CouetteFlow Documentation

Release 0.0.1

Sayop Kim

May 22, 2017

Contents

1 Contents **3**
- 1.1 Project description . . . 3
- 1.2 Code development . . . 4
- 1.3 How to run the code . . . 5
- 1.4 Results summary . . . 7

2 FORTRAN 90 Source code **17**
- 2.1 MakeList.txt . . . 17
- 2.2 io directory . . . 18
- 2.3 main directory . . . 20

This documentation pages are made for CFD class at Georgia Tech in 2014 Spring. This is online available at http://CouetteFlow.readthedocs.org. To view the movies, please visit this web site.

Author: Sayop Kim(sayopkim@gatech.edu)

Affiliation: School of Aerospace Engineering, Georgia Institute of Technology

CHAPTER 1

Contents

Project description

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.

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'.

CouetteFlow Code summary

The source code contains two directories, 'io', and 'main', for input/output related sources and main solver routines, respectively. 'CMakeLists.txt' file is also included for cmake compiling.

```
$ cd CouetteFlow/CODEdev/src/
$ ls
$ CMakeLists.txt  io  main
```

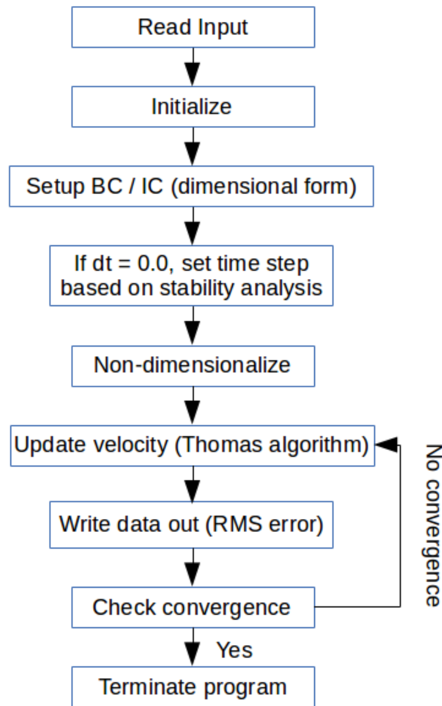
The **io** folder has **io.F90** file which contains **ReadInput()**, **WriteRMSlog()**, **WriteDataOut()** subroutines. it also includes **input** directory which contains a default **input.dat** file.

The **main** folder is only used for calculating essential subroutines required to solve the 'Couette Flow' equation by using implicit-explicit finite difference approximation. The main routine is run by **main.F90** which calls important subroutines from **main** folder itself and **io** folder when needed. All the fortran source files main folder contains are listed below:

```
> main.F90
> Parameters.F90
> SimulationSetup.F90
> SimulationVars.F90
```

Details of CouetteFlow development

The schematic below shows the flow chart of how the CouetteFlow code runs.



How to run the code

Machine platform for development

This CouetteFlow code has been developed on personal computer operating on linux system (Ubuntu Linux 3.2.0-38-generic x86_64). Machine specification is summarized as shown below:

vendor_id : GenuineIntel

cpu family : 6

model name : Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz

cpu cores : 4

Memory : 16418112 kB

Code setup

The CouetteFlow source code has been developed with version management tool, GIT. The git repository was built on 'github.com'. Thus, the source code as well as related document files can be cloned into user's local machine by following command:

```
$ git clone http://github.com/sayop/CouetteFlow.git
```

If you open the git-cloned folder **CouetteFlow**, you will see two different folders and README file. The **CODEdev** folder contains again **bin** folder, **Python** folder, and **src** folder. In order to run the code, use should run **setup.sh** script in the **bin** folder. **Python** folder contains python scripts that are used to postprocess data. It may contain **build** folder, which might have been created in the different platform. Thus it is recommended that user should remove **build** folder

before setting up the code. Note that the **setup.sh** script will run **cmake** command. Thus, make sure to have cmake installed on your system:

```
$ rm -rf build
$ ./setup.sh
-- The C compiler identification is GNU 4.6.3
-- The CXX compiler identification is GNU 4.6.3
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- The Fortran compiler identification is Intel
-- Check for working Fortran compiler: /opt/intel/composer_xe_2011_sp1.11.339/bin/
↪intel64/ifort
-- Check for working Fortran compiler: /opt/intel/composer_xe_2011_sp1.11.339/bin/
↪intel64/ifort  -- works
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Checking whether /opt/intel/composer_xe_2011_sp1.11.339/bin/intel64/ifort supports␣
↪Fortran 90
-- Checking whether /opt/intel/composer_xe_2011_sp1.11.339/bin/intel64/ifort supports␣
↪Fortran 90 -- yes
-- Configuring done
-- Generating done
-- Build files have been written to: /data/ksayop/GitHub.Clone/CouetteFlow/CODEdev/
↪bin/build
Scanning dependencies of target cfd.x
[ 20%] Building Fortran object CMakeFiles/cfd.x.dir/main/Parameters.F90.o
[ 40%] Building Fortran object CMakeFiles/cfd.x.dir/main/SimulationVars.F90.o
[ 60%] Building Fortran object CMakeFiles/cfd.x.dir/io/io.F90.o
[ 80%] Building Fortran object CMakeFiles/cfd.x.dir/main/SimulationSetup.F90.o
[100%] Building Fortran object CMakeFiles/cfd.x.dir/main/main.F90.o
Linking Fortran executable cfd.x
[100%] Built target cfd.x
```

If you run this, you will get executable named **cfd.x** and **input.dat** files. The input file is made by default. You can quickly change the required input options.

