

**Computional Methods**

**C++**

**Assignment**

***Author:***

*Wiktor Bednarek*

***Supervisors:***

*Dr Irene Moulitsas*

*Dr Peter Sherar*

December 17, 2016

**Abstract**

Thanks to growing branch of science which is Computational fluid dynamics there are many numerical methods provided.

This paper will discuss mathematical description and results of 4 schemes, following schemes will be considered:

* Explicit Upwind
* Implicit Upwind
* Lax-Wendroff
* Richtmyer multi-step

Together with this paper C++ program has been created. It gives as output results of computation of 4 above mentioned methods. It will be estimated which gives the most exact results. Also will be explained why there is an problem to choose one best method since each of them is works better for different inputs. Crucial variable is Courant–Friedrichs–Lewy (CFL) condition which is responsible for output qualty. For all schemes there are different stability conditions, that aspect will be covered as well.

Code is written in C++ and is object-oriented. Also it uses modern C++ methods and features to make it easy to manage and remaining high performance. UML diagram shows connections and dependencies between program classes.

Spis treści

Wpisz tytuł rozdziału (poziom 2)2

Wpisz tytuł rozdziału (poziom 3)3

**Wpisz tytuł rozdziału (poziom 1)4**

Wpisz tytuł rozdziału (poziom 2)5

Wpisz tytuł rozdziału (poziom 3)6

# **Introduction**

Numerical methods allows to solve problems which are not described by any function.

Using C++ language it is possible to perform this task effectively. First significant publication about Computational methods was created by Richard Courant, Kurt Friedrichs, and Hans Lewy in 1928. This scientist’s paper brought CFL (Courant–Friedrichs–Lewy) condition which is one of the most important values in C++ project created to present results in following paper. Numerical methods have many applications one essential is solving differential equations and differential systems equations. Those methods also helps to find approximation of unknown function and find zero points in polynomials when it’s degree is greater than two.

Methods used in this report have widely usage in CFD (Computational fluid dynamics). This branch of science describes fluid flow problem. Numerical methods in CFD helps to approximate various fluid parameters like: pressure, temperature, flow velocity or density. Those parameters are calculated using functions of space and time.

There are number of factors that are responsible for quality of results. Probably most common are precision errors. This error is result of way in which computers stores floating point values, because limited precision, practically in all cases of computation (especially in performing operations like multiplication and division) numbers with fractional parts ≠ 0 will occur errors . Second type of error is truncation error. Computers are not performing computations in continuous domain. Mathematical formulas need to be discretized what brings error because data are lost during that process. No matter how small is value between two next points in discretized function there will be always error as far function is not continuous.

# **Used methods**

* 1. **Initial conditions**

There is given advection equation or wave equation (one dimensional):

(1)

Above equation is used for motion of one-dimesional description, u stands for speed.

Direction of wave depend on u value:

(2)

Equation (1) linear advection (one-dimensional), when x is direction and u is speed. When u is greater than 0 so wave moves in space axis direction (upstream direction). In this report this case is considered.

To solve all schemes it is needed to have initial boundaries conditions. Two sets of conditions are considered:

First set

This boundary condition is based on sign function which is described as follow:

)

Function in space and time is considered ,

where x is space point and t is time point.

When time is equals 0 following boundaries equations are given:

()

Second set

Space initialization function is based on exponential function:

)

)

()

For each initial boundary conditions set analytical solutions are given respectively:

For first set:

()

For second set:

)

Also there is given speed condition:

)

And space domain:

()

* 1. **Explicit Upwind Scheme**

Upwind Schemes is one of numerical discretization methods. Upwind schemes is used for solving hyperbolic partial differential equations.

First order Upwind scheme is described by following equations:

)

()

For program purposed and since u > 0 in this report case only first equation was used to computation. From above scheme equation it is noticeable that method subdivides time and space respectively by and .

.

For stability reasons Courant number is important. Method gives stable results for following condition:

()

Where

)

Explicit Upwind Scheme gives the best results for Courant number which is closest to 1.

* 1. **Implicit Upwind Scheme**

The essential fact about Implicit Upwind Scheme is unconditional stability of this method. It also allows bigger time steps, so to get results less computations are needed comparing to other solutions. Implicit Upwind Scheme is first order accurate as well in space and in time.

Below equation describes that scheme:

)

After transformations described in Appendix following for was used for implementation:

()

Since algorithm is based on previous values the same as Explicit upwind scheme boundaries initialization are necessary for computation.

* 1. **Lax-Wendroff scheme**

Lax-Wendroff scheme is useful when it comes to approximate solutions of hyperbolic partial differential equations. It is second order accurate in time and space either. It’s current instant depends on only previous step value as in other differential schemes.

This Scheme is described by following equation:

)

* 1. **Richtmyer multi-step scheme**

Richtmyer is also called two-step Lax–Wendroff method[4]. In the first step Richtmyer method values for f(u(x, t)) at half time steps (half time steps) and (half grid points are calculated. Also Richtmyer is considered as Jacobian free method. General approach to that method is to calculate it in two steps.

For Richtmyer multi-step scheme stability (Equation 21) condition described by Counant umber is following:

(21)

The same as in previous methods because of it’s predictive character initial boundaries conditions are required to perform computation.

* 1. **Conditions and norms**
     1. **Courant–Friedrichs–Lewy (CFL) condition**

In 1928 Richard Courant, Kurt Friedrichs, and Hans Lewy introduced their paper about computational methods *On the partial difference equations of mathematical physics[1]*. CFL value was introduced in that publication.

Considering one-dimensional case CLFhas it’s general form is as following;

()

**Where**

– time step

– step in space domain

Cmax depends on actual scheme type there are different Cmax values for Explicit Upwind Scheme and for Richtmyer multi-step scheme. The usual value for Cmax is 1, while in Richtmyer multi-step scheme this value equals 2.

When it comes to implementation calculation of CFL is important to know that it is based on Courant number (C, left part of equation).

There are couple conditions for CFL and it is necessary to define following quantities:

1. Time – it shows how system behaves in time
2. Spatial coordinate – defines points in space for problem
3. Spatial problem dimension - represents number n of spatial dimensions
   * 1. **Von Neumann stability**

This approach determines maximum value which provides stable solution. It is defined by dividing two next time values and when result of that operation is greater than stability condition (which is 1) . Below equation represents stability calculation:

()

Stability requirement is as follow:

)

There are many cases when above solution may give false results, for instance G factor is complex number. In that case result from above (25) may be false, that’s why it is often more reasonable to use squared absolute value such cases.

)

* + 1. **Errors and norms**

There are many ways to cause mistake in program. One of the most common is programmer mistake which is often result of inattention. Really often occur mistakes like exceed scope of vector, wrong reference to object, wrong reference to specific element (it happens when programmer tries to access element out of array/vector scope) and typical logic mistakes.

Next errors source is limitation of variables precision. All floating point primitive types (such as float or double) have limited number of values after comma (fractional part). Especially when a lot of multiplications or divisions are performed bigger error will occur.

Considering computer computation there is always discretization error. All operations which computer performs is discrete that why we are losing data each time when we discretizing some continuous function. Also infinite value in computation is result of specific agreement, there are always limited resources for example memory. That king of inconvenience occurs often when functions that’s operate on infinite domain are considered. One of example is Fourier transform because of limited domain undesirable effect like aliasing occurs.

There are plenty of methods that helps to compare two or more data sets. On o them is for instance correlation and convolution. But in program which was made to this report really useful indicators are vector/matrix norms. They’re help to estimate solution quality.

For this exercise purposes three types of norm will be presented:

Uniform norm:

)

Two norm:

)

Second norm is defined as sum of vector’s elements absolute values.

Three norm:

)

Third norm is defined as sum of vector’s squared absolute values.

1. **Results**
   1. **Analytical solution**

Here analytical solution results are presented.

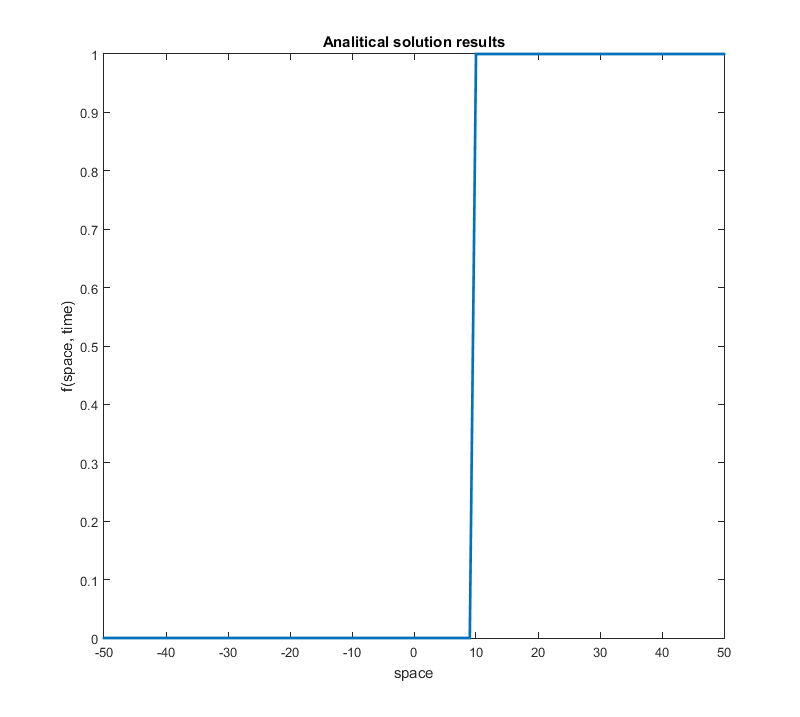


Fig. 1 Analytical solution results for sign type of initial boundary t=5, CFL = 0.999, number of points = 100

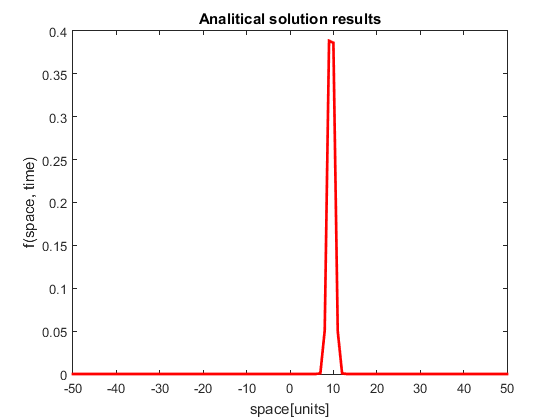


Fig. 2 Analytical solution results for exponential type of initial boundary t=5, CFL = 0.999, number of points = 100

As can be observed output of analytical solution function look very different form analytical solution despite of function is the same in both cases. That means the input data (initial boundaries in this case) naturally dictating output of function.

On the other hand there is significant similarity between those two outputs behavior. Functions soar about point 10 of space in sign and exponential boundary type. Nevertheless in first case from that point it remain at the same level until the end of space scope, but in second graph we can observe that function quickly returns to its initial state after point 10.

