ADVANCED MATHEMATICAL PROGRAMMING

MTH399 / U15161 – Level 3

UNIVERSITY OF PORTSMOUTH

Spring Semester Coursework, Referral/Deferral

Academic Session: 2010–2011

INSTRUCTIONS

a) **Deadline: Friday, July 22, at 3:30 on Victory**

b) This assignment serves as a **referral/deferral** for the unit.

c) Complete **at least** Part 1; Part 2 is left as a bonus.

d) Upload all code files to the Victory assignment; no printed copies are needed.

e) Make sure to label all materials with your reference number.

f) Explain your solutions using comments in the code.

g) If you cannot complete a question, show what you attempted to do.

h) Assignment must be undertaken **alone**.

STUDENT ID NUMBER: ....

Part 1. You will solve linear systems using Gaussian elimination. Let $n \in \mathbb{N}$ and

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} \in \mathrm{Mat}_n(\mathbb{R}), \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n,$$

be a square $n \times n$ matrix and a column n-vector with real entries, respectively. Our aim is to solve the system $A\boldsymbol{x} = \boldsymbol{b}$ for a vector of n unknowns $\boldsymbol{x}$, that is,

$$\left\{ \begin{array}{rcl} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n & = & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n & = & b_2 \\ & \vdots & \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n & = & b_n \end{array} \right. \tag{1}$$

We usually portray this system by using an augmented matrix $(A \mid \boldsymbol{b})$, written in the form

$$\left(\begin{array}{cccc|c} a_{1,1} & a_{1,2} & \dots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & +\dots & a_{n,n} & b_n \end{array} \right) \tag{2}$$

The process in which to solve system (1) or, alternatively, (2), comprises two steps:

a) reducing (2) to a triangular system $\left(A^{(n-1)} \mid \boldsymbol{b}^{(n-1)}\right)$ of the sort

$$\left(\begin{array}{ccccc|c} a_{1,1}^{(0)} & a_{1,2}^{(0)} & a_{1,3}^{(0)} & \dots & a_{1,n}^{(0)} & b_1^{(0)} \\ 0 & a_{2,2}^{(1)} & a_{2,3}^{(1)} & \dots & a_{2,n}^{(1)} & b_2^{(1)} \\ 0 & 0 & a_{3,3}^{(2)} & \dots & a_{3,n}^{(2)} & b_3^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n}^{(n-1)} & b_n^{(n-1)} \end{array} \right) \tag{3}$$

having zeroes under the diagonal. This is the process of Gaussian elimination proper. It starts with $\left(A^{(0)} \mid \boldsymbol{b}^{(0)}\right) = (A \mid \boldsymbol{b})$ and, for every $k = 0, \dots, n-1$,

$$i = k+1, \dots, n: \quad \left\{ \begin{array}{rcll} b_i^{(k+1)} & = & b_i^{(k)} - \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} b_k^{(k)}, & \\ a_{i,j}^{(k+1)} & = & a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} a_{k,j}^{(k)}, & j = 1, \dots, n \end{array} \right\} \tag{4}$$

At the end of the process, and except special cases mentioned in the comments, A should have been reduced to an upper triangular form $U = A^{(n-1)}$ as in (3).

b) Once A has been transformed into the upper triangular matrix U,

$$U = A^{(n-1)} = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{n,n} \end{pmatrix},$$

all it takes to complete the process is solving system $U\boldsymbol{x} = \boldsymbol{b}^{(n-1)}$. This is sometimes called **backward elimination**: it starts with $x_n = \frac{b_n^{(n-1)}}{u_{n,n}}$ and proceeds as follows:

$$x_i = \frac{1}{u_{i,i}} \left(b_i^{(i-1)} - \sum_{j=i+1}^{n} u_{i,j} x_j \right), \qquad i = n-1, n-2, \dots, 1. \tag{5}$$

Your project should contain, at least, the following:

a) a function `gauss` having at least three inputs: A, $\boldsymbol{b}$, and an output file, and transforming the first two inputs into $A^{(n-1)}$ and $\boldsymbol{b}^{(n-1)}$, or else returning an error message if straight Gaussian elimination was not possible (see Comments). For each step k of the elimination process, $\left(A^{(k)} \mid \boldsymbol{b}^{(k)}\right)$ (or an exit message if the step was unfeasible) will be written down on the output file in a well-structured manner.

b) a function `backward` having at least three inputs: an upper triangular matrix U, a vector $\boldsymbol{v}$, and the same output `ofstream` file as `gauss`, and returning the solution vector $\boldsymbol{x}$ to $U\boldsymbol{x} = \boldsymbol{v}$. Your function must perform a further check that such algorithm is feasible, meaning there is no element in the diagonal of U having an absolute value below a given tolerance tol.

c) a function named `norm` computing the vector norm of a given vector $\boldsymbol{v} \in \mathbb{R}^n$. You can use any of the common (and equivalent) Hölder norms available on $\mathbb{R}^n$, e.g.

$$\|\boldsymbol{v}\|_2 := \sqrt{v_1^2 + \cdots + v_n^2}, \qquad \|\boldsymbol{v}\|_1 := \sum_{i=1}^{n} |v_i|, \qquad \|\boldsymbol{v}\|_\infty := \max_{1 \le i \le n} |v_i|.$$

d) a function `check`, having inputs $A, \boldsymbol{x}, \boldsymbol{b}$, as well as the output file used above, which will compute the norm of the difference vector, $\|A\boldsymbol{x} - \boldsymbol{b}\|$. If such norm is smaller than tol, we will accept the solution as valid; otherwise, chances are there is a mistake somewhere in your project.