Input file setup

The CouetteFlow code allows user to set multiple options to solve the unsteady Couette flow problem by reading **input.dat** file at the beginning of the computation. Followings are default setup values you can find in the input file when you run **setup.sh** script:

```
# Input file for tecplot print
Couette Flow
uTop            1.0
distL           1.0
nu              1.0
jmax            51
theta           0.0
dt              0.0
iterMax         999999
```

```
nIterOut        500
RMSlimit        1.0e-7
```

- **First line** ('Couette Flow' by default): Project Name
- **uTop**: velocity at which top plate is moving
- **distL**: distance between top plate and bottom plate
- **nu**: viscosity coefficient
- **jmax**: spatial resolution in height (y-direction)
- **theta**: Θ, a weighting parameter for implicit scheme running
- **dt**: temporal step size
- **iterMax**: Maximum number of iteration to be converged. If the code run beyond this number, it will be forced to be terminated.
- **nIterOut**: Interval of time step between writing out the data files.
- **RMSlimit**: RMS error limit to determine convergence

Results summary

A) Result #1

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

- Non-dimensionalized variables:

  $u' = \frac{u}{u_{top}}$, $t' = \frac{t}{\tau}$, $y' = \frac{y}{L}$

  where $\tau = \frac{L^2}{\nu}$
- Non-dimensionalized governing PDE:

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

B) Result #2

Q. 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) Result #3

Q. 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

$$\begin{aligned} A_j &= -r\theta \\ B_j &= 1 + 2r\theta \\ C_j &= -r\theta \\ D_j &+ r(1-\theta)\left\{u_{j-1}^n - 2u_j^n + u_{j+1}^n\right\} \end{aligned}$$

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) Result #4

Q. 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

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

  where N is number of inner grid points.

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

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

- The convergence criteria is limited by the following relation:

  $$\text{RMS}_{\text{SS}}(t) \leqslant 1 \times 10^{-7}$$

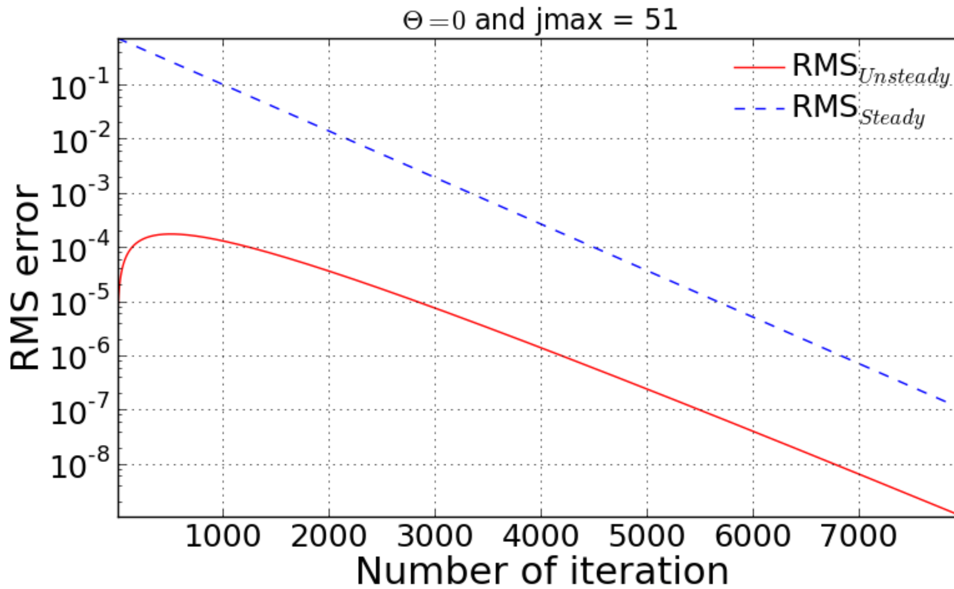
E) Result #5

Q. For $\theta = 0$ and jmax $= 51$, state the maximum value of Δt for which a stable solution is obtained. Provide a semi-log plot of the RMS error (relative to the time-dependent exact solution) vs iteration number (using a Δt for which the code is stable). Create a similar plot of the RMS error (relative to the steady-state exact solution) vs. iteration number.

A. Given conditions, the non-dimensional spatial step size results in $\Delta y' = 0.0002$. Performing Von Neumann stability analysis on the given conditions give rise to the below time step criterion:

$$\Delta t' \leqslant \frac{\Delta y'^2}{4\left(\frac{1}{2} - \theta\right)}$$

Thus, the maximum time step to stabilize the scheme is determined as $\Delta t' = 0.0002$.

