

Graduate CFD-I: Project Report #2

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

Sharif University of Technology — January 6, 2022

Contents

1	Problem Description and Governing Equations	2
1.1	Objectives	2
1.2	Initial Condition	2
1.3	Boundary Condition	2
2	Exact Solution	2
3	Numerical Methods	3
3.1	Introduction	3
3.2	Euler Implicit Method	3
3.3	Lax Method	3
3.4	FTBS Method	4
4	Results and Discussion	4
4.1	Euler Implicit Method	4
4.2	Lax Method	4
4.3	FTBS Method	4
5	References	6
6	Appendices	9
6.1	Code for The Euler Implicit Method	9
6.2	Code for The Lax Method	16
6.3	Code for The FTBS Method – Array-based	20
6.4	Code for The FTBS Method – Pointer-based	23

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 :

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

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:

$$\frac{\partial u}{\partial t} - c \frac{\partial u}{\partial x} = 0 \quad (2)$$

$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = 0 \quad (3)$$

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

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

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](#), 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:

$$u(0, t) = 0 \quad (5)$$

1.3 Boundary Condition

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

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

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:

(7)

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

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):

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

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:

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

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):

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

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:

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

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):

$$\frac{u_i^{n+1} - u_i^n}{dt} + C \frac{u_i^{n+1} - u_{i-1}^{n+1}}{dx} = 0 \quad (12)$$

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 and unconditionally unstable and is implemented based on the following difference equation:

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

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 referrals 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.

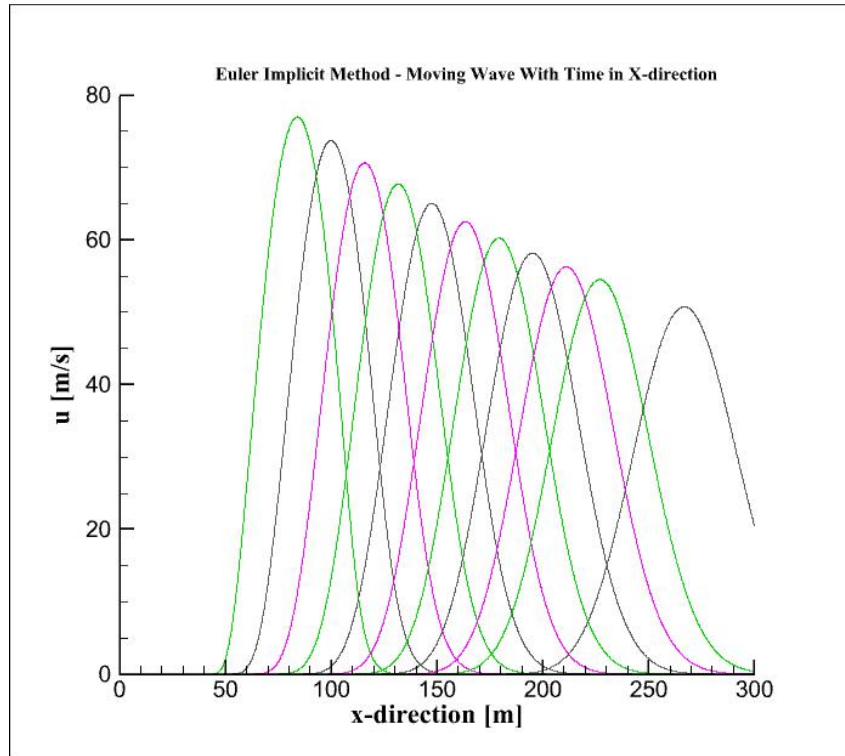


Figure 1: Euler Implicit Method - Number of Mesh Nodes: 2401 - Number of Time Steps: 100

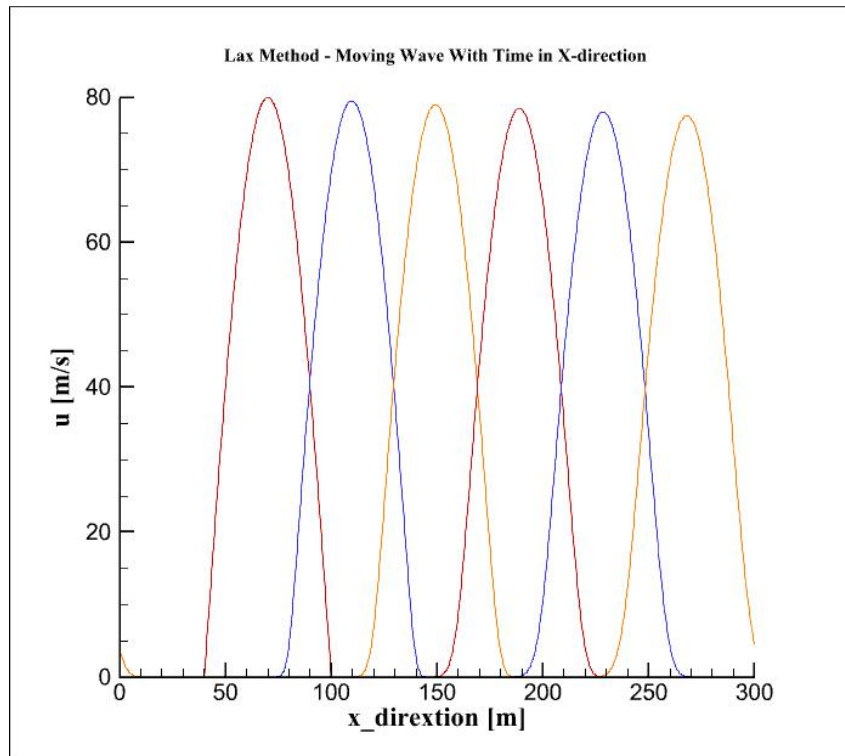


Figure 2: Lax Implicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 1000

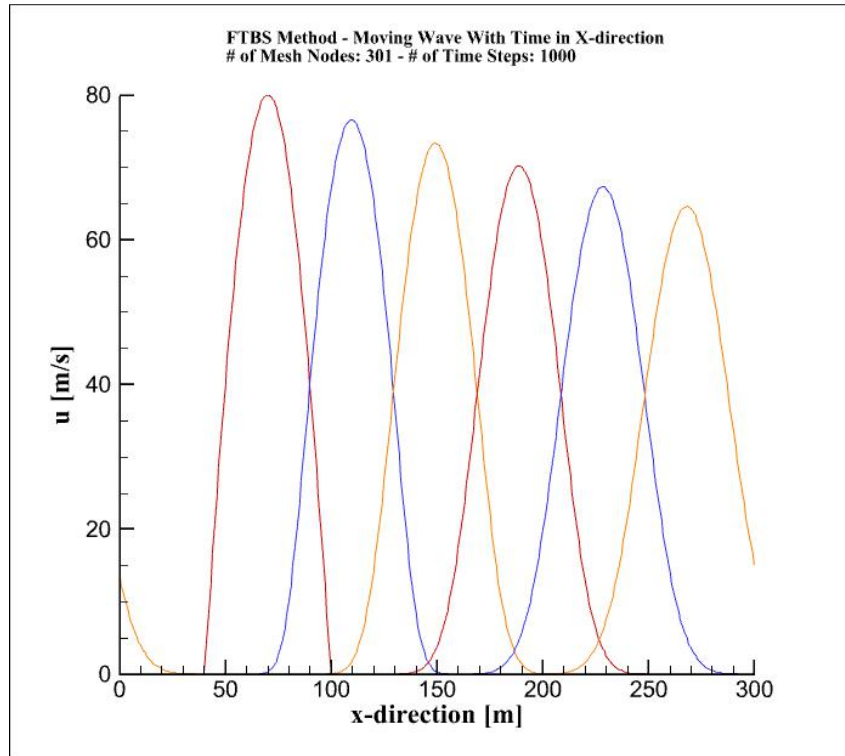


Figure 3: FTBS Explicit Method - Number of Mesh Nodes: 301 - Number of Time Steps: 1000

