CouetteFlow: a solver for viscous flow between two plates

Release

Marc Salvadori

Mar 15, 2018

CONTENTS

Author: Marc Salvadori

Email: msalvadori3@gatech.edu

Numerical solution schemes are often referred to as being explicit or implicit. When a direct computation of the dependent variables can be made in terms of known quantities, the computation is said to be explicit. When the dependent variables are defined by coupled sets of equations, and either a matrix or iterative technique is needed to obtain the solution, the numerical method is said to be implicit. In computational fluid dynamics, the governing equations are nonlinear, and the number of unknown variables is typically very large. Under these conditions implicitly formulated equations are almost always solved using iterative techniques.

Iterations are used to advance a solution through a sequence of steps from a starting state to a final, converged state. This is true whether the solution sought is either one step in a transient problem or a final steady-state result. In either case, the iteration steps resemble a time-like process. Of course, the iteration steps usually do not correspond to a realistic time-dependent behavior. In fact, it is this aspect of an implicit method that makes it attractive for steady-state computations, because the number of iterations required for a solution is often much smaller than the number of time steps needed for an accurate transient that asymptotically approaches steady conditions.

On the other hand, it is also this "distorted transient" feature that leads to the question, "What are the consequences of using an implicit versus an explicit solution method for a time-dependent problem?" The answer to this question has two parts. The first part has to do with numerical stability and the second part with numerical accuracy.

The purpose of this project is investigate the use of numerical schemes to try to answer the above questions.

CHAPTER

ONE

CONTENTS

1.1 Project Description

1.1.1 Given task

In this exercise you calculate the viscous flow between two parallel paltes.

Such flow is described by the diffusion equation:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial y^2}$$

Each plage is a distance L apart and the boundary conditions are $u(y = 0) = 0$ and $u(y = L) = 1$. The exact solution for this equation for any location in space and time can be written as:

$$u_{exact}(y,t) = \frac{y}{L} + \sum_{n=1}^{\infty} a_n sin\left(n\pi \frac{y}{L}\right) exp\left[-\nu\left(\frac{n\pi}{L}\right)^2 t\right]$$

where the constants, a_n, in the infinite series depend on the initial condition specified. For this project you must solve the flow for the following initial condition:

$$u(y) = \frac{y}{L} + sin\left(\pi \frac{y}{L}\right)$$

The following combined implicit-explicit difference formulation (Combined Method A in section 4.2.5 in Tannehill, Anderson and Pletcher) should be used:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{\nu}{\Delta y^2}\left[\theta\left(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\right) + (1-\theta)\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right)\right]$$

First, non-dimensionalize t and y by τ and L, respectively. Select an expression for τ that essentially removes ν from the governing flow equation. Write out the non-dimensional for this problem. You will numerically solve the non-dimensional form of the problem on a uniformly spaced mesh (i.e. constant Δy) with jmax grid points (including the bottom and top wall).

Compute the time-dependent and steady state solution using a direct solution technique (i.e. non-iterative) at each time step. To do this, first rearrange the discretized equation (non-dimensional form) so that it is in tri-diagonal form. You will be able to solve for all jmax points simultaneously at each time step by writing a computer program which uses the Thomas Algorithm.

1.2 Setup

1.2.1 Setting up Couette Solver

1.2.2 Code Structure

The |CouetteFlow| code is divided into three major parts:

1. **Input**: User input is entered into a file called `input_file.xml`.
2. **Main Program**: The main program for solving the elliptical grid is `main.F90` in the `<parent directory>/src/main` folder.
3. **Output**: Once the code is run, the results are stored in the `<parent directory>/output/` folder.

1.2.3 Input File

The inputs for CouetteFlow solver are specified in the `input_file.xml`. The input file is accessed as follows:

1. Go to the parent directory:

   ```
   $ cd <path to |CoetteFlow|>
   ```

   Make sure that you are in the directory that contains the files **`setup.py`**, **`input_file.xml`**, and the folder **`src`**.

2. Open the input file:

   ```
   $ vi input_file.xml
   ```

3. The main parts of the input file are shown below

The input file is divided into different parts. The geometry is set in the `<geometry>` module, which is shown below:

```
<geometry>
  <jmax>51</jmax>
</geometry>
```

For setup of the solver, most of the inputs are set in the `<setup>` module. A snippet of this module is shown below:

```
<setup>
  <Project>2D_CouetteFlow</Project>
  <Utop>1.0</Utop>
  <nu>1.0</nu>
  <nmax>1000000000</nmax>
  <nout>1</nout>
  <L>1.0</L>
  <dt>10000.0</dt>
  <theta>1.0</theta>
  <RMSres>1.0e-7</RMSres>
</setup>
```

1.2.4 Compilation

The following sequence of commands is used to compile a single simulation.

1. Go to the parent directory:

```
$ cd <path to |CouetteFlow|>
```

   Make sure that you are in the directory that contains the files **setup.py**, **input_file.xml**, and the folder **src**.

2. Clean existing results:

```
$ ./setup.py -u clean heavy
```

   This command removes the existing files in the output folder |CouetteFlow|/output/ and deletes the object files from previous compilations. **Backup any required results before using this command.**

3. Set working directory path:

```
$ ./setup.py -u set_path
```

   This command sets the working directory path

4. Compile the build:

```
$ ./setup.py -e configure
```

   An empty CMake window opens. Press [c] on the keyboard to configure the program.

   This brings up the CMake window. There are two options for the CMAKE_BUILD_TYPE :

   - Release: This compiles the program in regular mode; debugging flags are disabled.
   - Debug: This compiles the program in debug mode; errors and warnings are displayed on the terminal.

   Press [Enter] on the keyboard to edit the option (to change from Release to Debug or vice versa)

   The file **|CouetteFlow|.x** will now be generated in the parent directory

5. Execute the program:

```
$ ./couette.x
```

   This command runs the program. If Debug mode is enabled in **COUETTE_COMPILE_DEFS**, appropriate output is printed on the Terminal screen.

1.2.5 Results

The results are stored in the output/ folder inside the parent directory. The output directory contains several files **.dat** and **.tec** where the calculations are written. In addition, there is also a output/plot folder, where figures from the calculated data are plotted. To plot the results. open the inputfile and enter the name of the **.dat** file that was generated in the files entry (as shown below).

```
<PostProcessing>
    <plot>
       <files>RESULTDATFILE</files>
       ...
    </plot>
</PostProcessing>
```

Then, from the parent directory execute the following command to plot the results using |CouetteFlow|'s built-in plotting utility:

```
$ ./setup.py -p multi_plot
```

This will generate the compined solution plots from the results, which will be stored in the `output/plot` folder.

