Cranfield University

Academic Year: 2022-2023

Department:

Computation and Software Techniques in Engineering (SETC Option)

Computational Methods & C++ Assignment

Student Name:

Amit Tamhane (S375967)

Tejas Patil (S387998)

Date of Submission:
21/November/2022

Table of Contents

Table of Figures

Table of Tables

Table of Equations

Abstract

In this assignment we will be solving the wave equation given using four different discretization methods for different values of Δt. The methods are Explicit Upwind FTBS (Forward time, Backward space), Implicit Upwind FTBS (Forward time, Backward space), Lax-Wendroff and Implicit FTCS (Forward time, Central space). We will be studying their stability and accuracy.

1.0 Introduction

In this assignment we will be solving the wave equation given using four different discretization methods for different values of Δt. The methods are Explicit Upwind FTBS (Forward time, Backward space), Implicit Upwind FTBS (Forward time, Backward space), Lax-Wendroff and Implicit FTCS (Forward time, Central space). We will be studying their stability and accuracy.

We will now define some of the terminologies used in this assignment below.

Finite Difference Method: The most straightforward way for discretizing partial differential equations is the finite-difference approach. Consider a location in space where a set of discrete equations known as finite-difference equations are used to substitute the continuum representation of the equations. Since the finite-difference approach is often specified on a regular (Zhou, 1993)(*Https://Www.Machinedesign.Com/3d-Printing-Cad/Fea-and-imulation/Article/21832072/Whats-the-Difference-between-Fem-Fdm-and-Fvm*, n.d.).

Partial Differential Equation: The partial derivatives in the equation are approximated using finite difference relations in order to solve a specific Partial Differential Equation (PDE) using numerical techniques (Polyanin et al., 2008).

Initial and boundary conditions: To obtain a unique solution of a PDE we need to provide a set of conditions to determine the function that is obtained from the integration of the PDE (Hoffmann & Chiang, n.d.).

An **initial condition** is a requirement for which the dependent variable is specified at some initial state (Hoffmann & Chiang, n.d.).

A **boundary condition** is a requirement that the dependent variable or its derivatives must satisfy on the boundary of the domain of the PDE (Hoffmann & Chiang, n.d.).

Stability: A numerical scheme is said to be stable if any error introduced in the finite difference equation does not grow with the solution of the finite difference equation (Hoffmann & Chiang, n.d.).

Implicit method: When more than one unknown appears in the FDE it is defined as being implicit.

Von Neumann Stability Analysis is the procedure used to find the stability of a finite difference equation. In this we expand a solution of the FDE using Fourier Series. Based on the value of the Amplification Factor (G) we can say whether the FDE is stable or not (Hoffmann & Chiang, n.d.).

Truncation error: It represents the difference between an exact differential equation and its finite difference representation at a point in space and time.

2.0 Methodology

As mentioned earlier, in this assignment we have use C++ to solve a first order wave equation using the following methods:

- Explicit Upwind FTBS (Forward time, Backward space)
- Implicit Upwind FTBS (Forward time, Backward space)
- Lax-Wendroff
- Implicit FTCS (Forward time, Central space)

The first order wave equation is given below:

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0$$

Equation 1: First Order Wave Equation

The initial and boundary conditions are as shown below:

$$\begin{aligned} f(x,t) &= 0 && 0 \le x \le 50 + 250t \\ f(x,t) &= 100\left\{\sin\left[\pi\left(\frac{x-50-250t}{60}\right)\right]\right\} && 50+250t \le x \le 110+250t \\ f(x,t) &= 0 && 110+250t \le x \le L \end{aligned}$$

Equation 2: The Initial and Boundary Conditions

Let’s look at the methods now.

The Explicit upwind FTBS (Forward time, Backward space) is given by the following formula:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} + u\frac{f_i^n - f_{i-1}^n}{\Delta x} = 0$$

Equation 3: The Explicit upwind FTBS (Forward time, Backward space) Formula

We get this equation when we discretize the first order wave equation in space and time. In this method we have only one unknown f_i^{n+1} and using the initial condition we can solve equation. We will get the numerical solution as shown in the equation below:

$$f_i^{n+1} = f_i^n - u\frac{\Delta t}{\Delta x}\left(f_i^n - f_{i-1}^n\right)$$

Equation 4: Discretized First Order Wave Equation

In Implicit upwind FTBS we have the following formula:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} + u\frac{f_i^{n+1} - f_{i-1}^{n+1}}{\Delta x} = 0$$

Equation 5: The Implicit upwind FTBS (Forward time, Backward space) Formula

In this method we have solve for next step in time and space so we will have two unknowns. We can compute the values using the boundary condition given in the question. Using the value, we get we use it for the next loop and calculate values.

For Implicit FTCS, we have the following formula:

$$\frac{f_i^{n+1} - f_i^n}{\Delta t} + a\frac{f_{i+1}^{n+1} - f_{i-1}^{n+1}}{2\Delta x} = 0$$

Equation 6: The Implicit FTCS (Forward time, Central space) Formula

This equation is not straight forward to solve as we don't have boundary conditions for the points in the central space. To solve this, we will use two loops. The outer loop will have n=0 and write the resulting equation. Then we will take i=1 as an inner loop. Using this we will get an equation, but it will still not be solvable as it will have 3 unknowns. After this we increase the values i and get a range of equations which we can express in the form of matrix. A sample matrix is shown below:

$$\underset{A}{\begin{pmatrix} b_1 & c_1 & \dots & \dots & 0 \\ a_2 & b_2 & c_2 & \dots & 0 \\ 0 & a_3 & b_3 & c_3 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & a_n & b_n \end{pmatrix}} \underset{x}{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}} = \underset{B}{\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}}$$

Equation 7: Tridiagonal Matrix

The matrix (Equation 7) is a tri-diagonal matrix which can be solved to get the values of X matrix, which we need.

For Lax-Wendorff, we have the following formula:

$$f_i^{n+1} = \frac{1}{2}\frac{u^2\Delta t}{\Delta x^2}\left[f_{i+1}^n - 2f_i^n - f_{i-1}^n\right] - \frac{1}{2}\frac{u\Delta t}{\Delta x}\left[f_{i+1}^n - f_{i-1}^n\right] + f_i^n$$

Equation 8: The Lax-Wendorff Formula

This method is a quadratic method i.e., it takes second order derivatives. By solving it we will get the solution.

We will also be calculating the norms L_0 ,L_1,L_2 to analyze the errors in the methods.

$L_0 = \max |e_i|$

$L_1 = \sum |e_i|$

$L_2 = \sqrt{\sum e_i^2}$

The L_0 norm gives us the maximum error. L_1 norm is the sum of all the absolute values of the error. L_2 norm is the sum of squares of the error. Squaring the errors gives more importance to the bigger errors and hence giving a better analysis than L_1 and L_0.

2.1 Code Design

Since the assignments primary topic was calculate numerical solutions partial differential equations, we used just one Main.cpp file, which contains all the functions we require, to create our code. As a result, creating and calculating our numerical approaches and analytical values is our main responsibility. We created specific functions for each technique's calculus (Explicit Upwind FTBS, Implicit Upwind FTBS, Lax-Wendroff, Implicit FTCS and Analytical Values). Apart from the Implicit FTCS function, we save the calculated values in a single vector and rewrite it for the following iteration before outputting it before values change. We wrote our code in the same manner as we would have in Python for the Implicit FTCS function. Because the implicit FTCS function involves a tridiagonal matrix, we solved it using the Thomas algorithm. The user has the choice of the computation time and the method to test. We are specifically manually extracting Excel data only from 0.0 to 0.5 seconds after the 0.1 seconds indicated in the inquiry. The main function must call the others only when necessary and on demand, then print the requested data. Calculating numerically, which isn't clean coding generally, was incredibly easy and effective in our case.

3.0 Results and Analysis

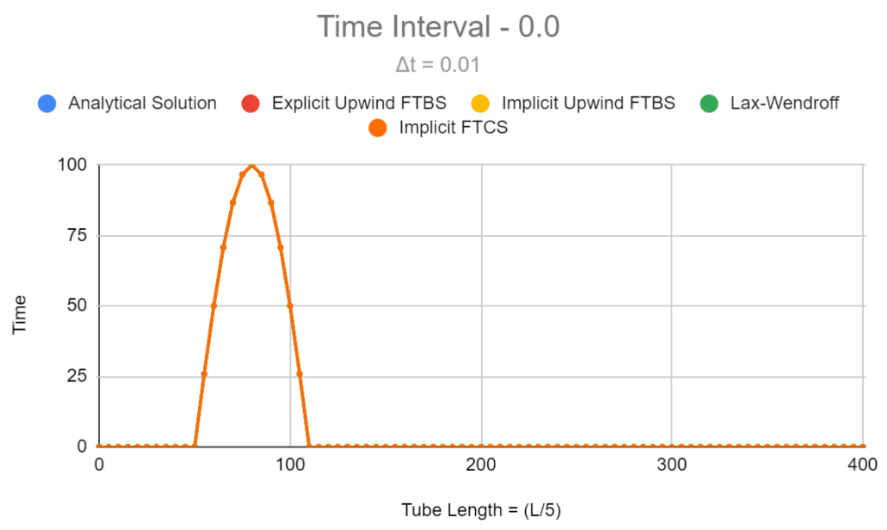


Figure 1: Analytical and Numerical Solutions for time interval 0.0 when Δt = 0.01

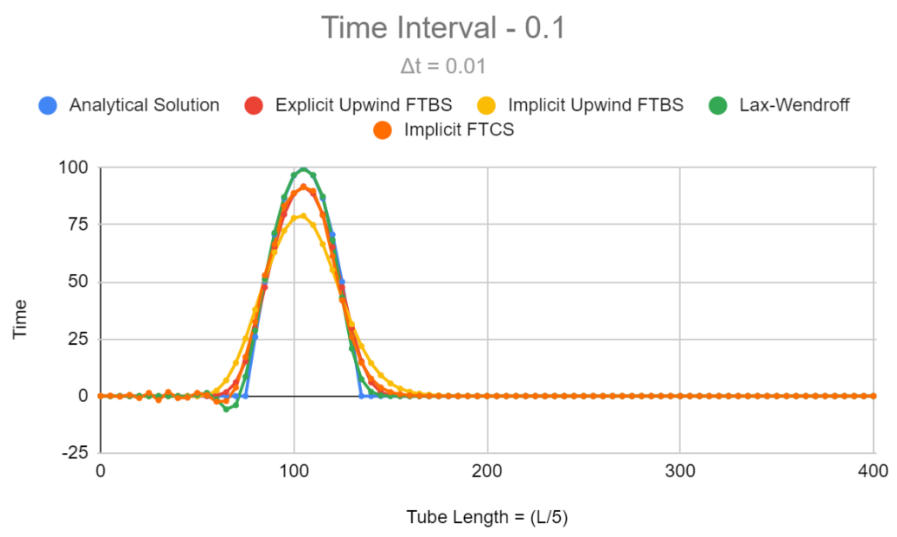


Figure 2: Analytical and Numerical Solutions for time interval 0.1 when Δt = 0.01