* 1. **Explicit Upwind Scheme**
     1. **Results for sign type of boundary conditions**

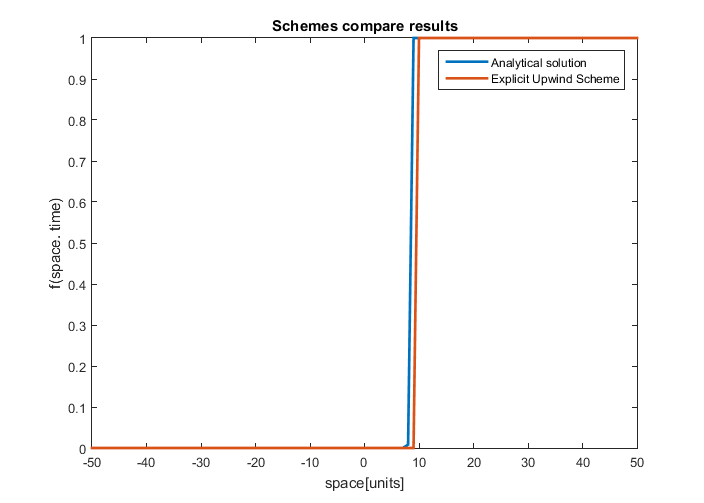
****

Fig. 3 Comparing Analytical solution and Explicit Upwind Scheme results for sign type of initial boundary t=5, CFL = 0.999, number of points = 100

Checking results for different Courant number (C = { 0.25 ,0.5, 0.75, 0.999, 1.25, 1.5, 1.75, 2 }

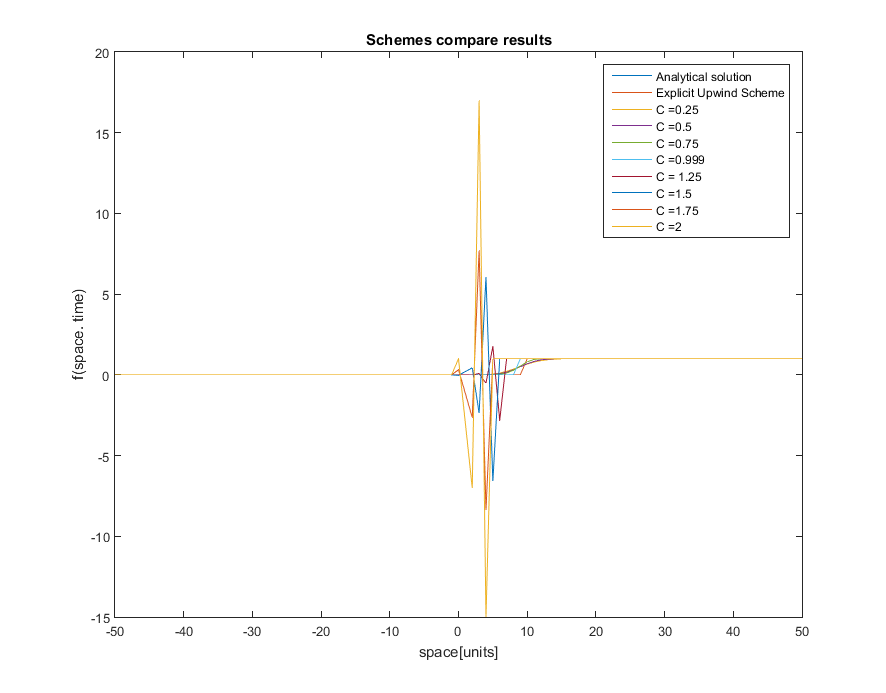


Fig. 4 Comparing Analytical solution and Explicit Upwind Scheme results for sign type of initial boundary t=5, CFL = { 0.25 ,0.5, 0.75, 0.999, 1.25, 1.5, 1.75, 2}, number of points = 100

As can be observed for C > 1 huge instability occurs. Results of this experiment is consistent with theory where .

Next step is to check results for all for better graph visibility.

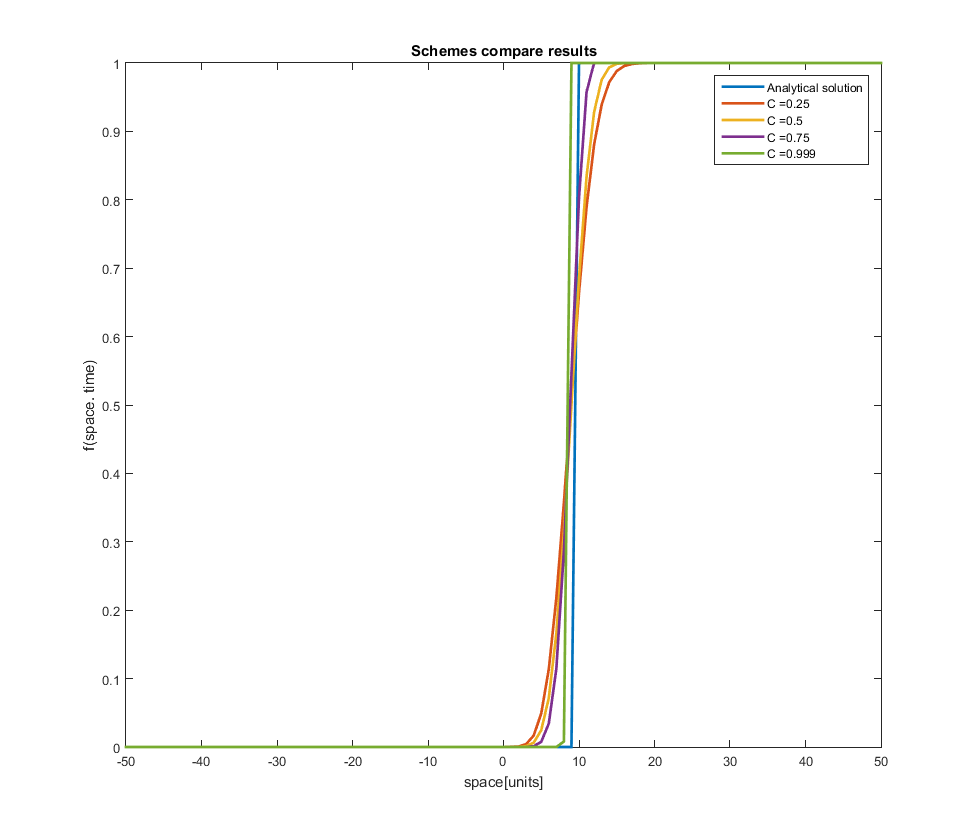


Fig. 5 Comparing Analytical solution and Explicit Upwind Scheme results for sign type of initial boundary t=5, CFL = { 0.25 ,0.5, 0.75, 0.999}, number of points = 100. Stable results.

Table 1 Norms values depending on Courant number in Explicit Upwind Scheme. Data results for t=5, CFL = { 0.25 ,0.5, 0.75, 0.999}, number of points = 100.

|  |  |  |  |
| --- | --- | --- | --- |
| **Courant Number** | **Infinite norm** | **Norm one** | **Norm two** |
| 0,25 | 0,663967 | 0,0237407 | 0,00978942 |
| 0,5 | 0,685471 | 0,0204018 | 0,00941058 |
| 0,75 | 0,544799 | 0,0122866 | 0,00658793 |
| 0,999 | 0,262517 | 0,01008 | 0,0100003 |
| 1,25 | 2,8147 | 0,0819531 | 0,0378341 |
| 1,5 | 6,59375 | 0,195 | 0,0949023 |
| 1,75 | 8,37891 | 0,230313 | 0,118548 |
| 2 | 17 | 0,45 | 0,238537 |

According to Table 1 as Courant number is closer to 1 then results are better, errors are lower. Norms values for C > 1 are significantly bigger and growing fast.

Now checking Explicit Upwind Scheme behavior for different time and number of space points.

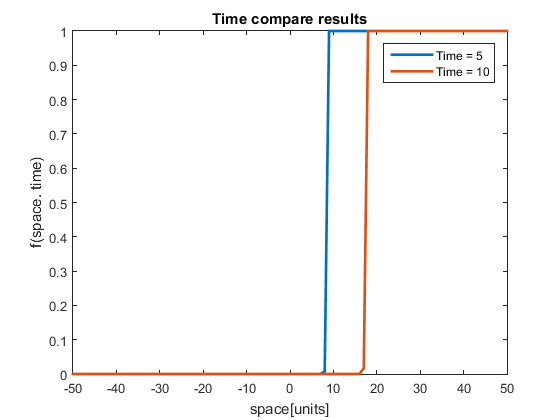


Fig. 6 Comparing Explicit Upwind Scheme results for different times, sign type of initial boundary t=5 and 10, CFL = 0.999, number of points = 100.

For higher time value Explicit Upwind Scheme results is shifted to the right in space domain. Whole function shape remained same.

Finally compare results that scheme for different number of points.

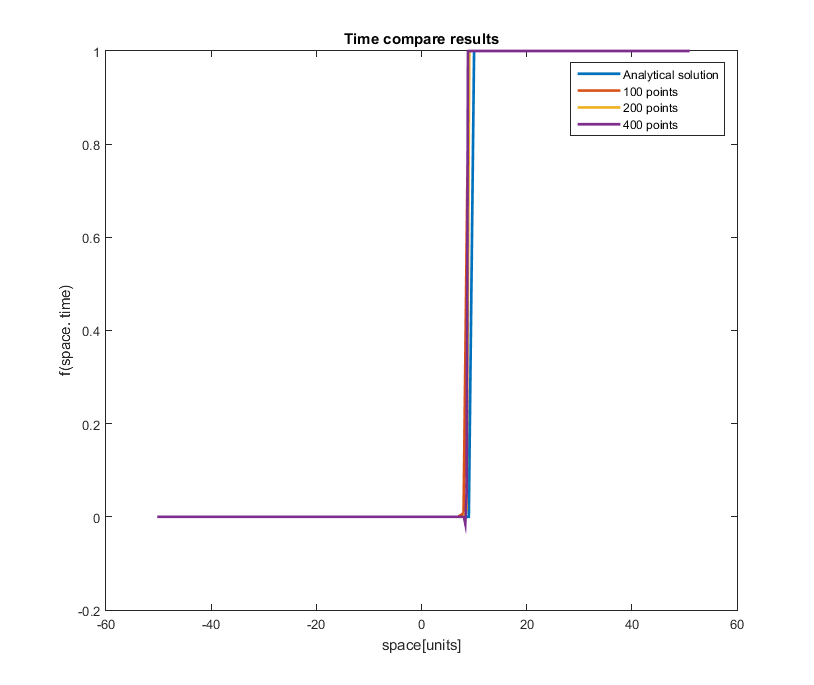


Fig. 7 Comparing Explicit Upwind Scheme results for different number of points, sign type of initial boundary t=5, CFL = 0.999, number of points = 100, 200, 400

Table 2 Norms values depending on number of points in Explicit Upwind Scheme. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of points in space** | **Infinite norm** | **Norm one** | **Norm two** |
| 100 | 0,0715698 | 0,01008 | 0,0100003 |
| 200 | 0,0168647 | 8,50E-05 | 8,43E-05 |
| 400 | 0,0334449 | 8,50E-05 | 8,36E-05 |

More points gives better results comparing case with 100 and 200 points. Between 200 and 400 points in space there is no significant difference. Overall for 200 and 400 points errors are really small.

