Instructor: Emre Mengi

Math 304 (Spring 2010)

Study Guide for Weeks 6-7
Homework 5 concerns the topics listed below.

- The inverse and Rayleigh iterations (Watkins 5.3)
- Hessenberg matrix; recall that before applying the QR algorithm the matrix whose eigenvalues are sought is reduced into an Hessenberg matrix by applying orthogonal similarity transformations (Watkins 5.5)
- The QR algorithm without shifts (Watkins - 5.6, Fausett - 5.3)
- The QR algorithm with shifts, in particular with Rayleigh and Wilkinson shifts (Watkins - 5.6, Fausett - 5.3)
- The equivalence of the QR algorithm and simultaneous iteration (Watkins - 6.2, we did not discuss this in detail in class, but see question **2.** below. That question is self-contained and describes the simultaneous iteration.)
- The bisection, Newton's and secant methods for finding the roots of a univariate function

Homework 5 (due on April 16th, Friday by 4pm)
In Matlab question (questions **2**, **4**, **5**) attach the Matlab outputs and print-outs of the Matlab routines that you implemented.

1. The QR algorithm is one of the standard approaches to compute the eigenvalues of a matrix $A \in \mathbf{R}^{n\times n}$. Below a pseudocode is provided for the QR algorithm. It generates a sequence of matrices $\{A_k\}$ that usually converges to an upper triangular matrix in the limit as $k \to \infty$.

Algorithm 1 The QR Algorithm without Shifts

$A_0 \leftarrow A$
for $k = 0, 1, \ldots$ **do**
  Compute a QR factorization $A_k = Q_{k+1}R_{k+1}$
  $A_{k+1} \leftarrow R_{k+1}Q_{k+1}$
end for

The QR Algorithm converges only linearly. To speed-up its convergence one can use the following variant given below with shifts.

Algorithm 2 The QR Algorithm with Shifts

$A_0 \leftarrow A$
for $k = 0, 1, \ldots$ **do**
  Choose a shift μ_k
  Compute a QR factorization $A_k - \mu_k I = Q_{k+1} R_{k+1}$
  $A_{k+1} \leftarrow R_{k+1} Q_{k+1} + \mu_k I$
end for

The QR algorithm with Rayleigh or Wilkinson shifts converges quadratically. Below in both parts perform calculations by hand.

(a) Apply one iteration of the QR algorithm to the matrix A provided below. (Note : $\lambda_1 = 5$ and $\lambda_2 = 1$ are the eigenvalues of A.)

$$A = \begin{bmatrix} 12 & -2 \\ 5 & 5 \end{bmatrix}$$

(b) Apply one iteration of the QR algorithm to the matrix A given in **(a)** with shift $\mu = 24$.

2. The QR algorithm without shifts (Algorithm 2 in Question **4.**) is equivalent to the simultaneous (power) iteration given below.

Algorithm 3 Simultaneous Iteration

for $k = 1, \ldots$ **do**
  Compute a QR factorization $A^k = \hat{Q}_k \hat{R}_k$
  $A_k \leftarrow \hat{Q}_k^T A \hat{Q}_k$
end for

In particular it can be shown that the sequences $\{A_k\}$ generated by the simultaneous iteration and the QR algorithm are the same.

Recall that the iterates of the power iteration $q_k = A^k q_0 / \|A^k q_0\|$ generically converges to the eigenvector associated with the eigenvalue with largest modulus as $k \to \infty$. Therefore the Rayleigh quotient $q_k^T A q_k$ converges to the associated eigenvalue with largest modulus.

The simultaneous iteration above can be viewed as the power iteration applied to the initial vectors $e_1, e_2, \ldots, e_n$ simultaneously, where e_j denotes the jth column of the identity matrix. For a symmetric matrix $A_k = \hat{Q}_k^T A \hat{Q}_k$ generically converges to a diagonal matrix with eigenvalues on the diagonal as $k \to \infty$. For a non-symmetric matrix A_k generically converges to an upper triangular matrix whose diagonal entries are the eigenvalues of A.

The purpose of this question is to verify the equivalence of the QR algorithm with the simultaneous iteration in practice on the following example.

$$A = \begin{bmatrix} 3 & -1 & -3 \\ -1 & 3 & -1 \\ -3 & -1 & 3 \end{bmatrix}$$

Calculate A_5 generated by the simultaneous iteration (use Algorithm 3) and the QR algorithm without shifts in Matlab. Do you obtain identical matrices that are close to a diagonal matrix?
(Note: The command `[Q,R] = qr(A)` computes the QR factorization $A = QR$ in Matlab.)

3. The computation of eigenvalues (for dense matrices) is typically performed in two stages. In the first stage the matrix A is reduced into Hessenberg form. The second stage is the QR algorithm as discussed in question **1.** The QR algorithm reduces the Hessenberg form into a triangular form by applying orthogonal similarity transformations.

