ESCUELA POLITECNICA NACIONAL

MÉTODOS NUMÉRICOS - GR1CC

Darlin Joel Anacicha Sanchez

2024-07-27

CONJUNTO DE EJERCICIOS

1.Encuentre las primeras dos iteraciones del método de Jacobi para los siguientes sistemas lineales, por medio de ( ) = 0:

```
import numpy as np

def jacobi(A, b, x0, iteraciones):
    n = len(A)
    x = x0.copy()
    for it in range(iteraciones):
        x_new = np.zeros_like(x)
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        x = x_new
        print(f"Iteración {it + 1}: {x}")
    return x

# Ejercicio 1a
A1a = np.array([[3, -1, 1], [3, 6, 2], [3, 3, 7]], dtype=np.float32)
b1a = np.array([1, 0, 4], dtype=np.float32)
x0 = np.zeros(len(b1a))
print("Ejercicio 1a:")
jacobi(A1a, b1a, x0, 2)

# Ejercicio 1b
```

```python
A1b = np.array([[10, -1, 0], [-1, 10, -2], [0, -2, 10]], dtype=np.float32)
b1b = np.array([9, 7, 6], dtype=np.float32)
x0 = np.zeros(len(b1b))
print("Ejercicio 1b:")
jacobi(A1b, b1b, x0, 2)

# Ejercicio 1c
A1c = np.array([[10, 5, 0], [5, 10, -4], [-4, 8, -1]], dtype=np.float32)
b1c = np.array([6, 25, -11], dtype=np.float32)
x0 = np.zeros(len(b1c))
print("Ejercicio 1c:")
jacobi(A1c, b1c, x0, 2)

# Ejercicio 1d
A1d = np.array([
    [4, 1, 0, 1, 0],
    [-1, 3, 1, 1, 0],
    [2, 1, 5, 0, 1],
    [1, 1, 3, 4, 1],
    [2, 0, 1, 1, 4]
], dtype=np.float32)
b1d = np.array([6, 6, 6, 6, 6], dtype=np.float32)
x0 = np.zeros(len(b1d))
print("Ejercicio 1d:")
jacobi(A1d, b1d, x0, 2)
```

```
Ejercicio 1a:
Iteración 1: [0.33333333 0.         0.57142857]
Iteración 2: [ 0.14285714 -0.35714286  0.42857143]
Ejercicio 1b:
Iteración 1: [0.9 0.7 0.6]
Iteración 2: [0.97 0.91 0.74]
Ejercicio 1c:
Iteración 1: [ 0.6  2.5 11. ]
Iteración 2: [-0.65  6.6  28.6 ]
Ejercicio 1d:
Iteración 1: [1.5 2.  1.2 1.5 1.5]
Iteración 2: [ 0.625  1.6   -0.1   -0.65   0.075]

array([ 0.625,  1.6  , -0.1  , -0.65 ,  0.075])
```

2.Repita el ejercicio 1 usando el método de Gauss-Siedel.

```
def gauss_seidel(A, b, x0, iteraciones):
    n = len(A)
    x = x0.copy()
    for it in range(iteraciones):
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x[i] = (b[i] - s1 - s2) / A[i][i]
        print(f"Iteración {it + 1}: {x}")
    return x

# Ejercicio 2a
gauss_seidel(A1a, b1a, x0, 2)

# Ejercicio 2b
gauss_seidel(A1b, b1b, x0, 2)

# Ejercicio 2c
gauss_seidel(A1c, b1c, x0, 2)

# Ejercicio 2d
gauss_seidel(A1d, b1d, x0, 2)
```

```
Iteración 1: [ 0.33333333 -0.16666667  0.5         0.          0.        ]
Iteración 2: [ 0.11111111 -0.22222222  0.61904762  0.          0.        ]
Iteración 1: [0.9    0.79   0.758 0.     0.    ]
Iteración 2: [0.979   0.9495 0.7899 0.      0.     ]
Iteración 1: [ 0.6   2.2 26.2   0.    0. ]
Iteración 2: [ -0.5    13.23 118.84    0.      0.   ]
Iteración 1: [1.5      2.5      0.1      0.425    0.61875]
Iteración 2: [0.76875    2.08125    0.3525     0.3684375  0.93539062]
```

```
array([0.76875   , 2.08125   , 0.3525    , 0.3684375 , 0.93539062])
```

3.Utilice el método de Jacobi para resolver los sistemas lineales en el ejercicio 1, con TOL = 10-3.

