18.335 Problem Set 4 Solutions

Problem 1: Hessenberg ahead! (10+10+10 points)

(a) (Essentially the same recurrence is explained in equation 30.8 in the text.) You can derive this recurrence relation by recalling the expression for determinants in terms of cofactors or minors that you should have learned in a previous linear-algebra course. In $\det B$, the coefficient the (i, j) entry B_{ij} is $(-1)^{i+j}$ multiplied by the determinant of the submatrix (minor) of B with the i-th row and j-th column removed (this determinant, mutiplied by $(-1)^{i+j}$, is the *cofactor* of B_{ij}). The cofactor formula tells us that the determinant is equal to any row (or column) of B multiplied by its cofactors. Take the last row of B, which has only two nonzero entries B_{mm} and $B_{m,m-1}$. The cofactor of B_{mm} is $+\det B_{1:m-1,1:m-1}$. The cofactor of $B_{m,m-1}$ is the -1 times the determinant of the matrix whose last column is all zeros except for $B_{m-1,m}$ in the lower-right. To get the determinant of that matrix we take everything in that column and multiply by their cofactors; everythign is zero except for $B_{m-1,m}$, whose cofactor is $+\det B_{1:m-2,1:m-2}$. Putting these together, we have $\det B = B_{m,m}\det B_{1:m-1,1:m-1} - B_{m-1,m}B_{m,m-1}\det B_{1:m-2,1:m-2}$ as desired.

This gives a three-term recurrence relation, since the determinant of each submatrix is given in terms of the the determinants of the two smaller submatrices. At each iteration, we only need the determinants of the submatrices of size $n \times n$ and $(n-1) \times (n-1)$ to get the determinant of the submatrix of size $(n+1) \times (n+1)$ using three multiplications and one addition, or four flops. We have to apply the recurrence $m-1$ times to get the full $\det(H - zI)$, for $4(m-1) + m = O(m)$ flops (where the $+m$ flops was for subtracting z from the m diagonals). The code is given in part (b) below (including the derivative.

To test it, we computed a random H matrix as suggested in the problem, for $m = 20$, and computed the relative error `(evalpoly(H,z)-det(H-z*eye(m)))/det(H-z*eye(m))` for a few randomly chosen values of z. The relative error was always on the order of the machine precision $\approx 10^{-16}$.

(b) If we simply take the derivative $\frac{d}{dz}$ of our recurrence relation for $\det B$, we get a three-term recurrence relation for $(\det B)'$:

$$(\det B)' = -\det B_{1:m-1,1:m-1} + B_{m,m}(\det B_{1:m-1,1:m-1})' - B_{m-1,m}B_{m,m-1}(\det B_{1:m-2,1:m-2}),$$

using the fact that for $B = H - zI$ we have $B' = -I$ and hence $B'_{mm} = -1$ while $B'_{m-1,m} = B'_{m,m-1} = 0$.

This is implemented in the following Matlab function:

```
function [p,pderiv] = evalpoly(H,z)
  m = size(H,1);
  d1 = (H(1,1) - z);
  d2 = 1;
  d1deriv = -1;
  d2deriv = 0;
  for i = 2:m
    d = (H(i,i) - z) * d1 - H(i-1,i)*H(i,i-1) * d2;
    dderiv = -d1 + (H(i,i) - z) * d1deriv - H(i-1,i)*H(i,i-1)* d2deriv;
    d2 = d1;
    d1 = d;
    d2deriv = d1deriv;
    d1deriv = dderiv;
  end
```

```
    p = d1;
    pderiv = d1deriv;
```

If we choose $\Delta z = 10^{-4}$, and compare $[p(z+\Delta z) - p(z-\Delta z)]/2\Delta z$ to `pderiv` computed by this function for a random H (as above) and for various random $z \in [0,1]$, then the relative difference is usually on the order of 10^{-6} or so. Decreasing Δz to 10^{-6}, the relative difference is around 10^{-10} or so; exactly the convergence of $O(\Delta z^2)$ that one would expect: Taylor expanding $p(z+\Delta z) = p(z) + p'(z)\Delta z + p''(z)\Delta z^2/2 + O(\Delta z^3)$, one finds $[p(z+\Delta z) - p(z-\Delta z)]/2\Delta z = p'(z) + O(\Delta z^2)$. Of course, you can't make Δz too small or you run into the limitations of floating-point precision (e.g. $\Delta z = 10^{-20}$ will just give zero for $p(z+\Delta z) - p(z-\Delta z)$ if $z \sim 1$).