The reason for the initial reduction into Hessenberg form is the computational efficiency. Each iteration of the QR algorithm requires $O(n^2)$ flops when the algorithm is started with an Hessenberg matrix, whereas the cost of each QR iteration would be $O(n^3)$ without the initial reduction.

In this question you will discover the advantages of the initial reduction for computational complexity.

(a) Suppose A_k generated by the QR algorithm (Algorithm 1) is a nonsingular matrix in Hessenberg form. Show that A_{k+1} is also a nonsingular Hessenberg matrix.
(Hint: The product of a triangular matrix with an Hessenberg matrix is another Hessenberg matrix.)

(b) Taylor an algorithm to compute the QR factorization of a matrix in Hessenberg form exploiting the structure and by performing $O(n^2)$ flops rather than $O(n^3)$ flops. Your algorithm should be based on 2×2 Householder reflectors. Write down a pseudocode for your algorithm.

4. This is a continuation of question **3.** from the previous homework. Implement a Matlab routine `inverse_iter.m` to compute an eigenvalue λ closest to a given shift σ and an associated eigenvector v of an $n \times n$ matrix A by inverse iteration.

Your routine must must look like

```
function [eigval, eigvec] = inverse_iter(A,sigma)

[n,n1] = size(A);

q = randn(n,1);
qold = zeros(n,1);

while norm(q-qold) > 10^-15
      ...
end

...
return;
```

Test your implementation with the following matrices.

$$B_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & -0.8 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 1 & 0 & 1/10 \\ 0 & -0.8 & 0 \\ 1/10 & 0 & 1 \end{bmatrix}, \quad B_3 = \begin{bmatrix} 1 & 0 & 1/100 \\ 0 & -0.8 & 0 \\ 1/100 & 0 & 1 \end{bmatrix}$$

(a) Run your inverse iteration routine to compute the eigenvalue closest to $\sigma = 1.2$ and an associated eigenvector for each of B_1, B_2, B_3. For which of these matrices the convergence is fastest, for which the convergence is slowest? Explain the convergence behavior you observe for B_1, B_2, B_3.

(b) A Matlab m-file `rayleigh_iter.m` is provided on the course website. It takes a matrix $A \in \mathbb{C}^{n \times n}$ as the input argument and returns a pair of an eigenvalue and an associated eigenvector. (The correct way to run it is `[eigval,eigvec] = rayleigh_iter(A)`) Run this routine to compute an eigenvalue and an associated eigenvector for B_3 only. Run it several times (*e.g.* fifteen, sixteen times). Does it always converge to the same eigenvalue? Can you observe that the rate of convergence is quadratic (perhaps even faster)?

5. Consider the polynomials $f(x) = x^3 + x$ and $g(x) = x^2$.

(a) Give the update rules of Newton's method (without backtracking) and the secant method for the functions f and g.

(b) Using the update rules from part **(a)** derive the rates of convergence of Newton's method for f and g.

(c) Generate an m-file `fun.m`

```
function [f,g] = fun(x)
% Task : Computes the function x^2 and its derivative.
f = x^2;
g = 2*x;

return
```

which computes the function $g(x) = x^2$ and the derivative $g'(x) = 2x$ and stores these values in the output arguments f and g, respectively.

(d) Generate an m-file similar to the one in **(c)** to compute the function value of $f(x) = x^3 + x$ and its derivative. The name of the m-file must be same as the function name.

(e) A Matlab implementation of Newton's method `Newton.m` for zero finding is provided on the course webpage. Type `help Newton` for information about its input and output arguments. Specifically, to compute the root of a function $f : \mathbb{R} \to \mathbb{R}$, Newton's method needs the function value $f(x)$ and the derivative $J(x) = f'(x)$ at every iteration. (The matlab routine is indeed for the more general multivariate case when $f : \mathbb{R}^n \to \mathbb{R}^m$. In

this case the derivative is replaced by the Jacobian matrix $J(x)$ as we shall see in class soon. You don't have to worry about the Jacobian matrix for now. Simply assume that J in the m-file keeps the derivative.) The second input argument to `Newton.m` is the name of an m-file that takes a vector $x \in \mathbb{R}^n$ as input and returns the function value $f(x)$ and the derivative $J(x) = f'(x)$ as output. For instance type

```
>> [xmin,fmin] = Newton(2,'fun');
```

to retrieve the root of $f(x) = x^2$ starting from the initial guess $x_0 = 2$ after generating the m-file `fun.m` in **(d)**.

Using the Matlab routine provided compute the roots of $f(x) = x^3 + x$ and $g(x) = x^2$ with the initial guesses $x_0 = 1$ and $x_0 = 4$. Include your Matlab output. Do you observe the convergence rates that you derived in part **(b)**?

(f) Implement the secant method by modifying the m-file `Newton.m`. Run your implementation to find the roots of $f(x) = x^3 + x$ and $g(x) = x^2$ starting from $x_0 = 2$ and $x_1 = 2.2$ (recall that the secant method requires two initial guesses). How fast does the secant method converge as compared to Newton's method for these two functions?