AM6007 Assignment 4

1 Introduction

Write a console app (program) in C# which solves a system of linear equations using the iterative Gauss-Jacobi method.

2 Gauss-Jacobi Method

Consider a linear system of n equations with n unknowns:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots &= \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \end{aligned}$$

Let

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$$

and

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

The system can be written in matrix/vector notation as:

$$\mathbf{Ax} = \mathbf{b}$$

The Gauss-Jacobi method splits $\mathbf{A}$ as follows:

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix} - \begin{pmatrix} 0 & 0 & \cdots & 0 \\ -a_{2,1} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n,1} & -a_{n,2} & \cdots & 0 \end{pmatrix} - \begin{pmatrix} 0 & -a_{1,2} & \cdots & -a_{1,n} \\ 0 & 0 & \cdots & -a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

That is:

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$$

where

1. $\mathbf{D}$ consists of the diagonal elements of $\mathbf{A}$ with zeroes everywhere else,
2. $\mathbf{L}$ consist of the negative of the lower diagonal elements of $\mathbf{A}$ with zeroes on the diagonal and above,
3. $\mathbf{U}$ consist of the negative of the upper diagonal elements of $\mathbf{A}$ with zeroes on the diagonal and below.

Hence our system becomes:

$$\begin{aligned}(\mathbf{D} - \mathbf{L} - \mathbf{U})\mathbf{x} &= \mathbf{b}\\ \mathbf{D}\mathbf{x} &= (\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{b}\end{aligned}$$

The inverse of $\mathbf{D}$ exists if none of the diagonal elements of $\mathbf{A}$ are zero so that:

$$\mathbf{x} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}$$

where:

$$\mathbf{D}^{-1} = \begin{pmatrix} \frac{1}{a_{1,1}} & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{2,2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{a_{n,n}} \end{pmatrix}.$$

The Gauss-Jacobi iterative scheme then follows:

$$\mathbf{x}_{k+1} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}_k + \mathbf{D}^{-1}\mathbf{b}$$

where $\mathbf{x}_k$ is the current estimate of the solution and $\mathbf{x}_{k+1}$ is the next estimate. Usually taking $\mathbf{x}_0$ to be all zeroes is a good enough initial guess. Furthermore let:

$$\begin{aligned}\mathbf{T} &= \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\\ \mathbf{c} &= \mathbf{D}^{-1}\mathbf{b}\end{aligned}$$

and notice that neither $\mathbf{T}$ nor $\mathbf{c}$ change during the calculation and can be calculated once at the begining. The iteration simplifies to:

$$\mathbf{x}_{k+1} = \mathbf{T}\mathbf{x}_k + \mathbf{c}$$

The stopping condition for ending the iteration is usually related to how close subsequent estimates are, such as:

$$\frac{|\mathbf{x}_{k+1} - \mathbf{x}_k|}{|\mathbf{x}_k|} < \epsilon$$

where ϵ is a tolerance (say 10^{-7}).

Thinking ahead to coding the solution, note that calculating $\mathbf{T}$ involves a matrix multiplication, calculating $\mathbf{c}$ involves multiplying a matrix by a vector. The iteration involves multiplying a matrix by a vector and vector addition. Implementing the stopping condition involves vector subtraction and calculating the norm (or absolute value) of a vector.

3 Implementation in C#

Details of the requirements:

1. Read and understand the previous section.
2. Create a class named Matrix which encapsulates an n by n square matrix.
3. Create the following methods:
   (a) Store the matrix data in a suitably sized 2D array.
   (b) A default constuctor which sizes the matrix to 3 by 3 and sets all values to zero.
   (c) A constructor public Matrix(int size);
   Make sure size > 1 otherwise throw an exception. Size the matrix accordingly and set all values to zero.
   (d) public static Matrix operator*(Matrix lhs, Matrix rhs)
   Implements matrix multiplication and checks that the matrices are of the same size otherwise throw an exception.
   (e) public static Vector operator*(Matrix lhs, Vector rhs)
   Implements multiplication of a matrix by a vector and checks that the matrix and vector are of the same size otherwise throw an exception
   (f) public double this[int row, int col]
   Implements an indexer function to allow users to access or set elements of the matrix. Check for valid indices and throw an exception if they are invalid.
   (g) public override string ToString()
   Returns a string representation of the matrix for use in Console.WriteLine.
4. Create a class named Vector which encapsulates a vector of size n.
5. Create the following methods:
   (a) Store the vector data in a suitably sized 1D array.
   (b) A default constuctor which sizes the vector to 3 and sets all values to zero.
   (c) A constructor public Vector(int size);
   Make sure size > 1 otherwise throw an exception. Size the vector accordingly and set all values to zero.

(d) static public Vector operator +(Vector a, Vector b)
Implements vector addition and checks that the vectors are of the same size otherwise throw an exception.

(e) static public Vector operator -(Vector a, Vector b)
Implements vector subtraction and checks that the vectors are of the same size otherwise throw an exception

(f) public double this[int index]
Implements an indexer function to allow users to access or set elements of the vector. Check for valid index and throw an exception if it is invalid.

(g) public double Norm()
Implements the norm (abolute value) of a vector.

(h) public override string ToString()
Returns a string representation of the vector for use in Console.WriteLine.

6. Create a class named LinSolve which implements the Gauss-Jacobi method

7. Create the following methods/data:

   (a) Provide a data member maxiters (= 100) to prevent infinite loops.

   (b) public Vector Solve(Matrix A, Vector b)
   Solves the matrix equation Ax = b and returns the resulting vector of the solution.

8. Your code should run against the code shown in Fig. 1. See also the output in Fig. 2.

9. Submit your answer by putting all code into a single file with an extension .cs. Check that your code runs and works before submitting. There will be marks allocated for tidy, readable and commented code.

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        try
        {
            Matrix m = new Matrix(4);
            Vector b = new Vector(4);
            m[0, 0] = 9; m[0, 1] =-2; m[0, 2] = 3; m[0, 3] = 2;
            m[1, 0] = 2; m[1, 1] = 8; m[1, 2] = -2; m[1, 3] = 3;
            m[2, 0] = -3; m[2, 1] = 2; m[2, 2] = 11; m[2, 3] = -4;
            m[3, 0] = -2; m[3, 1] = 3; m[3, 2] = 2; m[3, 3] = 10;
            b[0] = 54.5; b[1] = -14; b[2] = 12.5; b[3] = -21;
            Console.WriteLine("The matrix m is {0}", m);
            Console.WriteLine("The vector b is {0}", b);
            LinSolve l = new LinSolve();
            Vector ans = l.Solve(m, b);
            Console.WriteLine("The solution to m x = b is {0}", ans);

        }
        catch(Exception e)
        {
            Console.WriteLine("Error encountered: {0}", e.Message);
        }
        Console.ReadLine();
    }
}
```

Figure 1: Test your implementation against this code.

```
The matrix m is
{9, -2, 3, 2}
{2, 8, -2, 3}
{-3, 2, 11, -4}
{-2, 3, 2, 10}

The vector b is {54.500, -14.000, 12.500, -21.000}
The solution to m x = b is {5.000, -2.000, 2.500, -1.000}
```

Figure 2: Sample output.