Assignment 10

Due Monday 11 December 2023, at 5 pm in my Chapman 101 box

Please read Lectures 24, 25, 26, 27, and 28 in the textbook *Numerical Linear Algebra* by Trefethen and Bau. This Assignment mostly covers eigenvalues, including some iterations which approximate them: power, inverse, and Rayleigh quotient iterations. We will not get to the actual/practical QR algorithm for eigenvalues (Lecture 29), nor to material beyond that.

DO THE FOLLOWING EXERCISES from Lecture 24:

- **Exercise 24.1**

DO THE FOLLOWING ADDITIONAL EXERCISES.

P19. An *in place* Gauss elimination algorithm uses no more memory to store L and U than is already used to store A.

(a) Write a function with signature `Z = iplu(A)` which takes as input a square $m \times m$ matrix A and computes $A = LU$ by Algorithm 20.1. It will not create separate matrices L and U. It will produce a matrix Z which has the numbers l_{jk} and u_{jk} in the corresponding locations. You will be able to recover matrices L and U as follows:

```
>> Z = iplu(A);
>> U = triu(Z)
>> L = tril(Z,-1) + diag(ones(m,1))
```

Demonstrate that `iplu(A)` works by applying it to the matrix A in (20.3) and recovering the factors in (20.5).

(b) Now write another function with signature `x = bslash(A,b)` which solves square systems $Ax = b$. It calls `iplu(A)` to compute the in-place LU factorization. Then it solves the system from Z *without* forming L or U.[1] It will have loops which implement forward- and backward-substitution (Alg. 17.1) using the entries of Z. Show it works by comparing to "\" on some randomly-generated 10×10 system $Ax = b$:

```
>> x1 = bslash(A,b);
>> x2 = A \ b;
>> norm(x1 - x2) / norm(x2)
```

(c) Extra Credit Regarding stability, Algorithm 20.1 is not a good idea. Implement Gauss elimination with partial pivoting (Algorithm 21.1) using an integer permutation vector `p` for the row swaps. (Do not actually move values in memory.) Demonstrate correctness of your updated `bbslash(A,b)`[2] as in part **(b)**. Then find an example for which this updated version produces substantially reduced floating-point error.

[1] And, of course, without using MATLAB's backslash operation!

[2] "Better backslash."

P20. A *circulant matrix* is one where constant diagonals "wrap around":

$$C = \begin{bmatrix} c_1 & c_m & \dots & c_3 & c_2 \\ c_2 & c_1 & c_m & & c_3 \\ \vdots & c_2 & c_1 & \ddots & \vdots \\ c_{m-1} & & \ddots & \ddots & c_m \\ c_m & c_{m-1} & \dots & c_2 & c_1 \end{bmatrix} \tag{1}$$

Each entry of $C \in \mathbb{C}^{m\times m}$ is determined from the entries $c_1, \dots, c_m$ in its first column:

$$C_{jk} = \begin{cases} c_{j-k+1}, & j \ge k, \\ c_{m+j-k+1}, & j < k. \end{cases}$$

Specifying the first column of a circulant matrix describes it completely.

An extraordinary fact about a circulant matrix is that it has a complete set of complex eigenvectors *that are known in advance*, without knowing the eigenvalues. Specifically, define $f_k \in \mathbb{C}^m$ by

$$(f_k)_j = \exp\left(-i(j-1)(k-1)\frac{2\pi}{m}\right) = e^{-i2\pi(k-1)(j-1)/m}, \tag{2}$$

where, as usual, $i = \sqrt{-1}$. These vectors are *waves*, i.e. combinations of familiar sines and cosines, and in fact this exercise can be regarded as how one "discovers" Fourier series (and Fourier-type ideas generally). After some warm-up exercises you will show in part **(e)** that $Cf_k = \lambda_k f_k$ for a computable eigenvalue λ_k.