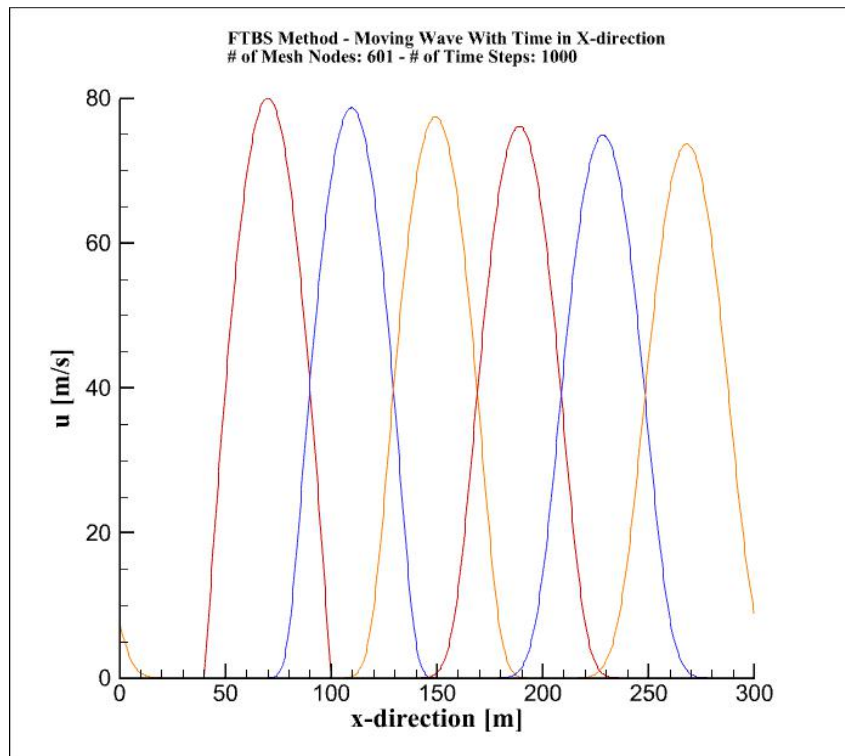


Figure 4: FTBS Explicit Method - Number of Mesh Nodes: 601 - Number of Time Steps: 1000

