18.335 Problem Set 3 Solutions

Problem 1: QR and RQ

(a) Trefethen, problem 10.4:

(i) e.g. consider $\theta = \pi/2$ ($c = 0$, $s = 1$): $Je_1 = -e_2$ and $Je_2 = e_1$, while $Fe_1 = e_2$ and $Fe_2 = e_1$. J rotates clockwise in the plane by θ. F is easier to interpret if we write it as J multiplied on the right by $[-1,0;0,1]$: i.e., F corresponds to a mirror reflection through the y (e_2) axis followed by clockwise rotation by θ. More subtly, F corresponds to reflection through a mirror plane corresponding to the y axis rotated clockwise by $\theta/2$. That is, let $c_2 = \cos(\theta/2)$ and $s_2 = \cos(\theta/2)$, in which case (recalling the identities $c_2^2 - s_2^2 = c$, $2s_2c_2 = s$):

$$\begin{pmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{pmatrix} \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{pmatrix} = \begin{pmatrix} -c_2 & s_2 \\ s_2 & c_2 \end{pmatrix} \begin{pmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{pmatrix} = \begin{pmatrix} -c & s \\ s & c \end{pmatrix} = F,$$

which shows that F is reflection through the y axis rotated by $\theta/2$.

(ii) The key thing is to focus on how we perform elimination under a single column of A, which we then repeat for each column. For Householder, this is done by a single Householder rotation. Here, since we are using 2×2 rotations, we have to eliminate under a column one number at a time: given 2-component vector $x = \begin{pmatrix} a \\ b \end{pmatrix}$ into $Jx = \begin{pmatrix} \|x\|_2 \\ 0 \end{pmatrix}$, where J is clockwise rotation by $\theta = \tan^{-1}(b/a)$ [or, on a computer, atan2(b,a)]. Then we just do this working "bottom-up" from the column: rotate the bottom two rows to introduce one zero, then the next two rows to introduce a second zero, etc.

(iii) The flops to compute the J matrix itself are asymptotically irrelevant, because once J is computed it is applied to many columns (all columns from the current one to the right). To multiply J by a single 2-component vector requires 4 multiplications and 2 additions, or 6 flops. That is, 6 flops per row per column of the matrix. In contrast, Householder requires each column x to be rotated via $x = x - 2v(v^*x)$. If x has m components, v^*x requires m multiplications and $m-1$ additions, multiplication by $2v$ requires m more multiplications, and then subtraction from x requires m more additions, for $4m-1$ flops overall. That is, asymptotically 4 flops per row per column. The 6 flops of Givens is 50% more than the 4 of Householder.

The reason that Givens is still considered interesting and useful is that (as seen in problem 28.2 below) it can be used to exploit *sparsity*: because it rotates only two elements at a time in each column, from the bottom up, if a column ends in zeros then the zero portion of the column can be skipped.

(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. In contrast, if A is complex tridiagonal with $A^T = A$, the stated result is not true.)

It is sufficient to show that RQ is upper Hessenberg: since $RQ = Q^*AQ$ and A is Hermitian, 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) 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 2×2 Givens rotation or 2×2 Householder reflection of the k-th and $(k+1)$-st rows, rotating $\begin{pmatrix} \cdot \\ \cdot \end{pmatrix} \to \begin{pmatrix} \bullet \\ 0 \end{pmatrix}$. Each 2×2 rotation/reflection 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 rotations in a given row are zero. That is, the process looks like

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

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 2: Distribution and association

(a) The first method is slower for large m, because it involves matrix–matrix operations, which are $\Theta(m^3)$, whereas the second involves only $\Theta(m^2)$ matrix–vector and $\Theta(m)$ vector–vector operations.

More precisely, the flop count for a matrix–matrix product like AB is $\approx 2m^3$, the flop count for matrix addition is m^2, the flop count for a matrix-vector product like Dy or x^TA is $\approx 2m^2$, and the flop count for a dot product is $\approx 2m$. So, computing $x^T(AB+CD)y$ requires $2\times 2m^3 + O(m^2) \approx 4m^3$ flops, whereas computing $((x^TA)B)y + ((x^TC)D)y$ (left-associative) requires $2\times(2\times 2m^2) + O(m) \approx 8m^2$ flops. For $m = 1000$, this is $500\times$ fewer operations—even accounting for differences in cache utilization etcetera, a factor of 500 is big enough to likely swamp all other effects and make the second method faster.

(b) In Julia, we can set `m=1000` and allocate random inputs with `A,B,C,D = [rand(m,m) for i=1:4]` and `x,y = [rand(m) for i=1:2]`. Then, using the `@btime` macro as suggested, I find that `@btime $x'*($A*$B+$C*$D)*$y` is about $25\times$ times slower than `@btime $x'*$A*$B*$y+$x'*$C*$D*$y` (your exact numbers will vary depending on your machine, of course).