1.3 Input File

The input file is central location for setting any and all parameters for all the features in |CouetteFlow|. Depending on the type of operation being performed on |CouetteFlow|, the relevant input options are set in the input file.

1.3.1 Structure of the Geometry Module

```
<geometry>
    <jmax>51</jmax>
 </geometry>
```

1.3.2 Structure of Setup Module

```
<setup>
   <Project>2D_CouetteFlow</Project>
   <Utop>1.0</Utop>
   <nu>1.0</nu>
   <nmax>1000000000</nmax>
   <nout>1</nout>
   <L>1.0</L>
   <dt>10000.0</dt>
   <theta>1.0</theta>
   <RMSres>1.0e-7</RMSres>
 </setup>
```

1.3.3 Structure of Postprocessing Module

|CouetteFlow| also allows the user to graphically visualize the results through the use of graphs and contours. Inputs to this plotting utility are also provided through the input file. The relevant block for this utility is shown below, and linked to the dedicated plotting utility page.

```
<PostProcessing>
  <plot>
      <iPost>1</iPost>
      <Method>TecPlot</Method>
      <files>rmslog.dat</files>
      <style>k +r</style>
      <label>Grid</label>
      <LegendFontSize>16</LegendFontSize>
      <FigureSize>10 7</FigureSize>
      <AxisLabelSize>21</AxisLabelSize>
      <AxisTitleSize>22</AxisTitleSize>
      <XTickSize>23</XTickSize>
      <YTickSize>24</YTickSize>
```

```
  </plot>
</PostProcessing>
```

1.4 Code development

The present project is aimed to develop a computer program for solving 1-D unsteady 'Couette Flow' problem. Hereafter, the program developed in this project is called 'CouetteFlow'.

1.4.1 CouetteFlow Code summary

The source code contains the following directories:

- io - input/output related routines
- main - main program driver
- math - thomas algorithm
- modules - main couette flow solver routines
- utils - list of useful FORTRAN utilities used within the program
- couettepy - python wrapper for gridgen main program

  Also a 'CMakeLists.txt' file is also included for cmake compiling.

```
$ cd CouetteFlow/src/
$ ls
$ CMakeLists.txt  io  main math modules utils couettepy
```

The **io** folder has **io.F90** file which contains **ReadInput(inputData)** subroutine. It also includes **input_file_xml** which describes the structure of the user run-time input file located in the main 'src' directory, and **output.F90** for storing data in bothb Tecplot and Python format.

The **main** folder is only used for containing the code driver file. The main routines is run by **couette.F90** which calls important subroutines from the rest of folders.

1.4.2 Details of CouetteFlow development

The source code shown below is **couette.F90** and it calls skeletal subroutines for generating grid structure. The main features of the main code is to (1) read input file, (2) make initialized variable arrays, (3) se the BCs and ICs, (4) set the time step for the solver, (5) Non-dimensionalize the variables , (6) use the thomas algorithm to calculate and update the velocity, and (7) finally write output files along with the RMS:

```
PROGRAM main

      USE xml_data_input_file
      USE CouetteSetup_m,ONLY:Init,EndVars,TimeStep,NonDim2DimVars,&
                              Dim2NonDimVars
      USE parameters_m,ONLY:wp
      USE SimVars_m,ONLY:fileLength,c1,c2,cr,elapse_time,rate,iflag,&
                         dt,t,nmax,rms_SS,rms_US,maxRMS_US
      USE CouetteSolver_m,ONLY:TriDiag,SteadySoln,UnsteadySoln,SteadyRMS,UnsteadyRMS
      USE output_m,ONLY:WritePlotFile,WriteRMS
```

```
      IMPLICIT NONE

      TYPE(input_type_t) :: inputData
      CHARACTER(LEN=fileLength) ::output= 'data'
      CHARACTER(LEN=fileLength) :: rmsout = 'rms'
      INTEGER :: n

      ! Start the time measurements
      elapse_time = 0.0_wp

      CALL system_clock(count_rate=cr)
      rate = REAL(cr)
      CALL system_clock(c1)

      ! Call the initialization of variables
      CALL Init(inputData)
      ! Setup the time step based

      IF (dt == 0.0_wp) THEN
              iflag = 0
              WRITE(*,*)'-------------------------------------------------------------'
              CALL TimeStep(iflag)
              WRITE(*,*)'-------------------------------------------------------------'
              IF (iflag == 1) STOP

      END IF

      CALL SteadySoln()
      CALL NonDim2DimVars()
      CALL UnsteadySoln()
      CALL NonDim2DimVars()

      ! Output Initial Solutions
      WRITE(*,*) 'Printing Initial Solution.....................................'
      CALL WritePlotFile(output,'"y","u","uExact","yp","up","upExact"',inputData,t)
      WRITE(*,*)'-------------------------------------------------------------------'

      ! Time loop
      DO n = 1,nmax

        t = t+dt

        CALL Dim2NonDimVars()
        CALL TriDiag()
        CALL UnsteadySoln()
        CALL NonDim2DimVars()
        CALL UnsteadyRMS()
        CALL SteadyRMS()
        MaxRMS_US = MAX(MaxRMS_US,rms_US)
        CALL WriteRMS(n,rms_SS,rms_US,rmsout,inputData)

        IF (MOD(n,inputData%setup%nout) == 0) THEN
            CALL WritePlotFile(output,'"y","u","uExact","yp","up","upExact"',
↪inputData,t)
        END IF

        IF (rms_SS < inputData%setup%RMSres) THEN
                iflag = 1
```

```
                WRITE(*,*) '-------------------------------------------------------
↪-----------'
                WRITE(*,*) 'Convergence Successful=======> Printing Solution.....
↪.....'
                CALL WritePlotFile(output,'"y","u","uExact","yp","up","upExact"',
↪inputData,t)
                WRITE(*,*) '-------------------------------------------------------
↪-----------'
                EXIT
        ENDIF

       END DO

       ! Last check for non-convergence

       IF (iflag /= 1) THEN
               WRITE(*,'(A,X,I6.6,X,A)') 'CONVERGENCE FAILURE WITH',nmax,'ITERATIONS'
       END IF

      CALL system_clock(c2)

      elapse_time = REAL(c2-c1,KIND=wp)/rate

      WRITE(*,*) ""
      WRITE(*,'(A,F10.6,A)') "| Total elapse time: ",elapse_time, " [s]|"

END PROGRAM main
```

1.5 CouetttePy

CouettePy is a python-based library of **|CouetteFlow|** that is developed to assist the user with i/o procedures, utilities and code options/testing. In the following, the commands are listed by category and discussion is provided in each respective section.