(a) Define the *periodic convolution* $u * w \in \mathbb{C}^m$ of vectors $u, w \in \mathbb{C}^m$ by

$$(u * w)_j = \sum_{k=1}^{m} u_{\mu(j,k)} w_k \qquad \text{where} \qquad \mu(j,k) = \begin{cases} j-k+1, & j \ge k, \\ m+j-k+1, & j < k. \end{cases}$$

Show that $u * w = w * u$.

(b) Show that $Cu = v * u$ if C is a circulant matrix and v is the first column of C.

(c) Show that the vectors $f_1, \dots, f_m$ defined in (2) are orthogonal. (*Hints.* Remember the conjugate in the inner product. Then use a fact about finite geometric series.)

(d) For $m = 20$, use Matlab to plot the real parts of the vectors $f_1, \dots, f_5$, together in a single figure. (*Hint.* They should look like discretized waves.)

(e) For a general circulant matrix, C in (1) above, give a formula for its eigenvalues λ_k in terms of the entries $c_1, \dots, c_m$. That is, show via by-hand calculation that

$$Cf_k = \lambda_k f_k$$

where f_k is given by (2). Your solution will contain a formula for λ_k.

P21. **(a)** Implement Algorithm 26.1, Householder reduction to Hessenberg form. Specifically, build a code with the signature

```
H = hessen(A,stages)
```

Your code will check that A is square, print the stages if `stages` is `true`, and finally return a Hessenberg matrix H such that $A = QHQ^*$ for some unitary Q. Note that your code can discard the vectors v_k after they are used.

(b) For a random 5×5 matrix A of your choice, run the code and show the four stages A, $Q_1^* A Q_1$, $Q_2^* Q_1^* A Q_1 Q_2$, and $H = Q_3^* Q_2^* Q_1^* A Q_1 Q_2 Q_3$. (*Hint.* This illustrates the cartoons on pages 197–198, in the **A Good Idea** subsection.) Use the built-in `eig()` to show that the eigenvalues of A and H are the same to within rounding error.

(c) Construct a new 4×4 Hermitian matrix S and compute `T=hessen(S)`. Check that T is tridiagonal and Hermitian. Show that the eigenvalues of S and T are the same within rounding error.

P22. **(a)** Implement Algorithm 27.3, Rayleigh quotient iteration. Specifically, write a code with signature

```
[lam,v] = rqi(A,v0)
```

which returns an eigenvalue `lam` corresponding to the eigenvector `v`, and which starts the iteration from a given vector `v0`. As a stopping criterion, to avoid a warning when solving the linear system with the matrix $B = A - \lambda^{(k-1)} I$, I suggest

```
rcond(B) < 10*eps
```

or equivalent; using Matlab or other documentation, explain what this criterion means.

(b) Show your code works by *(i)* reproducing the iterates $\lambda^{(0)}$, $\lambda^{(1)}$, $\lambda^{(2)}$ in Example 27.1, and *(ii)* by matching one of the eigenvalues and eigenvectors, computed by the built-in command `eig()`, of a random 20×20 Hermitian matrix.

Extra Credit A. Theorem 15.1 requires that your algorithm be *backward stable*. What if it is merely *stable* according to the definition given in Lecture 14? To my surprise, I was able to prove a theorem about the relative error which is nearly as strong. Show:

Theorem. *Suppose a stable algorithm* $\tilde{f} : X \to Y$ *is applied to solve a problem* $f : X \to Y$ *with condition number* κ *on a computer satisfying (13.5), (13.7). Then there is a constant* $\gamma \geq 0$ *so that the relative errors satisfy*

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O\left((\kappa(x) + \gamma)\epsilon_{machine}\right) \qquad \text{as} \quad \epsilon_{machine} \to 0.$$

Hints. Roughly follow the proof of Theorem 15.1. Replace "$\tilde{f}(x) = f(\tilde{x})$" with $\tilde{f}(x) = \tilde{f}(x) - f(\tilde{x}) + f(\tilde{x})$. You will need the triangle inequality in addition to steps already in the proof of Theorem 15.1. Make it clear how the constant "γ" arises; what does it depend on?