These numbers are a bit deceptive however, because these large matrix–matrix products can use multiple threads more efficiently than cheaper matrix–vector products, so we were comparing multi-threaded performance to single-threaded performance. If I tell Julia to use only single-threaded BLAS via `using LinearAlgebra; LinearAlgebra.BLAS.set_num_threads(1)` similar to the matrix-multiplication notebook from class, then I get about $62\times$ slower for the matrix–matrix version.

This is still a far cry from the $500\times$ slower that you would predict from the flop count alone, and that's because more than just flops matter. A significant fraction of the time is taken by just memory allocation for the results and for temporary matrices/vectors, and those allocations are similar in the two cases. Moreover, matrix–vector operations are slowed down by the fact that they require $\Theta(m^2/L)$ cache misses where L is the cache-line length—there is not enough temporal locality to save a factor of $\sqrt{Z}$ as in the matrix–matrix case—so they are more limited by the speed of memory than the matrix–matrix case.

Problem 3: Least squares

Trefethen, problem 11.2. See the pset 3 solution notebook for Julia code and accompanying explanations.

Problem 4: Schur factorization

(a) First, let us show that T is normal: substituting $A = QTQ^*$ into $AA^* = A^*A$ yields $QTQ^*QT^*Q^* = QT^*Q^*QTQ^*$ and hence (cancelling the Qs) $TT^* = T^*T$.

The (1,1) entry of T^*T is the squared L_2 norm ($\|\cdot\|_2^2$) of the first column of T, i.e. $|t_{1,1}|^2$ since T is upper triangular, and the (1,1) entry of TT^* is the squared L_2 norm of the first row of T, i.e. $\sum_i |t_{1,i}|^2$. For these to be equal, we must obviously have $t_{1,i} = 0$ for $i > 1$, i.e. that the first row is diagonal.

We proceed by induction. Suppose that the first $j-1$ rows of T are diagonal, and we want to prove this of row j. The (j,j) entry of T^*T is the squared norm of the j-th column, i.e. $\sum_{i\leq j} |t_{i,j}|^2$, but this is just $|t_{j,j}|^2$ since $t_{i,j} = 0$ for $i < j$ by induction. The (j,j) entry of TT^* is the squared norm of the j-th row, i.e. $\sum_{i\geq j} |t_{j,i}|^2$. For this to equal $|t_{j,j}|^2$, we must have $t_{j,i} = 0$ for $i > j$, and hence the j-th row is diagonal. Q.E.D.

(b) The eigenvalues are the roots of $\det(T - \lambda I) = \prod_i (t_{i,i} - \lambda) = 0$—since T is upper-triangular, the roots are obviously therefore $\lambda = t_{i,i}$ for $i = 1, \ldots, m$. To get the eigenvector for a given $\lambda = t_{i,i}$, it suffices to compute the eigenvector x of T, since the corresponding eigenvector of A is Qx.

x satisfies

$$0 = (T - t_{i,i}I)x = \begin{pmatrix} T_1 & u & B \\ & 0 & v^* \\ & & T_2 \end{pmatrix} \begin{pmatrix} x_1 \\ \alpha \\ x_2 \end{pmatrix},$$

where we have broken up $T - t_{i,i}I$ into the first $i-1$ rows $(T_1\, u B)$, the i-th row (which has a zero on the diagonal), and the last $m-i$ rows T_2; similarly, we have broken up x into the first $i-1$ rows x_1, the i-th row α, and the last $m-i$ rows x_2. Here, $T_1 \in \mathbb{C}^{(i-1)\times(i-1)}$ and $T_2 \in \mathbb{C}^{(m-i)\times(m-i)}$ are upper-triangular, and are non-singular because by assumption there are no repeated eigenvalues and hence no other $t_{j,j}$ equals $t_{i,i}$. $u \in \mathbb{C}^{i-1}$, $v \in \mathbb{C}^{m-i}$, and $B \in \mathbb{C}^{(i-1)\times(m-i)}$ come from the upper triangle of T and can be anything. Taking the last $m-i$ rows of the above equation, we have $T_2 x_2 = 0$, and hence $x_2 = 0$ since T_2 is invertible. Furthermore, we can scale x arbitrarily, so we set $\alpha = 1$. The first $i-1$ rows then give us the equation $T_1 x_1 + u = 0$, which leads to an upper-triangular system $T_1 x_1 = -u$ that we can solve for x_1.

Now, let us count the number of operations. For the i-th eigenvalue $t_{i,i}$, to solve for x_1 requires

$\sim (i-1)^2 \sim i^2$ flops to do backsubstitution on an $(i-1) \times (i-1)$ system $T_1 x_1 = -u$. Then to compute the eigenvector Qx of A (exploiting the $m-i$ zeros in x) requires $\sim 2mi$ flops. Adding these up for $i = 1 \ldots m$, we obtain $\sum_{i=1}^{m} i^2 \sim m^3/3$, and $2m \sum_{i=0}^{m-1} i \sim m^3$, and hence the overall cost is $\sim \frac{4}{3} m^3$ flops ($K = 4/3$).