5 References

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

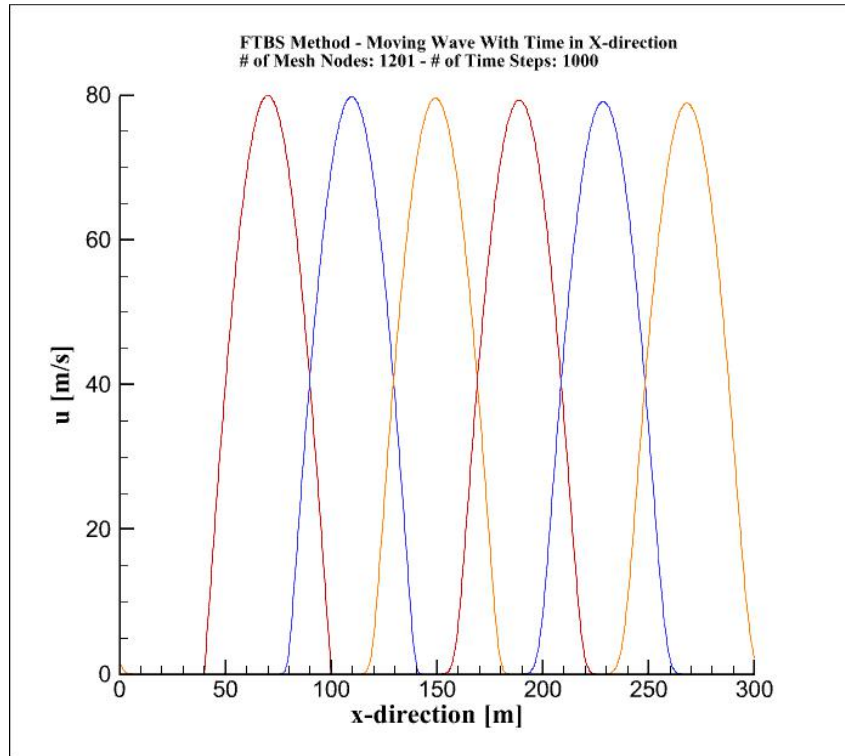


Figure 5: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 1000

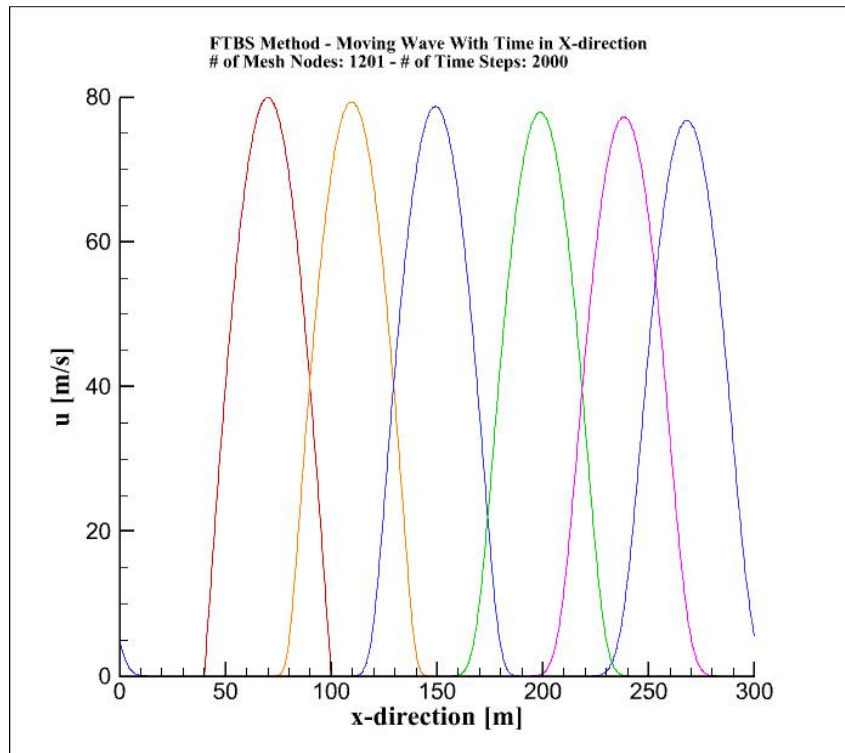


Figure 6: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 2000

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

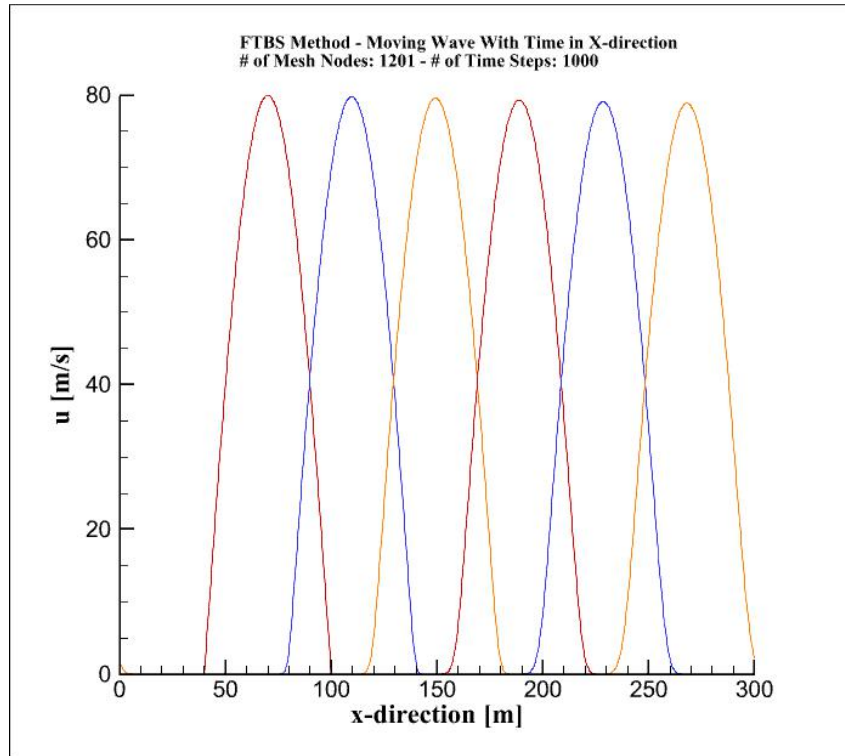


Figure 7: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 1000

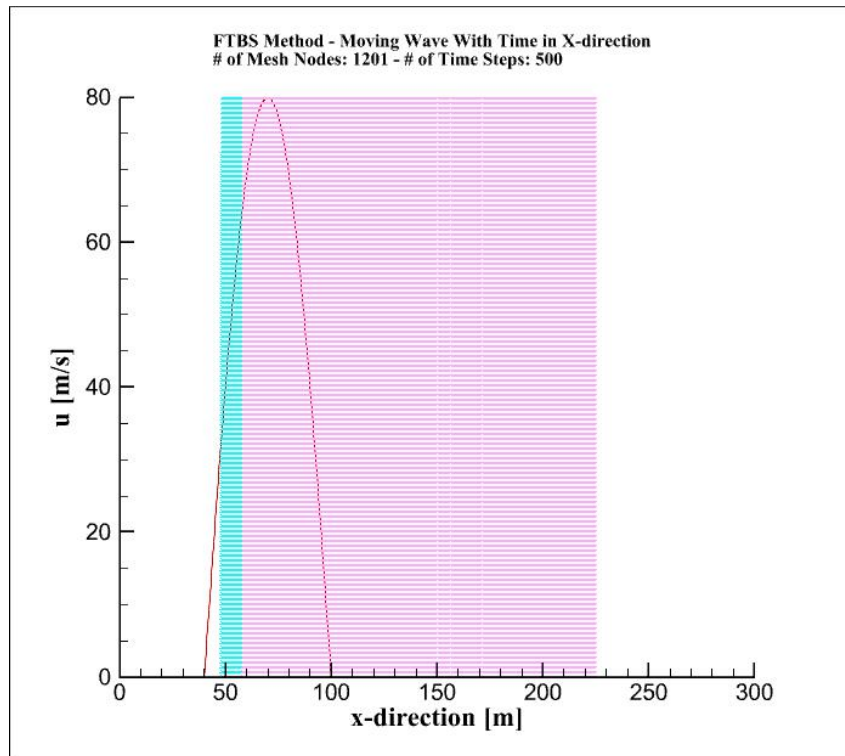


Figure 8: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 500 - Zoom out

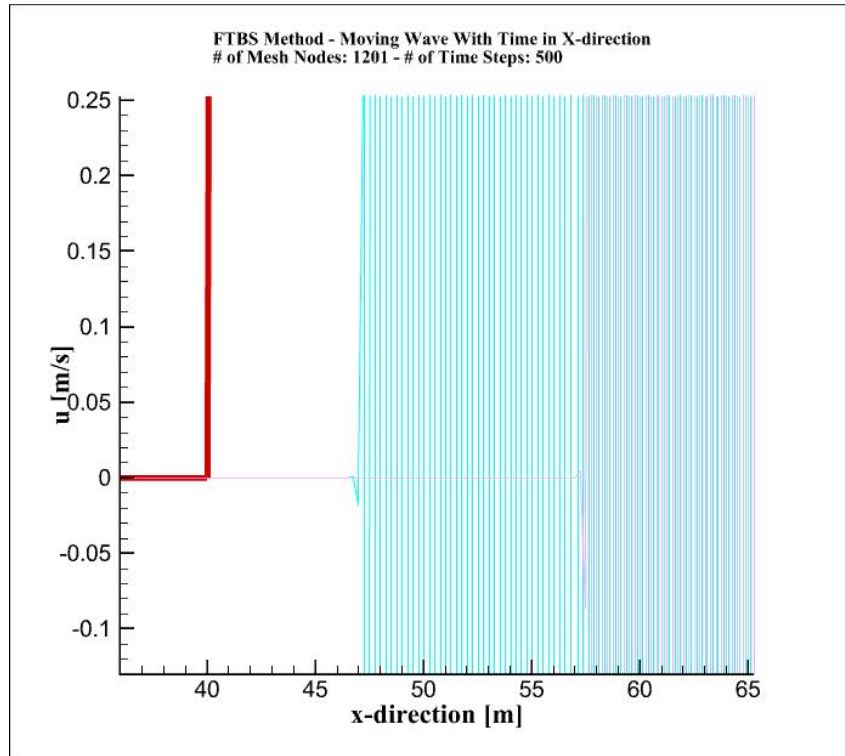


Figure 9: FTBS Explicit Method - Number of Mesh Nodes: 1201 - Number of Time Steps: 500 - Zoom in

6 Appendices

6.1 Code for The Euler Implicit Method

```

1 %*****
2 # include <cstdlib>
3 # include <iostream>
4 # include <iomanip>
5 # include <fstream>
6 # include <ctime>
7 # include <cmath>
8
9 // Language: C++
10 // Author: Dina Soltani Tehrani.
11 // Course: Advanced CFD1 – by Dr. Taeibi Rahni
12 // Sharif University of Technology
13 //
14 // Equation:  $u_t = -c * u_x$ 
15 // Method: Finite difference , FTCS, Explicit
16 //
17 // The finite difference form is as bellow:
18 //
19 // 
$$\frac{U(X,T+dt) - U(X,T)}{dt} = -c * \frac{(U(X-dx,T+dt) + U(X+dx,T+dt))}{2 * dx} + F(X, T+dt)$$

20 //
21 //
22 //
23 // 
$$+ \quad c * dt / dx / 2 \quad * \quad U(X+dx,T+dt)$$

24 // 
$$+ \quad 1 \quad * \quad U(X, T+dt)$$

25 // 
$$- \quad c * dt / dx / 2 \quad * \quad U(X-dx,T+dt)$$

26 // 
$$= \quad U(X, T)$$

27 // 
$$+ \quad dt \quad * \quad F(X, T+dt)$$

28
29 using namespace std;
30 //*****
31 // Functions
32
33 int main ( );