* + 1. **Results for exponential type of boundary conditions**

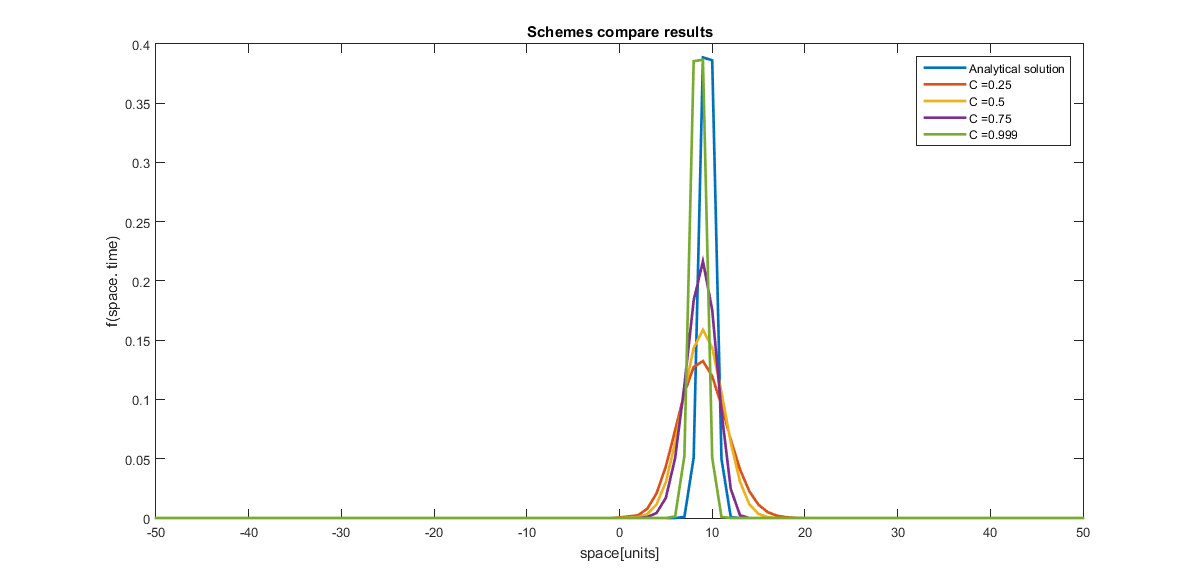
****

Fig. 8 Comparing Explicit Upwind Scheme results for different number of points, exponential type of initial boundary t=5, CFL { 0.25 ,0.5, 0.75, 0.999}, number of points = 100

Table 3 Norms values depending on number of points in Explicit Upwind Scheme, exponential boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Courant Number** | **Infinite norm** | **Norm one** | **Norm two** |
| 0,25 | 0.380291 | 0.0102704 | 0.00439509 |
| 0,5 | 0.35623 | 0.0090336 | 0.00414326 |
| 0,75 | 0.294644 | 0.00740797 | 0.00364687 |
| 0,999 | 0.335505 | 0.00774656 | 0.00479188 |

Results are curious is in spite of the fact the chart for C = 0.75 look completely different from output for C = 0.99 which is more similar to Analytical solution, norms for these two outputs are really similar. It could be computation error or confirmation of fact that visual similarity could be deceptive.

**Compare this case for different time**

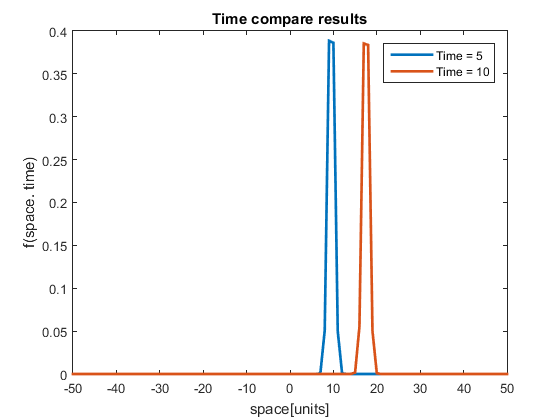
****

Fig. 9 Comparing Explicit Upwind Scheme results for different times, exponential type of initial boundary t=5 and 10, CFL = 0.999, number of points = 100.

**Finally Compare this case for different number of points**

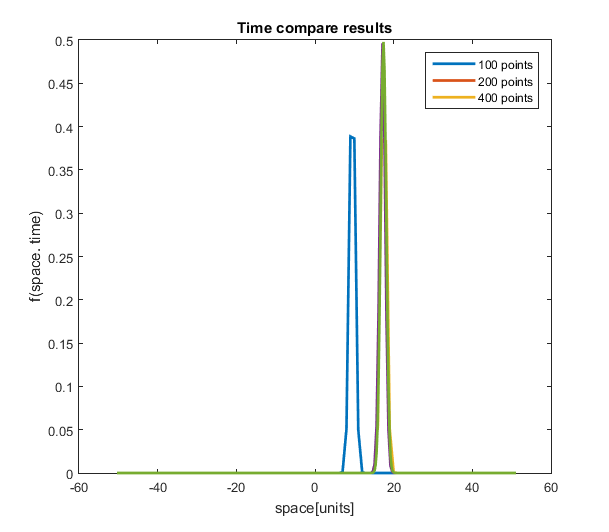
****

Fig. 10 Comparing Explicit Upwind Scheme results for different times, exponential type of initial boundary t=10, CFL = 0.999, number of points = 100, 200 and 400.

Table 4 Norms values depending on number of points in Explicit Upwind Scheme with exponential type of initial boundary. Data results for t=10, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of points in space** | **Infinite norm** | **Norm one** | **Norm two** |
| 100 | 0.333254 | 0.00778686 | 0.0047627 |
| 200 | 0.204957 | 0.00496823 | 0.00190988 |
| 400 | 0.105701 | 0.00249042 | 0.000694047 |

Table 4 shows that as number of points increase norm value getting smaller value

* 1. **Lax-Wendroff Scheme**
     1. **Results for sign type of boundary conditions**

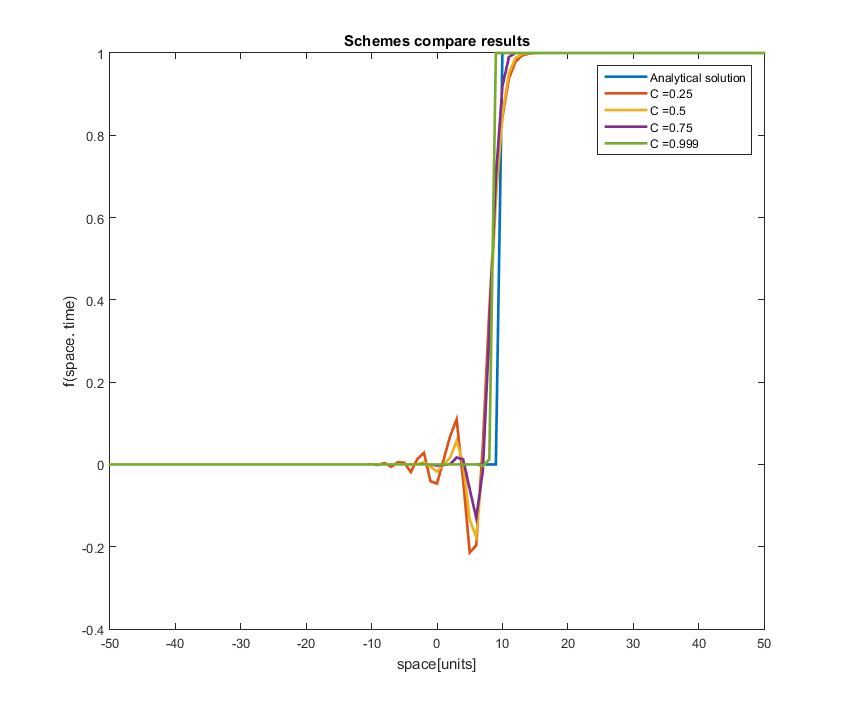
****

Fig. 11 Comparing Analytical solution and Lax-Wendroff scheme results for sign type of initial boundary t=5, C = { 0.25 ,0.5, 0.75, 0.999}, number of points = 100. Stable results.

**Fig. 11 Shows Lax-Wendroff scheme behavior.** Near point of fast value change scheme tries to stabilize before that point. It works different than previous schemes. For C = 0.99 there is no significant stabilization.

Table 5 Norms values depending on Courant number in Lax-Wendroff scheme, sign boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Courant Number** | **Infinite norm** | **Norm one** | **Norm two** |
| 0,25 | 0.843482 | 0.0280907 | 0.0118393 |
| 0,5 | 0.850934 | 0.0230004 | 0.0113918 |
| 0,75 | 0.696607 | 0.013489 | 0.00785805 |
| 0,999 | 1 | 0.0101588 | 0.0100008 |

**Checking Lax-Wendroff results for time change**

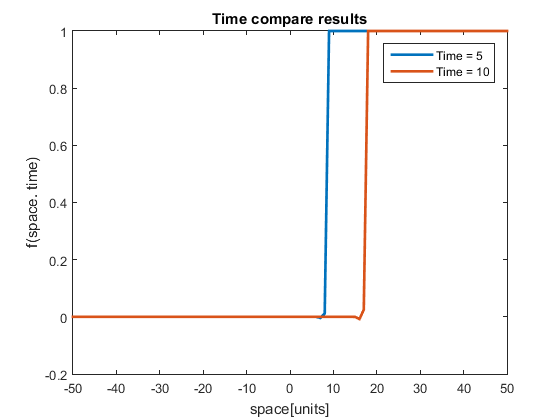
****

Fig. 12 Fig. 6 Comparing Lax-Wendroff results for different times, sign type of initial boundary t=5 and 10, CFL = 0.999, number of points = 100.

According to expectations for higher time function is shifted to the right in space domain. Also it is worth to notice that function tries to stabilize before point of sudden values change .

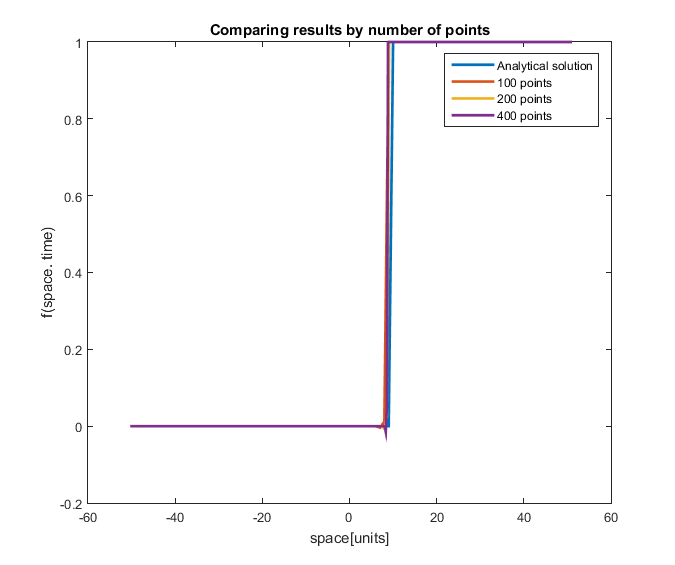


Fig. 13 Lax-Wenfroff results for different times, sign type of initial boundary t=5, CFL = 0.999, number of points = 100, 200 and 400.

It is really hard to visually distinct which solution is the best. For each number of points solutions seems to be accurate.

