Gauss Elimination programming assignment

Jan I.C. Vermaak[1,2]

[1]Center for Large Scale Scientific Simulations, Texas A&M Engineering Experiment Station, College Station, Texas, USA.
[2]Nuclear Engineering Department, Texas A&M University, College Station, Texas, USA.

Abstract:
Gauss elimination (GE) forms the basis of many direct solvers. In this assignment we explore the definition of routines that can help us solve linear systems of equations, with all the routines based on either GE-itself or some variant, i.e., the Tri-diagonal Matrix Algorithm (TDMA) or general banded matrix solvers.

Keywords: gauss elimination, tri-diagonal matrix, banded matrix solvers

1 Introduction

In C++ we can define a dense matrix and a vector in the following format:

```
typedef std::vector<double> Vec;
typedef std::vector<Vec> Mat;

Mat A(100,Vec(100,0.0));
Mat b(100,0.0);
```

For this case, we defined the matrix to be of dimension 100×100 and is initially zero. The vector b is of dimension 100×1 and is also zero. If the matrix is filled with entries then we can define the most rudimentary solution of a system

$$Ax = b$$

as

$$x = A^{-1}b$$

Gauss Elimination is the most fundamental algorithm to solve for x.

2 Gauss Elimination without pivoting

Algorithm 1: Gauss elimination without pivoting

Forward elimination:
for $j = 0;\ j < N-1;\ ++j$ **do**
  for $i = j+1;\ i < N;\ ++i$ **do**
    $c = \frac{a_{ij}}{a_{jj}}$
    for $k = 0;\ k < N;\ ++k$ **do**
      $a_{ik} = a_{ik} - ca_{jk}$

Back substitution:
for $i = N-1;\ i >= 0;\ --i$ **do**
  $S = b_i$
  for $j = i+1;\ j < N;\ ++j$ **do**
    $S = S - a_{ij}x_j$
  $x_i = \frac{S}{a_{ii}}$

The algorithm has two parts, a forward elimination part and a back-substitution part. The forward elimination step involves choosing a column j for which to eliminate all the values below the diagonal. Each row i, where $i = j+1$,

then subtracts from itself c times the row above it (i.e. row j). The factor c is chosen such that entry a_{ji} becomes zero. The forward elimination process is graphically shown below.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$\begin{array}{c} \cdots \\ \text{row}_1 = \text{row}_1 - \frac{a_{10}}{a_{00}}\ \text{row}_0 \\ \cdots \end{array} \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a^1_{11} & a^1_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b^1_1 \\ b_2 \end{bmatrix}$$

$$\begin{array}{c} \cdots \\ \cdots \\ \text{row}_2 = \text{row}_2 - \frac{a_{20}}{a_{00}}\ \text{row}_0 \end{array} \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a^1_{11} & a^1_{12} \\ 0 & a^2_{21} & a^2_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b^1_1 \\ b^2_2 \end{bmatrix}$$

$$\begin{array}{c} \cdots \\ \cdots \\ \text{row}_2 = \text{row}_2 - \frac{a_{21}}{a_{11}}\ \text{row}_1 \end{array} \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ 0 & a^1_{11} & a^1_{12} \\ 0 & 0 & a^3_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b^1_1 \\ b^3_2 \end{bmatrix}$$

The superscripts on a here just denote the value of a after a step. The second part of the algorithm is just simple back substitution.

3 The Tri-diagonal matrix algorithm (TDMA)

Also known as the Thomas algorithm. The algorithm assumes the system in the following form

$$\begin{bmatrix} b_0 & c_0 & & & 0 \\ a_1 & b_1 & c_1 & & \vdots \\ & a_2 & b_2 & \ddots & \\ & & \ddots & \ddots & c_{N-2} \\ 0 & & & a_{N-1} & b_{N-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix}$$

