Final Exam
(take home)

Due Wednesday 8 December, 2021, at 5pm

Submit by email `elbueler@alaska.edu` or my Chapman 101 box

100 points total

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

Rules. You may use reference books or reference articles, print or electronic, *but they must be clearly cited*. References to the textbook should be used as needed for clarity. You **may not** search for complete or substantial solutions to these particular problems. You **may not** talk about this exam, electronically or otherwise, with anyone other than Ed Bueler. **This exam is your own work.**

F1. (*15 pts*) Over the course of the semester, several times we have used a two-word phrase (2WP) to describe a factorization. In fact there are 15 possible 2WPs which can be formed from a column-A adjective and a column-B noun from this table:

column A	*column B*
triangular	triangularization
orthogonal	orthogonalization
unitary	diagonalization
	tridiagonalization
	hessenbergization

(a) Consider only the textbook Lectures which we covered, namely Lectures 1–17 and 20–27. For square matrices, which distinct factorizations *do* use, or *easily could* use, one of these 15 possible 2WPs? For each such factorization, state the matrix factorization, its name and/or algorithm reference(s) in the textbook, and its 2WP(s).

(b) The adjectives "orthogonal" and "unitary" are essentially synonyms. However, the textbook systematically uses "unitary" for a certain kind of factorization. Explain, in a few sentences, the distinct purpose of the "unitary" factorizations.

(c) (*1 pt* ***Extra Credit***) The SVD factorization is *not* one of the answers in part **(a)**. Why? Invent a good 2WP to describe it.

(d) (*3 pts* ***Extra Credit***) Create, implement, and test an algorithm corresponding to a 2WP which is unused. That is, invent an algorithm which is *not* in the textbook and which is *not* listed in your part **(a)** answer.

Hints. In **(a)** I found 8 distinct factorizations corresponding to 7 distinct 2WPs. The matching is not one-to-one, but fairly close. You can get full credit in part **(a)** without perfect agreement with my list, but not if you miss important factorization ideas.

F2. (*15 pts*) The producers and I believe we have a great new reality series, *Naked and Calculating*. (It will do great in the North Korean market . . . little competition.) The plan is to have three contestants, each seeking riches, on three islands. Each contestant will have unlimited supply of pencils and paper, plus adequate food and drink, but nothing else. Naturally, there are hidden cameras so we can watch the exciting action!

Each island has a problem, and algorithm which the contestant *must* use:

island 1: Compute the determinant of an $m \times m$ matrix using expansion in minors.
island 2: Solve an $m \times m$ upper-triangular linear system using back-substitution.
island 3: Solve an $m \times m$ linear system using Gaussian elimination.

After finding-out which one is their island, a contestant chooses the m value, but then the producers will choose generic values to fill the matrix entries. By choosing m the contestants are gambling that they can compute the correct answer, *by hand*, in one month of work. In fact, at the end of one month a contestant either gets 2^m US dollars for *correctly* computing the solution of a problem, or zero dollars if their answer is incorrect. (All entries in the final answer have to be correct to three digits.)

(a) As a contestant, which island would you choose? Most avoid? Explain.

(b) For *each* island, state the specific m you would choose if you are put on that island. You must justify your choices via a *quantitative*, though necessarily speculative, explanation.

(c) For excitement, at the last minute before going on their island, after already having chosen an m value, contestants are told that in fact they *can* choose their algorithm, and that they can revise their m choice. For one of the islands this represents a huge improvement in the pay-out. Explain; give a new m value with explanation.

Hints. Algorithm stability is *not* the issue. Conditioning is *not* the issue. Work estimates! Good writing is needed for full credit in all parts, but a correct answer exists to part **(a)**. Parts **(b)** and **(c)** require that you give *specific values* of m based on a *quantitative* and *reasonable* estimate of how fast you can do arithmetic by hand. (There is no one correct value for m, but you must choose an m and justify it!) Note that you, as a contestant, would want to re-check your work *many times*! Expansion in minors appeared on the solutions to Assignment #2. Don't worry about the losing contestants; this is not *Squid Game*!

