Solutions

March 22, 2024

1 Question 1

If we take

$$M = \begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix}$$

Then it is clear that

$$\begin{aligned} \begin{bmatrix} X_t \\ X_{t-1} \end{bmatrix} &= \begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_{t-1} \\ X_{t-2} \end{bmatrix} \\ &= \begin{bmatrix} 3X_{t-1} - 2X_{t-2} \\ X_{t-1} \end{bmatrix} \end{aligned}$$

The eigenvalues of M are $\lambda_1 = 1$ and $\lambda_2 = 2$, the corresponding eigenvectors are $\boldsymbol{v_1} = (1, 1)^T$ and $\boldsymbol{v_2} = (2, 1)^T$. We can now write M as:

$$\begin{aligned} M &= \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 1 & -1 \end{bmatrix} \end{aligned}$$

Next, using

$$\begin{bmatrix} X_t \\ X_{t-1} \end{bmatrix} = M^{t-1} \begin{bmatrix} X_1 \\ X_0 \end{bmatrix},$$

we get

$$\begin{aligned} \begin{bmatrix} X_t \\ X_{t-1} \end{bmatrix} &= \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1^{t-1} & 0 \\ 0 & 2^{t-1} \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2^{t-1} \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned}$$

Expanding this expression out gives

$$X_t = 2^t - 1.$$

2 Question 2

The element-wise expression is given by:

$$\vec{x}_j^{(i)} = \frac{1}{a_{jj}}\left(b_j - \sum_{k\neq j}^{n} a_{jk} x_k^{(i-1)}\right)$$

To see this, it is important to realise that for a diagonal matrix

$$D = \begin{pmatrix} a_{11} & 0 & \dots & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0\dots & \dots & \dots & a_{n,n} \end{pmatrix}$$

the inverse D^{-1} is:

$$D^{-1} = \begin{pmatrix} \frac{1}{a_{11}} & 0 & \dots & \dots & 0 \\ 0 & \frac{1}{a_{22}} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0\dots & \dots & \dots & \frac{1}{a_{nn}} \end{pmatrix}$$

Thus, $D^{-1}b$ is a vector $\left[\frac{b_i}{a_{ii}}\right]_{i=1\dots n}$. And

$$D^{-1}(L+R) = \begin{pmatrix} 0 & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} & \frac{a_{14}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{22}}{a_{22}} & 0 & \frac{a_{23}}{a_{22}} & \frac{a_{24}}{a_{22}} & \dots & \frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \frac{a_{n3}}{a_{nn}} & \frac{a_{n4}}{a_{nn}} & \frac{a_{n(n-1)}}{a_{nn}} & 0 \end{pmatrix}$$

The expression should follow. Try with a general case of 3×3 if necessary as an intermediate step.

[4]:

```python
import numpy as np
from matplotlib import pyplot as plt
from scipy.linalg import norm
```

[5]:

```python
def norm_2(x):
    return norm(x, ord=2)

def norm_inf(x):
    return norm(x, ord=np.inf)

x = np.array([1,2,3,4])

norm_2(x), norm_inf(x)
```

[5]:

```
(5.477225575051661, 4.0)
```

3 Random Linear System Generator

[6]:
```
# LOW = -20
# HIGH = 20
# def generate_problem(n):
#     A = np.random.randint(LOW, HIGH, (n,n))
#     A = A + 10*HIGH*np.eye(n)
#     x = np.random.randint(LOW, HIGH, (n))
#     b = A @ x
#     return A, b, x

def generate_problem(n):
    A = np.random.random(size=(n, n))
    D = np.diag(A.sum(axis=1))
    A = A+D
    x = np.random.random(n)
    b = np.dot(A, x)
    return A, b, x
```

[7]:
```
A = np.array([
        [ 10,  -1,   2,   0],
       [ -1, -11,  -1,   3],
       [  2,  -1,  10,  -1],
       [  0,   3,  -1,   8]])

b = np.array([  6,  25, -11,  15])
A, b
np.linalg.solve(A, b)
```