```
import numpy as np

def jacobi_tol(A, b, x0, tol, max_iter=30):
    n = len(A)
```

```python
    x = x0.copy()
    x_new = x0.copy()
    it = 0
    while it < max_iter:
        it += 1
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        if np.allclose(x, x_new, atol=tol):
            break
        x = x_new.copy()
        print(f"Iteración {it}: {x}")
    return x

# Resolver sistemas del ejercicio 1 con TOL = 10^{-3} y un máximo de 30 iteraciones
tol = 1e-3
jacobi_tol(A1a, b1a, x0, tol)
jacobi_tol(A1b, b1b, x0, tol)
jacobi_tol(A1c, b1c, x0, tol)
jacobi_tol(A1d, b1d, x0, tol)
```

```
Iteración 1: [1.  1.4 1. ]
Iteración 2: [0.15 0.6  0.2 ]
Iteración 3: [0.75 1.27 0.75]
Iteración 4: [0.3075     0.8        0.32666667]
Iteración 5: [0.63666667 1.15016667 0.63083333]
Iteración 6: [0.39704167 0.89183333 0.40438889]
Iteración 7: [0.57484722 1.08089722 0.570375  ]
Iteración 8: [0.44458819 0.94101667 0.44808519]
Iteración 9: [0.54070324 1.04363005 0.53813171]
Iteración 10: [0.47002663 0.96795171 0.4718889 ]
Iteración 11: [0.52206762 1.02360624 0.52067389]
Iteración 12: [0.4837615  0.98262465 0.48477538]
Iteración 13: [0.51195615 1.01278803 0.51120462]
Iteración 14: [0.49120069 0.99058539 0.49175194]
Iteración 15: [0.50647768 1.0069292  0.50607131]
Iteración 16: [0.49523205 0.99489913 0.49553104]
Iteración 17: [0.5035097  1.00375456 0.50328961]
Iteración 18: [0.49741655 0.99723626 0.49757858]
Iteración 19: [0.50190165 1.00203435 0.5017824 ]
Iteración 20: [0.49860021 0.99850253 0.498688  ]
```

```
Iteración 21: [0.50103037 1.00110227 0.50096575]
Iteración 22: [0.49924156 0.99918863 0.49928912]
Iteración 23: [0.50055828 1.00059724 0.50052327]
Iteración 24: [0.49958905 0.99956038 0.49961483]
Iteración 1: [0.9 0.7 0.6]
Iteración 2: [0.97 0.91 0.74]
Iteración 3: [0.991 0.945 0.782]
Iteración 4: [0.9945 0.9555 0.789 ]
Iteración 5: [0.99555 0.95725 0.7911 ]
Iteración 1: [ 0.6  2.5 11. ]
Iteración 2: [-0.65  6.6  28.6 ]
Iteración 3: [-2.7   14.265 66.4  ]
Iteración 4: [ -6.5325  30.41   135.92  ]
Iteración 5: [-14.605    60.13425 280.41   ]
Iteración 6: [-29.467125 121.9665   550.494   ]
Iteración 7: [ -60.38325    237.4311625 1104.6005   ]
Iteración 8: [-118.11558125  474.531825   2151.9823    ]
Iteración 9: [-236.6659125   922.35071063 4279.716925  ]
Iteración 10: [-460.57535531 1832.71972625 8336.469335  ]
Iteración 11: [ -915.75986313  3567.37541166 16515.05923125]
Iteración 12: [-1783.08770583  7066.40362406 32213.04274575]
Iteración 13: [-3532.60181203 13779.26095121 63674.57981581]
Iteración 14: [ -6889.03047561  27238.63283234 124375.49485784]
Iteración 15: [-13618.71641617  53197.21318094 245476.18456115]
Iteración 16: [-26598.00659047 105002.33203255 480063.57111219]
Iteración 17: [-52500.56601627 205326.93174011 946421.68262225]
Iteración 18: [-102662.86587006  404821.45605704 1852628.71798598]
Iteración 19: [-202410.12802852  792385.42012942 3649234.11193651]
Iteración 20: [-396192.11006471 1560901.20878886 7148734.87314942]
Iteración 21: [ -780450.00439443  3057592.50429212 14071989.11056974]
Iteración 22: [-1528795.65214606  6019023.14642511 27582551.0519147 ]
Iteración 23: [-3009510.97321256 11797420.74683891 54267378.77998513]
Iteración 24: [-5.89870977e+06  2.32117095e+07  1.06417421e+08]
Iteración 25: [-1.16058541e+07  4.55163257e+07  2.09288526e+08]
Iteración 26: [-2.27581623e+07  8.95183400e+07  4.10554033e+08]
Iteración 27: [-4.47591694e+07  1.75600697e+08  8.07179380e+08]
Iteración 28: [-8.78003479e+07  3.45251339e+08  1.58384226e+09]
Iteración 29: [-1.72625669e+08  6.77437082e+08  3.11321212e+09]
Iteración 30: [-3.38718541e+08  1.33159768e+09  6.10999935e+09]

IndexError: index 3 is out of bounds for axis 0 with size 3
```

4.Utilice el método de Gauss-Siedel para resolver los sistemas lineales en el

ejercicio 1, con TOL = 10-3.

