Homework 4

Due: Wednesday 11:59pm, Nov 13, 2024, on Gradescope

Notes: Please read the syllabus for information on programming expectations, the late homework policy, and the collaboration policy. Please submit both the written and coding portions of your homework on Gradescope. The written portion should be submitted as a PDF (prepared however you like, but LaTeX is excellent), with plots and tables in the appropriate places. Results and plots requested in the problems should be placed in your write-up—you should not simply refer to your code. Please mark the location of each problem in your Gradescope submission—this makes it much easier for us to grade.

Plotting and formatting: Put thought into presenting your data and results in the clearest way possible. Think about what information you want to convey, and how to convey it most effectively. Sometimes, using a table can be useful, but at other times data is best presented using a plot. All tables and plots should be accompanied by some discussion of what we are intended to learn from them.

Choose proper ranges and scales (e.g., semilogx, semilogy, loglog), always label axes, and give meaningful titles. Make sure your axis numbering and labels are large enough to be readable—getting this right will require some exploration in MATLAB, or whatever language you are using. If you do print numbers, in MATLAB for example, use `fprintf` to format the output nicely. Use `format compact` and other format commands to control how MATLAB prints things. When you create figures using MATLAB (or Python or Julia), please try to export them in a vector graphics format (.eps, .pdf, .dxf) rather than raster graphics or bitmaps (.jpg, .png, .gif, .tif). Vector graphics-based plots avoid pixelation and thus look much cleaner.

Code: Try to write clean, concise, and easy-to-read code, with meaningful variable names. Your code must be well-commented. Every line of code should be explained by comments inside the code, unless it is absolutely self-explanatory. Commenting your code is like "showing your work", and grading will take this into account in a similar manner.

You should submit three code files for this assignment: `prob1.m`, `rqi_deflation.m`, and `mycg.m`.

---

1. This problem will explore the convergence rate of the power method, inverse iteration, and Rayleigh quotient iteration. Submit a code, `prob1.m`, which implements the different parts of this problem.

   (a) **[5 pts]** Generate a random symmetric $n \times n$ matrix A with eigenvalues $\lambda_k = 1/k$ for $k = 1, \ldots, n$, for $n = 100$, using the following strategy. Begin with a diagonal matrix Λ with diagonal entries $\Lambda_{kk} = \lambda_k$. Then generate a random orthogonal matrix Q by computing the QR factorization of a random matrix. Finally, set $A = Q\Lambda Q^T$. Using MATLAB's `eig` function, compute the true eigenvalues of A, and verify that they given by $\lambda_1, \ldots, \lambda_n$. Report the maximum absolute error

between the true eigenvalues and the computed eigenvalues here. Note: you will probably need to sort the output of `eig` using MATLAB's `sort` function.

(b) **[5 pts]** Which eigenvalue/eigenvector pair do you expect the power method to converge to for this matrix? How do you expect the Euclidean norm error e_k of the estimated eigenvector to decay as a function of k? Implement the power method, and run it for this problem until this error is less than 10^{-12} (note that you have the true leading eigenvector to compare against: it is the first column of your random orthogonal matrix Q!). Plot the error against the iteration number. Also plot your prediction of the error e_k, and discuss whether or not the error behaves as you expect. Note: There is a subtlety involved in measuring the error of the leading eigenvector. Indeed, if v is a normalized eigenvector, $-v$ is also a normalized eigenvector. Thus, there is a sign ambiguity in the normalized eigenvector. You can remove this ambiguity by ensuring that $v_1 \geq 0$; simply replace v by $-v$ if $v_1 < 0$. You can do this both to the known true eigenvector, and to your estimated eigenvector at each iteration of the power method.

(c) **[5 pts]** Use the power method with deflation to compute the second largest eigenvalue of A. Include a re-orthogonalization step at every iteration to avoid stability issues. How do you expect the eigenvector error to decay as a function of k? Again, run your code until this error is less than 10^{-12}, and plot the error against the iteration number, along with your prediction of the error.

(d) **[5 pts]** Implement inverse iteration to compute the eigenvalue closest to $s = 51/1000$, which is $\lambda_{20} = 1/20$. How do you expect the eigenvector error to decay as a function of k? Run your code until this error is less than 10^{-12}, and plot the error against the iteration number, along with your prediction of the error. To solve each linear system, you can use MATLAB's backslash command (even though this might not be the most efficient scheme).

(e) **[5 pts]** Implement Rayleigh quotient iteration with initial guess $\lambda^{(0)} = 51/1000$. Plot the eigenvector error against the iteration number. How many iterations does it take before the error is less than 10^{-12}, compared with inverse iteration? You can again use MATLAB's backslash command to solve the linear systems. Note: I recommend turning off MATLAB's ill-conditioned matrix warning using the command `warning('off', 'MATLAB:nearlySingularMatrix')`; Rayleigh quotient iteration involves solving ill-conditioned linear systems, but in this case, for rather subtle reasons, the ill-conditioning is not a problem.