(c) Newton's method is just the iteration $z \leftarrow z - p(z)/p'(z)$. You can do this "by hand" in Matlab, but I wrote a little Matlab script to repeat this until the answer stops changing to within $10\varepsilon_{\text{machine}}$. (Better stopping criteria could be devised, but this works well enough for illustration purposes.)

```
function lam = newtpoly(H, lamguess)
  lam = lamguess;
  for i = 1:1000
    [p,pderiv] = evalpoly(H, lam);
    disp(lamnew = lam - p / pderiv);
    if (abs(lamnew - lam) < 10*eps*abs(lamnew))
      break;
    end
    lam = lamnew;
  end
```

Applying this to my random A and H from above, I run `eig(A)` to get the eigenvalues, and then execute the Newton solver with starting points that are somewhere nearby one of the eigenvalues (e.g. the eigenvalue rounded to two significant digits). You can run the command `format long` to have Matlab print out values to 15 decimal places, to better see what is going on. A typical output is:

```
>> newtpoly(H, 0.96)
0.961143436106079
0.961118817820461
0.961118806268620
0.961118806268617
0.961118806268617
ans = 0.961118806268617
```

In comparison, the corresponding eigenvalue returned by `eig(A)` is $\lambda \approx 0.961118806268619$, which differs only in the 15th decimal place from the Newton result. If you repeat this with several eigenvalues (i.e., different starting points), you find that it consistently finds λ to nearly machine precision. Notice that the number of accurate digits roughly doubles on each step, which is typical of Newton's method: this is "quadratic" convergence.

If you tried a test case with a bigger value of m, say $m = 1000$, you problably noticed a problem: the determinant grows so large that it overflows the range of floating-point arithmetic, leading to results of NaN or Inf! However, this is easily fixed. For example, as one is computing the determinant, one can periodically check the magnitude and rescale to prevent it from growing too large (or too small)—obviously, $p(z)$ multiplied by any constant still has the same roots. There are also slightly more clever recurrences

2

that rescale the result automatically; See Barth, Martin, and Wilkinson (1967).[1] The other key to finding eigenvalues in this way, described in lecture 30 of Trefethen, is that there is a simple technique to count the number of eigenvalues in any given interval, leading to the "bisection" algorithm I mentioned in class. I didn't expect you to do any of that stuff however; I hope you all just picked a small enough m to avoid overflow.

Problem 2: Q's 'R us (10+15 points)

(a) In finite precision, instead of $w = A^{-1}v$, we will get $\tilde{w} = w + \delta w$ where $\delta w = -(A + \delta A)^{-1}\delta A w$ (from the formula on page 95), where $\delta A = O(\varepsilon_{\text{machine}})\|A\|$ is the backwards error. [Note that we cannot use $\delta w \approx -A^{-1}\delta A w$, which neglects the $\delta A \delta w$ terms, because in this case δw is not small.] The key point, however, is to show that δw is mostly parallel to q_1, the eigenvector corresponding to the smallest-magnitude eigenvalue λ_1 (it is given that all other eigenvalues have magnitude $\geq |\lambda_2| \gg |\lambda_1|$). Since w is also mostly parallel to q_1, this will mean that $\tilde{w}/\|\tilde{w}\|_2 \approx q_1 \approx w/\|w\|_2$.

First, exactly as in our analysis of the power method, note that $w = A^{-1}v = \alpha_1 q_1[1 + O(\lambda_1/\lambda_2)]$, since A^{-1} amplifies the q_1 component of v by $1/|\lambda_1|$ which is much bigger than the inverse of all the other eigenvalues. Thus, $w/\|w\|_2 = q_1[1 + O(\lambda_1/\lambda_2)]$.

