Hoja de Trabajo No.3

Métodos Numericos 1

José Rodrigo Oliveros González 22848

Ejercicios a : d utilizando método JACOBI

In [141…

```python
import numpy as np
import pandas as pd

def JACOBI(A, b, max_iter=100, error=1e-2):
    """
    Argumentos:
    A: Matriz cuadrada(n x n)
    b: Vector (n)
    max_iter: Máximo número de iteraciones
    error: Tolerancia para convergencia

    Returns:
    tabla: Lista de iteraciones [i, x1, x2, ..., xn]
    solucion: Vector solución final
    """
    # Numero de incognitas o ecuaciones a trabajar
    n = len(b)

    # Programación defensiva
    if A.shape != (n, n):
        raise ValueError("La matriz A debe ser cuadrada, n filas n columnas. Vector b tiene n elementos")
    if np.any(np.diag(A) == 0):
        raise ValueError("La diagonal de A no puede tener ceros")

    # Inicializar tabla
    tabla = []
    x_actual = np.zeros(n)
    tabla.append([0] + x_actual.tolist())

    # Iteraciones
    for k in range(1, max_iter + 1):
        x_nuevo = np.zeros(n)

        for i in range(n):
            # Suma de a_ij * x_j para j != i (elimina la diagonal)
            suma = np.dot(A[i, :], x_actual) - A[i, i] * x_actual[i]
            x_nuevo[i] = (b[i] - suma) / A[i, i]

        # Agregar a la tabla
        tabla.append([k] + x_nuevo.tolist())

        # Porcentaje de error
        if np.linalg.norm(x_nuevo - x_actual) < error:
            break

        x_actual = x_nuevo.copy() # Para no perder los valores actuales

    return tabla, x_actual

print("\n--- Inciso a ---")
A0 = np.array([[1, 1, 3],
              [3, 5, 1],
              [4, 1, 2]], dtype=float)
b0 = np.array([3, 7, 4], dtype=float)

tabla0, sol0 = JACOBI(A0, b0, max_iter=5)

df0 = pd.DataFrame(tabla0, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b0))])
print(df0)

print("\n--- Inciso b ---")
A1 = np.array([[5, -1, -1],
              [1, -1, 2],
              [3, -1, 2]], dtype=float)
b1 = np.array([3, 0, 4], dtype=float)

tabla1, sol1 = JACOBI(A1, b1, max_iter=5)
df1 = pd.DataFrame(tabla1, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b1))])
print(df1)

print("\n--- Inciso c ---")
A2 = np.array([[1,2,4,8,16],
              [1,6,36,216,1296],
              [1,10,100,1000,10000],
              [1,20,400,8000,160000],
              [1,30,900,27000,810000]], dtype=float)
b2 = np.array([13.4, 30.4, 41.8,57.9,66.5], dtype=float)

tabla2, sol2 = JACOBI(A2, b2, max_iter=5)
df2 = pd.DataFrame(tabla2, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b2))])
print(df2)

print("\n--- Inciso d ---")
A3 = np.array([
    [8, 0, 6, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 9, 0, 0, 0, 0, 5, 0, 2, 1, 0],
    [0, 1, 7, 0, 0, 1, 2, 0, 0, 1, 0],
    [0, 1, 0, 6, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 9, 0, 0, 0, 1, 1, 1],
    [0, 1, 0, 2, 0, 10, 1, 0, 3, 0, 0],
    [0, 0, 5, 0, 2, 2, 10, 0, 0, 0, 0],
    [0, 0, 0, 6, 1, 0, 0, 15, 0, 2, 0],
    [0, 2, 0, 0, 4, 0, 1, 1, 20, 1, 0],
    [0, 0, 0, 0, 0, 3, 0, 6, 5, 25, 1],
    [1, 0, 3, 1, 5, 0, 7, 0, 0, 1, 12]
], dtype=float)
b3 = np.array([1,5,8,0,8,1,0,3,0,1,2], dtype=float)

tabla3, sol3 = JACOBI(A3, b3, max_iter=30)
df3 = pd.DataFrame(tabla3, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b3))])
print(df3)
```

```
--- Inciso a ---
   Iteración      x1      x2     x3
0          0    0.00   0.000   0.00
1          1    3.00   1.400   2.00
2          2   -4.40  -0.800  -4.70