Fig. 14 Norms values depending on number of points in Lax-Wendroff scheme, sign boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of points in space** | **Infinite norm** | **Norm one** | **Norm two** |
| 100 | 0.673435 | 0.0101588 | 0.0100008 |
| 200 | 0.025188 | 0.00016719 | 0.000132128 |
| 400 | 0.0497415 | 0.000164281 | 0.000129863 |

Lax-Wendroff is Characterised by good results for sign initial boundary condition. For 400 we have really low norms values. Results of this approach are worse than Explicit Upwind Scheme but still they could be considered as high quality results.

* + 1. **Results for exponential type of boundary conditions**

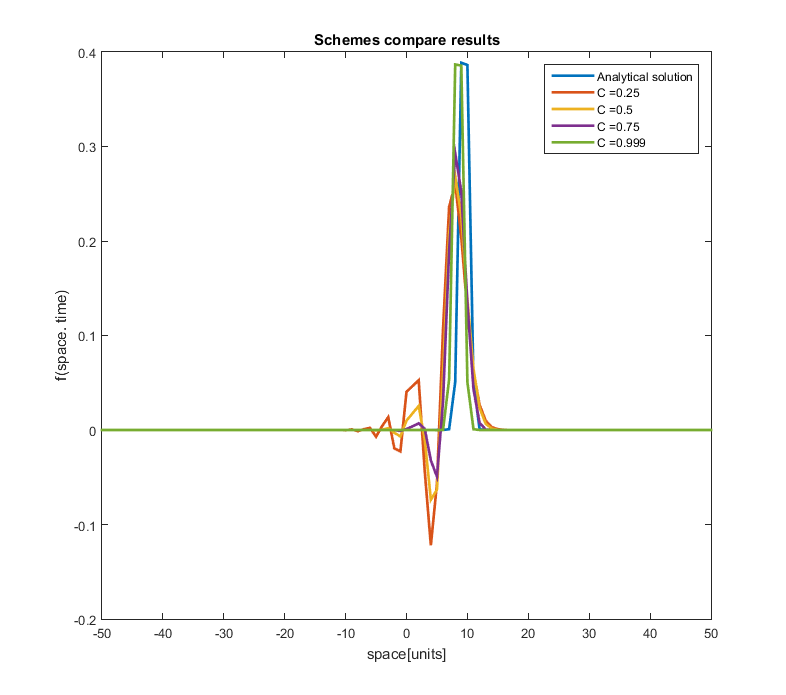
****

Fig. 15 Comparing Lax-Wendroff Scheme results for different number of points, exponential type of initial boundary t=5, CFL { 0.25 ,0.5, 0.75, 0.999}, number of points = 100

Table 6 Norms values depending on number of points in Lax-Wendroff Scheme, exponential boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Courant Number** | **Infinite norm** | **Norm one** | **Norm two** |
| 0,25 | 0.375093 | 0.0153503 | 0.00558898 |
| 0,5 | 0.361667 | 0.0127334 | 0.00521556 |
| 0,75 | 0.336339 | 0.0101199 | 0.00481173 |
| 0,999 | 0.336012 | 0.00777735 | 0.00480413 |

Similar to sign type of scheme this time scheme tries to adapt to Analytical solution as well. Errors are quite small but still higher than in Explicit Upwind Scheme.

**Compare this case for different time**

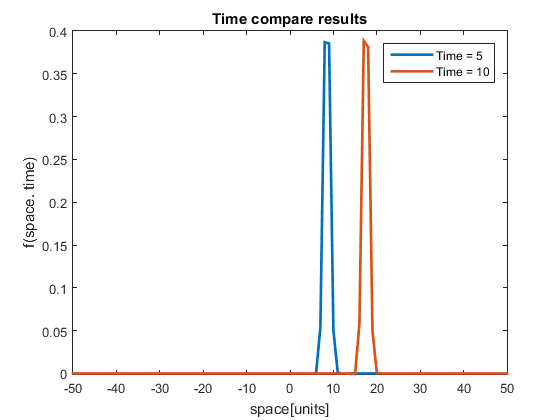
****

Fig. 16 Lax-Wendroff Scheme results for different times, exponential type of initial boundary t=5 and 10, CFL = 0.999, number of points = 100.

In this case expected behavior can be observed. Higher time results are on higher point in space domain.

**Finally Compare this case for different number of points**

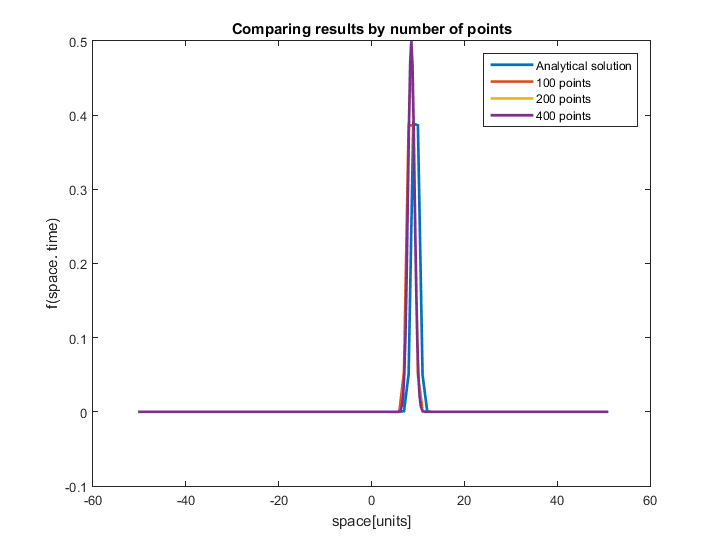
****

Fig. 17 Lax-Wendroff Scheme results for different times, exponential type of initial boundary t=5, CFL = 0.999, number of points = 100, 200 and 400.

Table 7 Norms values depending on number of points in Lax-Wendroff Scheme with exponential type of initial boundary. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of points in space** | **Infinite norm** | **Norm one** | **Norm two** |
| 100 | 0.336012 | 0.00777735 | 0.00480413 |
| 200 | 0.207711 | 0.00500207 | 0.0019291 |
| 400 | 0.106051 | 0.00250254 | 0.000698759 |

Table 7 shows that as number of points increase norm value getting smaller values. Those errors are smaller just a little smaller than in Explicit Upwind Scheme.

Overall Lax-Wendroff is good quality scheme with results comparable to Explicit Upwind Scheme.

* 1. **Richtmyer multi-step Scheme**
     1. **Results for sign type of boundary conditions**

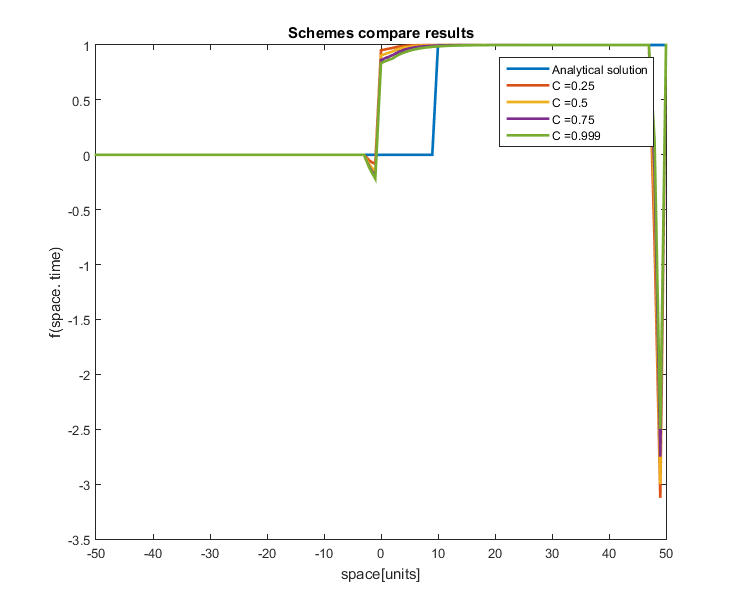
****

Fig. 18 Analytical solution and Richtmyer multi-step scheme results for sign type of initial boundary t=5, C = { 0.25 ,0.5, 0.75, 0.999}, number of points = 100. Stable results.

Fig. 18 Shows Richtmyer multi-step scheme behavior. Unfortunately my implementation is unstable for this solution. Especially at the very end some huge error occurs.

Table 8 Norms values depending on Courant number in Richtmyer multi-step scheme, sign boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Courant Number** | **Infinite norm** | **Norm one** | **Norm two** |
| 0,25 | 4.125 | 0.159607 | 0.0548335 |
| 0,5 | 4 | 0.154968 | 0.0527197 |
| 0,75 | 3.75 | 0.138316 | 0.0486656 |
| 0,999 | 3.4965 | 0.13158 | 0.0457204 |

**As can be seen errors have big values.**

**Checking Richtmyer multi-step results for time change**

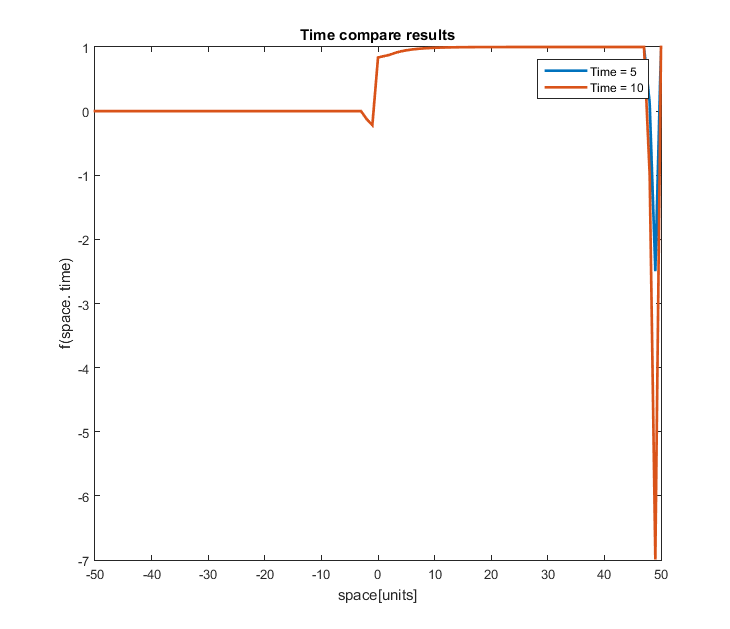
****

Fig. 19 Fig. 6 Richtmyer multi-step scheme results for different times, sign type of initial boundary t=5 and 10, CFL = 0.999, number of points = 100.

In this case scheme doesn’t work fine as well. Especially at the end of space domain we can observe relatively big negative value.

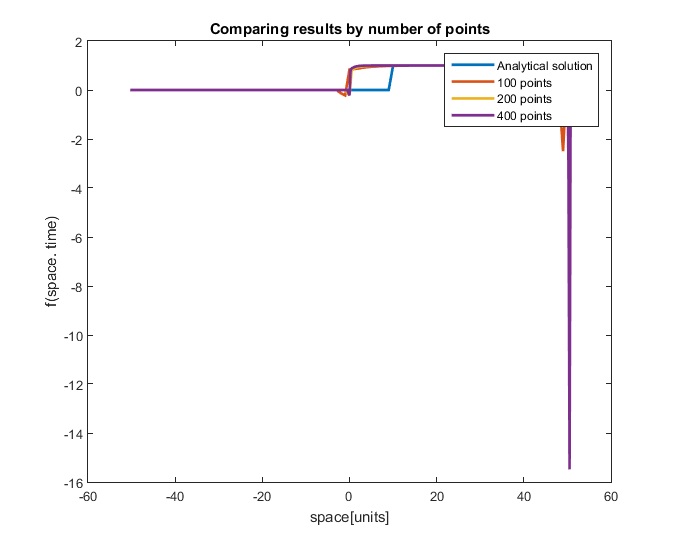


Fig. 20 Richtmyer results for different times, sign type of initial boundary t=5, CFL = 0.999, number of points = 100, 200 and 400.

