Scientific Computing, Spring 2012
Assignment III: SVD, Nonlinear Equations and Optimization

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

Posted March 1st, 2012
Due by Sunday **March 25th**, 2012

A total of 100 points is possible (100 points = A+), plus 30 extra credit points (make up for previous homeworks).

1 [35 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));
```

1.1 [10pts] English

[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* (also note that *colorbar* may be useful, and the hint from MATLAB's help page that *pcolor* "colors each cell with a single color. The last row and column of the matrix are not used in this case").

1.2 [10pts] PCA

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

1.3 [10pts] What can we learn from the PCA?

[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'
```

1.4 [5pts] Problems

[2.5pts] Are there any obvious features of the English language that the rank-2 approximation misses or gets wrong?

[2.5 pts] 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 longer text. How long does the text need to be to see the features clearly?

2 [35 points] Newton-Raphson Method in One Dimension

Consider finding the three roots of the polynomial

$$f(x) = 816x^3 - 3835x^2 + 6000x - 3125,$$

which happen to all be real and all contained in the interval $[1.4, 1.7]$ [due to Cleve Moler].

2.1 [5pts] The roots

Plot this function on the interval $[1.4, 1.7]$ and find all of the zeros of this polynomial using the MATLAB function *fzero* [*Hint: The roots values can be obtained in MATLAB using the built-in function roots but Maple tells us the roots are* $25/16$, $25/17$ *and* $5/3$].

2.2 [15 pts] Newton's Method

[7.5pts] Implement Newton's method (no safeguards necessary) and test it with some initial guess in the interval $[1.4, 1.7]$.

[7.5pts] Verify that the order of convergence is quadratic, as predicted by the theory from class:

$$\frac{\left|e^{k+1}\right|}{\left|e^k\right|^2} \to \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|.$$

[*Hint: Due to roundoff errors and the very fast convergence, the error quickly becomes comparable to roundoff, so one must be careful not to use very large* k]

2.3 [15pts] Robustness

[7.5pts] Starting from many (say 100) guesses in the interval $[1.4, 1.7]$, run 100 iterations of Newton's method and see plot the value to which it converges, if it does, as a function of the initial guess. If the initial guess is sufficiently close to one of the roots α, i.e., if it is within the *basin of attraction* for root α, it should converge to α. What is the basin of attraction for the middle root ($\alpha = 25/16$) based on the plot?

[5pts] The theory from the lecture suggested an estimate for the width of each basin of attraction around a given root α of the form:

$$\left|x^0 - \alpha\right| \leq \left|\frac{f''(\alpha)}{2f'(\alpha)}\right|^{-1}.$$

Compare these estimates to what is actually observed numerically.

[2.5pts] Is the estimate particularly bad for one of the roots, and if so, can you think of reasons why?

3 [30 pts + 30 extra credit] Nonlinear Least-Squares Fitting

In homework 2 you considered fitting a data series (x_i, y_i), $i = 1, \ldots, m$, with a function that that depends linearly on a set of unknown fitting parameters $\boldsymbol{c} \in \mathbb{R}^n$. Consider now fitting data to a nonlinear function of the fitting parameters, $y = f(x; \boldsymbol{c})$. The least-squares fit is the one that minimizes the squared sums of errors,

$$\boldsymbol{c}^\star = \arg\min_{\boldsymbol{c}} \sum_{i=1}^{m} \left[f(x_i; \boldsymbol{c}) - y_i \right]^2 = \arg\min_{\boldsymbol{c}} \left(\boldsymbol{f} - \boldsymbol{y} \right)^T \left(\boldsymbol{f} - \boldsymbol{y} \right), \tag{1}$$

where $\boldsymbol{f}(\boldsymbol{c}) = f(\boldsymbol{x}; \boldsymbol{c})$ is a vector of m function values, evaluated at the data points, for a given set of the parameters $\boldsymbol{c}$.

We will consider here fitting an exponentially-damped sinusoidal curve with four unknown parameters (amplitude, decay, period, and phase, respectively),

$$f(x; \boldsymbol{c}) = c_1 e^{-c_2 x} \sin\left(c_3 x + c_4\right), \tag{2}$$

to a synthetic data set.

3.1 [5pts] Synthetic Data

Generate synthetic data by generating $m = 100$ points randomly and uniformly distributed in the interval $0 \leq x \leq 10$ by using the *rand* function. Compute the actual function

$$f(x; \boldsymbol{c}) = e^{-x/2} \sin\left(2x\right), \tag{3}$$

and then add perturbations with absolute value on the order of 10^{-2} to the y values (use the *rand* or the *randn* function). Compare the synthetic data to the actual function on the same plot to see how closely the data set follows the relation (3).

3.2 [25 pts] Gauss-Newton Method

The basic idea behind the Gauss-Newton method is to make a linearization of the function $f(x_i; \boldsymbol{c})$ around the current estimate $\boldsymbol{c}_k$,

$$\boldsymbol{f}(\boldsymbol{c}) \approx \boldsymbol{f}(\boldsymbol{c}_k) + \left[\boldsymbol{J}\left(\boldsymbol{c}_k\right)\right] \left(\boldsymbol{c} - \boldsymbol{c}_k\right) = \boldsymbol{f}(\boldsymbol{c}_k) + \left[\boldsymbol{J}\left(\boldsymbol{c}_k\right)\right] \Delta \boldsymbol{c}_k,$$

where the Jacobian $m \times n$ matrix is the matrix of partial derivatives $\partial f / \partial c$ evaluated at the data points:

$$\boldsymbol{J}\left(\boldsymbol{c}\right) = \boldsymbol{\nabla}_{\boldsymbol{c}} \boldsymbol{f}(\boldsymbol{c}).$$