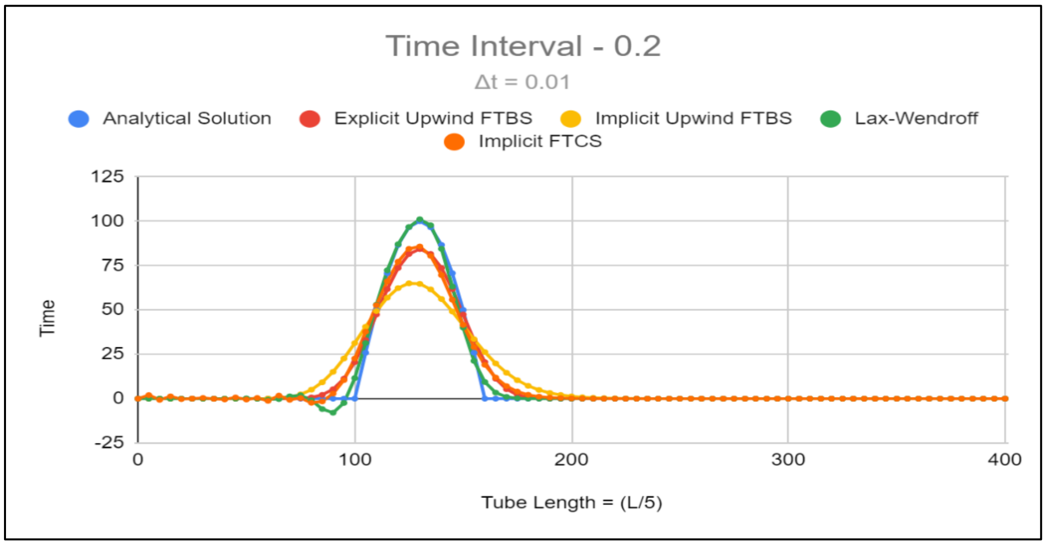


Figure 3: Analytical and Numerical Solutions for time interval 0.2 when Δt = 0.01

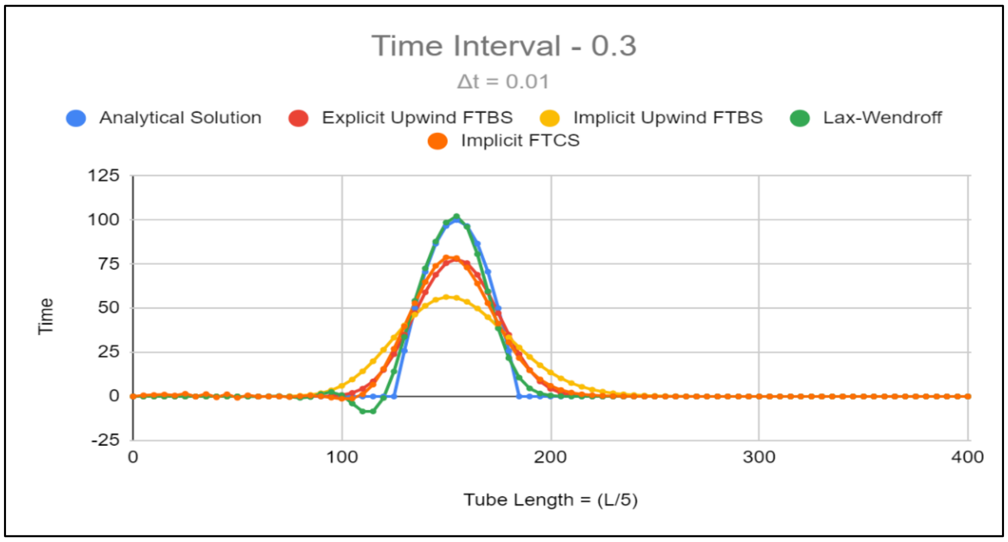


Figure 4: Analytical and Numerical Solutions for time interval 0.3 when Δt = 0.01

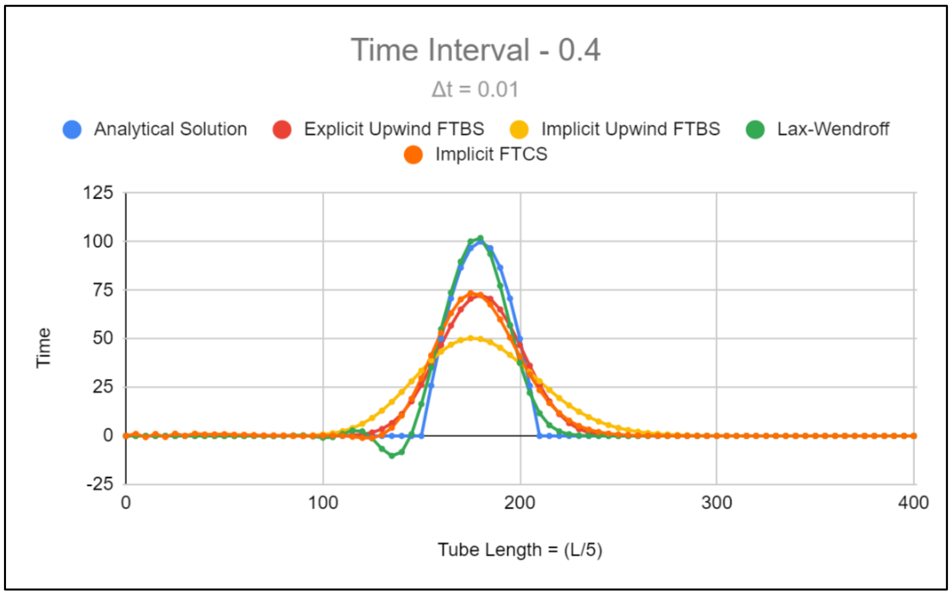


Figure 5: Analytical and Numerical Solutions for time interval 0.4 when Δt = 0.01

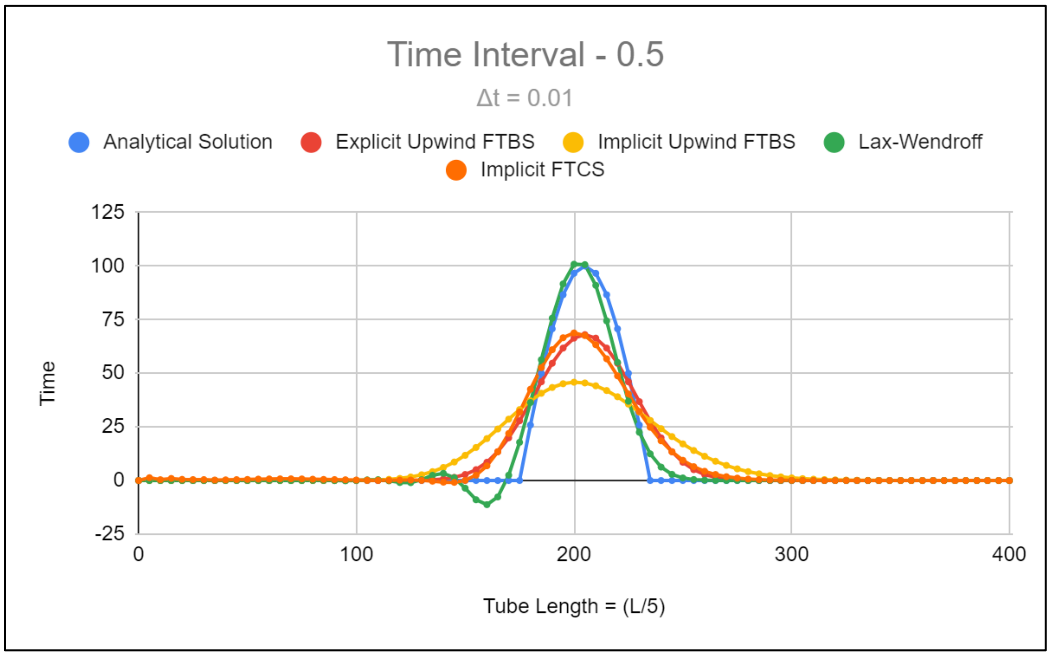


Figure 6: Analytical and Numerical Solutions for time interval 0.5 when Δt = 0.01

Time Step	Analytical Solution	Explicit Upwind FTBS	Implicit Upwind FTBS	Lax-Wendroff	Implicit FTCS
0	100	100	100	100	100
0.1	100	91.7668	78.8866	99.535	91.4634
0.2	100	84.2887	65.0022	101.082	85.7725
0.3	100	77.8569	56.3265	102.161	78.8386
0.4	100	72.4873	50.2911	101.831	73.4475
0.5	100	67.9985	45.825	100.825	68.7915

Table 1: Peak Values for Analytical and Numerical Solutions for 0.0 to 0.5 time step when Δt = 0.01

The above graphs (Fig1-6) show the analytical and numerical solution (Implicit FTBS, Explicit FTBS, Implicit FTCS and Lax-Wendroff) for our wave equation at Δt = 0.01 for various time steps from 0 to 0.5 . We can see that the maximum value for the analytical solution is 100. From these graphs we can see that Lax-Wendroff method is closest to the analytical solution. Explicit FTBS and Implicit FTCS have similar solutions. The Implicit FTBS differs most from the analytical solution indicating it will have most error. We can also see that as the time step increase from 0.0 to 0.5 the difference between the analytical and numerical solution is increasing.

From the graph we can also see that Lax-Wendroff method and Implicit FTCS are unstable. The stability of Lax-Wendroff depends on the "c" value. In this case as Δt is changing the value of c also changes. If c>1 then the method becomes unstable. So even though it looks more accurate it is not the best method to go with. Stability for Implicit FTCS will be discussed later.

