CS 6220 A5
Marcus Lim (mkl65)

P2. Many rectangular matrices from the UF collection of sparse matrices were singular, so $\sigma_{\min}$ would actually be 0. However, to make the problem interesting, I looked for the smallest singular value above 10^{-8}.

Of note, methods 1 and 3 are very similar, as Matlab actually uses method 3 when given a problem for method 1.

I've realized that calling svds(A, 1, 0, options) does not work, as Matlab actually forms matrix C defined in method 3, and calls eigs on C, but

$$rank(C) = 2n < m + n$$

Hence, C is singular, so 0 is an eigenvalue, and eigs fails. Hence, we need some initial guess other than 0.

Notice that:

$$\mathrm{K}(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$$

So for matrices that are not too ill-conditioned, the ratio $\sigma_{\max}/\sigma_{\min}$ should not be too large, and $\sigma_{\max}$ is easy to find. Hence, I made my initial guess $\sigma_{\max}$ and repeatedly try eigs/svds with my guess divided by 10, since we want an eigenvalue correct to a factor of 10. This is repeated until eigs/svds finds an value that is less than 10^{-8} (my tolerance), so I treat the value before this as $\hat{\sigma}_{\min}$. This works for both methods 1 and 3.

Lastly, I've also noticed that method 2 is prone to numerical error, so I came up with method 4 that uses the estimate from method 2 as an initial guess for method 3. (For details, see code attached.)

For comparison, since the matrices were not too big, I calculated all singular values using dense SVD to get the actual $\sigma_{\min}$.

The table below shows the UFid and some statistics of the sparse matrix. Values t1 to t4 are the running times for each method, and E1 to E4 are the ratios of the $\sigma_{\min}$ found to the actual $\sigma_{\min}$.

id	**m**	**n**	**nnz**	**t1**	**t2**	**t3**	**t4**	**E1**	**E2**	**E3**	**E4**	**Actual $\sigma_{\min}$**
2066	1485	66	2970	0.08	0.39	0.06	0.45	1.11	0.99	1.06	1.00	6.63
2030	1176	56	2352	0.03	0.20	0.04	0.24	1.09	0.99	1.04	1.00	6.40
1984	1019	60	1513	0.04	0.07	0.03	0.09	1.00	1.00	1.00	1.00	2.17
2061	990	55	1980	0.02	0.16	0.03	0.18	1.12	0.99	1.07	1.00	5.92
12	958	292	1916	0.02	0.07	0.06	0.13	3.20	1.02	2.97	1.01	1.32
			Average:	0.04	0.18	0.04	0.22	1.50	1.00	1.43	1.00	

Note that the average for the ratios is actually the geometric average.

CS 6220 A5
Marcus Lim (mkl65)

We can see that all methods give $\sigma_{\min}$ to within a factor of 10, and the performance of method 1 and method 3 are very similar. Method 2 gives a better estimate of $\sigma_{\min}$, and the extra step in method 4 improves the estimate.

Code:

```
function ShowSigmaMinMethods()
s = warning('off');
ufidx = [2066, 2030, 1984, 2061, 12];

minEig = 1e-8;

fprintf('id\tm\tn\tnnz\tt1\tt2\tt3\tt4\t');
fprintf('E1\tE2\tE3\tE4\tActual\n');

options = struct('issym',1,'disp',0,'tol',0.1,'p',2);
for k=1:length(ufidx)
    B = UFget(ufidx(k));
    A = sparse(B.A);
    [m,n] = size(A);
    fprintf('%d\t%d\t%d\t%d\t',ufidx(k),m,n,nnz(A));

    t1 = tic;
    s1 = svds(A, 1, 'L', options);
    guess = s1/10;
    while ~isempty(s1)
        last = s1;
        s1 = svds(A, 1, guess, options);
        s1 = s1(s1>minEig);
        guess = guess/10;
    end
    s1 = last;
    t1 = toc(t1);

    t2 = tic;
    s2 = sqrt(eigs(@(x) A\(x'/A)', n, 1, 'sm', options));
    t2 = toc(t2);

    t3 = tic;
    C = sparse( m+n, m+n );
    C( 1:m, m+1:m+n ) = A;
    C( m+1:m+n, 1:m ) = A';
    s3 = eigs(C, 1, 'LA', options);
    guess = s3/10;
    while ~isempty(s3)
        last = s3;
        s3 = eigs(C, 1, guess, options);
        s3 = s3(s3>minEig);
        guess = guess/10;
    end
    s3 = last;
    t3 = toc(t3);

    t4 = tic;
    C = sparse( m+n, m+n );
```

```
    C( 1:m, m+1:m+n ) = A;
    C( m+1:m+n, 1:m ) = A';
    guess = s2;
    s4 = eigs(C, 1, guess, options);
    s4 = s4(s4>minEig);
    t4 = toc(t4);

    sigma = (svd(full(A)));
    sigma = min(sigma(sigma>minEig));
    E1 = s1 / sigma;
    E2 = s2 / sigma;
    E3 = s3 / sigma;
    E4 = s4 / sigma;
    fprintf('%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\n',...
        t1,t2,t3,t2+t4,...
        E1,E2,E3,E4,...
        sigma);
end
warning(s);
end
```