Numerical Analysis II Hw

Kevin Ho

1) Let

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

a) use Gerschgorin-Taussky theorem to find the range of eigenvalues of A.

R_1, center $= 2$ $r = 1$ so $\lambda = [1, 3]$

R_2, center $= 2$ $r = 2$ $\lambda = [0, 4]$

R_3, center $= 2$ $r = 1$ $\lambda = [1, 3]$

So the range of eigenvalues of A is $[0, 4]$

b) is A nonsingular?

So let's find the determinant of the matrix.

$2(4-1) - (-1)(-2) = 6 - 2 = 4$ ← non-zero thus non-singular. □

2)

$$A = \begin{bmatrix} 2 & -2 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \quad b = \begin{bmatrix} -2 \\ -1 \\ -2 \\ 14 \end{bmatrix}$$

a) is A irreducible diagonally dominant?

So diagonally dominant Thus diagonally dominant.

diag $\geq$

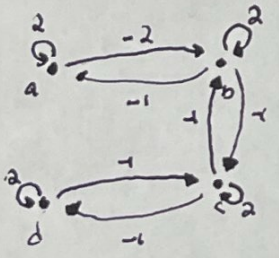
R_1 $2 \geq 2$

R_2 $2 \geq 2$

R_3 $2 \geq 2$

R_4 $2 > 1$

Now to show irreducibe,



Every node is strongly connected to every other node from a graph representation so thus irreducible.

b-d and 3. are shown in output of code

Numerical Analysis 10.2 and 10.3

OUTPUTS:
/usr/local/bin/python3.11 /Users/kevinho/PycharmProjects/Homeworks/Numerical Analysis/Homework10,23.py
=== 10.2 (b): Jacobi Iterations ===
Jacobi Method:
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1. -1. -1. 7.]
ITERATION 2: [-2. -2.   2.   6.5]
ITERATION 3: [-3.   -1.   1.25 8.]
ITERATION 4: [-2.   -1.875 2.5   7.625]
ITERATION 5: [-2.875 -0.75   1.875 8.25]
ITERATION 6: [-1.75   -1.5   2.75   7.9375]
ITERATION 7: [-2.5   -0.5   2.21875 8.375]
ITERATION 8: [-1.5   -1.140625 2.9375   8.109375]
ITERATION 9: [-2.140625 -0.28125   2.484375 8.46875]
ITERATION 10: [-1.28125   -0.828125   3.09375   8.2421875]
Jacobi Final Approximation: [-1.28125   -0.828125   3.09375   8.2421875]

=== 10.2 (b): Gauss-Seidel Iterations ===
Gauss-Seidel Method:
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.   -1.5 -1.75 6.125]
ITERATION 2: [-2.5   -3.125 0.5   7.25]
ITERATION 3: [-4.125   -2.8125   1.21875 7.609375]
ITERATION 4: [-3.8125   -2.296875 1.65625   7.828125]
ITERATION 5: [-3.296875   -1.8203125   2.00390625 8.00195312]
ITERATION 6: [-2.8203125 -1.40820312 2.296875   8.1484375]
ITERATION 7: [-2.40820312 -1.05566406 2.54638672 8.27319336]
ITERATION 8: [-2.05566406 -0.75463867 2.75927734 8.37963867]
ITERATION 9: [-1.75463867 -0.49768066 2.940979   8.4704895]
ITERATION 10: [-1.49768066 -0.27835083 3.09606934 8.54803467]
Gauss-Seidel Final Approximation: [-1.49768066 -0.27835083 3.09606934 8.54803467]

=== 10.2 (c): SOR - Finding Best Omega ===
SOR Method (omega = 1.0):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.   -1.5 -1.75 6.125]
ITERATION 2: [-2.5   -3.125 0.5   7.25]
ITERATION 3: [-4.125   -2.8125   1.21875 7.609375]
ITERATION 4: [-3.8125   -2.296875 1.65625   7.828125]
ITERATION 5: [-3.296875   -1.8203125   2.00390625 8.00195312]

