HW1_Q2

November 11, 2022

1 HW1 - Q2: Eigenvalues and Eigenvectors (30 points)

Notes: * Questions (a), (b) need to be typewritten. * Questions (c), (d) need to be programmed. * For typewritten solution: * Write all the steps of the solution . * Use proper LATEX formatting and notation for all mathematical equations, vectors, and matrices. * For programming solution: * Properly add comments to your code.

1.0.1 (a) Write down the characteristic equation for matrix

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

Use the above characteristic equation to solve for eigenvalues and normalized eigenvectors of matrix A. (7 points)

Your answer here:

$$\begin{aligned}
\mathbf{Ax} = \ \mathbf{x} = \ \mathbf{Ix} \quad & \tag{1}\\
\mathbf{Ax} - \ \mathbf{Ix} = 0 \quad & \tag{2}\\
(\mathbf{A} - \ \mathbf{I})\mathbf{x} = 0 \quad & \tag{3}\\
x \neq 0 \implies \det(\mathbf{A} - \ \mathbf{I}) = 0 \quad & \tag{4}\\
& \tag{5}\\
\mathbf{A} - \ \mathbf{I} = \begin{bmatrix} 2 - \ & 2 \\ 5 & -1 - \ \end{bmatrix} \quad & \tag{6}\\
\det(\mathbf{A} - \ \mathbf{I}) = \ ((2 - \ ) \cdot (-1 - \ )) - (2 \cdot 5) = (-2 - 2 \ + \ + \ ^2) - 10 \quad & \tag{7}\\
\det(\mathbf{A} - \ \mathbf{I}) = \ ^2 - \ - 12 \quad & \tag{8}\\
\det(\mathbf{A} - \ \mathbf{I}) = (\ - 4)(\ + 3) \quad & \tag{9}\\
& \tag{10}\\
& \tag{11}\\
\lambda_1 = 4 \quad & \tag{12}\\
(\mathbf{A} - 4\mathbf{I})\mathbf{x} = \begin{bmatrix} -2 & 2 \\ 5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \quad & \tag{13}\\
-2x_1 + 2x_2 = 0 \quad & \tag{14}\\
5x_1 + 5x_2 = 0 \quad & \tag{15}\\
x_1 = 1 \quad & \tag{16}\\
x_2 = 1 \quad & \tag{17}\\
\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad & \tag{18}\\
& \tag{19}\\
& \tag{20}\\
\lambda_1 = -3 \quad & \tag{21}\\
(\mathbf{A} + 3\mathbf{I})\mathbf{x} = \begin{bmatrix} 5 & 2 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \quad & \tag{22}\\
5x_1 + 2x_2 = 0 \quad & \tag{23}\\
x_1 = \frac{-2x_2}{5} \quad & \tag{24}\\
x_2 = 5 \quad & \tag{25}\\
x_1 = -2 \quad & \tag{26}\\
\mathbf{x} = \begin{bmatrix} -2 \\ 5 \end{bmatrix} \quad & \tag{27}\\
& \tag{28}
\end{aligned}$$

1.0.2 (b) Prove that if a real matrix $A_{n\times n}$ has unique eigenvalues (i.e. $\lambda_i \neq \lambda_j \,\forall\, i \neq j$), then the eigenvectors x_i are linearly independent. (7 points)

1.0.3 Hint: Prove by contradiction, start with $n = 2$ case.

Your answer here: Suppose $\mathbf{A_{2x2}}$ has two unique eigenvalues λ_1 and λ_2 where $\mathbf{Ax_1} = {}_{\mathbf{1}}\mathbf{x_1}$ and $\mathbf{Ax_2} = {}_{\mathbf{2}}\mathbf{x_2}$

To prove by contradiction that $\mathbf{x_1}$ and $\mathbf{x_2}$ are **linearly independent**, we first assume they are **linearly dependent**:

$$\mathbf{x_2} = c_1\mathbf{x_1} \text{ s.t. } c_1 \neq 0 \tag{1}$$

Multiple **(1)** by **A**:

$$\mathbf{Ax_2} = c_1\mathbf{Ax_1} \tag{2}$$

Now use $\mathbf{Ax_i} = {}_{\mathbf{i}}\mathbf{x_i}$:

$$\lambda_2\mathbf{x_2} = \lambda_1 c_1\mathbf{x_1} \tag{3}$$

Multiply **(1)** by λ_2:

$$\lambda_2\mathbf{x_2} = \lambda_2 c_1\mathbf{x_1} \tag{4}$$

Substract **(3)** from **(4)**:

$$\lambda_2 c_1\mathbf{x_1} - \lambda_1 c_1\mathbf{x_1} = 0$$

$$(\lambda_2 - \lambda_1)c_1\mathbf{x_1} = 0$$

Since $c_1 \neq 0$:

$$\lambda_2 - \lambda_1 = 0$$
$$\lambda_2 = \lambda_1$$

This **contradicts** $\lambda_i \neq \lambda_j \,\forall\, i \neq j$, so we conclude that $\mathbf{x_1}$ and $\mathbf{x_2}$ are **linearly independent**

