Numerical Methods I, Fall 2010
Assignment III: Eigen and Singular Values

Aleksandar Donev
Courant Institute, NYU, *donev@courant.nyu.edu*

October 7th, 2010
Due: October 21st, 2010

Choose the problems that interest you, including any of the extra credit ones. Anything above 60 points is very good (A), above 70 is excellent (A+). "Extra credit" simply marks problems that are more free-style and do not come with very specific directions.

1 [Pen-and-pencil 20 points] Ill-Conditioning of the Inverse Power Method

[Due to Trefethen and Bau]

The inverse power method solves the linear system

$$(\boldsymbol{A} - \lambda \boldsymbol{I})\,\boldsymbol{y} = \boldsymbol{x}_{k-1}$$

at every step. As λ becomes closer to a true eigenvalue, this system becomes closer to singular. Show that this is not a problem anyway, that is, show that if the linear system above is solved using a stable method, even though the computed solution $\hat{\boldsymbol{y}}$ may be quite far from the true $\boldsymbol{y}$ due to ill-conditioning, the normalized $\hat{\boldsymbol{y}}/\left\|\hat{\boldsymbol{y}}\right\|$ will point in the right direction, that it is, it will not be far from the correct direction $\boldsymbol{y}/\left\|\boldsymbol{y}\right\|$ [*Hint: It is OK to make simplifying assumptions, such as that all the eigenvalues are distinct or that $\boldsymbol{x}_{k-1}$ is "generic"*].

2 [Up to 85 points] Ill-Conditioned Matrices

2.1 [25 pts] Solving Linear Systems via SVD: The Hilbert Matrix

Recall the example of a very ill-conditioned symmetric positive-definite matrix from Homework 2, the Hilbert matrix, whose entries are:

$$a_{ij} = \frac{1}{i+j-1}.$$

Let the size of the matrix be $n = 15$. Compute the right-hand side (rhs) $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$ so that the exact solution is $\boldsymbol{x} = 1$ (all unit entries). Solve the linear system in MATLAB and report the relative error in the approximate solution $\hat{\boldsymbol{x}}$ (for example, in the Euclidian norm, $\delta x = \left\|\boldsymbol{x} - \hat{\boldsymbol{x}}\right\|_2$).

[5 pts] Now compute the SVD decomposition of $\boldsymbol{A}$. Look at the singular values of $\boldsymbol{A}$ and comment on whether you can see how ill-conditioned this matrix is based on this [*Hint: The MATLAB function diag can be used to extract the diagonal of a matrix or to construct a diagonal matrix*]. Construct the matrix pseudo-inverse $\boldsymbol{A}^{\dagger}$ from the SVD, and from it a solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^{\dagger}\boldsymbol{b}$, and see if this is any more accurate than the previous direct solution. [*Hint: To check your answers you can use the MATLAB function pinv and compare to your answer for small n*].

[15pts] For a given relative tolerance ε, a modified pseudo-inverse $\hat{\boldsymbol{A}}^{\dagger}$ is obtained by first setting to zero all singular values that are smaller than $\epsilon\sigma_1$, where σ_1 is the largest singular value. For several logarithmically-spaced tolerances (for example, $\varepsilon = 10^{-i}$ for $i = 1, 2, \ldots, 16$), compute the modified pseudo-inverse and then a solution $\hat{\boldsymbol{x}} = \hat{\boldsymbol{A}}^{\dagger}\boldsymbol{b}$. Plot the relative error in the modified solution versus the tolerance on a log-log scale. You should see a clear minimum error for some $\varepsilon = \tilde{\varepsilon}$. Report this optimal $\tilde{\varepsilon}$ and the smallest error, and verify your solution by using the built-in function

$$\tilde{x} = pinv(A, \tilde{\varepsilon}\sigma_1) \star b.$$

Note that the SVD itself is stable against perturbations so *pinv* does not suffer from ill-conditioning.