ITERATION 6: [-2.8203125 -1.40820312 2.296875   8.1484375]
ITERATION 7: [-2.40820312 -1.05566406 2.54638672 8.27319336]
ITERATION 8: [-2.05566406 -0.75463867 2.75927734 8.37963867]
ITERATION 9: [-1.75463867 -0.49768066 2.940979   8.4704895]
ITERATION 10: [-1.49768066 -0.27835083 3.09606934 8.54803467]
Omega 1.00: Approximation = [-1.49768066 -0.27835083 3.09606934 8.54803467], Error = 1.4976806640625
SOR Method (omega = 1.05):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.05   -1.60125   -1.89065625 6.35740547]
ITERATION 2: [-2.6788125 -3.36890859 0.61349367 7.3542139]
ITERATION 3: [-4.4534134 -2.89751243 1.25909359 7.64331344]
ITERATION 4: [-3.86971738 -2.27570187 1.7050414   7.86298106]
ITERATION 5: [-3.24600109 -1.74521875 2.07657315 8.04705185]
ITERATION 6: [-2.72017963 -1.30063247 2.38804152 8.2013692]
ITERATION 7: [-2.27965511 -0.92806551 2.64908236 8.33069978]
ITERATION 8: [-1.91048603 -0.61583365 2.8678506   8.43908658]
ITERATION 9: [-1.60110103 -0.35416479 3.05119141 8.52992116]
ITERATION 10: [-1.34181798 -0.13487071 3.20484191 8.60604595]
Omega 1.05: Approximation = [-1.34181798 -0.13487071 3.20484191 8.60604595], Error = 1.341817981961054
SOR Method (omega = 1.1):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.1   -1.705   -2.03775   6.5792375]
ITERATION 2: [-2.8655   -3.6262875   0.7278975   7.44241988]
ITERATION 3: [-4.80236625 -2.97832906 1.2824602   7.66111112]
ITERATION 4: [-3.89592534 -2.23957292 1.75359999 7.89836888]
ITERATION 5: [-3.17393768 -1.65722844 2.15726724 8.0966601]
ITERATION 6: [-2.60555752 -1.18083681 2.48797609 8.25872084]
ITERATION 7: [-2.13836473 -0.78963008 2.75920231 8.39168919]
ITERATION 8: [-1.75475661 -0.46859186 2.9817833   8.5008119]
ITERATION 9: [-1.43997538 -0.20514646 3.16443766 8.59035952]
ITERATION 10: [-1.18166357 0.0110404   3.31432619 8.66384345]
Omega 1.10: Approximation = [-1.18166357 0.0110404   3.31432619 8.66384345], Error = 1.1816635670435645
SOR Method (omega = 1.1500000000000001):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.15   -1.81125   -2.19146875 6.78990547]
ITERATION 2: [-3.0604375 -3.89815859 0.84147477 7.51536217]
ITERATION 3: [-5.17381676 -3.05637286 1.28769764 7.66312182]
ITERATION 4: [-3.88875627 -2.18715278 1.80552755 7.93871007]
ITERATION 5: [-3.08191226 -1.55584829 2.24931639 8.15255041]
ITERATION 6: [-2.4769387 -1.04750558 2.59800332 8.32096935]
ITERATION 7: [-1.98309062 -0.63929936 2.87725975 8.45627895]

