# Eigen Values of a Matrix

## Objective -

To find the eigen values of a matrix

A = 1 2 3 4

2 1 2 2

3 2 1 3

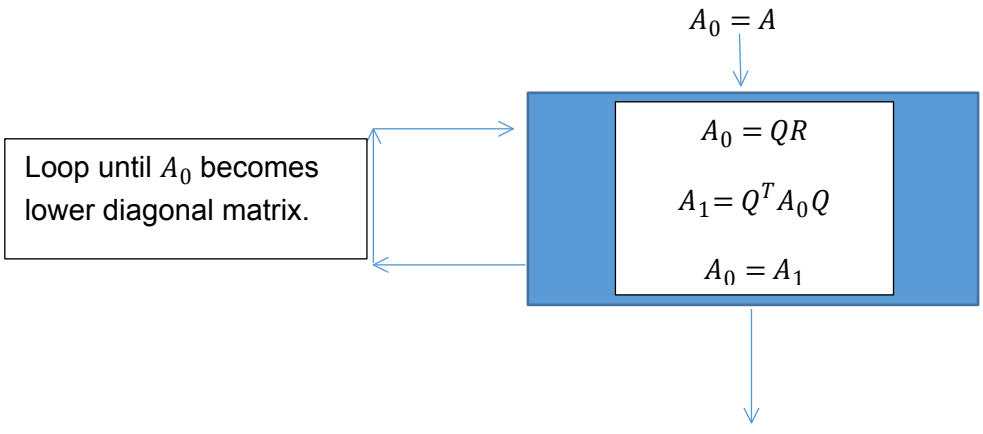
4 2 3 1

By using -

1. QR iteration using Householder's method
2. Find Heisenberg matrix and then use QR iteration method for Heisenberg matrix

## QR iteration using Householder's method -

Here is the algorithm for this method -




Matrix Q of a matrix A can be found by following code -

```c
/*Forms kth matrix in Householder method*/
int Q_form(double *a,int k,double *out,int n) {
    int temp=n-k;
    double *v=(double *)malloc(sizeof(double)*(temp));
    for (int i=k;i<n;i++) v[i-k]=a[n*i+k];
    double temp_norm=norm_sq(v,temp);
    v[0]=v[0]+my_sign(v[0])*sqrt(temp_norm);
    temp_norm=norm_sq(v,temp);
    if (my_abs(temp_norm)<=ZERO) return -1;
    double *F=(double *)malloc(sizeof(double)*temp*temp);
    double *temp1=(double *)malloc(sizeof(double)*(temp));
    double *temp2=(double *)malloc(sizeof(double)*temp);
    for (int i=0;i<temp;i++) temp1[i]=(-2)*v[i];
    transpose(v,temp,1,temp2);
    matrix_multiply(temp1,temp,1,temp2,1,temp,F);
    for (int i=0;i<temp;i++) {
        for (int j=0;j<temp;j++) F[temp*i+j]/=temp_norm;
    }
    for (int i=0;i<temp;i++) F[temp*i+i]+=1;
    for (int i=0;i<n;i++) {
        for (int j=0;j<n;j++) {
            if (i<k) {
                if (i==j) out[n*i+j]=1.0;
                else out[n*i+j]=0.0;
            }
            else {
                if (j<k) out[n*i+j]=0.0;
                else out[n*i+j]=F[temp*(i-k)+j-k];
            }
        }
    }
    free(temp1);
    free(temp2);
    free(F);
    free(v);
    return 0;
}
```

```c
/*does QR decomposition of a and stores Q in matrix Q
return 0 if succeded
else return -1*/
int householder(double *a,int n,double *Q) {
    double *Q0=(double *)malloc(sizeof(double)*n*n);
    double *ac=(double *)malloc(sizeof(double)*n*n);
    equate_array(ac,a,n*n);
    my_eye(Q0,n);
    for (int k=0;k<n;k++) {
        double *temp=(double *)malloc(sizeof(double)*n*n);
        double *temp2=(double *)malloc(sizeof(double)*n*n);
        if (Q_form(ac,k,temp,n)==-1) return -1;
        matrix_multiply(temp,n,n,ac,n,n,temp2);
        equate_array(ac,temp2,n*n);
        double *temp1=(double *)malloc(sizeof(double)*n*n);
        transpose(temp,n,n,temp1);
        matrix_multiply(Q0,n,n,temp1,n,n,Q);
        equate_array(Q0,Q,n*n);
        free(temp1);
        free(temp);
        free(temp2);
    }
    free(Q0);
    free(ac);
    return 0;
}
```