```
import numpy as np

def gauss_seidel_tol(A, b, x0, tol, max_iter=30):
    n = len(A)
    x = x0.copy()
    it = 0
    while it < max_iter:
        it += 1
        x_old = x.copy()
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x_old[j] for j in range(i + 1, n))
            x[i] = (b[i] - s1 - s2) / A[i][i]
        if np.allclose(x, x_old, atol=tol):
            break
        print(f"Iteración {it}: {x}")
    return x

# Resolver sistemas del ejercicio 1 con TOL = 10^{-3} y un máximo de 30 iteraciones
tol = 1e-3
gauss_seidel_tol(A1a, b1a, x0, tol)
gauss_seidel_tol(A1b, b1b, x0, tol)
gauss_seidel_tol(A1c, b1c, x0, tol)
gauss_seidel_tol(A1d, b1d, x0, tol)
```

```
Iteración 1: [1.  0.8 0.4]
Iteración 2: [0.6  0.96 0.48]
Iteración 3: [0.52  0.992 0.496]
Iteración 4: [0.504  0.9984 0.4992]
Iteración 5: [0.5008  0.99968 0.49984]
Iteración 1: [0.9   0.79   0.758]
Iteración 2: [0.979   0.9495 0.7899]
Iteración 3: [0.99495  0.957475 0.791495]
Iteración 1: [ 0.6  2.2 26.2]
Iteración 2: [ -0.5   13.23 118.84]
Iteración 3: [ -6.015   53.0435 459.408 ]
Iteración 4: [ -25.92175   199.224075 1708.4796   ]
Iteración 5: [ -99.0120375   735.39785875 6290.23102   ]
Iteración 6: [ -367.09892938  2702.14187269 23096.530699  ]
```

```
Iteración 7: [-1350.47093634  9916.34774777 84743.66572755]
Iteración 8: [ -4957.57387389  36378.75322796 310871.32131925]
Iteración 9: [ -18188.77661398  133445.41683469 1140329.44113344]
Iteración 10: [ -66722.10841734  489495.33066205 4182862.07896578]
Iteración 11: [ -244747.06533102  1795520.86425182 15343166.17533869]
Iteración 12: [ -897759.83212591  6586148.88619843 56280241.41809111]
Iteración 13: [-3.29307384e+06  2.41586360e+07  2.06441394e+08]
Iteración 14: [-1.20793174e+07  8.86162189e+07  7.57247032e+08]
Iteración 15: [-4.43081089e+07  3.25052870e+08  2.77765540e+09]
Iteración 16: [-1.62526434e+08  1.19232538e+09  1.01887088e+10]
Iteración 17: [-5.96162690e+08  4.37356487e+09  3.73731697e+10]
Iteración 18: [-2.18678243e+09  1.60426591e+10  1.37088403e+11]
Iteración 19: [-8.02132955e+09  5.88460258e+10  5.02853525e+11]
Iteración 20: [-2.94230129e+10  2.15852916e+11  1.84451538e+12]
Iteración 21: [-1.07926458e+11  7.91769382e+11  6.76586089e+12]
Iteración 22: [-3.95884691e+11  2.90428670e+12  2.48178324e+13]
Iteración 23: [-1.45214335e+12  1.06532046e+13  9.10342104e+13]
Iteración 24: [-5.32660231e+12  3.90769853e+13  3.33922292e+14]
Iteración 25: [-1.95384927e+13  1.43338163e+14  1.22485927e+15]
Iteración 26: [-7.16690815e+13  5.25778251e+14  4.49290233e+15]
Iteración 27: [-2.62889125e+14  1.92860549e+15  1.64804005e+16]
Iteración 28: [-9.64302747e+14  7.07431156e+15  6.04517034e+16]
Iteración 29: [-3.53715578e+15  2.59492593e+16  2.21742697e+17]
Iteración 30: [-1.29746296e+16  9.51843937e+16  8.13373668e+17]

IndexError: index 3 is out of bounds for axis 0 with size 3
```

5.El sistema lineal

```python
import numpy as np

def jacobi(A, b, x0, iteraciones):
    n = len(A)
    x = x0.copy()
    for it in range(iteraciones):
        x_new = np.zeros_like(x)
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        x = x_new
        print(f"Iteración {it + 1}: {x}")
```

```python
    return x

A = np.array([[2, -1, 1], [2, 2, 2], [-1, -1, 2]], dtype=np.float32)
b = np.array([-1, 4, -5], dtype=np.float32)
x0 = np.zeros(len(b))

print("Método de Jacobi:")
jacobi(A, b, x0, 25)
```

```
Método de Jacobi:
Iteración 1: [-0.5  2.  -2.5]
Iteración 2: [ 1.75  5.   -1.75]
Iteración 3: [2.875 2.    0.875]
Iteración 4: [ 0.0625 -1.75   -0.0625]
Iteración 5: [-1.34375  2.      -3.34375]
Iteración 6: [ 2.171875  6.6875   -2.171875]
Iteración 7: [3.9296875 2.        1.9296875]
Iteración 8: [-0.46484375 -3.859375    0.46484375]
Iteración 9: [-2.66210938  2.         -4.66210938]
Iteración 10: [ 2.83105469  9.32421875 -2.83105469]