ITERATION 8: [-1.58773067 -0.31262588 3.10101156 8.5646398]
ITERATION 9: [-1.27136016 -0.05105657 3.28015863 8.65139524]
ITERATION 10: [-1.01801103 0.15839336 3.42360465 8.72086339]
Omega 1.15: Approximation = [-1.01801103 0.15839336 3.42360465 8.72086339], Error = 1.018011026378016
SOR Method (omega = 1.2000000000000002):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.2   -1.92 -2.352 6.9888]
ITERATION 2: [-3.264   -4.1856   0.95232 7.573632]
ITERATION 3: [-5.56992   -3.13344   1.2736512 7.64946432]
ITERATION 4: [-3.846144 -2.11680768 1.86486374 7.98902538]
ITERATION 5: [-2.97094042 -1.44028447 2.3562718   8.215958]
ITERATION 6: [-2.33415328 -0.89867199 2.71911725 8.38827875]
ITERATION 7: [-1.81157574 -0.4757407   3.00369938 8.52456388]
ITERATION 8: [-1.40857369 -0.14777644 3.22533259 8.63028678]
ITERATION 9: [-1.095617   0.10738464 3.39753633 8.71246445]
ITERATION 10: [-0.85201503 0.30583585 3.53147291 8.77639086]
Omega 1.20: Approximation = [-0.85201503 0.30583585 3.53147291 8.77639086], Error = 0.8520150298059073
SOR Method (omega = 1.2500000000000002):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.25   -2.03125   -2.51953125 7.17529297]
ITERATION 2: [-3.4765625 -4.48974609 1.05834961 7.61764526]
ITERATION 3: [-5.99304199 -3.21174622 1.2390995   7.62002587]
ITERATION 4: [-3.76642227 -2.02664018 1.93609118 8.05505052]
ITERATION 5: [-2.84169465 -1.30934212 2.48204495 8.28751547]
ITERATION 6: [-2.17625399 -0.73154512 2.85197023 8.46060253]
ITERATION 7: [-1.6203679 -0.29736226 3.13903261 8.59674475]
ITERATION 8: [-1.21661086 0.02585416 3.35436617 8.69729267]
ITERATION 9: [-0.91352959 0.26905932 3.51537845 8.77278837]
ITERATION 10: [-0.68529345 0.4515383   3.63635955 8.82952763]
Omega 1.25: Approximation = [-0.68529345 0.4515383   3.63635955 8.82952763], Error = 0.6852934506568094
SOR Method (omega = 1.3000000000000003):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.3   -2.145   -2.69425   7.3487375]
ITERATION 2: [-3.6985   -4.8117875   1.1572925   7.64761887]
ITERATION 3: [-6.44577375 -3.29397656 1.18267975 7.57445618]
ITERATION 4: [-3.64843741 -1.91454951 2.02413541 8.14335116]
ITERATION 5: [-2.69438314 -1.16129617 2.63109512 8.36720648]
ITERATION 6: [-2.00137008 -0.54228987 2.99686726 8.53780177]
ITERATION 7: [-1.40456581 -0.1023171   3.28400486 8.67326263]
ITERATION 8: [-1.01164248 0.20773068 3.48744419 8.76485993]
ITERATION 9: [-0.72645738 0.43232223 3.63193515 8.83129987]

ITERATION 10: [-0.52004389 0.59303265 3.73623559 8.87916317]
Omega 1.30: Approximation = [-0.52004389 0.59303265 3.73623559 8.87916317], Error = 0.5200438934037921
SOR Method (omega = 1.3500000000000003):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.35   -2.26125   -2.87634375 7.50846797]
ITERATION 2: [-3.9301875 -5.15297109 1.2466807   7.66354569]
ITERATION 3: [-6.93094535 -3.38333875 1.10280143 7.51214998]
ITERATION 4: [-3.49167645 -1.77832207 2.13435334 8.26143601]
ITERATION 5: [-2.52864804 -0.9937362   2.8086737   8.45435215]
ITERATION 6: [-1.80651706 -0.32573659 3.1537797   8.61977805]
ITERATION 7: [-1.15746343 0.11152129 3.43980416 8.75494549]
ITERATION 8: [-0.79433406 0.39665987 3.62340216 8.83156554]
ITERATION 9: [-0.53649226 0.59483323 3.74462841 8.88657624]
ITERATION 10: [-0.35920285 0.72697062 3.82852419 8.92395214]
Omega 1.35: Approximation = [-0.35920285 0.72697062 3.82852419 8.92395214], Error = 0.35920284945530134
SOR Method (omega = 1.4000000000000004):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.4   -2.38   -3.066   7.6538]
ITERATION 2: [-4.172   -5.5146   1.32384   7.665168]
ITERATION 3: [-7.45164   -3.48362   0.9975476   7.43221612]
ITERATION 4: [-3.296412   -1.61575708 2.27250229 8.41786515]
ITERATION 5: [-2.34349511 -0.80339214 3.02113019 8.54764507]
ITERATION 6: [-1.58735096 -0.07499768 3.3224011   8.70662274]
ITERATION 7: [-0.87005637 0.34664038 3.60832375 8.84317753]
ITERATION 8: [-0.56668092 0.59049383 3.76024045 8.8948973]
ITERATION 9: [-0.34663627 0.75332539 3.84965971 8.93680287]
ITERATION 10: [-0.20668994 0.84874868 3.9100222   8.96229439]
Omega 1.40: Approximation = [-0.20668994 0.84874868 3.9100222   8.96229439], Error = 0.20668994185743567
SOR Method (omega = 1.4500000000000004):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.45   -2.50125   -3.26340625 7.78403047]
ITERATION 2: [-4.4243125 -5.89803359 1.38588055 7.65194969]
ITERATION 3: [-8.01120809 -3.59924735 0.86456295 7.33343078]
ITERATION 4: [-3.06386502 -1.42483269 2.44468029 8.62234936]
ITERATION 5: [-2.13726815 -0.58595149 3.27628233 8.64524748]
ITERATION 6: [-1.33785899 0.21903509 3.50227781 8.79879005]
ITERATION 7: [-0.53036258 0.60607276 3.79250052 8.94010735]
ITERATION 8: [-0.33253134 0.78574491 3.89461766 8.95054949]
ITERATION 9: [-0.16103077 0.90326528 3.94143777 8.97979511]
ITERATION 10: [-0.06780149 0.95191692 3.97684423 8.99230427]