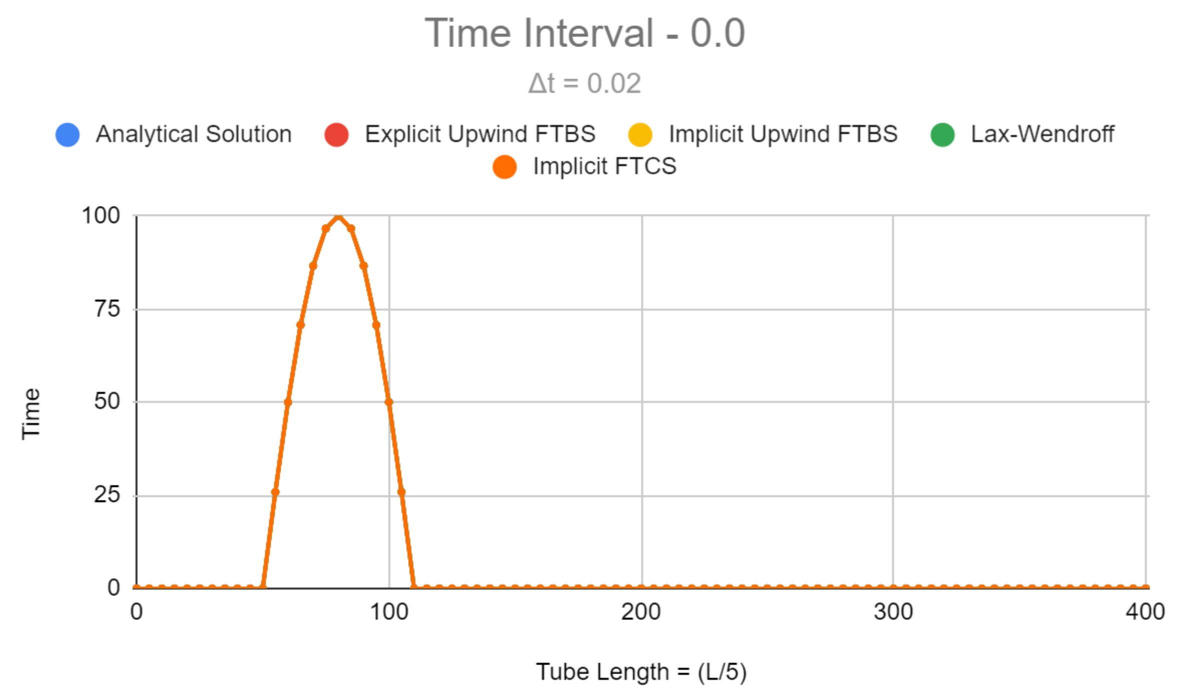


Figure 7: Analytical and Numerical Solutions for time interval 0.0 when Δt = 0.02

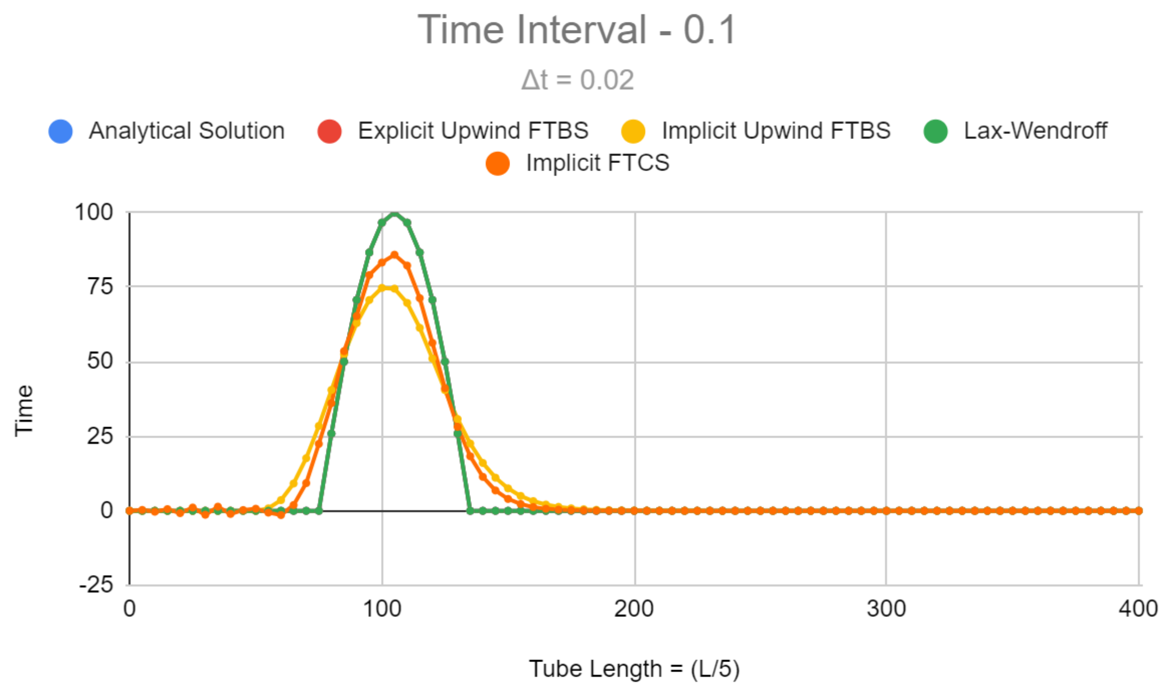


Figure 8: Analytical and Numerical Solutions for time interval 0.1 when Δt = 0.02

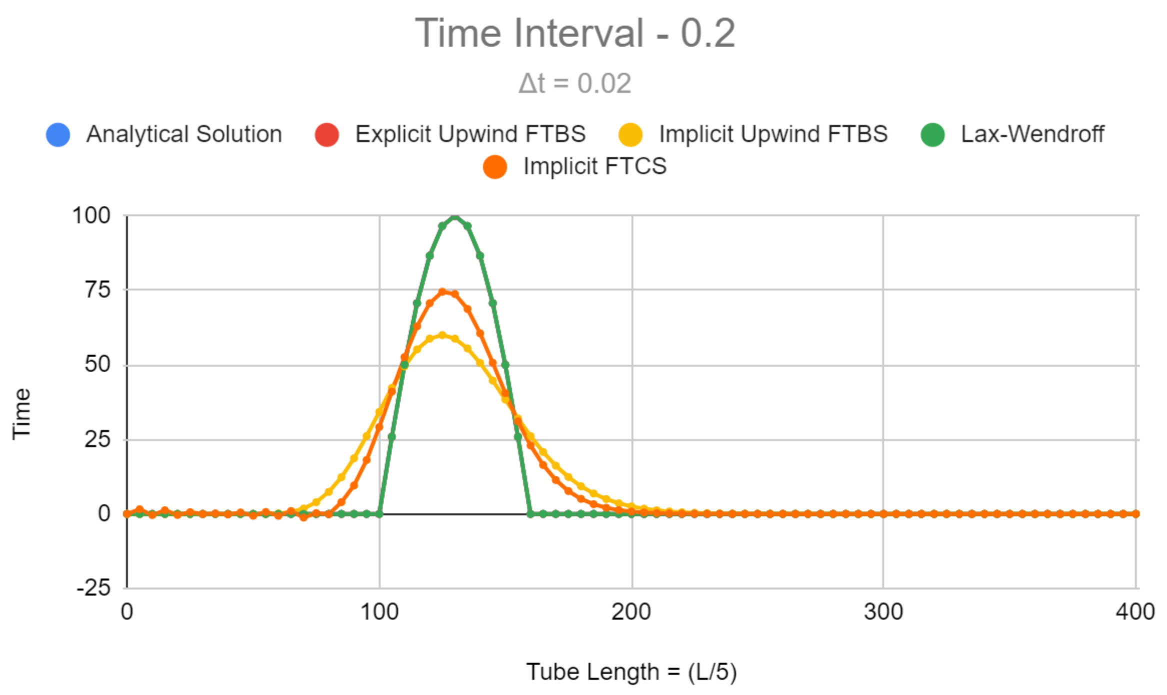


Figure 9: Analytical and Numerical Solutions for time interval 0.2 when Δt = 0.02

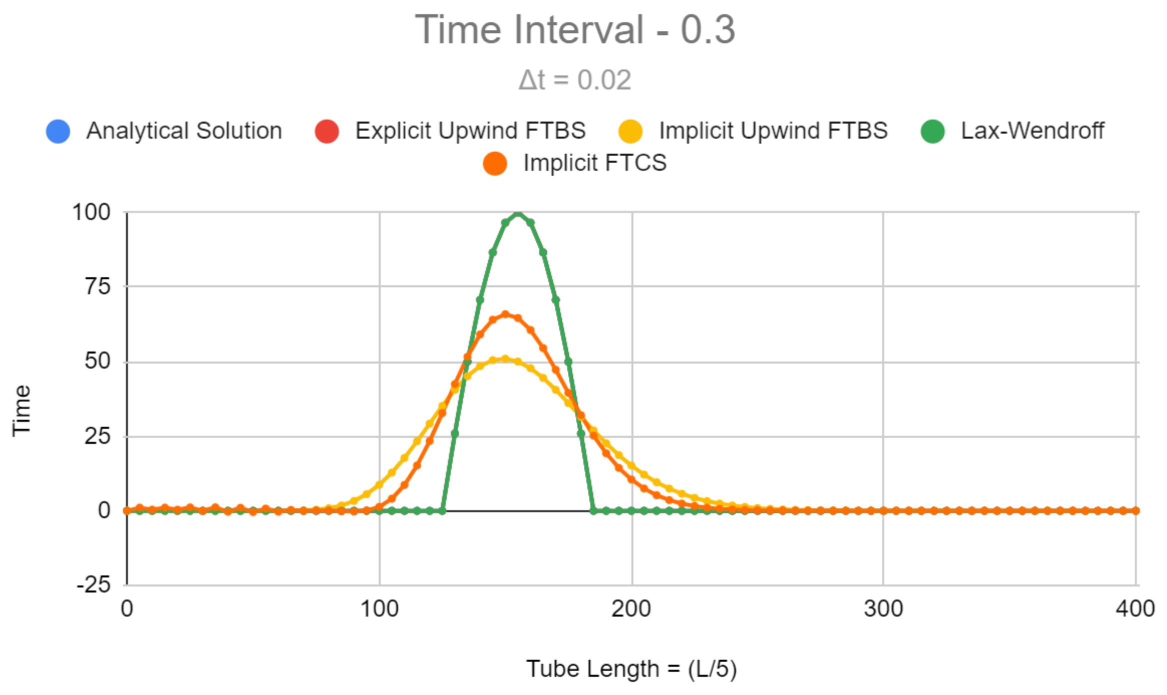


Figure 10: Analytical and Numerical Solutions for time interval 0.3 when Δt = 0.02