Iteración 11: [5.57763672 2.         3.57763672]
Iteración 12: [-1.28881836 -7.15527344  1.28881836]
Iteración 13: [-4.7220459  2.        -6.7220459]
Iteración 14: [ 3.86102295 13.4440918  -3.86102295]
Iteración 15: [8.15255737 2.         6.15255737]
Iteración 16: [ -2.57627869 -12.30511475   2.57627869]
Iteración 17: [-7.94069672  2.         -9.94069672]
Iteración 18: [ 5.47034836 19.88139343 -5.47034836]
Iteración 19: [12.1758709  2.        10.1758709]
Iteración 20: [ -4.58793545 -20.35174179   4.58793545]
Iteración 21: [-12.96983862   2.         -14.96983862]
Iteración 22: [ 7.98491931 29.93967724 -7.98491931]
Iteración 23: [18.46229827  2.         16.46229827]
Iteración 24: [ -7.73114914 -32.92459655   7.73114914]
Iteración 25: [-20.82787284   2.         -22.82787284]

array([-20.82787284,   2.        , -22.82787284])
```

6.El sistema lineal

```python
def gauss_seidel(A, b, x0, tol, max_iter):
    n = len(A)
    x = x0.copy()
    for it in range(max_iter):
        x_new = x.copy()
        for i in range(n):
            s1 = sum(A[i][j] * x_new[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            print(f"Convergencia alcanzada en iteración {it + 1}")
            break
        x = x_new
        print(f"Iteración {it + 1}: {x}")
    return x

x0 = np.zeros(len(b))
tol = 1e-5
max_iter = 100

print("Método de Gauss-Seidel:")
gauss_seidel(A, b, x0, tol, max_iter)
```

```
Método de Gauss-Seidel:
Iteración 1: [-0.5  2.5 -1.5]
Iteración 2: [ 1.5   2.   -0.75]
Iteración 3: [ 0.875  1.875 -1.125]
Iteración 4: [ 1.      2.125  -0.9375]
Iteración 5: [ 1.03125  1.90625 -1.03125]
Iteración 6: [ 0.96875   2.0625   -0.984375]
Iteración 7: [ 1.0234375  1.9609375 -1.0078125]
Iteración 8: [ 0.984375    2.0234375  -0.99609375]
Iteración 9: [ 1.00976562  1.98632812 -1.00195312]
Iteración 10: [ 0.99414062  2.0078125  -0.99902344]
Iteración 11: [ 1.00341797  1.99560547 -1.00048828]
Iteración 12: [ 0.99804688  2.00244141 -0.99975586]
Iteración 13: [ 1.00109863  1.99865723 -1.00012207]
Iteración 14: [ 0.99938965  2.00073242 -0.99993896]
Iteración 15: [ 1.00033569  1.99960327 -1.00003052]
Iteración 16: [ 0.99981689  2.00021362 -0.99998474]
Iteración 17: [ 1.00009918  1.99988556 -1.00000763]
Iteración 18: [ 0.99994659  2.00006104 -0.99999619]
```

```
Iteración 19: [ 1.00002861  1.99996758 -1.00000191]
Iteración 20: [ 0.99998474  2.00001717 -0.99999905]
Iteración 21: [ 1.00000811  1.99999094 -1.00000048]
Iteración 22: [ 0.99999571  2.00000477 -0.99999976]
Convergencia alcanzada en iteración 23
```

```
array([ 0.99999571,  2.00000477, -0.99999976])
```

```python
import numpy as np

def gauss_seidel(A, b, x0, tol, max_iter):
    n = len(A)
    x = x0.copy()
    for it in range(max_iter):
        x_new = x.copy()
        for i in range(n):
            s1 = sum(A[i][j] * x_new[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            print(f"Convergencia alcanzada en iteración {it + 1}")
            break
        x = x_new
        print(f"Iteración {it + 1}: {x}")
    return x

A = np.array([[1, 0, -1], [-0.5, 1, -0.25], [1, -0.5, 1]], dtype=np.float32)
b = np.array([0.2, -1.425, 2], dtype=np.float32)
x0 = np.zeros(len(b))
tol = 1e-5
max_iter = 300

print("Método de Gauss-Seidel:")
solution = gauss_seidel(A, b, x0, tol, max_iter)
print(f"Solución aproximada: {solution}")
```

```
Método de Gauss-Seidel:
Iteración 1: [ 0.2        -1.32499995  1.13750002]
Iteración 2: [ 1.33750002 -0.47187493  0.42656251]
Iteración 3: [ 0.62656251 -1.00507807  0.87089845]
Iteración 4: [ 1.07089846 -0.67182611  0.59318849]
```

```
Iteración 5: [ 0.79318849 -0.88010858  0.76675722]
Iteración 6: [ 0.96675722 -0.74993204  0.65827676]
Iteración 7: [ 0.85827676 -0.83129238  0.72607705]
Iteración 8: [ 0.92607705 -0.78044217  0.68370187]
Iteración 9: [ 0.88370187 -0.81222355  0.71018635]
