Eigenvalue problems III: Advanced Numerical Methods

Sam Sinayoko

Computational Methods 10

Contents


```
import numpy as np
import scipy as sp
import scipy.linalg
```

1 Learning Outcomes

By the end of this lecture, you should be able to

- List three alternatives to the Jacobi eigenvalue method for solving an eigenvalue problem.
- Discuss the principle behind each method and its main advantage.

2 Introduction

The the previous lectures on eigenvalue problems we introduced and used the Jacobi eigenvalue method, which uses successive rotation matrices to diagonalise the input matrix A. The Jacobi eigenvalue method requires the input matrix A to be symmetric. In this lecture, we will discuss alternative numerical methods, applicable to generic matrices, including:

- the inverse power method
- the Krylov subspace method
- the QR method

3 Inverse Power method: finding the smallest eigenvalue of a symmetric matrix

3.1 Theory

Consider an eigenvalue problem

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \tag{1}$$

where A is an $n \times n$ matrix.

We will use an iterative approach similar to the one introduced in lecture 01:

1. take an initial guess
2. solve the iterative problem to obtain the solution
3. set that solution as your new guess and go to step 2 until the solution converges.

Here we define the iterative problem as follows. Given a **unit vector v**, Find the vector **z** such that

$$\mathbf{A}\mathbf{z} = \mathbf{v}. \tag{2}$$

Compute $|\mathbf{z}|$ and let $\mathbf{v} = \mathbf{z}/|\mathbf{z}|$ for the next iteration. Stop when the magnitude of $|\mathbf{z}|$ has converged.

At the conclusion of the procedure, $|\mathbf{z}| = \pm 1/\lambda_1$, where λ_1 is the smallest eigenvalue of $\mathbf{A}$. Note that we have assumed here that λ_1 is real. If it is complex, then $|\mathbf{z}| = 1/|\lambda_1|$.

3.1.1 Why does this work?

If you expand $\mathbf{v}$ and $\mathbf{z}$ along the basis of eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \cdots$, it is easy to show that the component of $\mathbf{x}_1$ increases in $\mathbf{z}$ compared to $\mathbf{v}$, whereas the components of other eigenvectors $\mathbf{x}_i$, $i > 1$, decrease [1]. The decrease along these eigenvectors $\mathbf{x}_i$ is of the form λ_1/λ_i which is strictly smaller than 1 since λ_1 is the smallest eigenvalue. This means that the vector $\mathbf{z}$ becomes more aligned with $\mathbf{x}_1$ with each iteration.

In the limit, we get

$$\mathbf{z} = \frac{1}{\lambda_1}\mathbf{x}_1, \tag{3}$$

where $\mathbf{x}_1$ is a unit eigenvector and where λ_1 is the smallest eigenvalue of $\mathbf{A}$.

3.2 Implementation

```
def norm(x):
    """Return norm of input vector"""
    return np.sqrt(np.sum(x**2))


def invpower_eig(A, tol=1e-9, maxiter=1000):
    """Return the smallest eigenvalue and the cooresponding
    eigenvector, using the Inverse Power method"""

    def iterate(z):
        """Return new guess vector"""
        # Normalise z to get v
        v = z / norm(z)
        # Solve Az = v
        return sp.linalg.solve(A, v)

    def has_converged(zold, znew):
        return abs(norm(zold) - norm(znew)) < tol

    n = A.shape[0]
    zold = np.ones(n)
    znew = iterate(zold)
    count = 0
    while not has_converged(zold, znew) or count > maxiter :
        zold = znew  # store znew
        znew = iterate(zold)
        count += 1
    # Get the sign of the eigenvalue
    z = np.dot(A, znew)
    sign = np.sign(z[0] * znew[0])
    # Return the eigenvalue
    lambda1 = sign * 1.0 / norm(znew)
    return lambda1, znew
```

3.3 Testing

```
A= np.array([[3.,-1,0], [-1,2,-1] , [0 , -1 , 3]])
#A= np.array([[8.,-1,3,-1], [-1,6,2,0] , [3 , 2 , 9, 1], [-1, 0, 1, 7]])
lambda1, x1 = invpower_eig(A,tol=1.0e-9)
print 'smallest eigenvalue = ', lambda1
print 'eigenvector = ', x1 / x1[0]
```

We find $\lambda_1 = 1$ and $x_1 = (121$ as expected (see lecture 08).