3          3   17.90   4.980  11.20
4          4  -35.58 -11.580 -36.29
5          5  123.45  30.006  78.95

--- Inciso b ---
   Iteración     x1    x2    x3
0          0  0.000  0.00  0.00
1          1  0.600 -0.00  2.00
2          2  1.000  4.60  1.10
3          3  1.740  3.20  2.80
4          4  1.800  7.34  0.99
5          5  2.266  3.78  2.97

--- Inciso c ---
   Iteración         x1         x2        x3        x4        x5
0          0   0.000000   0.000000  0.000000  0.000000  0.000000
1          1  13.400000   5.066667  0.418000  0.007238  0.000082
2          2   1.535453   0.047050 -0.303252 -0.029646 -0.000828
3          3  14.769320   7.876332  0.777181  0.038646  0.001404
4          4  -5.793017  -3.752424 -1.044151 -0.081231 -0.002380
5          5  25.769373  15.735379  1.901440  0.117142  0.004096

--- Inciso d ---
    Iteración        x1        x2        x3        x4        x5        x6        x7        x8        x9       x10       x11
0           0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1           1  0.125000  0.555556  1.142857  0.000000  0.888889  0.100000  0.000000  0.200000  0.000000  0.040000  0.166667
2           2 -0.732143  0.551111  1.043492 -0.120370  0.865926  0.044444 -0.769206  0.135407 -0.245333 -0.026667 -0.503168
3           3 -0.642573  1.040374  1.281361  0.120211  0.975019  0.219484 -0.703820  0.193975 -0.195273  0.071362  0.066960
4           4 -0.851047  0.982032  1.153774 -0.067252  0.895217  0.100884 -0.879581  0.077400 -0.277117  0.003484 -0.111786
5           5 -0.731924  1.105406  1.238966  0.001556  0.931713  0.186340 -0.776107  0.166755 -0.237312  0.069213  0.094539
6           6 -0.804419  1.031772  1.170179 -0.070640  0.897062  0.137952 -0.843094  0.128035 -0.269876  0.021299 -0.023463
7           7 -0.743805  1.081547  1.213595 -0.027536  0.919116  0.176223 -0.792093  0.165612 -0.247902  0.047631  0.063297
8           8 -0.781754  1.045404  1.182683 -0.058792  0.904108  0.150932 -0.825865  0.143389 -0.263035  0.026155  0.002666
9           9 -0.754664  1.069916  1.204177 -0.037034  0.914913  0.168715 -0.802350  0.159756 -0.252546  0.039975  0.043905
10         10 -0.773504  1.052985  1.189442 -0.051912  0.907630  0.156414 -0.818814  0.148489 -0.259843  0.030166  0.015090
11         11 -0.760592  1.064843  1.199723 -0.041543  0.912732  0.164918 -0.807530  0.156234 -0.254816  0.036958  0.035040
12         12 -0.769600  1.056703  1.192620 -0.048726  0.909202  0.159022 -0.815392  0.150841 -0.258314  0.032275  0.021255
13         13 -0.763374  1.062368  1.197540 -0.043761  0.911643  0.163108 -0.809955  0.154573 -0.255897  0.035528  0.030827
14         14 -0.767685  1.058449  1.194129 -0.047207  0.909949  0.160280 -0.813720  0.151991 -0.257573  0.033276  0.024205
15         15 -0.764696  1.061163  1.196491 -0.044822  0.911121  0.162240 -0.811111  0.153783 -0.256412  0.034835  0.028794
```

Ejercicios a : d utilizando método GAUSS SEIDEL

In [144…

```python
import numpy as np
import pandas as pd

def gauss_seidel(A, b, x0=None, error=1e-2, max_iter=1000):
    """
    Argumentos:
        A: Matriz de coeficientes (n x n)
        b: Vector (n)
        x0: Vector inicial (por si tenemos idea de donde esta la solución)
        error: Tolerancia para convergencia
        max_iter: Número máximo de iteraciones

    Retorna:
        tabla: Lista de interaciones[i, x1, x2, ...]