[7]:
```
array([ 0.67288008, -1.59357828, -1.16118953,  2.32744316])
```

4 Extract L, D, U

4.0.1 With Explicit Loop Over Indices

[8]:
```
L = np.zeros_like(A)
D = np.zeros_like(A)
R = np.zeros_like(A)

for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        if i == j:
            D[i, j] = A[i, j]
        elif i < j:
            R[i, j] = A[i, j]
        else:
            L[i, j] = A[i, j]
```

4.0.2 With Numpy Functions

[9]:
```
D = np.diag(np.diag(A))
L = np.tril(A) - D
U = np.triu(A) - D
L, D, U
```

[9]:
```
(array([[ 0,  0,  0,  0],
        [-1,  0,  0,  0],
        [ 2, -1,  0,  0],
        [ 0,  3, -1,  0]]),
 array([[ 10,   0,   0,   0],
        [  0, -11,   0,   0],
        [  0,   0,  10,   0],
        [  0,   0,   0,   8]]),
 array([[ 0, -1,  2,  0],
        [ 0,  0, -1,  3],
        [ 0,  0,  0, -1],
        [ 0,  0,  0,  0]]))
```

5 Question 3

The Jacobi iteration is given by:

$$\vec{x}_j^{(i)} = \frac{1}{a_{jj}}\left(b_j - \sum_{k \neq j}^{n} a_{jk} x_k^{(i-1)}\right)$$

Gauss-Seidel relies on the fact that the computation of $x_i^{(j+1)}$ is independent of any $x_l^{(j+1)}$. Thus, you can use more information the following way. To calculate $x_2^{(j+1)}$ you can already use $x_1^{(j+1)}$. To calculate $x_3^{(j+1)}$, you can use $x_2^{(j+1)}$ and $x_1^{(j+1)}$. And so on.

We can do this, by breaking out the sum in the Jacobi iteration. Therefore:

$$\vec{x}_j^{(i)} = \frac{1}{a_{jj}}\left(b_j - \sum_{k=1}^{j-1} a_{jk}x_k^{(i)} - \sum_{k=j+1}^{n} a_{jk}x_k^{(i-1)}\right)$$

6 Jacobi Iteration

[10]:
```
D_inv = np.linalg.inv(D)
```

[11]:
```
x = np.array([1, 1, 1, 1])
x
```

[11]:
```
array([1, 1, 1, 1])
```

[12]:
```
for _ in range(10):
    x = np.dot(D_inv, b) + np.dot(np.dot(-D_inv, L+R), x)
    print(x)
np.linalg.solve(A, b)
```

```
[ 0.5        -2.18181818 -1.1         1.625     ]
[ 0.60181818 -1.775      -1.25568182  2.55568182]
[ 0.67363636 -1.51628099 -1.14229545  2.38366477]
[ 0.67683099 -1.58003151 -1.14798889  2.30081844]
[ 0.67159463 -1.6023988  -1.16328751  2.3240132 ]
[ 0.67241762 -1.59420614 -1.16215749  2.33048861]
[ 0.67301088 -1.59261766 -1.16085528  2.32755762]
[ 0.67290929 -1.59358934 -1.16110818  2.32712471]
[ 0.6728627  -1.59367518 -1.16122832  2.32745748]
[ 0.67287815 -1.59356927 -1.16119431  2.32747465]
```

[12]:
```
array([ 0.67288008, -1.59357828, -1.16118953,  2.32744316])
```

[13]:
```
def jacobi(A, b, max_iter=1000, tol=1e-10, verbose=False):
    n = A.shape[0]
    x = np.ones(n)
    y = np.ones(n)
    i = 0
    def conv(x):
        try:
            return norm_2(np.dot(A, x) - b) < tol
        except ValueError: # did not converge due to an x component going to␣
↪infinity. cant take 2 norm
            return False

    while (i<max_iter) and (not conv(x)):
        if verbose: print('iteration {0}, x={1}'.format(i, x))
        for j in range(0, n):
```

```
            y[j] = b[j]
            for k in range (0, j):
                y[j]=y[j]-A[j, k]*x[k]
            for k in range(j+1, n):
                y[j]=y[j]-A[j, k]*x[k]
            y[j]= y[j]/A[j, j]
        x = np.copy(y)
        i+=1
    return x
```

