Graduate CFD-I: Project Report #2

Dina Soltani Tehrani
dina.soltani@ae.sharif.edu

Sharif University of Technology — January 6, 2022

Contents

Introduction

Write a program in one of the Fortran, C +, C or C ++ languages that:

1. Solve the above equation by various finite difference methods presented in the booklet;
2. Draw the results using tecplot software;
3. According to the answers obtained, check and report the stability of each of the above methods;
4. Compare and report the accuracy of their answers with different spatial and temporal sizes of the computational network with the exact answer;
5. Compare the answers of the above methods and compare the advantages and disadvantages of each from the point of view of stability and accuracy Explain.

1 Problem Description and Governing Equations

The wave equation is a linear second-order partial differential equation which describes the propagation of oscillations at a fixed speed in some quantity u:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \tag{1}$$

The wave equation is a good description for a wide range of phenomena because it is typically used to model small oscillations about an equilibrium, for which systems can often be well approximated by Hooke's law. The wave equation's solutions are significant not just in fluid dynamics, but also in electromagnetic, optics, gravitational physics, and heat transfer. The solutions to the Fourier transform of the wave equation, which define Fourier series, spherical harmonics, and their generalizations, are particularly essential.
The 2D wave equation turns into two 1D equations as follows:

$$\frac{\partial u}{\partial t} - c\frac{\partial u}{\partial x} = 0 \tag{2}$$

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{3}$$

One popular solution for the wave equation is considered to be as follows:

$$u\left(x,t\right) = A\sin\left(kx - \omega t + \varphi\right), k = \frac{2\pi}{\lambda}, c = \frac{\lambda}{T} \tag{4}$$

1.1 Objectives

The objective of this project is to provide the solution of the one-dimensional wave equation, as described above using *three* different numerical schemes and to provide a comparison on the schemes in terms of Stability, Accuracy, and other general criteria of any numerical scheme. The whole algorithms are written in C++ language and are appended to this document. The codes are also available on github git, and will be emailed with this report as well.

1.2 Initial Condition

The initial condition for the problem is set to be zero or as below:

$$u(0,t) = 0 \tag{5}$$

1.3 Boundary Condition

The boundary condition for the problem is defined with the following piece-wise function:

$$\begin{cases} u(x,0) = 0 & 0 \le x \le 40 \\ u\left(x,0\right) = 80sin\left(\pi\frac{x-40}{60}\right) & 40 \le x \le 100 \\ u(x,0) = 0 & 100 \le x \le 300 \end{cases} \tag{6}$$

That says, the boundary condition is supposed to be a combination of a "Sinusoidal Condition" and a "Step Condition".

2 Exact Solution

As discussed earlier, the general form of the wave is a combination of a Sinusoidal wave and a Step excitation; therefore, the exact solution of the problem would be Sinusoidal tap moving along the x-dirextion in time. The equation below is representation of the exact solution:

$$
\begin{cases} u\left((x-ct),0\right) = 80sin\left(\pi\frac{(x-ct)-40}{60}\right) & 40 \leq (x-ct) \leq 100 \\ 0.0 & otherwise. \end{cases} \tag{7}
$$

3 Numerical Methods

3.1 Introduction

To solve the wave equation numerically, many explicit and implicit methods have been proposed, some of the most important of which are introduced below.

3.2 Euler Implicit Method

By applying this method to the wave equation, the following discrete shape is obtained. (In this method, c is assumed to be positive):

$$
\frac{u_i^{n+1} \ - \ u_i^n}{dt} \ + \ C \ \frac{u_{i+1}^{n+1} \ - \ u_{i-1}^{n+1}}{2dt} \ = \ 0 \tag{8}
$$

Here is a list of some important features of this method:

- The method has error of order "one in time" and "two in space".
- The method is "Unconditionally Stable".
- The method is an "Implicit" scheme.

As the method contains a central scheme, in order to be used for a biased problem (such as our wave-equation solution which is a hyperbolic problem, thus a biased one both in time and space), the algorithm needs a correction for the last node. For this last node, we would change the difference equation to the following equation:

$$
\frac{u_i^{n+1} - u_i^n}{dt} + C\ \frac{u_i^{n+1} - u_{i-1}^{n+1}}{dx} = 0 \tag{9}
$$

3.3 Lax Method

By applying this method to the wave equation, the following discrete shape is obtained. (In this method, c is assumed to be positive):

$$
u_i^{n+1} = \frac{1}{2}(u^{\ n}_{\ i+1} + u_{i-1}^n) - \frac{\nu}{2}(u_{i+1}^n - u_{i-1}^n) \tag{10}
$$

Here is a list of some important features of this method:

- The method has error of order "one in time" and "dx^2/dt in space".
- The method is "Conditionally Stable".
- The method is an "Explicit" scheme.

Same as the "Euler Implicit Method", the method contains a central scheme; therefore, in order to use the method for a biased problem (such as our wave-equation solution which is a hyperbolic problem, thus a biased one both in time and space), the algorithm needs a correction for the last node. For this last node, we again need to change the difference equation to the following equation:

$$
\frac{u_i^{n+1} - u_i^n}{dt} + C\ \frac{u_i^{n+1} - u_{i-1}^{n+1}}{dx} = 0 \tag{11}
$$

3.4 FTBS Method

This method is the chosen method for this report's in-depth study of the numerical scripting. The mesh and time study is done based on this method, and the results will be provided in the next section. By applying this method to the wave equation, the following discrete shape is obtained. (In this method, c is assumed to be positive):

$$\frac{u_i^{n+1} - u_i^n}{dt} + C\,\frac{u_i^{n+1} - u_{i-1}^{n+1}}{dx} = 0 \tag{12}$$

Here is a list of some important features of this method:

- The method has error of order "one in time" and "one in space".
- The method is "Conditionally Stable".
- The method is an "Explicit" scheme.

This method is the stable version of the FTCS method which is explicit an unconditionally unstable an is implemented based on the following difference equation:

$$\frac{u_i^{n+1} - u_i^n}{dt} + C\,\frac{u_{i+1}^{n} - u_{i-1}^{n}}{2dx} = 0 \tag{13}$$

4 Results and Discussion

4.1 Euler Implicit Method

The moving wave with time along the x-direction is shown in figure 1, solved with the Euler Implicit Method. If we consider the first wave in the green color in the figure as the reference, we can notice that this method provides both inaccuracy in the amplitude and the phase, as the amplitude is decaying with the wave moving with time and the deformation of the sinusoidal shape of the wave is the representative of the phase distortion.

4.2 Lax Method

The moving wave with time along the x-direction is shown in figure 2, solved with the Lax Method. If we consider the first wave in the red color in the figure as the references, we can notice that this method provides both inaccuracy in the amplitude and the phase, as well, since the amplitude is decaying with the wave moving with time and the deformation of the sinusoidal shape of the wave is the representative of the phase distortion. However, even with less number of mesh nodes in comparison to the Euler Implicit Method, the decay in the amplitude is much less.

4.3 FTBS Method

The moving wave with time along the x-direction solved with the FTBS Method is shown in following figure for different mesh node numbers and time step numbers.
For this method, first an array-based algorithm was written, which worked well for low number of mesh node numbers and time step numbers. However,the algorithm failed to compute the problem with finer discretization. Therefore, the whole algorithms for all other methods as well as the current method was transformed to a pointer-based structure which resulted in a more complex strategy, function referals as much as possible but provided considerable speed and memory optimization.
Figures 3, 4 , and 5 provide the moving wave with 1000 time step numbers and 301, 601, and 1201 mesh node numbers. It is apparent from the figures that the results have considerably improved in terms of the accuracy in both amplitude and phase with an increase in the mesh node number.
Figures 6, 7, 8, and 9 provide the moving wave with 1201 mesh node numbers and 500, 1000, and 2000 time step numbers. One can observe that the best result is with 1000 number of time step. While the results of the case with 2000 number of time steps is less accurate in terms of both amplitude and phase, the method fails to provide acceptable result with too low number of time step, 500 number of time steps as shown in figures 8, and 9.