Second, if we Taylor-expand $(A + \delta A)^{-1}$ in powers of δA, i.e. in powers of $\varepsilon_{\text{machine}}$, we obtain: $(A + \delta A)^{-1} = A^{-1} - A^{-1}\delta A A^{-1} + O(\varepsilon_{\text{machine}}^2)$. Since all of the terms in this expansion are proportional to A^{-1}, when multiplied by *any* vector they will again amplify the q_1 component much more than any other component. In particular, the vector $\delta A w$ is a vector in a random direction (since δA comes from roundoff and is essentially random) and hence will have some nonzero q_1 component. Thus, $\delta w = -(A + \delta A)^{-1}\delta A w = \beta_1 q_1[1 + O(\lambda_1/\lambda_2)]$ for some constant β_1.

Putting these things together, we see that $\tilde{w} = (\alpha_1 + \beta_1)q_1[1 + O(\lambda_1/\lambda_2)]$, and hence $\tilde{w}/\|\tilde{w}\|_2 = q_1[1 + O(\lambda_1/\lambda_2)] = \frac{w}{\|w\|_2}[1 + O(\lambda_1/\lambda_2)]$. Q.E.D.

(b) Trefethen, problem 28.2:

(i) In general, r_{ij} is nonzero (for $i < j$) if column i is non-orthogonal to column j. For a tridiagonal matrix A, only columns within two columns of one another are non-orthogonal (overlapping in the nonzero entries), so R should only be nonzero (in general) for the diagonals and for two entries above each diagonal; i.e. r_{ij} is nonzero only for $i = j$, $i = j - 1$, and $i = j - 2$.

Each column of the Q matrix involves a linear combination of all the previous columns, by induction (i.e. q_2 uses q_1, q_3 uses q_2 and q_1, q_4 uses q_3 and q_2, q_5 uses q_4 and q_3, and so on). This means that an entry (i, j) of Q is zero (in general) only if $a_{i,1:j} = 0$ (i.e., that entire row of A is zero up to the j-th column). For the case of tridiagonal A, this means that Q will have upper-Hessenberg form.

(ii) **Note:** In the problem, you are told that A is symmetric and tridiagonal. You must also assume that A is real, or alternatively that A is Hermitian and tridiagonal. (This is what must be meant in the problem, since tridiagonal matrices only arise in the QR method if the starting point is Hermitian.) In contrast, if A is complex tridiagonal with $A^T = A$, the stated result is not true (RQ is not in general tridiagonal, as can easily be verified using a random tridiagonal complex A in Matlab).

It is sufficient to show that RQ is upper Hessenberg: since $RQ = Q^*AQ$ and A is Hermitian,

---

[1] W. Barth, R. S. Martin, and J. H. Wilkinson, "Calculation of the eigenvalues of a symmetric tridiagonal matrix by the bisection method," *Numer. Math.* **9**, 386–393 (1967).

then RQ is Hermitian and upper-Hessenberg implies tridiagonal. To show that RQ is upper-Hessenberg, all we need is the fact that R is upper-triangular and Q is upper-Hessenberg.

Consider the (i, j) entry of RQ, which is given by $\sum_k r_{i,k} q_{k,j}$. $r_{i,k} = 0$ if $i > k$ since R is upper triangular, and $q_{k,j} = 0$ if $k > j+1$ since Q is upper-Hessenberg, and hence $r_{i,k} q_{k,j} \neq 0$ only when $i \leq k \leq j+1$, which is only true if $i \leq j+1$. Thus the (i, j) entry of RQ is zero if $i > j+1$ and thus RQ is upper-Hessenberg.