[14]:
```
np.linalg.solve(A, b)
```

[14]: array([0.67288008, -1.59357828, -1.16118953, 2.32744316])

[15]:
```
jacobi(A, b, verbose=True)
```

```
iteration 0, x=[1. 1. 1. 1.]
iteration 1, x=[ 0.5        -2.18181818 -1.1         1.625     ]
iteration 2, x=[ 0.60181818 -1.775      -1.25568182  2.55568182]
iteration 3, x=[ 0.67363636 -1.51628099 -1.14229545  2.38366477]
iteration 4, x=[ 0.67683099 -1.58003151 -1.14798889  2.30081844]
iteration 5, x=[ 0.67159463 -1.6023988  -1.16328751  2.3240132 ]
iteration 6, x=[ 0.67241762 -1.59420614 -1.16215749  2.33048861]
iteration 7, x=[ 0.67301088 -1.59261766 -1.16085528  2.32755762]
iteration 8, x=[ 0.67290929 -1.59358934 -1.16110818  2.32712471]
iteration 9, x=[ 0.6728627  -1.59367518 -1.16122832  2.32745748]
iteration 10, x=[ 0.67287815 -1.59356927 -1.16119431  2.32747465]
iteration 11, x=[ 0.67288194 -1.59356908 -1.16118509  2.32743919]
iteration 12, x=[ 0.67288011 -1.59357994 -1.16118938  2.32744027]
iteration 13, x=[ 0.67287988 -1.59357908 -1.16118999  2.3274438 ]
iteration 14, x=[ 0.67288009 -1.59357804 -1.1611895   2.32744341]
iteration 15, x=[ 0.6728801  -1.59357821 -1.16118948  2.32744308]
iteration 16, x=[ 0.67288007 -1.59357831 -1.16118953  2.32744315]
iteration 17, x=[ 0.67288008 -1.59357828 -1.16118953  2.32744317]
iteration 18, x=[ 0.67288008 -1.59357827 -1.16118953  2.32744316]
iteration 19, x=[ 0.67288008 -1.59357828 -1.16118953  2.32744316]
iteration 20, x=[ 0.67288008 -1.59357828 -1.16118953  2.32744316]
iteration 21, x=[ 0.67288008 -1.59357828 -1.16118953  2.32744316]
iteration 22, x=[ 0.67288008 -1.59357828 -1.16118953  2.32744316]
iteration 23, x=[ 0.67288008 -1.59357828 -1.16118953  2.32744316]
```

[15]: array([0.67288008, -1.59357828, -1.16118953, 2.32744316])

7 Convergence Estimation

[16]:
```
n=4
tol=1e-10
A_rand, b_rand, x = generate_problem(n)
x = np.linalg.solve(A_rand, b_rand)
print(x)
x_jac = jacobi(A=A_rand, b=b_rand, max_iter=1000, tol=1e-10, verbose=False)
print(x_jac)
conv = norm_2(np.dot(A_rand, x) - b_rand) < tol
```

```
[0.20889484 0.30111788 0.33983607 0.02818618]
[0.20889484 0.30111788 0.33983607 0.02818618]
```