Here is the implication of algorithm -

```
/*stores eigen values of a in eig_arr array using householder's method
return number of iterations if succeded
else return -1*/
int house_eigenvalues(double *a,int n,double *eig_arr) {
    double *ac0=(double *)malloc(sizeof(double)*n*n);
    double *ac=(double *)malloc(sizeof(double)*n*n);
    equate_array(ac0,a,n*n);
    int it=0;
    while (check(ac0,n)!=1) {
        it+=1;
        double *Q_ac=(double *)malloc(sizeof(double)*n*n);
        if (householder(ac0,n,Q_ac)==-1) return -1;
        double *Q_acT=(double *)malloc(sizeof(double)*n*n);
        transpose(Q_ac,n,n,Q_acT);
        double *temp1=(double *)malloc(sizeof(double)*n*n);
        matrix_multiply(Q_acT,n,n,ac0,n,n,temp1);
        matrix_multiply(temp1,n,n,Q_ac,n,n,ac);
        equate_array(ac0,ac,n*n);
        free(temp1);
        free(Q_acT);
        free(Q_ac);
        if (it==MAX_IT) return -1;
    }
    for (int i=0;i<n;i++) eig_arr[i]=ac0[n*i+i];
    free(ac0);
    free(ac);
    return it;
}
```

**Result -**

Eigen values are 9.1581, -3.00, -1.7115, -0.4466

Number of iterations - 64

Time taken - 0.001623s

# Using Heisenberg matrix -

Property of an upper Heisenberg matrix -

$$A_{ij} = 0 \quad if\ i > j + 1$$

Here is the code for finding Heisenberg matrix of A -

```c
/*Makes out as heisenberg matrix corresponding to matrix a of size nxn*/
int heisenberg(double *a,int n, double *out) {
    double *a0=(double *)malloc(sizeof(double)*n*n);
    equate_array(a0,a,n*n);
    for (int i=0;i<n-2;i++) {
        double *H=(double *)malloc(sizeof(double)*n*n);
        double *temp1=(double *)malloc(sizeof(double)*n*n);
        double *temp2=(double *)malloc(sizeof(double)*n*n);
        if (H_form(a0,i,H,n)==-1) return -1;
        transpose(H,n,n,temp1);
        matrix_multiply(temp1,n,n,a0,n,n,temp2);
        matrix_multiply(temp2,n,n,H,n,n,out);
        equate_array(a0,out,n*n);
        free(H);
        free(temp1);
        free(temp2);
    }
    free(a0);
    return 0;
}
```

```c
/*Forms H matrix required in heisenberg's algorithm*/
int H_form(double *a,int k,double *out,int n) {
    int temp=n-k-1;
    double *v=(double *)malloc(sizeof(double)*(temp));
    for (int i=k+1;i<n;i++) v[i-k-1]=a[n*i+k];
    double temp_norm=norm_sq(v,temp);
    v[0]=v[0]+my_sign(v[0])*sqrt(temp_norm);
    temp_norm=norm_sq(v,temp);
    if (my_abs(temp_norm)<=ZERO) return -1;
    double *F=(double *)malloc(sizeof(double)*temp*temp);
    double *temp1=(double *)malloc(sizeof(double)*(temp));
    double *temp2=(double *)malloc(sizeof(double)*temp);
    for (int i=0;i<temp;i++) temp1[i]=(-2)*v[i];
    transpose(v,temp,1,temp2);
    matrix_multiply(temp1,temp,1,temp2,1,temp,F);
    for (int i=0;i<temp;i++) {
        for (int j=0;j<temp;j++) F[temp*i+j]/=temp_norm;
    }
    for (int i=0;i<temp;i++) F[temp*i+i]+=1;
    for (int i=0;i<n;i++) {
        for (int j=0;j<n;j++) {
            if (i<=k) {
                if (i==j) out[n*i+j]=1.0;
                else out[n*i+j]=0.0;
            }
            else {
                if (j<=k) out[n*i+j]=0.0;
                else out[n*i+j]=F[temp*(i-k-1)+j-k-1];
            }
        }
    }
    free(temp1);
    free(temp2);
    free(F);
    free(v);
    return 0;
}
```

**Result -**

Eigen values are 9.1581, -3.00, -1.7115, -0.4466

Number of iterations - 65

Time taken - 0.001062s