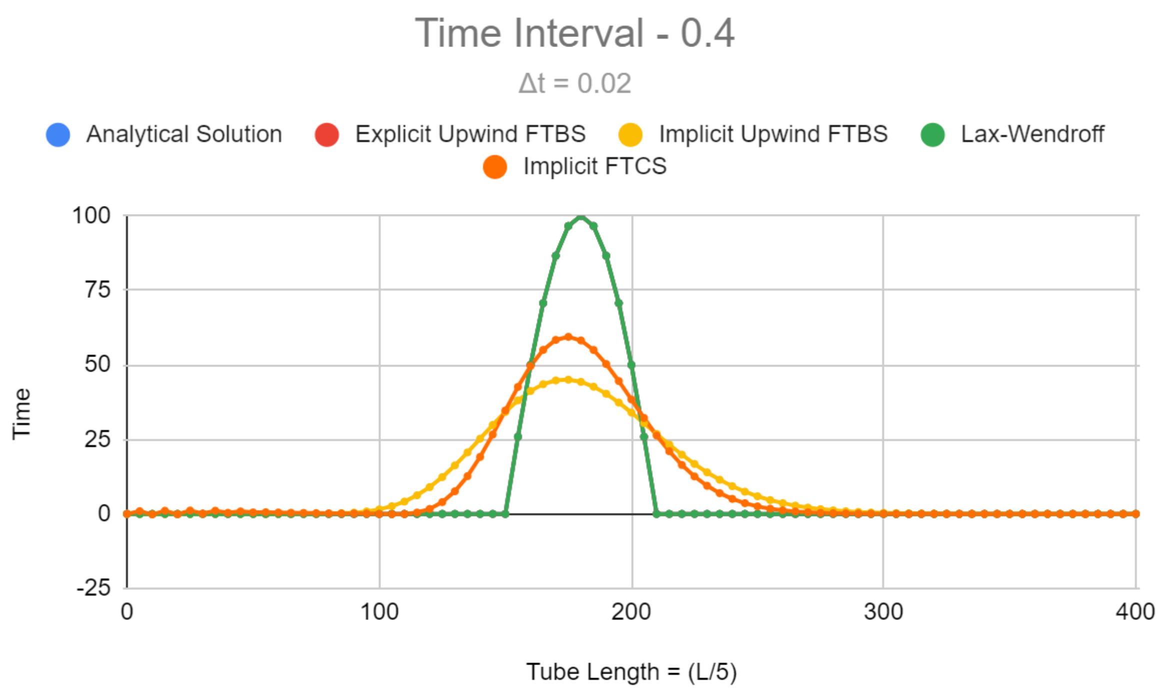


Figure 11: Analytical and Numerical Solutions for time interval 0.4 when Δt = 0.02

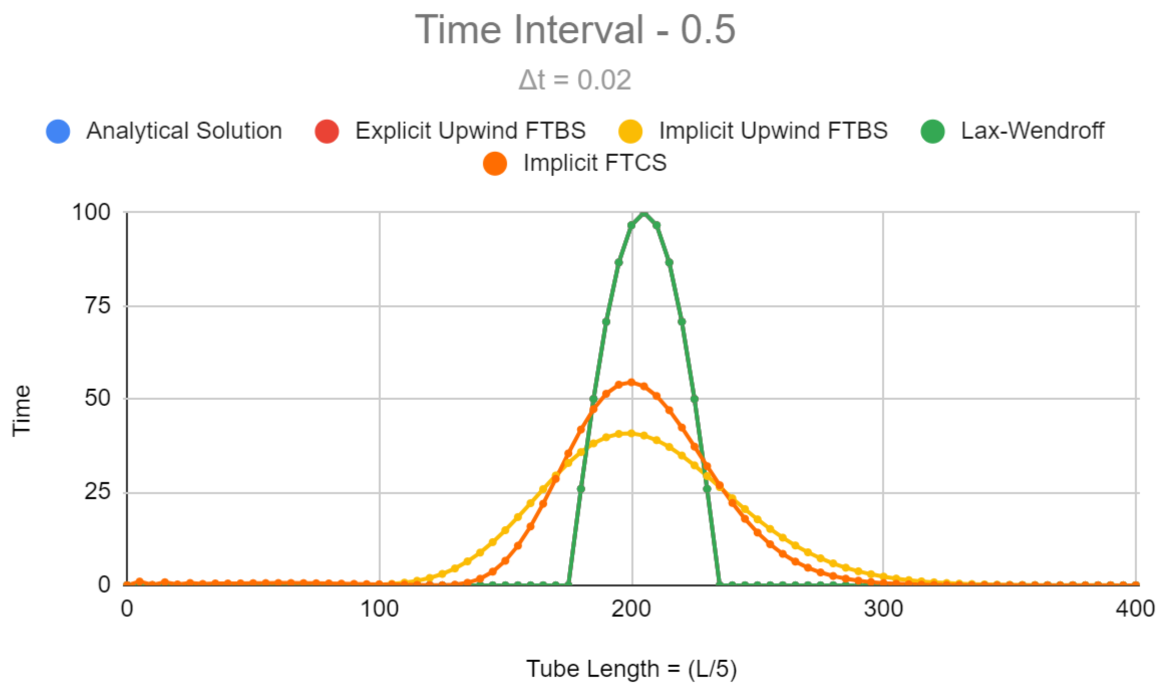


Figure 12: Analytical and Numerical Solutions for time interval 0.5 when Δt = 0.02

	Analytical Solution	Explicit Upwind FTBS	Implicit Upwind FTBS	Lax-Wendroff	Implicit FTCS
0	100	100	100	100	100
0.1	100	100	74.7266	100	85.8484
0.2	100	100	60.0007	100	74.5248
0.3	100	100	50.9865	100	65.906
0.4	100	100	45.0449	100	59.4229
0.5	100	100	40.7775	100	54.4737

Table 2: Peak Values for Analytical and Numerical Solutions for 0.0 to 0.5 time step when Δt = 0.02

The above graphs (Fig 7-12) show the numerical and analytical solution for Δt = 0.02 sec. From these graphs we can see that the solution for Lax-Wendroff and Explicit FTBS is closest to the analytical solution. As we saw for Δt = 0.01 sec, Implicit FTBS differs most from the analytical solution again indicating that it has most error. Similar to Δt = 0.01 sec we can see that with each time step the numerical solution moves further away from analytical solution. The peak value for Implicit FTBS at time step 0.5 for Δt = 0.02 is lower (40.7775) compared to the peak value for the same method at Δt = 0.01 and time step 0.5 (45.825).

From the graphs we can also see that only Implicit FTCS is unstable.

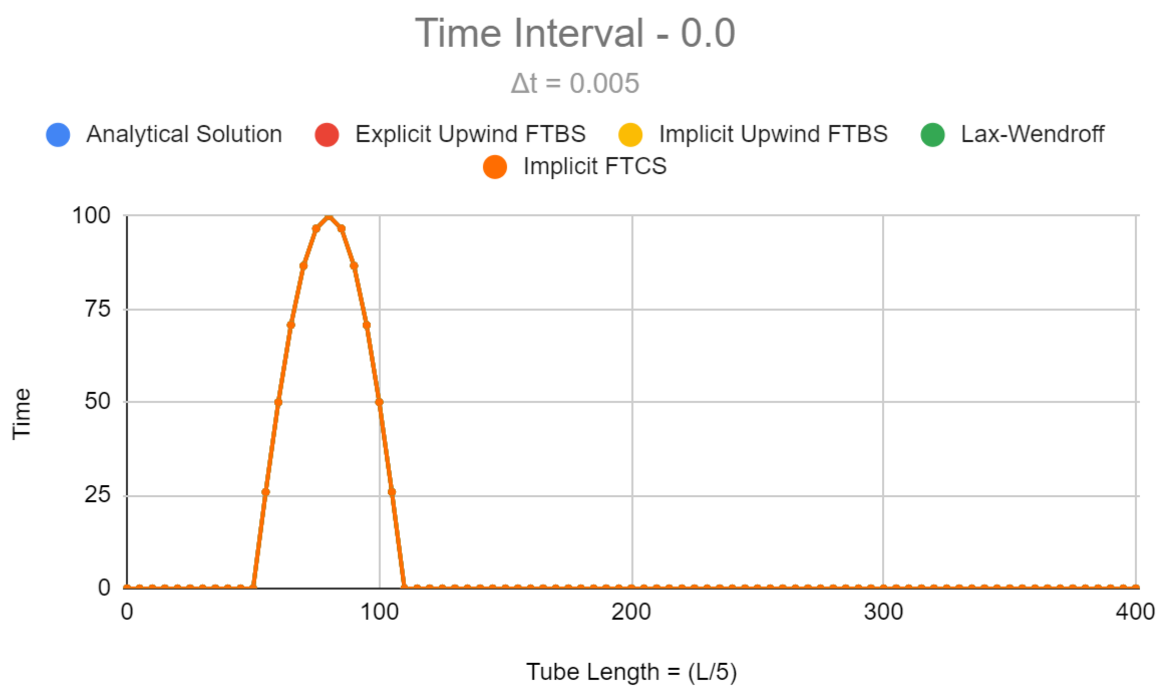


Figure 13: Analytical and Numerical Solutions for time interval 0.0 when Δt = 0.005