Algorithm 2: The tri-diagonal matrix algorithm

Require: $\overrightarrow{a}$, $\overrightarrow{b}$, $\overrightarrow{c}$, $\overrightarrow{d}$

for $i = 1;\ i < N;\ ++i$ **do**

$$w = \frac{a_i}{b_{i-1}}$$
$$b_i = b_i - wc_{i-1}$$
$$d_i = d_i - wd_{i-1}$$

$$x_{N-1} = \frac{d_{N-1}}{b_{N-1}}$$

for $i = N - 2;\ i \geq 0;\ --i$ **do**

$$x_i = \frac{(d_i - c_i x_{i+1})}{b_i}$$

4 Assignment

Part 1. Write a c++ function that performs Gauss Elimination. The function needs to have the following signature:

```
typedef std::vector<double> Vec;
typedef std::vector<Vec> Mat;

Vec GaussElimination(const Mat& A, const Vec& b);
```

Details:

- The matrix A and vector b is passed as a constant reference, so they cannot be modified. Make an internal copy of A and b.
- The function needs to return a new vector x.
- The function can deduce the size of the system, N, from the number of rows in the matrix but it should check that this is the same as the number of elements in b. If it is not then an error message needs to be printed and the program should quite using

```
//print message then...
exit(EXIT_FAILURE);
```

Part 2. Write a c++ function that executes the TDMA. The function needs to have the following signature:

```
typedef std::vector<double> Vec;

Vec TDMA(const Vec& a, const Vec& b, const Vec& c, const Vec& d);
```

Details:

- None of the vectors may be modified, so internal copies will need to be made.
- The size of the vectors a, b, c, d need to be equal. If it is not then print a message and make the program exit the same way as for Part 1.
- The function should return the vector x.

Part 3. Test the algorithms in part 1 and 2 with the following system: for row i

$$(2 + h^2)x_i - x_{i-1} - x_{i+1} = h^2 \quad i \in [0, N-1]$$
$$x_{-1} = x_N = 0$$
$$h = \frac{1}{N}$$

Test with $N = 10$ and $N = 20$

(Bonus)Part 4a. Make a variant of the TDMA that generalizes to a banded matrix with N_b number of bands with each band having equal number of entries (although not all the entries are actually used). The function needs to have the following signature:

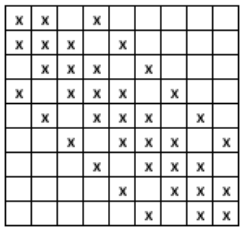
```
typedef std::vector<double> Vec;
typedef std::vector<int> VecInt;

Vec BandedSolver(std::vector<Vec>& a, Vec& b, const VecInt& offsets);
```

Details:

- The number of bands, N_b, can only be an odd number, i.e., given $k > 0$, $N_b = 2 * k + 1$.

- The offsets contain an offset for each band and is symmetric about the middle band. For example, 5 bands can have offsets $\{-3, -1, 0, +1, +3\}$ corresponding to the following system:



- The vector of vectors a is used for the matrix A's bands while b is used for the right-hand side of $Ax = b$.

(Bonus)Part 4b. Test the banded solver with the following system describing a 2D finite difference spatial discretization of the boundary value problem (BVP),

$$-\boldsymbol{\nabla} \cdot (\boldsymbol{\nabla} x) + x = 1,$$

that results in the discrete set of equations, given the constant N_c,

$$(4 + h^2)x_{ij} - x_{i-1j} - x_{i+1j} - x_{ij-1} - x_{ij+1} = h^2$$

$$\begin{aligned} x_{-1,j} = 0 \quad x_{N_c,j} &= 0 \\ x_{i,-1} = 0 \quad x_{i,N_c} &= 0 \end{aligned}$$

$$h = \frac{1}{N_c}$$

The cell-wise index ij maps to a single index as

$$(i, j) \mapsto k \ : \ k = i + jN_c$$

which represents a lexicographical sorting of cells at (x, y) index (i, j). Consequently the $N_c \times N_c$ number of cells map to a system of equations where the matrix dimension is $N_c^2 \times N_c^2$ and the dimension of the vectors are $N_c^2 \times 1$.