1.0.4 (c) Write function `power_method(A,x)`, which takes as input matrix A and a vector x, and uses power method to calculate eigenvalue and eigenvector. Get the largest eigenvalue and eigenvector for matrix

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 3 & 2 \\ 2 & 4 & 1 \end{bmatrix}$$

1.0.5 **using above function. Start with intial eigenvector guesses: `[-1, 0.5, 3]` and `[2,-6,0.2]`. For each of the vectors, iterate until convergence. Plot how the eigenvalue changes w.r.t. iterations. Report the number of steps it took to converge, eigenvalue and eigenvector. Match your output with the results generated by the numpy API: `numpy.linalg.eig`**

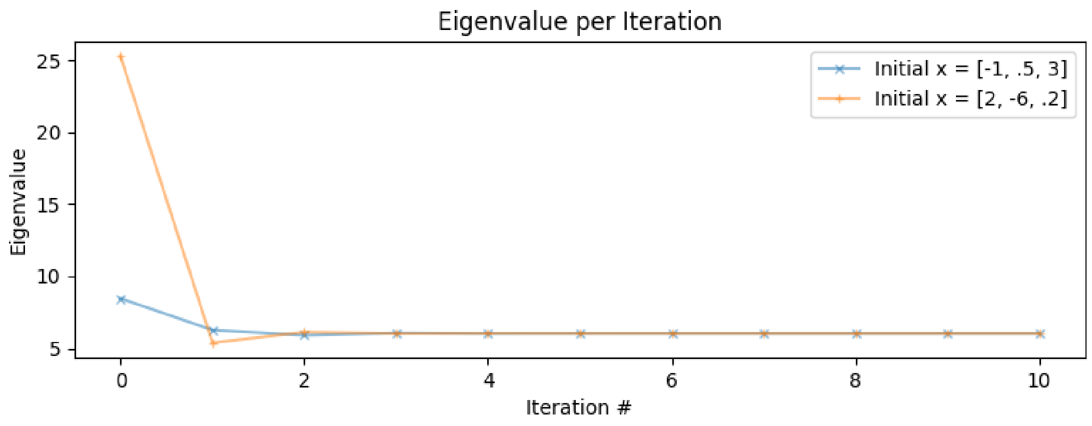
1.0.6 **Note that you only need to look at magnitudes of eigenvalues. Use an absolute tolerance of 10^{-6} between eigenvalue output of previous and current iteration as stopping criteria. You may also need to normalize the final eigenvector to match with output of numpy API `numpy.linalg.eig`. (8 points)**

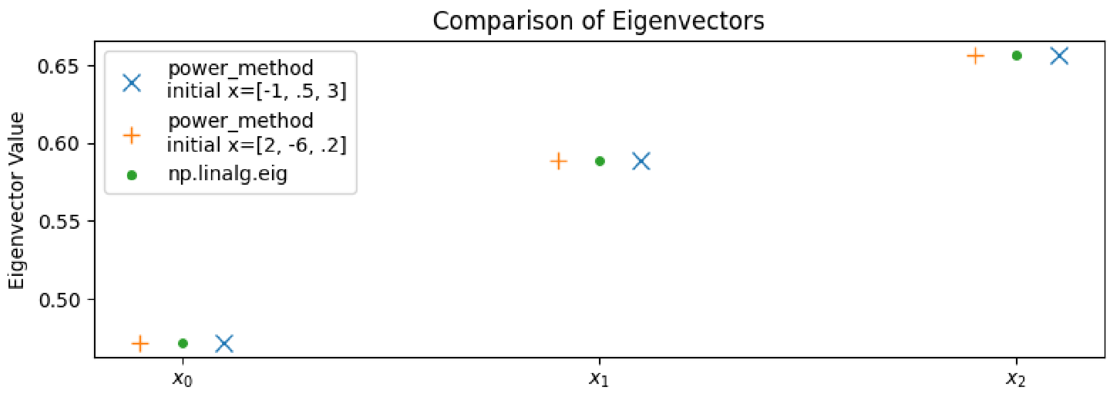
[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

def power_method(A, x, abs_tol=1e-6, max_iter=1000):
    """Compute eigenvalue & eigenvector via power method."""