This approximation (linearization) transforms the non-linear problem (1) into a linear least-squares problem, i.e., an overdetermined linear system

$$\left[\boldsymbol{J}\left(\boldsymbol{c}_k\right)\right] \Delta \boldsymbol{c}_k = \boldsymbol{J}_k \, \Delta \boldsymbol{c}_k = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{c}_k), \tag{4}$$

which you know how to solve from previous homeworks and lectures. The standard approach is to use the normal equations, which does not lead to substantial loss of accuracy if one assumes that the original problem is well-conditioned. Gauss-Newton's algorithm is a simple iterative algorithm of the form

$$\boldsymbol{c}_{k+1} = \boldsymbol{c}_k + \Delta \boldsymbol{c}_k,$$

starting from some initial guess $\boldsymbol{c}_0$. The iteration is terminated, for example, when the increment $\|\Delta \boldsymbol{c}_k\|$ becomes too small.

[15 pts] Implement Gauss-Newton's algorithm and see whether it works for the problem at hand, using an initial guess $\boldsymbol{c}_0$ that is close to the correct values.

[5pts] If you start with $\boldsymbol{c}_0 = (1, 1, 1, 1)$, does the method converge to the correct answer? Play around a bit with initial guesses and see if the method converges most of the time, and whether it converges to the "correct" solution or other solutions.

[5pts] Is this method the same or even similar to using Newton's method (for optimization) to solve the non-linear problem (1).

3.3 [30pts Extra Credit] Levenberg-Marquardt Algorithm

The Gauss-Newton algorithm is not very robust. It is not guaranteed to have even local convergence. A method with much improved robustness can be obtained by using a modified (regularized) version of the normal equations for the overdetermined system (4),

$$\left[\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right) + \lambda_k \,\mathrm{Diag}\,\left(\boldsymbol{J}_k^T \boldsymbol{J}_k\right)\right] \Delta \boldsymbol{c}_k = \boldsymbol{J}_k^T \left(\boldsymbol{y} - \boldsymbol{f}\right), \tag{5}$$

where $\lambda_k > 0$ is a damping parameter that is used to ensure that $\Delta \boldsymbol{c}_k$ is a descent direction, in the spirit of quasi-Newton algorithms for optimization. Here $\mathrm{Diag}(\boldsymbol{A})$ denotes a diagonal matrix whose diagonal is the same as the diagonal of $\boldsymbol{A}$.

If λ_k is large, the method will converge slowly but surely, while a small λ_k makes the method close to the Gauss-Newton Method, which converges rapidly if it converges at all. So the idea is to use a larger λ_k when far from the solution, and then decrease λ_k as approaching the solution. The actual procedure used to adjust the damping parameter is somewhat of an art, and here we study one simple but effective strategy.

[20 pts] Implement a code that repeats the following cycle for $k = 1, 2, \ldots$ until the increment $\|\Delta \boldsymbol{c}_k\|$ becomes too small, starting with an initial value $\lambda = 1$ and some guess $\boldsymbol{c}_0$:

1. Evaluate the function for the present estimate of the parameters, $\boldsymbol{f}_k = f(\boldsymbol{x}; \boldsymbol{c}_k)$, and then compute the residual
$$r_k = \left(\boldsymbol{f}_k - \boldsymbol{y}\right)^T \left(\boldsymbol{f}_k - \boldsymbol{y}\right).$$
Recall that the goal is to minimize the residual.
2. Using $\lambda_k = \lambda$, compute a new trial point $\boldsymbol{c}_{k+1}$ by solving the system (5), evaluate the new $\boldsymbol{f}_{k+1} = f(\boldsymbol{x}; \boldsymbol{c}_{k+1})$, and then compute the new residual r_{k+1}.
3. Repeat step 2 for $\lambda_k = \lambda/2$ and compute the resulting residual $\tilde{r}_{k+1}$. We now have three residuals, the previous value r_k, and the two new trial values r_{k+1} and $\tilde{r}_{k+1}$.
4. If $\tilde{r}_{k+1}$ is the smallest of the three residuals, then accept the trial value $\boldsymbol{c}_{k+1}$ obtained for $\lambda_k = \lambda/2$, decrease $\lambda \leftarrow \lambda/2$, and repeat the cycle again.
5. If r_{k+1} is the smallest of the three residuals, then accept the trial value $\boldsymbol{c}_{k+1}$ obtained for $\lambda_k = \lambda$, and repeat the cycle again.
6. If r_k is the smallest of the three residuals, then increase $\lambda \leftarrow 2\lambda$ and compute the resulting residual. Keep doubling λ until the residual is smaller than r_k. Accept the new value of $\boldsymbol{c}_{k+1}$ and repeat the cycle.

[*Hint: The MATLAB call diag(diag(A)) can be used to obtain* Diag$\boldsymbol{A}$*, as used in (5).*]

Test that the algorithm converges using an initial guess $\boldsymbol{c}_0$ that is close to the correct values.

[2.5 pts] If you start with $\boldsymbol{c}_0 = (1, 1, 1, 1)$, does the improved method converge to the correct answer?

[2.5 pts] Can you find some initial guess $\boldsymbol{c}_0$ for which even the improved method does not converge to the correct values? Does it converge to another local minimum or not converge at all?

[5 pts] If the synthetic data points have no error (i.e., $y_i = f(x_i; \boldsymbol{c})$ to within roundoff error), how many digits of accuracy in $\boldsymbol{c}$ can you obtain, starting with $\boldsymbol{c}_0 = (1, 1, 1, 1)$? How many steps do you need to achieve this accuracy.