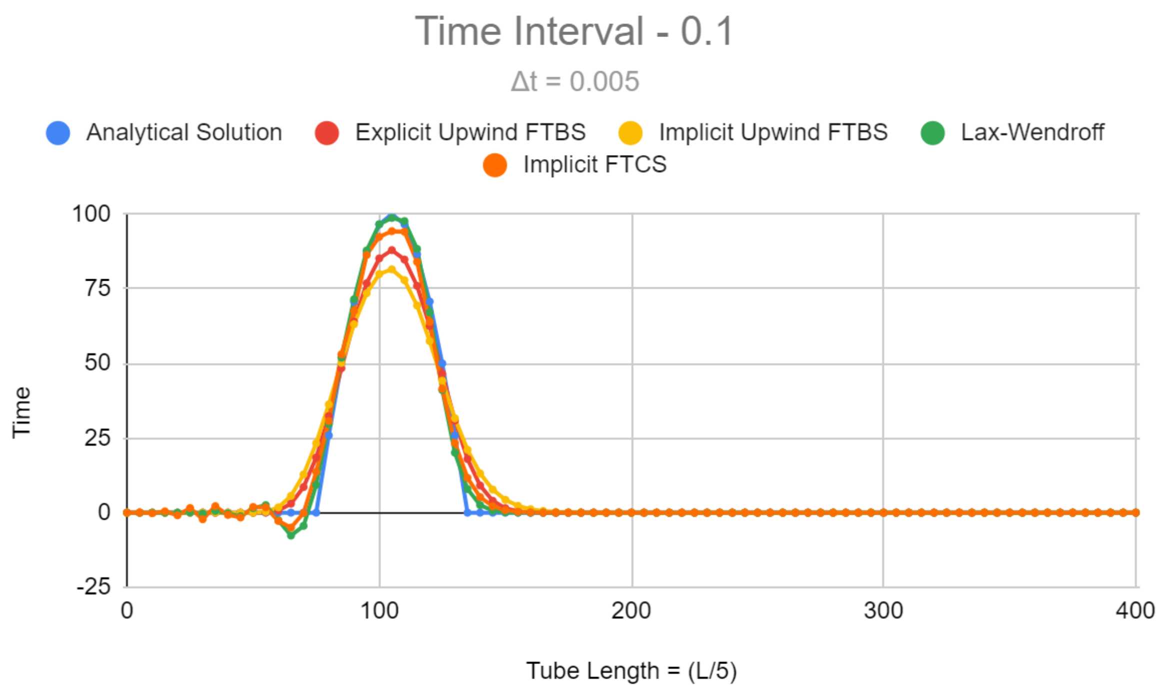


Figure 14: Analytical and Numerical Solutions for time interval 0.1 when Δt = 0.005

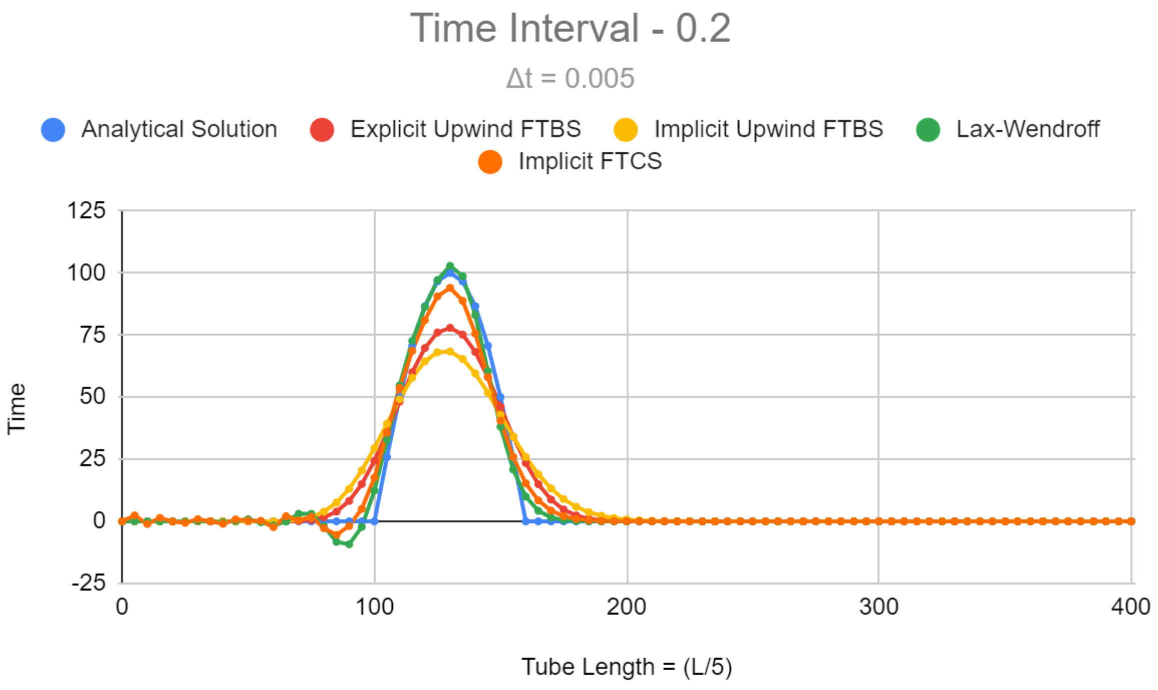


Figure 15: Analytical and Numerical Solutions for time interval 0.2 when Δt = 0.005

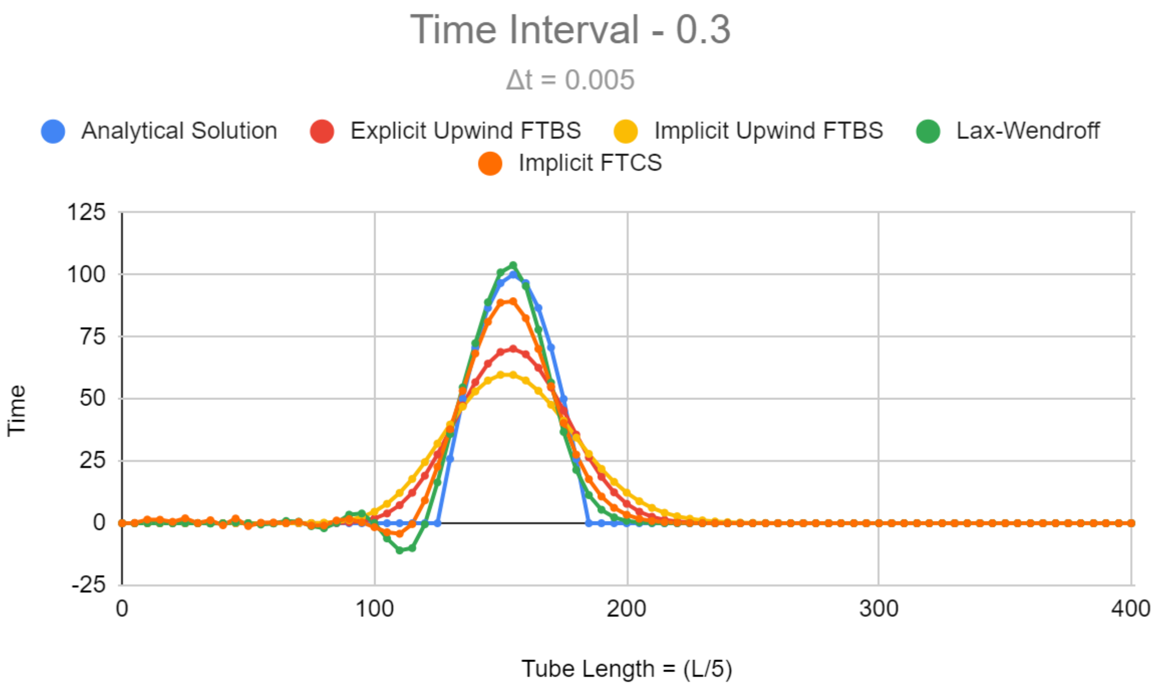


Figure 16: Analytical and Numerical Solutions for time interval 0.3 when Δt = 0.005

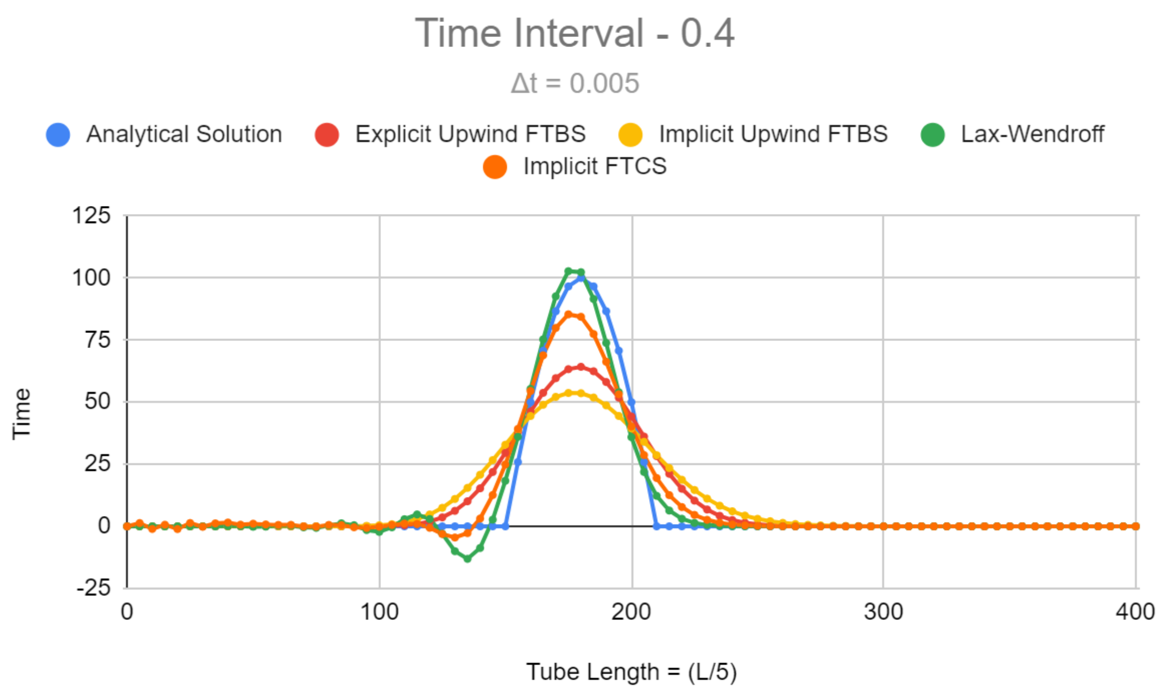


Figure 17: Analytical and Numerical Solutions for time interval 0.4 when Δt = 0.005