[17]:
```
tol=1e-8
reps = 100
n=10
conv = 0
for i in range(reps):
    A_rand, b_rand, x = generate_problem(n)
    try:
        x_jac = jacobi(A=A_rand, b=b_rand, max_iter=100, tol=1e-10,␣
 ↪verbose=False)
        conv += norm_2(np.dot(A_rand, x_jac) - b_rand) < tol
    except ValueError:
        conv+=False
    print('rep: {0}, convergence: {1}'.format(i, conv))
print(conv/reps)
```

```
rep: 0, convergence: 0
rep: 1, convergence: 0
rep: 2, convergence: 1
rep: 3, convergence: 2
rep: 4, convergence: 2
rep: 5, convergence: 2
rep: 6, convergence: 3
rep: 7, convergence: 3
rep: 8, convergence: 3
rep: 9, convergence: 3
rep: 10, convergence: 3
rep: 11, convergence: 3
rep: 12, convergence: 3
rep: 13, convergence: 3
rep: 14, convergence: 4
rep: 15, convergence: 4
rep: 16, convergence: 4
rep: 17, convergence: 5
rep: 18, convergence: 5
```

```
rep: 19, convergence: 5
rep: 20, convergence: 6
rep: 21, convergence: 6
rep: 22, convergence: 6
rep: 23, convergence: 6
rep: 24, convergence: 6
rep: 25, convergence: 6
rep: 26, convergence: 6
rep: 27, convergence: 6
rep: 28, convergence: 7
rep: 29, convergence: 7
rep: 30, convergence: 7
rep: 31, convergence: 8
rep: 32, convergence: 8
rep: 33, convergence: 8
rep: 34, convergence: 9
rep: 35, convergence: 10
rep: 36, convergence: 10
rep: 37, convergence: 10
rep: 38, convergence: 10
rep: 39, convergence: 10
rep: 40, convergence: 11
rep: 41, convergence: 11
rep: 42, convergence: 11
rep: 43, convergence: 11
rep: 44, convergence: 11
rep: 45, convergence: 11
rep: 46, convergence: 11
rep: 47, convergence: 12
rep: 48, convergence: 13
rep: 49, convergence: 13
rep: 50, convergence: 13
rep: 51, convergence: 13
rep: 52, convergence: 13
rep: 53, convergence: 13
rep: 54, convergence: 13
rep: 55, convergence: 13
rep: 56, convergence: 14
rep: 57, convergence: 14
rep: 58, convergence: 14
rep: 59, convergence: 14
rep: 60, convergence: 14
rep: 61, convergence: 15
rep: 62, convergence: 15
rep: 63, convergence: 15
rep: 64, convergence: 16
rep: 65, convergence: 17
rep: 66, convergence: 18
```

```
rep: 67, convergence: 19
rep: 68, convergence: 20
rep: 69, convergence: 20
rep: 70, convergence: 20
rep: 71, convergence: 20
rep: 72, convergence: 20
rep: 73, convergence: 20
rep: 74, convergence: 21
rep: 75, convergence: 22
rep: 76, convergence: 23
rep: 77, convergence: 24
rep: 78, convergence: 24
rep: 79, convergence: 24
rep: 80, convergence: 25
rep: 81, convergence: 25
rep: 82, convergence: 26
rep: 83, convergence: 26
rep: 84, convergence: 26
rep: 85, convergence: 26
rep: 86, convergence: 26
rep: 87, convergence: 26
rep: 88, convergence: 26
rep: 89, convergence: 26
rep: 90, convergence: 26
rep: 91, convergence: 26
rep: 92, convergence: 27
rep: 93, convergence: 27
rep: 94, convergence: 27
rep: 95, convergence: 27
rep: 96, convergence: 27
rep: 97, convergence: 28
rep: 98, convergence: 29
rep: 99, convergence: 29
0.29
```

8 Gauss Siedel

[18]:
```
def gauss(A, b, max_iter=1000, tol=1e-10, verbose=False):
    n = A.shape[0]
    x = np.ones(n)
    y = np.ones(n)
    i = 0
    conv = lambda x: norm_2(np.dot(A, x) - b) < tol
    while (i<max_iter) and (not conv(x)):
        if verbose: print('iteration {0}, x={1}'.format(i, x))
        for j in range(0, n):
            y[j] = b[j]
```

```
            for k in range (0, j):
                y[j]=y[j]-A[j, k]*y[k]
            for k in range(j+1, n):
                y[j]=y[j]-A[j, k]*x[k]
            y[j]= y[j]/A[j, j]
        x = np.copy(y)
        i+=1
    return x
```

