HOMEWORK
David F. Gleich
November 17, 2023

Homework 5

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Gradescope, around 4am on Monday November 20nd

Remember that this is a graduate class. There may be elements of the problem statements that require you to fill in appropriate assumptions. You are also responsible for determining what evidence to include. An answer alone is rarely sufficient, but neither is an overly verbose description required. Use your judgement to focus your discussion on the most interesting pieces. The answer to "should I include 'something' in my solution?" will almost always be: Yes, if you think it helps support your answer.

Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

Problem 1: Updating factorizations of linear systems of equations

In class, we showed how to solve $(\boldsymbol{A} + \mathbf{u}\mathbf{v}^T)\mathbf{x} = \mathbf{b}$ when given a fast factorization method to solve $\boldsymbol{A}\mathbf{x} = \mathbf{b}$. In this problem, we will address the question of how to update the factorization itself. Suppose that we are given a Cholesky factorization of $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$ as we saw in class. Show how to update this factorization to produce $\boldsymbol{L}'\boldsymbol{D}'\boldsymbol{L}'^T = (\boldsymbol{A} + \mathbf{u}\mathbf{u}^T)$. Your algorithm should do no more than $O(n^2)$ work.

Problem 2: Matrix functions in practice

In general, we don't compute Matrix functions by taking their eigenvalues and applying a function to their eigenvalues – this is just a nice theory that works to explain what's going on in the simple case.

Instead, we create customized algorithms to compute a function of a matrix. One such algorithm that computes the cosine of a matrix was proposed by Serbin and Blalack and analyzed by Higham and Smith in 2003 https://login.ezproxy.lib.purdue.edu/login?url=https://link.springer.com/article/10.1023/A:1026152731904

The idea is that use the cosine double angle formula:

$$\cos(2\boldsymbol{A}) = 2\cos(\boldsymbol{A})^2 - \boldsymbol{I}$$

combined with some way of estimating $\boldsymbol{C}_0 = \cos(2^{-m}\boldsymbol{A})$. Then, we can compute $\cos(2^{-m+1}\boldsymbol{A}) = 2\boldsymbol{C}_0^2 - \boldsymbol{I}$. Also, for $\cos(2^{-m}\boldsymbol{A})$ then we should be able to use a simple formula to get a good enough approximation.

1. Show that the double angle formula holds for symmetric matrices $\boldsymbol{A}$.

2. In the paper, they show that if $\|\boldsymbol{A}\|_\infty \leq 1$, then there is a specific type of approximation of $\cos(\boldsymbol{A})$ called a Pad'{e} approxiomation that gives an answer down to machine precision. The specific approximation is

$$\cos(x) \approx r_{88}(x) = \frac{1 - \frac{260735}{545628}x^2 + \frac{4375409}{141863280}x^4 - \frac{7696415}{13108167072}x^6 + \frac{80737373}{23594700729600}x^8}{1 + \frac{12079}{545628}x^2 + \frac{34709}{141863280}x^4 + \frac{109247}{65540835360}x^6 + \frac{11321}{1814976979200}x^8}.$$

   When applied to a matrix with $\|\boldsymbol{A}\|_\infty \leq 1$, this approximation guarantees

$$\frac{\|\cos(\boldsymbol{A}) - r_{88}(\boldsymbol{A})\|_\infty}{\|\cos(\boldsymbol{A})\|_\infty} \leq 3.26 \times 10^{-16}.$$

   Note that this is a rational function $r_{88}(x) = p(x)/q(x)$ for some polynomials $p(x)$ and $q(x)$. One way to evaluate this for a matrix is compute $p(\boldsymbol{A}) \cdot q(\boldsymbol{A})^{-1}$.

   Based on r_{88}, let's create values π and μ so that

$$p(x) = \pi_0 + \pi_2 x^2 + \pi_4 x^4 + \pi_6 x^6 + \pi_8 x^8$$

$$q(x) = \mu_0 + \mu_2 x^2 + \mu_4 x^4 + \mu_6 x^6 + \mu_8 x^8$$

   Then to evaluate $r_{88}(\boldsymbol{A})$ we have the algorithm.

```
A2 = A*A
A4 = A2*A2
A6 = A4 * A2
A8 = A4 * A4
P = pi0*I + pi2*A2 + pi4*A4 + pi6*A6 + pi8*A8
Q = mu0*I + mu2*A2 + mu4*A4 + mu6*A6 + mu8*A8
R = Q\P
```

   Implement this algorithm and show that for matrices where $\|\boldsymbol{A}\|_\infty \leq 1$ we have the given bound on accuracy. (**Instruction note: While I have tried to be careful to check the accuracy of the coefficients, we will verify this ourselves and let you know if there are issues and note any correction to the problem.**)

   You may assume that calling $\cos(\boldsymbol{A})$ in Julia is accurate.