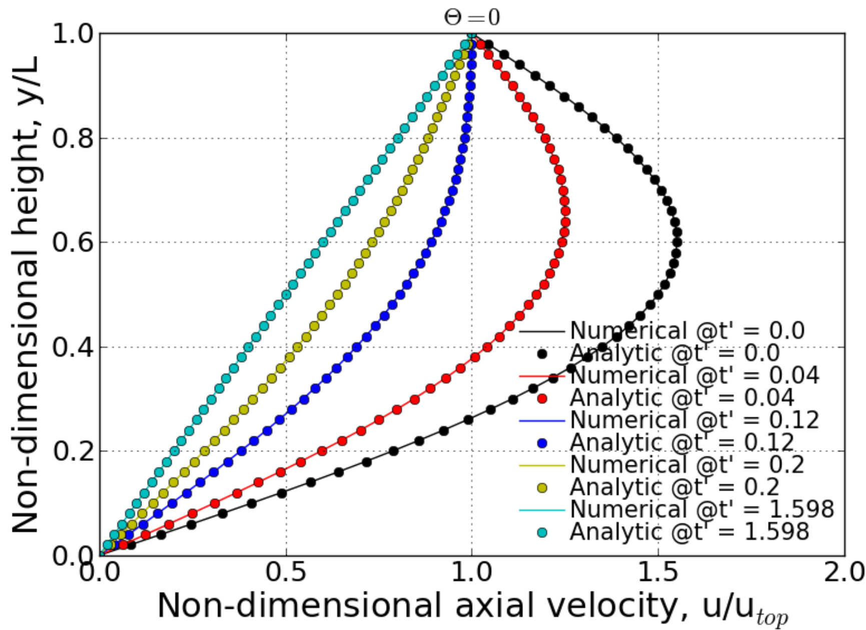


F) Result #6

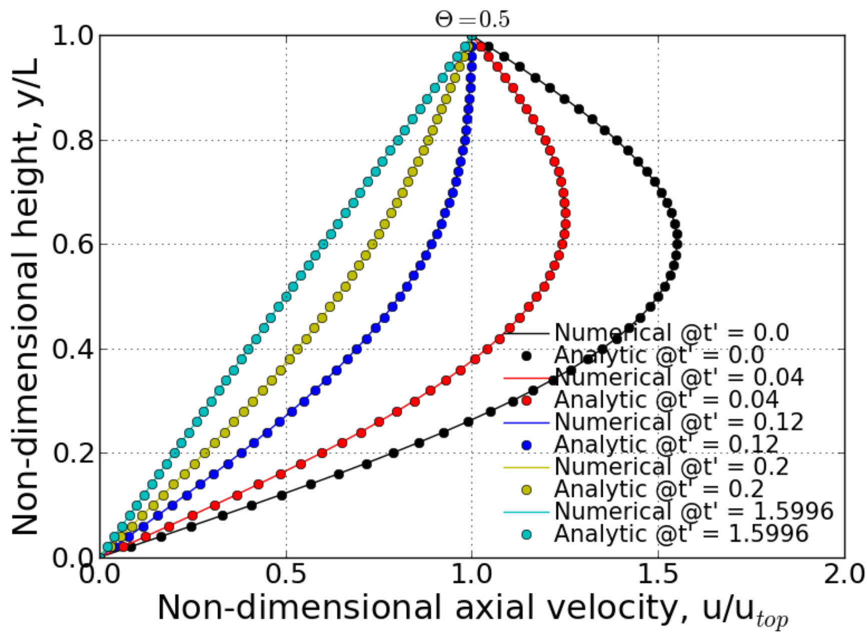
Q. For $\theta = 0$, present a graph which clearly shows the progression of velocity profiles during the flow development when jmax $= 51$. The plot should show the initial profile, final steady state profile and at least 3 other non-steady-state profiles (i.e. all on the same plot). Overlay the exact 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.

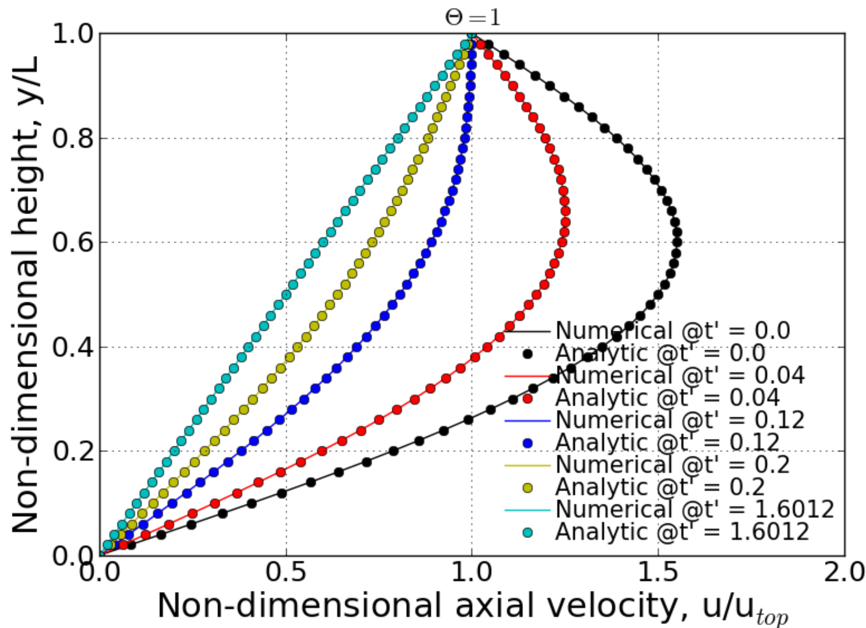
1. θ = 0 (Fully explicit scheme): Converged at iteration number of 7990.