[19]:
```
gauss(A, b, 10)
```

[19]: array([0.67288008, -1.59357827, -1.16118953, 2.32744316])

[20]:
```
tol=1e-8
reps = 100
n=10
conv = 0
for i in range(reps):
    A_rand, b_rand, x = generate_problem(n)
    try:
        x_gauss = gauss(A=A_rand, b=b_rand, max_iter=100, tol=1e-10,␣
 ↪verbose=False)
        conv += norm_2(np.dot(A_rand, x_gauss) - b_rand) < tol
    except ValueError:
        conv+=False
    print('rep: {0}, convergence: {1}'.format(i, conv))
print(conv/reps)
```

```
rep: 0, convergence: 1
rep: 1, convergence: 2
rep: 2, convergence: 3
rep: 3, convergence: 4
rep: 4, convergence: 5
rep: 5, convergence: 6
rep: 6, convergence: 7
rep: 7, convergence: 8
rep: 8, convergence: 9
rep: 9, convergence: 10
rep: 10, convergence: 11
rep: 11, convergence: 12
rep: 12, convergence: 13
rep: 13, convergence: 14
rep: 14, convergence: 15
rep: 15, convergence: 16
rep: 16, convergence: 17
rep: 17, convergence: 18
rep: 18, convergence: 19
rep: 19, convergence: 20
```

```
rep: 20, convergence: 21
rep: 21, convergence: 22
rep: 22, convergence: 23
rep: 23, convergence: 24
rep: 24, convergence: 25
rep: 25, convergence: 26
rep: 26, convergence: 27
rep: 27, convergence: 28
rep: 28, convergence: 29
rep: 29, convergence: 30
rep: 30, convergence: 31
rep: 31, convergence: 32
rep: 32, convergence: 33
rep: 33, convergence: 34
rep: 34, convergence: 35
rep: 35, convergence: 36
rep: 36, convergence: 37
rep: 37, convergence: 38
rep: 38, convergence: 39
rep: 39, convergence: 40
rep: 40, convergence: 41
rep: 41, convergence: 42
rep: 42, convergence: 43
rep: 43, convergence: 44
rep: 44, convergence: 45
rep: 45, convergence: 46
rep: 46, convergence: 47
rep: 47, convergence: 48
rep: 48, convergence: 49
rep: 49, convergence: 50
rep: 50, convergence: 51
rep: 51, convergence: 52
rep: 52, convergence: 53
rep: 53, convergence: 54
rep: 54, convergence: 55
rep: 55, convergence: 56
rep: 56, convergence: 57
rep: 57, convergence: 58
rep: 58, convergence: 59
rep: 59, convergence: 60
rep: 60, convergence: 61
rep: 61, convergence: 62
rep: 62, convergence: 63
rep: 63, convergence: 64
rep: 64, convergence: 65
rep: 65, convergence: 66
rep: 66, convergence: 67
rep: 67, convergence: 68
```

```
rep: 68, convergence: 69
rep: 69, convergence: 70
rep: 70, convergence: 71
rep: 71, convergence: 72
rep: 72, convergence: 73
rep: 73, convergence: 74
rep: 74, convergence: 75
rep: 75, convergence: 76
rep: 76, convergence: 77
rep: 77, convergence: 78
rep: 78, convergence: 79
rep: 79, convergence: 80
rep: 80, convergence: 81
rep: 81, convergence: 82
rep: 82, convergence: 83
rep: 83, convergence: 84
rep: 84, convergence: 85
rep: 85, convergence: 86
rep: 86, convergence: 87
rep: 87, convergence: 88
rep: 88, convergence: 89
rep: 89, convergence: 90
rep: 90, convergence: 91
rep: 91, convergence: 92
rep: 92, convergence: 93
rep: 93, convergence: 94
rep: 94, convergence: 95
rep: 95, convergence: 96
rep: 96, convergence: 97
rep: 97, convergence: 98
rep: 98, convergence: 99
rep: 99, convergence: 100
1.0
```