3.3.1 Self study

Modify the power method code above to return the smallest eigenvalue and eigenvector when the eigenvalue is complex. Use the following matrix as an example:

$$A = \begin{pmatrix} 3 & -9 \\ 4 & -3 \end{pmatrix} \tag{4}$$

Solution: the eigenvalues are $\lambda_1 = \pm 3\sqrt{3}i$.

3.4 Discussion

Recall that convergence depends on the ratio λ_1/λ_i $(i > 1)$. Convergence is therefore slow if λ_i is close to λ_1. However we can shift all the eigenvalues by an arbitrary constant s, so the ration becomes $(\lambda_1 - s)/(\lambda_i - s)$ which can help speed up the convergence if we have a good estimate s for λ_1 [1]. An important application of this is to find all the eigenvectors of a matrix once the eigenvalues are known. The `QR Method`, introduced below, is a popular method for computing all the eigenvalues of a matrix.

A great advantage of this approach is that it does not transform the input matrix $\mathbf{A}$. It is well suited to finding the eigenvalues of a sparse matrix.

There is a similar method called the *Power method* to compute the largest eigenvalue of A. The main differences with the inverse Power method is that it solves the system

$$\mathbf{z} = \mathbf{A}\mathbf{v} \tag{5}$$

at each iteration, and that $|\mathbf{z}|$ converges to $\pm\lambda_n$, and $\mathbf{v}$ converges to $\mathbf{x}_n$.

4 Krylov subspace method

The inverse Power method can be generalised to obtain the first n eigenvalues and eigenvectors of a matrix. This is the purpose of Arnoldi's itera-

tion method, which starts from a guess vector $\mathbf{v}$ and generates the vectors $(\mathbf{A}^i\mathbf{v})_{0\leq i<n}$. These vectors form a space called a *Krylov subspace*, and orthonormalising these vectors provides a good estimate for the eigenvectors.

5 QR iteration for eigenvalues

A classic method for finding all the eigenvalues of a symmetric matrix is the QR method. It is based on the QR factorization of a symmetric matrix A, where Q is an orthogonal matrix ($Q^TQ = 1$) and R is an upper diagonal matrix.

It is easy to show that if $A = QR$, then $A_1 = RQ$ has the same eigenvalues as A. If we carry out the QR decomposition of $A_n = Q_nR_n$, and define $A_{n+1} = R_nQ_n$, we can obtain a series of matrices $(A_n)_{n\geq 1}$ based on A.

For a symmetric matrix A, the list $(A_n)_{n\geq 1}$ leads to upper diagonal matrices, whose diagonal coefficients tend towards the eigenvalues of A.

6 Scipy

The Scipy linalg module `scipy.linalg`, and its sparse version `scipy.sparse.linalg` implement several functions for eigenvalue problems including:

- `sp.linalg.eig` : returns eigenvalues and eigenvectors of a generic (real or complex) matrix
- `sp.linalg.eigh` : returns eigenvalues and eigenvectors of a symmetric or Hermitian matrix.
- `sp.sparse.linalg.eigs` : returns the first k eigenvalues and eigenvectors of a sparse matrix (Scipy version 0.15.1).

Matlab has an additional function `eigs` that returns the first k eigenvalues and eigenvectors of a general matrix.

7 Conclusions

- Numerous algorithms are available for computing the eigenvalues and eigenvectors of a matrix. Ideally the user should employ an algorithm that is optimal for the matrix at hand. For example, it should exploit the symmetry, sparsity or bandedness of the matrix.

- The QR algorithm can be used to find the eigenvalues of a symmetric matrix.
- The inverse Power method can be used to find the eigenvectors of a matrix given the eigenvalues, or to find the smallest eigenvalue and its eigenvector.
- Kryolv subspace methods can be be used to find the first k eigenvectors of a general matrix.

8 References

1. J. Kiusalaas, Numerical Methods in Engineering with Python, Cambridge University Press (2010).