24 def Regresivo():
25     # Formula regresivo
26     aproximacion = (f(valoraevaluar) - f(valoraevaluar - h)) / h
27     print("=== REGRESIVO ===")
28     print("El valor de h es: ", h)
29     print("La primera derivada evaluada en el punto es ", aproximacion)
30
31 Centrado()
32 Progresivo()
```

The bottom panel shows the Run output for the script `DerivacionNumerica.py`. The output is as follows:

```
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/DerivacionNumerica.py
=== CENTRADO ===
El valor de h es: 0.0001
La primera derivada evaluada en el punto es -2.4542495203938586

=== PROGRESIVO ===
El valor de h es: 0.0001
La primera derivada evaluada en el punto es -2.454635704476349

=== REGRESIVO ===
El valor de h es: 0.0001
La primera derivada evaluada en el punto es -2.453863336311368

Process finished with exit code 0
```

Figura 16: Salida del código de Derivación de funciones

Integración de funciones

Código:

```
import numpy as np

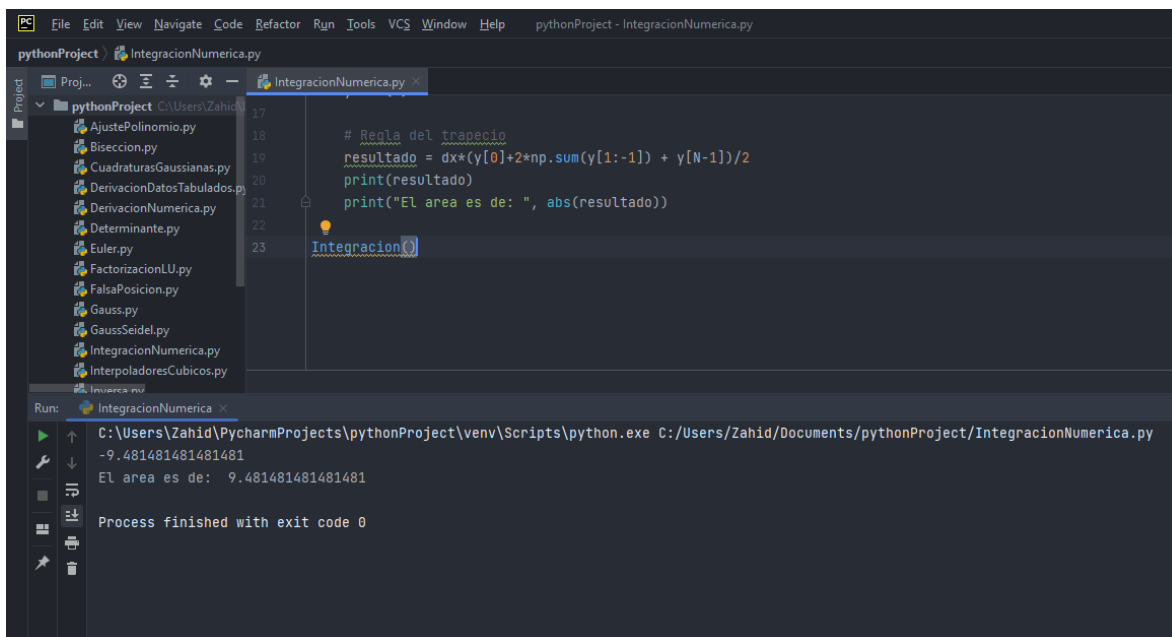
# Funcion
def f(x):
    return x**2 - 4*x

def Integracion(a, b, N):
    x = np.linspace(a, b, N)
    # Distancia entre cada dato
    dx = (b - a) / (N - 1)
    y = f(x)

    # Regla del trapecio
    resultado = dx*(y[0]+2*np.sum(y[1:-1]) + y[N-1])/2
    print(resultado)
    print("El area es de: ", abs(resultado))

Integracion(0, 4, 4)
# a y b son los intervalos
```


Ejecución:



The screenshot shows an IDE window titled 'pythonProject - IntegracionNumerica.py'. The left sidebar displays a project tree with various Python files, including 'IntegracionNumerica.py'. The main editor area shows the code for 'IntegracionNumerica.py' with the following content:

```
17  
18 # Regla del trapecio  
19 resultado = dx*(y[0]+2*np.sum(y[1:-1]) + y[N-1])/2  
20 print(resultado)  
21 print("El area es de: ", abs(resultado))  
22  
23 Integracion()
```

Below the editor, the 'Run' console shows the execution output:

```
Run: IntegracionNumerica  
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/IntegracionNumerica.py  
-9.481481481481481  
El area es de: 9.481481481481481  
Process finished with exit code 0
```

Figura 17: Salida del código de Integración de funciones

Integrador en cuadraturas Gaussianas.

Código:

```
# Integración: Cuadratura de Gauss de dos puntos
# modelo con varios tramos entre [a,b]
import numpy as np

# cuadratura de Gauss de dos puntos
def integraCuadGauss2p(funcionx,a,b):
    x0 = -1/np.sqrt(3)
    x1 = -x0
    xa = (b+a)/2 + (b-a)/2*(x0)
    xb = (b+a)/2 + (b-a)/2*(x1)
    area = ((b-a)/2)*(funcionx(xa) + funcionx(xb))
    return(area)

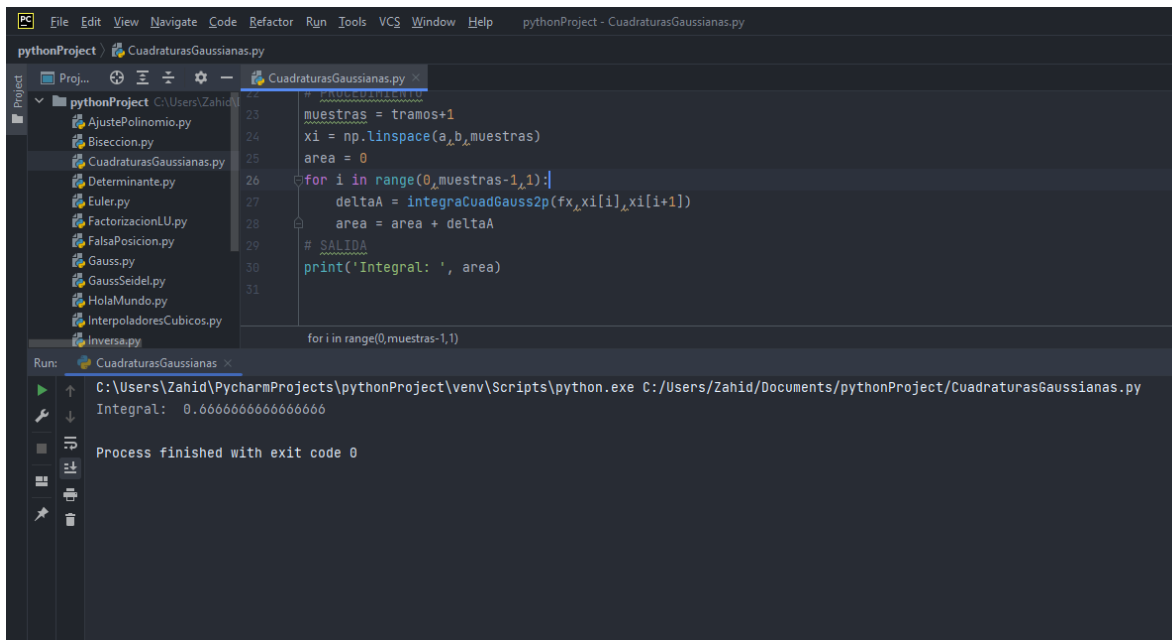
# INGRESO
fx = lambda x: (x-2)**2

# intervalo de integración
a = 1
b = 3
tramos = 1

# PROCEDIMIENTO
muestras = tramos+1
xi = np.linspace(a,b,muestras)
area = 0
for i in range(0,muestras-1,1):
    deltaA = integraCuadGauss2p(fx,xi[i],xi[i+1])
    area = area + deltaA

# SALIDA
print('Integral: ', area)
```

Ejecución:



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The project name is 'pythonProject'. The left sidebar shows a file explorer with a list of Python files: AjustePolinomio.py, Biseccion.py, CuadraturasGaussianas.py (selected), Determinante.py, Euler.py, FactorizacionLU.py, FalsaPosicion.py, Gauss.py, GaussSeidel.py, HolaMundo.py, InterpoladoresCubicos.py, and Inversa.py. The main editor window displays the code for 'CuadraturasGaussianas.py'. The code includes a docstring, variable assignments for 'muestras' and 'xi', a loop for calculating the integral, and a print statement. The bottom panel shows the 'Run' output, indicating the command executed and the resulting integral value.