2. $\theta = 0.5$ (Crank-Nicolson scheme): Converged at iteration number of 7998.



3. $\theta = 1$ (Fully implicit scheme): Converged at iteration number of 8006.



G) Result #7

Q. 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. From the HW#3's solution, 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.

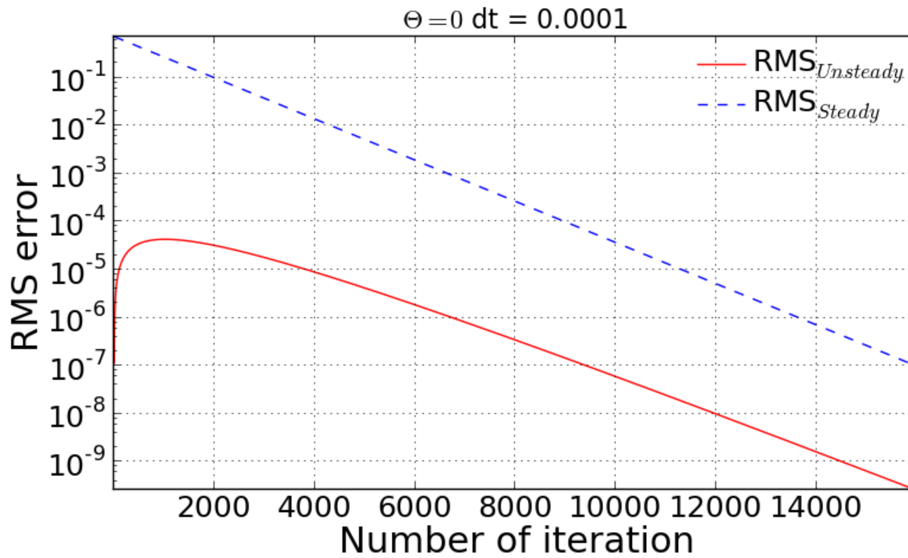
$$\Delta t \leqslant \frac{\Delta y^2}{4\left(\frac{1}{2} - \theta\right)}$$

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. If you want to see the movies, please visit: http://couetteflow.readthedocs.org/en/latest/Results/contents.html#g-result-7

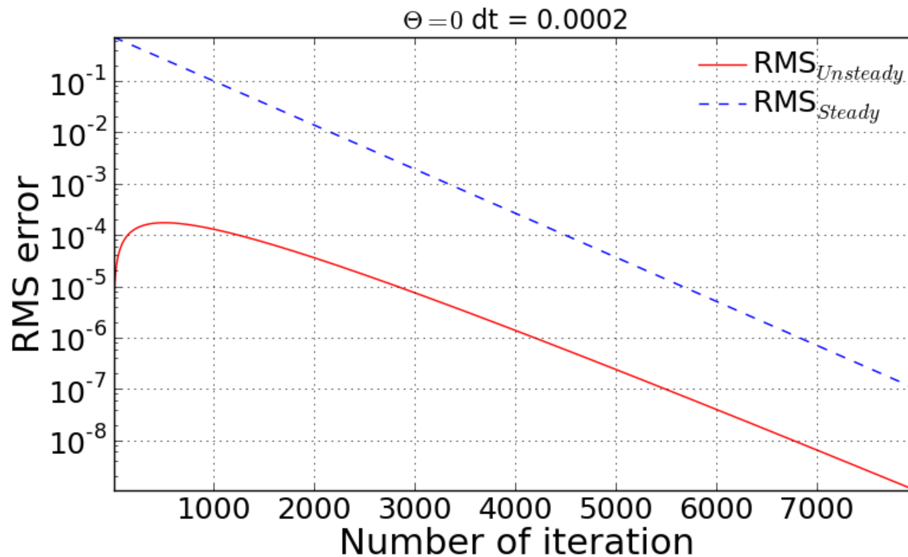
The figure below is the case with dt' = 0.0001 that is ensured for the stability for fully explicit scheme.

- $dt' = 0.0001$
  - RMS error



In this condition, the time step should not be over 0.0002 in order to obtain the stable solution. The following figures and movies prove the stability criterion in terms of time-step.

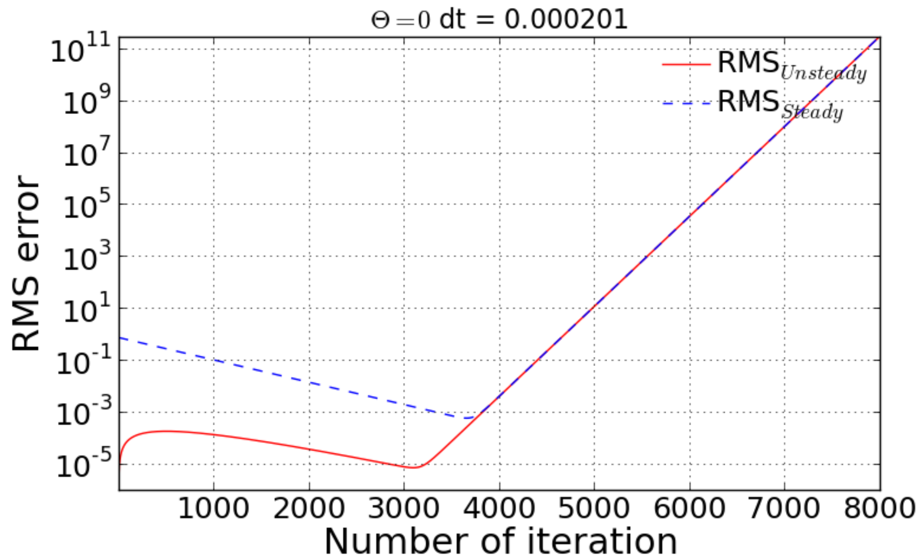
- $dt' = 0.0002$
  - RMS error



  - Movie of velocity profile (online available)

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.

