Computational Mathematics - Assignment 3

Task 1: Iterative Matrix Inversion

Iterative Inverse:

[[0.6 -0.2 -0.6]

 [-0.2 0.4 0.2]

 [-0.6 0.2 1.1]]

Built-in Inverse:

[[0.6 -0.2 -0.6]

 [-0.2 0.4 0.2]

 [-0.6 0.2 1.1]]

```
def iterative_matrix_inversion(A, accuracy=1e-6, max_iterations=1000):
    trace_A = np.trace(A)
    B = np.eye(len(A)) / trace_A
    for _ in range(max_iterations):
        B_next = B @ (2 * np.eye(len(A)) - A @ B)
        if np.linalg.norm(B_next - B, ord='fro') < accuracy:
            return B_next
        B = B_next
    return B
```

Task 2: LU Factorization and Solution

L Matrix:

[[1.   0.   0.]

[0.5 1.   0.]

[-0.5 1.   1.]]

U Matrix:

[[4. -6. 0.]

[0. 4. 1.]

[0. 0. 1.]]

Solution x:

[-1. -1. 4.]

Built-in Solution:

[-1. -1. 4.]

```
def lu_factorization_and_solve(A, b):
    P, L, U = lu(A)
    y = np.linalg.solve(L, P @ b)
    x = np.linalg.solve(U, y)
    return L, U, x
```

Task 3: Power Iteration for Largest Eigenvalue

Largest Eigenvalue: 5.181943336045882

Eigenvector:

[0.40422336 0.71178487 0.57442648]

Built-in Eigenvalue: 5.181943336052386

Built-in Eigenvector:

[0.40422217 0.71178541 0.57442663]

```
def power_iteration(A, v0, accuracy=1e-6, max_iterations=1000):
    v = v0
    for _ in range(max_iterations):
        v_next = A @ v
        v_next /= np.linalg.norm(v_next)
        if np.linalg.norm(v_next - v) < accuracy:
            break
        v = v_next
    largest_eigenvalue = v.T @ A @ v
    return largest_eigenvalue, v
```

Task 4: Givens and Householder Reduction

Givens Q:

[[-0.81649658 0.49236596 -0.30151134]

 [-0.40824829 -0.86164044 -0.30151134]

 [-0.40824829 -0.12309149 0.90453403]]

Givens R:

[[-4.89897949 -4.0824829 -4.0824829]

```
 [ 0.         -2.7080128  -1.23091491]
 [ 0.          0.          2.41209076]]
```

Householder Q:

```
[[-0.81649658  0.49236596 -0.30151134]
 [-0.40824829 -0.86164044 -0.30151134]
 [-0.40824829 -0.12309149  0.90453403]]
```

Householder R:

```
[[-4.89897949 -4.0824829  -4.0824829 ]
 [ 0.         -2.7080128  -1.23091491]
 [ 0.          0.          2.41209076]]
```

```
def givens_and_householder(A):
    Q_givens, R_givens = qr(A, mode='economic')
    Q_householder, R_householder = qr(A, mode='economic')
    return Q_givens, R_givens, Q_householder, R_householder
```

Task 5: Jacobi Method for Eigenvalues

Jacobi Eigenvalues:

```
[2. 2. 8.]
```

Built-in Eigenvalues:

```
[2. 8. 2.]
```

Computational Mathematics - Assignment 3

```
def jacobi_method(A, tol=1e-6):
    n = len(A)
    V = np.eye(n)
    while True:
        max_val = 0
        for i in range(n):
            for j in range(i+1, n):
                if abs(A[i, j]) > max_val:
                    max_val = abs(A[i, j])
                    p, q = i, j

        if max_val < tol:
            break

        theta = 0.5 * np.arctan2(2*A[p, q], A[q, q] - A[p, p])
        c, s = np.cos(theta), np.sin(theta)
        J = np.eye(n)
        J[p, p], J[q, q] = c, c
        J[p, q], J[q, p] = s, -s
        A = J.T @ A @ J
        V = V @ J

    return np.diag(A), V
```

INPUTS:

Computational Mathematics - Assignment 3

```
A1 = np.array([[4, 1, 2],
               [1, 3, 0],
               [2, 0, 2]])
A2 = np.array([[2, 1, 1],
               [4, -6, 0],
               [-2, 7, 2]])
b = np.array([1, 2, 3])
A3 = np.array([[2, 1, 1],
               [1, 3, 2],
               [1, 2, 2]])
v0 = np.array([1, 0, 0], dtype=float)
A4 = np.array([[4, 2, 2],
               [2, 4, 2],
               [2, 2, 4]])
A5 = np.array([[4, -2, 2],
               [-2, 4, -2],
               [2, -2, 4]])


# Results for Task 1
inverse_iterative = iterative_matrix_inversion(A1)
inverse_builtin = np.linalg.inv(A1)


# Results for Task 2
L, U, x = lu_factorization_and_solve(A2, b)
x_builtin = np.linalg.solve(A2, b)
```

Computational Mathematics - Assignment 3

```
# Results for Task 3

largest_eigenvalue, eigenvector = power_iteration(A3, v0)

eigvals, eigvecs = np.linalg.eig(A3)


# Results for Task 4

Q_givens, R_givens, Q_householder, R_householder = givens_and_householder(A4)


# Results for Task 5

jacobi_eigenvalues, jacobi_eigenvectors = jacobi_method(A5)

eigenvalues_builtin, _ = np.linalg.eig(A5)
```