Omega 1.45: Approximation = [-0.06780149 0.95191692 3.97684423 8.99230427], Error = 0.06780149370554582
SOR Method (omega = 1.5000000000000004):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.5   -2.625   -3.46875   7.8984375]
ITERATION 2: [-4.6875   -6.3046875   1.4296875   7.62304688]
ITERATION 3: [-8.61328125 -3.73535156 0.70092773 7.21417236]
ITERATION 4: [-2.79638672 -1.20391846 2.65722656 8.88583374]
ITERATION 5: [-1.90768433 -0.33588409 3.58384895 8.74496984]
ITERATION 6: [-1.04998398 0.56834078 3.69305849 8.89730895]
ITERATION 7: [-0.12249684 0.89375085 3.9967656   9.04891973]
ITERATION 8: [-0.09812531 0.97710479 4.02113559 8.99139183]
ITERATION 9: [0.01471985 1.03833918 4.01173046 9.01310193]
ITERATION 10: [0.05014885 1.02723989 4.02439114 9.01174239]
Omega 1.50: Approximation = [0.05014885 1.02723989 4.02439114 9.01174239], Error = 0.05014884774573197
SOR Method (omega = 1.5500000000000005):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.55   -2.75125   -3.68221875 7.99628047]
ITERATION 2: [-4.9619375 -6.73603359 1.45191164 7.57727726]
ITERATION 3: [-9.26178645 -3.8978345   0.50301674 7.07233548]
ITERATION 4: [-2.49766093 -0.95204027 2.91656958 9.22055691]
ITERATION 5: [-1.65194891 -0.04629683 3.95593829 8.84454588]
ITERATION 6: [-0.71318819 0.98859459 3.8949178   9.00406106]
ITERATION 7: [0.37457511 1.21512998 4.22766827 9.17420933]
ITERATION 8: [0.12743516 1.15688367 4.13137952 9.006004]
ITERATION 9: [0.17308035 1.14967038 4.04838891 9.0341992]
ITERATION 10: [0.1367949 1.06119874 4.04731951 9.01786306]
Omega 1.55: Approximation = [0.1367949 1.06119874 4.04731951 9.01786306], Error = 0.1367949008041693
SOR Method (omega = 1.6000000000000005):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-1.6   -2.88   -3.904   8.0768]
ITERATION 2: [-5.248   -7.1936   1.44896 7.513088]
ITERATION 3: [-9.96096   -4.09344   0.2663424   6.90522112]
ITERATION 4: [-2.172928   -0.66920448 3.22900787 9.64007363]
ITERATION 5: [-1.36697037 0.29115269 4.40757633 8.94201689]
ITERATION 6: [-0.31397347 1.50019067 4.10922025 9.12216607]
ITERATION 7: [0.98868916 1.57821312 4.4947712 9.32251732]
ITERATION 8: [0.3319275 1.31443109 4.21269601 8.97664641]
ITERATION 9: [0.30393324 1.22464475 4.03341532 9.04074441]
ITERATION 10: [0.17707165 1.03360273 4.03942852 9.00709617]
Omega 1.60: Approximation = [0.17707165 1.03360273 4.03942852 9.00709617], Error = 0.17707164851870016

Best Omega (smallest error after 10 iterations): 1.5000000000000004 with error: 0.05014884774573197

=== 10.2 (d): Spectral Radii ===
Spectral Radius of Jacobi Iteration Matrix: 0.9238795325112874
Spectral Radius of Gauss-Seidel Iteration Matrix: 0.853553390593274
Spectral Radius of SOR Iteration Matrix (with omega = 1.5000000000000004): 0.5000000000000008

=== 10.3 (b): Solving with GMRES ===
GMRES Solution: [1.53571509e-15 1.00000000e+00 4.00000000e+00 9.00000000e+00]
GMRES Exit Code (0 means successful): 0
GMRES Residual Norm: 1.831026719408895e-15

