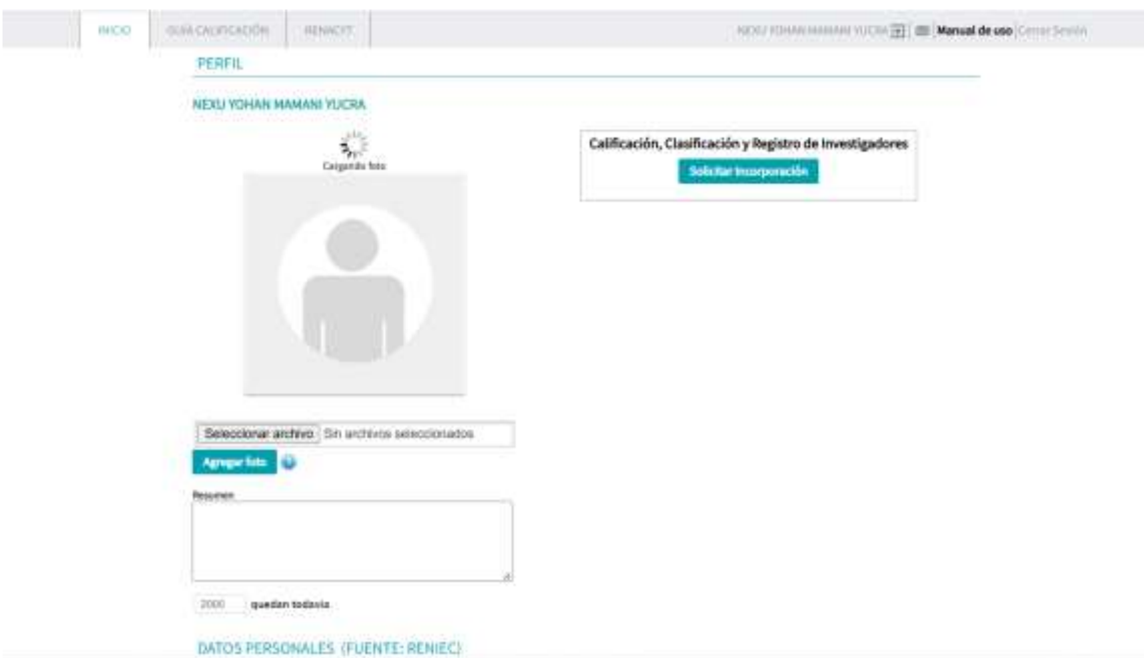
Trabajo encargado 1

NÚMERO	NOMBRES	APELLIDOS	ESPECIALIDAD	ÍNDICE h	DOCUMENTOS	CITA DE DOCUMENTOS	IDENTIFICADOR DE SCOPUS
n.1	Remo	Choquejahua Acero	Dr. Estadística Aplicada	1	2	2	58491331500
n.2	José Pánfilo	Tito Lipa	D. Sc. Ciencias Computación	0	3	0	58088282100
n.3	Juan Carlos	Juarez Vargas	Dr. Administración	1	3	2	58898176700
n.4	Vladimiro	Ibáñez Quispe	Dr. Administración	5	21	52	57201897173
n.5	Ángel	Javier Quispe Carita		1	3	1	
n.6	Alejandro	Apaza Tarqui	D. Sc. Ciencias Computación	1	5	6	57224362115
n.7	Elqui Yeye	Pari Condori	M.Sc. Informática, Mención GTIC	1	3	1	58284529900
n.8	Romel P.	Melgarejo-Bolívar		3	6	3	
n.9	Edgar Eloy	Carpio Vargas	Dr. Estadística e Informática, Dr. Gestión	3	8	27	57221112735
n.10	Ramiro	Laura Murillo		1	2	1	58284593600
n.11	Ernesto Nayer	Tumi Filgueira	M.Sc. Informática	3	6	23	57003617100
n.12	Leonid	Alemán Gonzales	Dr. Estadística e Informática	0	4	0	58898481200

NÚMERO	NOMBRES	APELLIDOS	ESPECIALIDAD	ÍNDICE h	DOCUMENTOS	CITA DE DOCUMENTOS	IDENTIFICADOR DE SCOPUS
n.13	Percy	Huata Panca	Dr. Economía y Gestión	2	8	14	57517171100
n.14	Fred	Torres-Cruz		4	40	40	
n.15	Leonel	Coyla Dme	Dr. Economía y Gestión, Dr. Estadística	1	5	1	58930847700
n.16	Milton Antonio	Lopez Cueva	Dr. Estadística e Informática	1	6	4	58671914800
n.18	Charles Ignacio	Mendoza Mollocondo	M.Sc. Informática	3	8	17	57562709900
n.19	Bernabé	Canqui Flores	Dr. Estadística Aplicada	3	9	20	57214094525

Trabajo encargado 2



TRABAJO ENCARGADO 3

En este tercer trabajo encargado se presentan cuatro métodos numéricos implementados en el lenguaje de programación Python.