```

```

34 void dtable_data_write ( ofstream &output, int m, int n, double table[] );
35
36 void dtable_write ( string output_filename, int m, int n, double table[],
37     bool header );
38
39 void f ( double a, double b, double t0, double t, int n, double x[],
40     double value[] );
41
42 int r83_np_fa ( int n, double a[] );
43
44 double *r83_np_sl ( int n, double a_lu[], double b[], int job );
45
46 void timestamp ( );
47
48 void u0 ( double a, double b, double t0, int n, double x[], double value[] );
49
50 double ua ( double a, double b, double t0, double t );
51 double ub ( double a, double b, double t0, double t );
52
53 // *****
54
55 int main ( )
56 // *****
57 {
58     double *a;
59     double *b;
60     double *fvec;
61     bool header;
62     int i;
63     int j;
64     int job;
65     double k;
66     double *t;
67     double dt;
68     string t_file;
69     double t_max;
70     double t_min;
71     int nt;
72     double *u;
73     string u_file;
74     double w;
75     double *x;
76     double dx;
77     string x_file;
78     double x_max;
79     double x_min;
80     int nx;
81     string final_file;
82     double c; // wave speed
83     timestamp ( );
84
85     // nu= 5.0E-07;
86     c = 330.0;
87     // *****
88     // Set X values.
89     x_min = 0.0;
90     x_max = 300.0;
91     nx = 2401;
92     dx = ( x_max - x_min ) / ( double ) ( nx - 1 );
93
94     x = new double[nx];
95
96     for ( i = 0; i < nx; i++ )
97     {
98         x[i] = ( ( double ) ( nx - i - 1 ) * x_min
99             + ( double ) ( i ) * x_max )
100             / ( double ) ( nx - 1 );
101     }
102
103     // *****
104     // Set Time values.
105     t_min = 0.0;

```

```

106 t_max = 0.6;
107 nt = 100;
108 dt = ( t_max - t_min ) / ( double ) ( nt - 1 );
109
110 t = new double[nt];
111
112 for ( j = 0; j < nt; j++ )
113 {
114     t[j] = ( ( double ) ( nt - j - 1 ) * t_min
115             + ( double ) ( j ) * t_max )
116           / ( double ) ( nt - 1 );
117 }
118
119 // *****
120 // Set the initial data, for time T_MIN.
121
122 u = new double[nx*nt];
123
124 // returning the initial condition at the starting time:
125 u0 ( x_min, x_max, t_min, nx, x, u );
126
127 // *****
128 // Defining the matrix A. This matrix is constant and does not change with time.
129 // We can set it once, factor it once, and solve repeatedly.
130
131 w = c * dt / dx / 2; // w = stab as the stability checker.
132
133 a = new double[3*nx];
134
135 a[0+0*3] = 0.0;
136 a[1+0*3] = 1.0;
137 a[0+1*3] = 0.0;
138
139 for ( i = 1; i < nx - 1; i++ )
140 {
141     a[2+(i-1)*3] = - w;
142     a[1+ i *3] = 1.0;
143     a[0+(i+1)*3] = + w;
144 }
145
146 a[2+(nx-2)*3] = -2*w;
147 a[1+(nx-1)*3] = 1.0 + 2*w;
148 a[2+(nx-1)*3] = 0.0;
149
150 // Factor the matrix.
151
152 // factoring the R83 system without pivoting:
153 r83_np_fa ( nx, a );
154
155 // *****
156 // The right-hand side of the system of equation:
157
158 b = new double[nx];
159 fvec = new double[nx];
160
161 for ( j = 1; j < nt; j++ )
162 {
163
164     // Set the right hand side B.
165     //
166     b[0] = ua ( x_min, x_max, t_min, t[j] ); // returns the Dirichlet boundary condition at
167     the left endpoint.
168
169     // returning the right hand side of the 1d wave equation:
170     // for this case, the excitation term is set to be zero;
171
172     f ( x_min, x_max, t_min, t[j-1], nx, x, fvec );
173
174     for ( i = 1; i < nx; i++ )
175     {
176         b[i] = u[i+(j-1)*nx] + dt * fvec[i];
177     }
178 }

```

```

177 //b[nx-1] = ub ( x_min, x_max, t_min, t[j] ); //returns the Dirichlet boundary condition
178 at the right endpoint.
179
180 delete [] fvec;
181
182 job = 0;
183 // solveing the R83 system factored by R83_NP_FA:
184 fvec = r83_np_sl ( nx, a, b, job );
185
186 for ( i = 0; i < nx; i++ )
187 {
188     u[i+j*nx] = fvec[i];
189 }
190 }
191 //*****
192 // data handling:
193
194 x_file = "x.txt";
195 header = false;
196 dtable_write ( x_file , 1, nx, x, header ); // writes information to a DTABLE file.
197
198 cout << "\n";
199 cout << " X data written to \" << x_file << "\".\n";
200
201 t_file = "t.txt";
202 header = false;
203 dtable_write ( t_file , 1, nt, t, header ); // writes information to a DTABLE file.
204
205 cout << " T data written to \" << t_file << "\".\n";
206
207 u_file = "u.txt";
208 header = false;
209 dtable_write ( u_file , nx, nt, u, header ); // writes information to a DTABLE file.
210
211 final_file = "EuImp_x2401_t100_ed2.txt";
212 header = false;
213 dtable_write ( final_file , nx, nt, u, header ); // writes information to a DTABLE file.
214
215 cout << " Final data written to \" << final_file << "\".\n";
216
217 delete [] a;
218 delete [] b;
219 delete [] fvec;
220 delete [] t;
221 delete [] u;
222 delete [] x;
223 //
224 // Terminate.
225 //
226 cout << "\n";
227 cout << " Finite Difference: Euler Implicit for 1D Wave Equation\n";
228 cout << " Normal end of execution.\n";
229 cout << "\n";
230 timestamp ( );
231
232 return 0;
233 }
234 //*****
235
236 void dtable_data_write ( ofstream &output, int nx, int nt, double table[] )
237 {
238     int i;
239     int j;
240     //double *vect;
241     double x_min = 0;
242     double x_max = 300;
243     double t_min = 0.0;
244     double t_max = 0.6;
245     double dx;
246     //double dt;

```

```

248 double vect[nt][nx];
249 dx = ( x_max - x_min ) / ( double ) ( nx - 1 );
250 //dt = ( t_max - t_min ) / ( double ) ( nt - 1 );
251
252 for ( j = 0; j < nt; j++ )
253 {
254     output << "zone\n";
255     for ( i = 0; i < nx; i++ )
256     {
257         vect[j][i] = table[i+j*nx];
258         output << x_min + i*dx << '\t' << vect[j][i] << '\n';
259         //output << setw(10) << table[i+j*dx] << " ";
260     }
261     output << "\n";
262 }
263
264 return;
265 }
266 // *****
267
268 void dtable_write ( string output_filename, int m, int n, double table[],
269 bool header )
270 {
271     ofstream output;
272
273     output.open ( output_filename.c_str ( ) );
274
275     if ( !output )
276     {
277         cerr << "\n";
278         cerr << "DTABLE_WRITE - Fatal error!\n";
279         cerr << " Could not open the output file.\n";
280         return;
281     }
282
283     if ( header )
284     {
285         // dtable_header_write ( output_filename, output, m, n );
286     }
287
288     dtable_data_write ( output, m, n, table );
289
290     output.close ( );
291
292     return;
293 }
294 // *****
295
296 void f ( double a, double b, double t0, double t, int n, double x[],
297 double value[] )
298 {
299     {
300         int i;
301
302         for ( i = 0; i < n; i++ )
303         {
304             value[i] = 0.0;
305         }
306         return;
307     }
308 }
309 // *****
310
311 int r83_np_fa ( int n, double a[] )
312 {
313     int i;
314
315     for ( i = 1; i <= n-1; i++ )
316     {
317         if ( a[1+(i-1)*3] == 0.0 )
318         {
319

