Homework 5, CS 6210, Fall 2020
Instructor: Austin R. Benson
Due November, 13, 2020 at 10:19am ET on CMS (before lecture)

Policies

Collaboration. You are encouraged to discuss and collaborate on the homework, but you have to write up your own solutions and write your own code.
Programming language. You can use any programming language for the coding parts of the assignment. Code snippets that we provide and demos in class will use Julia.
Typesetting. Your write-up should be typeset with LaTeX. Handwritten homeworks are not accepted.
Submission. Submit your write-up as a single PDF on CMS: https://cmsx.cs.cornell.edu.

Problems

1. *Richardson PageRank iteration.*

   Let $P \in \mathbb{R}^{n \times n}$ be a column-stochastic matrix, meaning that the entries in each column of P are nonnegative and sum to one. You can think of P as representing the probabilities of a discrete-time Markov chain, where $P_{ji} = \Pr(X_{t+1} = j \mid X_t = i)$. The PageRank vector x_{pr} is the solution to the linear system

   $$(I - \alpha P)x_{\text{pr}} = (1 - \alpha)v,$$

   where v is a stochastic vector (nonnegative and sums to one) and $\alpha \in (0, 1)$. The solution x_{pr} is also a stochastic vector, as it represents the stationary distribution of the Markov chain defined by the column-stochastic matrix $\alpha P + (1 - \alpha)ve^T$, where e is the vector of all ones.

   Given a linear system $Ax = b$, the Richardson iteration is the iterative method

   $$x^{(k+1)} = x^{(k)} + \omega(b - Ax^{(k)}),$$

   where ω is a scalar.

   Show that when $\omega = 1$ and $x^{(0)} = 0$, the Richardson iterates on the PageRank system satisfy $x_{\text{pr}} - x^{(k)} \geq 0$ and $\|x_{\text{pr}} - x^{(k)}\|_1 = \alpha^k$.

2. *Bisection.*

   (a) The *interia* of symmetric matrix $A \in \mathbb{R}^{n \times n}$ is a 3-tuple (n_-, n_0, n_+), where n_- is the number of negative eigenvalues of A, n_0 is the number of zero eigenvalues of A, and n_+ is the number of positive eigenvalues of A. Show that for any nonsingular X, the inertia of $X^T AX$ is equal to the inertia of A.

   (b) Let $A \in \mathbb{R}^{n \times n}$ be tridiagonal and symmetric. Suppose that we can factorize $A - zI = LDL^T$ for some scalar z, where L is unit lower triangle and D is diagonal with $D_{i,i} \neq 0$. Following Gaussian elimination, show that the entries of D are given by the recurrence

   $$D_{1,1} = A_{1,1} - z, \quad D_{i+1,i+1} = A_{i+1,i+1} - z - \frac{A_{i,i+1}^2}{D_{i,i}}.$$

   Thus, we can compute the D in $O(n)$ time.

   (We should worry about numerical stability given that we are not pivoting. Remarkably, this procedure is actually stable, since A is tridiagonal. We should also worry about encountering zeros on the diagonal, but it turns out there is a way to deal with this, too.)

(c) For a given scalar z, what do the diagonal entries of D from part (b) say about the number of eigenvalues of A that are less than, equal to, or greater than z?

(d) Sketch an algorithm that takes as input an interval $[a, b)$, a symmetric tridiagonal matrix A, a tolerance τ, and outputs a partition of $[a, b)$ into sub-intervals that

- contain no eigenvalues of A;
- contain exactly one eigenvalue of A; or
- are smaller than τ.

This is the idea of the *bisection algorithm* for computing eigenvalues of A in some interval.

How can you use your algorithm if A is a general symmetric matrix (i.e., not necessarily tridiagonal)?

(e) In part (d), if τ is reasonably small and we have an interval that contains an eigenvalue, the midpoint of that interval is a good estimate for an eigenvalue. We can then use inverse iteration to compute a corresponding eigenvector quickly.

Implement inverse iteration and confirm with your code that, given a good initial estimate of an eigenvalue, you get a good estimate of an eigenvector.

(No need to turn anything in for this part.)

(f) One problem with the procedure in part (e) for computing multiple eigenvectors is that they may not be close to orthogonal, even if they are individually close to eigenvectors. Make a plot to demonstrate this by computing several eigenvectors with inverse iteration from several eigenvalue estimates, showing that the problem gets worse as the eigenvalue estimates are closer together. You do not need to get the estimates from bisection — you can just call a library routine to get the true eigenvalues and create "synthetic" estimates.