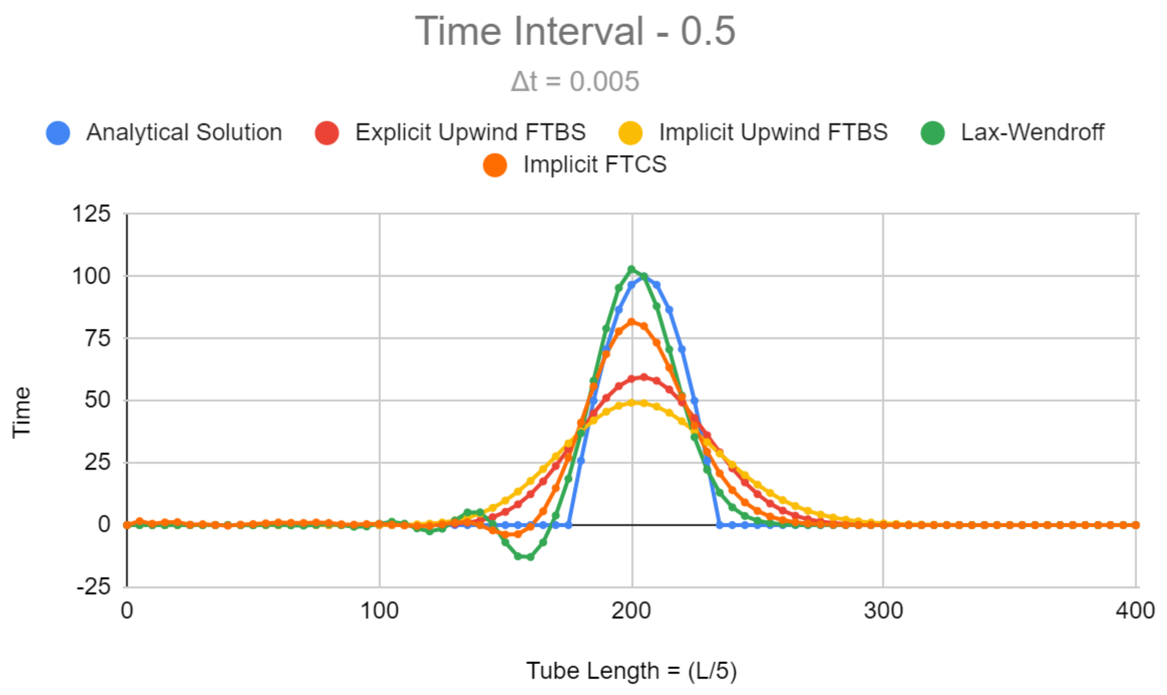


Figure 18: Analytical and Numerical Solutions for time interval 0.5 when Δt = 0.005

	Analytical Solution	Explicit Upwind FTBS	Implicit Upwind FTBS	Lax-Wendroff	Implicit FTCS
0	100	100	100	100	100
0.1	100	87.9603	81.5257	98.7476	94.3135
0.2	100	77.9231	68.3671	102.901	94.0264
0.3	100	70.1915	59.7062	103.802	89.2815
0.4	100	64.2349	53.6957	102.722	85.3595
0.5	100	59.5208	49.1564	102.892	81.7639

Table 3: Peak Values for Analytical and Numerical Solutions for 0.0 to 0.5 time step when $\Delta t = 0.005$

The above graphs show the numerical and analytical solutions for $\Delta t = 0.05$ sec. Again Lax-Wendroff is closest to the analytical solution and Implicit FTBS differs the most. We can see that the difference between the analytical and numerical solutions is less at $\Delta t = 0.05$ compared to at $\Delta t = 0.02$ sec and $\Delta t = 0.01$ sec.

We can also see that Implicit FTCS, and Lax-Wendroff is unstable.

3.1 Errors

For Δt = 0.01

	Explicit Upwind FTBS			Implicit Upwind FTBS			Lax-Wendroff			Implicit FTCS		
Time Interval	L0	L1	L2	L0	L1	L2	L0	L1	L2	L0	L1	L2
0	0	0	0	0	0	0	0	0	0	0	0	0
0.1	15.2146	1.339069	3.404714	25.2017	3.114377	7.125077	8.47943	0.652061	1.838201	17.1036	1.601748	1.442934
0.2	20.5936	2.344008	5.539235	35.2954	4.971181	10.73369	11.5795	1.0382	2.653446	22.6164	2.611529	1.764581
0.3	23.9564	3.173032	7.240434	44.1044	6.300886	13.02992	14.1604	1.389241	3.352644	26.9718	3.421145	1.95937
0.4	27.5127	3.882142	8.637467	50.1053	7.292326	14.61972	16.3705	1.741981	3.968964	29.7218	4.13131	2.043295
0.5	32.0015	4.49955	9.806251	54.5169	8.063357	15.79353	17.8114	2.064409	4.514994	33.2866	4.753348	2.055603

Table 4: L_0, L_1 and L_2 Norms for Numerical Solutions for for 0.0-to-0.5-time step when Δt = 0.01

For Δt = 0.02

	Explicit Upwind FTBS			Implicit Upwind FTBS			Lax-Wendroff			Implicit FTCS		
Time Interval	L0	L1	L2	L0	L1	L2	L0	L1	L2	L0	L1	L2
0	0	0	0	0	0	0	0	0	0	0	0	0
0.1	0	0	0	28.4452	3.764774	8.41237	0	0	0	22.4343	2.447645	5.502433
0.2	0	0	0	41.1881	5.830933	12.31563	0	0	0	29.1367	3.95484	8.616255
0.3	0	0	0	49.9656	7.249982	14.61816	0	0	0	35.9758	5.103875	10.83848
0.4	0	0	0	55.6907	8.259643	16.1369	0	0	0	41.7985	6.012709	12.48792
0.5	0	0	0	59.8044	9.02031	17.22651	0	0	0	46.6065	6.808058	13.75003

Table 5: L_0, L_1 and L_2 Norms for Numerical Solutions for for 0.0-to-0.5-time step when Δt = 0.02

For Δt = 0.005

	Explicit Upwind FTBS			Implicit Upwind FTBS			Lax-Wendroff			Implicit FTCS		
Time Interval	L0	L1	L2	L0	L1	L2	L0	L1	L2	L0	L1	L2
0	0	0	0	0	0	0	0	0	0	0	0	0
0.1	18.4753	1.864412	4.531127	23.2852	2.733121	6.366238	9.31593	0.888232	2.23391	13.671	1.227732	2.8136
0.2	24.3634	3.167636	7.23298	31.6329	4.454834	9.75907	12.3688	1.369653	3.205056	17.5783	1.899788	4.125327
0.3	29.8085	4.195813	9.240862	40.3166	5.7151	12.01557	16.3947	1.809122	4.097344	22.7201	2.458774	5.347359
0.4	35.7651	5.038863	10.79503	46.4135	6.680899	13.62547	18.3322	2.291653	4.799657	24.9032	2.981049	6.3657
0.5	40.4792	5.74541	12.03523	50.972	7.447304	14.83756	18.6654	2.659709	5.402615	27.0189	3.421913	7.303781

Table 6: L_0, L_1 and L_2 Norms for Numerical Solutions for for 0.0-to-0.5-time step when Δt = 0.005

3.2 Stability Analysis

We will be using Von Neumann Stability Analysis to study the stability of the Implicit Forward Time Central Space method. It is regularly used to determine the stability of Finite Difference equation. In this process, we first use Fourier Series to expand the solution of the given finite difference equation. We then look at the decay factor or the growth of amplification factor (G) to determine if the FDE is stable or not. If max G>1, then the FDE is unstable. If max G<1 the FDE is stable.

Let's look at the steps for Von Neumann Stability Analysis for the Implicit Forward Time Central Space.

First, we will choose the method that we want to use. In our case that is Implicit Forward Time Central space.

Write the numerical solution as combination of analytical plus error.

Substitute the obtained equation into the discrete equation.

As the Analytical solution must be satisfied, it will be cancelled out and only an equation of the residuals will remain.

Now check the ratio between two consecutive time steps. If it is less than one, then the method is stable.

The mathematical derivation for this has been attached in the Appendix of this report.

From our derivation we can see that we plug in our values of Δx and Δt the method is unstable.

4.0 Conclusion

In this assignment we have seen how to solve a given wave equation using 4 discretisation methods. We have plotted graphs for the numerical solutions and compared them to the graphs of the analytical solution. For $\Delta t = 0.01$, Lax-Wendroff has closest solution to analytical but it is unstable so it will not be suitable. Explicit FTBS is stable and should be considered. For $\Delta t = 0.02$, Lax-Wendroff and Explicit FTBS has closest solution to the analytical solution. For $\Delta t = 0.005$, Lax-Wendroff has the closest solution to the analytical solution but again is unstable. Explicit FTBS is the next best option.

We also did Von Neumann Stability analysis for the Implicit FTCS method. By solving the equation we understood that this method is unstable.

References

Hoffmann, K. A., & Chiang, S. T. (n.d.). *Computational fluid dynamics for engineers* (Vol. 1). Wichita : Engineering Education System, 1993.

