Final Exam

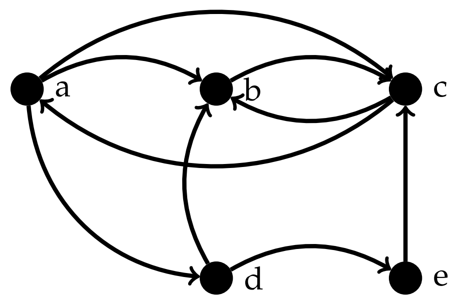
Due Thursday 17 December, 2015, at NOON in my box in Chapman 101

150 points total

As stated on the syllabus, this Final is 25% of your course grade.

Rules. You may use any reference book or reference article, print or electronic, as long as it is clearly cited. (References to the textbook itself should be used as needed for clarity, but are not required.) You may not, however, search out complete or substantial solutions to these particular problems. You may not talk or communicate about this exam with any person other than Ed Bueler. This exam is your own work.

F1. (*15 pts*) Consider the graph of five web pages a,b,c,d,e shown below, where each page links to some of the other pages as shown. The goal of this problem is to compute the Google PageRank$^{\text{TM}}$ of the pages.



I handed out the original technical report[1] by Page and Brin and others, but their description of the calculation is confusing. For a clearer alternative written in 2002 by Cleve Moler, go to

`www.mathworks.com/company/newsletters/articles/the-world-s-largest-matrix-computation.html`

He shows how to build a matrix G with only entries 0 and 1. Compute G for the "web" above. Then he shows how to build a matrix A for which an eigenvector gives the PageRank$^{\text{TM}}$s. Build A from G, using $p = 0.85$ as suggested, for the "web" above. Finally, use MATLAB's `eig` to compute the PageRank$^{\text{TM}}$s. (*Hint.* Pay attention to how the eigenvector is normalized.) Explain why one webpage gets the largest PageRank$^{\text{TM}}$.

Extra Credit. (*3 pts*) For the above problem, build a random walk program which traverses your "web", as would a Google "bot", and builds an estimate of PageRank$^{\text{TM}}$ based on the fraction of the time spent at a given page. It is up to you how to represent a "web" in some data structure; it does not need to be the matrix G. (*This is how PageRank$^{\text{TM}}$ is actually computed. The matrix A is never formed and no explicit eigenvalue algorithm is used.*)

[1] L. Page and others, *The PageRank citation ranking: . . .* , Stanford InfoLab Technical Report 422, 1999.

F2. (*5 pts*) Consider the following 4×4 matrix A:

$$A = \begin{bmatrix} 2 & 3 & 5 & 7 \\ -1 & 2 & 3 & 5 \\ -1 & -1 & 2 & 3 \\ -1 & -1 & -1 & 2 \end{bmatrix}$$

Find the rank 2 matrix B which is closest to A in 2-norm. (*Hint*. This is intended to be a very easy question if you remember ideas from the beginning of the semester. Yes, you can use MATLAB but make it clear what you are doing and why.)

F3. **(a)** (*0 pts*) Download and play with my codes `tridiag.m`, `qralg.m`, and `eigherm.m` from the `.zip` archive posted at

`bueler.github.io/M614F15/matlab/qr-for-eigs.zip`

Make sure you agree that I solved all parts of Exercise 29.1. Make sure that you basically understand what is going on. (*There is nothing to turn in on this part.*)

(b) (*10 pts*) Using the matrix form H in equation (31.2), which is correct even when A is non-square, implement a program `mysvd.m` that calls `eigherm.m` to compute the singular values of any real matrix $A \in \mathbb{R}^{m\times n}$. (*Hint*. My code is 4 lines long. Yours should be comparably short or you are doing something wrong.) Show it works by comparing its results to those from MATLAB's built-in `svd()` on A in problem **F2**. Also show it works on some random non-square matrix of size 6×4; how big is H in this case?

F4. (*15 pts*) Theorem 15.1, the "Fundamental Theorem of Numerical Linear Algebra," assumes your algorithm is *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 which was nearly as strong as Theorem 15.1. Show the following:

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*

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O\left((\kappa(x) + \gamma)\epsilon_{machine}\right)$$

as $\epsilon_{machine} \to 0$.

(*Hints*. Start by understanding/recalling the proof of Theorem 15.1. Note that a first step in that proof is to use the fact "$\tilde{f}(x) = f(\tilde{x})$." Replace that fact 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 sure it is clear how the constant "γ" arises.)

F5. (*10 pts*) Implement Algorithm 27.3. Specifically, run the loop until $A - \lambda^{(k-1)} I$ is so badly conditioned that you must stop; what is a cheap way to determine this? Show it works by producing the iterates $\lambda^{(0)}$, $\lambda^{(1)}$, $\lambda^{(2)}$ in Example 27.1.

F6. (a) (*10 pts*) Implement s steps of power iteration, Algorithm 27.1, in a MATLAB function `[lam,v] = powerit(A,v0,s)`. Test it using `v0=randn(3,1)` on the matrix A shown in Example 27.1 in the textbook, the same matrix considered in problem **F5**. By comparing to an eigenvalue computed using `eig()`, measure the rate of convergence in a meaningful way, and show a graph.

(b) (*10 pts*) Apply `powerit()` to the matrix A in problem **F2**, using `v0=randn(4,1)`. Explain what is happening; show evidence for your explanation.

(c) (*10 pts*) Revise your implementation of `powerit()` so that it only applies A to a vector once per iteration. (Specifically, do not do a matrix-vector multiplication in the "apply A" step if it already happened in the "Rayleigh quotient" step.) Suppose $A \in \mathbb{R}^{m\times m}$. Exactly how many floating-point operations are needed to do s steps of power iteration?

(d) (*10 pts*) Suppose A is tridiagonal. Recompute the work estimate in **(c)**. What speed-up can be achieved if $m = 10^4$?

F7. (a) (*10 pts*) Reproduce Figure 12.1 using MATLAB's `roots` command. You will need to start by computing the exact coefficients a_k of the polynomial (12.8).

(b) (*10 pts*) In your own words, write a paragraph or two—say, at least six good sentences but at most half a page—on how `roots` works and to what degree it succeeds in getting accurate estimates of the roots of a polynomial from its coefficients.

In particular, consider this extract from the MATLAB technical documentation:[2]

> *The roots of the polynomial are calculated by computing the eigenvalues of the companion matrix A:*
>
> ```
> A = diag(ones(n-1,1),-1);
> A(1,:) = -c(2:n+1)./c(1);
> eig(A)
> ```
>
> *The results produced are the exact eigenvalues of a matrix within roundoff error of the companion matrix. However, this does not mean that they are the exact roots of a polynomial whose coefficients are within roundoff error of those in p.*

You should answer these questions: In terms of those in Trefethen & Bau, what algorithms do you think `eig` uses? What theorem in the textbook comes closest to showing that the "results produced are the exact eigenvalues of a matrix within roundoff error of the companion matrix"? How would you want to generalize that theorem for the current problem? How does the conditioning of the problem(s) affect the answer?

Exercise 21.4 in Lecture 21. (*10 pts*) Do part **(a)** only.

Exercise 22.2 in Lecture 22. (*10 pts*)

Exercise 23.2 in Lecture 23. (*15 pts*)

[2] See www.mathworks.com/help/matlab/ref/roots.html.