1.5.1 Compilation Options

|CouetteFlow| has several builtin compilation options, that can be accessed through the command **`./setup.py -e`** and allied utilities that can be accessed using **`./setup.py -u`**. These are described here.

Configure

The default method for compiling **|CouetteFlow|** from scratch is using the command:

```
$ ./setup.py -e configure couetteflow
```

This generates a CCMake window with configuration options that can be chosen by the user. Refer to the *compilation section* of the *|CouetteFlow| Setup page* for details on using this method.

Compile

This option should be used only if **|CouetteFlow|** has been configured first (using `-e configure`). This recompiles the code with any changes, while retaining the build directory and related objects. It can be executed using the

command:

```
$ ./setup.py -e compile couetteflow
```

1.5.2 Setting the path

|CouetteFlow| requires the path to the working directory be set every time a run is executed from scratch (i.e. after clearing all the compilations). The |CouetteFlow| executable `couette.x` will not run without this path set. To set the path, compile/configure |CouetteFlow| using any of the options, and then run:

```
$ ./setup.py -u set_path
```

1.5.3 Cleaning commands

After completion of a simulation, or before running a fresh simulation, |CouetteFlow| can be cleared of compiled objects, results and other files. There are three variants to clean |CouetteFlow|. The first is to perform a complete clean, which removes the build directories, the executables and any generated results. This can be accomplished by running:

```
$ ./setup.py -u clean heavy
```

On the other hand, the build directories and executables can be retained while deleting only the results by running the command:

```
$ ./setup.py -u clean results
```

The last variant is where the build directories alone are cleared, retaining the results and the executables, which is done using the command:

```
$ ./setup.py -u clean
```

1.5.4 Plotting Utility

|CouetteFlow| has a builtin plotting utility that allows for plotting of the generated data without the use of external tools. Similar to all the other features, the plotting utility is also accessed through the input file, which is accessed as follows.

1. Go to the parent directory:

   ```
   $ cd <path to |CouetteFlow|>
   ```

   Make sure that you are in the directory that contains the files **`setup.py`**, **`input_file.xml`**, and the folder **`src`**.

2. Open the input file:

   ```
   $ vi input_file.xml
   ```

The section of the inputfile devoted to post-processing is shown below:

```
<PostProcessing>
  <plot>
    <iPost>1</iPost>
```

```
    <Method>TecPlot</Method>
    <files>file1.dat</files>
    <style>k +r -c :g -.y --m</style>
    <label>Var1</label>
    <LegendFontSize>16</LegendFontSize>
    <FigureSize>10 7</FigureSize>
    <AxisLabelSize>21</AxisLabelSize>
    <AxisTitleSize>22</AxisTitleSize>
    <XTickSize>23</XTickSize>
    <YTickSize>24</YTickSize>
  </plot>
</PostProcessing>
```

Options in the plotting module

1. The data files generated from a |CouetteFlow| run (or any external data file) is placed in the **output** folder. The name of the file should not have any spaces or special characters (like colon :, quotation marks "" or ", brackets or parenthesis () [] etc). The file name is entered into the `<files>` field in the input file. If there are more than 1 file, they are entered one after another.
2. The `<style>` entry refers to the line style and color used. There are seven available colors in Python by default: RGBCMYK (Red, Green, Blue, Cyan, Magenta, Yellow and Black). Markers can be placed on the lines using the marker symbols (eg. `+r` generates a red line with + shaped markers). Line styles can also be changed using appropriate symbols (eg. `--m` generates a dashed magenta line, while `-.y` generates a dot-dashed yellow line).
3. The `<label>` entry is to populate the legend. If there are N files for N different variables, then the legend is populated according to the entries in this field.
4. The remaining entries are to adjust the figure parameters, like font size and figure size.

Using the plotting module

1. Enter the names of the data files in the input file
2. Set the required number of legend entries and line styles, depending on how many files/variables are being plotted.
3. Adjust any other plot parameters as required.
4. The plotting module also provides a generalized feature to make plots from input data. This feature is also accessed through the `input_file.xml`. The input data (.dat) files are entered in the `files` section of the input file. The plotting utility is then accessed using the command:

```
$ ./setup.py -p single_plot options "{<plot options>}"
```

An example of this input for the plotting utility is:

```
$ ./setup.py -p single_plot options "{'title':['RMS Residual for Grid#5'],
  'xlabel':['Number of Iterations'],'ylabel':['RMS'],
  'grid':['on'],'legend':['False'],'legend_frame':['True'],
  'legend_loc':['lower right']}"
```

The options give flexibility to set the graph title, the labels for the axes, select the columns of data from the input .dat file t

Note that if no options are provided as in the example:

```
$ ./setup.py -p single_plot options
```

default options wil be used for the picture name and axes/title labels. Note that the file and picture options must be provided in the input file, in the <PostProcessing> section.

Note: The above command is meant to be used with a generic .dat file, without header, where data is distributed column-wise.

1.5.5 Documentation

The |CouetteFlow| documentation is written using restructured text (reST) Sphinx, and can be manipulated using python routines in couettepy.

Building

The documentation can be built using the command:

```
$ ./setup.py -u doc build
```

This runs Sphinx in the documentation directory, and creates the html files using a make command. The command runs Sphinx twice to make sure references are interlinked properly.

Viewing

This command is used for viewing the documentation. It is only available after running the build command above, so that the documentation is created. The documentation can be opened using the command:

```
$ ./setup.py -u doc open
```

The documentation requires a compatible version of a web broswer (preferably Google Chrome or Mozilla Firefox). If a compatible browser is installed, the documentation will open when the above command is executed. Otherwise, an error message will be displayed. In case an error message is displayed, the documentation can manually be opened on a web browser from the directory **<GridGen parent directory>/doc/_build/html/index.html**

Cleaning

The built documentation can be cleaned (i.e. all the compiled files can be deleted while retaining the source content) using the command:

```
$ ./setup.py -u doc clean
```

Closing Notes

1. As with any browser-based content, the appearance of the content is dependent on the capability of the browser to render the elements on the webpage. Depending on the bworser present on the machine, the content may appear different.

1.6 Evaluation Cases

1.6.1 Couette Flow Results

Results summary

Note: Some of the contents in this page are animations of the solution. In order to view please visit ReadDocs

A) Show the expression for τ that non-dimensionalizes the governing PDE. Show the non-dimensionalized form of the governing PDE.

