18.335 Midterm Solutions, Fall 2010

Problem 1: SVD Stability (20 points)

Consider the problem of computing the SVD $A = U\Sigma V^*$ from a matrix A (the input). In this case, we are computing the function $f(A) = (U,\Sigma,V)$: the outputs are of the SVD are 3 matrices, i.e. a triple (U,Σ,V) and *not* just the product $U\Sigma V^*$.

(a) In floating-point, we compute $\tilde{f}(A) = (\tilde{U},\tilde{\Sigma},\tilde{V})$. If this were backwards stable, there would be some δA with $\|\delta A\| = \|A\|O(\varepsilon_{\text{machine}})$ such that $f(A+\delta A) = (\tilde{U},\tilde{\Sigma},\tilde{V})$, i.e. $(\tilde{U},\tilde{\Sigma},\tilde{V})$ are the exact SVD of $A+\delta A$. Note that this is a much stronger statement than simply that $A+\delta A = \tilde{U}\tilde{\Sigma}\tilde{V}^*$!

(b) For it to be backwards stable, $(\tilde{U},\tilde{\Sigma},\tilde{V})$ would have to be the exact SVD of *something*, which would mean that $\tilde{U}$ and $\tilde{V}$ would have to be exactly unitary, which is extremely unlikely under roundoff errors.

(c) If the exact SVD of $A+\delta A$ is $f(A+\delta A) = (U',\Sigma',V')$, then stability would mean that, in addition to $\|\delta A\| = \|A\|O(\varepsilon_{\text{machine}})$, we would also have to have $\|(U',\Sigma',V') - (\tilde{U},\tilde{\Sigma},\tilde{V})\| = \|(U,\Sigma,V)\|O(\varepsilon_{\text{machine}})$, for any norm $\|(U,\Sigma,V)\|$ on triples (U,Σ,V). For example, we could use $\|(U,\Sigma,V)\| = \max(\|U\|,\|\Sigma\|,\|V\|)$ for any matrix norm. e.g. for the L_2 induced norm, $\|U\| = \|V\| = 1$ and $\|\Sigma\| = \|A\|$, so we would have

$$\max(\|U' - \tilde{U}\|, \|\Sigma' - \tilde{\Sigma}\|, \|V - \tilde{V}\|) = \|A\|O(\varepsilon_{\text{machine}}),$$

or equivalently

$$\|U' - \tilde{U}\| = \|A\|O(\varepsilon_{\text{machine}}),$$
$$\|\Sigma' - \tilde{\Sigma}\| = \|A\|O(\varepsilon_{\text{machine}}),$$
$$\|V' - \tilde{V}\| = \|A\|O(\varepsilon_{\text{machine}}).$$

It is tempting to instead put $\|U\|O(\varepsilon_{\text{machine}})$, $\|\Sigma\|O(\varepsilon_{\text{machine}})$, and $\|V\|O(\varepsilon_{\text{machine}})$ on the right-hand sides, but this appears to be a much stronger condition (which may well be true in SVD algorithms, but is not what was given).

Problem 2: Least squares (20 points)

Suppose that we want to solve the **weighted least-squares** problem

$$\min_x \|B^{-1}(Ax-b)\|_2$$

where B ($m \times m$) is a nonsingular square matrix and A ($m \times n$) has full column rank. This can be solved a bit trivially because we can write it down as an ordinary least squares problem

$$\min_x \|A'x - b')\|_2$$

for $A' = B^{-1}A$ and $b' = B^{-1}b$.

(a) The normal equations are $A'^*A'x = A'^*b'$ from ordinary least-squares, i.e.

$$A^*\left(B^{-1}\right)^* B^{-1}Ax = A^*\left(B^{-1}\right)^* B^{-1}x.$$

(b) We can solve it exactly as for the ordinary least-squares problem in A' and b'. First, compute $A' = U^{-1}L^{-1}A$ by backsubstitution, where $B = LU$ e.g. by Gaussian elimination. Then QR factorize $A' = QR$ e.g. by Householder, then solve $R^*x = Q^*b' = Q^*U^{-1}L^{-1}b$. As in ordinary QR for least-squares, all of the squareing of A' has been cancelled analytically, and both the right and left-hand sides of this equation are multiplications of things with $\kappa(R) = \kappa(B^{-1}A)$ and $\kappa(B^{-1})$, respectively.