        solucion: Vector solución
    """
    # Numero de variables que vamos a resolver
    n = len(b)

    # Programación defensiva
    if A.shape != (n, n):
        raise ValueError("La matriz A debe ser cuadrada, n filas n columnas. Vector b tiene n elementos")
    if np.any(np.diag(A) == 0):
        raise ValueError("La diagonal de A no puede tener ceros")

    # Inicialización
    tabla = []
    x = np.zeros(n) if x0 is None else x0.copy() # Si ya conocemos donde mas o menos esta la respuesta podemos iniciar un valor cercarno
    tabla.append([0] + x.tolist())

    # Iteraciones
    for k in range(1, max_iter + 1):
        x_anterior = x.copy()

        for i in range(n):
            # Suma de a_ij*x_j usando el valor actual
            suma = 0.0
            for j in range(n):
                if j != i:
                    suma += A[i, j] * x[j]

            # Actualización el valor actual
            x[i] = (b[i] - suma) / A[i, i]

        # Registrar iteración
        tabla.append([k] + x.tolist())

        # Porcentaje de eror
        if np.linalg.norm(x - x_anterior) < error:
            break

    return tabla, x


print("\n--- Inciso a ---")
A0 = np.array([[1, 1, 3],
              [3, 5, 1],
              [4, 1, 2]], dtype=float)
b0 = np.array([3, 7, 4], dtype=float)

tabla0, sol0 = gauss_seidel(A0, b0, max_iter=5)

df0 = pd.DataFrame(tabla0, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b0))])
print(df0)

print("\n--- Inciso b ---")
A1 = np.array([[5, -1, -1],
              [1, -1, 2],
              [3, -1, 2]], dtype=float)
b1 = np.array([3, 0, 4], dtype=float)

tabla1, sol1 = gauss_seidel(A1, b1, max_iter=50)
df1 = pd.DataFrame(tabla1, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b1))])
print(df1)

print("\n--- Inciso c ---")
A2 = np.array([[1,2,4,8,16],
              [1,6,36,216,1296],
              [1,10,100,1000,10000],
              [1,20,400,8000,160000],
              [1,30,900,27000,810000]], dtype=float)
b2 = np.array([13.4, 30.4, 41.8,57.9,66.5], dtype=float)

tabla2, sol2 = gauss_seidel(A2, b2, max_iter=50)
df2 = pd.DataFrame(tabla2, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b2))])
print(df2)

print("\n--- Inciso d ---")
A3 = np.array([
    [8, 0, 6, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 9, 0, 0, 0, 0, 5, 0, 2, 1, 0],
    [0, 1, 7, 0, 0, 1, 2, 0, 0, 1, 0],
    [0, 1, 0, 6, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 9, 0, 0, 0, 1, 1, 1],
    [0, 1, 0, 2, 0, 10, 1, 0, 3, 0, 0],
    [0, 0, 5, 0, 2, 2, 10, 0, 0, 0, 0],
    [0, 0, 0, 6, 1, 0, 0, 15, 0, 2, 0],
    [0, 2, 0, 0, 4, 0, 1, 1, 20, 1, 0],
    [0, 0, 0, 0, 0, 3, 0, 6, 5, 25, 1],
    [1, 0, 3, 1, 5, 0, 7, 0, 0, 1, 12]
], dtype=float)
b3 = np.array([1,5,8,0,8,1,0,3,0,1,2], dtype=float)

tabla3, sol3 = gauss_seidel(A3, b3, max_iter=50)
df3 = pd.DataFrame(tabla3, columns=['Iteración'] + [f'x{i+1}' for i in range(len(b3))])
print(df3)
```

```
--- Inciso a ---
   Iteración         x1          x2          x3
0          0     0.0000     0.00000     0.00000
1          1     3.0000    -0.40000    -3.80000
2          2    14.8000    -6.72000   -24.24000