- Non-dimensionalized variables:

$$u' = \frac{u}{u_{top}}$$

$$t' = \frac{t}{\tau}$$

$$y' = \frac{y}{L}$$

$$\tau = \frac{L^2}{\nu}$$

- Non-dimensionalized governing PDE:

$$\frac{\partial u'}{\partial t'} = \frac{\partial^2 u'}{\partial y'^2}$$

B) Show the non-dimensionalized form of the time-dependent exact solution expression for the specified boundary and initial conditions given in this problem.

To find the time-dependent exact solution, we need to first find a_n which satisfies the given initial velocity profile. The resolved form of a_n is then re-written as:

$$a_n = \begin{cases} 1 \text{ if } n = 1 \\ 0 \text{ if } n \neq 1 \end{cases}$$

Thus, applying the resolved a_n into the given exact solution results in:

$$u'_{exact}(t', y') = y' + \sin(\pi y')\exp[-\pi^2 t']$$

C) Provide a brief description of the finite difference scheme (in non-dimensional form), the solution method used and exactly how the boundary and initial conditions are applied.

Given finite difference scheme has a weighting parameter θ to put an effect of implicit solution. If θ is equal to 1, the scheme becomes to fully implicit, otherwise, the scheme can be partially implicit or explicit (θ = 0). Rearranging the given finite difference equation leads to the following simplified form:

$$a_j u_{j+1}^{n+1} + b_j u_j^{n+1} + c_j u_{j-1}^{n+1} = d_j$$

where

$$a_j = -r\theta$$
$$b_j = 1 + 2r\theta$$
$$c_j = -r\theta$$
$$d_j = u_j^n + r(1-\theta)\left\{u_{j-1}^n - 2u_j^n + u_{j+1}^n\right\}$$

Here, the resulting equation has simplified coefficient $r = \frac{\Delta t'}{\Delta y'^2}$.

For the boundary condition, non-slip condition is applied to both upper and bottom plates. Thus, $y(0) = 0$ and $y(L) = 1$ remain unchanged while the inner point quantities varies during the transient phase. The initial condition described earlier can satisfy the given boundary condition here. The Thomas algorithm is set to unchange the boundary condition as the time varies.

D) Show the expression used for calculating the RMS Error relative to the time-dependent exact solution. Also show the expression used for calculating the RMS Error relative to the steady-state exact solution. Also, give a statement of the criteria used to end the calculations.

In this project, two different types of RMS error formulation are used:

- RMS error relative to the exact time-dependent solution

$$\mathrm{RMS_{NSS}}(t) = \sqrt{\frac{1}{N}\sum_{\mathrm{j=2}}^{\mathrm{jmax}-1}\left[\left(u'_{exact,j}(t) - u'^n\right)^2\right]}$$

  where N is number of inner grid points.

- RMS error relative to the exact steady-state solution:

$$\mathrm{RMS_{SS}}(t) = \sqrt{\frac{1}{N}\sum_{\mathrm{j=2}}^{\mathrm{jmax}-1}\left[\left(u'_{exact,j}(t=\infty) - u'^n\right)^2\right]}$$

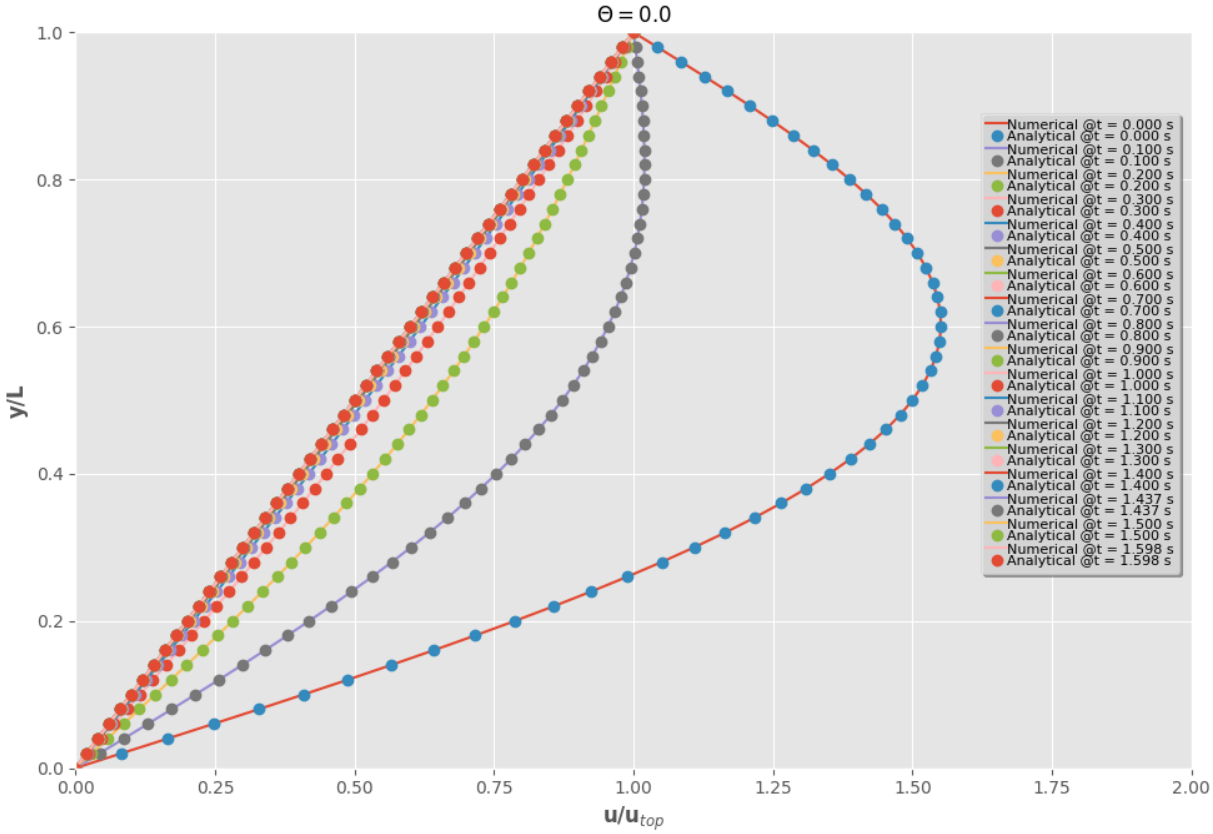
- The convergence criteria is limited by the following relation:

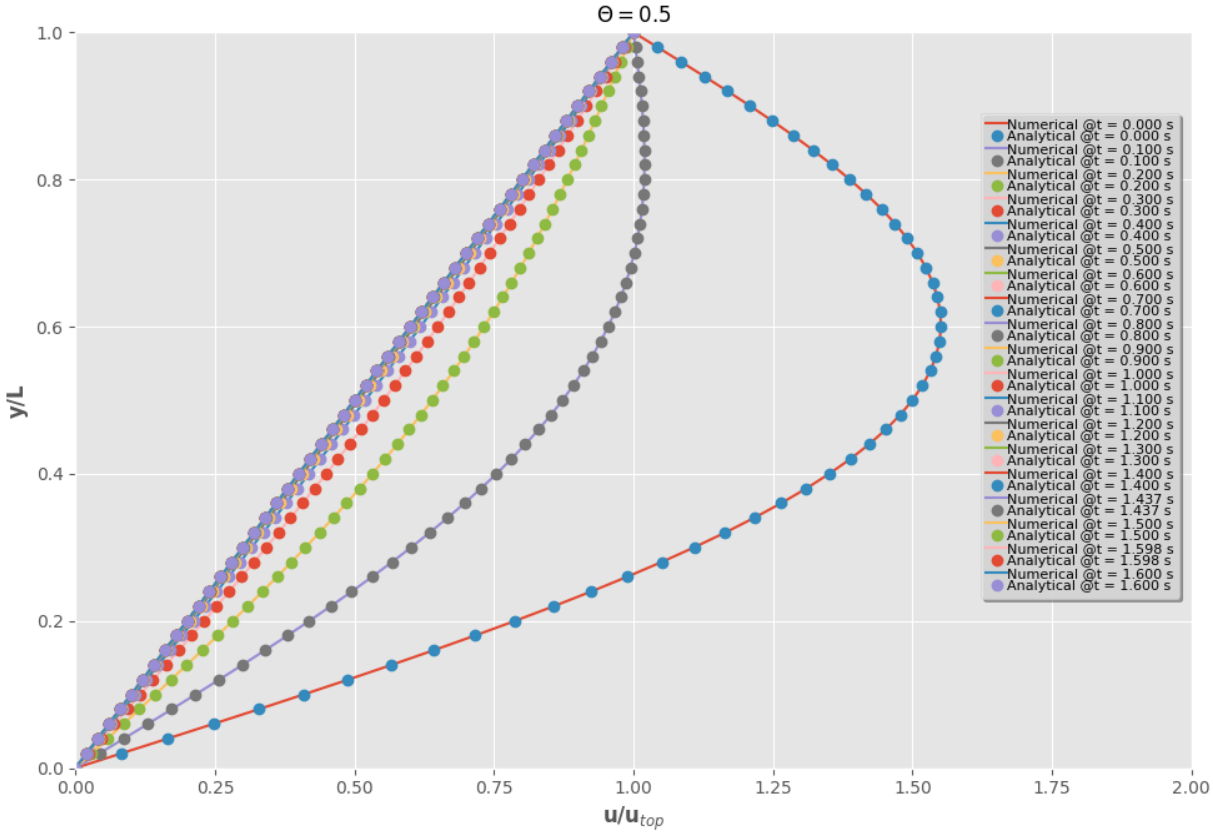
$$\mathrm{RMS_{SS}}(t) \leqslant 1 \times 10^{-7}$$

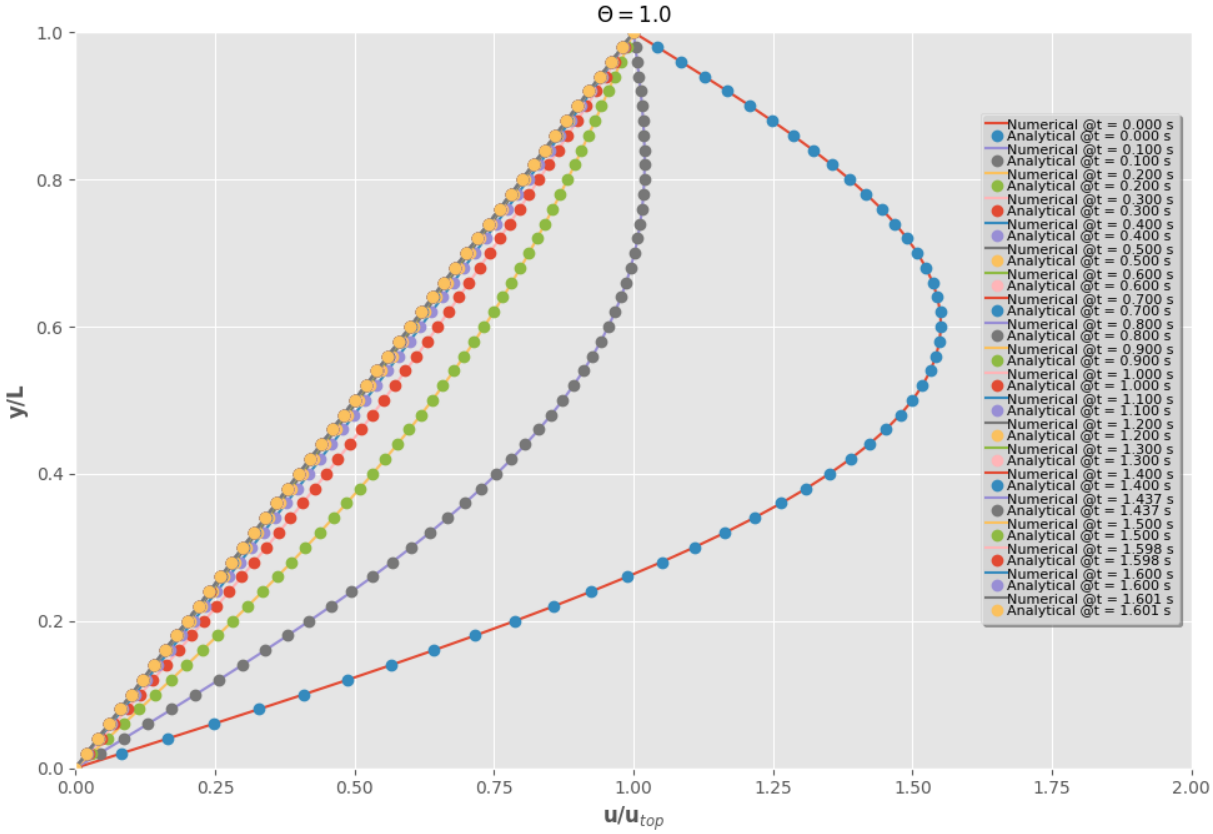
F) For $\theta = 0$, present a graph which clearly shows the progression of velocity profiles during the flow development when jmax $= 51$. The plot should show the initial profile, final steady state profile and at least 3 other non-steady-state profiles (i.e. all on the same plot). Overlay the exact numerical velocity profiles on this plot for the same points in time. Create similar plots for $\theta = 1/2$ and $\theta = 1$.