[5pts] Explain what kinds of errors are traded off, that is, what kind of errors dominate for large versus for small tolerances [*Hint: You should recall similar plots of errors versus parameter from the first homework*].

2.2 [25 pts] Defective Matrices

The MATLAB code:

```
diagonals=ones(n,2); diagonals(:,1)=2;
A = full(spdiags(diagonals,[0,1],n,n));
```

constructs an $n \times n$ matrix that has twos on the diagonal and ones along the upper diagonal (these types of matrices appear in finite-difference algorithms). The characteristic equation for this matrix is $(\lambda - 2)^n = 0$ so that the only eigenvalue is $\lambda = 2$ and it has algebraic multiplicity n, but there is only one eigenvector, i.e., the geometric multiplicity is 1. This matrix is thus (very) defective ("defectiveness" is $d = n - 1$). In the lectures I claimed that this leads to severe ill-conditioning of the eigenvalue problem. Interestingly, because of the special structure, MATLAB's *eig* function works well for this problem (verify this), but see problem 2.3 for a problematic case.

Theory suggests that perturbing the matrix to $\boldsymbol{A} + \varepsilon\,(\delta\boldsymbol{A})$, where ε is a small perturbation parameter and $\delta\boldsymbol{A}$ is some random perturbation (for example, generated using MATLAB's *randn* function), perturbs the eigenvalues by

$$\delta\lambda \sim \left\|\varepsilon\delta\boldsymbol{A}\right\|^{1/(1+d)} \sim \varepsilon^{1/(1+d)}, \tag{1}$$

where d is the defectiveness of the eigenvalue, in this case, $d = n - 1$. Verify this scaling claim by doing numerical perturbations of the matrix and computing the new eigenvalues using MATLAB's *eig* for $n = 4, 8$ and 16. Choose the perturbation scales ε logarithmically-spaced (for example, $\varepsilon = 10^{-i}$ for $i = 1, 2, \ldots, 16$), and plot the magnitude of the resulting perturbation in the eigenvalues (for example, you can use $norm(eig(A) - 2)$ in MATLAB) versus ε on a log-log scale. From the plot verify the theoretical prediction given by Eq. (1) [*Hint: A power-law scaling shows up in a log-log plot as a straight line, and the exponent is determined by the slope. So showing the theoretical estimate $\varepsilon^{1/n}$ on the same plot will do the trick*].

[5pts] Give some theoretical justification for Eq. (1), even if not a complete theorem.

2.3 [20 pts + 10 pts extra credit] Rayleigh Quotient Iteration

In MATLAB $A = gallery(5)$ produces an "interesting" 5×5 matrix that satisfies the equation $\boldsymbol{A}^5 = \boldsymbol{0}$ (try it!), which means that the only eigenvalue is the solution to $\lambda^5 = 0$, i.e., $\lambda = 0$ is a 5-fold degenerate eigenvalue.

[5 pts] Use MATLAB's *svd* to determine the rank of this matrix by computing its SVD, and from the rank calculate how many linearly independent eigenvectors correspond to the eigenvalue of algebraic multiplicity 5 (i.e., determine the geometric multiplicity) [*Hint: This is a defective matrix*]. Now try $eig(A)$ and see if the results are correct and how far they are from being correct.

Now let's see if an iterative method can help, as it has for some other problems in the other homeworks. The Rayleigh quotient iteration is a very fast method to refine an initial normalized guess for an eigenvector $\boldsymbol{x}_0$ and to improve the initial eigenvalue estimate $\lambda_0 = \boldsymbol{x}_0^\star \boldsymbol{A}\boldsymbol{x}_0$. It is basically an inverse power method but it updates the eigenvalue estimate at every step, $k = 1, 2, \ldots$:

1. Solve $(\boldsymbol{A} - \lambda_{k-1}\boldsymbol{I})\,\boldsymbol{y} = \boldsymbol{x}_{k-1}$.
2. Normalize the solution, $\boldsymbol{x}_k = \boldsymbol{y}/\left\|\boldsymbol{y}\right\|$.
3. Update the eigenvalue estimate, $\lambda_k = \boldsymbol{x}_k^\star \boldsymbol{A}\boldsymbol{x}_k$.

