Software Assignment

EE24Btech11024 - G. Abhimanyu Koushik

November 18, 2024

Introduction

An $N \times N$ matrix A is said to have an eigenvector $\mathbf{x}$ and corresponding eigenvalue λ if

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

Any multiple of an eigenvector is also an eigenvector, but we will not be considering such multiples as disticnt vectors.
Evidently equation (1) holds only if

$$\det |A - \lambda I| = 0 \tag{2}$$

which, if expanded out, is an N^{th} degree polynomial equation in λ whose roots are eigenvalues. Root searching in the characteristic equation (2) is a very poor computational method of finding eigenvalues.
The above two equations show that there is a corresponding eigenvector or every eigenvalue: If λ is set to an eigenvalue then the matrix $A - \lambda I$ is singular so it has atleast one non-zero vector in its nullspace.
A matrix is called symmetric if it is equal to its transpose.

$$A = A^{\top} \tag{3}$$

A matrix is called orthogonal if its inverse is equal to its transpose.

$$A^{\top} \cdot A = A \cdot A^{\top} = I \tag{4}$$

Multiply two orthogonal matrices gives another orthogonal matrix. The eigenvalues of symmetric values are all real. The eigenvalues of a non-symmetric matrix can be complex.

Diagonalization of a Matrx

A matrix is said to undergo Similarity transformation if it undergoes the following process.

$$A \implies Z^{-1} \cdot A \cdot Z \tag{5}$$

For some matrix Z. When a matrix undergoes similarity transform, the eigenvalues of the matrix remain unchanged.

$$\begin{aligned} \det\left|Z^{-1} \cdot A \cdot Z - \lambda I\right| &= \det\left|Z^{-1} \cdot A \cdot Z - \lambda Z^{-1} \cdot I \cdot Z\right| && (6)\\ &= \det\left|Z^{-1}\left(A - \lambda I\right) Z\right| && (7)\\ &= \det\left|Z^{-1}\right| \det|A - \lambda I| \det|Z| && (8)\\ &= \det|A - \lambda I| && (9) \end{aligned}$$

If a matrix undergoes similarity transform, the properties like symmetry, normality, unitary remain preserved.. For a matrix A, let the eigen values be $\lambda_1, \lambda_2, \ldots, \lambda_N$, then we can write the following equation.

$$X \cdot A = \operatorname{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_N\right) \cdot X \tag{10}$$

$$\tag{11}$$

where RHS is a diagonal matrix with entries $\lambda_1, \lambda_2, \ldots, \lambda_N$, if we multiply with X^{-1} on both side we will get

$$X \cdot A \cdot X^{-1} = \operatorname{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_N\right) \tag{12}$$

This is the basis for Jacobian Transformation

1 Jacobian Transformation

The Jacobian transformation consists of a sequence of orthogonal similarity transformation (X is orthogonal) to tranform matrix A into a diagonal matrix. Each orthogonal transformation is designed to annilate one off-diangonal element. Successive tranformations might undo previously set zeroes but the off-diagonal elements nevertheless get smaller and smaller, until the matrix is diagonal to machine precision.

1.1 Mathematical Background

Let A be a symmetric matrix. The Jacobian method aims to diagonalize A into:

$$A = V \Lambda V^{\top} \tag{13}$$

where Λ is a diagonal matrix of eigenvalues, and V is an orthogonal matrix of eigenvectors. The rotation matrix P_{pq} is defined as:

$$P_{pq} = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \iddots & \vdots \\ 0 & \cdots & c & s & \cdots & 0 \\ 0 & \cdots & -s & c & \cdots & 0 \\ \vdots & \iddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{14}$$

Here all diagonal elements are 1 except for the two elements c in rows (and columns) p and q. All off-diagonal elements are 0 except for s and $-s$. The numbers c and s are the cosine and sine of rotation angle ϕ, so $c^2 + s^2 = 1$.
A plane rotation such as P_{pq} in equation (14) is used to transform matrix A according to

$$A' = P_{pq}^{\top} A P_{pq} \tag{15}$$

Now, $P_{pq}^{\top} A$ changes only rows p and q of A., while AP_{pq} change only the columns p and q. Thus the changed elements of A in equation (15) are only in p and q rows and columns indicated as below

$$A' = \begin{pmatrix} & \cdots & a'_{1p} & \cdots & a'_{1q} & \cdots & \\ \vdots & & \vdots & & \vdots & & \vdots \\ a'_{p1} & \cdots & a'_{pp} & \cdots & a'_{pq} & \cdots & a'_{pn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ a'_{q1} & \cdots & a'_{qp} & \cdots & a'_{qq} & \cdots & a'_{qn} \\ \vdots & & \vdots & & \vdots & & \vdots \\ & \cdots & a'_{np} & \cdots & a'_{nq} & \cdots & \end{pmatrix} \tag{16}$$

