

Figure 1: Fractional error in minimum eigenvalue λ as a function of iterations (matrix-vector multiplies) for Lanczos and for restarted Lanczos (restarting with the best Ritz vector every 10 iterations).

18.335 Problem Set 5 Solutions

Problem 1 (10+10+10 pts):

The results for parts (a) and (b) are shown in figure 1. Both Lanczos and restarted Lanczos converge to more than six significant digits to the minimal λ in under 50 iterations, but the restarted version has much more erratic convergence.

My Matlab code (very inefficient, but does the job) is given below. Note that, in order to restart, we must keep track of the Q_n matrix (save the q vectors) in order to compute the Ritz vector $Q_n y$ from an eigenvector y of T_n .

```
% Usage: [T, lamsmallest] = lanczos(A, b, nmax)
%
% Returns nmax-by-nmax tridiagonal matrix T after nmax steps of Lanczos,
% starting with a vector b (e.g. a random vector). T is
% unitarily similar to A (in theory).
%
% If nrestart is passed, repeat the process nrestart times, each time
% starting over with the Ritz vector for the minimim eigenvalue.
%
% Also returns an array lamsmallest of the smallest Ritz values
% for each iteration.
%
% (This routine is not for serious use! Very inefficient, simplistic
% restarting, no re-orthogonalization, etc.)
function [T,lamsmallest] = lanczos(A, b, nmax, nrestart)
    if (nargin < 4)
        nrestart = 1;
```

```

end
lamsmallest = [];
m = size(A,1);
for irestart = 1:nrestart
    beta(1) = 0;
    qprev = zeros(m, 1);
    q = b / norm(b);
    Q = q;
    for n = 1:nmax
        v = A*q;
        alpha(n) = q' * v;
        v = v - beta(n) * qprev - alpha(n) * q;
        beta(n+1) = norm(v);
        qprev = q;
        q = v / beta(n+1);
        Q = [Q,q];

        beta1 = beta(2:end-1);
        T = diag(alpha) + diag(beta1,1) + diag(beta1,-1);
        lamsmallest = [lamsmallest, min(eig(T))];
    end
    [V,D] = eig(T);
    lams = diag(D);
    ilam = min(find(lams == min(lams)));
    b = Q(:,1:nmax) * V(:,ilam);
end

```

In part (c), I asked you to repeat the same process, except selecting the $\min|\lambda|$ eigenvalue at each step. For this particular A , the smallest two eigenvalues are -0.3010 and $+0.3828$, so the minimum- $|\lambda|$ eigenvalue is the same as the minimum- λ eigenvalue, so at first glance you might expect the same results as before. However, the algorithm doesn't "know" this fact, and Lanczos is much better at getting extremal (min/max) eigenvalues correct than eigenvalues in the interior of the spectrum. The results are shown in figure 2. The nonrestarted case appears to converge the same as before at first, but periodically a spurious/ghost eigenvalue appears near 0 that messes up convergence until it goes away. The restarted case is even worse. (The situation would be even worse if we tried to get an eigenvalue even further in the interior of the spectrum, e.g. the eigenvalue closest to 2.)

Problem 2 (25 pts):

The residual curves are plotted in figure 3. Several things are apparent. First, as expected, CG (conjugate gradient) converges much faster than SD (steepest descent). Since the condition number $\kappa(A) \approx 397$ for this A , we expect SD to take several hundred iterations to converge, while CG should take only several dozen. Second the computed residual for CG matches the actual residual until we hit machine precision ($\approx 10^{-16}$), at which point rounding errors prevent the actual residual from decreasing further (while the computed residual keeps going down). Third, the estimate $2(\sqrt{\kappa} - 1)^n(\sqrt{\kappa} + 1)^n$ from theorem 38.5 is grossly pessimistic.

What is happening, as discussed in class, is that the CG process is effectively shrinking the size of the vector space as it goes along, because once it searches a direction it never needs to search that direction again, and the condition number of A restricted to the remaining subspace gets better and better—we get *superlinear convergence*, observed in the fact that the residual on decreases faster than exponentially (which would be a straight line on a semilog scale).