The iteration converges very rapidly, specifically, cubically. You can read more about it on Wikipedia, for example.

[15 pts] Implement the Rayleigh quotient iteration in MATLAB (do not use the Wikipedia code!) and test it on some "harmless" matrix, for example, $A = randn(5)$. That is, verify that for any initial guess the algorithm converges to an eigenvector/eigenvalue pair, in some sense close to the initial guess. Then try it on the weird matrix $gallery(5)$ and see if it converges and obtains a much better estimate of the eigenvalue $\lambda = 0$ than MATLAB's built-in function *eig*.

[Extra credit 15pts] Show that, under suitable assumptions, the Rayleigh iteration converges cubically.

3 [Up to 40 points] PCA: Digraph Matrix of English

[Due to Cleve Moler]

There are many interesting applications of principal component analysis (which is nothing more than the singular-value decomposition) in varied disciplines. This one focuses on a simple but hopefully interesting example of analysing some of the structure of written language (spelling).

For this assignment use English, which has 26 letters, indexed from $1 - 26$ in some way. The ASCII encoding of characters is one way to index the letters ('A' has ASCII code 65), but for this assignment you may find that putting the vowels (AEIOUY) first will be better. A digraph frequency matrix is a 26×26 matrix where each entry a_{ij} counts how many times the letter i-th letter follows the j-th letter in some piece of text. All blanks and non-letter characters are removed, and the text is capitalized so there is no difference between 'A' and 'a', and often it is assumed that the first letter follows the last letter. A digraph matrix can be constructed in MATLAB from a piece of text saved in a file *'text.txt'* by first converting the text to a vector of integers k where each element is in the range $1 - 26$:

```
% Read the file:
file='text.txt'; % Sample text
fid = fopen(file); txt = fread(fid); fclose(fid);

% Convert to integers between 1 and 26:
numtxt = upper(char(txt)) - 'A' + 1;
% Eliminate any weird characters and spaces:
k=numtxt((numtxt >= 1) & (numtxt <= 26));
```

[5pts] Find some (large) piece of English text (cut and paste from Wikipedia, for example) and convert it to a in integer vector k, and then from that construct the digraph matrix $\boldsymbol{A}$ for that text. Normalize the matrix so that the largest element in the matrix is 1.0. Visualize (plot) the matrix, for example, using the MATLAB function *pcolor* (alse note that *colorbar* may be useful).

[10pts] Now compute the SVD (PCA),

$$\boldsymbol{A} = \sum_{i=1}^{26} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\star,$$

and visualize (plot) the first (principal) rank-1 component $\boldsymbol{A}_1 = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^\star$, the second principal component $\boldsymbol{A}_2 = \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^\star$, and the rank-2 approximation to the matrix, $\boldsymbol{A}_1 + \boldsymbol{A}_2$.

[5pts] Can you see some features of the English language that the first principal component captures? [*Hint: Doing sum(A) computes the frequency of occurence of the different letters (make sure you understand why!)*]

[5pts] Can you see some features of the English language that the second principal component captures? *Hint: The following permutation will reorder the letters so that the vowels come first:*

```
p=[1,5,9,15,21,25,2:4,6:8,10:14,16:20,22:24,26];
char(p+'A'-1) % The re-ordered 'alphabet'
```

[5pts] Are there any obvious features that the rank-2 approximation misses?

[10 pts extra credit] Do some testing to make sure that what you are observing is a feature of the language and not the particular text you chose. For example, try a different, a longer text, and maybe even something in another language. How long does the text need to be to see the features?