Cada método incluye su respectiva explicación, código fuente y una tabla de iteraciones que muestra el proceso de convergencia hacia la raíz.

- Los métodos desarrollados son:
  - Método de la Falsa Posición
  - Método de la Secante
  - Método del Punto Fijo
  - Método de la Bisección

Método de la Falsa Posición

El método de la falsa posición (o regula falsi) es un procedimiento numérico para encontrar la raíz de una ecuación f(x)=0.
Se escogen dos valores iniciales a y b tales que f(a) y f(b) tengan signos opuestos, garantizando la existencia de una raíz entre ellos.
Luego se calcula el punto de intersección de la recta que une (a, f(a)) y (b, f(b)) con el eje x.
Dicho punto se convierte en la nueva aproximación de la raíz, repitiendo el proceso hasta alcanzar la tolerancia establecida.

Código en Python:

```
def falsa_posicion(f, a, b, tol=1e-6, max_iter=100):
    print("\n=== MÉTODO DE LA FALSA POSICIÓN ===")
    if f(a) * f(b) > 0:
        print("Error: f(a) y f(b) deben tener signos opuestos")
        return None
    print(f"{'Iter':<6} {'a':<15} {'b':<15} {'c':<15} {'f(c)':<15} {'Error':<15}")
    print("-" * 95)
    c_anterior = a
    for i in range(max_iter):
        fa, fb = f(a), f(b)
        c = (a * fb - b * fa) / (fb - fa)
        fc = f(c)
        error = abs(c - c_anterior) if i > 0 else abs(b - a)
        print(f"{i+1:<6} {a:<15.8f} {b:<15.8f} {c:<15.8f} {fc:<15.8e} {error:<15.8e}")
        if error < tol or abs(fc) < tol:
            print(f"\nConvergió en {i+1} iteraciones")
            print(f"Raíz aproximada: x = {c:.8f}")
```

```python
            return c
        if fa * fc < 0:
            b = c
        else:
            a = c
        c_anterior = c
    print(f"\nNo convergió en {max_iter} iteraciones")
    return c

def f(x): return x**2 - 2
raiz = falsa_posicion(f, 1, 2)
```

<u>salida:</u>

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win3
Enter "help" below or click "Help" above for more information.

============== RESTART: D:/trabajos de programacion  umeriac/2.py ==============

=== MÉTODO DE LA FALSA POSICIÓN ===
Iter   a              b              c              f(c)           Error
-----------------------------------------------------------------------------------------------
1      1.00000000     2.00000000     1.33333333     -2.22222222e-01 1.00000000e+00
2      1.33333333     2.00000000     1.40000000     -4.00000000e-02 6.66666667e-02
3      1.40000000     2.00000000     1.41176471     -6.92041522e-03 1.17647059e-02
4      1.41176471     2.00000000     1.41379310     -1.18906064e-03 2.02839757e-03
5      1.41379310     2.00000000     1.41414141     -2.04060810e-04 3.48310693e-04
6      1.41414141     2.00000000     1.41420118     -3.50127797e-05 5.97692905e-05
7      1.41420118     2.00000000     1.41421144     -6.00728684e-06 1.02550429e-05
8      1.41421144     2.00000000     1.41421320     -1.03068876e-06 1.75949467e-06
9      1.41421320     2.00000000     1.41421350     -1.76838272e-07 3.01881780e-07

Convergió en 9 iteraciones
Raíz aproximada: x = 1.41421350
```

Método de la Secante

El método de la secante es un procedimiento iterativo que busca la raíz de una función f(x)=0.
Utiliza dos valores iniciales y traza una secante entre ellos para aproximar la raíz.
Este proceso se repite hasta alcanzar la precisión deseada.