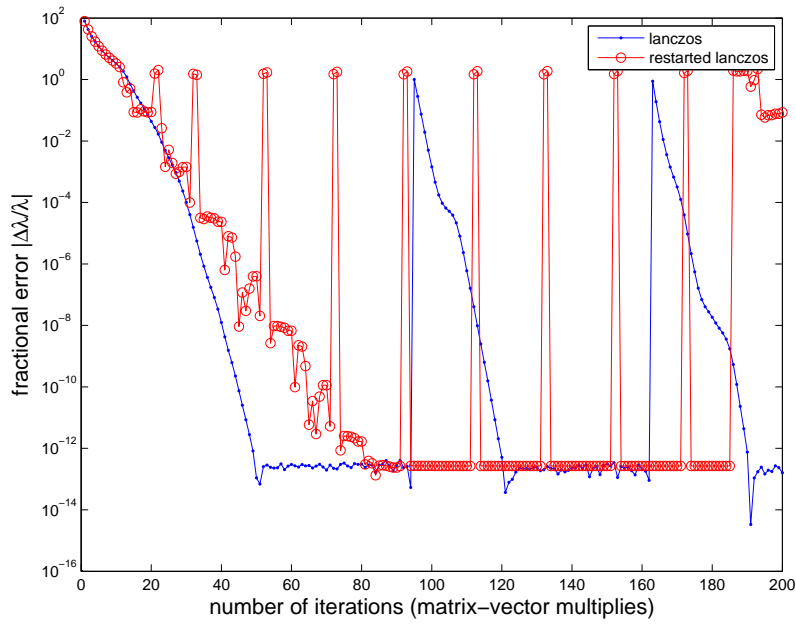


Figure 2: Fractional error in minimum- $|\lambda|$ eigenvalue λ as a function of iterations (matrix-vector multiplies) for Lanczos and for restarted Lanczos (restarting with the best Ritz vector every 10 iterations). The convergence is irregular because Lanczos has trouble getting the interior of the spectrum accurately.

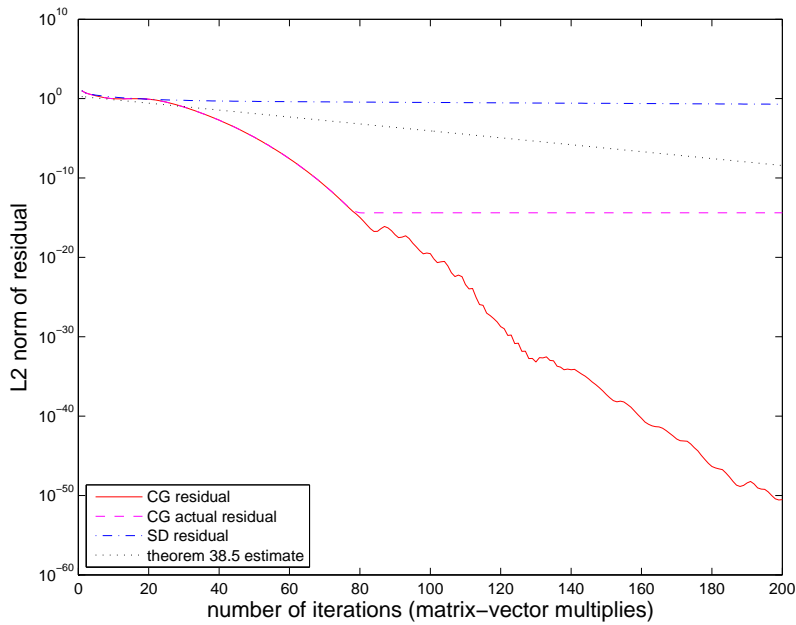


Figure 3: Convergence of the residual for problem 38.6, using conjugate-gradient (CG) or steepest-descent (SD), and also the actual CG residual $\|Ax - b\|$ as well as the estimated upper bound on the residual from theorem 38.5.

We can quantify this in various ways. For example, since on each step the residual is “supposed” to decrease by a factor $f = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$, we can define an “effective” condition number $\tilde{\kappa}_n$ from the actual rate of decrease $f_n = (\sqrt{\tilde{\kappa}_n} - 1)/(\sqrt{\tilde{\kappa}_n} + 1) = \|r_{n+1}\|/\|r_n\|$. Thus, $\sqrt{\tilde{\kappa}_n} = (1 + f_n)/(1 - f_n)$. By the time 30 iterations have passed, $\tilde{\kappa}_{30}$ is only 38; by 40 iterations, $\tilde{\kappa}_{40} \approx 21$; by 60 iterations, $\tilde{\kappa}_{60} \approx 8.8$. If we look at the set of eigenvalues, we see that eliminating just the two smallest λ would give a condition number of 34, and eliminating the 11 smallest eigenvalues would give a condition number of 8.4. (Eliminating the largest eigenvalues does not reduce the condition number nearly as quickly.) So, what seems to be happening is that CG is effectively eliminating the smallest eigenvalues from the spectrum as the algorithm progresses, improving the condition number of A and hence the convergence rate is better than the pessimistic bound of theorem 38.5.