=== 10.3 (b): Solving with CG on Symmetrized System ===
Conjugate Gradient Method (CG on symmetrized system):
ITERATION 0: [0. 0. 0. 0.]
ITERATION 1: [-0.22341651 0.22341651 -1.78733205 3.3512476]
ITERATION 2: [3.06053421e-03 -1.21938243e+00 -1.07869140e+00 4.96081234e+00]
ITERATION 3: [-2.82851829 -1.69577053 1.90670789 7.85975849]
ITERATION 4: [3.31290551e-13 1.00000000e+00 4.00000000e+00 9.00000000e+00]

CG Final Approximation (Symmetrized System): [3.31290551e-13 1.00000000e+00 4.00000000e+00 9.00000000e+00]
CG Residual Norm: 6.960542472835948e-12

=== Comparison ===
Norm of difference between GMRES and CG solutions: 4.70382236250951e-13
GMRES and CG produce nearly identical results but does it in only one iteration of GMRES
So thus GMRES is better than the other methods to find the unique/exact solution.

Process finished with exit code 0

CODE

```
import numpy as np
from numpy.linalg import norm, eigvals, inv


# ----------------------------
# 10.2 - Iterative Methods
# ----------------------------

def jacobi_relaxation(A, b, x0, iterations=10, tolerance=1e-6):
    n = len(b)
    x = x0.copy()
    print("Jacobi Method:")
    print(f"ITERATION 0: {x}")
    for it in range(1, iterations + 1):
        x_new = np.zeros_like(x)
        for i in range(n):
            s = 0
            for j in range(n):
                if j != i:
                    s += A[i, j] * x[j]
            x_new[i] = (b[i] - s) / A[i, i]
        print(f"ITERATION {it}: {x_new}")
        if norm(x_new - x, ord=np.inf) < tolerance:
            return x_new
        x = x_new.copy()
    return x


def gauss_seidel(A, b, x0, iterations=10, tolerance=1e-6):
    n = len(b)
    x = x0.copy()
    print("Gauss-Seidel Method:")
    print(f"ITERATION 0: {x}")
    for it in range(1, iterations + 1):
        x_new = x.copy()
        for i in range(n):
            sum1 = sum(A[i, j] * x_new[j] for j in range(i))  # updated
values
            sum2 = sum(A[i, j] * x[j] for j in range(i + 1, n))  # old values
            x_new[i] = (b[i] - sum1 - sum2) / A[i, i]
        print(f"ITERATION {it}: {x_new}")
        if norm(x_new - x, ord=np.inf) < tolerance:
            return x_new
        x = x_new.copy()
    return x


def sor(A, b, x0, omega=1.25, iterations=10, tolerance=1e-6):
    n = len(b)
    x = x0.copy()
```

```
    print(f"SOR Method (omega = {omega}):")
    print(f"ITERATION 0: {x}")
    for it in range(1, iterations + 1):
        x_new = x.copy()
        for i in range(n):
            sum1 = sum(A[i, j] * x_new[j] for j in range(i))
            sum2 = sum(A[i, j] * x[j] for j in range(i + 1, n))
            x_new[i] = (1 - omega) * x[i] + (omega / A[i, i]) * (b[i] - sum1
- sum2)
        print(f"ITERATION {it}: {x_new}")
        if norm(x_new - x, ord=np.inf) < tolerance:
            return x_new
        x = x_new.copy()
    return x


# The given system (10.101)
A = np.array([[2, -2, 0, 0],
              [-1, 2, -1, 0],
              [0, -1, 2, -1],
              [0, 0, -1, 2]], dtype=float)
b = np.array([-2, -2, -2, 14], dtype=float)
x0 = np.zeros(len(b))

# Exact solution given
x_exact = np.array([0, 1, 4, 9], dtype=float)

# Run Jacobi and Gauss-Seidel for 10 iterations
print("=== 10.2 (b): Jacobi Iterations ===")
jacobi_solution = jacobi_relaxation(A, b, x0, iterations=10)
print("Jacobi Final Approximation:", jacobi_solution, "\n")

print("=== 10.2 (b): Gauss-Seidel Iterations ===")
gs_solution = gauss_seidel(A, b, x0, iterations=10)
print("Gauss-Seidel Final Approximation:", gs_solution, "\n")

# ----------------------------
# 10.2 (c): SOR Best Omega
# ----------------------------
print("=== 10.2 (c): SOR - Finding Best Omega ===")