```
22 """
23 muestras = tramos+1
24 xi = np.linspace(a,b,muestras)
25 area = 0
26 for i in range(0,muestras-1,1):
27     deltaA = integraCvadGauss2p(fx,xi[i],xi[i+1])
28     area = area + deltaA
29 # SALIDA
30 print('Integral: ', area)
31
```

Run: CuadraturasGaussianas

C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/CuadraturasGaussianas.py

Integral: 0.6666666666666666

Process finished with exit code 0

Figura 18: Salida del código de Integrador en cuadraturas Gaussianas.

Ecuaciones diferenciales

Euler centrado

Código:

```
import numpy as np
import matplotlib.pyplot as plt

# Metodo de Euler
def euler(f, x0, y0, x1, n):
    h = (x1 - x0) / n # tamaño de paso
    xi = np.zeros(n + 1) # vector de x variable independiente
    yi = np.zeros(n + 1) # vector de y variable dependiente
    xi[0] = x0 # tiempo inicial
    yi[0] = y0 # concentracion inicial

    for i in range(n):
        yi[i + 1] = yi[i] + h * f(xi[i], yi[i])
        xi[i + 1] = xi[i] + h
    return xi, yi # Vector de valores calculados

def f(x, y):
    return -2 * y

def f2(x):
    return np.exp(-2 * x) * 1.5

def main():
    x0 = 0 # valor inicial de tiempo
    y0 = 1.5 # valor inicial de la concentracion
    x1 = 0.6 # valor final del tiempo
    n = 20 # numero de pasos
    # llamada a la funcion euler
    x, y = euler(f, x0, y0, x1, n)
    print('x = ', x)
    print('y = ', y)
    # Grafica
    fig = plt.figure()
    plt.plot(x, y, '--', label='Euler ')
    plt.plot(np.linspace(x0, x1, 50), f2(np.linspace(x0, x1, 50)),
    label = 'Funcion ')
    plt.grid()
    plt.legend()
    plt.title('Metodo de Euler con ' + str(n) + ' pasos ')
    plt.xlabel('tiempo ')
    plt.ylabel('Concentracion ')
    plt.show()

main ()
```

Ejecución:

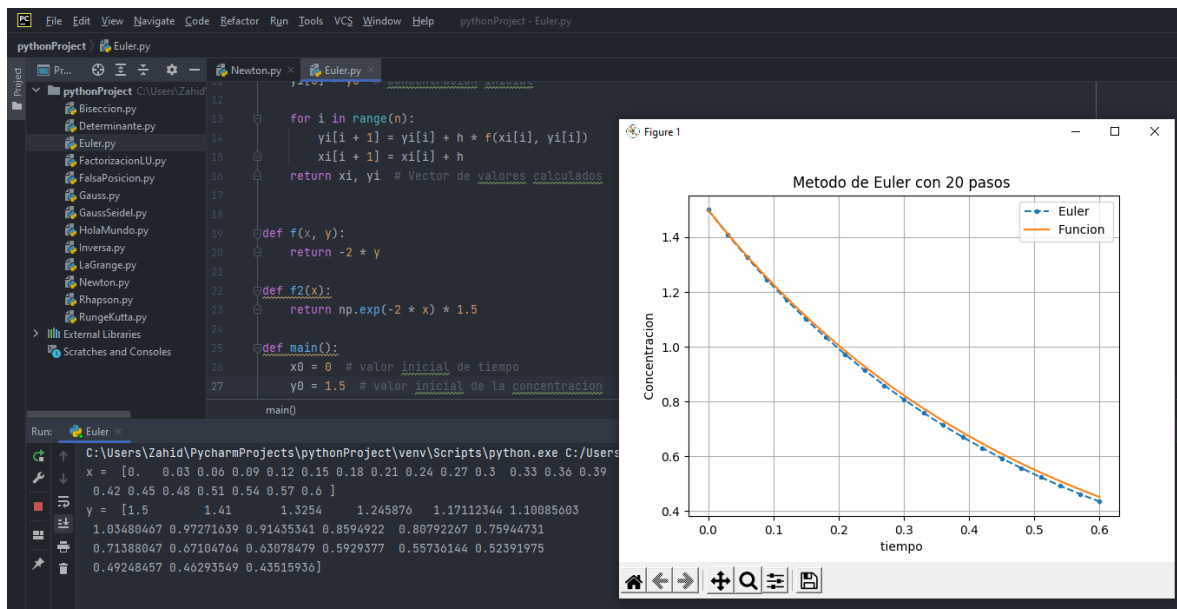


Figura 19: Salida del código de Método Euler

Métodos de Runge/Kutta 3o orden

Código:

```
import math
def f(y, t):
    #EDO a calcular
    return 2*y*t

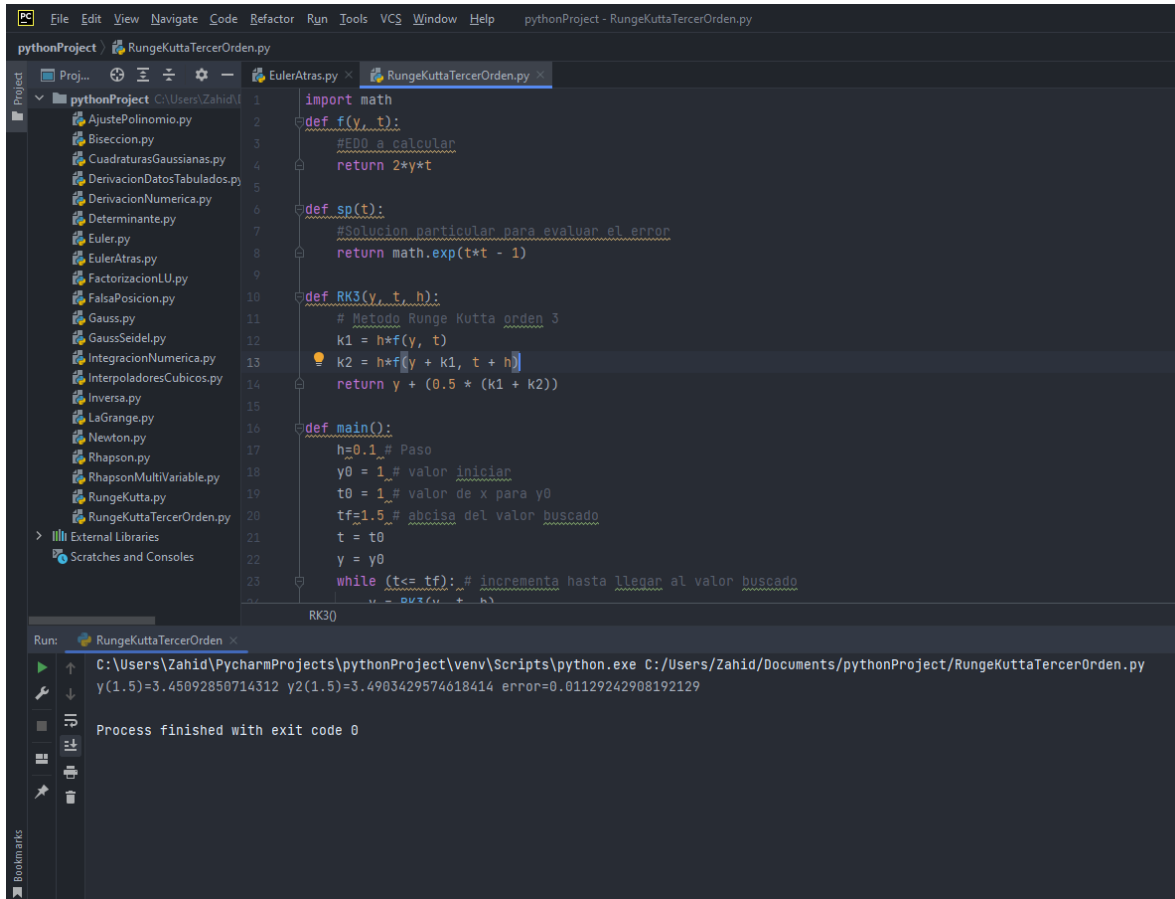
def sp(t):
    #Solucion particular para evaluar el error
    return math.exp(t*t - 1)