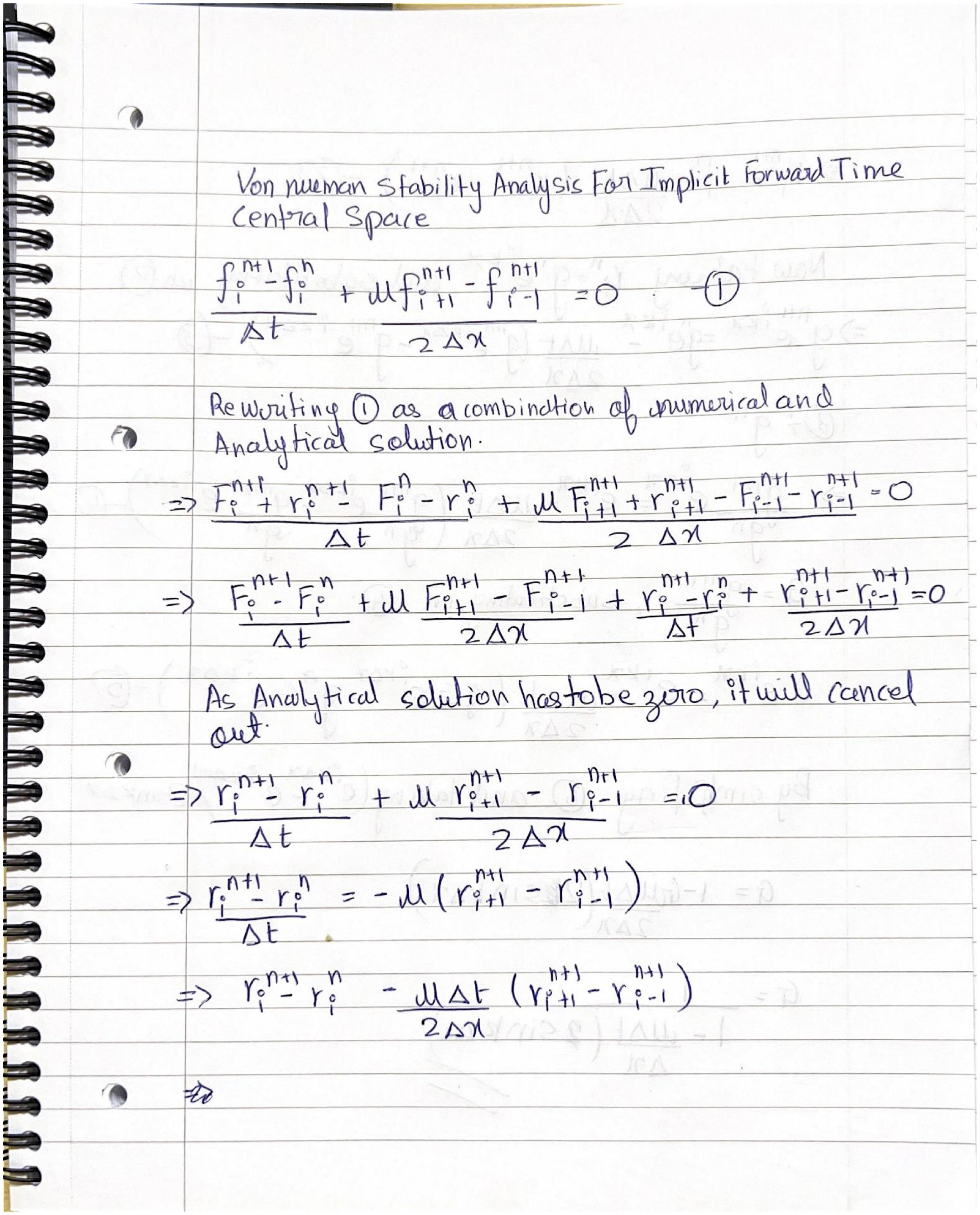
https://www.machinedesign.com/3d-printing-cad/fea-and-simulation/article/21832072/whats-the-difference-between-fem-fdm-and-fvm. (n.d.).

Polyanin, A. D., Schiesser, W. E., & Zhurov, A. I. (2008). Partial differential equation. *Scholarpedia*, *3*(10), 4605. https://doi.org/10.4249/scholarpedia.4605

Zhou, P. (1993). Finite Difference Method. In *Numerical Analysis of Electromagnetic Fields* (pp. 63–94). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-50319-1_3

Appendix I

Von Neumann Stability Analysis for Implicit FTCS



Von nueman Stability Analysis For Implicit Forward Time Central Space

$$\frac{f_i^{n+1}-f_i^{n}}{\Delta t} + \mu \frac{f_{i+1}^{n+1} - f_{i-1}^{n+1}}{2\Delta x} = 0 \quad -①$$

Rewriting ① as a combination of numerical and Analytical solution.

$$\Rightarrow \frac{F_i^{n+1} + r_i^{n+1} - F_i^{n} - r_i^{n}}{\Delta t} + \mu \frac{F_{i+1}^{n+1} + r_{i+1}^{n+1} - F_{i-1}^{n+1} - r_{i-1}^{n+1}}{2\ \Delta x} = 0$$

$$\Rightarrow \frac{F_i^{n+1} - F_i^{n}}{\Delta t} + \mu \frac{F_{i+1}^{n+1} - F_{i-1}^{n+1}}{2\Delta x} + \frac{r_i^{n+1} - r_i^{n}}{\Delta t} + \frac{r_{i+1}^{n+1} - r_{i-1}^{n+1}}{2\Delta x} = 0$$

As Analytical solution has to be zero, it will cancel out.

$$\Rightarrow \frac{r_i^{n+1} - r_i^{n}}{\Delta t} + \mu \frac{r_{i+1}^{n+1} - r_{i-1}^{n+1}}{2\Delta x} = 0$$

$$\Rightarrow \frac{r_i^{n+1} - r_i^{n}}{\Delta t} = -\mu \left(r_{i+1}^{n+1} - r_{i-1}^{n+1} \right)$$

$$\Rightarrow r_i^{n+1} - r_i^{n} \quad - \frac{\mu \Delta t}{2\Delta x} \left(r_{i+1}^{n+1} - r_{i-1}^{n+1} \right)$$

$\Rightarrow$

$$\Rightarrow r_i^{n+1} = r_i^n - \frac{\mu\Delta t}{2\Delta x}\left(r_{i+1}^{n+1} - r_{i-1}^{n+1}\right) \quad - ②$$

Now taking $r_i^n = g^n e^{ikx}$ and substiteting in ②

$$\Rightarrow g^{n+1} e^{ikx} = g^n e^{ikx} - \frac{\mu\Delta t}{2\Delta x}\left(g^{n+1} e^{ik\Delta x} - g^{n+1} e^{-ik\Delta x}\right) \quad - ③$$

$③ \div g^n$

$$\Rightarrow \frac{g^{n+1}}{g^n} e^{ikx} = e^{ikx} - \frac{\mu\Delta t}{2\Delta x}\left(\frac{g^{n+1}}{g^n} e^{ik\Delta x} - \frac{g^{n+1}}{g^n} e^{-ik\Delta x}\right) \quad - ④$$

$G = \frac{g^{n+1}}{g^n}$, substituting in ④.

$$G e^{ikx} = e^{ikx} - \frac{\mu\Delta t}{2\Delta x}\left(\cancel{g} G e^{ik\Delta x} - \cancel{g} G e^{-ik\Delta x}\right) \quad - ⑤$$

By simpifing ⑤ and takinp $\left(e^{ik\Delta x} - e^{-ik\Delta x}\right) = 2\sin k\Delta x$

$$G = 1 - G\frac{\mu\Delta t}{2\Delta x}\left(2\sin k\Delta x\right)$$

$$G = \frac{1}{1 + \frac{\mu\Delta t}{\Delta x}\left(\cancel{2}\sin k\Delta x\right)}$$

Appendix II

INDIVIDUAL CONTRIBUTIONS

Amit’s Contribution

I had a great time doing this assignment. Tejas helped me a lot to understand C++ programming as some concepts were new to me. We both wrote code for two methods each and divided the report and worked on it together.

Tejas’s Contribution

This assignment was great learning experience for me. Amit helped me understand the theory of the methods and how to solve them. I helped him understand the coding part and we worked on the report together.