[21]:

```
tol=1e-8
reps = 100
n=10
conv_jac = 0
conv_gauss = 0

for i in range(reps):
    A_rand, b_rand, x = generate_problem(n)
    try:
        x_jac = jacobi(A=A_rand, b=b_rand, max_iter=100, tol=1e-10,␣
 ↪verbose=False)
        conv_jac += norm_2(np.dot(A_rand, x_jac) - b_rand) < tol
    except ValueError:
```

```
        conv_jac+=False

    try:
        x_gauss = gauss(A=A_rand, b=b_rand, max_iter=100, tol=1e-10,␣
 ↪verbose=False)
        conv_gauss += norm_2(np.dot(A_rand, x_gauss) - b_rand) < tol
    except ValueError:
        conv_gauss+=False


    print('rep: {0}, jacobi: {1}, gauss: {2}'.format(i, conv_jac, conv_gauss))

print(conv_jac/reps, conv_gauss/reps)
```

```
rep: 0, jacobi: 0, gauss: 1
rep: 1, jacobi: 1, gauss: 2
rep: 2, jacobi: 1, gauss: 3
rep: 3, jacobi: 2, gauss: 4
rep: 4, jacobi: 2, gauss: 5
rep: 5, jacobi: 2, gauss: 6
rep: 6, jacobi: 3, gauss: 7
rep: 7, jacobi: 3, gauss: 8
rep: 8, jacobi: 3, gauss: 9
rep: 9, jacobi: 4, gauss: 10
rep: 10, jacobi: 5, gauss: 11
rep: 11, jacobi: 5, gauss: 12
rep: 12, jacobi: 6, gauss: 13
rep: 13, jacobi: 6, gauss: 14
rep: 14, jacobi: 6, gauss: 15
rep: 15, jacobi: 6, gauss: 16
rep: 16, jacobi: 6, gauss: 17
rep: 17, jacobi: 6, gauss: 18
rep: 18, jacobi: 6, gauss: 19
rep: 19, jacobi: 6, gauss: 20
rep: 20, jacobi: 7, gauss: 21
rep: 21, jacobi: 7, gauss: 22
rep: 22, jacobi: 7, gauss: 23
rep: 23, jacobi: 7, gauss: 24
rep: 24, jacobi: 7, gauss: 25
rep: 25, jacobi: 7, gauss: 26
rep: 26, jacobi: 7, gauss: 27
rep: 27, jacobi: 8, gauss: 28
rep: 28, jacobi: 8, gauss: 29
rep: 29, jacobi: 9, gauss: 30
rep: 30, jacobi: 9, gauss: 31
rep: 31, jacobi: 9, gauss: 32
rep: 32, jacobi: 10, gauss: 33
rep: 33, jacobi: 11, gauss: 34
```

```
rep: 34, jacobi: 11, gauss: 35
rep: 35, jacobi: 11, gauss: 36
rep: 36, jacobi: 11, gauss: 37
rep: 37, jacobi: 11, gauss: 38
rep: 38, jacobi: 11, gauss: 39
rep: 39, jacobi: 12, gauss: 40
rep: 40, jacobi: 12, gauss: 41
rep: 41, jacobi: 12, gauss: 42
rep: 42, jacobi: 12, gauss: 43
rep: 43, jacobi: 12, gauss: 44
rep: 44, jacobi: 13, gauss: 45
rep: 45, jacobi: 14, gauss: 46
rep: 46, jacobi: 14, gauss: 47
rep: 47, jacobi: 15, gauss: 48
rep: 48, jacobi: 15, gauss: 49
rep: 49, jacobi: 15, gauss: 50
rep: 50, jacobi: 16, gauss: 51
rep: 51, jacobi: 16, gauss: 52
rep: 52, jacobi: 16, gauss: 53
rep: 53, jacobi: 17, gauss: 54
rep: 54, jacobi: 17, gauss: 55
rep: 55, jacobi: 17, gauss: 56
rep: 56, jacobi: 17, gauss: 57
rep: 57, jacobi: 17, gauss: 58
rep: 58, jacobi: 18, gauss: 59
rep: 59, jacobi: 18, gauss: 60
rep: 60, jacobi: 19, gauss: 61
rep: 61, jacobi: 20, gauss: 62
rep: 62, jacobi: 20, gauss: 63
rep: 63, jacobi: 20, gauss: 64
rep: 64, jacobi: 20, gauss: 65
rep: 65, jacobi: 21, gauss: 66
rep: 66, jacobi: 21, gauss: 67
rep: 67, jacobi: 22, gauss: 68
rep: 68, jacobi: 22, gauss: 69
rep: 69, jacobi: 22, gauss: 70
rep: 70, jacobi: 22, gauss: 71
rep: 71, jacobi: 22, gauss: 72
rep: 72, jacobi: 22, gauss: 73
rep: 73, jacobi: 22, gauss: 74
rep: 74, jacobi: 22, gauss: 75
rep: 75, jacobi: 22, gauss: 76
rep: 76, jacobi: 22, gauss: 77
rep: 77, jacobi: 22, gauss: 78
rep: 78, jacobi: 22, gauss: 79
rep: 79, jacobi: 22, gauss: 80
rep: 80, jacobi: 22, gauss: 81
rep: 81, jacobi: 23, gauss: 82
```

```
rep: 82, jacobi: 23, gauss: 83
rep: 83, jacobi: 23, gauss: 84
rep: 84, jacobi: 23, gauss: 85
rep: 85, jacobi: 23, gauss: 86
rep: 86, jacobi: 23, gauss: 87
rep: 87, jacobi: 24, gauss: 88
rep: 88, jacobi: 24, gauss: 89
rep: 89, jacobi: 24, gauss: 90
rep: 90, jacobi: 24, gauss: 91
rep: 91, jacobi: 25, gauss: 92
rep: 92, jacobi: 25, gauss: 93
rep: 93, jacobi: 25, gauss: 94
rep: 94, jacobi: 25, gauss: 95
rep: 95, jacobi: 25, gauss: 96
rep: 96, jacobi: 26, gauss: 97
rep: 97, jacobi: 26, gauss: 98
rep: 98, jacobi: 26, gauss: 99
rep: 99, jacobi: 26, gauss: 100
0.26 1.0
```