3          3    82.4400   -43.21600  -141.27200
4          4   470.0320  -252.36480  -811.88160
5          5  2691.0096 -1450.82944 -4654.60448

--- Inciso b ---
    Iteración        x1        x2        x3
0           0  0.000000  0.000000  0.000000
1           1  0.600000  0.600000  1.400000
2           2  1.000000  3.800000  2.400000
3           3  1.840000  6.640000  2.560000
4           4  2.440000  7.560000  2.120000
5           5  2.536000  6.776000  1.584000
6           6  2.272000  5.440000  1.312000
7           7  1.950400  4.574400  1.361600
8           8  1.787200  4.510400  1.574400
9           9  1.816960  4.965760  1.757440
10         10  1.944640  5.459520  1.812800
11         11  2.054464  5.680064  1.758336
12         12  2.087680  5.604352  1.670656
13         13  2.055002  5.396314  1.615654
14         14  2.002394  5.233702  1.613261
15         15  1.969393  5.195914  1.643868
16         16  1.967956  5.255693  1.675912
17         17  1.986321  5.338144  1.689591
18         18  2.005547  5.384729  1.684044
19         19  2.013754  5.381842  1.670289
20         20  2.010426  5.351005  1.659863
21         21  2.002174  5.321900  1.657689
22         22  1.995918  5.311297  1.661772
23         23  1.994614  5.318157  1.667158

--- Inciso c ---
    Iteración         x1        x2        x3        x4        x5
0           0   0.000000  0.000000  0.000000  0.000000  0.000000
1           1  13.400000  2.833333  0.000667 -0.001554  0.000012
2           2   7.742913  3.825608 -0.027616 -0.002147  0.000033
3           3   5.875898  4.323197 -0.054918 -0.002221  0.000050
4           4   4.990250  4.633675 -0.078036 -0.002064  0.000060
5           5   4.460352  4.852885 -0.097232 -0.001787  0.000064
6           6   4.096426  5.017743 -0.113311 -0.001443  0.000065
7           7   3.828257  5.146350 -0.127009 -0.001060  0.000063
8           8   3.622805  5.249433 -0.138891 -0.000659  0.000059
9           9   3.461018  5.334051 -0.149374 -0.000251  0.000055
10         10   3.330532  5.405077 -0.158761  0.000154  0.000049
11         11   3.222871  5.465940 -0.167270  0.000552  0.000043
12         12   3.132093  5.519082 -0.175063  0.000939  0.000037
13         13   3.053984  5.566248 -0.182257  0.001312  0.000031
14         14   2.985539  5.608698 -0.188940  0.001671  0.000025
15         15   2.924601  5.647345 -0.195181  0.002014  0.000019
16         16   2.869615  5.682857 -0.201029  0.002342  0.000013
17         17   2.819450  5.715730 -0.206525  0.002654  0.000008
18         18   2.773277  5.746335 -0.211702  0.002952  0.000003
19         19   2.730479  5.774955 -0.216586  0.003234 -0.000002
20         20   2.690594  5.801810 -0.221199  0.003503 -0.000007
21         21   2.653266  5.827072 -0.225561  0.003758 -0.000012
22         22   2.618220  5.850884 -0.229688  0.004000 -0.000016
23         23   2.585237  5.873361 -0.233595  0.004230 -0.000020
24         24   2.554138  5.894600 -0.237294  0.004448 -0.000024
25         25   2.524775  5.914688 -0.240798  0.004655 -0.000028
26         26   2.497023  5.933697 -0.244118  0.004851 -0.000031