Iteración 10: [ 0.91018636 -0.79236019  0.69363355]
Iteración 11: [ 0.89363355 -0.80477479  0.70397905]
Iteración 12: [ 0.90397906 -0.79701566  0.69751311]
Iteración 13: [ 0.89751312 -0.80186512  0.70155433]
Iteración 14: [ 0.90155433 -0.79883421  0.69902857]
Iteración 15: [ 0.89902857 -0.80072852  0.70060717]
Iteración 16: [ 0.90060717 -0.79954458  0.69962054]
Iteración 17: [ 0.89962055 -0.80028454  0.70023718]
Iteración 18: [ 0.90023719 -0.79982206  0.69985178]
Iteración 19: [ 0.89985179 -0.80011111  0.70009266]
Iteración 20: [ 0.90009266 -0.79993046  0.69994211]
Iteración 21: [ 0.89994211 -0.80004337  0.7000362 ]
Iteración 22: [ 0.90003621 -0.7999728   0.6999774 ]
Iteración 23: [ 0.8999774  -0.8000169   0.70001415]
Iteración 24: [ 0.90001415 -0.79998934  0.69999118]
Iteración 25: [ 0.89999118 -0.80000657  0.70000554]
Iteración 26: [ 0.90000554 -0.7999958   0.69999656]
Convergencia alcanzada en iteración 27
Solución aproximada: [ 0.90000554 -0.7999958   0.69999656]
```

```
        for i in range(n):
            s1 = sum(A[i][j] * x_new[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            print(f"Convergencia alcanzada en iteración {it + 1}")
            break
        x = x_new
        print(f"Iteración {it + 1}: {x}")
    return x

A = np.array([[1, 0, -1], [-0.5, 1, -0.25], [1, -0.5, 1]], dtype=np.float32)
b = np.array([0.2, -1.425, 2], dtype=np.float32)
x0 = np.zeros(len(b))
tol = 1e-5
max_iter = 300
```

```python
print("Método de Gauss-Seidel:")
solution = gauss_seidel(A, b, x0, tol, max_iter)
print(f"Solución aproximada: {solution}")
```

```python
b_nuevo = np.array([0.3, -1.5, 2.1], dtype=np.float32)

print("Método de Gauss-Seidel con sistema cambiado:")
solution_nuevo = gauss_seidel(A, b_nuevo, x0, tol, max_iter)
print(f"Solución aproximada con sistema cambiado: {solution_nuevo}")
```

```
Método de Gauss-Seidel con sistema cambiado:
Iteración 1: [ 0.30000001 -1.34999999  1.1249999 ]
Iteración 2: [ 1.42499991 -0.50625007  0.42187496]
Iteración 3: [ 0.72187497 -1.03359377  0.86132805]
Iteración 4: [ 1.16132806 -0.70400396  0.58666987]
Iteración 5: [ 0.88666988 -0.90999759  0.75833123]
Iteración 6: [ 1.05833124 -0.78125157  0.65104288]
Iteración 7: [ 0.95104289 -0.86171784  0.7180981 ]
Iteración 8: [ 1.01809811 -0.81142642  0.67618859]
Iteración 9: [ 0.9761886  -0.84285856  0.70238203]
Iteración 10: [ 1.00238204 -0.82321347  0.68601113]
Iteración 11: [ 0.98601114 -0.83549165  0.69624294]
Iteración 12: [ 0.99624295 -0.82781779  0.68984806]
Iteración 13: [ 0.98984807 -0.83261395  0.69384486]
Iteración 14: [ 0.99384487 -0.82961635  0.69134686]
Iteración 15: [ 0.99134687 -0.83148985  0.69290811]
Iteración 16: [ 0.99290812 -0.83031891  0.69193233]
Iteración 17: [ 0.99193234 -0.83105075  0.69254219]
Iteración 18: [ 0.9925422  -0.83059335  0.69216103]
Iteración 19: [ 0.99216104 -0.83087922  0.69239925]
Iteración 20: [ 0.99239927 -0.83070055  0.69225036]
Iteración 21: [ 0.99225037 -0.83081222  0.69234342]
Iteración 22: [ 0.99234343 -0.83074243  0.69228526]
Iteración 23: [ 0.99228527 -0.83078605  0.69232161]
Iteración 24: [ 0.99232162 -0.83075879  0.69229889]
Iteración 25: [ 0.9922989  -0.83077583  0.69231309]
Iteración 26: [ 0.9923131  -0.83076518  0.69230421]
Convergencia alcanzada en iteración 27
Solución aproximada con sistema cambiado: [ 0.9923131  -0.83076518  0.69230421]
```

***8.Un cable coaxial está formado por un conductor interno de 0.1 pulgadas**

cuadradas y un conductor externo de 0.5 pulgadas cuadradas. El potencial en un punto en la sección transversal del cable se describe mediante la ecuación de Laplace. a. ¿La matriz es estrictamente diagonalmente dominante? b. Resuelva el sistema lineal usando el método de Jacobi con x(0) = 0 y TOL = 10-2. c. Repita la parte b) mediante el método de Gauss-Siedel.