Multiplying equation (15) and using the symmetry of A will get us the equations

$$a'_{rp} = ca_{rp} - sa_{rq} \tag{17}$$

$$a'_{rq} = ca_{rq} + sa_{rp} \tag{18}$$

$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq} \tag{19}$$

$$a'_{qq} = c^2 a_{pp} + s^2 a_{qq} + 2sca_{pq} \tag{20}$$

$$a'_{pq} = \left(c^2 - s^2\right) a_{pq} + sc\left(a_{pp} - a_{qq}\right) \tag{21}$$

for $r \neq s$. Accordingly, if we set $a'_{pq} = 0$, equation (21) gives the following expression

$$\theta = \cot 2\phi = \frac{c^2 - s^2}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}} \tag{22}$$

If we let $t = \frac{s}{c}$, the definition of θ can be written as

$$t^2 + 2t\theta - 1 = 0 \tag{23}$$

We get t as the following value if we consider the smaller root

$$t = \frac{sgn\,(\theta)}{|\theta| + \sqrt{\theta^2 + 1}} \tag{24}$$

It now follows that

$$c = \frac{1}{\sqrt{t^2 + 1}} \tag{25}$$

$$s = \frac{t}{\sqrt{t^2 + 1}} \tag{26}$$

Once we substitute the c and s in the matrix and multiply as given in equation (15), the a'_{pq} gets nulled. We not do the same thing for all the other elements as well. Since the matrix is symmetric, there is no need to iterate through every off-diagonal element. Just going through the lower triangular indexed elements would be enough as it will automatically null upper triangular ones as well.
One can see the convergence of Jacobi method by considering sum of square of off-diagonal elements.

$$S = \sum_{r \neq s} |a_{rs}|^2 \tag{27}$$

The new sum after an iteration will be

$$S' = S - 2\left|a_{pq}\right|^2 \tag{28}$$

Since the sequence is monotonically decreasing and is bounded below by 0, it coverges to 0.
Eventually, one obtains a matrix D which is diagonal to machine precision. The elements of the matrix D will be the eigenvalues of A since

$$D = V^\top A V \tag{29}$$

where

$$V = P_1 \cdot P_2 \cdot P_3 \cdots \tag{30}$$

One set of $\frac{n(n-1)}{2}$ set of Jacobi rotations is called a sweep, nulling every lower triangular element once. We will implement 8 sweeps so as to make sure the elements are nulled completely and then stop.

1.2 Code

Below is the implementation of the Jacobian transformation in C:

```
double **jacobian(double **A, int dim) {
    double **jacobian = copyMat(A, dim, dim);
    double threshold = 1e-10;
```

Initialising jacobian matrix (The diagonal matrix), and keep a threshold.

```
    for (int sweep = 0; sweep < 8; sweep++) {
        for (int p = 1; p < dim; p++) {
            for (int q = 0; q < p; q++) {
                if (fabs(jacobian[p][q]) > threshold) {
```

Applying jacobian transform 8 times, and iterating through each lower triangular element and applying transformation if its absolute value is greater than threshold (basically greater than 0).

```
                    double theta = (jacobian[q][q] - jacobian[p][p]) / (2 * jac
                    double sgn = (theta != 0) ? (theta > 0 ? 1 : -1) : 0;
                    double t = sgn / (fabs(theta) + sqrt(theta * theta + 1));
                    double c = 1 / sqrt(t * t + 1);
                    double s = t / sqrt(t * t + 1);
```

Calculatint the values of c,s,t and θ.

```
                    double **Ppq = identity(dim);
                    Ppq[p][p] = c;
                    Ppq[p][q] = s;
                    Ppq[q][p] = -s;
                    Ppq[q][q] = c;
```

Forming the matrix P_{pq}.

```
                    jacobian = Matmul(jacobian, Ppq, dim, dim, dim);
                    double **PpqT = transposeMat(Ppq, dim, dim);
                    jacobian = Matmul(PpqT, jacobian, dim, dim, dim);
```

Assigning new A as $P_{pq}^{\top} A P_{pq}$

```
                    freeMat(Ppq, dim);
                    freeMat(PpqT, dim);
                }
```

Free the matrices

```
                else{
                 jacobian[p][q] = 0;
                }
            }
        }
    }
    return jacobian;
}
```

Assigning the values 0 if they are less than threshold, and return the diagonal matrix

2 Hessenberg Reduction

Jacobian is a very inefficient way of finding the eigenvalues as its time complexity is $O\left(n^4\right)$ and its use is very limited as it can only be used to find the eigenvalues of real and symmetric matrices. For finding the eigen values of a non-symmetric, the best way is to reduce the matrix to a suitable form and then apply QR decomposition algorithm to that, which the will be explained later. This is suitable for turns out to be Hessenberg form. Turning a matrix into hessenberg form is a process of time complexity $O\left(n^3\right)$, and then applying QR decomposition on hessenberg is of time complexity $O\left(n^2\right)$. Hence the actual time complexity of order $O\left(n^3\right)$.