def RK3(y, t, h):
    # Metodo Runge Kutta orden 3
    k1 = h*f(y, t)
    k2 = h*f(y + k1, t + h)
    return y + (0.5 * (k1 + k2))

def main():
    h=0.1 # Paso
    y0 = 1 # valor iniciar
    t0 = 1 # valor de x para y0
    tf=1.5 # abcisa del valor buscado
    t = t0
    y = y0
    while (t<= tf): # incrementa hasta llegar al valor buscado
        y = RK3(y, t, h)
        t += h # incrementa un paso
    yp = sp(tf) # valor exacto
    err = (yp - y) / yp # error
    print("y(%s)=%s y2(%s)=%s error=%s" %(tf, y, tf, yp, err))

main()
```

Ejecución:



```
pythonProject RungeKuttaTercerOrden.py
1 import math
2 def f(y, t):
3     #E00 a calcular
4     return 2*y*t
5
6 def sp(t):
7     #Solucion particular para evaluar el error
8     return math.exp(t*t - 1)
9
10 def RK3(y, t, h):
11     # Metodo Runge Kutta orden 3
12     k1 = h*f(y, t)
13     k2 = h*f(y + k1, t + h)
14     return y + (0.5 * (k1 + k2))
15
16 def main():
17     h=0.1 # Paso
18     y0 = 1 # valor iniciar
19     t0 = 1 # valor de x para y0
20     tf=1.5 # abscisa del valor buscado
21     t = t0
22     y = y0
23     while (t<= tf): # incrementa hasta llegar al valor buscado
24         y = RK3(y, t, h)
25         t = t + h
26
27 RK30
```

Run: RungeKuttaTercerOrden

```
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/RungeKuttaTercerOrden.py
y(1.5)=3.45092850714312 y2(1.5)=3.4903429574618414 error=0.01129242908192129
Process finished with exit code 0
```

Figura 20: Salida del código de Runge/Kutta 3o orden

Métodos de Runge/Kutta 4o orden

Código:

```
from math import *

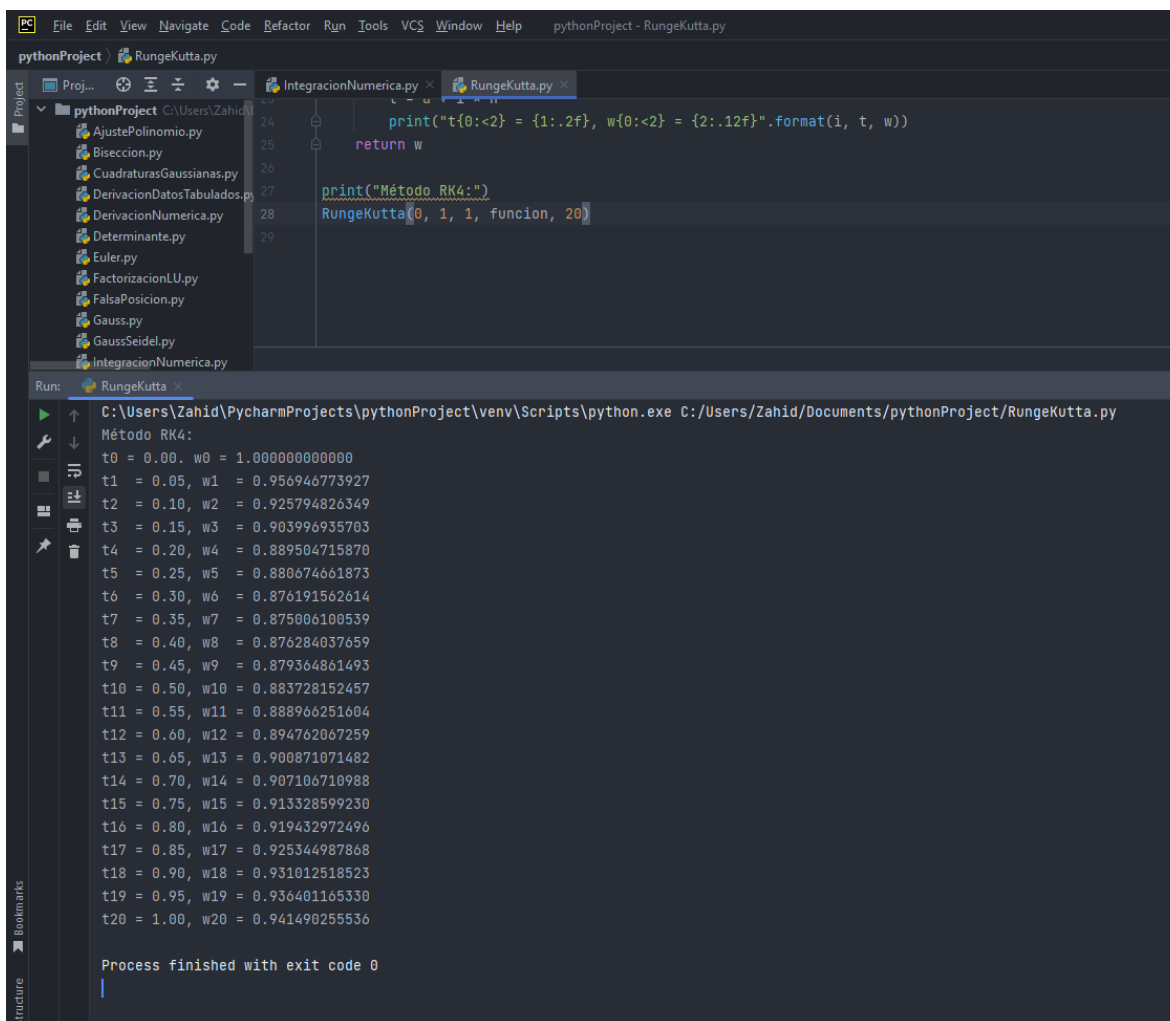
def funcion(t, y):
    return 2 - exp(-4 * t) - 2 * y