3. Show how to determine the smallest m such that $\|2^{-m}\boldsymbol{A}\|_\infty \leq 1$.

4. Implement the Serbin/Blalack/Higham/Smith procedure to compute $\cos(\boldsymbol{A})$, i.e.

```
compute m such that ||2^{-m} A||_oo <= 1
compute R = r_88(2^{-m} A)
for i=1:m
    R = 2*R*R - I
end
return R
```

   For a 1000 by 1000 matrix, compare the runtime to compute $\cos(\boldsymbol{A})$ via a spectral eigenvalue decomposition.

Problem 3: Weighted orthogonality for SVD.

One possible weighted generalization of the SVD involves producing a factorization

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where $\boldsymbol{U}^T\boldsymbol{R}\boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{V}^T\boldsymbol{S}\boldsymbol{V} = \boldsymbol{I}$, where $\$\boldsymbol{R}\$$ and $\boldsymbol{S}$ are symmetric positive definite matrices.

A common theme in solving more advanced problems is converting or translating them into problems that we know how to solve.

Show how to use a standard SVD computation to produce this weighted SVD factorization.

Problem 4: Implementing a rank-k solution update.

Given $\mathbf{x}$ and an LU factorization of $\boldsymbol{A}$, then updating the solution of a linear system $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ to a new solution $(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{V}^T)\mathbf{y} = \mathbf{b}$ can be done fairly efficiently.

Write code or fairly detailed pseudo-code to do this. This uses the Julia factorization `F` which enables you to solve systems with $\boldsymbol{A}$ without recomputing the factorization.

```
function update(x::Vector, b::Vector, F::LUFact, U::Matrix, V::Matrix)
end
```

Problem 5: Adding and deleting an equation

Recall that a linear system represents the simultaneously solution of a set of linear equations

$$\begin{aligned}\mathbf{a}_1^T\mathbf{x} &= b_1\\ \mathbf{a}_2^T\mathbf{x} &= b_2\\ &\vdots\\ \mathbf{a}_n^T\mathbf{x} &= b_n.\end{aligned}$$

Consider a set of n equations and n unknowns with a unique solution for any possible set of values $b_1, \ldots, b_n$.

Let `alg`($\mathbf{b}$) be an algorithm to solve for $\mathbf{x}$ when given $b_1, \ldots b_n$.

Show how to use `alg` in order to solve for $\mathbf{y}$ in the system of equations

$$\begin{aligned}\mathbf{c}_1^T\mathbf{y} &= d_1\\ \mathbf{a}_2^T\mathbf{y} &= b_2\\ &\vdots\\ \mathbf{a}_n^T\mathbf{y} &= b_n.\end{aligned}$$

That is, we deleted the equation with $\mathbf{a}_1$ and added a new equation with $\mathbf{c}_1$ instead. If you use `alg` too many times, you may lose points. Be efficient!

(Hint: this is essentially a special case of other problems on this homework or other things we've seen in class.)

Problem 6: The Krylov subspace and eigenvalue algorithms

In class, we showed that the Krylov subspace:

$$\mathbb{K}_k(\boldsymbol{A}, \mathbf{b}) = \text{span}(\mathbf{b}, \boldsymbol{A}\mathbf{b}, \ldots, \boldsymbol{A}^k\mathbf{b})$$

arises when solving $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ via the simple Neumann series method. Here, we note that it also arises in the power-method to identify the largest eigenvalue of a matrix. Note that the simple monomial basis for the Krylov subspace is:

$$\boldsymbol{X}_k = \begin{bmatrix}\mathbf{b} & \boldsymbol{A}\mathbf{b} & \ldots & \boldsymbol{A}^k\mathbf{b}\end{bmatrix}$$

In this case, the power-method uses the vector $\boldsymbol{X}_k \mathbf{e}_k$ as the estimate of the largest eigenvalue.

1. State an algorithm to search the Krylov subspace for the largest eigenvalue of a symmetric matrix using the fact that the largest eigenvalue solves the problem:
$$\lambda_{\max} = \begin{array}{ll} \text{maximize} & \mathbf{x}^T \boldsymbol{A} \mathbf{x} \\ \text{subject to} & \|\mathbf{x}\| = 1. \end{array}$$
(Note, that you are welcome to use ideas that we discuss in subsequent lectures, but you don't have to. Everything can be done with the material presented through lecture 21.) Note that you may use any routine you want on a $(k+1)$-by-$(k+1)$ matrix.
2. Implement your algorithm and compare the eigenvalue estimates for a few small matrices. (Hint, a good idea here would be to show convergence plots for the error in the largest eigenvalue compared with iterations – usually on a log-y-scale.)
3. Comment on any issues that arise in your implementation.