

# CS 420 MACHINE PROJECT 2

Due date: Friday, March 10, 2006

## MP2 Outline

In this Machine Project, you are to implement a linear solver. The way we want you to numerically solve a system of linear equations is by (i) first performing the *Gaussian elimination* (possibly with *pivoting*), and then (ii) solving the resulting triangular system by *back-substitution*. Your system is to be *quadratic* (that is,  $n$  equations in  $n$  unknowns) and *non-homogeneous*.

## Mathematical Background: Systems of Linear Equations

A system of  $n$  linear equations in  $m$  unknowns has the following general form:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1m}x_m &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2m}x_m &= b_2 \\ &\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nm}x_m &= b_n \end{aligned} \tag{1}$$

In the sequel, we will only consider *non-homogeneous* systems where at least one  $b_i$  is not equal to zero. Also, for simplicity, instead of arbitrary *rectangular* systems as the one above, we will only consider the *square* systems where  $n = m$ .

*Gaussian elimination*, in terms of elementary matrix operations, is a systematic way of reducing an arbitrary *dense* matrix to its *upper triangular form*. An upper triangular matrix has all entries below the main diagonal equal to zero, i.e.,  $a_{ij} = 0$  for  $i > j$ . After the system (1) is reduced to its upper triangular form, it is subsequently solved for the values of  $x_n, \dots, x_1$  via the *back substitution*. This procedure is essentially how you were taught to solve a 2-by-2 or a 3-by-3 system of equations in high school. The first step of the back substitution is to solve for  $x_n$ ; that is done by simple division,  $x_n = b'_n/a'_{nn}$ , where  $b'_i$  and  $a'_{ij}$  pertain to the values on the right-hand side of the system and to the coefficients on the left-hand side, respectively, once the Gaussian elimination procedure has been completed. The general back substitution step is of the form

$$x_i = \frac{1}{a'_{ii}} \cdot (b'_i - \sum_{j=i+1}^n a'_{ij} x_j) \quad (2)$$

where the values of  $x_{i+1}, \dots, x_n$  have already been determined.

Notice that the above system may fail to have a solution altogether; this can happen when the matrix of coefficients  $A$  is singular. For more details, see any linear algebra textbook.

## Matrix Formulation

Let  $A \in R^n \times R^n$  be a square matrix with real entries, and let  $b \in R^n$  be an  $n$ -dimensional real-valued vector such that  $b^T \neq 0^n$  (where  $T$  stands for transpose). By convention, vectors such as  $b$  are treated as  $n$ -by-1 matrices (i.e., single column matrices), while their transposes are treated as 1-by- $n$  matrices (that is, single row matrices). A non-homogeneous system of linear equations can be written in the matrix notation as

$$A \cdot x = b \quad (3)$$

and the goal is to solve this system for the vector of  $n$  unknowns,  $x = [x_1, \dots, x_n]^T$ . Gaussian elimination procedure transforms the equation (3) into its equivalent form,

$$A' \cdot x = b' \quad (4)$$

where the resulting matrix  $A'$  is upper-triangular. Then one solves for the unknowns  $x_i$  via the *back-substitution* method (which is described in detail in your textbook [Quinn]).

Just like matrix multiplication, Gaussian elimination (in its default form) is a triple-nested-loop procedure that can be implemented in several different ways. The canonical *kij* implementation is given by the following pseudo-code:

```

DO k = 1 to n - 1
  a(k + 1 : n, k) = a(k + 1 : n, k) / a(k, k)
  DO i = k + 1 to n
    DO j = k + 1 to n
      a(i, j) = a(i, j) - a(i, k) * a(k, j)
    END DO
  END DO
END DO

```

You are required to implement, as a sequential program, any of the remaining five canonical versions, and then to parallelize that sequential program. You are required to implement two variants of the parallel Gaussian elimination: the row-oriented (see subsection §§12.4.3. in [Quinn]) and the column-oriented (see §§12.4.4. in [Quinn]). You also need to briefly compare and contrast the performances in these two cases, as well as with respect to the sequential execution. As for the back-substitution stage, it suffices that you do one parallel implementation: so, just implement the row-oriented parallel algorithm. Hence, there is a total of two parallel variants of a linear solver that you are required to implement.

## Partial Pivoting

Gaussian elimination with back substitution involves repeated division by the main diagonal elements, (the current value of)  $a'_{ii}$  (or, in the 2D array notation,  $a'(i, i)$ ). Obviously, this cannot be done if  $a'_{ii} = 0$ . Moreover, even if this value isn't exactly zero, but is very close to it (say, if  $a'_{ii} = 0.001$ , and some or all of other  $a'_{ij}$ 's are much larger in absolute value than  $a'_{ii}$ ), then, even though the floating point division is possible, the outlined algorithm may suffer from numerical instability. To avoid such problems, you are to implement, if needed, *partial pivoting*. Partial pivoting is a systematic procedure of pairwise swapping of matrix rows, so that, in each iteration, the current value of  $a'_{ii}$  (called *the pivot*) which is to appear as a denominator, is of a sufficiently big absolute value. One rule of thumb on how to systematically choose the pivot is that, in each iteration, the absolute value of the pivot should be the largest in its column.

So, before performing each division by the then current  $a'_{kk}$ , you should first test if  $|a'_{kk}| > \epsilon$ , where  $\epsilon$  is some small positive threshold constant. If  $a'_{kk}$  passes the test, you proceed the way you would without pivoting. If, however,  $a'_{kk}$  fails the test, then you look for a row  $j$  such that, if you swap rows  $j$  and  $k$ , the element  $a'_{jk}$ , which would then become the new pivoting element in the position  $(k, k)$ , passes the  $\epsilon$ -test and is as big (in the absolute value) as possible. Clearly, each time you swap two matrix rows  $j$  and  $k$ , you also must swap the RHS entries  $b_j$  and  $b_k$ . Once again, this swapping of the matrix rows (and the corresponding entries of vector  $b$ ) is called *partial pivoting*.

For more details, see [Quinn] or any standard text on numerical linear algebra.

## MP2 Implementation and Submission

Similarly to MP1, you can again choose either Fortran or C as your implementation language. The parallel part of your project again is to be implemented using Open MP within Fortran or C. You are expected to submit (i) a single program file with an implementation of all required variants of the Gaussian elimination based linear solver, (ii) a .doc or .pdf (or plain ascii text) file with a brief analysis and performance plots, and, optionally, (iii) the program file that generates the linear system of equations (i.e., your input) for the linear solver code provided in (i).

Your sequential and parallel linear solvers should work for  $n \times n$  systems of any (reasonable) size. You are to generate the matrix of coefficients,  $A$ , and the right-hand side vector,  $b$ . Choose the values of  $a(i, j)$  and  $b(i)$  in some bounded range, say, the real (really, floating point) interval  $[-100.00, +100, 00]$ . Expect  $n$  to be of the order of magnitude of several hundreds.

More details on the specific requirements will be posted to the course newsgroup in the upcoming days. Feel free to ask any questions you may have, either via the newsgroup or by emailing the TA directly.

**Note for the graduate students registered for 4 credit hours: there is no extra credit part for MP2**