Comments:

a) Your project must comprise separate header (`*.h`) and `.cpp` files.

b) There will be matrices for which straight Gaussian elimination will not be possible at some step, or even in the first step, unless an additional process of row exchange called *pivoting* (which will not be the focus of attention in this exercise) is used. In practice, `gauss` should return an error message at a given step k whenever the element $a_{k,k}^{(k)}$ in the denominator in (4) is too small in absolute value. For instance,

$$A = \begin{pmatrix} 1 & 2 & 4 & 8 \\ 0 & 0 & -3 & 1 \\ 0 & 1 & 2 & -1 \\ 0 & 0 & 0 & 15 \end{pmatrix}$$

will cause a division by zero in the first step $k = 0 \to k = 1$; same applies to

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 5 & 7 & 13 \\ 7 & 6 & 1 & 4 \end{pmatrix}$$

for $k = 2 \to k = 3$; check this yourselves by hand, as well as using your program.

c) A further reason why linear system solution might not be possible using your method, is that the matrix be not invertible, meaning $\det A = 0$. Your project will still return an error message such as the one in the previous item – namely, that for some step k there is a pivot $a_{k,k}^{(k)}$ having an absolute value below tol, hence making the division in (4) numerically inadequate. However, there is still the question of whether the matrix chosen failed the process because it was non-invertible, or simply out of a poor choice of row order (such as in examples A and B in the previous item). Make sure to find examples of both cases.

d) As always, make sure to pass your output file by reference as an argument of your function; never declare output files globally.

e) Needless to say, everything described above has been written for entries ranging from 1 to n but this might not be possible or practical in your project; adapt the above computations accordingly.

Part 2. (Bonus) The purpose of this Part is to invert matrices using Gaussian elimination. Given $A \in \mathrm{Mat}_n(\mathbb{R})$, you will solve the system $AX = \mathrm{Id}_n$, where Id_n is the identity matrix and X is an $n \times n$ matrix. This is a novelty with respect to the previous system, in that both the unknown and the right-hand terms are now matrices instead of vectors. Nevertheless, you can adapt your previous function `gauss` into a new one called `matrixgauss` in a relatively easy manner, so as to replace the entries of $\boldsymbol{b}^{(k)}$ in (4) by the entries of a matrix $B^{(k)}$ which is equal to the identity matrix for $k = 0$. In other words, you are expected to solve n linear systems instead of one:

$$A\boldsymbol{x}_1 = \boldsymbol{e}_1, \quad A\boldsymbol{x}_2 = \boldsymbol{e}_2, \quad \dots \quad A\boldsymbol{x}_n = \boldsymbol{e}_n,$$

where $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$ (resp. $\boldsymbol{e}_1, \dots, \boldsymbol{e}_n$) are the column vectors of X (resp. Id_n). If done properly, `matrixgauss` should return an augmented $n \times 2n$ matrix

$$\left(A^{(n-1)} \mid B^{(n-1)}\right) = \left(A^{(n-1)} \mid \boldsymbol{b}_1^{(n-1)}\, \boldsymbol{b}_2^{(n-1)} \cdots \boldsymbol{b}_n^{(n-1)}\right)$$

and calling function `backward` n times, namely, on systems

$$A^{(n-1)}\boldsymbol{x}_1 = \boldsymbol{b}_1^{(n-1)}, \quad A^{(n-1)}\boldsymbol{x}_2 = \boldsymbol{b}_2^{(n-1)}, \quad \dots \quad A^{(n-1)}\boldsymbol{x}_n = \boldsymbol{b}_n^{(n-1)},$$

should yield the desired columns of the inverse matrix, $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n$.

You should check the validity of your result by computing $AX - \mathrm{Id}_n$ and verifying how "small" this difference is, namely that $\|AX - \mathrm{Id}_n\| < \mathrm{tol}$ for an adequate matrix norm $\|\cdot\|$, e.g. the Frobenius norm, the sub-1 norm or the sub-∞ norm:

$$\|B\|_F := \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |b_{i,j}|}, \qquad \|B\|_1 := \max_{1\le j\le n} \sum_{i=1}^{n} |b_{i,j}|\,, \qquad \|B\|_\infty := \max_{1\le i\le n} \sum_{j=1}^{n} |b_{i,j}|\,.$$

A function named `matrixnorm` could take care of this.

Comments:

a) Use separate header and `.cpp` files.

b) Maintain an output file as in Part 1 and write everything down on it.