    # Initialize eigenvector
    v = x
    v_diff = np.inf
    v_last = np.inf

    # Initilize eigenvalue
    w_norm = np.zeros(max_iter)

    # Iterative search
    k = 0
    while v_diff > abs_tol and k < max_iter:

        # Update eigenvalue
        w = np.dot(A, v)
        w_norm[k] = np.linalg.norm(w)

        # Update eigenvector
        v = w / w_norm[k]

        # Delta eigenvector
        v_diff = np.linalg.norm(v-v_last)
        v_last = v

        k += 1

    w_norm = w_norm[:k]

    return w_norm, v


def _print_results(w0, w1, v0, v1):
    np.set_printoptions(precision=3)

    sep = "\t\t"
    print('\t', "n_steps", "eigenvalues", "eigenvectors", sep=sep)
    print(f"x=[-1, .5, 3]{sep}{len(w0)}{sep}{w0[-1]:.3f}{sep}\t{v0}")
    print(f"x=[-1, .5, 3]{sep}{len(w1)}{sep}{w1[-1]:.3f}{sep}\t{v1}")


def _plot_results(w0, w1, v0, v1, v_np):

    fig, axs = plt.subplots(2, 1, figsize=(8, 6))

    axs[0].plot(w0, label='Initial x = [-1, .5, 3]', alpha=.5, marker='x', ms=5)
    axs[0].plot(w1, label='Initial x = [2, -6, .2]', alpha=.5, marker='+', ms=5)

    axs[0].legend()
    axs[0].set_xlabel('Iteration #')
    axs[0].set_ylabel('Eigenvalue')
    axs[0].set_title('Eigenvalue per Iteration')

    offset = 0.1
    axs[1].plot([1+offset, 2+offset, 3+offset], np.abs(v0), marker="x", ms=8,␣
 ↪ls='',
                label='power_method\ninitial x=[-1, .5, 3]')
    axs[1].plot([1-offset, 2-offset, 3-offset], np.abs(v1), marker="+", ms=8,␣
 ↪ls='',
                label='power_method\ninitial x=[2, -6, .2]')
    axs[1].plot([1, 2, 3], np.abs(v_np), marker=".", ms=8, ls='',
                label='np.linalg.eig')

    axs[1].legend()
    axs[1].set_title('Comparison of Eigenvectors')
    axs[1].set_xticks([1, 2, 3], labels=[r'$x_0$', r'$x_1$', r'$x_2$'])
    axs[1].set_ylabel('Eigenvector Value')

    fig.tight_layout()
```

[2]:
```
# Define arrays
A = np.array([
    [2, 1, 2],
    [1, 3, 2],
    [2, 4, 1]
])

x0 = np.array([-1, .5, 3])
x1 = np.array([2, -6, .2])
```

```
# Compute with custom function
w0, v0 = power_method(A, x0)
w1, v1 = power_method(A, x1)

