

1. Count to leading order (i.e. find the constant in front of n^3) the number of FLOPS it takes to compute the QR factorization of an n -by- n matrix A (i.e. $A = QR$) using Householder matrices (here A does not need to be symmetric). Count the additions/subtractions and multiplications/divisions/square roots separately. Compare this to the results for LU decomposition.
2. Bradie, p. 320 problem 17 (a)–(d)
3. At the heart of the QR iteration for similarity transforming a symmetric tridiagonal matrix A is the recursion $A^{(k-1)} = Q^{(k)}R^{(k)}$, $A^{(k)} = R^{(k)}Q^{(k)}$, where we can denote the elements of $A^{(k-1)}$ and $A^{(k)}$ as follows

$$A^{(k-1)} = \begin{bmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & a_3 & a_3 & b_4 & \\ & & \ddots & \ddots & \ddots \\ & & & b_n & a_n \end{bmatrix}, \quad A^{(k)} = \begin{bmatrix} \bar{a}_1 & \bar{b}_2 & & & \\ \bar{b}_2 & \bar{a}_2 & \bar{b}_3 & & \\ & \bar{b}_3 & \bar{a}_3 & \bar{b}_4 & \\ & & \ddots & \ddots & \ddots \\ & & & \bar{b}_n & \bar{a}_n \end{bmatrix}$$

If we are not interested in simultaneously computing the eigenvectors, then it turns out that all the steps involved in going from $A^{(k-1)}$ to $A^{(k)}$ can be formulated extremely conveniently as one simple recursion which does not require either trigonometric functions or square roots. The recursion is given in the ‘classic’ book *The Algebraic Eigenvalue Problem* by J.H. Wilkinson (1965) as follows:

$$\left. \begin{aligned} u_0 &= 0, \quad c_0 = 1, \quad b_{n+1} = 0, \quad a_{n+1} = 0, \\ \left. \begin{aligned} \gamma_j &= a_j - u_{j-1} \\ p_j^2 &= \gamma_j^2 / c_{j-1}^2 && \text{if } c_{j-1} \neq 0 \\ &= c_{j-2}^2 b_j^2 && \text{if } c_{j-1} = 0 \\ \bar{b}_j^2 &= s_{j-1}^2 (p_j^2 + b_{j+1}^2) && \text{if } j \neq 1 \\ s_j^2 &= b_{j+1}^2 / (p_j^2 + b_{j+1}^2) \\ c_j^2 &= p_j^2 / (p_j^2 + b_{j+1}^2) \\ u_j &= s_j^2 (\gamma_j + a_{j+1}) \\ \bar{a}_j &= \gamma_j + u_j \end{aligned} \right\} \text{for } j = 1 \dots n \end{aligned} \right.$$

The algorithm above calculates \bar{a}_j and \bar{b}_j^2 from the values a_j and b_j^2 . This suffices since the signs of the b_j have no influence on the eigenvalues (see the extra credit problem).

- (a) Implement the algorithm above in MATLAB, and use it to reproduce the example in Bradie, p. 322.
- (b) Incorporate the ‘single’ shift strategy in your code and then compare it with the example in Bradie, pp. 329–330.
- (c) Extend your code so that it automatically determines all the eigenvalues of a general symmetric tridiagonal matrix A by using deflation. Test it on the matrix with entries $a_k = -2$, $k = 1, 2, \dots, 10$ and $b_k = 1$, $k = 2, \dots, 10$. Compare your computation with the analytic result $\lambda_k = -2 \left(1 + \cos \frac{\pi k}{n+1} \right)$, $k = 1, 2, \dots, 10$.

Note: It is possible for the single shifted QR iteration to get ‘hung-up’ (hint, hint), this problem can usually be ameliorated by inserting a single random value for the shift. Such a shift can for example be done during the very first iteration.

- (d) (Extra credit: 10 points) Show that the eigenvalues of $A^{(k)}$ are unchanged if the signs of the b_j s are swapped.

4. QR Decomposition

- (a) Write a program, using for example MATLAB, for computing the QR decomposition of a real m -by- n matrix A via Householder matrices. Your program should take as input the matrix A and return as output the matrices Q and R , such that Q is orthogonal, R is upper-triangular, and $A = QR$. You may not use any functions available in software libraries, such as MATLAB's `qr`, to compute the decomposition. However, you may wish to check your code with one of these functions. Your code should also avoid unnecessary FLOPs such as multiplying an identity matrix times another matrix. Test your code by computing the QR decomposition of following two matrices:

$$(i) \quad A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad (ii) \quad A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Turn in a listing of your function along with the results for the two test problems.

- (b) Consider the following overdetermined linear system of equations

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \\ 1 & x_6 & x_6^2 \end{bmatrix}}_A \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}}_c = \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix}}_f$$

where $x_j = \frac{(j-1)\pi}{6}$, $j = 1, 2, \dots, 6$, and $f_j = \sin(x_j)$, $j = 1, 2, \dots, 6$. Use your algorithm from part (a) to compute the best least squares solution to this system. Report your solution for c_1, c_2, c_3 .

Note: The solution to this linear system gives the coefficients to the quadratic polynomial $p(x) = c_1 + c_2x + c_3x^2$ that fits the data (x_j, f_j) in the least-squares sense.

5. Download the bitmap image file `utah.bmp` from the course webpage. The image (given above) can be thought of as a matrix of pixels with each entry representing the shade of gray to display for that pixel.



- (a) Load the image into MATLAB and then display the image using the commands

```
>>A = double(imread('utah.bmp'));
>>imagesc(A), colormap gray
```

Note that the image must be stored in the directory you are currently operating MATLAB from. Upon execution of the first command the variable A will contain the image as a matrix. Using MATLAB's `svd` command compute the SVD of A . Plot the singular values of A as a function of their position on the diagonal of the singular value matrix returned by MATLAB. Based on the plot approximate the rank of A .

- (b) For each $j = 2, 4, 6, 8, 10$, construct the rank j matrix $B^{(j)}$ that best approximates A in the 2-norm sense. Use the command `imagesc` as above to display images for each $B^{(j)}$. Print each of the images and include them with your assignment. Discuss how each image $B^{(j)}$ compares to A .