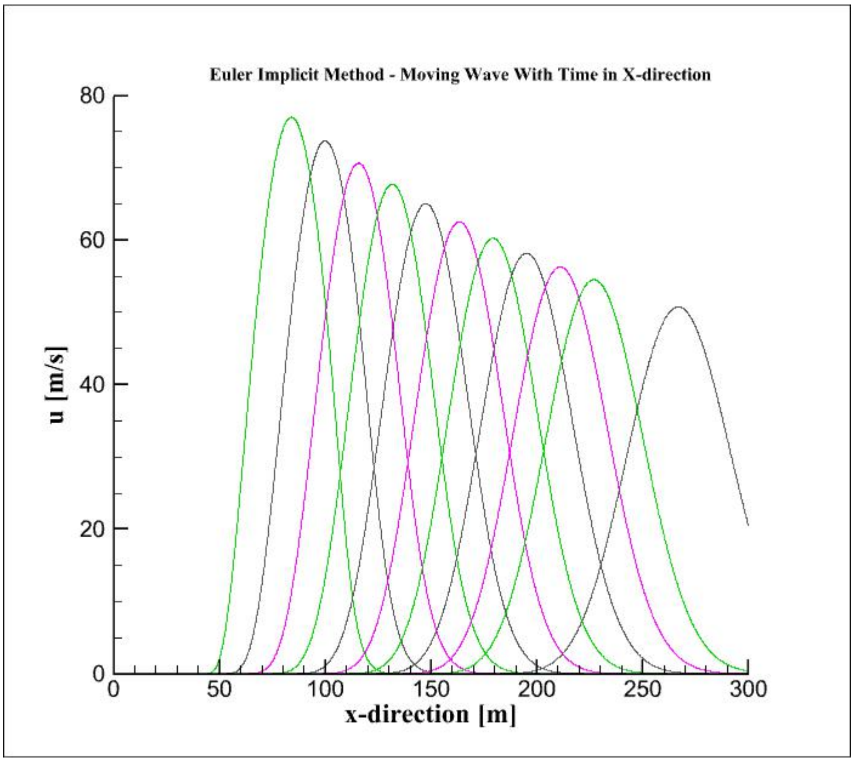

Figure 1: Euler Implicit Method - Number of Mesh Nodes: 2401 - Number of Time Steps: 100

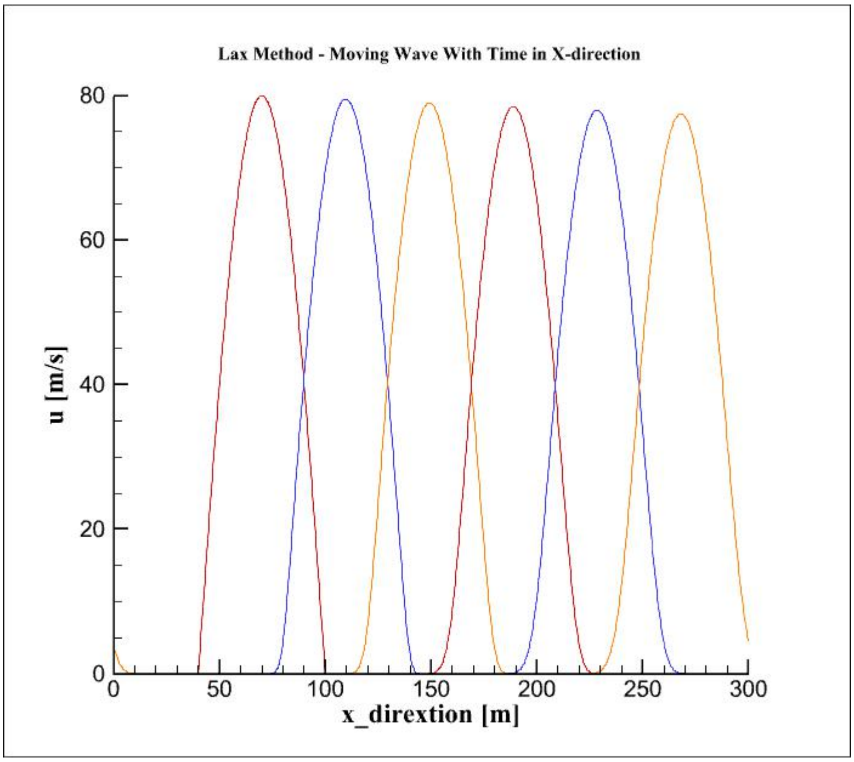

Figure 2: Lax Implicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 1000

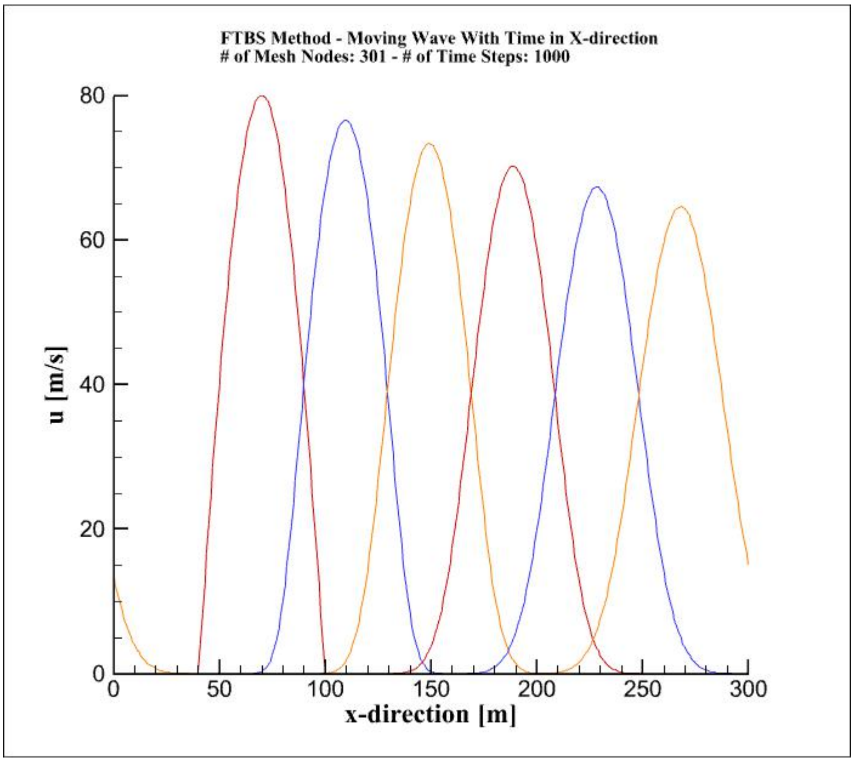

Figure 3: FTBS Explicit Method - Number of Mesh Nodes: 301 - Number of Time Steps: 1000

