CS450: Numerical Analysis
Fall 2015

Homework 2

University of Illinois, Urbana-Champaign, Prof. Paul Fischer

Due: September 16, 2015, 5.00 pm. Submit PDF + code files on Moodle.

Problem 1

Consider the expression

$$\frac{1}{1-x} - \frac{1}{1+x} \tag{1}$$

assuming $x \neq \pm 1$.

(a) For what range of values of x is it difficult to compute this expression *accurately* in floating-point arithmetic? (We are not looking for values of x that result in division by zero.)

(b) Give a rearrangement of the terms such that, for the range of x in part a, the computation is more accurate in floating-point arithmetic.

(c) Show computationally the improvement.

Problem 2

Suppose that $\boldsymbol{A}$ is a singular $n \times n$ matrix. Prove that if the linear systems $Ax = b$ has at least one solution $\boldsymbol{x}$, then it has infinitely many solutions.

Problem 3

Consider the linear system $A\mathbf{x} = \mathbf{b}$ with

$$\begin{bmatrix} e & 1 & 1 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix},$$

and $e = 8^{-20}$. All of these entries are exactly representable as IEEE 64-bit floating point numbers.

(a) Write down the new system and right-hand side after applying one round of Gaussian elimination without pivoting. Write your results in terms of numbers that are representable as IEEE floating point.

(b) What will happen if you execute the next round of Gaussian elimination? What has gone wrong here? Where has information been lost?

(c) Repeat (a) after applying partial pivoting to the original system. What information has been lost? Has anything important gone wrong here?

Problem 4

Prove that the Sherman-Morrison formula

$$(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{u}(1 - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1} \tag{2}$$

given in Section 2.4.9 is correct. (*Hint:* Multiply both sides by $\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T$.)

Problem 5

(a) Write a program implementing Gaussian elimination with no pivoting. (*Hint:* It may be helpful to verify the accuracy of your code using one of the examples from the book.)

(b) Generate several linear systems with random matrices as following

$$\boldsymbol{A} = \boldsymbol{I} + 0.01 * \text{rand}(n, n) \tag{3}$$

with $n = 50, 100, 200, 400$. Use the vector of all ones $\hat{\boldsymbol{x}} = [1, 1, \ldots, 1]^T$ to find the right hand side vector $\boldsymbol{b} := \boldsymbol{A}\hat{\boldsymbol{x}}$. Use your code from part a to solve for $\boldsymbol{x}$ in $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Compare your computed solution to the exact solution $\hat{\boldsymbol{x}}$ by computing the accuracy and residual.

(c) Use library functions to solve the same systems in part b. Measure the time taken solving them for both your code and library functions, plot them on the same graph. Compare your computed solution to the exact solution by computing the accuracy and residual. How does the performance and accuracy of your code compare to the library function?(*Hint:* backslash in MATLAB, `numpy.linalg.solve` in Python.)

Problem 6

Consider the polynomial

$$(1) \qquad p_N(x) := c_0x^0 + c_1x^1 + c_2x^2 + \cdots + c_Nx^N.$$

If we evaluate $p_N(x)$ at a set of points $\mathbf{x} = [x_1\, x_2 \cdots x_m]^T$, we can write the output as the matrix-vector product

$$(2) \qquad p_N(\mathbf{x}) := \begin{bmatrix} p_N(x_1) \\ p_N(x_2) \\ \vdots \\ \vdots \\ p_N(x_m) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & \cdots & x_1^N \\ 1 & x_2 & x_2^2 & \cdots & \cdots & x_2^N \\ \vdots & \vdots & \vdots & & & \vdots \\ \vdots & \vdots & \vdots & & & \vdots \\ 1 & x_m & x_m^2 & \cdots & \cdots & x_m^N \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_N \end{bmatrix} = A\,\mathbf{c}$$

If A is a square matrix with $m = n$, where $n = N+1$ is the number of columns, then we can formally use the equation above to solve the interpolation problem: *Find coefficients $c_0, \cdots, c_N$ such that $p_N(x_i) = f(x_i)$ for i=1, 2, $\ldots, N-1$, and a given function $f(x)$.* Library functions are recommended for solving the linear systems (for example, backslash in MATLAB, `numpy.linalg.solve` in Python).

(a) Write a code to set up the matrix A and solve for $\mathbf{c}$ for a set of points $x_j = -1 + (j-1)\Delta x$, $j = 1, \ldots, n$ with $\Delta x := 2/N$ (i.e., $n = N + 1$ uniformly distributed points on [-1,1]), with

$$f(x) \quad = \quad \frac{1}{1 + 25x^2}.$$

Once you have $\mathbf{c}$, evaluate p_N, using either (1) or (2), with a new set of points $x'_j = -1 + j\Delta x'$, $j = 0, \ldots, m'$ with $\Delta x' := 2/m'$ and m'=200. Plot both $f(x')$ and $p_N(x')$ vs. x' at these points for the case N=10.

(b) Repeat Part (a) with x_i taken to be the Chebyshev points $x_i = \cos(\pi j/N)$, $j = 0, \ldots, N$. As before, sample the new p_N on the uniformly distributed points $x'_j = -1 + j\Delta x'$, $j = 0, \ldots, m'$, with $\Delta x' := 2/m'$ and m'=200, and plot $f(x')$ and $p_N(x')$ vs. x' for $N = 10$.

(c) Rerun both cases (a) and (b), with $N = 2, 3, 4, \ldots, 40$. For each case, compute the maximum error,

$$e_N \quad := \quad \max_j \left| \frac{f(x'_j) - p_N(x'_j)}{f(x'_j)} \right|$$

and plot e_N vs. N on a semilogy plot. You should have two graphs on a single plot, one for the uniform points and one for the Chebyshev points.

(d) For the cases in Part (c), also monitor the condition number, (`cond(A)` in MATLAB; `numpy.linalg.cond(A)` in Python), and plot this value as a function of N for both cases using semilogy.

Note: There is some discussion of this topic in Secs. 7.3.1 and 7.3.5 of the text. We will also revisit this problem in the context of linear least squares.