2.1 Mathematical Background

We say a matrix A is in hessenberg form if it is in form shown below

$$H = \begin{pmatrix} \times & \times & \times & \cdots & \times \\ \times & \times & \times & \cdots & \times \\ 0 & \times & \times & \cdots & \times \\ 0 & 0 & \times & \cdots & \times \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \times \end{pmatrix} \tag{31}$$

We will use householder method to reduce any matrix into hessenberg form.
It reduces an $n \times n$ matrix to hessenberg form by $n-2$ orthogonal trasformations. Each transformations annihilates the required part of a whole column at a time

rather than element wise elimination. The basic ingredient for a house holder matrix is P which is in the form

$$P = I - 2\mathbf{w}\mathbf{w}^\top \tag{32}$$

where w is a vector with $|w|^2 = 1$. The matrix P is orthogonal as

$$\begin{aligned} P^2 &= (I - 2\mathbf{w}\mathbf{w}^\top) \cdot (I - 2\mathbf{w}\mathbf{w}^\top) && (33)\\ &= I - 4\mathbf{w}\mathbf{w}^\top + 4\mathbf{w} \cdot (\mathbf{w}^\top \mathbf{w}^\top) \cdot \mathbf{w}^\top && (34)\\ &= I && (35) \end{aligned}$$

Therefore, $P = P^{-1}$ but $P = P^\top$, so $P = P^\top$
We can rewrite P as

$$P = I - \frac{\mathbf{u}\mathbf{u}^\top}{H} \tag{36}$$

where the scalar H is

$$H = \frac{1}{2}|\mathbf{u}|^2 \tag{37}$$

Where $\mathbf{u}$ can be any vector. Suppose $\mathbf{x}$ is the vector composed of the first column of A. Take

$$\mathbf{u} = \mathbf{x} \mp |\mathbf{x}|\,\mathbf{e}_1 \tag{38}$$

Where $\mathbf{e}_1 = \begin{pmatrix} 1 & 0 & \dots \end{pmatrix}^\top$, we will take the choice of sign later. Then

$$\begin{aligned} P \cdot \mathbf{x} &= \mathbf{x} - \frac{\mathbf{u}}{H} \cdot (\mathbf{u} \mp |\mathbf{x}|\,\mathbf{e}_1)^\top \cdot \mathbf{x} && (39)\\ &= \mathbf{x} - \frac{2\mathbf{u}\left(|x|^2 \mp |x|\,x_1\right)}{2\,|x|^2 \mp |x|\,x_1} && (40)\\ &= \mathbf{x} - \mathbf{u} && (41)\\ &= \mp |\mathbf{x}|\,\mathbf{e}_1 && (42) \end{aligned}$$

To reduce a matrix A into Hessenberg form, we choose vector $\mathbf{x}$ for the first householder matrix to be lower $n-1$ elements of the first column, then the lower $n-2$

elements will be zeroed.

$$P_1 \cdot A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & p_{11} & p_{12} & \cdots & p_{1n} \\ 0 & p_{21} & p_{22} & \cdots & p_{2n} \\ 0 & p_{31} & p_{32} & \cdots & p_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix} \begin{pmatrix} a_{00} & \times & \times & \cdots & \times \\ a_{10} & \times & \times & \cdots & \times \\ a_{20} & \times & \times & \cdots & \times \\ a_{30} & \times & \times & \cdots & \times \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n0} & \times & \times & \cdots & \times \end{pmatrix} \tag{43}$$

$$= \begin{pmatrix} a'_{00} & \times & \times & \cdots & \times \\ 0 & \times & \times & \cdots & \times \\ 0 & \times & \times & \cdots & \times \\ 0 & \times & \times & \cdots & \times \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \times & \times & \cdots & \times \end{pmatrix} \tag{44}$$

Now if we multiply the matrix P_1A with P_1, the eigenvalues will be conserved as it is a similarity transformation.
Now we choose the vector $\mathbf{x}$ for the householder matrix to be the bottom $n-2$ elements of the second column, and from it construct the P_2

$$P_2 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & p_{22} & \cdots & p_{2n} \\ 0 & 0 & p_{32} & \cdots & p_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & p_{n2} & \cdots & p_{nn} \end{pmatrix} \tag{45}$$

Now if do similarity transform PAP, we will zero out the $n-3$ elements in second column. If we continue this pattern we will get the hessenberg form of a the matrix A.