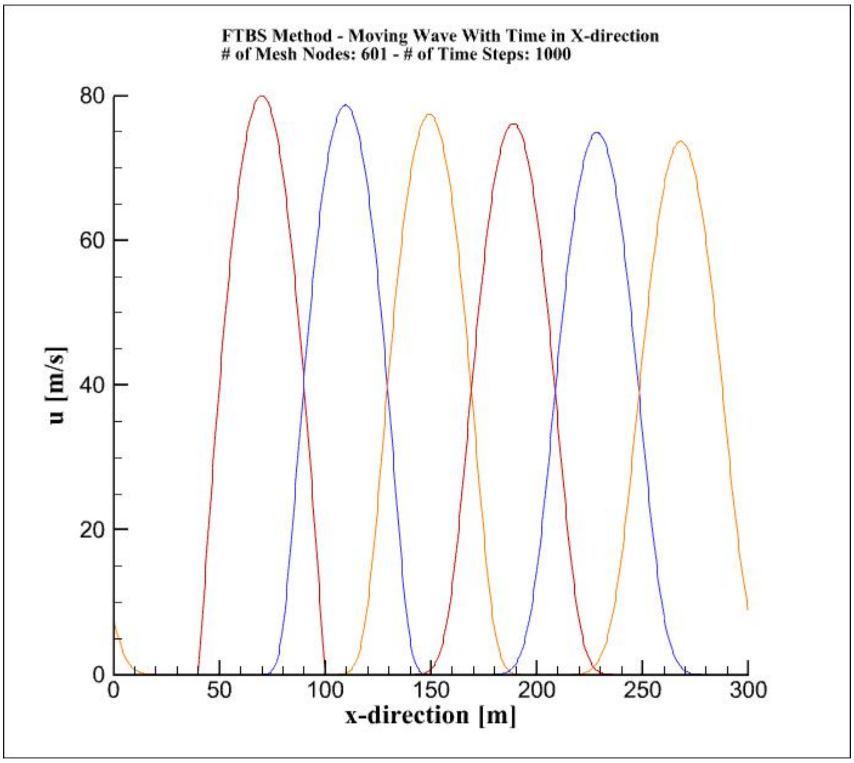

Figure 4: FTBS Explicit Method - Number of Mesh Nodes: 601 - Number of Time Steps: 1000

5 References

1. George Lindfield, John Penny, Numerical Methods Using MATLAB, Second Edition, Prentice Hall, 1999, ISBN: 0-13-012641-1, LC: QA297.P45.

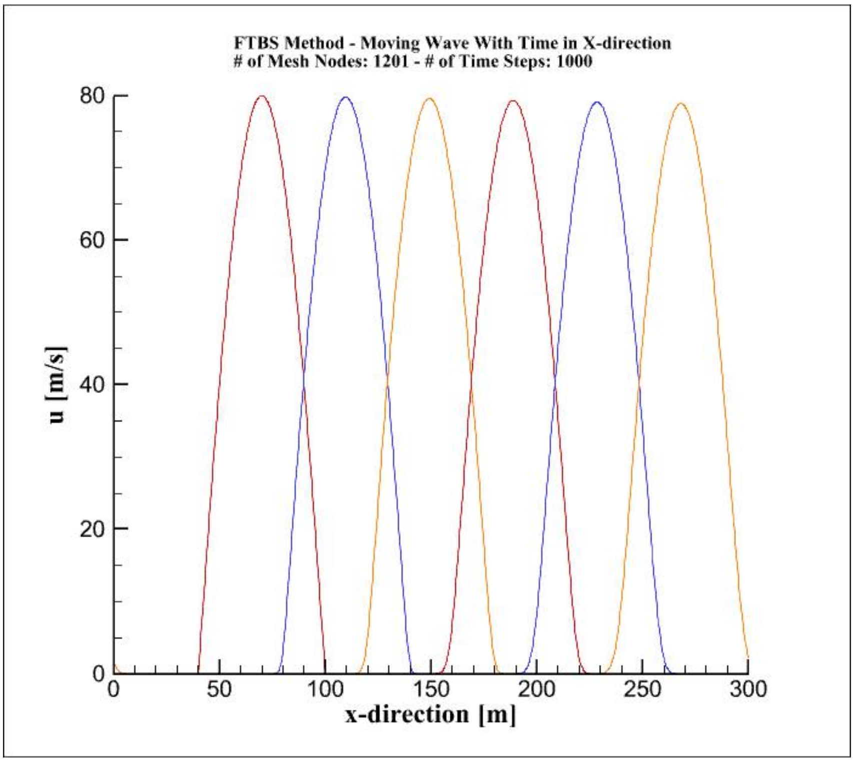

Figure 5: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 1000

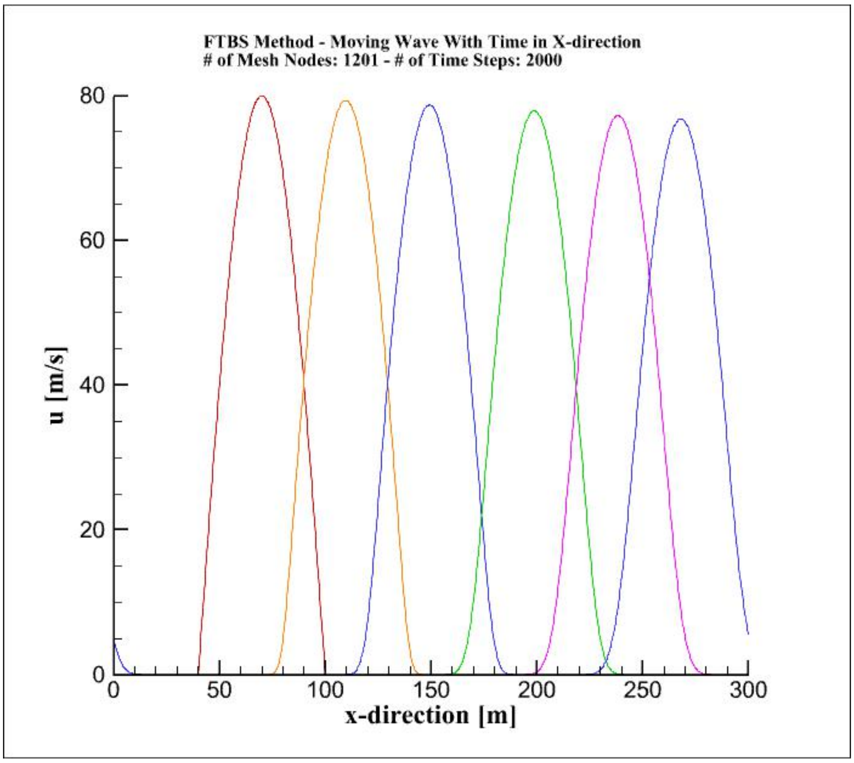

Figure 6: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 2000

2. Simon Fraser University, Computer Science Department, Numerical Methods Resource Website of John Burkardt, John Burkardt, 2020.