In this problem, the time step was employed as $\Delta t'$ = 0.0002 in order to have stable convergence for every θ cases. This time step was then applied to the other θ cases. As the following three figures show, the numerical solution well follows the analytical solution in both time and spatial domain.

1. θ = 0 (Fully explicit scheme): Converged at iteration number of 7990.
2. θ = 0.5 (Crank-Nicolson scheme): Converged at iteration number of 7998.
3. θ = 1 (Fully implicit scheme): Converged at iteration number of 8006.


Θ = 0.0
y/L
u/u_top
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Numerical @t = 0.000 s
Analytical @t = 0.000 s
Numerical @t = 0.100 s
Analytical @t = 0.100 s
Numerical @t = 0.200 s
Analytical @t = 0.200 s
Numerical @t = 0.300 s
Analytical @t = 0.300 s
Numerical @t = 0.400 s
Analytical @t = 0.400 s
Numerical @t = 0.500 s
Analytical @t = 0.500 s
Numerical @t = 0.600 s
Analytical @t = 0.600 s
Numerical @t = 0.700 s
Analytical @t = 0.700 s
Numerical @t = 0.800 s
Analytical @t = 0.800 s
Numerical @t = 0.900 s
Analytical @t = 0.900 s
Numerical @t = 1.000 s
Analytical @t = 1.000 s
Numerical @t = 1.100 s
Analytical @t = 1.100 s
Numerical @t = 1.200 s
Analytical @t = 1.200 s
Numerical @t = 1.300 s
Analytical @t = 1.300 s
Numerical @t = 1.400 s
Analytical @t = 1.400 s
Numerical @t = 1.437 s
Analytical @t = 1.437 s
Numerical @t = 1.500 s
Analytical @t = 1.500 s
Numerical @t = 1.598 s
Analytical @t = 1.598 s


Θ = 0.5
y/L
u/u_top
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Numerical @t = 0.000 s
Analytical @t = 0.000 s
Numerical @t = 0.100 s
Analytical @t = 0.100 s
Numerical @t = 0.200 s
Analytical @t = 0.200 s
Numerical @t = 0.300 s
Analytical @t = 0.300 s
Numerical @t = 0.400 s
Analytical @t = 0.400 s
Numerical @t = 0.500 s
Analytical @t = 0.500 s
Numerical @t = 0.600 s
Analytical @t = 0.600 s
Numerical @t = 0.700 s
Analytical @t = 0.700 s
Numerical @t = 0.800 s
Analytical @t = 0.800 s
Numerical @t = 0.900 s
Analytical @t = 0.900 s
Numerical @t = 1.000 s
Analytical @t = 1.000 s
Numerical @t = 1.100 s
Analytical @t = 1.100 s
Numerical @t = 1.200 s
Analytical @t = 1.200 s
Numerical @t = 1.300 s
Analytical @t = 1.300 s
Numerical @t = 1.400 s
Analytical @t = 1.400 s
Numerical @t = 1.437 s
Analytical @t = 1.437 s
Numerical @t = 1.500 s
Analytical @t = 1.500 s
Numerical @t = 1.598 s
Analytical @t = 1.598 s
Numerical @t = 1.600 s
Analytical @t = 1.600 s


Θ = 1.0
y/L
u/u_top
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Numerical @t = 0.000 s
Analytical @t = 0.000 s
Numerical @t = 0.100 s
Analytical @t = 0.100 s
Numerical @t = 0.200 s
Analytical @t = 0.200 s
Numerical @t = 0.300 s
Analytical @t = 0.300 s
Numerical @t = 0.400 s
Analytical @t = 0.400 s
Numerical @t = 0.500 s
Analytical @t = 0.500 s
Numerical @t = 0.600 s
Analytical @t = 0.600 s
Numerical @t = 0.700 s
Analytical @t = 0.700 s
Numerical @t = 0.800 s
Analytical @t = 0.800 s
Numerical @t = 0.900 s
Analytical @t = 0.900 s
Numerical @t = 1.000 s
Analytical @t = 1.000 s
Numerical @t = 1.100 s
Analytical @t = 1.100 s
Numerical @t = 1.200 s
Analytical @t = 1.200 s
Numerical @t = 1.300 s
Analytical @t = 1.300 s
Numerical @t = 1.400 s
Analytical @t = 1.400 s
Numerical @t = 1.437 s
Analytical @t = 1.437 s
Numerical @t = 1.500 s
Analytical @t = 1.500 s
Numerical @t = 1.598 s
Analytical @t = 1.598 s
Numerical @t = 1.600 s
Analytical @t = 1.600 s
Numerical @t = 1.601 s
Analytical @t = 1.601 s

G) Provides a comparison of the stability behavior of your solver to the stability analysis performed in Homework Assignment #3. Compute jmax $= 51$ cases with $\theta = 0$, $1/2$, and 1 using various values of Δt to explore the stability boundaries of your solver. Show and discuss whether or not your solver follows the theoretical stability behavior of these three numerical schemes.

A. The stability analysis can be summarized by:

- Unconditionally stable if $\theta \geqslant \frac{1}{2}$
- Conditionally stable if $0 \leqslant \theta < \frac{1}{2}$

In the case of conditionally stable scheme, the maximum time step can be determined by using below relation so that the scheme is stable with given θ.