def RungeKutta(a, b, y0, f, N):
    # a == inicio intervalo
    # b == fin intervalo
    # y0 aproximacion lineal
    # f == funcion
    # N = pasos
    h = (b - a) / N
    t = a
    w = y0
    print("t0 = {0:.2f}. w0 = {1:.12f}".format(t, w))

    for i in range(1, N + 1):
        k1 = h * f(t, w)
        k2 = h * f(t + h / 2, w + k1 / 2)
        k3 = h * f(t + h / 2, w + k2 / 2)
        k4 = h * f(t + h, w + k3)
        w = w + (k1 + 2 * k2 + 2 * k3 + k4) / 6
        t = a + i * h
        print("t{0:<2} = {1:.2f}, w{0:<2} = {2:.12f}".format(i, t, w))
    return w

print("Método RK4:")
RungeKutta(0, 1, 1, funcion, 20)
```


Ejecución:



```
pythonProject - RungeKutta.py
pythonProject
├── AjustePolinomio.py
├── Biseccion.py
├── CuadraturasGaussianas.py
├── DerivacionDatosTabulados.py
├── DerivacionNumerica.py
├── Determinante.py
├── Euler.py
├── FactorizacionLU.py
├── FalsaPosicion.py
├── Gauss.py
├── GaussSeidel.py
├── IntegracionNumerica.py
└── RungeKutta.py

Run: RungeKutta
C:\Users\Zahid\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/Zahid/Documents/pythonProject/RungeKutta.py
Método RK4:
t0 = 0.00, w0 = 1.000000000000
t1 = 0.05, w1 = 0.956946773927
t2 = 0.10, w2 = 0.925794826349
t3 = 0.15, w3 = 0.903996935703
t4 = 0.20, w4 = 0.889504715870
t5 = 0.25, w5 = 0.880674661873
t6 = 0.30, w6 = 0.876191562614
t7 = 0.35, w7 = 0.875006100539
t8 = 0.40, w8 = 0.876284037659
t9 = 0.45, w9 = 0.879364861493
t10 = 0.50, w10 = 0.883728152457
t11 = 0.55, w11 = 0.888966251604
t12 = 0.60, w12 = 0.894762067259
t13 = 0.65, w13 = 0.900871071482
t14 = 0.70, w14 = 0.907106710988
t15 = 0.75, w15 = 0.913328599230
t16 = 0.80, w16 = 0.919432972496
t17 = 0.85, w17 = 0.925344987868
t18 = 0.90, w18 = 0.931012518523
t19 = 0.95, w19 = 0.936401165330
t20 = 1.00, w20 = 0.941490255536

Process finished with exit code 0
```

Figura 21: Salida del código de Runge/Kutta 4o orden