2.2 Code

```
double complex **makehessberg(double complex **A, int dim) {
    double complex **toreturn = copyMat(A, dim, dim);
```

Copying inputted matrix to toreturn;

```
    for (int k = 0; k < dim - 2; k++) {
        int u_dim = dim - k - 1;
        double complex **u = createMat(u_dim, 1);
```

```
for (int i = 0; i < u_dim; i++) {
    u[i][0] = toreturn[k + 1 + i][k];
}
```

Making a loop for $n - 2$ iteration and initializing $\mathbf{u}$ to the required subcolumn vector of A

```
double complex rho;
if(cabs(u[0][0])!=0){
 rho = - u[0][0] / cabs(u[0][0]);
}
else{
 rho = 1;
}
u[0][0] += -1 * rho * Matnorm(u, u_dim);
```

Substituting $\mathbf{u} = \mathbf{x} - |\mathbf{x}|\,\mathbf{e}_1$

```
double norm_u = Matnorm(u, u_dim);
if (norm_u != 0.0) {
     for (int i = 0; i < u_dim; i++) {
        u[i][0] /= norm_u;
    }
}
else{
 continue;
}
```

Making $\mathbf{u}$ an unit vector and if norm of $\mathbf{u} = 0$, skipping the process as the column is already in required form

```
double complex **Hk = identity(dim);
double complex **u_transpose = transposeMat(u, u_dim, 1);
double complex **u_ut = Matmul(u, u_transpose, u_dim, 1, u_dim);

for (int i = 0; i < u_dim; i++) {
    for (int j = 0; j < u_dim; j++) {
        Hk[k + 1 + i][k + 1 + j] -= 2 * u_ut[i][j];
    }
}
```

Forming the Householder matrix

```
        toreturn = Matmul(Hk, toreturn, dim, dim, dim);
        toreturn = Matmul(toreturn, Hk, dim, dim, dim);
```

Using similarity trasformation with Householder matrix to A.

```
        freeMat(u, u_dim);
        freeMat(u_transpose, 1);
        freeMat(u_ut, u_dim);
        freeMat(Hk, dim);
    }
    return toreturn;
}
```

Freeing the matrices and returning the hessenberg form of A.

3 QR Decomposition Algorithm

3.1 Mathematical Background

In this algorithm, we decompose matrix given in Hessenberg form to two matrices Q and R such that Q is an orthogonal matrix and R is an upper triangular matrix. Then we assign the new matrix A' to be $A' = RQ$, and we do this iteratively. Theoritically, as the number of iterations go to infinite, the matrix A' will converge to an upper triangular matrix whose diagonal elements are the eigenvalues of A. There will be a minor problem in this method when the entries are real while the eigenvalues are complex, we will solve this issue shortly. The eigenvalues of the matrix A will not change because of the following

$$A = QR \tag{46}$$
$$R = Q^\top A \tag{47}$$
$$A' = RQ \tag{48}$$
$$A' = Q^\top AQ \tag{49}$$

As the matrix A is undergoing similarity transformation, the eigenvalues will not change.
The rate of covergence of A depends on the ratio of absolute values of the eigenvalues. That is, if the eigenvalues are $|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \cdots \geq |\lambda_n|$ then, the elements of A_k below the diagonal to converge to zero like

$$\left|a_{ij}^{(k)}\right| = O\left(\left|\frac{\lambda_i}{\lambda_j}\right|^k\right) \; i > j \tag{50}$$

3.2 Givens Rotations

The QR decomposition is implemented using the Givens rotation technique. This approach is robust and numerically stable, making it ideal for QR decomposition, especially in iterative methods like eigenvalue computations. It is every similar to Jacobian Transformation. We define a rotation matrix P, to zero out the elements which are non-diagonal, since the matrix which we are dealing is a Hessenberg matrix, we need to zero out the elements which are just below the diagonal elements. This shows the significance of Hessenberg form. Initially we zeroed complete columns at once. If we were to apply givens to each and every element, it will take $\frac{n(n-1)}{2}$ orthogonal rotations. Now, computing P and the new A, costs in order $O\left(n^2\right)$. The whole process together would be in cost of order $O\left(n^4\right)$. But now since we only need to do $n-1$ orthogonal rotations, the order becomes $O\left(n^3\right)$. The rotation matrix P is defined as

$$P = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \iddots & \vdots \\ 0 & \cdots & c & s & \cdots & 0 \\ 0 & \cdots & -s & c & \cdots & 0 \\ \vdots & \iddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{51}$$

