Michael Hanke
CSC NA
September 24, 2009

DN2222
Numerical Algebra

KTH Computer Science and Communication

Programming assignment 3

Scope:

1. To plot functions and data series.

2. To understand simple properties of the singular value decomposition (SVD) and its use to determine numerical rank of a matrix.

3. Use of SVD in classification and data mining.

Read the chapter on SVD in the lecture notes.

You are encouraged to work in groups of two, and you may work at any time. The course assistant, Henrik Holst, will answer questions concerning these assignments at the scheduled lab sessions.

Hand in a report by December 14, at the Student Expedition! The report may be hand written, but would preferably include plots and matrices from MATLAB.

Do not hand in your entire MATLAB code, only a few lines where the interesting computations are done. *I will not like to see long* MATLAB *programs or* `diary` *files!*

1 Least squares curve fitting

Formulate the least squares problem you get when you are fitting a polynomial to a series of measurement data. If you have a series of m points t_i, $i = 1 : m$ and use polynomials of degree $n - 1$ you will get an $m \times n$ matrix A. The measurements y_i will make up a column vector with m elements.

A 1. Study the singular value decomposition of the matrix A when we have chosen equidistant t with $m = 101$ (use `t = linspace(-1,1,`m`)`) and $n = 20$:

   (a) The singular values σ_k decrease rather fast in size for larger k. Use `semilogy` to plot them. If data are given to moderate accuracy, say 4 decimals, there is no great benefit to use any singular values σ_k smaller than 10^{-4}. How large numerical rank r should we choose then?

(b) The columns of A are the first powers of t. Plot them as functions of t. See that they start to look very similar for higher degrees, this means that the matrix columns are nearly linearly dependent.

(c) Plot the first columns of U, the matrix of (left) singular vectors. They are orthogonal to each other. How does that show up? Look at the number of sign changes!

(d) As a comparison, do a QR factorization of A. Now each column q_k of the orthogonal factor Q is a polynomial of degree $k-1$. Plot them and compare to the columns of U!

Remark: The kth column q_k of Q is a polynomial of degree $k-1$. All columns of u_k of U are polynomials of the full degree $m-1$ but they should oscillate like the trigonometric functions, $\sin(k\pi t)$.

A 2. Use your insights from the previous assignment to compute polynomials to fit the following data series:

(a) $y(t) = \exp(-t)$. This is an entire function, which can be approximated by polynomials of any order. The approximation is better the higher degree you choose.

(b) $y(t) = |t|$. This function has a discontinuous derivative at $t = 0$ and you always get a large error close to that point.

(c) `y=exp(t)+e*randn(m,1)`, where the random perturbation has its size given by the parameter e. Choose a sequence of values of $e = 10^{-16}, 10^{-12}, 10^{-8}, \ldots$ until you do not longer recognize the exponential function.

In all these three cases, look at the transformed right hand side $b = U^T y$. Its elements b_k should get smaller and smaller for larger k.

Compare to the singular values of the matrix A. When an element b_k is significantly larger in absolute value than the singular value σ_k, we get a large component in the solution which does not decrease the residual by a significant amount.

The case with a random perturbation is of special interest. Here it is not useful to include any singular values σ_k smaller than the perturbation e.

(a) Plot the residual $r_k(t) = y(t) - p_k(t) = y - A * x_k$ in some interesting cases. Here x_k is the solution one gets by using the first k elements in b and a rank k approximation to the matrix A.

(b) Plot $\|r_k\|_2$ as a function of $\|x_k\|_2$. There is a break even point when increasing the rank k only gives a marginally smaller residual norm at the cost of a very large solution norm.

This last plot is a simple example of an L-curve. See works by Per Christian Hansen in Copenhagen if you want to know more!

2 Use SVD for pattern recognition

We borrow the following exercise from a NGSSC (National Graduate School of Scientific Computation) course given by Lars Eldén. If you want to know more, read his recent paper: Lars Eldén: Numerical linear Algebra in data mining, *Acta Numerica* (2006), pp. 327-384.[1] Algorithms of this kind are used to recognize hand written digits in postal codes, zip codes in US Postal service terminology. The data is from their test of prospective algorithms.

Construct an algorithm in MATLAB for character recognition of handwritten digits. Using a training set, compute an SVD of each matrix of digits of one kind. Use the first few (5-20) singular vectors as basis and classify unknown test digits according to how well they can be represented in terms of the respective bases (use the residual vector in the least squares problem as a measure). Try to tune the algorithm for accuracy of classification (varying the number of basis vectors). Check if all digits $0, 1, \ldots, 9$ are equally easy to classify or if some digits are easy to confuse.

Report the number of incorrectly classified digits in a table. Also look at some of those, and see that in many cases they are very badly written.

If time permits, check the singular values of the different digits, and see if it is motivated to use different numbers of basis vectors for different digits.

The test data can be fetched via the course homepage. Read the data with the command `load('zipdata')` The following files are then loaded:

1. `dtrain` and `atrain`: the first is a vector that holds the digits (the number) and the second is an array of dimension 256×1707 that holds the training images. The images are vectors of dimension 256, that have been constructed from 16×16 images.

2. The test data are given in `dtest` and `atest`. Use your algorithm on the columns of the matrix `atest` to see if you guess the coresponding values in the vector `dtest`! There are 2007 digits in the test set.

3. There is a function `ima2.m` that takes an image vector as input and displays it.

[1] Copies of some important pages will be distributed during the lecture.