```

```

320     cout << "\n";
321     cout << "R83_NP_FA - Fatal error!\n";
322     cout << " Zero pivot on step " << i << "\n";
323     return i;
324 }
325 //
326 // Store the multiplier in L.
327 //
328 a[2+(i-1)*3] = a[2+(i-1)*3] / a[1+(i-1)*3];
329 //
330 // Modify the diagonal entry in the next column.
331 //
332 a[1+i*3] = a[1+i*3] - a[2+(i-1)*3] * a[0+i*3];
333 }
334
335 if ( a[1+(n-1)*3] == 0.0 )
336 {
337     cout << "\n";
338     cout << "R83_NP_FA - Fatal error!\n";
339     cout << " Zero pivot on step " << n << "\n";
340     return n;
341 }
342
343 return 0;
344 }
345 // *****
346
347 double *r83_np_sl ( int n, double a_lu[], double b[], int job )
348 {
349     int i;
350     double *x;
351
352     x = new double[n];
353
354     for ( i = 0; i < n; i++ )
355     {
356         x[i] = b[i];
357     }
358
359     if ( job == 0 )
360     {
361         //
362         // Solve L * Y = B.
363         //
364         for ( i = 1; i < n; i++ )
365         {
366             x[i] = x[i] - a_lu[2+(i-1)*3] * x[i-1];
367         }
368         //
369         // Solve U * X = Y.
370         //
371         for ( i = n; 1 <= i; i-- )
372         {
373             x[i-1] = x[i-1] / a_lu[1+(i-1)*3];
374             if ( 1 < i )
375             {
376                 x[i-2] = x[i-2] - a_lu[0+(i-1)*3] * x[i-1];
377             }
378         }
379     }
380     else
381     {
382         //
383         // Solve U' * Y = B
384         //
385         for ( i = 1; i <= n; i++ )
386         {
387             x[i-1] = x[i-1] / a_lu[1+(i-1)*3];
388             if ( i < n )
389             {
390                 x[i] = x[i] - a_lu[0+i*3] * x[i-1];
391             }

```

```

392     }
393 }
394 //
395 // Solve L' * X = Y.
396 //
397 for ( i = n-1; 1 <= i; i-- )
398 {
399     x[i-1] = x[i-1] - a_lu[2+(i-1)*3] * x[i];
400 }
401 }
402
403 return x;
404 }
405 // *****
406
407 void timestamp ( )
408 {
409 # define TIME_SIZE 40
410
411 static char time_buffer[TIME_SIZE];
412 const struct tm *tm;
413 time_t now;
414
415 now = time ( NULL );
416 tm = localtime ( &now );
417
418 strftime ( time_buffer, TIME_SIZE, "%d %B %Y %l:%M:%S %p", tm );
419
420 cout << time_buffer << "\n";
421
422 return;
423 # undef TIME_SIZE
424 }
425 // *****
426
427 void u0 ( double a, double b, double t0, int nx, double x[], double value[] )
428 {
429
430 {
431     int i;
432     double pi = 4*atan(1.0);
433
434     for ( i = 0; i < nx; i++ )
435     {
436         if ( 40 <= x[i] && x[i] <= 100 )
437         {
438             value[i] = 80 * sin( (pi/60)*(x[i] - 40) );
439         }
440         else
441         {
442             value[i] = 0.0;
443         }
444     }
445
446     return;
447 }
448 // *****
449
450 double ua ( double a, double b, double t0, double t )
451 {
452     double value;
453
454     value = 0;
455
456     return value;
457 }
458 // *****
459
460 double ub ( double a, double b, double t0, double t )
461 {
462
463 {

```

```

464     double value;
465
466     value = 2.14618;
467
468     return value;
469 }
470
471 %*****

```

6.2 Code for The Lax Method

```

1  %*****
2  # include <cstdlib>
3  # include <iostream>
4  # include <fstream>
5  # include <iomanip>
6  # include <cmath>
7  # include <ctime>
8  # include <cstring>
9
10 using namespace std;
11
12 int main ( );
13 int i4_modp ( int i, int j );
14 int i4_wrap ( int ival, int ilo, int ihi );
15 double *initial_condition ( int nx, double x[] );
16 double *r8vec_linspace_new ( int n, double a, double b );
17 void timestamp ( );
18
19 // *****
20
21 int main ( )
22
23 // *****
24 //
25 // Language: C++
26 // Author: Dina Soltani Tehrani.
27 // Course: Advanced CFD1 – by Dr. Taeibi Rahni
28 // Sharif University of Technology
29 //
30 // Equation:  $u_t = -c * u_x$ 
31 // Method: Finite difference, Lax Method, Explicit
32 //
33 // The finite difference form is as bellow:
34 //
35 // 
$$\frac{U(X,T+dt) - U(X,T)}{dt} = -c * \frac{(U(X-dx,T+dt) + U(X+dx,T+dt))}{2 * dx} + F(X, T+dt)$$

36 //
37 //
38 //
39
40 {
41     double a;
42     double b;
43     double c;
44     double L;
45     double duration;
46     double pi = 4*atan(1.0);
47     string data_filename = "Lax_x1201_t1000.txt";
48     ofstream data_unit;
49     double dt; // time-step
50     double dx; // size-step
51     int i;
52     int j;
53     int jml;
54     int jpl;
55     int nx;
56     int nt;
57     int nt_step;
58     int plotstep;
59     double t;

```



```

60 double *u; // old velocity
61 double *unew; // new velocity
62 double *x;
63
64 cout << "\n";
65 cout << "Finite Difference: Lax Method for 1D Wave Equation:\n";
66 cout << "  C++ language\n";
67 cout << "\n";
68 cout << "  Solve the constant-velocity advection equation in 1D,\n";
69 cout << "    du/dt = - c du/dx\n";
70 cout << "  over the interval:\n";
71 cout << "    0.0 <= x <= 300.0\n";
72 cout << "  with a given boundary conditions, and\n";
73 cout << "  with a given initial condition\n";
74 cout << "    u(0,x) = 80 * sin( (pi/60)* (x-40) ) for 40.0 <= x <= 100.0\n";
75 cout << "    = 0 elsewhere.\n";
76 cout << "\n";
77 cout << "  We use a method known as Lax:\n";
78 cout << "    FT: Forward Time : du/dt = (u(t+dt,x)-u(t,x))/dt\n";
79 cout << "    BS: Backward Space: du/dx = (u(t,x)-u(t,x-dx))/dx\n";
80 cout << "\n";
81 timestamp ( );
82
83 L = 300; // Length (m)
84 nx = 1201; // Mid Space Resolution
85 dx = L / ( double ) ( nx - 1 ); // size-step
86 a = 0.0; // x-start
87 b = 300.0; // x-end
88 x = r8vec_linspace_new ( nx, a, b );
89 nt = 1000; // Time Resolution
90 duration = 0.6; // Duration (s)
91 dt = duration / ( double ) ( nt ); // time-step
92 c = 330.0;
93
94 u = initial_condition ( nx, x ); // Initializing old velocity with initial condition.
95
96 // ***** Opening data file , and writing solutions as they are computed. *****
97
98 data_unit.open ( data_filename.c_str ( ) );
99
100 // Writing the initial data.
101 t = 0.0;
102 data_unit << x[0]
103     << " " << u[0] << "\n";
104
105 // Writing data as we move forward in space.
106 for ( j = 0; j < nx; j++ )
107 {
108     data_unit << x[j]
109         << " " << u[j] << "\n";
110 }
111 data_unit << "zone";
112 data_unit << "\n";
113
114
115
116 //
117     *****
118
119 nt_step = 100;
120 unew = new double[nx]; // Creating the new velocity vector
121
122 // ***** Core Calculation: begin *****
123 for ( i = 0; i < nt; i++ )
124 {
125     //data_unit << "zone\n";
126     // First do this for all x in xn:
127     for ( j = 0; j < nx; j++ )
128     {
129         jml = i4_wrap ( j - 1, 0, nx - 1 ); // phi_nj-1
130         jp1 = i4_wrap ( j + 1, 0, nx - 1 ); // phi_nj+1
131         if ( j != nx-1)