As expected also in this case results are far from being correct. Probably in a code is little logic mistake.

Table 9 Norms values depending on number of points in Richtmyer scheme, sign boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of points in space** | **Infinite norm** | **Norm one** | **Norm two** |
| 100 | 3.4965 | 0.13158 | 0.0457204 |
| 200 | 7.992 | 0.133451 | 0.0457428 |
| 400 | 16.4835 | 0.135743 | 0.0448285 |

Until now it hard to say about Richtmyer multi-step method accuracy. It now working properly.

* + 1. **Results for exponential type of boundary conditions**

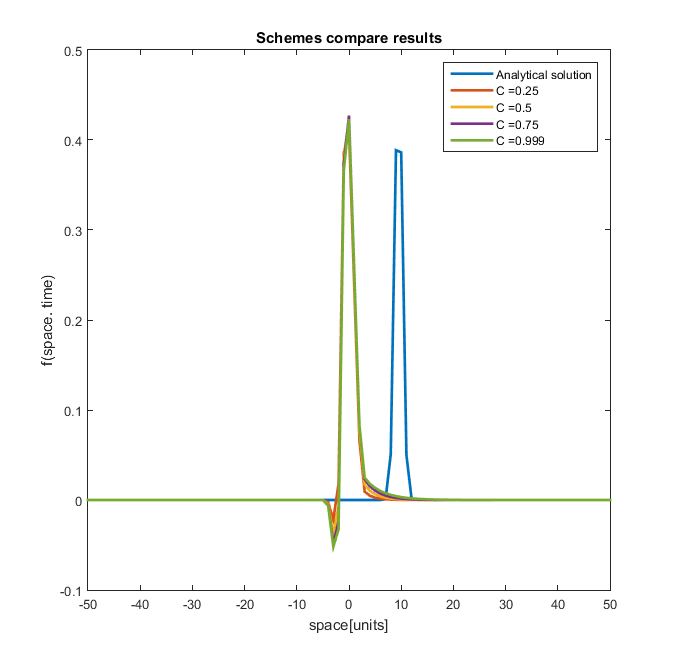
****

Fig. 21 Comparing Richtmyer’s multi-step method results for different number of points, exponential type of initial boundary t=5, CFL { 0.25 ,0.5, 0.75, 0.999}, number of points = 100

Surprisingly for exponential boundary type scheme works better, is still not fully accurate graph seems to be shifted. Because of that fact maybe there is something wrong with sign initial boundary values.

Table 10 Norms values depending on number of points in Richtmyer’s multi-step method, exponential boundary type. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Courant Number** | **Infinite norm** | **Norm one** | **Norm two** |
| 0,25 | 0.499837 | 0.0180175 | 0.00798902 |
| 0,5 | 0.499377 | 0.0184629 | 0.00804342 |
| 0,75 | 0.468691 | 0.018925 | 0.00800204 |
| 0,999 | 0.423177 | 0.019019 | 0.00789675 |

Norms values confirms that method is quite accurate is exponential boundary condition case. This situation is unusual and curious.

**Compare this case for different time**

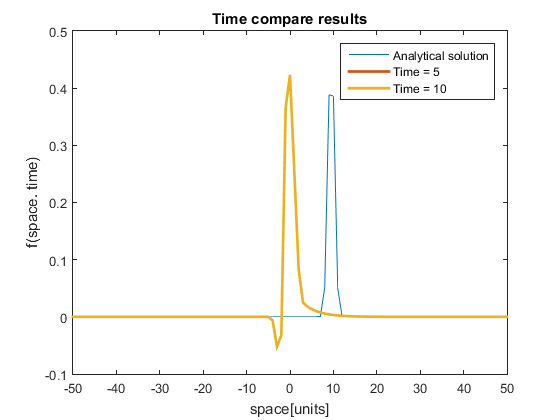
****

Fig. 22 Richtmyer’s multi-step method results for different times, exponential type of initial boundary t=5 and 10, CFL = 0.999, number of points = 100.

Graph doesn’t seems to be shifted by a time or shift is too small to obserbate. It stands in one place.

**Finally Compare this case for different number of points**

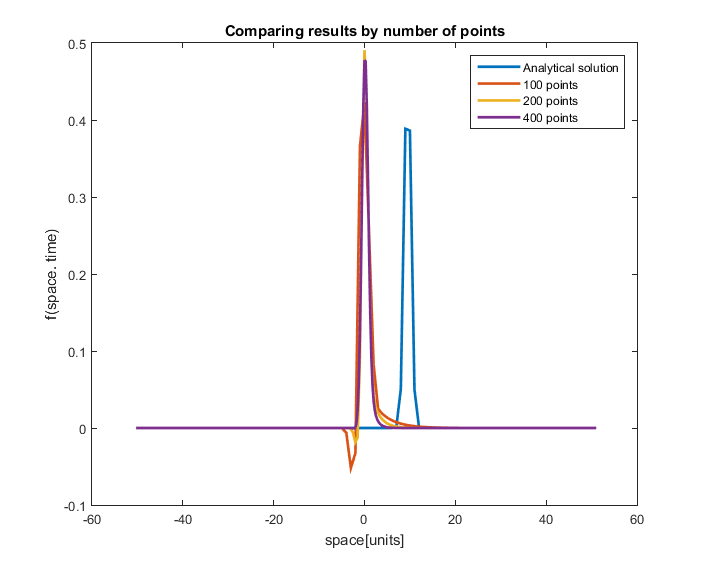
****

Fig. 23 Richtmyer’s multi-step method results for different times, exponential type of initial boundary t=5, CFL = 0.999, number of points = 100, 200 and 400.

Again graph is not shifted in time but despite of that it’s accuracy seems to be quite high.

Table 11 Norms values depending on number of points in Richtmyer’s multi-step method with exponential type of initial boundary. Data results for t=5, CFL = 0.999 number of points = 100, 200, 400.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of points in space** | **Infinite norm** | **Norm one** | **Norm two** |
| 100 | 0.423177 | 0.019019 | 0.00789675 |
| 200 | 0.499408 | 0.0178462 | 0.00551312 |
| 400 | 0.499982 | 0.0175496 | 0.00388348 |

Table 11 shows that norms have relatively small values. Especially norm one and two have decent (small) values.

Overall Richtmyer’s multi-step method doesn’t work properly in this case. Probably there is some issue with time, because for each time step values are the same.

# **Conclusions**

In this report four schemes type were exanimated. All of them offers quite different approach and all of then except Analytical Scheme are based on prediction, that means previous values creating function (scheme) next values.

Firstly Explicit Upwind Scheme was tested. Results were really satisfying considering the fact that this approach is the easiest. Errors were at the low level and results “covered” Analytical solution on graphs. Explicit Upwind Scheme is not as efficient as Lax-Wendroff’s method but it is comparably accurate for given problem. Very important value is Courant number, when it is to high stability requirement is not fulfilled. In that situation as it was showed on Fig. 4 results are highly inaccurate. Nevertheless to low Courant number causes low accuracy, that’s why optimal value (where optimality condition is highest possible accuracy) is 1 or value as less as possible less than one.

Lax-Wendroff’s method is another researched scheme type. At the output results were as well accurate, comparably accurate as Explicit Upwind Scheme but still that accuracy level is satisfying. Characteristic to Lax-Wendroff’s method behavior is attempt to adapt before sudden value change. Method is has little values fluctuation at this point. Norms and errors were at promising level what makes it approach really solid numerical method.

There was an issue with Richtmyer’s multi-step. For sign type initial boundary type this method was in this assignment case unstable. Surprisingly method worked much better for exponential initial boundary type what was confirmed by relatively low errors. This bug should be fixed in near future. Generally as literature describes Richtmyer multi-step approach is resistant for exceeding stability condition. This fact makes that method really useful when it comes to reliability issue.

There is no one the best method and each of them have its advantages and limitations. Considering robust condition for stability the best solution will be Richtmyer’s multi-step method. When efficiency is an issue Lax-Wendroff’s should fulfil user requirements. When easy and fast implementation is needed Explicit Upwind Scheme will be approvable.

References

[1] R. Courant, K. Friedrichs, H. Lewy, *On the partial difference equations of mathematical physics*, IBM Journal, March 1967, pp. 215-234.

[2] G. Cohen, Higher-Order Numerical Methods for Transient Wave equation, Springer-Verlag, 2001

[3] R. M. M. Mattheij et al., Partial Differential Equations: Modeling, Analysis, Computation, SIAM, ISBN: 978-0-898715-94-1

[4] G. D. Smith.Numerical Solution of Partial Differential Equations: Finite Difference Methods Oxford Univesity Press, third edition, 1986

Appendix A

1. Courant number

)

**Where**

– time step

– step in space domain

1. Explicit Upwind Scheme

()

(33)

(34)

1. Implicit Upwind Scheme

)

Following for was used for implementation:

()

1. Lax-Wendroff method

Using Taylor series we can assume:

(37)

(38)

(39)

(40)

Applying (41) to Equation (37):

(41)

Knowing:

)

Simplifying and getting implementation form:

)

1. Richtmyer multi-step method

In this method there are two steps:

(44)

And:

(45)

Following with:

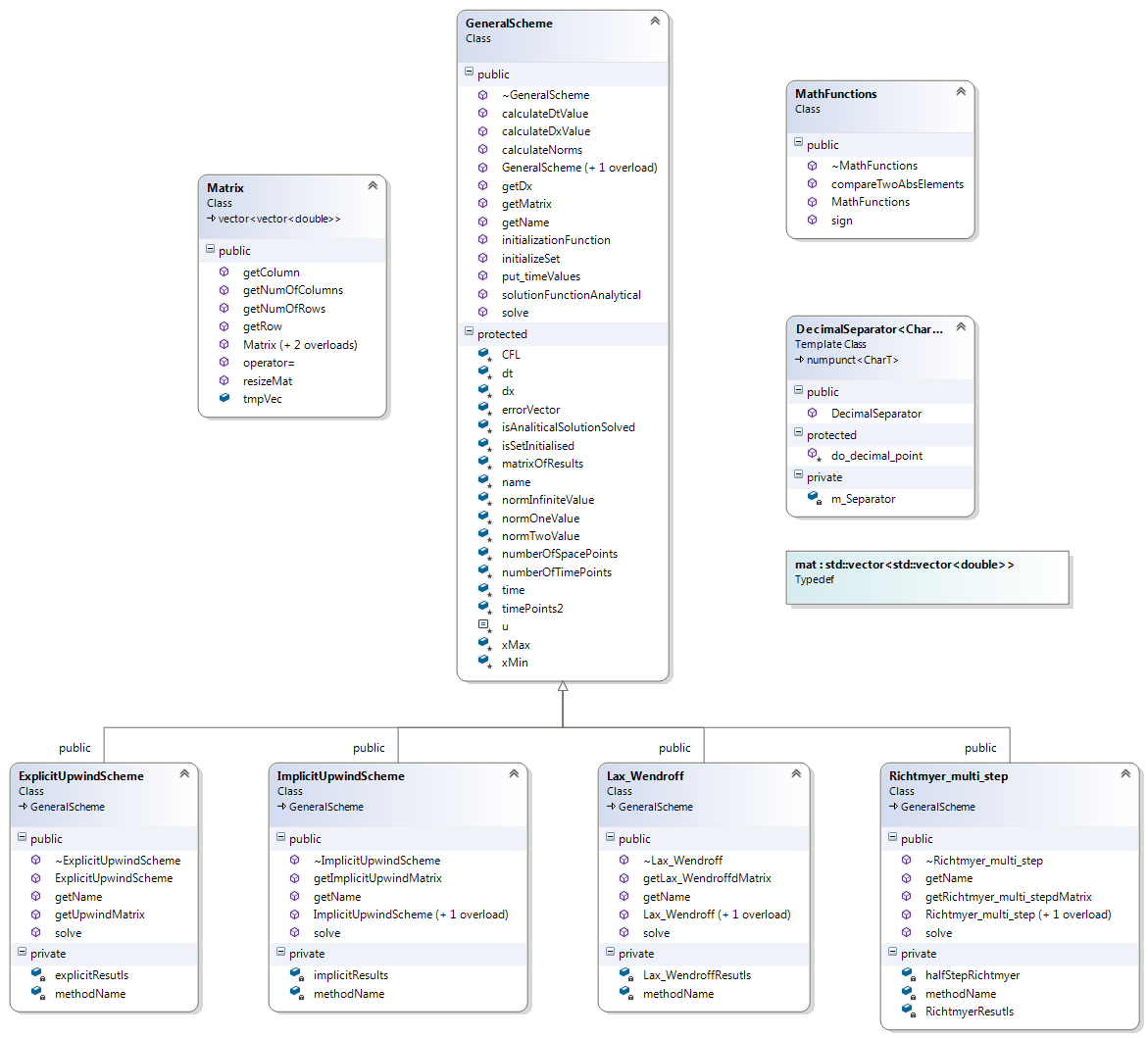
) (46)

())

Simplifying to easy to implement form:

)

**UML Diagram**

****

**Main.cpp**

#include <iostream>

#include <vector>

#include <stdio.h>

#include <iostream>

#include <iterator>

#include <string>

#include <memory>

#include <map>

#include <iomanip>

#include "MathFunctions.h"

#include "GeneralScheme.h"

#include "ExplicitUpwindScheme.h"

#include "ImplicitUpwindScheme.h"

#include "Lax\_Wendroff.h"

#include "Richtmyer\_multi\_step.h"

#include "Display.h"

using std::vector;

using std::cin;

using std::cout;

using std::endl;

//Own display namespace with user friendly displaying functions

using namespace display;

template<typename CharT>

class DecimalSeparator : public std::numpunct<CharT>

{

public:

DecimalSeparator(CharT Separator)

: m\_Separator(Separator)

{}

protected:

CharT do\_decimal\_point()const

{

return m\_Separator;

}

private:

CharT m\_Separator;

};

void runSchemes(int numberOfBoundaryConditionSet, vector <double> initialSettings, std::string typeOfExtension)

{

GeneralScheme general = GeneralScheme(initialSettings[0], initialSettings[1], initialSettings[2], initialSettings[3], initialSettings[4]);

general.solve(numberOfBoundaryConditionSet);

ExplicitUpwindScheme upwindScheme(initialSettings[0], initialSettings[1], initialSettings[2], initialSettings[3], initialSettings[4]);

upwindScheme.solve(numberOfBoundaryConditionSet);

ImplicitUpwindScheme implicitUpwindScheme(initialSettings[0], initialSettings[1], initialSettings[2], initialSettings[3], initialSettings[4]);

implicitUpwindScheme.solve(numberOfBoundaryConditionSet);

Lax\_Wendroff laxWendroff(initialSettings[0], initialSettings[1], initialSettings[2], initialSettings[3], initialSettings[4]);

laxWendroff.solve(numberOfBoundaryConditionSet);

Richtmyer\_multi\_step solutionRichtmyer(initialSettings[0], initialSettings[1], initialSettings[2], initialSettings[3], initialSettings[4]);

solutionRichtmyer.solve(numberOfBoundaryConditionSet);

std::ofstream osGeneralScheme;

std::ofstream osUpwindScheme;

std::ofstream osImplicitScheme;

std::ofstream osLaxFile;

std::ofstream osRichtmyer;

//Operation helps to plot charts in programs such as Exel. Setting type of decimal separator depending on current geographical location. In some countries comma in default separator in numbers in others dot

osGeneralScheme.imbue(std::locale(std::cout.getloc(), new DecimalSeparator<char>(',')));

osUpwindScheme.imbue(std::locale(std::cout.getloc(), new DecimalSeparator<char>(',')));

osImplicitScheme.imbue(std::locale(std::cout.getloc(), new DecimalSeparator<char>(',')));

osLaxFile.imbue(std::locale(std::cout.getloc(), new DecimalSeparator<char>(',')));

osRichtmyer.imbue(std::locale(std::cout.getloc(), new DecimalSeparator<char>(',')));

//Open/create file with selected extension. It clold be for instance exel files extensions (.xls; .xlsx).

osGeneralScheme.open("C:/Users/Domowy/Desktop/Results/" + getInitialBoundaryConditionName(numberOfBoundaryConditionSet) + "\_" + general.getName() + "Results\_t=" + std::to\_string((int)initialSettings[2]) + "\_points=" + std::to\_string((int)initialSettings[3]) + "\_CFL=" + std::to\_string(initialSettings[4]) + typeOfExtension);

osUpwindScheme.open("C:/Users/Domowy/Desktop/Results/" + getInitialBoundaryConditionName(numberOfBoundaryConditionSet) + "\_" + upwindScheme.getName() + "Results\_t=" + std::to\_string((int)initialSettings[2]) + "\_points=" + std::to\_string((int)initialSettings[3]) + "\_CFL=" + std::to\_string(initialSettings[4]) + typeOfExtension);

osImplicitScheme.open("C:/Users/Domowy/Desktop/Results/" + getInitialBoundaryConditionName(numberOfBoundaryConditionSet) + "\_" + implicitUpwindScheme.getName() + "Results\_t=" + std::to\_string((int)initialSettings[2]) + "\_points=" + std::to\_string((int)initialSettings[3]) + "\_CFL=" + std::to\_string(initialSettings[4]) + typeOfExtension);

osLaxFile.open("C:/Users/Domowy/Desktop/Results/" + getInitialBoundaryConditionName(numberOfBoundaryConditionSet) + "\_" + laxWendroff.getName() + "Results\_t=" + std::to\_string((int)initialSettings[2]) + "\_points=" + std::to\_string((int)initialSettings[3]) + "\_CFL=" + std::to\_string(initialSettings[4]) + typeOfExtension);

osRichtmyer.open("C:/Users/Domowy/Desktop/Results/" + getInitialBoundaryConditionName(numberOfBoundaryConditionSet) + "\_" + solutionRichtmyer.getName() + "Results\_t=" + std::to\_string((int)initialSettings[2]) + "\_points=" + std::to\_string((int)initialSettings[3]) + "\_CFL=" + std::to\_string(initialSettings[4]) + typeOfExtension);

//Saving schemes calculated results

osGeneralScheme << general.getMatrix();

osUpwindScheme << upwindScheme.getUpwindMatrix();

osImplicitScheme << implicitUpwindScheme.getImplicitUpwindMatrix();

osLaxFile << laxWendroff.getLax\_WendroffdMatrix();

osRichtmyer << solutionRichtmyer.getRichtmyer\_multi\_stepdMatrix();

//Closing alle opened streams at the end

osGeneralScheme.close();

osUpwindScheme.close();

osImplicitScheme.close();

osLaxFile.close();

osRichtmyer.close();

}

int main()

{

//Number of boundary condition set. 1 for sign boundary set type ; 2 for exp boundary set type

vector<int> setNumber = {1};

//Initial setings values are respectively: xMin, xMax, time, number of spacePoints, CFL value

cout << "Actual parameters are: ";

//Extension type of file which storing results of schemes computation. It could be Exel (.xls; .xlsx) file type for instance.

std::string typeOfExtension = ".xls";

//Running program using above settings for all initial boundary types

vector<double> courantNumberSet = { 0.25 ,0.5, 0.75, 0.999, 1.999, 3 ,5};

vector<double> pointsSet = { 100, 200, 400 };

vector<double> timeSet = { 5, 10 };

for (auto v : setNumber)

{

for (int j = 0; j < pointsSet.size(); ++j)

{

for (int i = 0; i < timeSet.size(); ++i)

{

for (int k = 0; k < courantNumberSet.size(); ++k)

{

vector <double> initialSettings = { -50, 50, timeSet[i], pointsSet[j], courantNumberSet[k] };

runSchemes(v, initialSettings, typeOfExtension);

}

}

}

}

return 0;

}

**Display.h**

#pragma once

#include <vector>

#include <iostream>

/\*\*

@brief Own namespace to display vectors and names

\*/

namespace display

{

/\*\*

@brief Fucntion to display given input vector to console

@param Vector to display

\*/

void displayVector(std::vector<double> vector);

/\*\*

@brief Display name of setected boundary condition. This function contains two sets of boundary condition: exponential and sign

@param Number of Boundary condition to dsplay. Takes values 1 or 2

\*/

std::string getInitialBoundaryConditionName(int & numberOfBoundaryCondition);

}

**Display.cpp**

#include "Display.h"

void display::displayVector(std::vector<double> vector)

{

for (auto v : vector)

{

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

}

std::cout << std::endl;

}

std::string display::getInitialBoundaryConditionName(int & numberOfBoundaryCondition)

{

switch (numberOfBoundaryCondition)

{

case 1:

return "Sign";

break;

case 2:

return "Exp";

break;

default:

break;

}

}

//Overloaded operator << for easy loading vector to file

template<typename T>

std::ostream &operator <<(std::ostream &out, const std::vector<T> &v)

{

std::copy(v.begin(), v.end(), std::ostream\_iterator<T>(out, "\n"));

return out;

}

**GeneralSheme.h**

#include "Display.h"

void display::displayVector(std::vector<double> vector)

{

for (auto v : vector)

{

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

}

std::cout << std::endl;

}

std::string display::getInitialBoundaryConditionName(int & numberOfBoundaryCondition)

{

switch (numberOfBoundaryCondition)

{

case 1:

return "Sign";

break;

case 2:

return "Exp";

break;

default:

break;

}

}

//Overloaded operator << for easy loading vector to file

template<typename T>

std::ostream &operator <<(std::ostream &out, const std::vector<T> &v)

{

std::copy(v.begin(), v.end(), std::ostream\_iterator<T>(out, "\n"));

return out;

}

**GeneralScheme.cpp**

#include "GeneralScheme.h"

/\*\*

Default constructor

\*/

GeneralScheme::GeneralScheme()

{

}

GeneralScheme::~GeneralScheme()

