

Author: Jonas Semprini Næss

MAT4110: Introduction to Numerical Analysis

Problem 1.

a.) Find a Householder transformation H such that $B = HAH^T$ is a tridiagonal matrix. This involves some bookkeeping, so it may help to use the aid of a computer for calculating the linear algebra in steps that you describe.

Solution:

We observe that $A \in \mathbb{R}^{3 \times 3}$, meaning there is only a single Householder iteration needed to obtain our Householder matrix H . This is due to the Householder method only requiring $n - 2$ reflection transformations to obtain the designated tridiagonal matrix, where n naturally is the dimension of A .

Now let $\mathbf{v} = \mathbf{x} + \text{sign}(x_{21})\mathbf{e}_1$ where $\mathbf{x} = [0, 1/\sqrt{2}, -1/\sqrt{2}]^T$, which yields

$$\mathbf{v} = \left[0, \frac{1 + \sqrt{2}}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right]^T.$$

Hence our Householder transformation looks accordingly

$$H = I_3 - 2 \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}.$$

By calculation $\mathbf{v}^T\mathbf{v} = 0^2 + (\frac{1+\sqrt{2}}{\sqrt{2}})^2 + -(\frac{1}{\sqrt{2}})^2 = 2 + \sqrt{2}$, meaning the expression now becomes

$$H = I_3 - \frac{2}{2 + \sqrt{2}}\mathbf{v}\mathbf{v}^T$$

whereby the outer product $\mathbf{v}\mathbf{v}^T$ then gives

$$H = I_3 - \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}+1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & -\frac{1}{\sqrt{2}} & \frac{\sqrt{2}-1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}.$$

Which is then our Householder matrix.

To really verify that H is the transformation we are looking for, we can further calculate

the product $B = HAH^T$.

$$B = HAH^T = HAH \quad (\text{H is symmetric})$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 5 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{5}{2} & \frac{7}{2} \\ -\frac{1}{\sqrt{2}} & \frac{7}{2} & \frac{5}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$= \begin{pmatrix} 5 & -1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & 6 \end{pmatrix}$$

where B is a tridiagonal matrix. ■

b.) Use Gershgorin's second theorem in combination with a similarity transformation to estimate the spectrum of A . Show that all eigenvalues are distinct and describe the location of the eigenvalues as accurately as you can.

Solution:

By Gershgorin's Circle Theorem we know that when approximating the eigenvalues of A , the respective eigenvalues has to lay within at least one of the discs

$$D(a_{ii}, R_i) \subseteq \mathbb{C}.$$

Where a_{ii} are the diagonal elements of A and $R_i = \sum_{j \neq i} |a_{ij}|$.

By this result we can hence elaborate further on the eigenvalues of A by observing the discs

$$D_1(5, \sqrt{2}) \tag{1}$$

$$D_2\left(\frac{5}{2}, \frac{7 + \sqrt{2}}{2}\right). \tag{2}$$

Since $D_1 \subseteq D_2 \Rightarrow D_1 \cap D_2 \neq \emptyset$ it becomes beneficial to dissect these discs even farther seeing we wish to have as many disjoint discs possible ¹ To do so it is advantageous to perform a similarity transformation (looking into the same input space of A , but from the viewpoint of a different basis). Let

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

¹Direct consequence of Greshgorin's second theorem stating that: "If the union of k discs is disjoint from the union of the other $n - k$ discs then the former union contains exactly k and the latter $n - k$ eigenvalues of A , when the eigenvalues are counted with their algebraic multiplicities."

for $\alpha > 0$ and notice that since B from (a.) is simply a reflection transformation of A the spectrum of A is conserved. This means that

$$\begin{aligned}\sigma(A) &= \sigma(HAH^T) \\ &= \sigma(B) \\ &= \sigma(TBT^{-1}).\end{aligned}$$

Thus, by taking the product TBT^{-1} we have that

$$\begin{aligned}TBT^{-1} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & -1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & 6 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\alpha} & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 5 & -\frac{1}{\alpha} & 0 \\ -\alpha & -1 & 0 \\ 0 & 0 & 6 \end{pmatrix}\end{aligned}$$