2. (a) **[20 pts]** Write and submit a function, `rqi_deflation.m`, implementing Rayleigh quotient iteration with deflation to compute all the eigenvalues and eigenvectors of a symmetric matrix. The eigenvectors should be normalized to 1 in the Euclidean norm. Your function should include an error tolerance, such that an eigenvalue estimate is accepted once the Euclidean norm difference between the eigenvector estimate and the previous eigenvector estimate is less than the tolerance. (Remember the sign ambiguity involved in measuring the error between computed eigenvectors, discussed above. You should implement the same solution here.) You can also include a parameter specifying the maximum number of Rayleigh

quotient iterations per eigenvalue, and print an error message if the maximum number of iterations is reached before the error tolerance is met. Your function should return a vector containing all of the computed eigenvalues, and a matrix containing the corresponding normalized eigenvectors as its columns. Your method should perform a re-orthogonalization step at every iteration, which will need to re-orthogonalize against the *subspace* spanned by the already-converged eigenvectors: you will need to recall the matrix of projection onto a subspace spanned by a given collection of orthonormal vectors.

(b) **[5 pts]** Use your function to compute all the eigenvalues of the 100×100 matrix given in the previous problem. Set the error tolerance to 10^{-12}, and the maximum number of iterations per eigenvalue to 100. Make a plot showing the absolute error of each of the 100 computed eigenvalues (you will need to sort your computed eigenvalues using MATLAB's `sort` function).

3. (a) **[5 pts]** Show that the centered finite difference approximation to the second derivative of a smooth function $u(x)$ introduced in class is second-order accurate: that is, show that

$$\left| u''(x) - \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \right| = \mathcal{O}(h^2).$$

(b) **[10 pts]** Using the method described in class, compute an approximation of the one-dimensional electric potential $u(x)$ for $x \in [0,1]$ due to a Gaussian charge density $q(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$ with $\mu = 1/2$ and $\sigma = 0.05$, with a perfect conductor in $(-\infty, 0)$ and $(1, \infty)$: that is, solve

$$-u''(x) = q(x), \quad u(0) = u(1) = 0.$$

Plot the solution $u(x)$ on $[0,1]$ using $h = 1/(n+1)$ with $n = 100$. Bonus **[3 pts]**: Give a physical interpretation of your result.

(c) **[10 pts]** The "particle in a box" is a famous simple model in quantum mechanics, and in fact described the origin of energy quantization. We consider a quantum particle (say, an electron) in a one-dimensional box $[0,1]$, with hard walls at $x = 0$ and $x = 1$. From physical considerations, it can be shown that this model is mathematically described by the eigenvalue equation

$$-u''(x) = \lambda u(x), \quad u(0) = u(1) = 0.$$

The eigenvalues λ represent the quantum mechanically-allowed energies of the particle, which are discrete ("quantized"). The corresponding eigenfunction $u(x)$ is the eigenfunction of a particle with energy λ, and $|u(x)|^2$ is interpreted as the probability density (if it is properly normalized to integrate to 1) of finding a particle with energy λ at position x.

Using an approach similar to that described in class for electrostatics, show how to approximate this eigenvalue problem using the centered second-order finite difference approximation above as a matrix eigenvalue equation. Using MATLAB's `eig`

function, compute the first 3 eigenvalues and eigenvectors. Report the first three eigenvalues λ_1, λ_2, and λ_3, and plot the first 3 eigenfunctions $u_1(x)$, $u_2(x)$, and $u_3(x)$ (the so-called ground state, and first two excited states). On another plot, plot the probability densities $|u_1(x)|^2$, $|u_2(x)|^2$, and $|u_3(x)|^2$ (technically, these will be a constant multiple of the probability density, unless you properly normalize them to integrate to 1). Does a quantum-mechanical particle in the ground state have the same probability of being found anywhere in the box? What about a particle in the first two excited states? Whether you know it or not, you have just given your first demonstration of the concepts of *quantization*, and *quantum interference*.

4. (a) **[15 pts]** Implement the conjugate gradient method to solve $Ax = b$ in a function `mycg.m`, and submit this. Your function should take as input a symmetric positive definite matrix A, a right hand side b, an initial guess $x^{(0)}$, an error tolerance, and a maximum number of iterations. It should return the approximate solution x, and a vector containing the estimated errors e_k at each iteration, estimated as the $||\cdot||_A$ norm difference between successive iterates. The function should terminate when either e_k is less than the error tolerance, or the maximum number of iterations has been reached. Make sure to write your implementation so that it only requires a single matrix-vector product per iteration (except for computing the error estimate, which also requires another matrix-vector product), and only stores a fixed number of vectors in memory.

   (b) **[10 pts]** Generate two random 1000×1000 symmetric positive-definite matrices A, using the method described in Problem 1. The first should have random eigenvalues in $[1, 4]$, and the second should have random eigenvalues in $[1, 100]$ (to do this, you will have to figure out how to generate random numbers in a given interval $[a, b]$). How do you expect the errors $\|e_k\|_A$ of the conjugate gradient method to decrease with the iteration number k for each of these matrices? Solve $Ax = b$ (for randomly-generated b with $\|b\|_2 = 1$) in each case using your implementation of the conjugate gradient method, with a tolerance 10^{-12}, and plot the errors $\|e_k\|_A$ against the iteration number, along with your predictions of the errors for both matrices.