I should mention that there are even more efficient/accurate ways to solve this problem than doing ordinary least-squares with $B^{-1}A$. LAPACK provides something called a "generalized QR" factorization $A = QR$, $B = QTZ$ for this, where Z is also unitary and T is upper triangular, to avoid the necessity of a separate LU factorization of B.

Problem 3: Eigenvalues (20 points)

(a) Suppose our initial guess is expanded in the eigenvectors q_i as $x_1 = c_1 q_1 + c_2 q_2 + \cdots$ (where for a random x_1 all the c_i are $\neq 0$ in general), in which case $x_n = (c_1 \lambda_1^n q_1 + c_2 \lambda_2^n q_2 + \cdots)/\|\cdots\|$. If $|\lambda_1| = |\lambda_2|$, then the q_1 and q_2 terms will grow at the same rate, and it will never be dominated by q_1 alone—x_n will "bounce around" in a two-dimensional subspace spanned by q_1 and q_2. Note that the algorithm will *not* in general converge: for example, if $\lambda_2 = -\lambda_1$, then the relative signs of the q_1 and q_2 terms will oscillate. (More generally, $\lambda_2 = e^{i\phi}\lambda_1$ for some phase ϕ, and the q_2 term will rotate in phase by ϕ relative to q_1 on each step.)

However, if $\lambda_1 = \lambda_2$, then $x_n = [\lambda_1^n (c_1 q_1 + c_2 q_2) + O(|\lambda_3/\lambda_1|^n)]/\|\cdots\|$, and x_n therefore becomes parallel to $c_1 q_1 + c_2 q_2$ as $n \to \infty$ (assuming $|\lambda_3| < |\lambda_1|$). But both q_1 and q_2 are eigenvectors with the same eigenvalue λ_1, so $c_1 q_1 + c_2 q_2$ is also an eigenvector of λ_1. Therefore, this is not a problem: we still get an eigenvector of λ_1. Equivalently, all of the eigenvectors for $\lambda = \lambda_1$ form a subspace which is magnified relative to other eigenvalues by multiplying repeatedly by A, as long as other eigenvalues $\neq \lambda_1$ have smaller magnitude.

(b) Applying Lanczos to $(A - \mu I)^2$ is computationally easy because we just need to multiply by $(A - \mu I)$ twice in each step (cheap if A is sparse, cost $\sim$ #nonzeros). However, we have squared the condition number of $A - \mu I$, and thus we square the rate at which the *largest*-$|\lambda|$ eigenvalues appear in the Krylov space, correspondingly slowing the rate (~ doubling the number of iterations) at which we get the *smallest* eigenvalue (the one closest to μ), and exacerbating problems with roundoff errors and ghost eigenvalues.

On the other hand, $(A - \mu I)^{-1}$ does not square any condition numbers, since the desired eigenvalue is now the largest-magnitude one, there should be no problem with ghost eigenvalues or roundoff (from homework) and Lanczos should converge well. However, we now have to multiply by $(A - \mu I)^{-1}$ at each step, which means that on each step we need to solve a linear system with $A - \mu I$. If A is amenable to sparse-direct factorization, then we can do this *once* and re-use it throughout the Lanczos process (only somewhat slower than multiplying by A repeatedly, since the sparse-direct factorization generally loses some sparsity). If sparse-direct solvers are impractical and we have to use an iterative solver like GMRES etcetera, then we have the expense of repeating this iterative process on each step of Lanczos.

Because of these tradeoffs, neither method is ideal for computing interior the eigenvalue closest to μ—computing interior eigenvalues is tricky.

18.335J Introduction to Numerical Methods

Spring 2019