. Applying Greshgorin's Circle Theorem again yields the following discs

$$\tilde{D}_1 \left(5, \frac{1}{\alpha} \right) \tag{3}$$

$$\tilde{D}_2 (-1, \alpha). \tag{4}$$

Seeing that (from row 3) $|k - 6| \leq 0$, $k \in \mathbb{R}$ we can conclude with $\lambda_1 = 6$ being one of the eigenvalues of A . Moreover, to hence determine the potential other eigenvalues we must demand that

$$\begin{aligned}\tilde{D}_1 \cap \tilde{D}_2 = \emptyset &\Rightarrow 5 - \frac{1}{\alpha} = -1 + \alpha \\ &\Downarrow \\ -\alpha^2 + 6\alpha - 1 &= 0.\end{aligned}$$

This systems has solutions $\alpha_1 \approx 5.828427$, $\alpha_2 \approx 0.1715728$, which yields $\lambda_2 \in \tilde{D}_1(\alpha \approx 5.828427) = B((5, 0.1715716))$ and $\lambda_3 \in \tilde{D}_2(\alpha \approx 0.1715728) = B((-1, 0.1715728))$, telling us that

$$\begin{aligned}\lambda_2 &\in [5 - 0.1715716, 5 + 0.1715716] \\ \lambda_3 &\in [-1 - 0.1715728, -1 + 0.1715728].\end{aligned}$$