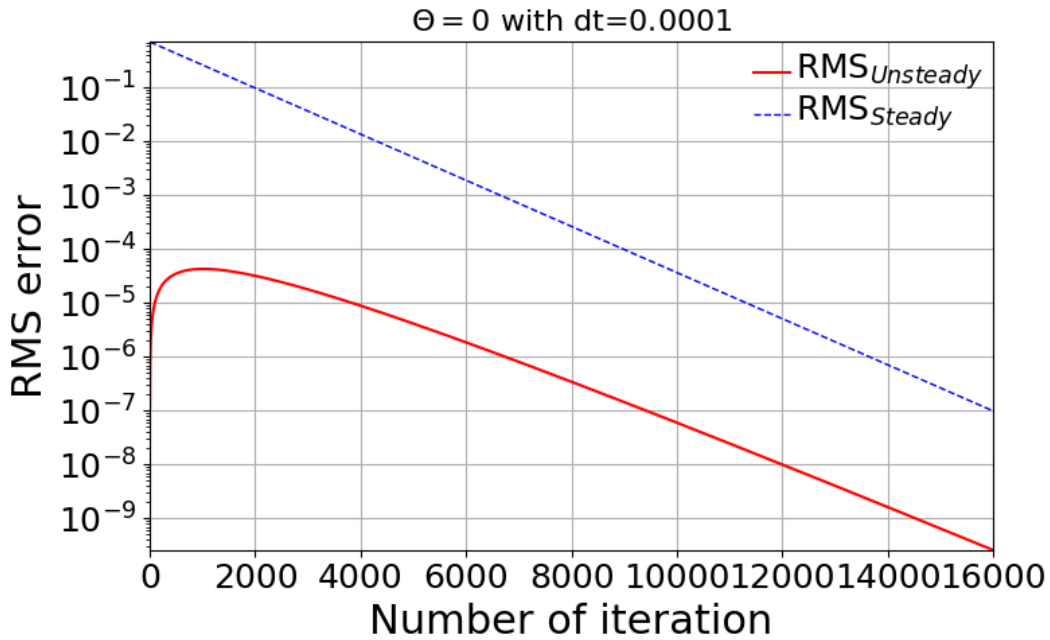
$$\Delta t \leqslant \frac{\Delta y^2}{4\left(\frac{1}{2} - \theta\right)}$$

1) θ = 0 (Fully explicit)

According to the above relation, for $\theta = 0$, the maximum time step should be 0.0002 to make the scheme stable. Following figures show the convergence history for three different time step cases: (1) ensurely stable time step, (2). maximum time step and (3). slightly bigger time-step than the maximum value. If you can't see the movies below, you are seeing the printed version of document.

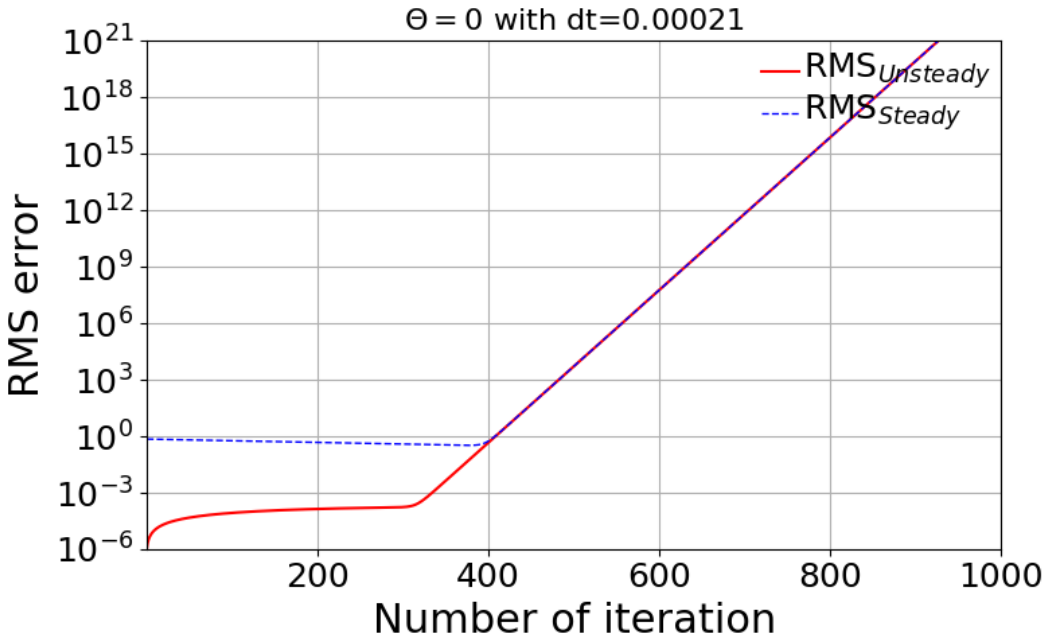
The figure below is the case with dt' = 0.0001 that is ensured for the stability for fully explicit scheme.

- $dt' = 0.0001$
  - Movie of velcoity profile (online available)
  - RMS error



Even the slightly bigger time-step causes the unstable solution and thus, the RMS error is taken off and goes to infinity after a certain number of iteration.

- $dt' = 0.00021$
  - Movie of velocity profile (online available)
  - RMS error



2) θ = 1/2 (Crank-Nicolson scheme)

- **Convergence check with the various time step:**

Non-dimensional time step $\Delta t'$	Maximum iteration for convergence
0.0001	15996
0.001	1600
0.01	160
0.1	15
1.0	39
10.0	390
100.0	3893
1000.0	38927
10000.0	389268

All the cases above seem to be stable but the convergence is strongly sensitive to how big or small time step is. The interesting pattern to be observed here is that the maximum iteration number for convergence shows quadratic behavior. That is, quite small and quite big time step require long iterations. In particular, big time steps, 1000, 10000, and 100000 for examples, take long period to make the scheme converged into the specified RMS residual. This is somewhat unphysical. If 10,000 sec is taken as a time step, it will take about 123 years for the flow to be settled down to the steady-state.

The stability check can be done by looking at the movies as a function of different time-step. If you can't see the movies below, you are seeing the printed version of document.

- $dt' = 0.0001$

The movies shown below is to show the velocity profile calculated by the present numerical solution and analytic solution. In this case, sufficiently small time-steps can ensure the physically proper behavior of the numerical solution.

  - Movie of velocity profile (online available)

- $dt' = 100$

As already mentioned above, since the given θ condition gives the stable solution, the improperly big time-step give rise to the extremely long period to have convergence. The second movie below shows the abnormal behavior of velocity profile. This may have to be involved with the inaccurate time gradient due to the big time-step, thus it leads to the negative velocity instantaneously and fluctuation of velocity profile.

  - Movie of velocity profile (online available)

3) θ = 1 (Fully implicit)

- **Convergence check with the various time step:**

Non-dimensional time step $\Delta t'$	Maximum iteration for convergence
0.0001	16004
0.001	1608
0.01	168
0.1	23
1.0	7
10.0	4
100.0	3
1000.0	2
10000.0	2

All the tested cases above are stable and the convergence performance is enhanced as the time step increases. Contrary to the Crank-Nicolson scheme case (θ = 0.5), the pattern of maximum iteration for convergence shows the linearity as a function of time step. Therefore, it can be concluded that the solver follows the theoretical stability behavior.