27         27   2.470772  5.951695 -0.247264  0.005037 -0.000035
28         28   2.445928  5.968740 -0.250246  0.005213 -0.000038
29         29   2.422404  5.984887 -0.253072  0.005380 -0.000041
30         30   2.400124  6.000187 -0.255750  0.005538 -0.000044
31         31   2.379016  6.014686 -0.258289  0.005689 -0.000046
32         32   2.359015  6.028427 -0.260696  0.005831 -0.000049
33         33   2.340062  6.041450 -0.262977  0.005966 -0.000051
34         34   2.322098  6.053795 -0.265140  0.006094 -0.000054
35         35   2.305072  6.065497 -0.267190  0.006216 -0.000056
36         36   2.288933  6.076590 -0.269134  0.006331 -0.000058
37         37   2.273635  6.087106 -0.270977  0.006440 -0.000060
38         38   2.259133  6.097074 -0.272724  0.006543 -0.000062
39         39   2.245385  6.106524 -0.274380  0.006641 -0.000063
40         40   2.232353  6.115483 -0.275950  0.006734 -0.000065
41         41   2.219998  6.123976 -0.277438  0.006822 -0.000067
42         42   2.208286  6.132028 -0.278849  0.006906 -0.000068
43         43   2.197183  6.139661 -0.280187  0.006985 -0.000070
44         44   2.186656  6.146897 -0.281455  0.007060 -0.000071
45         45   2.176677  6.153757 -0.282657  0.007131 -0.000072
46         46   2.167217  6.160261 -0.283797  0.007199 -0.000073
47         47   2.158248  6.166426 -0.284877  0.007263 -0.000075
48         48   2.149746  6.172271 -0.285902  0.007324 -0.000076
49         49   2.141685  6.177813 -0.286873  0.007381 -0.000077

--- Inciso d ---
   Iteración        x1        x2        x3        x4        x5        x6        x7        x8        x9       x10       x11
0          0  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1          1  0.125000  0.555556  1.063492 -0.092593  0.888889  0.062963 -0.722116  0.177778 -0.206116  0.031001 -0.053626
2          2 -0.661045  0.999090  1.193025 -0.037225  0.914305  0.141582 -0.807690  0.149803 -0.251426  0.039488  0.013501
3          3 -0.765116  1.055757  1.196936 -0.043595  0.910937  0.159340 -0.812524  0.151444 -0.256684  0.035329  0.026296
4          4 -0.767253  1.060073  1.195758 -0.045641  0.910562  0.161378 -0.812267  0.152842 -0.256915  0.034284  0.027033
```

Ejercicios a : d utilizando método Descomposición LU

In [145…

```python
import numpy as np
import pandas as pd

def lu_simplificado(A, b):
    """
    Resuelve Ax = b con descomposición LU manual
    Devuelve solo los resultados esenciales

    Argumentos:
        A: Matriz cuadrada (n x n)
        b: Vector (n)

    Returns:
        DataFrame con resultados
        Vector solución x
    """
    n = A.shape[0]
    U = A.copy().astype(float)
    L = np.eye(n)
    P = np.eye(n)

    for k in range(n-1):
        # Pivoteo parcial
        max_row = np.argmax(np.abs(U[k:, k])) + k
        if max_row != k:
            U[[k, max_row]] = U[[max_row, k]]
            P[[k, max_row]] = P[[max_row, k]]
            if k > 0:
                L[[k, max_row], :k] = L[[max_row, k], :k]

        # Eliminación
        for i in range(k+1, n):
            L[i, k] = U[i, k] / U[k, k]
            U[i, k:] -= L[i, k] * U[k, k:]