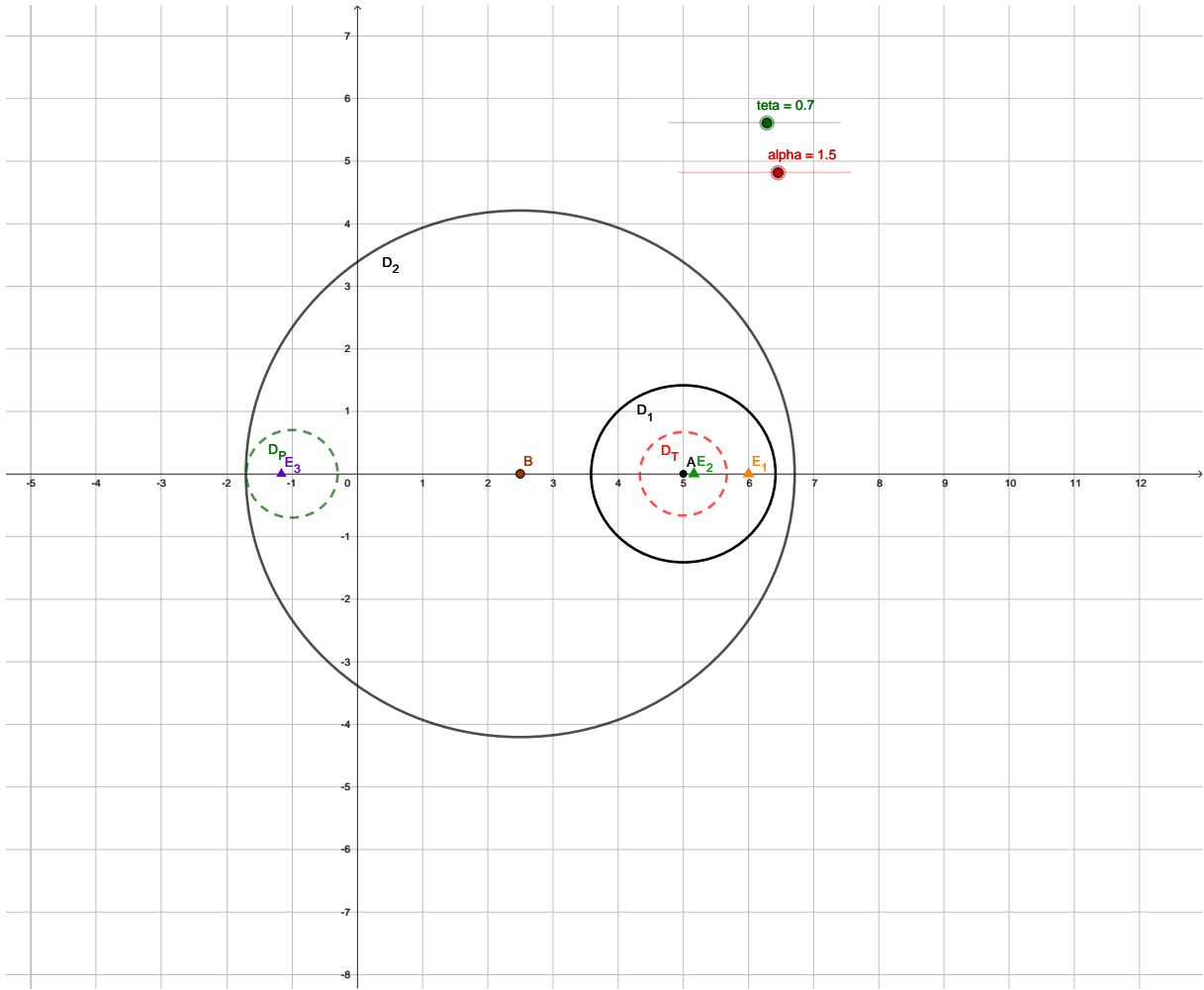


Figure 1: Visual of the Greshgorin circles. D_1, D_2 (Black) are the discs corresponding to A , E_1, E_2, E_3 are the actual eigenvalues of A , D_T, D_P the discs corresponding to TBT^{-1} and $\alpha_1 = 1.5$, $\alpha_2 = 0.7$

c.) *Computer exercise: Approximate the largest eigenvalue of A using power iteration and approximate the other two eigenvalues with inverse iteration (a shift is needed for the middle eigenvalue). Draw a random vector $x^{(0)}$ as start vector (for instance, with the components being independent, standard normal distributed). Include your implementation code and the output of 10 iterations for each approximation.*

Solution:

By running the implemented power iteration over 10 iterations through script 1 the following output is given

Approx Eigenvalue	Actual Eigenvalue	Error
6.0800585	6.0	0.0800585
5.6356824	6.0	0.3643176
5.7005412	6.0	0.2994588
5.7557640	6.0	0.2442360
5.8043507	6.0	0.1956493
5.8457834	6.0	0.1542166
5.8800852	6.0	0.1199148
5.9077903	6.0	0.0922097
5.9297235	6.0	0.0702765
5.9468134	6.0	0.0531866

Table 1: Output through 10 iterations of the power iteration.

For inverse iteration the following is given

Approx Eigenvalue	Actual Eigenvalue	Error
1.0000000	-1.1622777	2.1622777
4.6485161	-1.1622777	5.8107938
-6.1464352	-1.1622777	4.9841576
-0.7073727	-1.1622777	0.4549050
-1.2235930	-1.1622777	0.0613154
-1.1575017	-1.1622777	0.0047760
-1.1616153	-1.1622777	0.0006624
-1.1627635	-1.1622777	0.0004858
-1.1621031	-1.1622777	0.0001746
-1.1623296	-1.1622777	0.0000519
-1.1622635	-1.1622777	0.0000141

Table 2: Output through 10 iterations of the inverse iteration.

and lastly for the shifted inverse iteration with $\mu = 5$.

Approx Eigenvalue	Actual Eigenvalue	Error
6.0000000	5.1622777	8.3772234e-01
5.3504813	5.1622777	1.8820363e-01
5.2036234	5.1622777	4.1345701e-02
5.1691899	5.1622777	6.9122765e-03
5.1634094	5.1622777	1.1317319e-03
5.1624615	5.1622777	1.8380479e-04
5.1623075	5.1622777	2.9834424e-05
5.1622825	5.1622777	4.8415647e-06
5.1622784	5.1622777	7.8568265e-07
5.1622778	5.1622777	1.2749881e-07
5.1622777	5.1622777	2.0690213e-08

Table 3: Output through 10 iterations of the inverse iteration with shift ($\mu = 5$).

Problem 2.

a.) Describe the function $R(\beta)$ for the nonlinear least squares problem for determining this circle and compute its Jacobian $J_R(\beta) \in \mathbb{R}^{20 \times 3}$.