{

}

GeneralScheme::GeneralScheme(double xMin, double xMax, double time, double numberOfSpacePoints, double CFL)

:xMin(xMin), xMax(xMax), time(time), numberOfSpacePoints(numberOfSpacePoints), CFL(CFL), isSetInitialised(false), isAnaliticalSolutionSolved(false), name("GeneralScheme")

{

(\*this).dt = (\*this).calculateDtValue();

(\*this).dx = (\*this).calculateDxValue();

(\*this).numberOfTimePoints = std::ceil( (time) / ( ((\*this).CFL \* (\*this).dx)/u ) );

matrixOfResults = Matrix(numberOfSpacePoints, numberOfTimePoints);

(\*this).calculateDtValue();

}

double GeneralScheme::calculateDtValue()

{

return (\*this).dt = ((\*this).CFL \* (\*this).dx) / u;

}

double GeneralScheme::calculateDxValue()

{

return (\*this).dx = (std::abs((\*this).xMin) + std::abs((\*this).xMax) + 1) / (\*this).numberOfSpacePoints;

}

double GeneralScheme::getDx()

{

return dx;

}

double GeneralScheme::initializationFunction(int numberOfSet, double functionValue)

{

switch (numberOfSet)

{

case 1:

return (MathFunctions::sign(functionValue) + 1);

break;

case 2:

return std::exp((-1.0)\*std::pow(functionValue, 2));

break;

}

}

double GeneralScheme::solutionFunctionAnalytical(int numberOfSet, double actualSpaceValue, double actualTimeValue)

{

switch (numberOfSet)

{

case 1:

return 0.5 \* (MathFunctions::sign(actualSpaceValue - 1.75 \* actualTimeValue) + 1);

break;

case 2:

return 0.5 \* std::exp((-1.0)\*std::pow(actualSpaceValue - 1.75 \* actualTimeValue, 2));

break;

default:

break;

}

}

//Error and norms calculation

void GeneralScheme::calculateNorms(Matrix& toCalculateError)

{

errorVector.resize(toCalculateError.getNumOfRows());

for (auto i = 0; i < numberOfSpacePoints; ++i)

{

errorVector[i] = toCalculateError[i][numberOfTimePoints-1] - (\*this).matrixOfResults[i][numberOfTimePoints-1];

}

/\*

Second approach - quite complicated

(\*this).normInfiniteValue = std::max\_element(errorVector.begin(), errorVector.end(), MathFunctions::compareTwoAbsElements);

normInfiniteValue1 = std::distance(errorVector.begin(), normInfiniteValue);

double normInf = errorVector.at(normInfiniteValue1);

\*/

normInfiniteValue = 0;

for (auto i = 0; i < errorVector.size(); ++i)

{

if (abs(normInfiniteValue) < abs(errorVector.at(i)))

{

normInfiniteValue = abs(errorVector.at(i));

}

(\*this).normOneValue += std::abs(errorVector[i]);

(\*this).normTwoValue += std::pow(std::abs(errorVector[i]), 2);

}

normTwoValue = std::sqrt(normTwoValue);

normOneValue = normOneValue / numberOfSpacePoints;

normTwoValue = normTwoValue / numberOfSpacePoints;

std::vector<double> norms = {normInfiniteValue, normOneValue, normTwoValue, dx };

toCalculateError.resizeMat(numberOfSpacePoints, numberOfTimePoints+1);

for (int i = 0; i < norms.size(); ++i)

{

//Adding norms results to last matrix colunm

toCalculateError[i][numberOfTimePoints ] = norms.at(i);

}

}

void GeneralScheme::put\_timeValues()

{

double actualValue = 0;

for (int i = 0; i < numberOfTimePoints; ++i)

{

matrixOfResults[0][i] = actualValue;

actualValue += (\*this).dt;

}

}

std::string GeneralScheme::getName()

{

return name;

}

void GeneralScheme::initializeSet(int setNumber)

{

try

{

double actualValue = xMin;

for (int i = 0; i < numberOfSpacePoints ; ++i)

{

matrixOfResults[i][0] = (1.0 / 2.0) \* (\*this).initializationFunction(setNumber, actualValue);

actualValue += (\*this).dx;

}

if (setNumber == 1)

{

for (int i = 0; i < numberOfTimePoints ; ++i)

{

matrixOfResults[0][i] = 0;

matrixOfResults[numberOfSpacePoints - 1][i] = 1;

}

}

else

{

for (int i = 0; i < numberOfTimePoints ; ++i)

{

matrixOfResults[0][i] = 0;

matrixOfResults[numberOfSpacePoints - 1][i] = 0;

}

}

(\*this).isSetInitialised = true;

}

catch (std::exception & e)

{

std::cout << "Standard exception: " << e.what() << std::endl;

}

}

//Analytical scheme solving

void GeneralScheme::solve(int numberOfBoundaryConditionSet)

{

(\*this).initializeSet(numberOfBoundaryConditionSet);

try

{

if ((\*this).isSetInitialised == true)

{

std::cout << "Analytical solution runs and matrix is initialised\n";

//Variables hold values below 0. Thanks to that negative values could be passed to sign function, it makes loop iteration easier.

double actualSpaceValue = xMin;

//Variable assinged to dt because time at 0 point is initialised in function initializeSet()

double actualTimeValue = dt;

for (int i = 1; i < numberOfSpacePoints; ++i)

{

for (auto j = 1; j < numberOfTimePoints; ++j)

{

matrixOfResults[i][j] = solutionFunctionAnalytical(numberOfBoundaryConditionSet, actualSpaceValue, actualTimeValue);

actualTimeValue += dt;

}

actualTimeValue = dt;

actualSpaceValue += dx;

}

isAnaliticalSolutionSolved = true;

}

else

{

std::cout << "Matrix is not initialised\n";

}

}

catch (std::exception & e)

{

std::cout << "Standard exception: " << e.what() << std::endl;

}

}

Matrix GeneralScheme::getMatrix()

{

return matrixOfResults;

}

**ExplicitUpwindScheme.h**

#pragma once

#include "GeneralScheme.h"

/\*\*

\* Class created for calculating Explicit Upwind Scheme

\* Class inherits methods from GeneralScheme class

\*/

class ExplicitUpwindScheme : public GeneralScheme

{

std::string methodName;

Matrix explicitResutls;

public:

ExplicitUpwindScheme(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL);

~ExplicitUpwindScheme();

/\*\*

@brief Virtual method which solves Explicit Upwind Scheme

@param Variable for inidicating boundary initialization set type, 1 or 2 respectively for: sign type and exponential type

\*/

virtual void solve(int setNumber) override;

/\*\*

@brief Method returns Matrix where Explicit Upwind Scheme solution is stored

@return Matrix with calculated Explicit Upwind Scheme

\*/

Matrix ExplicitUpwindScheme::getUpwindMatrix();

/\*\*

@brief Virtual method to returns name of ExplicitUpwindScheme class

@return name of ExplicitUpwindScheme class

\*/

virtual std::string ExplicitUpwindScheme::getName() override;

};

**ExplicitUpwindScheme.cpp**

#include "ExplicitUpwindScheme.h"

ExplicitUpwindScheme::ExplicitUpwindScheme(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL) : GeneralScheme::GeneralScheme(xMin, xMax, time, numberOfSpacePoints, CFL), methodName("ExplicitUpwindScheme")

{

}

ExplicitUpwindScheme::~ExplicitUpwindScheme()

{

}

void ExplicitUpwindScheme::solve(int setNumber)

{

try

{

std::cout << "Explicit upwid scheme solution runs and matrix is initialised\n";

(\*this).initializeSet(setNumber);

explicitResutls = Matrix((\*this).getMatrix());

for (auto j = 0; j < numberOfTimePoints-1; ++j)

{

for (int i = 1; i < numberOfSpacePoints; ++i)

{

explicitResutls[i][j+1] = (explicitResutls[i][j] - CFL\*(explicitResutls[i][j] - explicitResutls[i - 1][j]));

}

}

GeneralScheme::solve(setNumber);

calculateNorms((\*this).explicitResutls);

}

catch (std::exception & e)

{

std::cout << "Standard exception: " << e.what() << std::endl;

}

}

Matrix ExplicitUpwindScheme::getUpwindMatrix()

{

return explicitResutls;

}

std::string ExplicitUpwindScheme::getName()

{

return methodName;

}

**ImplicitUpwindScheme.h**

#pragma once

#include "GeneralScheme.h"

class ImplicitUpwindScheme :

public GeneralScheme

{

std::string methodName;

Matrix implicitResults;

public:

ImplicitUpwindScheme();

ImplicitUpwindScheme(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL);

virtual ~ImplicitUpwindScheme();

/\*\*

@brief Virtual method which solves Implicit Upwind Scheme

@param Variable for inidicating boundary initialization set type, 1 or 2 respectively for: sign type and exponential type

\*/

virtual void solve(int setNumber) override;

/\*\*

@brief Method returns Matrix where Implicit Upwind Scheme solution is stored

@return Matrix with calculated Implicit Upwind Schem

\*/

Matrix getImplicitUpwindMatrix();

/\*\*

@brief Virtual method to returns name of Implicit Upwind Scheme class

@return name of Implicit Upwind Scheme class

\*/

virtual std::string ImplicitUpwindScheme::getName() override;

};

**ImplicitUpwindScheme.cpp**

#include "ImplicitUpwindScheme.h"

ImplicitUpwindScheme::ImplicitUpwindScheme()

{

}

ImplicitUpwindScheme::ImplicitUpwindScheme(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL) : GeneralScheme::GeneralScheme(xMin, xMax, time, numberOfSpacePoints, CFL), methodName("ImplicitUpwindScheme")

{

}

ImplicitUpwindScheme::~ImplicitUpwindScheme()

{

}

void ImplicitUpwindScheme::solve(int setNumber)

{

try

{

std::cout << "Implicit upwid scheme solution runs and matrix is initialised\n";

(\*this).initializeSet(setNumber);

implicitResults = Matrix((\*this).getMatrix());

for (auto j = 0; j < numberOfTimePoints - 1; ++j)

{

for (int i = 1; i < numberOfSpacePoints; ++i)

{

implicitResults[i][j + 1] = (-1.0 \* CFL) \* (implicitResults[i][j + 1] - implicitResults[i - 1][j + 1]) + implicitResults[i ][j];

}

}

GeneralScheme::solve(setNumber);

calculateNorms((\*this).implicitResults);

}

catch (std::exception & e)

{

std::cout << "Standard exception: " << e.what() << std::endl;

}

}

Matrix ImplicitUpwindScheme::getImplicitUpwindMatrix()

{

return implicitResults;

}

std::string ImplicitUpwindScheme::getName()

{

return methodName;

}

**Lax\_Wendroff.h**

#pragma once

#include "GeneralScheme.h"

class Lax\_Wendroff :

public GeneralScheme

{

std::string methodName;

Matrix Lax\_WendroffResutls;

public:

Lax\_Wendroff();

Lax\_Wendroff(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL);

virtual ~Lax\_Wendroff();

/\*\*

@brief Virtual method which solves Lax-Wendroff

@param Variable for inidicating boundary initialization set type, 1 or 2 respectively for: sign type and exponential type

\*/

virtual void solve(int setNumber) override;