Appendix III

Code

```
//STUDENT CODE: 387998 AND 375967


//fn0 = f^n+1 && fn = f^n
#include <iostream>
#include <vector>
#include <cmath>
#define TEMP 79

using namespace std;

vector<double> x;
    double Min = 0.0;
    double Max = 400.0;
    double delta_x = 5.0;
    double delta_t = 0.02;
    double T = 0.5;
    double N = T / delta_t;
    double diff = Max/delta_x;
    double t;
    double c;
    double u = 250;

double calculateAnalyticalValues(vector<double> fn, double i, double
t, double max){
     double k = fn.at(i);

     if(k >= 0 && k <= 50){
          k = 0.0;
     }
     else if (k > 50 + 250 * t && k <= 110 + 250 * t)
    {
        k = 100 * sin(3.14159265358979323846 * (k - 50 - 250 * t) /
60);
    }
    else if (k > 110 + 250 * t && k <= max)
    {
        k = 0.0;
    }
    else{
     k = 0.0;
     }
     return k;
}

void AnalyticalSolution(){
     vector<double> fn0, fn1, fn;
    fn = x;
    fn.at(0) = 0;
    fn.at(diff) = 0;

    fn0 = fn;
    fn1 = fn0;

     for(auto j = 0; j<=N; j++){
          t = j * delta_t;

          for(auto i=0; i<x.size(); ++i)
        {
            double k = calculateAnalyticalValues(fn, i, t, Max);
            cout<<k<<" ";
        }

        cout<<endl;
     }

}

//----------------------------------------------------------------------
-----------------------------------

double calculateInitialBoundaryValues(vector<double> fn, double i,
double max){
     double k = fn.at(i);

     if(k >= 0 && k <= 50){
          k = 0.0;
     }
     else if (k > 50 && k <= 110)
    {
        k = 100 * sin(3.14159265358979323846 * (k - 50) / 60);
    }
    else if (k > 110 && k <= max)
    {
        k = 0.0;
    }
    else{
     k = 0.0;
     }
     return k;
}

//----------------------------------------------------------------------
```

```
------------------------------------

double calculateExplicitUpwindFTBS(vector<double> fn, vector<double>
fn0, double i, double max){

     return fn.at(i) - ((u * delta_t )/ delta_x ) * (fn.at(i) -
fn.at(i-1));
}

void ExplicitUpwindFTBS(){

     vector<double> fn0, fn1, fn;
    fn = x;
    fn.at(0) = 0;
    fn.at(diff) = 0;

    fn0 = fn;
    fn1 = fn0;

     for(auto i=0; i<x.size(); ++i)
        {
            double k = calculateInitialBoundaryValues(fn, i, Max);
            fn0.at(i) = k;
            cout<<" "<<k;
        }

        cout<<endl;

          fn = fn0;
     for(auto j = 1; j<=N; j++){

          cout<<" "<<fn.at(0);
        for(auto i=1; i<fn0.size()-1; i++)
        {
          fn.at(0) = 0.0;
          fn.at(diff) = 0.0;

          fn0.at(0) = 0.0;
          fn0.at(diff) = 0.0;
            double k = calculateExplicitUpwindFTBS(fn, fn0, i, Max);
// x->fn, fn->x, fn0 = initial, fn = x, fn1 = future;
              if(i==diff){
              k = 0.0;
          fn0.at(i) = k;
              }
              else{
                   fn0.at(i) = k;
              }
```

```
               cout<<" "<<k;
          }
      cout<<" "<<fn.at(diff);
          fn = fn0;
          fn0 = fn1;
          cout<<endl;
      }

}

//------------------------------------------------------------------
----------------------------------

double calculateImplicitUpwindFTBS(vector<double> fn, vector<double>
fn0, double i, double max){

     return u * delta_t/(delta_x + u * delta_t)* (fn0.at(i - 1)) +
delta_x/(delta_x + u * delta_t) * fn.at(i);
}

void ImplicitUpwindFTBS(){

     vector<double> fn0, fn1, fn;
    fn = x;
    fn.at(0) = 0;
    fn.at(diff) = 0;

    fn0 = fn;
    fn1 = fn0;


     for(auto i=0; i<x.size(); ++i)
        {
            double k = calculateInitialBoundaryValues(fn, i, Max);
            fn0.at(i) = k;
            cout<<" "<<k;
        }

        cout<<endl;

          fn = fn0;
     for(auto j = 1; j<=N; j++){

          cout<<" "<<fn.at(0);
        for(auto i=1; i<fn0.size()-1; i++)
        {
          fn.at(0) = 0.0;
          fn.at(diff) = 0.0;
          fn0.at(0) = 0.0;
          fn0.at(diff) = 0.0;
            double k = calculateImplicitUpwindFTBS(fn, fn0, i, Max);
// x->fn, fn->x, fn0 = initial, fn = x, fn1 = future;
               if(i==diff){
               k = 0.0;
          fn0.at(i) = k;
               }
               else{
                    fn0.at(i) = k;
               }
               cout<<" "<<k;
        }
     cout<<" "<<fn.at(diff);
        fn = fn0;
        fn0 = fn1;
        cout<<endl;
     }

}

//----------------------------------------------------------------------
----------------------------------

double calculateLaxWendroff(vector<double> fn, vector<double> fn0,
double i, double max){

     return ((delta_t*delta_t)*(u*u))/(2*delta_x*delta_x) *
(fn.at(i+1)-2*fn.at(i)+fn.at(i-1)) - u*delta_t/(2*delta_x) *
(fn.at(i+1)-fn.at(i-1))+fn.at(i);
}

void LaxWendroff(){

     vector<double> fn0, fn1, fn;
    fn = x;
    fn.at(0) = 0;
    fn.at(diff) = 0;

    fn0 = fn;
    fn1 = fn0;

     for(auto i=0; i<x.size(); ++i)
        {
            double k = calculateInitialBoundaryValues(fn, i, Max);
            fn0.at(i) = k;
            cout<<" "<<k;
        }

        cout<<endl;

          fn = fn0;
          for(auto j = 1; j<=N; j++){

          cout<<" "<<fn.at(0);
        for(auto i=1; i<fn0.size()-1; i++)
        {
          fn.at(0) = 0.0;
          fn.at(diff) = 0.0;

          fn0.at(0) = 0.0;
          fn0.at(diff) = 0.0;
             double k = calculateLaxWendroff(fn, fn0, i, Max); // x-
>fn, fn->x, fn0 = initial, fn = x, fn1 = future;
                if(i==diff){
                k = 0.0;
          fn0.at(i) = k;
                }
                else{
                     fn0.at(i) = k;
                }
                cout<<" "<<k;
        }
    cout<<" "<<fn.at(diff);
        fn = fn0;
        fn0 = fn1;
        cout<<endl;
    }

}

//--------------------------------------------------------------------
----------------------------------

void calculateImplicitFTCS(vector<double>& a, vector<double>& b,
     vector<double>& c,vector<double>& d, vector<double>& x) {
    c.at(0) = c.at(0) / b.at(0);
    d.at(0) = d.at(0) / b.at(0);
    for (int i = 1; i < TEMP - 1; i++){
     c.at(i) = c.at(i) / (b.at(i) - c.at(i-1) * a.at(i));
     }
    for (int i = 1; i < TEMP; i++){
     d.at(i) = (d.at(i) - d.at(i-1) * a.at(i)) / (b.at(i) - c.at(i-1)
* a.at(i));
     }
    x[TEMP - 1] = d[TEMP - 1];
    for (int i = TEMP - 2; i >= 0; i--) {
        x.at(i) = d.at(i) - c.at(i) * x.at(i+1);
    }

}

void ImplicitFTCS(){
     N = (T + 1e-6) / delta_t;
     diff = (Max + 1e-6) / delta_x;

     vector<double> fn0, fn1;
    fn0 = x;
    fn1 = x;

    vector<double> z(diff + 1,0);
    vector<vector<double>> fn(N + 1, z);

     vector<double> a (TEMP);
     vector<double> b (TEMP);
     vector<double> c (TEMP);
     vector<double> d (TEMP);
     vector<double> x (TEMP);
     fill (a.begin(),a.begin()+TEMP,(-u * delta_t));
     fill (b.begin(),b.begin()+TEMP,(2 * delta_x));
     fill (c.begin(),c.begin()+TEMP,(u * delta_t));

    for(auto i=0; i<fn1.size(); ++i)
        {
            double k = calculateInitialBoundaryValues(fn1, i, Max);
            fn0.at(i) = k;
            fn[0][i] = k;
        }

    for (int i = 1; i <= N; i++)
    {
     fill (c.begin(),c.begin()+TEMP,(u * delta_t));

        for (int j = 0; j < TEMP; j++)
        {
            d.at(j) = 2 * delta_x * fn[i-1][j+1];
        }

        calculateImplicitFTCS(a, b, c, d, x);

        for (int j = 1; j <= TEMP; j++)
        {
            fn[i][j] = x.at(j-1);
        }

    }
     for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < diff; j++)
        {
            cout << fn[i][j] << " ";
        }
        cout << "\n";
    }
}

//----------------------------------------------------------------------
----------------------------------

int main()
{
    int ch1,ch2;

     //adding Total no of values into the vector
(0,5,10,15,20.....400)
        for(float j = 0; j<=diff; ++j){
              double temp = j;
              temp = temp * delta_x;
              x.push_back(temp);
        }

   //end here-------------------------------------------

    cout<<"\n ------------------- Select Computation Time -----------
--------";
    cout<<"\n 1: Delta_t = 0.02sec";
    cout<<"\n 2: Delta_t = 0.01sec";
     cout<<"\n 3: Delta_t = 0.005sec";
     cout<<"\n 4: Exit";
     cout<<"\n Enter Your Choice : ";
     cin>>ch1;

    switch(ch1)
     {
          case 1:
                    delta_t = 0.02;
                    N = T / delta_t;
               cout<<"\n   --------------------- * ------------------
---   "<<endl;
               cout<<"\n 1: Explicit Upwind FTBS (Forward time,
Backward space)";
               cout<<"\n 2: Implicit Upwind FTBS (Forward time,
Backward space)";
               cout<<"\n 3: Lax-Wendroff";
               cout<<"\n 4: Implicit FTCS (Forward time, Central
space)";
               cout<<"\n 5: Analytical Solution";
               cout<<"\n 6: Exit";
               cout<<"\n Enter Your Choice : ";
               cin>>ch2;

               switch(ch2)
                    {
                         case 1:
                                   cout<<endl;
                                   ExplicitUpwindFTBS();
                                   break;
                         case 2:
                                   cout<<endl;
                                   ImplicitUpwindFTBS();
                                   break;
                         case 3:
                                   cout<<endl;
                                   LaxWendroff();
                                   break;
                         case 4:
                                   cout<<endl;
                                   ImplicitFTCS();
                                   break;
                         case 5:
                                   cout<<endl;
                                   AnalyticalSolution();
                                   break;
                         case 6:
                                   exit(1);
                         default:
                                   cout<<"\n Wrong Choice!! Please Try
Again.. ";
                    }

                    break;
          case 2:
               delta_t = 0.01;
               N = T / delta_t;
               cout<<"\n   --------------------- * ------------------
---   "<<endl;
               cout<<"\n 1: Explicit Upwind FTBS (Forward time,
Backward space)";
```

```
               cout<<"\n 2: Implicit Upwind FTBS (Forward time,
Backward space)";
               cout<<"\n 3: Lax-Wendroff";
               cout<<"\n 4: Implicit FTCS (Forward time, Central
space)";
               cout<<"\n 5: Analytical Solution";
               cout<<"\n 6: Exit";
               cout<<"\n Enter Your Choice : ";
               cin>>ch2;

               switch(ch2)
                    {
                         case 1:
                                   cout<<endl;
                                   ExplicitUpwindFTBS();
                                   break;
                         case 2:
                                   cout<<endl;
                                   ImplicitUpwindFTBS();
                                   break;
                         case 3:
                                   cout<<endl;
                                   LaxWendroff();
                                   break;
                         case 4:
                                   cout<<endl;
                                   ImplicitFTCS();
                                   break;
                         case 5:
                                   cout<<endl;
                                   AnalyticalSolution();
                                   break;
                         case 6:
                                   exit(1);
                         default:
                                   cout<<"\n Wrong Choice!! Please Try
Again.. ";
                    }

                    break;
          case 3:
                    delta_t = 0.005;
                    N = T / delta_t;
               cout<<"\n   -------------------- * ------------------
---   "<<endl;
               cout<<"\n 1: Explicit Upwind FTBS (Forward time,
Backward space)";
               cout<<"\n 2: Implicit Upwind FTBS (Forward time,
Backward space)";
               cout<<"\n 3: Lax-Wendroff";
               cout<<"\n 4: Implicit FTCS (Forward time, Central
space)";
               cout<<"\n 5: Analytical Solution";
               cout<<"\n 6: Exit";
               cout<<"\n Enter Your Choice : ";
               cin>>ch2;

               switch(ch2)
                    {
                         case 1:
                                   cout<<endl;
                                   ExplicitUpwindFTBS();
                                   break;
                         case 2:
                                   cout<<endl;
                                   ImplicitUpwindFTBS();
                                   break;
                         case 3:
                                   cout<<endl;
                                   LaxWendroff();
                                   break;
                         case 4:
                                   cout<<endl;
                                   ImplicitFTCS();
                                   break;
                         case 5:
                                   cout<<endl;
                                   AnalyticalSolution();
                                   break;
                         case 6:
                                   exit(1);
                         default:
                                   cout<<"\n Wrong Choice!! Please Try
Again.. ";
                    }
                    break;
          case 4:
                    exit(1);
          default:
                    cout<<"\n Wrong Choice!! Please Try Again.. ";
     }

    return 0;
}
```