Solution:

The function $R(\beta)$ is simply what is defined as the cost function. By taking the difference of the model(estimation) against the true measurement, you will achieve what is called a residual, or error margin, which describes how much the model deviates at that certain point. Therefore, can $R(\beta)$ look as follows

$$R(\beta) = \begin{bmatrix} f(x_1, y_1; \beta) - b_1 \\ f(x_2, y_2; \beta) - b_2 \\ \vdots \\ f(x_n, y_n; \beta) - b_n \end{bmatrix}$$

which relative to this sample data gives

$$R(\beta) = \begin{bmatrix} f(x_1, y_1; \beta) - b_1 \\ f(x_2, y_2; \beta) - b_2 \\ \vdots \\ f(x_{20}, y_{20}; \beta) - b_{20} \end{bmatrix}.$$

For the case of the Jacobian we remember that a general Jacobian is defined as

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $x \in \mathbb{R}^n$. Hence will the Jacobian for $f(x, y; \beta)$ look as follows

$$J_R(\beta) = \begin{bmatrix} \frac{\partial f_1}{\partial \beta_1} & \cdots & \frac{\partial f_1}{\partial \beta_3} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{20}}{\partial x_1} & \cdots & \frac{\partial f_{20}}{\partial \beta_3} \end{bmatrix}$$

$$= \begin{bmatrix} -2(x_1 - \beta_1) & -2(y_1 - \beta_2) & -2\beta_3 \\ \vdots & \vdots & \vdots \\ -2(x_{20} - \beta_1) & -2(y_{20} - \beta_2) & -2\beta_3 \end{bmatrix}$$

b.) Find a reasonable guess for β^0 and compute the circle β^* that best describes the measurements in least squares sense using the Gauss-Newton method (3) on a computer. Include one plot of containing the circle β^* you obtain and the measurement points.

Solution:

Code can be found in the Appendix at 4, and produces the following plot.

c.) Solve the linear least squares problem with unknowns (x_c, y_c, w) on a computer and use the equation for w to determine the circle's radius, r . How does the solution of this linear least squares problem compare to what you obtained in part b)?

Solution:

By rewriting the equation we can implement the linear least squares as

$$2xx_c + 2yy_c + -x_i^2 - y_i^2 = 0$$

which by the following code (5) gives $r \approx 3.690027147759961$, which aligns well with the results of β_3 from (b.).

Problem 3.

a.) Describe $L_k(x)$ and $\hat{L}_p(y)$ and show that $p(x, y)$ in (7) with the functions $L_k(x)$ and $\hat{L}_p(y)$ that you define indeed is a solution of (6).

Solution:

We know by the rigid definition of the Lagrange Polynomial in one dimension that

$$L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^n \frac{x - x_i}{x_k - x_i} = \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1}) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}$$

which can also be generalised for y , in the same manner, implying that

$$\hat{L}_p(y) = \prod_{\substack{j=0 \\ j \neq p}}^n \frac{y - y_j}{y_p - y_j} = \frac{(y - y_0)(y - y_1) \cdots (y - y_{j-1}) \cdots (y - y_n)}{(y_j - y_0)(y_j - y_1) \cdots (y_j - y_{j-1})(y_j - y_{j+1}) \cdots (y_j - y_n)}$$

Now observe for equation (6) that it becomes

$$\begin{aligned} p(x_i, y_j) &= \sum_{k=0}^n \sum_{p=0}^n f(x_k, y_p) L_k(x_i) \hat{L}_p(y_j) = \sum_{k=0}^n \sum_{p=0}^n f(x_k, y_p) \delta_{ik} \delta_{jp}, \quad i \neq k \wedge j \neq p. \\ &= f(x_i, y_j) \end{aligned}$$

which means that it interpolates the data, since each basis polynomial has degree n and hence must the sum (or linear combinations) be of degree $\leq n$. Conclusively must $p(x, y)$ be a solution of $p(x_i, y_j) = f(x_i, y_j)$, $\forall 0 \leq i, j \leq n$. ■

b.) *Leaning on the fact that any univariate polynomial p of degree $\leq n$ that has $n + 1$ or more zeros is equal to the 0 function, meaning $p \equiv 0$, show that the solution to the 2 dimensional interpolation problem (6) is unique.*

Solution:

Suppose that p is not unique and there exists another solution to the interpolation problem $q \in \mathcal{P}_{n,n}$. Now define $r = p - q$, implying that $r \in \mathcal{P}_{n,n}$, meaning we can write it as

$$\sum_{i=0}^n \sum_{j=0}^n c_{i,j} x^i y^j, \quad c_{i,j} \in \mathbb{R}.$$

Without loss of generality, we can fix any $0 \leq p \leq n$ such that the function $r(x, y_p)$ is now a univariate polynomial in x of degree $\leq n$ and can be written as

