Scientific Computing, Fall 2020
Assignment III: Eigen and Singular Values

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

October 1st, 2020
Due by **Sunday October 18th**, 2020

Remember to give an explanation of your results in your report, in addition to reporting the results. For the purposes of grading the maximum number of points is 70.

1 [30 pts] Solving Ill-Conditioned Linear Systems via SVD

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, $\delta x = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2$).

1.1 [10pts] Solving systems using the SVD

[10 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*].

1.2 [20pts] Regularized Pseudo-Inverse

For a given relative tolerance $\varepsilon \ll 1$, a modified or regularized pseudo-inverse $\hat{\boldsymbol{A}}^\dagger$ is obtained by first setting to zero all singular values that are smaller than $\varepsilon\sigma_1$, where σ_1 is the largest singular value. This can be obtained in MATLAB using the built-in function *pinv* as $\hat{\boldsymbol{A}}^\dagger = pinv(A, \varepsilon\sigma_1)$ but it is strongly recommended that you write your own function and only use the built-in one to check your code.

[15pts] 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 that you can get.

[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. Think what happens if ε is large versus when it approaches zero.*]

2 [45 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 analyzing 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 i-th letter follows the j-th letter in some piece of text (you can also swap the order to count). 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));
```

[10pts] 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 *imagesc* (alse note that *colorbar* may be useful, but note that if you use *pcolor* you may not get what you want – read the documentation first!).

[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, $\boldsymbol{A}_1 + \boldsymbol{A}_2$.

[7.5pts] Can you see some features of the English language that the first principal component captures? [*Hint: Doing sum(A) computes the frequency of occurrence of the different letters (make sure you understand why!)*]

[7.5pts] 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 (try it!):*

```
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'
```

[2.5pts] Are there any obvious features that the rank-2 approximation misses?

[7.5pts] 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?