- $dt' = 0.000201$
  - RMS error



– Movie of velocity profile (online available)

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
100000.0	Not converged within 999999 iterations

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. If you want to see the movies, please visit: http://couetteflow.readthedocs.org/en/latest/Results/contents.html#g-result-7

- $dt' = 0.0001$

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)

- $dt' = 1000$

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
100000.0	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.

H) Result #8

Q. 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.

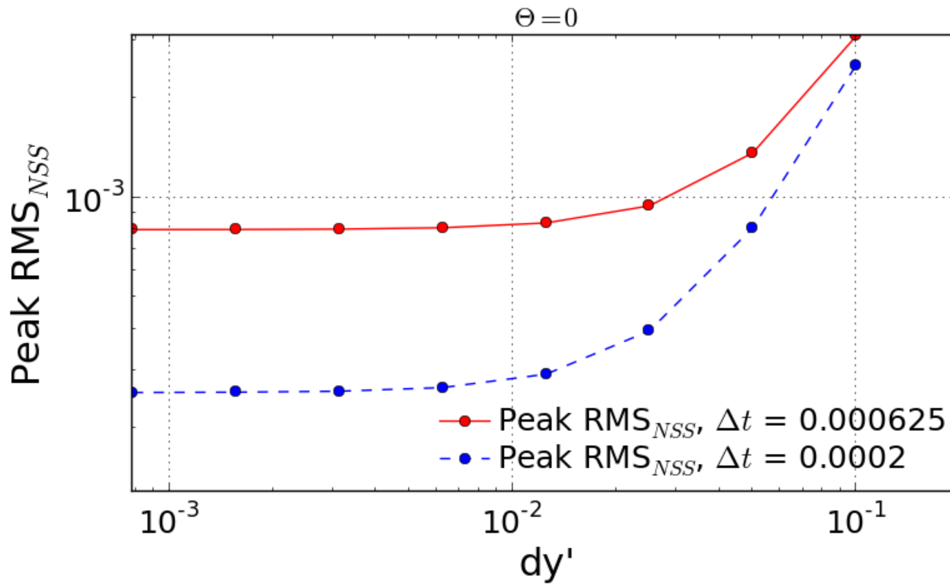
1. θ = 1/2 (Crank-Nicolson scheme): $\text{T.E.} = O\left[(\Delta t)^2, (\Delta x)^2\right]$
2. Simple explicit (θ = 0) and implicit (θ = 1): $\text{T.E.} = O\left[\Delta t, (\Delta x)^2\right]$
3. Special case ($\theta = \frac{1}{2} - \frac{(\Delta x)^2}{12\Delta t}$): $\text{T.E.} = O\left[(\Delta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

I) Result #9

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 and temporal steps

dy	jmax	Peak RMS error (Δt = 0.000625)	Peak RMS error (Δt = 0.0002)
0.1	11	0.309370E-02	0.252525E-02
0.05	21	0.136823E-02	0.811529E-03
0.025	41	0.945456E-03	0.395090E-03
0.0125	81	0.838836E-03	0.291753E-03
0.00625	161	0.811120E-03	0.265708E-03
0.003125	321	0.803589E-03	0.259019E-03
0.0015625	641	0.801397E-03	0.257250E-03
0.00078125	1281	0.800693E-03	0.256758E-03

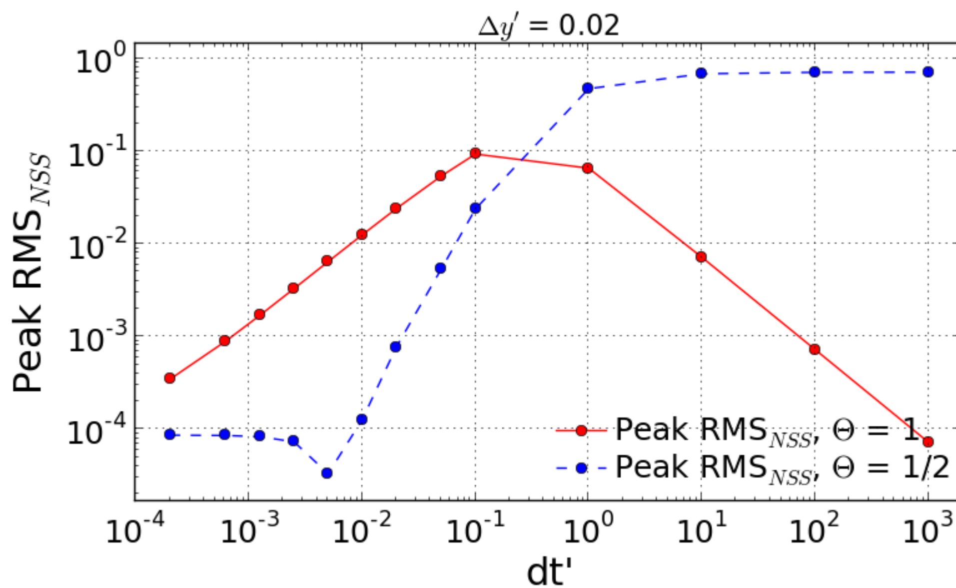


The previous theoretical analysis of accuracy investigated the order of accuracy in terms of spatial and time step size. For θ = 0, the truncation error is 1st order in time and 2nd order in space. The maximum RMS error for every test cases shows the quantitatively quadratic pattern as a function of spatial step size. Moreover, the smaller time step (here, $\Delta t'$ = 0.0002) 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) Result #10