$$\sum_{i=0}^n a_i(y_p) x^i$$

where $a_i(y_p) = \sum_{j=0}^n c_{i,j} y_p^j$. This polynomial has $n + 1$ distinct root, namely x_i , $i = 0, 1, \dots, n$. But a polynomial that has $n + 1$ distinct roots and is of degree $\leq n$ can only happen to exist as long its $\equiv 0$, which tells us that the only solution to $r(x, y_p) = 0$ is if all $a_i(y_p) = 0 \iff c_{i,j} = 0, \forall i, j$ meaning $r(x, y) \equiv 0$, which contradicts our assumption and $p \in \mathcal{P}_{n,n}$ must therefore be unique. ■

c.) *Determine $p_{i,j}(x, y)$ and verify that the 2D square-mesh trapezoidal rule is given by*

$$T(n) = \frac{h^2}{4} \sum_{i=1}^n \sum_{j=1}^n (f(x_{i-1}, y_{j-1}) + f(x_i, y_{j-1}) + f(x_{i-1}, y_j) + f(x_i, y_j))$$

Solution:

We recall from (a.) that a general $p(x, y) = \sum_{k=0}^n \sum_{p=0}^n f(x, y) L_k(x) \hat{L}_p(y)$, which if we restrict to only be of degree ≤ 1 yields

$$p_{i,j}(x, y) = \sum_{k=0}^1 \sum_{p=0}^1 f(x_{i-k}, y_{j-p}) L_k(x) \hat{L}_p(y)$$

which written out in full is

$$\begin{aligned}
\sum_{k=0}^1 \sum_{p=0}^1 f(x_{i-k}, y_{j-p}) L_k(x) \hat{L}_p(y) &= \sum_{k=0}^1 L_k(x) \left[\hat{L}_0(y)(f(x_{i-k}, y_j) + \hat{L}_1(y)(f(x_{i-k}, y_{j-1}))) \right] \\
&= L_0(x) \left[\hat{L}_0(y)(f(x_i, y_j) + \hat{L}_1(y)(f(x_i, y_{j-1}))) \right] \\
&\quad + L_1(x) \left[\hat{L}_0(y)(f(x_{i-1}, y_j) + \hat{L}_1(y)(f(x_{i-1}, y_{j-1}))) \right] \\
&= L_0(x) \hat{L}_0(y) f(x_i, y_j) + L_0(x) \hat{L}_1(y) f(x_i, y_{j-1}) \\
&\quad + L_1(x) \hat{L}_0(y) f(x_{i-1}, y_j) + L_1(x) \hat{L}_1(y) f(x_{i-1}, y_{j-1}).
\end{aligned}$$

With $p_{i,j}(x, y)$ explicitly defined we can then turn to the integral

$$\begin{aligned}
\int_{x_{i-1}}^{x_i} \int_{y_{j-1}}^{y_j} p_{i,j}(x, y) &= \int_{x_{i-1}}^{x_i} L_0(x) \frac{y_j - y_{j-1}}{2} \left[f(x_i, y_j) + f(x_i, y_{j-1}) \right] + \\
&\quad L_1(x) \frac{y_j - y_{j-1}}{2} \left[f(x_{i-1}, y_j) + f(x_{i-1}, y_{j-1}) \right] \\
&= \frac{x_i - x_{i-1}}{2} \frac{y_j - y_{j-1}}{2} \left[f(x_i, y_j) + f(x_i, y_{j-1}) + f(x_{i-1}, y_j) + f(x_{i-1}, y_{j-1}) \right]
\end{aligned}$$

Where we remember that the problem is enclosed within a 2D square mesh with $(x_i, y_j) = (ih, jh)$, hence inferring that

$$\frac{x_i - x_{i-1}}{2} = \frac{y_j - y_{j-1}}{2} = \frac{h}{2} \iff \frac{x_i - x_{i-1}}{2} \frac{y_j - y_{j-1}}{2} = \frac{h^2}{4}.$$