omegas = np.arange(1.0, 1.61, 0.05)
errors = {}
for omega in omegas:
    # Run SOR for 10 iterations with given omega
    sor_sol = sor(A, b, x0, omega=omega, iterations=10)
    err = norm(sor_sol - x_exact, ord=np.inf)
    errors[omega] = err
    print(f"Omega {omega:.2f}: Approximation = {sor_sol}, Error = {err}")

best_omega = min(errors, key=errors.get)
print("\nBest Omega (smallest error after 10 iterations):", best_omega,
      "with error:", errors[best_omega], "\n")


# ----------------------------
# 10.2 (d): Spectral Radii of Iteration Matrices
# ----------------------------
```

```
def spectral_radius(T):
    return max(abs(eigvals(T)))


# Jacobi iteration matrix: T_J = -D^{-1}(L+U)
D = np.diag(np.diag(A))
L_plus_U = A - D
T_jacobi = -inv(D) @ L_plus_U
rho_jacobi = spectral_radius(T_jacobi)

# Gauss-Seidel iteration matrix: T_GS = (D - L)^{-1} U
# Here L is the strict lower-triangular part, U is the strict upper-
triangular part.
L = np.tril(A, k=-1)
U = np.triu(A, k=1)
T_gs = inv(D - L) @ U
rho_gs = spectral_radius(T_gs)

# SOR iteration matrix: T_SOR = (D - ωL)^{-1} [ (1-ω)D + ωU ]
omega_best = best_omega  # best omega from above
T_sor = inv(D - omega_best * L) @ ((1 - omega_best) * D + omega_best * U)
rho_sor = spectral_radius(T_sor)

print("=== 10.2 (d): Spectral Radii ===")
print("Spectral Radius of Jacobi Iteration Matrix:", rho_jacobi)
print("Spectral Radius of Gauss-Seidel Iteration Matrix:", rho_gs)
print("Spectral Radius of SOR Iteration Matrix (with omega =", omega_best,
"):", rho_sor, "\n")

import numpy as np
from scipy.sparse.linalg import gmres

# Define the original system A x = b
A = np.array([[2, -2,  0,  0],
              [-1, 2, -1,  0],
              [0, -1,  2, -1],
              [0,  0, -1,  2]], dtype=float)
b = np.array([-2, -2, -2, 14], dtype=float)

# Define the symmetrized system A_sym x = b_sym (for CG)
A_sym = A.T @ A
b_sym = A.T @ b

# --- 1. Solve using GMRES ---
print("=== 10.3 (b): Solving with GMRES ===")
x0_gmres = np.zeros(len(b))  # Initial guess
gmres_solution, exit_code = gmres(A, b, x0=x0_gmres, maxiter=1)

print("GMRES Solution:", gmres_solution)
print("GMRES Exit Code (0 means successful):", exit_code)
print("GMRES Residual Norm:", norm(A @ gmres_solution - b, ord=2), "\n")

# --- 2. Solve using CG on the symmetrized system ---
def conjugate_gradient(A, b, x0, iterations=10, tol=1e-6):
    x = x0.copy()
    r = b - A @ x
    p = r.copy()
```

```
    print("Conjugate Gradient Method (CG on symmetrized system):")
    print(f"ITERATION 0: {x}")
    for it in range(1, iterations+1):
        Ap = A @ p
        alpha = (r.T @ r) / (p.T @ Ap)
        x = x + alpha * p
        r_new = r - alpha * Ap
        print(f"ITERATION {it}: {x}")
        if norm(r_new, ord=2) < tol:
            return x
        beta = (r_new.T @ r_new) / (r.T @ r)
        p = r_new + beta * p
        r = r_new
    return x

print("=== 10.3 (b): Solving with CG on Symmetrized System ===")
x0_cg = np.zeros(A_sym.shape[0])
cg_solution = conjugate_gradient(A_sym, b_sym, x0_cg, iterations=10)

print("\nCG Final Approximation (Symmetrized System):", cg_solution)
print("CG Residual Norm:", norm(A_sym @ cg_solution - b_sym, ord=2), "\n")

# --- 3. Compare Results ---
gmres_error = norm(gmres_solution - cg_solution, ord=2)
print("=== Comparison ===")
print("Norm of difference between GMRES and CG solutions:", gmres_error)


print("GMRES and CG produce nearly identical results but does it in only one
iteration of GMRES\n"
      "So thus GMRES is better than the other methods to find the
unique/exact solution.")
```