Homework 5

October 9, 2017

1 Task 1

We implemented the QR algorithm with Rayleigh shifts and deflations, as indicated (see Appendix for the code). For the Rayleigh shift we chose (as suggested in litterature) the last diagonal element of the matrix A (and of the submatrix due to the deflation). In fact, when trying the another element, the algorithm doesn't converge as well (sometimes not at all after 1000 iterations).

We handled the deflation in the following way: once the off-diagonal elements adjacent to the last diagonal element were close to zero (smaller than 1.e-8 in absolute value), we considered an an eigenvalue as found, namely the diagonal element. We saved it, set the off-diagonal elements to 0, and repeated the iterations, but with the submatrix without the last row and column.

The algorithm works mostly well for dimensions at least up to 50. Then it converges for around 100 iterations. For n=5 it converges for around 10 iterations. It happens sometimes (more often for larger dimensions) that it does not converge (well or at all). This could be caused by the eigenvalues of the matrix, if they are close to each other.

To be able to start the algortihm with a tridiagonal matrix, we wrote a method for the Householder reduction to a Hessenberg matrix (tridiagonal for a symmetric matrix A).

Something seems to happen to the matrix the first iterations, before any eigenvalue is found. The first diagonal element becomes big and does not change much while all the other values are found. But this value is not the last eigenvalue, whereas the algorithm should give that. So instead we take the diagonal element of the tridiagonal matrix we started out with as the last eigenvalue, when all the others are found (and this corresponds well).

2 Task 2

To proof that (A-λI) has rank at least m-1 if A$\in$ R^{mxm} we can use that A is a tridiagonal and symmetric matrix with all its sub- and superdiagonal entries nonzero. What we can use for the proof is that we know: a matrix has always rank greater or equal than it's submatrices. If we take the submatrix of our tridiagonal submatrix we will get a matrix with full rank (m-1), because if we take the submatrix S by deleting the first row and the last column: A = $\begin{pmatrix} d_1 & c_1 & 0 & ... & 0 \\ a_1 & d_2 & c_2 & \ddots & \vdots \\ 0 & a_2 & d_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{m-1} \\ 0 & \dots & 0 & a_{m-1} & d_m \end{pmatrix}$

it will look like this: S= $\begin{pmatrix} a_1 & d_2 & c_2 & 0 \\ & a_2 & d_3 & c_{m-2} \\ & & \ddots & d_{m-1} \\ 0 & & & a_{m-1} \end{pmatrix}$

This is a upper triangular matrix where we already know that all diagonal entries are nonzero. Thus the matrix has full rank. But we also know that the (A-λI) can't have full rank because this is the condition that λ is an eigenvalue of A. With this we can say for sure A-λI has the rank m-1 for each eigenvalue. If this applies for all eigenvalues they should all be distinct otherwise the matrix A-λI wouldn't have rank m-1 for each eigenvalue.

3 Task 3

With the help of the bisection algorithm we can find eigenvalues of a tridiagonal matrix in a certain interval (in our case). The bisection method can find the roots of an function very easy.

With help of task 2 we already know that a tridiagonal matrix has only distinct eigenvalues. And we can take advantage of the property that the eigenvalues of A^k interlace with the eigenvalues of A^{k+1}. A^k are the submatrices with k= 1,2,...m of the tridiagonal matrix A(mxm). With this fact we can count the number of eigenvalues which are smaller than a. If we do this also for another value b and subtract a from b, then we know how many eigenvalues are in the interval [a,b). Thus we can find with the bisection method the roots of the characteristic polynom in the interval [a,b).

The algorithm is attached.

4 Appendix

4.1 Task 1 – code

```
In [1]: from scipy import *
        from pylab import *
        import sys

        def hessenberg(A):
            """
            Reduces a matrix A to Hessenberg form. If A is hermitian (symmetric),
            the Hessenberg form will be tridiagonal.
            """
            m = shape(A)[0]
            for k in range(m-2):    #Householder reflection
                x = A[k+1:, k]
                v_k = sign(x[0])*array([norm(x, 2)]+(len(x)-1)*[0.]) + x
                v_k = v_k/norm(v_k, 2)
                b = dot(v_k,A[k+1:,k:])
                A[k+1:,k:] -= 2*outer(v_k,b)
                c = dot(A[0:,k+1:],v_k.T)
                A[0:,k+1:] -= 2*outer(c,v_k)
            #A[abs(A) < 1.e-15] = 0 #will it disturb anything?
            return A
```

```
def diag_qr(A):
    '''
    Diagonalizes a symmetric matrix A.
    Returns the diagonal matrix, its diagonal (the eigenvalues of A)
    and the number of iterates needed.
    '''
    A = hessenberg(A)
    A_0 = A
    n = shape(A_0)[0]
    eigvals = zeros(n)
    maxit = 1000
    for i in range(maxit):
        mu_i = A_0[n-1,n-1] #why this choice
        q_i, r_i = qr(A_0-mu_i*eye(n))
        A_0 = r_i@q_i + mu_i*eye(n)
        if abs(A_0[n-1,n-2]) < 1.e-15:
            eigvals[len(eigvals)-n] = A_0[n-1,n-1]
            A_0[n-2,n-1] = 0
            A_0[n-1,n-2] = 0
            A[n-1,:n] = A_0[n-1,:n]
            A[:n,n-1] = A_0[:n,n-1]
            A_0 = A_0[:n-1,:n-1]
            n -= 1
            if n == 1:
                eigvals[len(eigvals)-1] = A[0,0] #it does not work with A_(
                break
    A[abs(A) < 1.e-15] = 0 #will it disturb anything?
    return A, eigvals, i
```