Q. Investigate the temporal order of accuracy of the code for θ = 1 and θ = 1/2. Do this by using jmax = 51 and various $\Delta t'$ (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 $\Delta t'$ 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.

dt	Peak RMS error (θ = 1)	Peak RMS error (θ = 1/2)
1000	0.723888E-04	0.713996
100	0.723228E-03	0.711396
10	0.716697E-02	0.685903
1	0.656967E-01	0.473546
0.1	0.933255E-01	0.238631E-01
0.05	0.540879E-01	0.538846E-02
0.02	0.240539E-01	0.769763E-03
0.01	0.125364E-01	0.126926E-03
0.005	0.643658E-02	0.331436E-04
0.0025	0.329430E-02	0.731227E-04
0.00125	0.169854E-02	0.831203E-04
0.000625	0.894559E-03	0.856183E-04
0.0002	0.345497E-03	0.863658E-04



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. However, it is important to note that this analysis of accuracy is only well followable when the time step is less than 10^{-1}. This inaccuracy may have come from the spatial derivative order because the currently employed spatial step size is somewhat big enough to cause the truncation error.

more accurate numerical solution when θ value approaches to unity. However, the bigger time-step which is quite over the physically significant time scale should be avoided as already discussed earlier.Comparing two different θ cases proves that the Crank-Nicolson sheme (θ = 1/2) is more likely to ensure the accurate result only if the time step is sufficiently small. Otherwise, the bigger time step makes sure to give more accurate numerical solution when θ value approaches to unity. However, the bigger time-step which is quite over the physically significant time scale should be avoided as already discussed earlier.

CHAPTER 2

FORTRAN 90 Source code

MakeList.txt

```
cmake_minimum_required(VERSION 2.6)

project(CFD)

enable_language(Fortran)

#
# add sub-directories defined for each certain purpose
#
add_subdirectory(main)
add_subdirectory(io)

#
# set executable file name
#
set(CFD_EXE_NAME cfd.x CACHE STRING "CFD executable name")

#
# set source files
#
set(CFD_SRC_FILES ${MAIN_SRC_FILES}
                  ${IO_SRC_FILES})

#
# define executable
#
add_executable(${CFD_EXE_NAME} ${CFD_SRC_FILES})
```

io directory

CMakeLists.txt

```
set(IO_SRC_FILES
    ${CMAKE_CURRENT_SOURCE_DIR}/io.F90 CACHE INTERNAL "" FORCE)
```