3. Advanced CFD Course Lecture Notes by Dr. Taeibi Rahni, Aerospace Engineering Department, Sharif

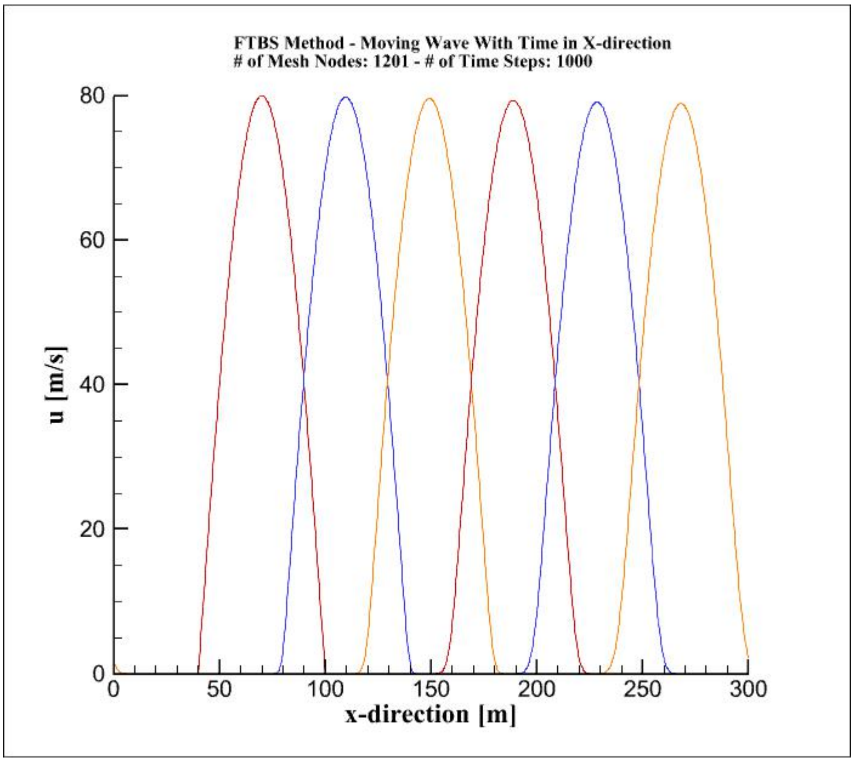

Figure 7: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 1000

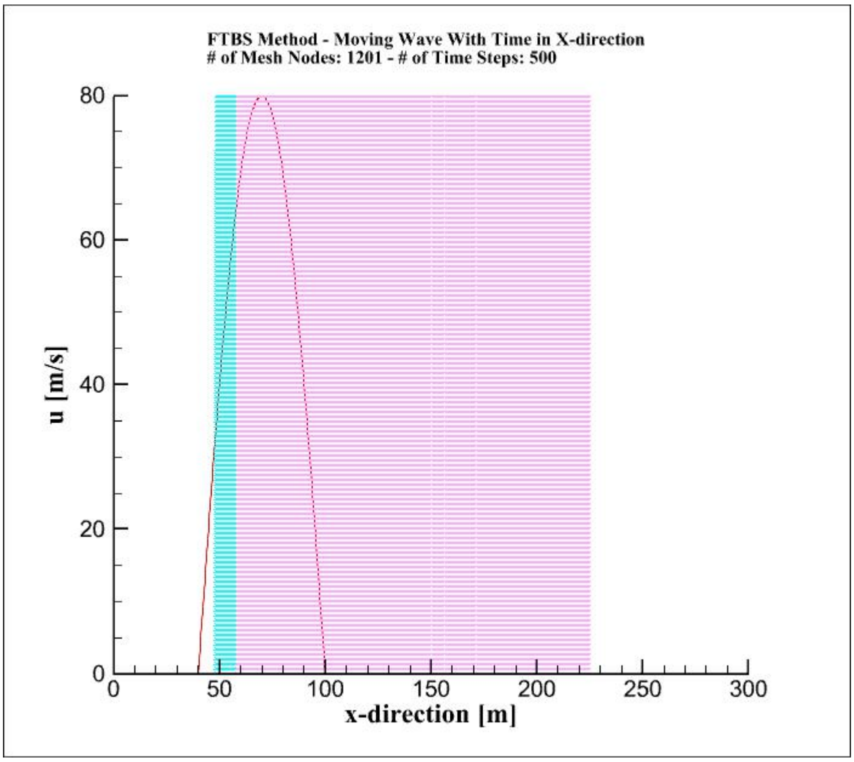

Figure 8: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 500 - Zoom out

University of Technology, 2021-2022.