```python
def secante(f, x0, x1, tol=1e-6, max_iter=100):
    print("\n=== MÉTODO DE LA SECANTE ===")
    print(f"{'Iter':<6} {'x_n-1':<15} {'x_n':<15} {'x_n+1':<15} {'f(x_n+1)':<15}
{'Error':<15}")
    print("-" * 95)
    for i in range(max_iter):
        fx0, fx1 = f(x0), f(x1)
```

```
        if abs(fx1 - fx0) < 1e-12:
            print("Error: División por cero")
            return x1
        x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
        fx2 = f(x2)
        error = abs(x2 - x1)
        print(f"{i+1:<6} {x0:<15.8f} {x1:<15.8f} {x2:<15.8f} {fx2:<15.8e}
{error:<15.8e}")
        if error < tol or abs(fx2) < tol:
            print(f"\nConvergió en {i+1} iteraciones")
            print(f"Raíz aproximada: x = {x2:.8f}")
            return x2
        x0, x1 = x1, x2
    print(f"\nNo convergió en {max_iter} iteraciones")
    return x2

def f(x): return x**2 - 2
raiz = secante(f, 1, 2)
```

salida:

```
============================== RESTART: D:/trabajos de programacion  umeriac/3.py ================

=== MÉTODO DE LA SECANTE ===
Iter   x_n-1           x_n             x_n+1           f(x_n+1)        Error
------------------------------------------------------------------------------------------
1      1.00000000      2.00000000      1.33333333      -2.22222222e-01 6.66666667e-01
2      2.00000000      1.33333333      1.40000000      -4.00000000e-02 6.66666667e-02
3      1.33333333      1.40000000      1.41463415      1.18976800e-03  1.46341463e-02
4      1.40000000      1.41463415      1.41421144      -6.00728684e-06 4.22707867e-04
5      1.41463415      1.41421144      1.41421356      -8.93145558e-10 2.12358245e-06

Convergió en 5 iteraciones
Raíz aproximada: x = 1.41421356
```

Método del Punto Fijo

El método del punto fijo transforma la ecuación f(x)=0 en una forma equivalente x=g(x). A partir de un valor inicial x0, se aplica la iteración x_{n+1}=g(x_n) hasta que los valores converjan.
Cuando el cambio entre iteraciones es pequeño, se considera que se ha alcanzado el punto fijo.

```
import math
```

```
def punto_fijo(g, x0, tol=1e-6, max_iter=100):
    """
    Encuentra la raíz mediante el método del punto fijo.
    g: función de iteración (x = g(x))
    x0: valor inicial
    tol: tolerancia
    max_iter: número máximo de iteraciones
    """
    print("\n=== MÉTODO DEL PUNTO FIJO ===")
    print(f"{'Iter':<6} {'x_n':<15} {'x_n+1':<15} {'Error':<15}")
    print("-" * 60)
    for i in range(max_iter):
        x1 = g(x0)
        error = abs(x1 - x0)
        print(f"{i+1:<6} {x0:<15.8f} {x1:<15.8f} {error:<15.8e}")
        if error < tol:
            print(f"\nConvergió en {i+1} iteraciones")
            print(f"Punto fijo (raíz aproximada): x = {x1:.8f}")
            return x1
        x0 = x1
    print(f"\nNo convergió en {max_iter} iteraciones")
    return x
# Definimos g(x)
def g(x):
```

```
    return math.cos(x)

# Valor inicial

raiz = punto_fijo(g, 0.5)
```

salida:

```
==================================================================== RESTART: D:/trabajos de programacion  umeriac/4.py ===============

=== MÉTODO DEL PUNTO FIJO ===
Iter   x_n             x_n+1           Error
------------------------------------------------------------
1      0.50000000      0.87758256      3.77582562e-01
2      0.87758256      0.63901249      2.38570068e-01
3      0.63901249      0.80268510      1.63672607e-01
4      0.80268510      0.69477803      1.07907074e-01
5      0.69477803      0.76819583      7.34178045e-02
6      0.76819583      0.71916545      4.90303853e-02
7      0.71916545      0.75235576      3.31903135e-02
8      0.75235576      0.73008106      2.22746963e-02
9      0.73008106      0.74512034      1.50392782e-02
10     0.74512034      0.73500631      1.01140323e-02
11     0.73500631      0.74182652      6.82021363e-03
12     0.74182652      0.73723573      4.59079720e-03
13     0.73723573      0.74032965      3.09392644e-03
14     0.74032965      0.73824624      2.08341355e-03
15     0.73824624      0.73964996      1.40372444e-03
16     0.73964996      0.73870454      9.45423413e-04
17     0.73870454      0.73934145      6.36912924e-04
18     0.73934145      0.73891245      4.29002949e-04
19     0.73891245      0.73920144      2.88994804e-04
20     0.73920144      0.73900678      1.94664355e-04
21     0.73900678      0.73913791      1.31130981e-04
22     0.73913791      0.73904958      8.83301671e-05
23     0.73904958      0.73910908      5.95008253e-05
24     0.73910908      0.73906900      4.00802165e-05
25     0.73906900      0.73909600      2.69986317e-05
26     0.73909600      0.73907781      1.81865506e-05
27     0.73907781      0.73909006      1.22507031e-05
28     0.73909006      0.73908181      8.25221014e-06
29     0.73908181      0.73908737      5.55879293e-06
30     0.73908737      0.73908363      3.74446756e-06
31     0.73908363      0.73908615      2.52231940e-06
32     0.73908615      0.73908445      1.69906423e-06
33     0.73908445      0.73908559      1.14451031e-06
34     0.73908559      0.73908482      7.70955819e-07