io.F90

```
!> \file: io.F90
!> \author: Sayop Kim
!> \brief: Provides routines to read input and write output

MODULE io_m
   USE Parameters_m, ONLY: wp

   IMPLICIT NONE

   PUBLIC :: ReadInput, WriteRMSlog, WriteDataOut

   INTEGER :: nIterOut
   INTEGER, PARAMETER :: IOunit = 10, filenameLength = 64
   CHARACTER(LEN=50) :: prjTitle

CONTAINS

!-----------------------------------------------------------------------------!
   SUBROUTINE ReadInput()
!-----------------------------------------------------------------------------!
!  Read input files
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: uTop, jmax, distL, nu, theta, dt, iterMax, &
                                  RMSlimit
      IMPLICIT NONE
      INTEGER :: ios
      CHARACTER(LEN=8) :: inputVar

      OPEN(IOunit, FILE = 'input.dat', FORM = 'FORMATTED', ACTION = 'READ', &
            STATUS = 'OLD', IOSTAT = ios)
      IF(ios /= 0) THEN
         WRITE(*,'(a)') ""
         WRITE(*,'(a)') "### Fatal error: Could not open the input data file."
         RETURN
      ELSE
         WRITE(*,'(a)') ""
         WRITE(*,'(a)') "### Reading input file for Couette Flow problem"
      ENDIF

      READ(IOunit,*)
      READ(IOunit,'(a)') prjTitle
      WRITE(*,'(4a)') '### Project Title:', '"',TRIM(prjTitle),'"'
      READ(IOunit,*) inputVar, uTop
      WRITE(*,'(a,f6.3)') inputVar, uTop
      READ(IOunit,*) inputVar, distL
      WRITE(*,'(a,f6.3)') inputVar, distL
```

```
      READ(IOunit,*) inputVar, nu
      WRITE(*,'(a,f6.3)') inputVar, nu
      READ(IOunit,*) inputVar, jmax
      WRITE(*,'(a,i6)') inputVar, jmax
      READ(IOunit,*) inputVar, theta
      WRITE(*,'(a,f6.3)') inputVar, theta
      READ(IOunit,*) inputVar, dt
      WRITE(*,'(a,f6.3)') inputVar, dt
      READ(IOunit,*) inputVar, iterMax
      WRITE(*,'(a,i6)') inputVar, iterMax
      READ(IOunit,*) inputVar, nIterOut
      WRITE(*,'(a,i6)') inputVar, nIterOut
      READ(IOunit,*) inputVar, RMSlimit
      WRITE(*,'(a,g15.6)') inputVar,RMSlimit
   END SUBROUTINE ReadInput

!------------------------------------------------------------------------------!
   SUBROUTINE WriteRMSlog(iter)
!------------------------------------------------------------------------------!
!  Write RMS error log relative to unsteady and steady solutions
!------------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: RMSerrUS, RMSerrSS

      IMPLICIT NONE
      INTEGER :: iter
      CHARACTER(LEN=filenameLength) :: fileName = 'RMSlog.dat'

      IF(iter .EQ. 1) THEN
         OPEN(IOunit, File = fileName, FORM = 'FORMATTED', ACTION = 'WRITE')
         WRITE(IOunit,'(A,3A10)') '#', 'Iteration', 'RMSerrUS', 'RMSerrSS'
      ELSE
         OPEN(IOunit, File = fileName, FORM = 'FORMATTED', ACTION = 'WRITE', &
              POSITION = 'APPEND')
      ENDIF

      WRITE(IOunit,'(i6,2g15.6)') iter, RMSerrUS, RMSerrSS
      CLOSE(IOunit)

   END SUBROUTINE WriteRMSlog

!------------------------------------------------------------------------------!
   SUBROUTINE WriteDataOut(iter,time)
!------------------------------------------------------------------------------!
!  Write RMS error log relative to unsteady and steady solutions
!------------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: y, yp, u, up, uExac, upExac, jmax

      IMPLICIT NONE
      INTEGER :: iter, j
      REAL(KIND=wp) :: time
      CHARACTER(LEN=filenameLength) :: fileName

      WRITE(fileName,'(A,i6.6,A)') "Data_", iter, ".dat"
      WRITE(*,'(2A)') "PRINTING FILE:", fileName
      OPEN(IOunit, File = fileName, FORM = 'FORMATTED', ACTION = 'WRITE')
      WRITE(IOunit,'(A,g15.6)') "#Time=", time
      WRITE(IOunit,'(A,6A15)') "#", "y", "u", "u_exac", "y'", "u'", "u'_exac"
      DO j = 1, jmax
```

```
         WRITE(IOunit,'(6g15.6)') y(j), u(j), uExac(j), yp(j), up(j), upExac(j)
      END DO
      CLOSE(IOunit)

   END SUBROUTINE WriteDataOut
END MODULE io_m
```

main directory

CMakeLists.txt

```
set(MAIN_SRC_FILES
   ${CMAKE_CURRENT_SOURCE_DIR}/main.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/SimulationSetup.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/SimulationVars.F90
   ${CMAKE_CURRENT_SOURCE_DIR}/Parameters.F90 CACHE INTERNAL "" FORCE)
```

main.F90

```
!> \file: main.F90
!> \author: Sayop Kim

PROGRAM main
   USE Parameters_m, ONLY: wp
   USE io_m, ONLY: ReadInput, WriteRMSlog, WriteDataOut, nIterOut
   USE SimulationVars_m, ONLY: t, tp, dt, iterMax, RMSerrSS, RMSerrUS, &
                               RMSlimit
   USE SimulationSetup_m, ONLY: Initialize, SetupBCIC, SetTimeStep, &
                                UpdateNonDimVars, UpdateDimVars, &
                                UpdateVelocity, CalSteadyExactSol, &
                                CalUnSteadyExactSol, CalRMSerrUnsteady, &
                                CalRMSerrSteady

   IMPLICIT NONE
   INTEGER :: iKill, nIter, iCONVERGE
   REAL(KIND=wp) :: MaxRMSerrUS
   MaxRMSerrUS = 0.0_wp

   CALL ReadInput()
   CALL Initialize()
   CALL SetupBCIC()
   IF(dt .EQ. 0.0_wp) THEN
      iKill = 0
      CALL SetTimeStep(iKill)
      IF(iKill .EQ. 1) STOP
   END IF

   CALL CalSteadyExactSol()
   CALL CalUnSteadyExactSol()
   CALL WriteDataOut(0,t)
   TimeLoop: DO nIter = 1, iterMax
      t = t + dt
```

```
      CALL UpdateNonDimVars()
      CALL UpdateVelocity()
      CALL CalUnSteadyExactSol()
      CALL CalRMSerrUnsteady
      CALL CalRMSerrSteady
      MaxRMSerrUS = MAX(MaxRMSerrUS,RMSerrUS)
      CALL WriteRMSlog(nIter)
      IF(MOD(nIter, nIterOut) .EQ. 0) THEN
         CALL WriteDataOut(nIter,t)
      ENDIF
      IF(RMSerrSS .LT. RMSlimit) THEN
         iCONVERGE = 1
         WRITE(*,'(A)') "### CONVERGENCE IS SUCCESSFULLY ACHIEVED!!!"
         CALL WriteDataOut(nIter,t)
         EXIT
      ENDIF
   END DO TimeLoop
   IF(iCONVERGE .NE. 1) THEN
      WRITE(*,'(A,I6.6,A)') "### CONVERGENCE IS NOT ACHIEVED WITHIN ",iterMax," ␣
↪ITERATIONS!!!"
   ENDIF
   WRITE(*,'(A,g15.6)') "### Maximum RMS error based on Unsteady-State: ", MaxRMSerrUS

END PROGRAM main
```