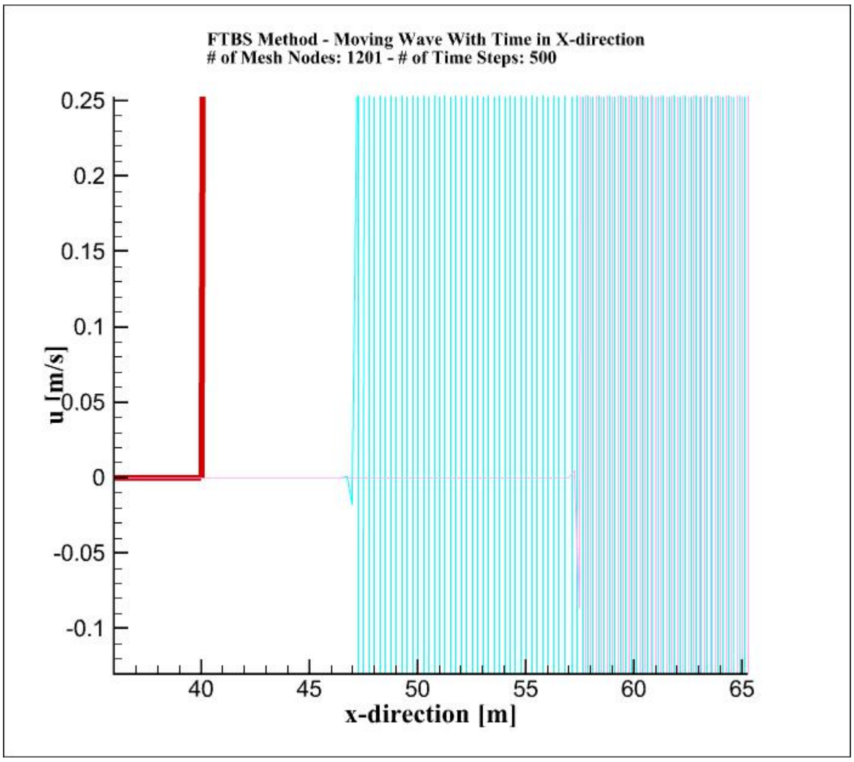

Figure 9: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 500 - Zoom in

6 Appendices

6.1 Code for The Euler Implicit Method

```
%**********************************************************************
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <fstream>
# include <ctime>
# include <cmath>

//  Language: C++
//  Author: Dina Soltani Tehrani.
//  Course: Advanced CFD1 - by Dr. Taeibi Rahni
//  Sharif University of Technology
//
//  Equation: ut = -c * ux
//  Method: Finite difference , FTCS, Explicit
//
//    The finite difference form is as bellow:
//
//      U(X,T+dt) - U(X,T)          ( U(X-dx,T+dt) + U(X+dx,T+dt) )
//      ------------------   = -c * -------------------------------- + F(X, T+dt)
//              dt                              2 * dx
//
//            +     c * dt / dx / 2    * U(X+dx,T+dt)
//            +     1                  * U(X,   T+dt)
//            -     c * dt / dx / 2    * U(X-dx,T+dt)
//      =                                U(X,   T)
//      +                        dt    * F(X,   T+dt)

using namespace std;
//************************************************************************
// Functions

int main ( );
```

```
void dtable_data_write ( ofstream &output, int m, int n, double table[] );

void dtable_write ( string output_filename, int m, int n, double table[],
  bool header );

void f ( double a, double b, double t0, double t, int n, double x[],
  double value[] );

int r83_np_fa ( int n, double a[] );

double *r83_np_sl ( int n, double a_lu[], double b[], int job );

void timestamp ( );

void u0 ( double a, double b, double t0, int n, double x[], double value[] );

double ua ( double a, double b, double t0, double t );
double ub ( double a, double b, double t0, double t );

//****************************************************************************

int main ( )
//****************************************************************************
{
  double *a;
  double *b;
  double *fvec;
  bool header;
  int i;
  int j;
  int job;
  double k;
  double *t;
  double dt;
  string t_file;
  double t_max;
  double t_min;
  int nt;
  double *u;
  string u_file;
  double w;
  double *x;
  double dx;
  string x_file;
  double x_max;
  double x_min;
  int nx;
  string final_file;
  double c; // wave speed
  timestamp ( );

//  nu= 5.0E-07;
  c = 330.0;
//****************************************************************************
//  Set X values.
  x_min = 0.0;
  x_max = 300.0;
  nx = 2401;
  dx = ( x_max - x_min ) / ( double ) ( nx - 1 );

  x = new double[nx];

  for ( i = 0; i < nx; i++ )
  {
    x[i] = ( ( double ) ( nx - i - 1 ) * x_min
           + ( double ) (          i   ) * x_max )
           / ( double ) ( nx     - 1 );
  }

//****************************************************************************
//  Set Time values.
  t_min = 0.0;
```

```
  t_max = 0.6;
  nt = 100;
  dt = ( t_max - t_min ) / ( double ) ( nt - 1 );

  t = new double[nt];

  for ( j = 0; j < nt; j++ )
  {
    t[j] = ( ( double ) ( nt - j - 1 ) * t_min
           + ( double ) (         j   ) * t_max )
           / ( double ) ( nt     - 1 );
  }

//***************************************************************************
//  Set the initial data, for time T_MIN.

  u = new double[nx*nt];

  // returning the initial condition at the starting time:
  u0 ( x_min, x_max, t_min, nx, x, u );

//***************************************************************************
//  Defining the matrix A. This matrix is constant and does not change with time.
//  We can set it once, factor it once, and solve repeatedly.

 w = c* dt /dx /2; // w = stab as the stability checker.

  a = new double[3*nx];

  a[0+0*3] = 0.0;
  a[1+0*3] = 1.0;
  a[0+1*3] = 0.0;

  for ( i = 1; i < nx - 1; i++ )
  {
    a[2+(i-1)*3] =           - w;
    a[1+ i    *3] =           1.0;
    a[0+(i+1)*3] =           + w;
  }

  a[2+(nx-2)*3] =           -2*w;
  a[1+(nx-1)*3] =      1.0 + 2*w;
  a[2+(nx-1)*3] =            0.0;

//  Factor the matrix.

  // factoring the R83 system without pivoting:
  r83_np_fa ( nx, a );

//***************************************************************************
// The right-hand side of the system of equation:

  b = new double[nx];
  fvec = new double[nx];

  for ( j = 1; j < nt; j++ )
  {

//  Set the right hand side B.
//
    b[0] = ua ( x_min, x_max, t_min, t[j] ); // returns the Dirichlet boundary condition at
    the left endpoint.

    // returning the right hand side of the 1d wave equation:
  // for this case, the excitation term is set to be zero;

  f ( x_min, x_max, t_min, t[j-1], nx, x, fvec );

    for ( i = 1; i < nx; i++ )
    {
      b[i] = u[i+(j-1)*nx] + dt * fvec[i];
    }
```

```
    //b[nx-1] = ub ( x_min, x_max, t_min, t[j] ); //returns the Dirichlet boundary condition
     at the right endpoint.

    delete [] fvec;

    job = 0;
  // solveing the R83 system factored by R83_NP_FA:
    fvec = r83_np_sl ( nx, a, b, job );

    for ( i = 0; i < nx; i++ )
    {
      u[i+j*nx] = fvec[i];
    }
  }
//****************************************************************************80
// data handelling:

  x_file = "x.txt";
  header = false;
  dtable_write ( x_file, 1, nx, x, header ); // writes information to a DTABLE file.

  cout << "\n";
  cout << "  X data written to \"" << x_file << "\".\n";

  t_file = "t.txt";
  header = false;
  dtable_write ( t_file, 1, nt, t, header ); // writes information to a DTABLE file.

  cout << "  T data written to \"" << t_file << "\".\n";

  u_file = "u.txt";
  header = false;
  dtable_write ( u_file, nx, nt, u,  header ); // writes information to a DTABLE file.

  final_file = "EuImp_x2401_t100_ed2.txt";
  header = false;
  dtable_write ( final_file, nx, nt, u, header ); // writes information to a DTABLE file.

  cout << "  Final data written to \"" << final_file << "\".\n";

  delete [] a;
  delete [] b;
  delete [] fvec;
  delete [] t;
  delete [] u;
  delete [] x;
//
//  Terminate.
//
  cout << "\n";
  cout << "  Finite Difference: Euler Implicit for 1D Wave Equation\n";
  cout << "  Normal end of execution.\n";
  cout << "\n";
  timestamp ( );

  return 0;
}
//****************************************************************************80

void dtable_data_write ( ofstream &output, int nx, int nt, double table[] )

{
  int i;
  int j;
  //double *vect;
  double x_min = 0;
  double x_max = 300;
  double t_min = 0.0;
  double t_max = 0.6;
  double dx;
  //double dt;
```

```
  double vect[nt][nx];
  dx = ( x_max - x_min ) / ( double ) ( nx - 1 );
  //dt = ( t_max - t_min ) / ( double ) ( nt - 1 );

  for ( j = 0; j < nt; j++ )
  {
    output << "zone\n";
  for ( i = 0; i < nx; i++ )
    {
      vect[j][i] = table[i+j*nx];
    output << x_min + i*dx << '\t' << vect[j][i] << '\n';
    //output << setw(10) << table[i+j*dx] << "  ";
    }
    output << "\n";
  }

  return;
}
//****************************************************************************

void dtable_write ( string output_filename, int m, int n, double table[],
  bool header )

{
  ofstream output;

  output.open ( output_filename.c_str ( ) );

  if ( !output )
  {
    cerr << "\n";
    cerr << "DTABLE_WRITE - Fatal error!\n";
    cerr << "  Could not open the output file.\n";
    return;
  }

  if ( header )
  {
//  dtable_header_write ( output_filename, output, m, n );
  }

  dtable_data_write ( output, m, n, table );

  output.close ( );

  return;
}
//****************************************************************************

void f ( double a, double b, double t0, double t, int n, double x[],
  double value[] )

{
  int i;

  for ( i = 0; i < n; i++ )
  {
    value[i] = 0.0;
  }
  return;
}
//****************************************************************************

int r83_np_fa ( int n, double a[] )

{
  int i;

  for ( i = 1; i <= n-1; i++ )
  {
    if ( a[1+(i-1)*3] == 0.0 )
    {
```

```
      cout << "\n";
      cout << "R83_NP_FA - Fatal error!\n";
      cout << "  Zero pivot on step " << i << "\n";
      return i;
    }
//
//  Store the multiplier in L.
//
    a[2+(i-1)*3] = a[2+(i-1)*3] / a[1+(i-1)*3];
//
//  Modify the diagonal entry in the next column.
//
    a[1+i*3] = a[1+i*3] - a[2+(i-1)*3] * a[0+i*3];
  }

  if ( a[1+(n-1)*3] == 0.0 )
  {
    cout << "\n";
    cout << "R83_NP_FA - Fatal error!\n";
    cout << "  Zero pivot on step " << n << "\n";
    return n;
  }

  return 0;
}
//****************************************************************************80

double *r83_np_sl ( int n, double a_lu[], double b[], int job )

{
  int i;
  double *x;

  x = new double[n];

  for ( i = 0; i < n; i++ )
  {
    x[i] = b[i];
  }

  if ( job == 0 )
  {
//
//  Solve L * Y = B.
//
    for ( i = 1; i < n; i++ )
    {
      x[i] = x[i] - a_lu[2+(i-1)*3] * x[i-1];
    }
//
//  Solve U * X = Y.
//
    for ( i = n; 1 <= i; i-- )
    {
      x[i-1] = x[i-1] / a_lu[1+(i-1)*3];
      if ( 1 < i )
      {
        x[i-2] = x[i-2] - a_lu[0+(i-1)*3] * x[i-1];
      }
    }
  }
  else
  {
//
//  Solve U' * Y = B
//
    for ( i = 1; i <= n; i++ )
    {
      x[i-1] = x[i-1] / a_lu[1+(i-1)*3];
      if ( i < n )
      {
        x[i] = x[i] - a_lu[0+i*3] * x[i-1];
```

```
      }
    }
//
//  Solve L' * X = Y.
//
    for ( i = n-1; 1 <= i; i-- )
    {
      x[i-1] = x[i-1] - a_lu[2+(i-1)*3] * x[i];
    }
  }

  return x;
}
//****************************************************************************80

void timestamp ( )

{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}
//****************************************************************************80

void u0 ( double a, double b, double t0, int nx, double x[], double value[] )

{
  int i;
  double  pi = 4*atan(1.0);

  for ( i = 0; i < nx; i++ )
  {
    if  ( 40 <= x[i] && x[i] <= 100 )
    {
      value[i] = 80 * sin( (pi/60)*(x[i] - 40) );
    }
    else
    {
      value[i] = 0.0;
    }
  }

  return;
}
//****************************************************************************80

double ua ( double a, double b, double t0, double t )

{
  double value;

  value = 0;

  return value;
}
//****************************************************************************80

double ub ( double a, double b, double t0, double t )

{
```

```
  double value;

  value = 2.14618;

  return value;
}

%*****************************************************************************
```