```

```

130     {
131         unew[j] = (0.5 * ( 1 - (c * dt / dx) ) ) * u[jp1] + (0.5 * ( 1 + (c * dt / dx) ) ) *
u[jm1];
132         //if ( u[j] < 1e-04 ) { u[j] = 0; }
133     }
134     else
135     {
136         unew[j] = u[j] - (c * dt / dx) * ( u[j] - u[jm1] );
137     }
138 }
139
140 // Then, do this, again for all x in xn:
141 for ( j = 0; j < nx; j++ )
142 {
143     u[j] = unew[j];
144     //data_unit << x[j] << " " << u[j] << "\n";
145 }
146
147 if ( i == nt_step - 1 )
148 {
149     t = ( double ) ( i ) * dt;
150     for ( j = 0; j < nx; j++ )
151     {
152         //if ( u[j] < 1e-05 ) { u[j] = 0; }
153         data_unit << x[j]
154             << " " << u[j] << "\n";
155     }
156     data_unit << "zone";
157     data_unit << "\n";
158     nt_step = nt_step + 100;
159 }
160
161 //data_unit << "zone";
162 //data_unit << "\n";
163 }
164 // ***** Core Calculation: end *****
165
166 //
167 // ***** Closing the data file as the computation is done. *****
168 //
169 data_unit.close ( );
170
171 cout << "\n";
172 cout << " Plot data written to the file \" " << data_filename << "\"\n";
173 //
174 //
175 // Free memory.
176 //
177 delete [] u;
178 delete [] unew;
179 delete [] x;
180 //
181 // Terminate.
182 //
183 cout << "\n";
184 cout << " Finite Difference: Lax for 1D Wave Equation\n";
185 cout << " Normal end of execution.\n";
186 cout << "\n";
187 timestamp ( );
188
189 return 0;
190 }
191
192 //
193 // ***** Functions *****
194
195 //*****
196 int i4_modp ( int i, int j )
197 {
198     int value;
199
200

```

```

201     if ( j == 0 )
202     {
203         cerr << "\n";
204         cerr << "I4_MODP - Fatal error!\n";
205         cerr << " I4_MODP ( I, J ) called with J = " << j << "\n";
206         exit ( 1 );
207     }
208
209     value = i % j;
210
211     if ( value < 0 )
212     {
213         value = value + abs ( j );
214     }
215
216     return value;
217 }
218
219 // *****
220 int i4_wrap ( int ival, int ilo, int ihi )
221 {
222     int jhi;
223     int jlo;
224     int value;
225     int wide;
226
227     if ( ilo <= ihi )
228     {
229         jlo = ilo;
230         jhi = ihi;
231     }
232     else
233     {
234         jlo = ihi;
235         jhi = ilo;
236     }
237
238     wide = jhi + 1 - jlo;
239
240     if ( wide == 1 )
241     {
242         value = jlo;
243     }
244     else
245     {
246         value = jlo + i4_modp ( ival - jlo, wide );
247     }
248
249     return value;
250 }
251
252 // *****
253 double *initial_condition ( int nx, double x[] )
254 {
255     int i;
256     double *u;
257     double pi = 4*atan(1.0);
258
259     u = new double[nx];
260
261     for ( i = 0; i < nx; i++ )
262     {
263         if ( 40 <= x[i] && x[i] <= 100 )
264         {
265             u[i] = 80 * sin( (pi/60)*(x[i] - 40) );
266         }
267         else
268         {
269             u[i] = 0.0;
270         }
271     }
272     return u;

```

```

273 }
274
275 // *****
276 double *r8vec_linspace_new ( int n, double a_first, double a_last )
277 {
278     double *a;
279     int i;
280
281     a = new double[n];
282
283     if ( n == 1 )
284     {
285         a[0] = ( a_first + a_last ) / 2.0;
286     }
287     else
288     {
289         for ( i = 0; i < n; i++ )
290         {
291             a[i] = ( ( double ) ( n - 1 - i ) * a_first
292                     + ( double ) ( i ) * a_last )
293                   / ( double ) ( n - 1 );
294         }
295     }
296     return a;
297 }
298
299 // *****
300 void timestamp ( )
301 {
302     # define TIME_SIZE 40
303
304     static char time_buffer[TIME_SIZE];
305     const struct std::tm *tm_ptr;
306     size_t len;
307     std::time_t now;
308
309     now = std::time ( NULL );
310     tm_ptr = std::localtime ( &now );
311
312     len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );
313
314     std::cout << time_buffer << "\n";
315
316     return;
317     # undef TIME_SIZE
318 }
319
320 %*****

```

6.3 Code for The FTBS Method – Array-based

```

1 %*****
2 # include <cstdlib>
3 # include <iostream>
4 # include <fstream>
5 # include <iomanip>
6 # include <cmath>
7 # include <ctime>
8 # include <cstring>
9 # include <stdio.h>      /* printf */
10 # include <math.h>      /* sin */
11 const double pi = 3.14159265359;
12 using namespace std;
13 using std::ofstream;
14 using std::cerr;
15 using std::endl;
16
17 //
18 // Language: C++
19 // Author: Dina Soltani Tehrani.

```

```

20 // Course: Advanced CFD1 – by Dr. Taeibi Rahni
21 // Sharif University of Technology
22 //
23 // Equation:  $u_t = -c * u_x$ 
24 // Method: Finite difference , FTBS Method, Explicit
25 // Pointer-based Method
26 //
27 // The finite difference form is as bellow:
28 //
29 // 
$$\frac{U(X,T+dt) - U(X,T)}{dt} = -c * \frac{(U(X-dx,T+dt) + U(X+dx,T+dt))}{2 * dx} + F(X, T+dt)$$

30 //
31 //
32 //
33
34 int main ( );
35
36 // *****
37
38 cout << "\n";
39 cout << "Finite Difference: FTBS for 1D Wave Equation:\n";
40 cout << " C++ language\n";
41 cout << "\n";
42 cout << " Solve the constant-velocity advection equation in 1D,\n";
43 cout << "  $du/dt = -c du/dx$ \n";
44 cout << " over the interval:\n";
45 cout << "  $0.0 \leq x \leq 300.0$ \n";
46 cout << " with a given boundary conditions, and\n";
47 cout << " with a given initial condition\n";
48 cout << "  $u(0,x) = 80 * \sin((\pi/60)*(x-40))$  for  $40.0 \leq x \leq 100.0$ \n";
49 cout << "  $= 0$  elsewhere.\n";
50 cout << "\n";
51 cout << " We use a method known as FTBS:\n";
52 cout << " FT: Forward Time :  $du/dt = (u(t+dt,x)-u(t,x))/dt$ \n";
53 cout << " BS: Backward Space:  $du/dx = (u(t,x)-u(t,x-dx))/dx$ \n";
54 cout << "\n";
55
56 timestamp ( );
57
58 {
59     double a;
60     double b;
61     double c;
62     double L;
63     double Endtime;
64     //string command_filename = "advection_commands.txt";
65     //ofstream command_unit;
66     //string data_filename = "advection_data.txt";
67     //ofstream data_unit;
68     string data_filename = "x301_t200.txt";
69     ofstream data_unit;
70     //ofstream output_t10_x6001;
71
72     double dt; // time-step
73     double dx; // size-step
74     double nu = 1.48 * 1E-5; // viscosity / density (SI)
75     double stab; // stability checker → Corant Number
76     int i;
77     int j;
78     //int jml;
79     //int jpl;
80     int nt = 200; // time steps
81     int nx = 301; // mesh numbers
82     //int nt_step;
83     //int plotstep;
84     //double t;
85     double u[nt][nx]; // old velocity
86     double unew[nt][nx]; // new velocity
87     double x[nx];
88
89
90     L = 300; // Length (m)
91     a = 0.0; // x-start