4.2 Task 3

```
In [26]: import numpy as np
         import scipy.linalg as sl

         class bisection:
             '''

             return: eigenvalues which are in the interval
             '''
             def __init__(self, A):
                 '''
                 Initialize the matrix

                 :param A: tridiagonalzed matrix as an numpy array
                 '''
```

```
        self.A = A
    def eigenvalues(self, interval):
        '''
        Creates a list of eigenvalues depending on the interval which is g

        :param interval: tuple of two integers (integer 1 must be smaller
        :return eig: list of all eigenvalues which are in the interval
        '''
        eig = []
        right = interval[1]
        left = interval[0]
        num_of_eig = self.number_of_eigenvalues(right)[1] - self.number_of
        print(num_of_eig)
        while num_of_eig > 0:
            midpoint = (left+right)/2
            num_of_midpoint = self.number_of_eigenvalues(midpoint)[1]
            num_of_right = self.number_of_eigenvalues(right)[1]
            while num_of_midpoint != num_of_right:
                #print(midpoint)
                left = midpoint
                midpoint = (left+right)/2
                num_of_midpoint = self.number_of_eigenvalues(midpoint)[1]
            if abs(self.number_of_eigenvalues(midpoint)[0]) < 1.e-9:
                eig.append(midpoint)
                num_of_eig -= 1
                right = left
                left = interval[0]
            else:
                right = midpoint
        return eig

    def number_of_eigenvalues(self, x):
        '''
        counts the number of sign changes and delivers the solution of the
        characteristic polynom function

        :param x: integer
        return p_i, num:solution of the characteristic polynom for x ,
                        number of eigenvalues in the interval between -in
        '''
        m = len(self.A)
        num = 0
        p_previous = self.A[0][0]-x
        if p_previous < 0:
                num += 1
        p_beforeprevious = 1
        for i in range(1,m):
            p_i = (self.A[i][i]-x)*p_previous - self.A[i-1][i]**2*p_before
```

```
                if p_i > 0 and p_previous <= 0 or p_i < 0 and p_previous >= 0
                    num += 1
                p_beforeprevious = p_previous
                p_previous = p_i
            return (p_i, num)
```

```
In [21]: # Creates an random tridiagonal matrix
         A = np.random.rand(5,5)
         A = (np.transpose(A) + A)
         A = sl.hessenberg(A)
         print(A)
```

```
[[  1.28883871e+00  -2.12316534e+00  -1.33221459e-16   1.14291149e-16
    2.79094981e-17]
 [ -2.12316534e+00   2.45608333e+00   2.26187766e+00   1.48004130e-16
   -5.34569700e-16]
 [  0.00000000e+00   2.26187766e+00   1.78650249e+00  -9.45781694e-01
   -6.22626018e-17]
 [  0.00000000e+00   0.00000000e+00  -9.45781694e-01  -8.36353820e-02
    5.64818818e-03]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   5.64818818e-03
    9.67046104e-01]]
```

```
In [22]: # the eigenvalues of A and the eigenvectors
         np.linalg.eig(A)
```

```
Out[22]: (array([ 5.19201163, -1.35248758, -0.15750991,  1.76575708,  0.96706404]),
          array([[  4.05423746e-01,   4.64903112e-01,   4.36305628e-01,
                   -6.55076780e-01,   2.91450679e-03],
                 [ -7.45320660e-01,   5.78363253e-01,   2.97221338e-01,
                    1.47147349e-01,   4.41705810e-04],
                 [ -5.20966393e-01,  -5.37461115e-01,   6.61102529e-02,
                   -6.59812870e-01,   2.44499136e-03],
                 [  9.33954037e-02,  -4.00619074e-01,   8.46704331e-01,
                    3.37436442e-01,   3.17473311e-03],
                 [  1.24856596e-04,   9.75528802e-04,  -4.25265200e-03,
                    2.38622553e-03,   9.99987627e-01]]))
```

```
In [27]: # Test of the bisection method
         bis = bisection(A)
         print(bis.eigenvalues((-2,3)))
```

```
4
[1.765757078945171, 0.9670640355527409, -0.15750991461146854, -1.352487582895678]
```