6.2 Code for The Lax Method

```
%*****************************************************************************
# include <cstdlib>
# include <iostream>
# include <fstream>
# include <iomanip>
# include <cmath>
# include <ctime>
# include <cstring>

using namespace std;

int main ( );
int i4_modp ( int i, int j );
int i4_wrap ( int ival, int ilo, int ihi );
double *initial_condition ( int nx, double x[] );
double *r8vec_linspace_new ( int n, double a, double b );
void timestamp ( );

//****************************************************************************

int main ( )

//****************************************************************************
//
// Language: C++
//  Author: Dina Soltani Tehrani.
//  Course: Advanced CFD1 - by Dr. Taeibi Rahni
//  Sharif University of Technology
//
//  Equation: ut = -c * ux
//  Method: Finite difference, Lax Method, Explicit
//
//    The finite difference form is as bellow:
//
//      U(X,T+dt) - U(X,T)          ( U(X-dx,T+dt) + U(X+dx,T+dt) )
//      ------------------  = -c * -------------------------------- + F(X, T+dt)
//              dt                              2 * dx
//

{
  double a;
  double b;
  double c;
  double L;
  double duration;
  double  pi = 4*atan(1.0);
  string data_filename = "Lax_x1201_t1000.txt";
  ofstream data_unit;
  double dt; // time-step
  double dx; // size-step
  int i;
  int j;
  int jm1;
  int jp1;
  int nx;
  int nt;
  int nt_step;
  int plotstep;
  double t;
```

```
  double *u; // old velocity
  double *unew; // new velocity
  double *x;

  cout << "\n";
  cout << "Finite Difference: Lax Method for 1D Wave Equation:\n";
  cout << "  C++ language\n";
  cout << "\n";
  cout << "  Solve the constant-velocity advection equation in 1D,\n";
  cout << "    du/dt = - c du/dx\n";
  cout << "  over the interval:\n";
  cout << "    0.0 <= x <= 300.0\n";
  cout << "  with a given boundary conditions, and\n";
  cout << "  with a given initial condition\n";
  cout << "    u(0,x) = 80 * sin( (pi/60)* (x-40) ) for 40.0 <= x <= 100.0\n";
  cout << "           = 0 elsewhere.\n";
  cout << "\n";
  cout << "  We use a method known as Lax:\n";
  cout << "   FT: Forward Time  : du/dt = (u(t+dt,x)-u(t,x))/dt\n";
  cout << "   BS: Backward Space: du/dx = (u(t,x)-u(t,x-dx))/dx\n";
  cout << "\n";
  timestamp ( );

  L = 300; // Length (m)
  nx = 1201; // Mid Space Resolution
  dx = L / ( double ) ( nx - 1 ); // size-step
  a = 0.0; // x-start
  b = 300.0; // x-end
  x = r8vec_linspace_new ( nx, a, b );
  nt = 1000; // Time Resolution
  duration = 0.6; // Duration (s)
  dt = duration / ( double ) ( nt ); // time-step
  c = 330.0;

  u = initial_condition ( nx, x ); // Initializing old velocity with initial condition.

//  ***** Opening data file, and writing solutions as they are computed. *****

  data_unit.open ( data_filename.c_str ( ) );

  // Writing the initial data.
  t = 0.0;
  data_unit << x[0]
            << "  " << u[0] << "\n";

  // Writing data as we move forward in space.
  for ( j = 0; j < nx; j++ )
  {
    data_unit << x[j]
              << "  " << u[j] << "\n";
  }
  data_unit << "zone";
  data_unit << "\n";



//
    *************************************************************************************************

  nt_step = 100;
  unew = new double[nx]; // Creating the new velocity vector

// ***** Core Calculation: begin *****
  for ( i = 0; i < nt; i++ )
  {
    //data_unit << "zone\n";
    // First do this for all x in xn:
    for ( j = 0; j < nx; j++ )
    {
      jm1 = i4_wrap ( j - 1, 0, nx - 1 ); // phi_n_j-1
      jp1 = i4_wrap ( j + 1, 0, nx - 1 ); // phi_n_j+1
      if ( j != nx-1)
```

```
      {
        unew[j] = (0.5 * ( 1 - (c * dt / dx) ) ) * u[jp1] + (0.5 * ( 1 + (c * dt / dx) ) ) *
     u[jm1];
        //if ( u[j] < 1e-04 ) { u[j] = 0; }
      }
      else
      {
        unew[j] = u[j] - (c * dt / dx) * ( u[j] - u[jm1] );
      }
    }

    // Then, do this, again for all x in xn:
    for ( j = 0; j < nx; j++ )
    {
      u[j] = unew[j];
      //data_unit << x[j] << "  " << u[j] << "\n";
    }

    if ( i == nt_step - 1 )
    {
      t = ( double ) ( i ) * dt;
      for ( j = 0; j < nx; j++ )
      {
        //if ( u[j] < 1e-05 ) { u[j] = 0; }
        data_unit << x[j]
                  << "  " << u[j] << "\n";
      }
      data_unit << "zone";
      data_unit << "\n";
      nt_step = nt_step + 100;
    }

    //data_unit << "zone";
    //data_unit << "\n";
  }
// ***** Core Calculation: end *****


//
//  ***** Closing the data file as the computation is done. *****
//
  data_unit.close ( );

  cout << "\n";
  cout << "  Plot data written to the file \"" << data_filename << "\"\n";
//

//  Free memory.
//
  delete [] u;
  delete [] unew;
  delete [] x;
//
//  Terminate.
//
  cout << "\n";
  cout << "  Finite Difference: Lax for 1D Wave Equation\n";
  cout << "  Normal end of execution.\n";
  cout << "\n";
  timestamp ( );

  return 0;
}

//
// ***** Functions *****

//****************************************************************************
int i4_modp ( int i, int j )
{
  int value;

```

```
  if ( j == 0 )
  {
    cerr << "\n";
    cerr << "I4_MODP - Fatal error!\n";
    cerr << "  I4_MODP ( I, J ) called with J = " << j << "\n";
    exit ( 1 );
  }

  value = i % j;

  if ( value < 0 )
  {
    value = value + abs ( j );
  }

  return value;
}

//****************************************************************************
int i4_wrap ( int ival, int ilo, int ihi )
{
  int jhi;
  int jlo;
  int value;
  int wide;

  if ( ilo <= ihi )
  {
    jlo = ilo;
    jhi = ihi;
  }
  else
  {
    jlo = ihi;
    jhi = ilo;
  }

  wide = jhi + 1 - jlo;

  if ( wide == 1 )
  {
    value = jlo;
  }
  else
  {
    value = jlo + i4_modp ( ival - jlo, wide );
  }

  return value;
}

//****************************************************************************
double *initial_condition ( int nx, double x[] )
{
  int i;
  double *u;
  double  pi = 4*atan(1.0);

  u = new double[nx];

  for ( i = 0; i < nx; i++ )
  {
    if  ( 40 <= x[i] && x[i] <= 100 )
    {
      u[i] = 80 * sin( (pi/60)*(x[i] - 40) );
    }
    else
    {
      u[i] = 0.0;
    }
  }
  return u;
```

```
}

//****************************************************************************
double *r8vec_linspace_new ( int n, double a_first, double a_last )
{
  double *a;
  int i;

  a = new double[n];

  if ( n == 1 )
  {
    a[0] = ( a_first + a_last ) / 2.0;
  }
  else
  {
    for ( i = 0; i < n; i++ )
    {
      a[i] = ( ( double ) ( n - 1 - i ) * a_first
             + ( double ) (         i ) * a_last )
             / ( double ) ( n - 1     );
    }
  }
  return a;
}

//****************************************************************************
void timestamp ( )
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct std::tm *tm_ptr;
  size_t len;
  std::time_t now;

  now = std::time ( NULL );
  tm_ptr = std::localtime ( &now );

  len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );

  std::cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}

%*******************************************************************
```