//double solutionFunctionExplicitScheme(int numberOfSet, Matrix toUpwindSchemeCalculations);

/\*\*

@brief Method returns Matrix where Lax-Wendroff solution is stored

@return Matrix with calculated Lax-Wendroff method

\*/

Matrix Lax\_Wendroff::getLax\_WendroffdMatrix();

/\*\*

@brief Virtual method to returns name of Lax-Wendroff class

@return name of Lax-Wendroff class

\*/

virtual std::string Lax\_Wendroff::getName() override;

};

**Lax\_Wendroff.cpp**

#include "Lax\_Wendroff.h"

Lax\_Wendroff::Lax\_Wendroff()

{

}

Lax\_Wendroff::Lax\_Wendroff(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL) : GeneralScheme::GeneralScheme(xMin, xMax, time, numberOfSpacePoints, CFL), methodName("Lax\_Wendroff")

{

}

Lax\_Wendroff::~Lax\_Wendroff()

{

}

void Lax\_Wendroff::solve(int setNumber)

{

try

{

std::cout << "Lax\_Wendroff upwind scheme solution runs and matrix is initialised\n";

(\*this).initializeSet(setNumber);

Lax\_WendroffResutls = Matrix((\*this).getMatrix());

auto T1 = (CFL \* (CFL + 1)) / 2;

auto T2 = 1 - (CFL \* CFL) ;

auto T3 = (CFL \* (CFL - 1)) / 2;

for (auto j = 0; j < numberOfTimePoints - 1; ++j)

{

for (int i = 1; i < numberOfSpacePoints - 1; ++i)

{

Lax\_WendroffResutls[i][j + 1] = T1 \* Lax\_WendroffResutls[i - 1][j] + T2 \* Lax\_WendroffResutls[i][j] + T3 \* Lax\_WendroffResutls[i + 1][j];

}

}

GeneralScheme::solve(setNumber);

calculateNorms((\*this).Lax\_WendroffResutls);

}

catch (std::exception & e)

{

std::cout << "Standard exception: " << e.what() << std::endl;

}

}

Matrix Lax\_Wendroff::getLax\_WendroffdMatrix()

{

return Lax\_WendroffResutls;

}

std::string Lax\_Wendroff::getName()

{

return methodName;

}

**Richtmyer\_multi\_step.h**

#pragma once

#include "GeneralScheme.h"

class Richtmyer\_multi\_step :

public GeneralScheme

{

std::string methodName;

Matrix RichtmyerResutls;

Matrix halfStepRichtmyer;

public:

Richtmyer\_multi\_step();

Richtmyer\_multi\_step(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL);

virtual ~Richtmyer\_multi\_step();

/\*\*

@brief Virtual method which solves Richtmyer\_multi\_step method

@param Variable for inidicating boundary initialization set type, 1 or 2 respectively for: sign type and exponential type

\*/

//Virtual solve method as in the rest classes. Overide is an "tip" for compiler in cases then method could be by an accident not overrided but created as new one

virtual void solve(int setNumber) override;

/\*\*

@brief Method returns Matrix where Richtmyer\_multi\_step solution is stored

@return Matrix with calculated Richtmyer\_multi\_step

\*/

Matrix Richtmyer\_multi\_step::getRichtmyer\_multi\_stepdMatrix();

/\*\*

@brief Virtual method to returns name of Richtmyer\_multi\_ste class

@return name of Richtmyer\_multi\_step class

\*/

virtual std::string Richtmyer\_multi\_step::getName() override;

};

**Richtmyer\_multi\_step.cpp**

#include "Richtmyer\_multi\_step.h"

Richtmyer\_multi\_step::Richtmyer\_multi\_step()

{

}

Richtmyer\_multi\_step::Richtmyer\_multi\_step(double xMin,

double xMax,

double time,

double numberOfSpacePoints,

double CFL) : GeneralScheme::GeneralScheme(xMin, xMax, time, numberOfSpacePoints, CFL), methodName("RichtmyerMultiStepScheme")

{

}

Richtmyer\_multi\_step::~Richtmyer\_multi\_step()

{

}

void Richtmyer\_multi\_step::solve(int setNumber)

{

try

{

std::cout << "Richtmyer\_multi\_step scheme solution runs and matrix is initialised\n";

//Preparing initial data with choosen setNumber: 1 - sign set; 2 - exp set

(\*this).initializeSet(setNumber);

RichtmyerResutls = Matrix((\*this).getMatrix());

halfStepRichtmyer = Matrix((\*this).getMatrix());

//Calculating Richtmyer\_multi\_step scheme cooeficients before loop for code claryti and performance profit

double coef1 = 0.5 \* (1 - (CFL / 2));

double coef2 = 0.5 \* (1 + (CFL / 2));

double coef3 = CFL / 2;

//Main time loop iterating for each time point

for (int j = 0; j < numberOfTimePoints - 1; ++j)

{

//First it is needed to calculate half step according to Richtmyer equation. Those results will be used in final computation.

for (int i = 1; i < numberOfSpacePoints - 1; ++i)

{

halfStepRichtmyer[i][j] = coef1 \* halfStepRichtmyer[i + 1][j] + coef2 \*halfStepRichtmyer[i - 1][j];

}

//In this loop previosly calculated half step is used

for (int i = 1; i < numberOfSpacePoints - 1; ++i)

{

RichtmyerResutls[i][j + 1] = RichtmyerResutls[i][j] - coef3 \* (halfStepRichtmyer[i + 1][j] - halfStepRichtmyer[i - 1][j]);

}

}

GeneralScheme::solve(setNumber);

calculateNorms((\*this).RichtmyerResutls);

}

catch (std::exception & e)

{

std::cout << "Standard exception: " << e.what() << std::endl;

}

}

Matrix Richtmyer\_multi\_step::getRichtmyer\_multi\_stepdMatrix()

{

return RichtmyerResutls;

}

std::string Richtmyer\_multi\_step::getName()

{

return methodName;

}

**MathFunctions.h**

#pragma once

#include <cmath>

/\*\*

@brief Class

\*/

class MathFunctions

{

public:

MathFunctions();

~MathFunctions();

static int sign(double x);

static bool compareTwoAbsElements(double first, double second);

};

**MathFunctions.cpp**

#include "MathFunctions.h"

MathFunctions::MathFunctions()

{

}

MathFunctions::~MathFunctions()

{

}

/\*\*

@brief Static function sign

Fucntion gives an output of sign function

1. -1 When input walue < 0

2. 0 when input value = 0

3. 1 When input value > 0

@param Input value to sign function

@return result of sign function

\*/

int MathFunctions::sign(double x)

{

if (x < 0)

{

return -1;

}

else if (x == 0)

{

return 0;

}

else

{

return 1;

}

}

bool MathFunctions::compareTwoAbsElements(double first, double second)

{

return (std::abs(first) < std::abs(second));

}

**Matrix.h**

#include <vector>

#include <ostream>

#include <iostream>

#include <fstream>

#include <memory>

class Matrix : private std::vector< std::vector<double> >

{

public:

using vector<vector<double> >::operator[];

Matrix();

std::shared\_ptr < std::vector<double> > tmpVec;

Matrix(int numOfRows, int numOfColumns);

Matrix(const Matrix & m);

/\*\*

@brief Method returns number of rows in Matrix

@return Number of rows in Matrix

\*/

int getNumOfRows() const;

/\*\*

@brief Method returns number of columns in Matrix

@return Number of columns in Matrix

\*/

int getNumOfColumns() const;

/\*\*

@brief Method returns selected row as vector

@return Returns selected row as vector

\*/

std::vector<double>& getRow(int rowNumber);

/\*\*

@brief Method returns selected column as vector

@return Selected column as vector

\*/

std::vector<double> & Matrix::getColumn(int columnnumber);

friend std::ostream& operator<<(std::ostream& os, Matrix& mat);

friend std::ofstream& operator<<(std::ofstream& ofs,

const Matrix& m);

Matrix& Matrix::operator=(const Matrix& m);

/\*\*

@brief Method returns selected column as vector

@param New rows quantity for Matrix resize

@param New columns quantity for Matrix resize

@return Selected column as vector

\*/

void resizeMat(int numOfRows, int numOfColumns);

};

**Matrix.cpp**

#include "Matrix.h"

/\*\*

Matrix class allows to inspired by Dr Peter Sherar's Matrix class

\*/

typedef std::vector< std::vector<double> > mat;

/\*\*

Default Constructor

\*/

Matrix::Matrix() : mat() {}

/\*\*

Constructor

\*/

Matrix::Matrix(int numOfRows, int numOfColumns) : mat()

{

(\*this).resize(numOfRows);

for (int i = 0; i < numOfRows; ++i)

{

(\*this)[i].resize(numOfColumns);

}

}

//Copy constructor

Matrix::Matrix(const Matrix& m) : std::vector<std::vector<double> >()

{

// set the size of the rows

(\*this).resize(m.size());

// set the size of the columns

std::size\_t i;

for (i = 0; i < m.size(); i++) (\*this)[i].resize(m[0].size());

// copy the elements

for (int i = 0; i < m.getNumOfRows(); i++)

for (int j = 0; j < m.getNumOfColumns(); j++)

(\*this)[i][j] = m[i][j];

}

int Matrix::getNumOfRows() const

{

return (\*this).size();

}

int Matrix::getNumOfColumns() const

{

return (\*this)[0].size();

}

std::vector<double> & Matrix::getRow(int rowNumber)

{

vector <double> tmp;

for (auto i = 0; i < (\*this).getNumOfColumns(); ++i)

{

tmp.push\_back((\*this)[rowNumber][i]);

}

return tmp;

}

std::vector<double> & Matrix::getColumn(int columnNumber)

{

vector <double> tmp;

for (auto i = 0; i < (\*this).getNumOfRows(); ++i)

{

tmp.push\_back((\*this)[i][columnNumber]);

}

return tmp;

}

std::ostream& operator<<(std::ostream& os, Matrix& mat)

{

for (int i = 0; i < mat.getNumOfRows(); ++i)

{

for (int j = 0; j < mat.getNumOfColumns(); ++j)

{

os << mat[i][j] << " ";

}

os << std::endl;

}

return os;

}

std::ofstream& operator<<(std::ofstream& ofs, const Matrix& m) {

//put matrix rownumber in first line (even if it is zero)

//ofs << "dt" << std::endl;

//put matrix columnnumber in second line (even if it is zero)

//ofs << m.getNumOfColumns() << std::endl;

//put data in third line (if size==zero nothing will be put)

for (int i = 0; i<m.getNumOfRows(); i++) {

for (int j = 0; j<m.getNumOfColumns(); j++) ofs << m[i][j] << "\t";

ofs << std::endl;

}

return ofs;

}

Matrix& Matrix::operator=(const Matrix& m)

{

(\*this).resize(m.size());

std::size\_t i;

std::size\_t j;

for (i = 0; i < m.size(); i++) (\*this)[i].resize(m[0].size());

for (i = 0; i<m.size(); i++)

for (j = 0; j<m[0].size(); j++)

(\*this)[i][j] = m[i][j];

return \*this;

}

void Matrix::resizeMat(int numOfRows, int numOfColumns)

{

(\*this).resize(numOfRows);

for (int i = 0; i < numOfRows; ++i)

{

(\*this)[i].resize(numOfColumns);

}

}