Where the value of c and s are

$$c = \frac{\overline{x_{i,i}}}{\sqrt{\left|x_{i,i}\right|^2 + \left|x_{i,i+1}\right|^2}} \tag{52}$$

3.2.1 Givens Rotation Matrix

A Givens rotation matrix is a 2x2 orthogonal matrix used to zero out specific elements of a matrix through rotation. The Givens rotation matrix G is defined as:

$$G(\theta) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for an angle θ, chosen such that the element below the diagonal in the matrix A is eliminated.

3.2.2 Steps in the QR Decomposition Algorithm

The algorithm proceeds as follows:

1. Initialize A as the input matrix, $Q = I$ (identity matrix), and $R = A$.

2. Iterate through the columns of A, and for each pair of elements in the column, compute the necessary Givens rotation matrix to zero out the subdiagonal element.

3. Apply the Givens rotation to both A and Q in each iteration.

4. The resulting matrix R will be upper triangular, and the product of all the Givens rotation matrices accumulated in Q will form the orthogonal matrix Q.

The implementation in `qr.c` handles the iterative application of the Givens rotations and the accumulation of the orthogonal matrix Q.

3.3 Rayleigh Quotient Iteration (rayleigh.c)

Rayleigh Quotient iteration is an iterative method used to find eigenvalues of a matrix, which can be especially useful when computing the dominant eigenvalue. The method uses the Rayleigh quotient as an approximation for the eigenvalue:

$$\lambda = \frac{v^T A v}{v^T v}$$

where v is the current vector approximation to the eigenvector. The basic idea of the Rayleigh Quotient iteration is to refine the approximation by solving for the eigenvector corresponding to the largest eigenvalue of a shifted matrix.

3.3.1 Steps in the Rayleigh Quotient Iteration Algorithm

The steps for Rayleigh Quotient iteration are as follows:

1. Start with an initial guess for the eigenvector v_0.

2. Compute the Rayleigh quotient $\lambda_0 = \frac{v_0^T A v_0}{v_0^T v_0}$.

3. Shift the matrix A by subtracting $\lambda_0 I$ from it, forming the shifted matrix $A - \lambda_0 I$.

4. Solve the linear system $(A - \lambda_0 I)v_1 = v_0$ for the new vector v_1.

5. Normalize v_1 and repeat the process, refining both the eigenvalue and eigenvector with each iteration.

The implementation of this method in `rayleigh.c` allows for the iterative calculation of eigenvalues with high accuracy.

3.4 Code Explanation

The following sections explain the contents of the `qr.c` and `rayleigh.c` files.

3.4.1 qr.c

The `qr.c` file contains the implementation for performing QR decomposition using Givens rotations. The main functions in the file are:

- `givens_rotation(double* A, int m, int n, int i, int j)`: This function computes the Givens rotation matrix to zero out the element $A[i, j]$ in the matrix A.
- `qr_decomposition(double* A, double* Q, double* R, int m, int n)`: This function applies the Givens rotations iteratively to compute the QR decomposition of matrix A. It updates matrices Q and R accordingly.

3.4.2 rayleigh.c

The `rayleigh.c` file implements the Rayleigh Quotient iteration method for finding eigenvalues. The main functions in the file are:

- `rayleigh_quotient(double* A, double* v, int n)`: This function computes the Rayleigh quotient $\lambda = \frac{v^T A v}{v^T v}$.
- `rayleigh_iteration(double* A, double* v, int n, int max_iter)`: This function performs the Rayleigh Quotient iteration, refining the eigenvalue and eigenvector with each iteration.

The code in `rayleigh.c` iteratively refines both the eigenvalue and eigenvector using the Rayleigh quotient and shifts to converge to the true eigenvalue.

3.5 Conclusion

The QR decomposition algorithm, implemented in `qr.c` using Givens rotations, is a stable and efficient method for decomposing matrices. The Rayleigh Quotient iteration, implemented in `rayleigh.c`, provides an iterative technique for refining eigenvalue approximations. Both algorithms play crucial roles in numerical linear algebra and are essential tools for solving systems of linear equations and finding eigenvalues.

4 QR with Rayleigh Quotient method

(Content derived from qr.c, explaining its importance for eigenvalue convergence.)

References

1. W. H. Press et al., *Numerical Recipes in C: The Art of Scientific Computing*.
2. P. Arbenz, *Lecture Notes on Numerical Linear Algebra*. Link.