6.3 Code for The FTBS Method – Array-based

```
%*******************************************************************
# include <cstdlib>
# include <iostream>
# include <fstream>
# include <iomanip>
# include <cmath>
# include <ctime>
# include <cstring>
# include <stdio.h>      /* printf */
# include <math.h>       /* sin */
const double pi = 3.14159265359;
using namespace std;
using std::ofstream;
using std::cerr;
using std::endl;

//
// Language: C++
//  Author: Dina Soltani Tehrani.
```

```
//  Course: Advanced CFD1 - by Dr. Taeibi Rahni
//  Sharif University of Technology
//
//  Equation: ut = -c * ux
//  Method: Finite difference, FTBS Method, Explicit
//  Pointer-based Method
//
//    The finite difference form is as bellow:
//
//      U(X,T+dt) - U(X,T)          ( U(X-dx,T+dt) + U(X+dx,T+dt) )
//      ------------------  = -c * ---------------------------------- + F(X, T+dt)
//              dt                              2 * dx
//

int main ( );

//****************************************************************************

  cout << "\n";
  cout << "Finite Difference: FTBS for 1D Wave Equation:\n";
  cout << "  C++ language\n";
  cout << "\n";
  cout << "  Solve the constant-velocity advection equation in 1D,\n";
  cout << "    du/dt = - c du/dx\n";
  cout << "  over the interval:\n";
  cout << "    0.0 <= x <= 300.0\n";
  cout << "  with a given boundary conditions, and\n";
  cout << "  with a given initial condition\n";
  cout << "    u(0,x) = 80 * sin( (pi/60)* (x-40) ) for 40.0 <= x <= 100.0\n";
  cout << "           = 0 elsewhere.\n";
  cout << "\n";
  cout << "   We use a method known as FTBS:\n";
  cout << "   FT: Forward Time  : du/dt = (u(t+dt,x)-u(t,x))/dt\n";
  cout << "   BS: Backward Space: du/dx = (u(t,x)-u(t,x-dx))/dx\n";
  cout << "\n";

  timestamp ( );

{
  double a;
  double b;
  double c;
  double L;
  double Endtime;
  //string command_filename = "advection_commands.txt";
  //ofstream command_unit;
  //string data_filename = "advection_data.txt";
  //ofstream data_unit;
  string data_filename = "x301_t200.txt";
  ofstream data_unit;
  //ofstream output_t10_x6001;

  double dt; // time-step
  double dx; // size-step
  double nu = 1.48 * 1E-5; // viscosity / density (SI)
  double stab; // stability checker --> Corant Number
  int i;
  int j;
  //int jm1;
  //int jp1;
  int nt = 200; // time steps
  int nx = 301; // mesh numbers
  //int nt_step;
  //int plotstep;
  //double t;
  double u[nt][nx]; // old velocity
  double unew[nt][nx]; // new velocity
  double x[nx];


  L = 300; // Length (m)
  a = 0.0; // x-start
```

```
  b = 300.0; // x-end
  c = 330; // Propagation Velocity
  Endtime = 0.6; // Duration (s)

//*****************************************************************************
  //nx = 6001; // nx: Number of nodes
  dx = L / ( double ) ( nx - 1 ); // mesh size
//*****************************************************************************
  // a loop to initialize the velocity vector.
  for ( i = 0; i < nx; i++ )
  {
      x[i] = x[i] + dx*(i);
      cout << x[i] << " ";
      cout << '\n';
  }
  cout << "x-direction mesh vector, set.\n";
//*****************************************************************************
  // setting boundary conditions:
  for ( i = 0; i < ( ((nx-1)/b) *40 ); i++ )
  {
      u[0][i] = 0;
      unew[0][i] = 0;
      cout << " u[0][" << i <<"] = " << u[0][i] << '\n';
  }
  cout << "B.C. part 1, set.\n";

  for ( i = ( ((nx-1)/b) *40 ); i < ( ((nx-1)/b) *100 ); i++ )
  {
      cout << "x = " << x[i] << '\n';
      u[0][i] =  80 * sin( (pi)*(x[i] - 40) / 60.0 );
      unew[0][i] =  80 * sin( (pi)*(x[i] - 40) / 60.0 );
  }
  cout << "B.C. part 2, set.\n";

  for ( i = (((nx-1)/b) *100 ); i < (nx ); i++ )
  {
      u[0][i] = 0;
      unew[0][i] = 0;
  }
  cout << "B.C. part 3, set.\n";
//*****************************************************************************
  //nt = 10; // Time Resolution
  dt = Endtime / ( double ) ( nt ); // time-step
  cout << "Time resolution, set. \n";
//*****************************************************************************
  // setting initial condition:
  cout << "setting initial condition ... \n";
  for ( i = 0; i < nt; i++ )
  {
      //cout << i << " ";
      u[i][0]=0;
      unew[i][0] = 0;
      //cout << i << "-->" << u[i][0] << " ";
  }
  cout << "Initial condition, set. \n";
//*****************************************************************************
  stab = nu * dt / dx / dx;
  cout << " courant number = " << stab << '\n';
  if ( stab > 0.5 )
  {
    cout << " --> The stability criteria do not match <-- ";
    return 0;
  }
//*****************************************************************************
// ***** Core Calculation: start *****
  //data_unit.open("C:\\Users\\Diamonds\\Documents\\CFD1\\x6001_t20.txt");
  data_unit.open( data_filename.c_str ( ) );
  cout << "file opened.";

  if( !data_unit ) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
      exit(1);
```

```
  }
  //data_unit.open("x6001_t20.txt", ios::app);
  for ( i = 0; i < nt; i++ )
  {
    //cout << "in calc loop \n";
    data_unit << "zone\n";
    //cout << "zone printed. \n";
    //data_unit << fixed << setprecision(2) << x[0] << '\t' << fixed << setprecision(2) << u[i
    ][0] << '\n';
    //data_unit << '\t' << x[0] << '\t' << i << '\t' << u[i][0] << '\n';
    data_unit << x[0] << '\t' << u[i][0] << '\n';
    //fprintf(fptr, "%f\t%f\n", x[0], u[i][0]);
    // First do this for all x in xn:
    for ( j = 1; j < nx; j++ )
    {
        unew[i+1][j] = u[i][j] - (c * dt / dx) * ( u[i][j] - u[i][j-1] );
        //cout << "u_new = " << unew[i+1][j] << '\n';
        u[i+1][j] = unew[i+1][j];
        //fprintf(fptr, "%f\t%f\n", x[j], u[i][j]);
        //data_unit << fixed << setprecision(2) << x[j] << '\t' << fixed << setprecision(2) <<
     u[i][j] << '\n';
        //data_unit << '\t' << x[j] << '\t' << i << '\t' << u[i][j] << '\n';
        data_unit << x[j] << '\t' << u[i][j] << '\n';
        //cout << "printed: x = " << x[j] << '\t' << "u = " << u[i][j] << '\n';
    }
    data_unit << "\n";
  }
  cout << "Calculation, done.\n";
  cout << " courant number = " << stab << '\n';
  data_unit.close();

// ***** Core Calculation: end  *****

//  Free memory.
//
//
//  Terminate.
//
  cout << "\n";
  cout << "  Finite Difference: FTBS for 1D Wave Equation\n";
  cout << "  Array-based Algorithm\n";
  cout << "  Normal end of execution.\n";
  cout << "\n";

  return 0;
}
//****************************************************************************
void timestamp ( )
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct std::tm *tm_ptr;
  size_t len;
  std::time_t now;

  now = std::time ( NULL );
  tm_ptr = std::localtime ( &now );

  len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );

  std::cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}
%*************************************************************************
```