The code for SD was posted on the web site; my code for CG in this problem is listed below (modified to add an additional stopping criteria for problem 3).

```
% Usage: [x, residnorm, residnorm2] = CG(A, b, x, nmax, tol)
%
% Performs nmax steps of conjugate gradient to solve Ax = b for x,
% given a starting guess x (e.g. a random vector). A should be
% Hermitian positive-definite. Returns the improved solution x.
%
% Stops whenever nmax steps are performed.
%
% residnorm is an array of length nmax of the residuals |r| as
% computed during the SD iterations. residnorm2 is the same thing,
% but using |b - A*x| instead of via the updated r vector.
function [x, residnorm, residnorm2] = CG(A, b, x, nmax)
    d = zeros(size(x));
    Ad = d;
    r = b - A*x;
    rr = r' * r;
    alpha = 0;
    initresid = norm(r);
    for n = 1:nmax
        r = r - alpha * Ad;
        residnorm(n) = norm(r);
        residnorm2(n) = norm(b - A*x);
        rrnew = r' * r;
        d = r + d * (rrnew / rr);
        rr = rrnew;
        Ad = A*d;
        alpha = rr / (d' * Ad);
        x = x + alpha * d;
    end
```

Problem 3 (15 points)

The results for this problem are shown in problem xxx. They are a bit odd, in that the conjugate-gradient method indeed converges to a null-space vector, but when the residual $\|Ax\|$ gets very small ($\approx 10^{-10}$), the iteration goes crazy and the residual norm shoots up again. This seems to be due to some sort of roundoff disaster. In practice, one could simply stop the iterations once the residual became sufficiently small. There may also be ways of mitigating the roundoff problems; for example, in blue are shown the results if CG is simply restarted once the residual becomes small, in which case the residual continues to decrease for

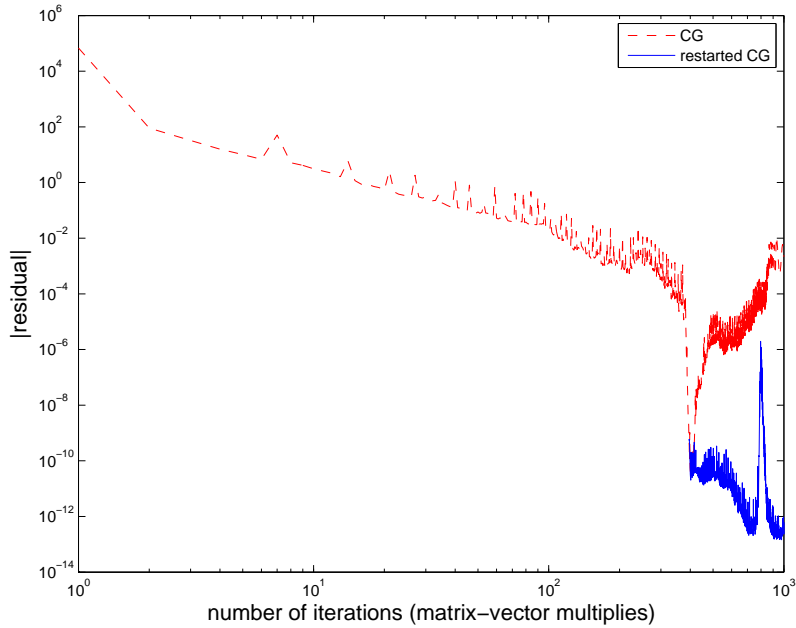


Figure 4: Convergence of conjugate gradient to a null-space vector of a positive *semi*-definite matrix A . The convergence breaks down once the residual $\|Ax\|$ becomes very small, probably because roundoff errors effectively give A a small negative eigenvalue. Simply stopping CG when the residual becomes small and restarting from there (blue line) improves matters somewhat, allowing the residual to be decreased by another couple of orders of magnitude before a breakdown resurfaces.

another couple orders of magnitude before errors resurface, and there are probably more clever corrections as well.

What is the source of the roundoff disaster? I haven't tried to diagnose it in detail, but a simple check of the eigenvalues of A reveals a clue. Although A should, in principle, be positive semidefinite, in practice because of roundoff errors I find that it could have at least one slightly negative eigenvalue. Once roundoff errors have made the problem indefinite, the well-known possibility of breakdown of the conjugate-gradient algorithm resurfaces.¹

¹See, e.g., Paige, Parlett, and Van der Vorst, "Approximate solutions and eigenvalue bounds from Krylov subspaces," *Numer. Lin. Alg. Appls.* **2**(2), 115–133 (1995), or Fischer, Hanke, and Hochbruck, "A note on conjugate-gradient type methods for indefinite and/or inconsistent linear systems," *Num. Algorithms* **11**, 181–187 (1996).