    # Resolver sistema
    y = np.linalg.solve(L, P @ b)
    x = np.linalg.solve(U, y)
    return x

A0 = np.array([[1, 1, 3],
              [3, 5, 1],
              [4, 1, 2]], dtype=float)
b0 = np.array([3, 7, 4], dtype=float)

# Resolver y mostrar
solucion0 = lu_simplificado(A0, b0)

print("\n=== Inciso A ===")
print("\nVector solución:")
print(np.array2string(solucion0, precision=6, suppress_small=True))


A1 = np.array([[5, -1, -1],
              [1, -1, 2],
              [3, -1, 2]], dtype=float)
b1 = np.array([3, 0, 4], dtype=float)

# Resolver y mostrar
solucion1 = lu_simplificado(A1, b1)

print("\n=== Inciso B ===")
print("\nVector solución:")
print(np.array2string(solucion1, precision=6, suppress_small=True))

A2 = np.array([[1,2,4,8,16],
              [1,6,36,216,1296],
              [1,10,100,1000,10000],
              [1,20,400,8000,160000],
              [1,30,900,27000,810000]], dtype=float)
b2 = np.array([13.4, 30.4, 41.8,57.9,66.5], dtype=float)


# Resolver y mostrar
solucion2 = lu_simplificado(A2, b2)

print("\n=== Inciso C ===")
print("\nVector solución:")
print(np.array2string(solucion2, precision=6, suppress_small=True))


A3 = np.array([
    [8, 0, 6, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 9, 0, 0, 0, 0, 5, 0, 2, 1, 0],
    [0, 1, 7, 0, 0, 1, 2, 0, 0, 1, 0],
    [0, 1, 0, 6, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 9, 0, 0, 0, 1, 1, 1],
    [0, 1, 0, 2, 0, 10, 1, 0, 3, 0, 0],
    [0, 0, 5, 0, 2, 2, 10, 0, 0, 0, 0],
    [0, 0, 0, 6, 1, 0, 0, 15, 0, 2, 0],
    [0, 2, 0, 0, 4, 0, 1, 1, 20, 1, 0],
    [0, 0, 0, 0, 0, 3, 0, 6, 5, 25, 1],
    [1, 0, 3, 1, 5, 0, 7, 0, 0, 1, 12]
], dtype=float)
b3 = np.array([1,5,8,0,8,1,0,3,0,1,2], dtype=float)


# Resolver y mostrar
solucion3 = lu_simplificado(A3, b3)

print("\n=== Inciso D ===")
print("\nVector solución:")
print(np.array2string(solucion3, precision=6, suppress_small=True))
```

```
=== Inciso A ===

Vector solución:
[0.5 1.  0.5]

=== Inciso B ===

Vector solución:
[2.       5.333333 1.666667]

=== Inciso C ===

Vector solución:
[ 1.994643  6.278899 -0.304588  0.00843  -0.000095]

=== Inciso D ===

Vector solución:
[-0.765918  1.060052  1.195524 -0.045798  0.910642  0.161438 -0.812178
  0.15305  -0.256887  0.034196  0.026916]
```

Explicaciones

- ¿Cuántas iteraciones necesitó en los incisos anteriores para obtener convergencia en 2 cifras significativas?
    - Con el método JACOBI solo fue posible resolver el ejercicio D, tomando 15 iteraciones. Con este mismo método los ejercicios a, b y c fallaron al tener un valor de xn cercano al infinito, en este caso fallaron a las 720, 4934 y 1409 iteraciones respectivamente.
    - Con el metodo GAUSS SEIDEL tampoco fue posible resolverl el ejercicio A, sin embargo se obtuvieron resultados satisfactorios en los ejercicios b, c y d. Estos convergerieron en 2 cifras significativas al llegar a las 23, 49 y 4 iteraciones respectivamente.
    - El método directo descomposición LU tuvo el mayor rango de acierto, con este método todos los ejercicios convergen en vectores solución como se muestra en los resultados del respectivo método.