6.4 Code for The FTBS Method – Pointer-based

```cpp
%*****************************************************************
# include <cstdlib>
# include <iostream>
# include <fstream>
# include <iomanip>
# include <cmath>
# include <ctime>
# include <cstring>

using namespace std;

int main ( );
int i4_modp ( int i, int j );
int i4_wrap ( int ival, int ilo, int ihi );
double *initial_condition ( int nx, double x[] );
double *r8vec_linspace_new ( int n, double a, double b );
void timestamp ( );

//****************************************************************************

int main ( )

//****************************************************************************
//
// Language: C++
//  Author: Dina Soltani Tehrani.
//  Course: Advanced CFD1 - by Dr. Taeibi Rahni
//  Sharif University of Technology
//
//  Equation: ut = -c * ux
//  Method: Finite difference, FTBS Method, Explicit
//  Pointer-based Method
//
//    The finite difference form is as bellow:
//
//      U(X,T+dt) - U(X,T)          ( U(X-dx,T+dt) + U(X+dx,T+dt) )
//      ------------------  = -c * ------------------------------- + F(X, T+dt)
//               dt                              2 * dx
//
{
  double a;
  double b;
  double c;
  double L;
  double duration;
  double  pi = 4*atan(1.0);
  string data_filename = "FTBS_x1201_t1000_p.txt";
  ofstream data_unit;
  double dt; // time-step
  double dx; // size-step
  int i;
  int j;
  int jm1;
  int jp1;
  int nx;
  int nt;
  int nt_step;
  int plotstep;
  double t;
  double *u; // old velocity
  double *unew; // new velocity
  double *x;

  cout << "\n";
  cout << "Finite Difference: FTBS for 1D Wave Equation:\n";
  cout << "  C++ language\n";
  cout << "\n";
  cout << "  Solve the constant-velocity advection equation in 1D,\n";
  cout << "    du/dt = - c du/dx\n";
  cout << "  over the interval:\n";
  cout << "    0.0 <= x <= 300.0\n";
  cout << "  with a given boundary conditions, and\n";
```

```
  cout << "  with a given initial condition\n";
  cout << "    u(0,x) = 80 * sin( (pi/60)* (x-40) ) for 40.0 <= x <= 100.0\n";
  cout << "           = 0 elsewhere.\n";
  cout << "\n";
  cout << "  We use a method known as FTBS:\n";
  cout << "   FT: Forward Time  : du/dt = (u(t+dt,x)-u(t,x))/dt\n";
  cout << "   BS: Backward Space: du/dx = (u(t,x)-u(t,x-dx))/dx\n";
  cout << "\n";

  timestamp ( );

  L = 300; // Length (m)
  nx = 1201; // Mid Space Resolution
  dx = L / ( double ) ( nx - 1 ); // size-step
  a = 0.0; // x-start
  b = 300.0; // x-end
  x = r8vec_linspace_new ( nx, a, b );
  nt = 1000; // Time Resolution
  duration = 0.6; // Duration (s)
  dt = duration / ( double ) ( nt ); // time-step
  c = 330.0;

  u = initial_condition ( nx, x ); // Initializing old velocity with initial condition.

//  ***** Opening data file , and writing solutions as they are computed. *****

  data_unit.open ( data_filename.c_str ( ) );

  // Writing the initial data.
  t = 0.0;
  data_unit << x[0]
            << "  " << u[0] << "\n";

  // Writing data as we move forward in space.
  for ( j = 0; j < nx; j++ )
  {
    data_unit << x[j]
              << "  " << u[j] << "\n";
  }
  data_unit << "zone";
  data_unit << "\n";



//
    *********************************************************************************************
  nt_step = 100;
  unew = new double[nx]; // Creating the new velocity vector

// ***** Core Calculation: begin *****
  for ( i = 0; i < nt; i++ )
  {
    //data_unit << "zone\n";
    // First do this for all x in xn:
    for ( j = 0; j < nx; j++ )
    {
      jm1 = i4_wrap ( j - 1, 0, nx - 1 ); // phi_n_j-1
      //jp1 = i4_wrap ( j + 1, 0, nx - 1 ); // phi_n_j+1
      unew[j] = u[j] - (c * dt / dx) * ( u[j] - u[jm1] );
      //if ( u[j] < 1e-04 ) { u[j] = 0; }
    }
    // Then, do this, again for all x in xn:
    for ( j = 0; j < nx; j++ )
    {
      u[j] = unew[j];
      //data_unit << x[j] << "  " << u[j] << "\n";
    }

    if ( i == nt_step - 1 )
    {
      t = ( double ) ( i ) * dt;
```

```
      for ( j = 0; j < nx; j++ )
      {
        //if ( u[j] < 1e-05 ) { u[j] = 0; }
        data_unit << x[j]
                  << "  " << u[j] << "\n";
      }
      data_unit << "zone";
      data_unit << "\n";
      nt_step = nt_step + 100;
    }

    //data_unit << "zone";
    //data_unit << "\n";
  }
// ***** Core Calculation: end *****


//
//  ***** Closing the data file as the computation is done. *****
//
  data_unit.close ( );

  cout << "\n";
  cout << "  Plot data written to the file \"" << data_filename << "\"\n";
//

//  Free memory.
//
  delete [] u;
  delete [] unew;
  delete [] x;
//
//  Terminate.
//
  cout << "\n";
  cout << "  Finite Difference: FTBS for 1D Wave Equation\n";
  cout << "  Pointer-based Method\n";
  cout << "  Normal end of execution.\n";
  cout << "\n";
  timestamp ( );

  return 0;
}

//
// ***** Functions *****

//****************************************************************************
int i4_modp ( int i, int j )
{
  int value;

  if ( j == 0 )
  {
    cerr << "\n";
    cerr << "I4_MODP - Fatal error!\n";
    cerr << "  I4_MODP ( I, J ) called with J = " << j << "\n";
    exit ( 1 );
  }

  value = i % j;

  if ( value < 0 )
  {
    value = value + abs ( j );
  }

  return value;
}

//****************************************************************************
int i4_wrap ( int ival, int ilo, int ihi )
```

```
{
  int jhi;
  int jlo;
  int value;
  int wide;

  if ( ilo <= ihi )
  {
    jlo = ilo;
    jhi = ihi;
  }
  else
  {
    jlo = ihi;
    jhi = ilo;
  }

  wide = jhi + 1 - jlo;

  if ( wide == 1 )
  {
    value = jlo;
  }
  else
  {
    value = jlo + i4_modp ( ival - jlo, wide );
  }

  return value;
}

//****************************************************************************80
double *initial_condition ( int nx, double x[] )
{
  int i;
  double *u;
  double  pi = 4*atan(1.0);

  u = new double[nx];

  for ( i = 0; i < nx; i++ )
  {
    if  ( 40 <= x[i] && x[i] <= 100 )
    {
      u[i] = 80 * sin( (pi/60)*(x[i] - 40) );
    }
    else
    {
      u[i] = 0.0;
    }
  }
  return u;
}

//****************************************************************************80
double *r8vec_linspace_new ( int n, double a_first, double a_last )
{
  double *a;
  int i;

  a = new double[n];

  if ( n == 1 )
  {
    a[0] = ( a_first + a_last ) / 2.0;
  }
  else
  {
    for ( i = 0; i < n; i++ )
    {
      a[i] = ( ( double ) ( n - 1 - i ) * a_first
             + ( double ) (         i ) * a_last )
```

```
              / ( double ) ( n - 1      );
    }
  }
  return a;
}

//****************************************************************************
void timestamp ( )
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct std::tm *tm_ptr;
  size_t len;
  std::time_t now;

  now = std::time ( NULL );
  tm_ptr = std::localtime ( &now );

  len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );

  std::cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}
%****************************************************************
```