```
import numpy as np

def jacobi(A, b, x0, tol, max_iter):
    n = len(A)
    x = x0.copy()
    for it in range(max_iter):
        x_new = np.zeros_like(x)
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            print(f"Convergencia alcanzada en iteración {it + 1}")
            break
        x = x_new
        print(f"Iteración {it + 1}: {x}")
    return x

def gauss_seidel(A, b, x0, tol, max_iter):
    n = len(A)
    x = x0.copy()
    for it in range(max_iter):
        x_new = x.copy()
        for i in range(n):
            s1 = sum(A[i][j] * x_new[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, n))
            x_new[i] = (b[i] - s1 - s2) / A[i][i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            print(f"Convergencia alcanzada en iteración {it + 1}")
            break
        x = x_new
        print(f"Iteración {it + 1}: {x}")
    return x

# Datos de la matriz y el vector b proporcionados en el problema
A = np.array([
```

```python
    [ 4, -1,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0],
    [-1,  4, -1,  0,  0,  0,  0, -1,  0,  0,  0,  0],
    [ 0, -1,  4, -1,  0,  0,  0,  0, -1,  0,  0,  0],
    [ 0,  0, -1,  4, -1,  0,  0,  0,  0, -1,  0,  0],
    [ 0,  0,  0, -1,  4, -1,  0,  0,  0,  0, -1,  0],
    [ 0,  0,  0,  0, -1,  4,  0,  0,  0,  0,  0, -1],
    [-1,  0,  0,  0,  0,  0,  4, -1,  0,  0,  0,  0],
    [ 0, -1,  0,  0,  0,  0, -1,  4, -1,  0,  0,  0],
    [ 0,  0, -1,  0,  0,  0,  0, -1,  4, -1,  0,  0],
    [ 0,  0,  0, -1,  0,  0,  0,  0, -1,  4, -1,  0],
    [ 0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  4, -1],
    [ 0,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  4]
], dtype=np.float32)

b = np.array([220, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 220], dtype=np.float3
x0 = np.zeros(len(b), dtype=np.float32)
tol = 1e-5
max_iter = 300

print("Método de Jacobi:")
solution_jacobi = jacobi(A, b, x0, tol, max_iter)
print(f"Solución aproximada con Jacobi: {solution_jacobi}\n")

print("Método de Gauss-Seidel:")
solution_gauss_seidel = gauss_seidel(A, b, x0, tol, max_iter)
print(f"Solución aproximada con Gauss-Seidel: {solution_gauss_seidel}")
```

```
Método de Jacobi:
Iteración 1: [55.  27.5 27.5 27.5 27.5 27.5 27.5 27.5 27.5 27.5 27.5 55. ]
Iteración 2: [68.75  55.    48.125 48.125 48.125 48.125 48.125 48.125 48.125 48.125
 55.    68.75 ]
Iteración 3: [80.78125 68.75    65.3125  63.59375 65.3125  56.71875 56.71875 65.3125
 63.59375 65.3125  68.75    80.78125]
Iteración 4: [86.36719  80.35156  76.484375 76.484375 74.765625 64.02344  64.02344
 74.765625 76.484375 76.484375 80.35156  86.36719 ]
Iteración 5: [91.09375 86.9043  85.83008 84.43359 82.71484 67.7832  67.7832  82.71484
 84.43359 85.83008 86.9043  91.09375]
Iteración 6: [93.671875 92.40967  91.44287  91.09375  87.28027  70.95215  70.95215
 87.28027  91.09375  91.44287  92.40967  93.671875]
Iteración 7: [95.840454 95.598755 96.14929  95.041504 91.11389  72.73804  72.73804
 91.11389  95.041504 96.14929  95.598755 95.840454]
Iteración 8: [97.0842   98.27591  98.92044  98.35312  93.344574 74.23859  74.23859
```

```
 93.344574 98.35312  98.92044  98.27591  97.0842  ]
Iteración 9: [ 98.128624  99.8373   101.24554  100.296364  95.2169    75.10719
  75.10719   95.2169   100.296364 101.24554   99.8373    98.128624]
Iteración 10: [ 98.73612  101.147766 102.607506 101.926994  96.31021   75.83638
  75.83638   96.31021  101.926994 102.607506 101.147766  98.73612 ]
Iteración 11: [ 99.24603 101.91346 103.75044 102.8813   97.22778  76.26158  76.26158
  97.22778 102.8813  103.75044 101.91346  99.24603]
Iteración 12: [ 99.54376  102.55606  104.419014 103.68217   97.764084  76.618454
  76.618454  97.764084 103.68217  104.419014 102.55606   99.54376 ]
Iteración 13: [ 99.793625 102.93172  104.9801   104.15053   98.21417   76.826965
  76.826965  98.21417  104.15053  104.9801   102.93172   99.793625]
Iteración 14: [ 99.93967  103.24698  105.3082   104.543594  98.4773    77.00195
  77.00195   98.4773   104.543594 105.3082   103.24698   99.93967 ]