- $dt' = 0.0001$
  - Movie of velocity profile (online available)

$dt' = 0.1$

  - Movie of velocity profile (online available)

H) Write down an expression(s) for the truncation error (TE) of this finite difference scheme and describe the order of accuracy of the scheme for different values of θ. Note: You are not required to derive the TE expression.

$$\text{T.E.} = \left[\left(\theta - \frac{1}{2}\right)\Delta t + \frac{\Delta x^2}{12}\right] u_{xxxx} + \left[\left(\theta^2 - \theta + \frac{1}{3}\right)\Delta t^2 + \frac{1}{3}\left(\theta - \frac{1}{2}\right)\Delta t \Delta x^2 + \frac{1}{360}\Delta x^4\right] u_{xxxxxx} + \cdots$$

According to the above equation, this combined method of explicit and implicit schemes has order of accuracy in time and space as a function of θ.

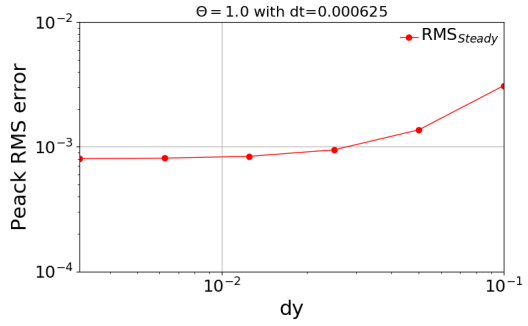
1. θ = 1/2 (Crank-Nicolson scheme): T.E. $= O\left[(\Delta t)^2, (\Delta x)^2\right]$
2. Simple explicit (θ = 0) and implicit (θ = 1): T.E. $= O\left[\Delta t, (\Delta x)^2\right]$
3. Special case ($\theta = \frac{1}{2} - \frac{(\Delta x)^2}{12\Delta t}$): T.E. $= O\left[(\Delta t)^2, (\Delta x)^4\right]$

l) Investigate the spatial order of accuracy of the code for θ = 1. Do this by using a small value of $\Delta t'$ = 0.000625 and running multiple cases of the code with different values of $\Delta y'$ (i.e. 0.1, 0.05, 0.025, 0.0125). Make a table and log-log plot of the peak RMS error (relative to the time-dependent exact solution) as a function of $\Delta y'$. Based on these results, discuss whether or not your solver follows the theoretical order of spatial accuracy given by the TE expression for the scheme. Also, explain why it is important to use a small $\Delta t'$ when we investigate the spatial accuracy of this scheme.

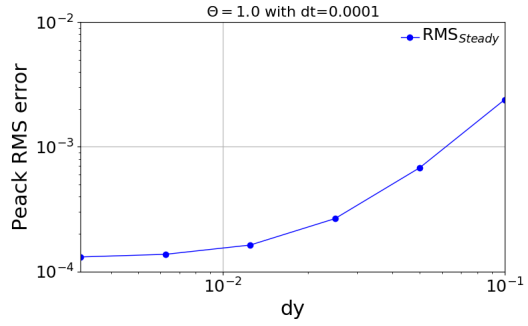
- Comparison of Peak RMS error as a function of spatial steps

dy	jmax	Max RMS error (Δt = 0.000625)	Max RMS error (Δt = 0.0001)
0.1	11	0.309370E-02	0.239121E-02
0.05	21	0.136823E-02	0.680258E-03
0.025	41	0.945456E-03	0.265312E-03
0.0125	81	0.838836E-03	0.162750E-03
0.00625	161	0.811120E-03	0.137099E-03
0.003125	321	0.803589E-03	0.130609E-03

Δt = 0.000625



Δt = 0.0001

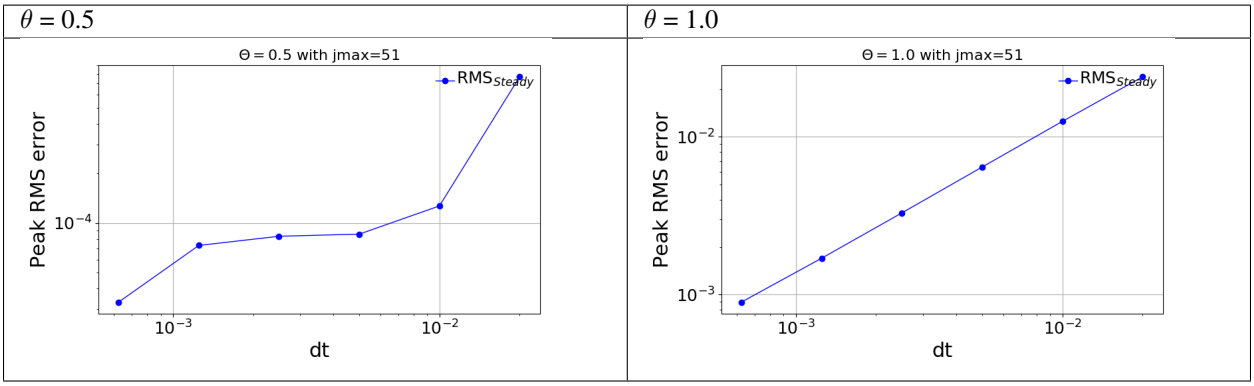


From the expression of the T.E. theoretical we see that for θ = 1, the truncation error is 1st order in time and 2nd order in space. The maximum RMS error for every test cases shows the quantitatively quadratic pattern as a function of spatial step size. Moreover, the smaller time step (here, $\Delta t'$ = 0.0001) makes this pattern more distinctive compared to the bigger time step. This is because the smaller time step can reduce the truncation error in time derivative and thus the RMS error is then significantly made by the spatial derivative terms.

J) Investigate the temporal order of accuracy of the code for θ = 1 and θ = 1/2. Do this by using jmax = 51 and various $\triangle t'$ (i.e. 0.02, 0.01, 0.005, 0.0025, 0.00125, 0.000625). Make tables and a log-log plots of the peak RMS error (relative to the time-dependent exact solution) as a function $\triangle t'$ for θ = 1 and θ = 1/2. Based on these results, discuss whether or not your solver follows the theoretical order of temporal accuracy given by the TE expression for the scheme.

- Comparison of Peak RMS error as a function of temporal steps

dt	Max RMS error (θ = 0.5)	Max RMS error (θ = 1.0)
0.02	0.769763E-03	0.240539E-01
0.01	0.126926E-03	0.125364E-01
0.005	8.561830E-05	0.643657E-02
0.0025	8.312029E-05	0.329439E-02
0.00125	7.312270E-05	0.169853E-02
0.000625	3.314359E-05	0.894558E-03



The tested results presented above show the accuracy of numerical solution as a function of time step. The previous discussion on the truncation error tells that the fully implicit scheme (θ = 1) follows the 1st order in time and the Crank-Nicolson scheme (θ = 1/2) follows the 2nd order in time.