Convergió en 34 iteraciones
Punto fijo (raíz aproximada): x = 0.73908482
```

Método de la Bisección

El método de la bisección es un algoritmo simple y confiable para encontrar raíces de funciones continuas.
Parte de un intervalo [a,b] donde f(a) y f(b) tienen signos opuestos, garantizando una raíz dentro.
En cada iteración, se calcula el punto medio y se selecciona el subintervalo donde la función cambia de signo,
repitiendo hasta alcanzar la precisión deseada.

```
def biseccion(f, a, b, tol=1e-6, max_iter=100):
    print("\n=== MÉTODO DE LA BISECCIÓN ===")
    if f(a) * f(b) > 0:
        print("Error: f(a) y f(b) deben tener signos opuestos")
        return None
```

```
    print(f"{'Iter':<6} {'a':<15} {'b':<15} {'c':<15} {'f(c)':<15} {'Error':<15}")
    print("-" * 95)
    for i in range(max_iter):
        c = (a + b) / 2
        fc = f(c)
        error = abs(b - a) / 2
        print(f"{i+1:<6} {a:<15.8f} {b:<15.8f} {c:<15.8f} {fc:<15.8e} {error:<15.8e}")
        if error < tol or abs(fc) < tol:
            print(f"\nConvergió en {i+1} iteraciones")
            print(f"Raíz aproximada: x = {c:.8f}")
            return c
        if f(a) * fc < 0:
            b = c
        else:
            a = c
    print(f"\nNo convergió en {max_iter} iteraciones")
    return c

def f(x): return x**2 - 2
raiz = biseccion(f, 1, 2)
```

salida:

```
======================================================================= RESTART: D:/trabajos de programacion  umeriac/5.py ================

=== MÉTODO DE LA BISECCIÓN ===
Iter   a               b               c               f(c)            Error
-----------------------------------------------------------------------------------------------
1      1.00000000      2.00000000      1.50000000      2.50000000e-01  5.00000000e-01
2      1.00000000      1.50000000      1.25000000      -4.37500000e-01 2.50000000e-01
3      1.25000000      1.50000000      1.37500000      -1.09375000e-01 1.25000000e-01
4      1.37500000      1.50000000      1.43750000      6.64062500e-02  6.25000000e-02
5      1.37500000      1.43750000      1.40625000      -2.24609375e-02 3.12500000e-02
6      1.40625000      1.43750000      1.42187500      2.17285156e-02  1.56250000e-02
7      1.40625000      1.42187500      1.41406250      -4.27246094e-04 7.81250000e-03
8      1.41406250      1.42187500      1.41796875      1.06353760e-02  3.90625000e-03
9      1.41406250      1.41796875      1.41601562      5.10025024e-03  1.95312500e-03
10     1.41406250      1.41601562      1.41503906      2.33554840e-03  9.76562500e-04
11     1.41406250      1.41503906      1.41455078      9.53912735e-04  4.88281250e-04
12     1.41406250      1.41455078      1.41430664      2.63273716e-04  2.44140625e-04
13     1.41406250      1.41430664      1.41418457      -8.20010900e-05 1.22070312e-04
14     1.41418457      1.41430664      1.41424561      9.06325877e-05  6.10351562e-05
15     1.41418457      1.41424561      1.41421509      4.31481749e-06  3.05175781e-05
16     1.41418457      1.41421509      1.41419983      -3.88433691e-05 1.52587891e-05
17     1.41419983      1.41421509      1.41420746      -1.72643340e-05 7.62939453e-06
18     1.41420746      1.41421509      1.41421127      -6.47477282e-06 3.81469727e-06
19     1.41421127      1.41421509      1.41421318      -1.07998130e-06 1.90734863e-06
20     1.41421318      1.41421509      1.41421413      1.61741718e-06  9.53674316e-07

Convergió en 20 iteraciones
Raíz aproximada: x = 1.41421413
```