# Compute with numpy
w_np, v_np = np.linalg.eig(A)

_plot_results(w0, w1, v0, v1, v_np[:,0])
_print_results(w0, w1, v0, v1)
```

```
                        n_steps         eigenvalues             eigenvectors
x=[-1, .5, 3]           11              6.029                   [0.472 0.589
0.656]
x=[-1, .5, 3]           11              6.029                   [-0.472 -0.589
-0.656]
```



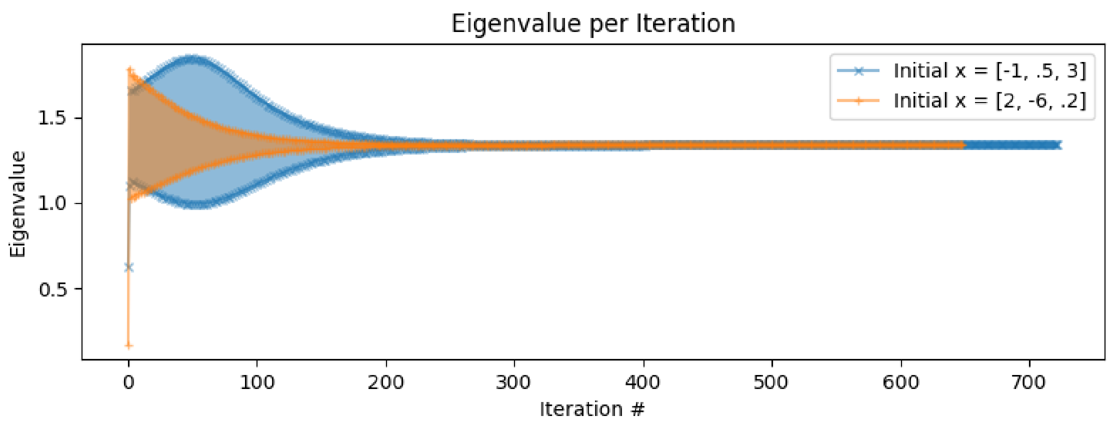


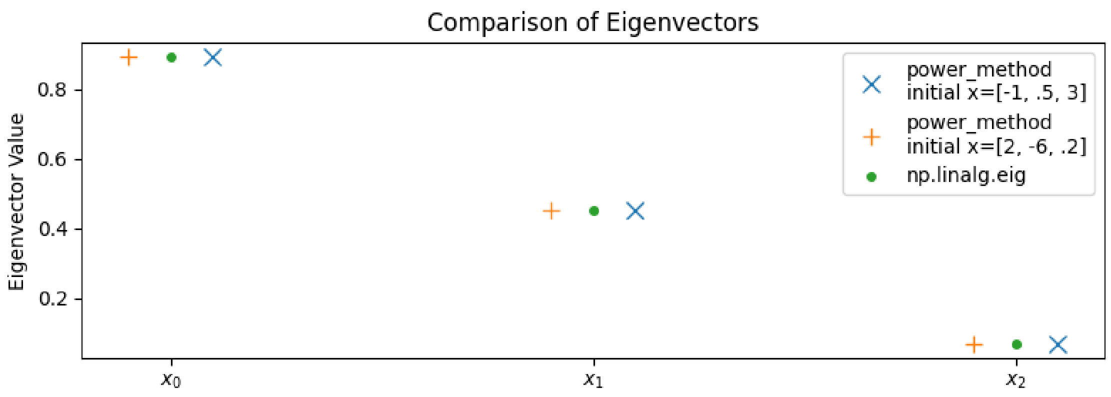
1.0.7 **(d) Write function `inverse_power_method(A,x)`, which takes as input matrix A and a vector x, and uses inverse power method to calculate the smallest eigenvalue and corresponding eigenvector. Solve for the smallest eigenvalue and corresponding eigenvector for the matrix from (c). Use the same intial eigenvector guesses as (c). Report how many iterations do you need for it to converge to the smallest eigenvalue. Plot the computed/estimated eigenvalue w.r.t iterations (keep in mind to plot $1/\lambda$ vs. number of iterations). Report the final eigenvalue and eigenvector you get. Match your answer with the results generated by the numpy API `numpy.linalg.eig`.**

1.0.8 **Note that you only need to look at magnitudes of eigenvalues. Use an absolute tolerance of 10^{-6} between eigenvalue output of previous and current iteration as stopping criteria. You may also need to normalize the final eigenvector to match with output of numpy API `numpy.linalg.eig`. (8 points)**

[3]:
```python
def inverse_power_method(A, x, abs_tol=1e-6, max_iter=1000):
    """Compute eigenvalue & eigenvector via inverse power method."""
    A_inv = np.linalg.inv(A)
    w_norm, v = power_method(A_inv, x, abs_tol, max_iter)
    return 1 / w_norm, v

w2, v2 = inverse_power_method(A, x0)
w3, v3 = inverse_power_method(A, x1)

_plot_results(w2, w3, v2, v3, v_np[:, 1])
_print_results(w2, w3, v2, v3)
```

```
                        n_steps         eigenvalues              eigenvectors
x=[-1, .5, 3]           723             1.336                    [-0.89   0.451
0.07 ]
x=[-1, .5, 3]           649             1.336                    [ 0.89  -0.451
-0.07 ]
```


Eigenvalue per Iteration
Eigenvalue
Iteration #
Initial x = [-1, .5, 3]
Initial x = [2, -6, .2]
1.5
1.0
0.5
0
100
200
300
400
500
600
700


Comparison of Eigenvectors
Eigenvector Value
power_method initial x=[-1, .5, 3]
power_method initial x=[2, -6, .2]
np.linalg.eig
0.8
0.6
0.4
0.2
x_0
x_1
x_2