Which in application of the trapezoidal rule conclusively yields

$$\begin{aligned}
T(n) &= \sum_{i=1}^n \sum_{j=1}^n \frac{h^2}{4} \left(f(x_{i-1}, y_{j-1}) + f(x_i, y_{j-1}) + f(x_{i-1}, y_j) + f(x_i, y_j) \right) \\
&= \frac{h^2}{4} \sum_{i=1}^n \sum_{j=1}^n \left(f(x_{i-1}, y_{j-1}) + f(x_i, y_{j-1}) + f(x_{i-1}, y_j) + f(x_i, y_j) \right)
\end{aligned}$$

and we are done. ■

d.) For $n^s = 2^{1+s}$, $s = 1, 2, \dots, 7$ compute $T(n^s)$ to estimate the integral

$$I = \int_0^1 \int_0^1 \exp(-(x - \sin(y^2))^3) dx dy$$

Estimate numerically the order of convergence r in

$$E(n) = cn^{-r} + \mathcal{O}(n^{-(r+1)})$$

Solution:

Appendix

Source code

Listing 1: Power Iteration

```
1 import numpy as np
2 import pandas as pd
3 import random
4
5 matrix = np.array(
6     [
7         [5, 1 / np.sqrt(2), -1 / np.sqrt(2)],
8         [1 / np.sqrt(2), 5 / 2, 7 / 2],
9         [-1 / np.sqrt(2), 7 / 2, 5 / 2],
10    ]
11 )
12
13 result = pd.DataFrame(columns=["Approx Eigenvalue", "Actual Eigenvalue", "Error"])
14
15 num_iterations = 10
16 actual_eigenvalues, _ = np.linalg.eig(matrix)
17 print(actual_eigenvalues)
18 actual_largest_eigenvalue = max(actual_eigenvalues)
19
20 n = matrix.shape[0]
21 b = np.random.rand(n)
22
23 for i in range(num_iterations):
24     # Power iteration
25     b = np.dot(matrix, b)
26     # Normalize the vector
27     eigenvalue = np.linalg.norm(b)
28     error = abs(eigenvalue - actual_largest_eigenvalue)
29     b /= eigenvalue
30     result.loc[i] = [eigenvalue, actual_largest_eigenvalue, error]
31
32
33 with pd.option_context(
34     "display.max_rows",
35     None,
36     "display.max_columns",
37     None
```

Listing 2: Inverse Iteration

```
1 import numpy as np
2 import pandas as pd
3 import random
4
5 matrix = np.array(
```

```

6      [
7          [5, 1 / np.sqrt(2), -1 / np.sqrt(2)],
8          [1 / np.sqrt(2), 5 / 2, 7 / 2],
9          [-1 / np.sqrt(2), 7 / 2, 5 / 2],
10     ]
11 )
12
13 result = pd.DataFrame(columns=["Approx Eigenvalue", "Actual Eigenvalue", "Error"])
14
15 num_iterations = 10
16 actual_eigenvalues, _ = np.linalg.eig(matrix)
17
18
19 def inverse_power_method(matrix, iter, tol=1e-15):
20     A = matrix
21     b = np.zeros((len(A), iter + 1))
22     b[:, 0] = np.random.normal(0, 0.1, A.shape[0])
23     rn = np.ones((iter + 1,))
24     for k in range(num_iterations):
25         b[:, k] = b[:, k] / np.linalg.norm(b[:, k])
26         b[:, k + 1] = np.linalg.solve(A, b[:, k])
27         rn[k + 1] = np.sum(b[:, k + 1]) / np.sum(b[:, k])
28         if abs(rn[k + 1] - rn[k]) < tol:
29             break
30     if k < iter:
31         rn[k + 2 :] = rn[k + 1]
32     inv_pow = 1.0 / rn
33     for i in range(0, len(inv_pow)):
34         eigenvalue = inv_pow[i]
35         error = abs(eigenvalue - actual_eigenvalues[1])
36         result.loc[i] = [eigenvalue, actual_eigenvalues[1], error]
37
38
39
40 inverse_power_method(matrix, iter=num_iterations)
41
42 with pd.option_context(
43     "display.max_rows",
44     None,
45     "display.max_columns",
46     None,
47     "display.precision",
48     7,
49 ):
50     print(result)

```

Listing 3: Inverse Iteration with shift

```

1 import numpy as np
2 import pandas as pd