```

```

92 b = 300.0; // x-end
93 c = 330; // Propagation Velocity
94 Endtime = 0.6; // Duration (s)
95
96 // *****
97 //nx = 6001; // nx: Number of nodes
98 dx = L / ( double ) ( nx - 1 ); // mesh size
99 // *****
100 // a loop to initialize the velocity vector.
101 for ( i = 0; i < nx; i++ )
102 {
103     x[i] = x[i] + dx*(i);
104     cout << x[i] << " ";
105     cout << '\n';
106 }
107 cout << "x-direction mesh vector, set.\n";
108 // *****
109 // setting boundary conditions:
110 for ( i = 0; i < ( (nx-1)/b ) *40 ; i++ )
111 {
112     u[0][i] = 0;
113     unew[0][i] = 0;
114     cout << " u[0][]" << i <<"] = " << u[0][i] << '\n';
115 }
116 cout << "B.C. part 1, set.\n";
117
118 for ( i = ( (nx-1)/b ) *40 ; i < ( (nx-1)/b ) *100 ; i++ )
119 {
120     cout << "x = " << x[i] << '\n';
121     u[0][i] = 80 * sin( (pi)*(x[i] - 40) / 60.0 );
122     unew[0][i] = 80 * sin( (pi)*(x[i] - 40) / 60.0 );
123 }
124 cout << "B.C. part 2, set.\n";
125
126 for ( i = (((nx-1)/b) *100 ); i < (nx ); i++ )
127 {
128     u[0][i] = 0;
129     unew[0][i] = 0;
130 }
131 cout << "B.C. part 3, set.\n";
132 // *****
133 //nt = 10; // Time Resolution
134 dt = Endtime / ( double ) ( nt ); // time-step
135 cout << "Time resolution, set. \n";
136 // *****
137 // setting initial condition:
138 cout << "setting initial condition ... \n";
139 for ( i = 0; i < nt; i++ )
140 {
141     //cout << i << " ";
142     u[i][0]=0;
143     unew[i][0] = 0;
144     //cout << i << "-->" << u[i][0] << " ";
145 }
146 cout << "Initial condition, set. \n";
147 // *****
148 stab = nu * dt / dx / dx;
149 cout << " courant number = " << stab << '\n';
150 if ( stab > 0.5 )
151 {
152     cout << " --> The stability criteria do not match <-- ";
153     return 0;
154 }
155 // *****
156 // ***** Core Calculation: start *****
157 //data_unit.open("C:\\Users\\Diamonds\\Documents\\CFD1\\x6001_t20.txt");
158 data_unit.open( data_filename.c_str ( ) );
159 cout << "file opened.";
160
161 if( !data_unit ) { // file couldn't be opened
162     cerr << "Error: file could not be opened" << endl;
163     exit(1);

```

```

164 }
165 //data_unit.open("x6001_t20.txt", ios::app);
166 for ( i = 0; i < nt; i++ )
167 {
168     //cout << "in calc loop \n";
169     data_unit << "zone\n";
170     //cout << "zone printed. \n";
171     //data_unit << fixed << setprecision(2) << x[0] << '\t' << fixed << setprecision(2) << u[i
172     ][0] << '\n';
173     //data_unit << '\t' << x[0] << '\t' << i << '\t' << u[i][0] << '\n';
174     data_unit << x[0] << '\t' << u[i][0] << '\n';
175     //fprintf(fp_ptr, "%f\t%f\n", x[0], u[i][0]);
176     // First do this for all x in xn:
177     for ( j = 1; j < nx; j++ )
178     {
179         unew[i+1][j] = u[i][j] - (c * dt / dx) * ( u[i][j] - u[i][j-1] );
180         //cout << "u_new = " << unew[i+1][j] << '\n';
181         u[i+1][j] = unew[i+1][j];
182         //fprintf(fp_ptr, "%f\t%f\n", x[j], u[i][j]);
183         //data_unit << fixed << setprecision(2) << x[j] << '\t' << fixed << setprecision(2) <<
184         u[i][j] << '\n';
185         //data_unit << '\t' << x[j] << '\t' << i << '\t' << u[i][j] << '\n';
186         data_unit << x[j] << '\t' << u[i][j] << '\n';
187         //cout << "printed: x = " << x[j] << '\t' << "u = " << u[i][j] << '\n';
188     }
189     data_unit << "\n";
190 }
191 cout << "Calculation, done.\n";
192 cout << " courant number = " << stab << '\n';
193 data_unit.close();
194
195 // ***** Core Calculation: end *****
196
197 // Free memory.
198 //
199 //
200 // Terminate.
201 //
202 cout << "\n";
203 cout << " Finite Difference: FTBS for 1D Wave Equation\n";
204 cout << " Array-based Algorithm\n";
205 cout << " Normal end of execution.\n";
206 cout << "\n";
207
208 return 0;
209 }
210 //*****
211 void timestamp ( )
212 {
213     # define TIME_SIZE 40
214
215     static char time_buffer[TIME_SIZE];
216     const struct std::tm *tm_ptr;
217     size_t len;
218     std::time_t now;
219
220     now = std::time ( NULL );
221     tm_ptr = std::localtime ( &now );
222
223     len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );
224
225     std::cout << time_buffer << "\n";
226
227     return;
228     # undef TIME_SIZE
229 }
230 %*****

```

6.4 Code for The FTBS Method – Pointer-based

```

1 %*****
2 # include <cstdlib>
3 # include <iostream>
4 # include <fstream>
5 # include <iomanip>
6 # include <cmath>
7 # include <ctime>
8 # include <cstring>
9
10 using namespace std;
11
12 int main ( );
13 int i4_modp ( int i, int j );
14 int i4_wrap ( int ival, int ilo, int ihi );
15 double *initial_condition ( int nx, double x[] );
16 double *r8vec_linspace_new ( int n, double a, double b );
17 void timestamp ( );
18
19 // *****
20
21 int main ( )
22
23 // *****
24 //
25 // Language: C++
26 // Author: Dina Soltani Tehrani.
27 // Course: Advanced CFD1 – by Dr. Taeibi Rahni
28 // Sharif University of Technology
29 //
30 // Equation:  $u_t = -c * u_x$ 
31 // Method: Finite difference, FTBS Method, Explicit
32 // Pointer-based Method
33 //
34 // The finite difference form is as bellow:
35 //
36 // 
$$\frac{U(X,T+dt) - U(X,T)}{dt} = -c * \frac{(U(X-dx,T+dt) + U(X+dx,T+dt))}{2 * dx} + F(X, T+dt)$$

37 //
38 //
39 //
40 {
41     double a;
42     double b;
43     double c;
44     double L;
45     double duration;
46     double pi = 4*atan(1.0);
47     string data_filename = "FTBS_x1201_t1000_p.txt";
48     ofstream data_unit;
49     double dt; // time-step
50     double dx; // size-step
51     int i;
52     int j;
53     int jml;
54     int jp1;
55     int nx;
56     int nt;
57     int nt_step;
58     int plotstep;
59     double t;
60     double *u; // old velocity
61     double *unew; // new velocity
62     double *x;
63
64     cout << "\n";
65     cout << "Finite Difference: FTBS for 1D Wave Equation:\n";
66     cout << "  C++ language\n";
67     cout << "\n";
68     cout << "  Solve the constant-velocity advection equation in 1D,\n";
69     cout << "     $du/dt = -c du/dx$ \n";
70     cout << "    over the interval:\n";
71     cout << "       $0.0 \leq x \leq 300.0$ \n";
72     cout << "    with a given boundary conditions, and\n";