F3. (*10 pts*) Consider the following $A \in \mathbb{R}^{3\times 4}$:

$$A = \begin{bmatrix} 2 & 3 & 5 & -1 \\ -1 & 7 & 3 & 5 \\ -1 & -1 & 2 & 4 \end{bmatrix}$$

Find the matrix $B \in \mathbb{R}^{3\times 4}$ of rank 2 which is closest to A in 2-norm.

Hint. This easy question arose near the beginning of the semester. Yes, you can use MATLAB, but make it clear what you are doing and why.

F4. (*10 pts*) Expansion in minors may be a terrible algorithm for computing determinants, but it is backward-stable. Prove this in the 1×1 and 2×2 matrix cases, assuming as usual that your computer satisfies axioms (13.5) and (13.7). (*An induction argument would show it in general, but this is not requested.*)

F5. (*20 pts*) *Recall the following ideas about our most trustworthy linear solver: Algorithm 10.1 is Householder triangularization $A = QR$. Code* `house.m` *implements this;* read and understand it! *It outputs a lower-triangular matrix W containing the "v_k" vectors, and an upper-triangular R, but of course $WR \neq A$. Next, Alg. 10.2 implements the action of Q^* from the vectors stored in W. Alg. 16.1 adds back-substitution (Alg. 17.1) to give a solver for square, nonsingular linear systems $Ax = b$. Theorem 16.2 shows that Alg. 16.1 is backward-stable.*

Implement Alg. 16.1, but do it *in-place*. In particular, write a Matlab function

```
x = inhousesolve(A,b)
```

This function should start by checking that the inputs make sense, i.e. A is square ($m \times m$) and b is a compatibly-sized column vector. After that, the next line of your code should *append one row of space to the bottom of the array* `A`, like this

```
A = [A; zeros(1,m)];
```

(*You will need the appended space!*) From now on, the only matrix or 2D array in your code is `A` itself, and `A` does not change size. You will modify entries of the array `A`, and the goal is to construct the vector `x`. Inside your function you will implement Alg. 10.1 by modifying the entries of `A` to store W and R, implement Alg. 10.2 by referring to entries of the array `A`, and implement Alg. 17.1 by referring to other entries of `A`.

Requirements. Do not use backslash, nor any built-in matrix factorization. Use only matrix/vector addition and multiplication, `for` loops, and colon notation.

Hint. Start your coding by modifying `house.m` to do its operations in-place on the padded `A` array. Check that you get the same W and R, but "stacked" in `A`.

Weird kind of moral support. Your resulting code will definitely be seen as a *black box* by the uninitiated. Just from the source code, someone who does not know Householder ideas will wonder what magic is occurring!

F6. (*15 pts*) **(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 (next part) if `stages` is `true`, and finally return a Hessenberg matrix H such that $A = QHQ^*$ for some unitary Q. Note that your code can throw-away the vectors v_k; they are not returned.

(b) For a specific 5×5 matrix A of your choice, run the code and show the four stages A, $Q_1^*AQ_1$, $Q_2^*Q_1^*AQ_1Q_2$, and $H = Q_3^*Q_2^*Q_1^*AQ_1Q_2Q_3$. That is, make concrete the cartoons on pages 197–198, "A Good Idea".

(c) For the same 5×5 matrix A, use the built-in `eig()` to show that the eigenvalues of A and H are the same within rounding error. Now construct a new 4×4 Hermitian matrix S and compute `T=hessen(S)`. Check that T is tridiagonal and Hermitian. Then show the eigenvalues of S and T are the same within rounding error.

F7. (*15 pts*) Implement Algorithm 27.3, namely Rayleigh quotient iteration. In particular, write a Matlab code with signature

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

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

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

(Using Matlab documentation, explain what this criterion means.) 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 of a random 20×20 Hermitian matrix. (For *(ii)*, assume that the built-in `eig()` is exact.)

Extra Credit. (*5 pts*) Theorem 15.1 assumes that your algorithm is *backward* stable. But what if it is merely "stable", i.e. according to the definition given in Lecture 14? To my surprise, I was able to prove a theorem which was 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 in the Thm. 15.1 proof. Make it clear how the constant "γ" arises.