Iteración 15: [100.06223  103.43129  105.58354  104.77342   98.698135  77.10424
  77.10424   98.698135 104.77342  105.58354  103.43129  100.06223 ]
Iteración 16: [100.13388  103.585976 105.74454  104.96631   98.82724   77.190094
  77.190094  98.82724  104.96631  105.74454  103.585976 100.13388 ]
Iteración 17: [100.194016 103.676414 105.87965  105.07908   98.93559   77.24028
  77.24028   98.93559  105.07908  105.87965  103.676414 100.194016]
Iteración 18: [100.22917 103.75231 105.95864 105.17372  98.99895  77.2824   77.2824
  98.99895 105.17372 105.95864 103.75231 100.22917]
Iteración 19: [100.25868 103.79669 106.02494 105.22906  99.05211  77.30703  77.30703
  99.05211 105.22906 106.02494 103.79669 100.25868]
Iteración 20: [100.27593  103.83393  106.063705 105.2755    99.08319   77.3277
  77.3277    99.08319  105.2755   106.063705 103.83393  100.27593 ]
Iteración 21: [100.290405 103.855705 106.09623  105.30265   99.10928   77.33978
  77.33978   99.10928  105.30265  106.09623  103.855705 100.290405]
Iteración 22: [100.298874 103.87398  106.11525  105.32544   99.124535  77.34992
  77.34992   99.124535 105.32544  106.11525  103.87398  100.298874]
Iteración 23: [100.30598 103.88467 106.13121 105.33876  99.13734  77.35585  77.35585
  99.13734 105.33876 106.13121 103.88467 100.30598]
Iteración 24: [100.31013  103.89363  106.14055  105.34994   99.14482   77.360825
  77.360825  99.14482  105.34994  106.14055  103.89363  100.31013 ]
Iteración 25: [100.313614 103.89887  106.14838  105.356476  99.1511    77.36374
  77.36374   99.1511   105.356476 106.14838  103.89887  100.313614]
Iteración 26: [100.31565  103.903275 106.152954 105.36196   99.15477   77.36618
  77.36618   99.15477  105.36196  106.152954 103.903275 100.31565 ]
Iteración 27: [100.31737  103.905846 106.1568   105.36517   99.15785   77.36761
  77.36761   99.15785  105.36517  106.1568   103.905846 100.31737 ]
Iteración 28: [100.31836  103.908005 106.15905  105.36786   99.15965   77.368805
  77.368805  99.15965  105.36786  106.15905  103.908005 100.31836 ]
Iteración 29: [100.3192   103.90926  106.160934 105.36944   99.16116   77.36951
  77.36951   99.16116  105.36944  106.160934 103.90926  100.3192  ]
```

```
Iteración 30: [100.319695 103.910324 106.16203  105.37076   99.16205   77.37009
  77.37009   99.16205  105.37076  106.16203  103.910324 100.319695]
Iteración 31: [100.3201   103.91094  106.162964 105.37153   99.162796  77.37044
  77.37044   99.162796 105.37153  106.162964 103.91094  100.3201  ]
Iteración 32: [100.32034 103.91147 106.1635  105.37218  99.16322  77.37073  77.37073
  99.16322 105.37218 106.1635  103.91147 100.32034]
Iteración 33: [100.32055  103.911766 106.163956 105.37256   99.16359   77.370895
  77.370895  99.16359  105.37256  106.163956 103.911766 100.32055 ]
Iteración 34: [100.32066  103.912025 106.16422  105.37288   99.1638    77.37103
  77.37103   99.1638   105.37288  106.16422  103.912025 100.32066 ]
Iteración 35: [100.32076  103.91217  106.164444 105.37306   99.16399   77.37112
  77.37112   99.16399  105.37306  106.164444 103.91217  100.32076 ]
Iteración 36: [100.32082  103.9123   106.16457  105.373215  99.164085  77.371185
  77.371185  99.164085 105.373215 106.16457  103.9123   100.32082 ]
Iteración 37: [100.32087 103.91237 106.16468 105.37331  99.16418  77.37123  77.37123
  99.16418 105.37331 106.16468 103.91237 100.32087]
Iteración 38: [100.3209  103.91243 106.16475 105.37338  99.16423  77.37126  77.37126
  99.16423 105.37338 106.16475 103.91243 100.3209 ]
Iteración 39: [100.32092  103.91247  106.164795 105.37343   99.16427   77.371284
  77.371284  99.16427  105.37343  106.164795 103.91247  100.32092 ]
Iteración 40: [100.32094 103.9125  106.16483 105.37347  99.16429  77.3713   77.3713
  99.16429 105.37347 106.16483 103.9125  100.32094]
Iteración 41: [100.32095  103.91251  106.164856 105.37349   99.164314  77.37131
  77.37131   99.164314 105.37349  106.164856 103.91251  100.32095 ]
Iteración 42: [100.32095  103.91253  106.16487  105.373505  99.16433   77.371315
  77.371315  99.16433  105.373505 106.16487  103.91253  100.32095 ]