```



```

73 cout << "    with a given initial condition\n";
74 cout << "    u(0,x) = 80 * sin( (pi/60)* (x-40) ) for 40.0 <= x <= 100.0\n";
75 cout << "    = 0 elsewhere.\n";
76 cout << "\n";
77 cout << "    We use a method known as FTBS:\n";
78 cout << "    FT: Forward Time : du/dt = (u(t+dt,x)-u(t,x))/dt\n";
79 cout << "    BS: Backward Space: du/dx = (u(t,x)-u(t,x-dx))/dx\n";
80 cout << "\n";
81
82 timestamp ( );
83
84 L = 300; // Length (m)
85 nx = 1201; // Mid Space Resolution
86 dx = L / ( double ) ( nx - 1 ); // size-step
87 a = 0.0; // x-start
88 b = 300.0; // x-end
89 x = r8vec_linspace_new ( nx, a, b );
90 nt = 1000; // Time Resolution
91 duration = 0.6; // Duration (s)
92 dt = duration / ( double ) ( nt ); // time-step
93 c = 330.0;
94
95 u = initial_condition ( nx, x ); // Initializing old velocity with initial condition.
96
97 // ***** Opening data file , and writing solutions as they are computed. *****
98
99 data_unit.open ( data_filename.c_str ( ) );
100
101 // Writing the initial data.
102 t = 0.0;
103 data_unit << x[0]
104 << " " << u[0] << "\n";
105
106 // Writing data as we move forward in space.
107 for ( j = 0; j < nx; j++ )
108 {
109     data_unit << x[j]
110     << " " << u[j] << "\n";
111 }
112 data_unit << "zone";
113 data_unit << "\n";
114
115
116
117 //
118
119 *****
120
121 nt_step = 100;
122 unew = new double[nx]; // Creating the new velocity vector
123
124 // ***** Core Calculation: begin *****
125 for ( i = 0; i < nt; i++ )
126 {
127     //data_unit << "zone\n";
128     // First do this for all x in xn:
129     for ( j = 0; j < nx; j++ )
130     {
131         jm1 = i4_wrap ( j - 1, 0, nx - 1 ); // phi_n_j-1
132         //jp1 = i4_wrap ( j + 1, 0, nx - 1 ); // phi_n_j+1
133         unew[j] = u[j] - ( c * dt / dx ) * ( u[j] - u[jm1] );
134         //if ( u[j] < 1e-04 ) { u[j] = 0; }
135     }
136     // Then, do this , again for all x in xn:
137     for ( j = 0; j < nx; j++ )
138     {
139         u[j] = unew[j];
140         //data_unit << x[j] << " " << u[j] << "\n";
141     }
142
143     if ( i == nt_step - 1 )
144     {
145         t = ( double ) ( i ) * dt;

```

```

143     for ( j = 0; j < nx; j++ )
144     {
145         //if ( u[j] < 1e-05 ) { u[j] = 0; }
146         data_unit << x[j]
147             << " " << u[j] << "\n";
148     }
149     data_unit << "zone";
150     data_unit << "\n";
151     nt_step = nt_step + 100;
152 }
153
154 //data_unit << "zone";
155 //data_unit << "\n";
156 }
157 // ***** Core Calculation: end *****
158
159 //
160 // ***** Closing the data file as the computation is done. *****
161 //
162 data_unit.close ( );
163
164 cout << "\n";
165 cout << " Plot data written to the file \"" << data_filename << "\"\n";
166 //
167 // Free memory.
168 //
169 delete [] u;
170 delete [] unew;
171 delete [] x;
172 //
173 // Terminate.
174 //
175 cout << "\n";
176 cout << " Finite Difference: FTBS for 1D Wave Equation\n";
177 cout << " Pointer-based Method\n";
178 cout << " Normal end of execution.\n";
179 cout << "\n";
180 timestamp ( );
181
182 return 0;
183 }
184
185 //
186 // ***** Functions *****
187
188 //*****
189
190 //*****
191 int i4_modp ( int i, int j )
192 {
193     int value;
194
195     if ( j == 0 )
196     {
197         cerr << "\n";
198         cerr << "I4_MODP - Fatal error!\n";
199         cerr << " I4_MODP ( I, J ) called with J = " << j << "\n";
200         exit ( 1 );
201     }
202
203     value = i % j;
204
205     if ( value < 0 )
206     {
207         value = value + abs ( j );
208     }
209
210     return value;
211 }
212
213 //*****
214 int i4_wrap ( int ival, int ilo, int ihi )

```

```

215 {
216     int jhi;
217     int jlo;
218     int value;
219     int wide;
220
221     if ( ilo <= ihi )
222     {
223         jlo = ilo;
224         jhi = ihi;
225     }
226     else
227     {
228         jlo = ihi;
229         jhi = ilo;
230     }
231
232     wide = jhi + 1 - jlo;
233
234     if ( wide == 1 )
235     {
236         value = jlo;
237     }
238     else
239     {
240         value = jlo + i4_modp ( ival - jlo , wide );
241     }
242
243     return value;
244 }
245
246 // *****
247 double *initial_condition ( int nx, double x[] )
248 {
249     int i;
250     double *u;
251     double pi = 4*atan(1.0);
252
253     u = new double[nx];
254
255     for ( i = 0; i < nx; i++ )
256     {
257         if ( 40 <= x[i] && x[i] <= 100 )
258         {
259             u[i] = 80 * sin( (pi/60)*(x[i] - 40) );
260         }
261         else
262         {
263             u[i] = 0.0;
264         }
265     }
266     return u;
267 }
268
269 // *****
270 double *r8vec_linspace_new ( int n, double a_first, double a_last )
271 {
272     double *a;
273     int i;
274
275     a = new double[n];
276
277     if ( n == 1 )
278     {
279         a[0] = ( a_first + a_last ) / 2.0;
280     }
281     else
282     {
283         for ( i = 0; i < n; i++ )
284         {
285             a[i] = ( ( double ) ( n - 1 - i ) * a_first
286                     + ( double ) ( i ) * a_last )

```

```

287         / ( double ) ( n - 1 );
288     }
289 }
290 return a;
291 }
292
293 // *****
294 void timestamp ( )
295 {
296     # define TIME_SIZE 40
297
298     static char time_buffer[TIME_SIZE];
299     const struct std::tm *tm_ptr;
300     size_t len;
301     std::time_t now;
302
303     now = std::time ( NULL );
304     tm_ptr = std::localtime ( &now );
305
306     len = std::strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm_ptr );
307
308     std::cout << time_buffer << "\n";
309
310     return;
311     # undef TIME_SIZE
312 }
313 %*****

```