9 Question 4

$x_0 = (1, 1, 1)$

$x_1 = (0.6, 0.2, 1)$

$x_2 = (0.45, 0.45, 1)$

$x_3 = (0.48, 0.55, 1)$

$x_4 = (0.5, 0.5, 1)$

$x_5 = (0.5, 0.5, 1)$

$\lambda = \frac{Ax \cdot x}{x \cdot x} = 3$

10 Power Method

[22]:
```
A = np.array([
    [1, 2, 0],
    [2, 0, 1],
    [0, 1, 2]
])

x0 = np.array([1, 1, 1])
```

[23]:
```
def power_iteration(A, x0, max_iter=1000):
    for i in range(max_iter):
        x0 = np.dot(A, x0)
```

```
        x0 = x0/max(x0)
    l = np.dot(np.dot(A, x0), x0)/(np.dot(x0, x0))
    return l, x0
```

[24]:
```
l, e = power_iteration(A, x0)
l, e
```

[24]:
```
(3.0, array([1., 1., 1.]))
```

[25]:
```
L, E = np.linalg.eig(A)
L, E
```

[25]:
```
(array([-1.73205081,  3.        ,  1.73205081]),
 array([[ 0.57735027,  0.57735027, -0.57735027],
        [-0.78867513,  0.57735027, -0.21132487],
        [ 0.21132487,  0.57735027,  0.78867513]]))
```

[26]:
```
np.sort(L)
L[-1]/L[-2]
```

[26]:
```
0.5773502691896256
```

[ ]:
```

```