(iii) As we saw in problem set 3 (Trefethen 10.4), the 2×2 Givens QR algorithm does elimination "bottom-up" on each column, introducing zeros one at a time in the current column of A. Obviously, if A is tridiagonal (or even just upper-Hessenberg), most of each column is already zero—we only need to introduce one zero into each column below the diagonal. Hence, for each column k we only need to do *one* Givens rotation of the k-th and $(k+1)$-st rows. Each 2×2 Givens rotation requires 6 flops (multiping a 2-component vector by a 2×2 matrix), and we need to do it for all columns starting from the k-th. However, actually we only need to do it for 3 columns for each k, since from above the conversion from A to R only introduces one additional zero above each diagonal, so most of the Givens rotations in a given row are zero. That is, the process looks like

$$\begin{pmatrix} \cdot & \cdot & & & \\ \cdot & \cdot & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \end{pmatrix} \rightarrow \begin{pmatrix} \bullet & \bullet & \bullet & & \\ 0 & \bullet & \bullet & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \end{pmatrix} \rightarrow \begin{pmatrix} \cdot & \cdot & \cdot & & \\ 0 & \bullet & \bullet & \bullet & \\ & 0 & \bullet & \bullet & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot \end{pmatrix} \rightarrow \begin{pmatrix} \cdot & \cdot & \cdot & & \\ 0 & \cdot & \cdot & \cdot & \\ & 0 & \bullet & \bullet & \bullet \\ & & 0 & \bullet & \bullet \\ & & & \cdot & \cdot \end{pmatrix},$$

where $\bullet$ indicates the entries that change on each step. Notice that it gradually converts A to R, with the two nonzero entries above each diagonal as explained above, and that each Givens rotation need only operate on three columns. Hence, only $O(m)$ flops are required, compared to $O(m^3)$ for ordinary QR! [Getting the exact number requires more care that I won't bother with, since we can no longer sweep under the rug the $O(m)$ operations required to construct the 2×2 Givens or Householder matrix, etc.]

Problem 3 (5+5+5+5+5 pts):

The underlying point in this problem is that Arnoldi will only stop partway if the starting vector **b** is in the span of a subset of only n of the eigenvectors. This is why multiplying by A more than n times does not give any new vectors, why the Ritz vectors are exact eigenvectors (spanned by Q_n), and why $A\mathbf{x} = \mathbf{b}$ has a solution $\mathbf{x} \in \mathcal{K}_n$. These results are shown explicitly by algebraic manipulation below.

(a) In this case, the q_{n+1} vector is multiplied by a zero row in $\tilde{H}_n$, and we can simplify 33.13 to $AQ_n = Q_n H_n$. If we consider the full Hessenberg reduction, $H = Q^* A Q$, it must have a "block Schur" form:

$$H = \begin{pmatrix} H_n & B \\ 0 & H' \end{pmatrix},$$

where H' is an $(m-n) \times (m-n)$ upper-Hessenberg matrix and $B \in \mathbb{C}^{n \times (m-n)}$. (It is *not* necessarily the case that $B = 0$; this is only true if A is Hermitian.)

(b) Q_n is a basis for $\mathcal{K}_n$, so any vector $x \in \mathcal{K}_n$ can be written as $x = Q_n y$ for some $y \in \mathbb{C}^n$. Hence, from above, $Ax = AQ_n y = Q_n H_n y = Q_n (H_n y) \in \mathcal{K}_n$. Q.E.D.

(c) The $(n+1)$ basis vector, $A^n b$, is equal to $A(A^{n-1}b)$ where $A^{n-1}b \in \mathcal{K}_n$. Hence, from above, $A^n b \in \mathcal{K}_n$ and thus $\mathcal{K}_{n+1} = \mathcal{K}_n$. By induction, $\mathcal{K}_\ell = \mathcal{K}_n$ for $\ell \geq n$.

4

(d) If $H_n y = \lambda y$, then $A Q_n y = Q_n H_n y = \lambda Q_n y$, and hence λ is an eigenvalue of A with eigenvector $Q_n y$.

(e) If A is nonsingular, then H_n is nonsingular (if it had a zero eigenvalue, A would too from above). Hence, noting that b is proportional to the first column of Q_n, we have: $x = A^{-1} b = A^{-1} Q_n e_1 \|b\| = A^{-1} Q_n H_n H_n^{-1} e_1 \|b\| = A^{-1} A Q_n H_n^{-1} e_1 \|b\| = Q_n H_n^{-1} e_1 \|b\| \in \mathcal{K}_n$. Q.E.D.

18.335J / 6.337J Introduction to Numerical Methods
Fall 2010