Iteración 43: [100.32096 103.91254 106.16489 105.37352  99.16434  77.37132  77.37132
  99.16434 105.37352 106.16489 103.91254 100.32096]
Convergencia alcanzada en iteración 44
Solución aproximada con Jacobi: [100.32096 103.91254 106.16489 105.37352  99.16434  77.37132
  99.16434 105.37352 106.16489 103.91254 100.32096]

Método de Gauss-Seidel:
Iteración 1: [55.        41.25      37.8125    36.953125 36.73828  36.68457  41.25
 48.125    48.984375 48.984375 48.930664 76.40381 ]
Iteración 2: [75.625    67.890625 65.95703  65.41992  65.25879  62.91565  58.4375
 71.328125 74.06738  74.60449  81.56677  91.120605]
Iteración 3: [86.58203  83.4668   83.238525 83.27545  84.43947  71.390015 66.97754
 83.62793  87.86774  90.67749  94.05939  96.36235 ]
Iteración 4: [92.611084 92.369385 93.37814  94.62378  92.518295 74.72016  71.55975
 90.44922  96.12621  98.70235  99.39575  98.52898 ]
Iteración 5: [ 95.982285  97.45241   99.5506   100.19281   96.07718   76.151535
  74.10788   94.42162  100.66864  102.5643   101.79262   99.48604 ]
```

```
Iteración 6: [ 97.890076 100.465576 102.83176  102.86831   97.70312   76.79729
  75.57793   96.67804  103.018524 104.41986  102.90225   99.92488 ]
Iteración 7: [ 99.01088  102.13017  104.50425  104.15681   98.46409   77.097244
  76.422226  97.89273  104.20421  105.31582  103.42619  100.13086 ]
Iteración 8: [ 99.6381   103.00877  105.342445 104.78059   98.826004  77.23921
  76.882706  98.523926 104.79555  105.75058  103.676865 100.22902 ]
Iteración 9: [ 99.97287  103.45981  105.75899  105.08389   98.99999   77.30725
  77.1242    98.84489  105.088615 105.96234  103.79784  100.276276]
Iteración 10: [100.146    103.68747  105.965    105.231834  99.08423   77.340126
  77.24773   99.00595  105.23332  106.06575  103.85657  100.29917 ]
Iteración 11: [100.233795 103.801186 106.06659  105.30414   99.125206  77.356094
  77.30994   99.08611  105.30461  106.11633  103.88518  100.31032 ]
Iteración 12: [100.27778  103.85762  106.11659  105.33953   99.1452    77.36388
  77.34097   99.1258   105.33968  106.1411   103.899155 100.31576 ]
Iteración 13: [100.29965  103.88551  106.14118  105.35687   99.154976  77.36768
  77.35636   99.145386 105.35692  106.15324  103.90599  100.31842 ]
Iteración 14: [100.31047  103.89926  106.15326  105.36537   99.15976   77.369545
  77.36397   99.15504  105.36539  106.15919  103.90934  100.31972 ]
Iteración 15: [100.31581  103.90603  106.159195 105.36954   99.16211   77.37045
  77.367714  99.15978  105.36954  106.1621   103.91098  100.32036 ]
Iteración 16: [100.318436 103.909355 106.16211  105.37158   99.16325   77.3709
  77.36955   99.16211  105.37158  106.163536 103.91179  100.32067 ]
Iteración 17: [100.319725 103.91099  106.163536 105.37258   99.16382   77.371124
  77.37046   99.16325  105.37258  106.16424  103.912186 100.32083 ]
Iteración 18: [100.32036 103.91179 106.16424 105.37308  99.16409  77.37123  77.3709
  99.16382 105.37308 106.16458 103.91238 100.3209 ]
Iteración 19: [100.32067  103.912186 106.16458  105.373314  99.16423   77.371284
  77.371124  99.16409  105.373314 106.16475  103.91247  100.32094 ]
Iteración 20: [100.32083 103.91238 106.16475 105.37343  99.16429  77.37131  77.37123
  99.16423 105.37343 106.16483 103.91251 100.32095]
Iteración 21: [100.3209   103.91247  106.16483  105.37349   99.16433   77.37132
  77.371284  99.16429  105.37349  106.16487  103.91254  100.32097 ]
Iteración 22: [100.32094  103.91251  106.16487  105.37352   99.164345  77.37133
  77.37131   99.16433  105.37352  106.164894 103.91255  100.32097 ]
Iteración 23: [100.32095  103.91254  106.164894 105.373535  99.16435   77.37133
  77.37132   99.164345 105.373535 106.1649   103.91255  100.32097 ]
Iteración 24: [100.32097  103.91255  106.1649   105.373535  99.16435   77.37133
  77.37133   99.16435  105.373535 106.1649   103.91255  100.32097 ]
Convergencia alcanzada en iteración 25
Solución aproximada con Gauss-Seidel: [100.32097  103.91255  106.1649   105.373535  99.16435
  77.37133   99.16435  105.373535 106.1649   103.91255  100.32097 ]
```