```

```

3 import random
4
5 matrix = np.array(
6     [
7         [5, 1 / np.sqrt(2), -1 / np.sqrt(2)],
8         [1 / np.sqrt(2), 5 / 2, 7 / 2],
9         [-1 / np.sqrt(2), 7 / 2, 5 / 2],
10    ]
11 )
12
13 result = pd.DataFrame(columns=["Approx Eigenvalue", "Actual Eigenvalue", "Error"])
14
15 num_iterations = 10
16 actual_eigenvalues, _ = np.linalg.eig(matrix)
17
18 def inverse_power_method(matrix, mu, iter, tol=1e-15):
19     Ashift = matrix - mu * np.identity(matrix.shape[0])
20     b = np.zeros((len(Ashift), iter + 1))
21     b[:, 0] = np.random.normal(0.5, 0.3, Ashift.shape[0])
22     rn = np.ones((iter + 1,))
23     for k in range(num_iterations):
24         b[:, k] = b[:, k] / np.linalg.norm(b[:, k])
25         b[:, k + 1] = np.linalg.solve(Ashift, b[:, k])
26         rn[k + 1] = np.sum(b[:, k + 1]) / np.sum(b[:, k])
27         if abs(rn[k + 1] - rn[k]) < tol:
28             break
29     if k < iter:
30         rn[k + 2 :] = rn[k + 1]
31     inv_pow = 1.0 / rn + mu
32     for i in range(0, len(inv_pow)):
33         eigenvalue = inv_pow[i]
34         error = abs(eigenvalue - actual_eigenvalues[0])
35         result.loc[i] = [eigenvalue, actual_eigenvalues[0], error]
36
37
38 mu = 5
39 inverse_power_method(matrix, mu, iter=num_iterations)
40
41 with pd.option_context(
42     "display.max_rows",
43     None,
44     "display.max_columns",
45     None,
46     "display.precision",
47     7,
48 ):
49     print(result)

```

Listing 4: Least Squares fit Circle Fit

```

1 import scipy.io
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 measurementData = scipy.io.loadmat("circle-measurements.mat")
6 x = measurementData["x"].reshape(-1)
7 y = measurementData["y"].reshape(-1)
8
9 phi = np.linspace(0, 1, 100)
10
11 b_1 = (max(x) + min(x)) / 2
12 b_2 = (max(y) + min(y)) / 2
13 b_3 = (abs(max(x)) + abs(min(x)) + abs((max(y)) + abs(min(x)))) / 4
14 beta = [b_1, b_2, b_3]
15
16 n = 3
17 matrix = np.zeros((len(x), n))
18
19
20 def circle_model(beta, x, y):
21     return (x - beta[0]) ** 2 + (y - beta[1]) ** 2 - beta[2] ** 2
22
23
24 def Jacobian(matrix, beta):
25     jac = matrix
26     for i in range(0, len(x)):
27         jac[i][0] = -2 * (x[i] - beta[0])
28         jac[i][1] = -2 * (y[i] - beta[1])
29         jac[i][2] = -2 * (beta[2])
30     return jac
31
32
33 def gauss_newton(beta, max_iter=100, tolerance=1e-6):
34     params = beta
35     for _ in range(max_iter):
36         J = Jacobian(matrix, params)
37         r = circle_model(params, x, y)
38         J_T = np.transpose(J)
39         S = np.linalg.pinv(np.matmul(J_T, J))
40         delta_params = np.matmul(S, np.matmul(J_T, r))
41
42         # print(delta_params.shape, r.shape)
43         params = params - delta_params
44
45         if np.linalg.norm(delta_params) < tolerance:
46             break
47
48     return params
49
50

```

```

51 beta_n = gauss_newton(beta)
52 print(beta_n)
53
54 s1 = beta_n[0] + beta_n[2] * np.cos(2 * np.pi * phi)
55 s2 = beta_n[1] + beta_n[2] * np.sin(2 * np.pi * phi)
56 plt.scatter(x, y, marker="o", color="green", label="Measure points")
57 plt.plot(s1, s2, color="red")
58 plt.xlim(min(x), max(x))
59 plt.ylim(min(y) - 1, max(y) + 2)
60 plt.legend()
61 plt.show()

```

Listing 5: Linear Least Squares fit Circle Fitt

```

1 import scipy.io
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 measurementData = scipy.io.loadmat("circle-measurements.mat")
6 x = measurementData["x"].reshape(-1)
7 y = measurementData["y"].reshape(-1)
8
9 A = np.c_[2 * x, 2 * y, np.ones_like(x)]
10
11 r = np.linalg.solve(A.T @ A, A.T @ (-(x**2) - (y**2)))
12
13 r_fit = np.sqrt(r[0] ** 2 + r[1] ** 2 - r[2])
14
15 print(r_fit)

```