Eigensystems of Matrices

Wanzhu Zheng

June 2023

1 Problem Description

This report demonstrates how to determine the eigensystem of matrices using power methods.

Suppose we have a matrix $A = 10I + B + B^T + B^T B$ with B being a 50×50 matrix with a normal distribution.

Power Method:
The Power Method is an iterative technique used to determine the eigenvalue of largest magnitude in a matrix. From there, you can find the associated eigenvector.

Inverse Power Method:
The Inverse Power Method is a modification of the power method that gives faster convergence. We first initialize a number q. The inverse method determines the eigenvalue of matrix A that is closest to q.

2 Results

Here, we demonstrate the result of finding the eigensystems for matrix A with $TOL < 10^{-8}$.

Using the power and inverse power method, we find

The largest eigenvalue of A is 34.73723941488993
It's associated eigenvector is: $\begin{bmatrix}-0.04093639 & 0.24878569 & 0.24878569... & -0.11401018\end{bmatrix}$

The smallest eigenvalue of A is 0.10583656882867419
It's associated eigenvector is: $\begin{bmatrix}-0.45547313 & 0.55333851 & -0.1343531... & 0.20653561\end{bmatrix}$

We repeat this experiment 25 times with different matrices A to find the average time it takes to find the eigenvalue, the average eignvalue error, and the average eigenvector error.

	λ Time	λ Error	Vector Error
Smallest Eig	0.035561	2.2078535×10^{-9}	0
Largest Eig	2.348089	1.2423501×10^{-9}	5.99342×10^{-8}

We see that inverse power method has a faster wall clock time. Inverse method is similar to power convergence where both of their run times are linear. However, unlike the power method we have a choice in what eigenvector to find by supplying an estimate q of the corresponding eigenvalue. Thus, we can control the rate of convergence by choosing which q to use. If q is much closer to one eigenvalue of A than to the others, then the largest eigenvalue will be much larger than the rest.

3 Appendix

```
import numpy as np
import time

B = np.random.normal(size=(50, 50))
A = 10 * np.eye(50) + B + B.T + B.T * B

# built-in method
eig_values, eig_vectors = np.linalg.eig(A)
print(eig_values)


# power method
def get_p(x):
    return np.argwhere(np.isclose(np.abs(x), np.linalg.norm(x, np.inf))).min()


def helper(A, x, p):
    y = np.dot(A, x)
    l = y[p]
    p = get_p(y)
    err = np.linalg.norm(x - y / y[p], np.inf)
    x = y / y[p]

    return err, p, l, x


def power_method(A, tol=1e-8, max_iter=1000):
    n = A.shape[0]
    x = np.ones(n)

    p = get_p(x)
    err = 1
    x = x / x[p]
    for i in range(max_iter):

        if err < tol:
            break
        err, p, l, x = helper(A, x, p)

    return l, x


print(power_method(A))


def lu(A):
    L = np.zeros_like(A)
    U = np.zeros_like(A)
    N = np.size(A, 0)

    for k in range(N):
        L[k, k] = 1
```

```
        U[k, k] = (A[k, k] - np.dot(L[k, :k], U[:k, k])) / L[k, k]
        for j in range(k + 1, N):
            U[k, j] = (A[k, j] - np.dot(L[k, :k], U[:k, j])) / L[k, k]
        for i in range(k + 1, N):
            L[i, k] = (A[i, k] - np.dot(L[i, :k], U[:k, k])) / U[k, k]
    return L, U


# forward substitution
def forward_sub(A, b):
    n = len(b)
    x = np.zeros(n)
    for i in range(n):
        x[i] = b[i] - np.dot(A[i, :i], x[:i]) / A[i][i]
    return x


def back_sub(A, b):
    n = len(b)
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]
    return x


# LUx = b
def lu_solver(A, b):
    L = lu(A)[0]
    U = lu(A)[1]
    y = forward_sub(L, b)
    x = back_sub(U, y)
    return x


# inverse power method
def inverse_power_method(A, tolerance=1e-8, max_iterations=1000):

    n = A.shape[0]
    x = np.ones(n)
    I = np.eye(n)

    q = 0
    p = get_p(x)
    err = 1
    x = x / x[p]

    for iter in range(max_iterations):
        if err < tolerance:
            break

        M = A - q * I
        y = lu_solver(M, x)
        miu = y[p]
        p = get_p(y)
```

```
        err = np.linalg.norm(x - y / y[p],  np.inf)
        x = y / y[p]
        miu = 1. / miu + q

    return miu, x


print(inverse_power_method(A))


# average time
def clock(type=0):
    B = np.random.normal(size=(50, 50))
    A = 10 * np.eye(50) + B + B.T + B.T * B
    if type == 0:
        t0 = time.time()
        power_method(A)
        t1 = time.time()
        total_time = t1 - t0
        return total_time
    if type == 1:
        t0 = time.time()
        inverse_power_method(A)
        t1 = time.time()
        total_time = t1 - t0
        return total_time


def error(type=0):
    B = np.random.normal(size=(50, 50))
    A = 10 * np.eye(50) + B + B.T + B.T * B
    if type == 0:
        l_true = max(eig_values)
        l = power_method(A)[0]
        eig_val_error = abs(l_true - l)

        x = power_method(A)[1]
        numer = np.linalg.norm(A * x - l * x, np.inf)
        denom = abs(l)
        eig_vec_error = numer / denom
        return eig_val_error, eig_vec_error
    if type == 1:
        l_true = eig_values[min(range(len(eig_values)), key = lambda i: abs(eig_values[i]))]
        l = inverse_power_method(A)[0]
        eig_val_error = abs(l_true - l)

        x = inverse_power_method(A)[1]
        numer = np.linalg.norm(A * x - l * x, np.inf)
        denom = abs(l)
        eig_vec_error = numer / denom
        return eig_val_error, eig_vec_error


smallest_avg_time = np.empty((0, 0))
```

```
largest_avg_time = np.empty((0, 0))
smallest_avg_eigval_error = np.empty((0, 0))
largest_avg_eigval_error = np.empty((0, 0))
smallest_avg_eigvec_error = np.empty((0, 0))
largest_avg_eigvec_error = np.empty((0, 0))
for t in range(25):
    smallest_avg_time = np.append(smallest_avg_time, clock(type=1))
    largest_avg_time = np.append(largest_avg_time, clock(type=0))
    smallest_avg_eigval_error = np.append(smallest_avg_eigval_error, error(type=1)[0])
    largest_avg_eigval_error = np.append(largest_avg_eigval_error, error(type=0)[0])
    smallest_avg_eigvec_error = np.append(smallest_avg_eigval_error, error(type=1)[1])
    slargest_avg_eigvec_error = np.append(largest_avg_eigval_error, error(type=0)[1])

print(sum(smallest_avg_time) / 25)
print(sum(largest_avg_time) / 25)
print(sum(smallest_avg_eigval_error) / 25)
print(sum(largest_avg_eigval_error) / 25)
print(sum(smallest_avg_eigvec_error) / 25)
print(sum(largest_avg_eigvec_error) / 25)
```