Parameters.F90

```
!> \file parameters.F90
!> \author Sayop Kim
!> \brief Provides parameters and physical constants for use throughout the
!! code.
MODULE Parameters_m
   INTEGER, PARAMETER :: wp = SELECTED_REAL_KIND(8)

   CHARACTER(LEN=10), PARAMETER :: CODE_VER_STRING = "V.001.001"
   REAL(KIND=wp), PARAMETER :: PI = 3.14159265358979323846264338_wp

END MODULE Parameters_m
```

SimulationVars.F90

```
!> \file: SimulationVars.F90
!> \author: Sayop Kim

MODULE SimulationVars_m
   USE parameters_m, ONLY : wp
   IMPLICIT NONE

   INTEGER :: jmax, iterMax
   REAL(KIND=wp), ALLOCATABLE, DIMENSION(:) :: yp, up, y, u, &
                                               upExac, upExacSS, &
                                               uExac, uExacSS
   REAL(KIND=wp) :: t, dt, dy
   REAL(KIND=wp) :: uTop, distL, nu, theta, tp, dtp, dyp
```

```
   REAL(KIND=wp) :: RMSerrSS, RMSerrUS, RMSlimit

END MODULE SimulationVars_m
```

SimulationSetup.F90

```
!> \file SimulationSetup.F90
!> \author Sayop Kim

MODULE SimulationSetup_m
   USE Parameters_m, ONLY: wp
   IMPLICIT NONE

   PUBLIC :: Initialize, SetupBCIC, SetTimeStep, UpdateNonDimVars, &
             UpdateDimVars, UpdateVelocity, CalSteadyExactSol, &
             CalUnSteadyExactSol, CalRMSerrSteady, CalRMSerrUnsteady

CONTAINS

!-----------------------------------------------------------------------------!
   SUBROUTINE Initialize()
!-----------------------------------------------------------------------------!
      USE Parameters_m, ONLY: CODE_VER_STRING
      USE SimulationVars_m, ONLY: t, tp, yp, up, y, u, jmax, uExac, uExacSS, &
                                  upExac, upExacSS
      IMPLICIT NONE

      ALLOCATE(yp(jmax))
      ALLOCATE(up(jmax))
      ALLOCATE(y(jmax))
      ALLOCATE(u(jmax))
      ALLOCATE(uExac(jmax))
      ALLOCATE(uExacSS(jmax))
      ALLOCATE(upExac(jmax))
      ALLOCATE(upExacSS(jmax))

      WRITE(*,'(a)') ""
      WRITE(*,'(3a)') "### CFD code Version: ", CODE_VER_STRING, "###"

      t = 0.0_wp
      tp = 0.0_wp
      y = 0.0_wp
      yp = 0.0_wp
      u = 0.0_wp
      up = 0.0_wp
      uExac = 0.0_wp
      uExacSS = 0.0_wp
   END SUBROUTINE Initialize

!-----------------------------------------------------------------------------!
   SUBROUTINE SetupBCIC()
!-----------------------------------------------------------------------------!
!   Setup Boundary Conditions and Initial Conditions
!-----------------------------------------------------------------------------!
      USE Parameters_m, ONLY: PI
      USE SimulationVars_m, ONLY: y, u, dy, &
```

```
                                        jmax, yp, up, distL, dyp, uTop

      IMPLICIT NONE
      INTEGER :: j

      WRITE(*,'(a)') ""
      WRITE(*,'(a)') "### Setup Initial Condition and Boundary Condition"

      ! Set y coordinate and initial condition
      dy = distL / (jmax - 1)
      DO j = 1, jmax
         y(j) = dy * (j - 1)
         !
         ! u(y) = u_top * ( y' + sin(pi x y') )
         !
         u(j) = uTop * ( y(j)/distL + sin(PI * y(j)/distL) )
      END DO

      WRITE(*,'(a,g15.6)') "### dy = ", dy
      CALL UpdateNonDimVars()

   END SUBROUTINE SetupBCIC

!-----------------------------------------------------------------------------!
   SUBROUTINE SetTimeStep(iKill)
!-----------------------------------------------------------------------------!
!   Setup computational time step based on Von-Neumann stability analysis
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: dt, dtp, dyp, theta

      IMPLICIT NONE
      INTEGER :: iKill

      IF(theta .GE. 0.5_wp) THEN
         WRITE(*,'(a)') ""
         WRITE(*,'(a)') "### Unconditionally stable!!"
         WRITE(*,'(a)') "### Input any value of 'dt' in input.dat and rerun!!"
         iKill = 1
      ELSE
         dtp = dyp**2 / 4.0_wp / (0.5_wp - theta)   ! Non-Dimensionalized form
         CALL UpdateDimVars()
         WRITE(*,'(a)') ""
         WRITE(*,'(a)') "### Setup Time Step for stable running"
         WRITE(*,'(a,g15.6)') "### This scheme is stable if dt is equal to or less␣
↪than", dt
         WRITE(*,'(a,g15.6)') "### dt is selected as ", dt
         WRITE(*,'(a,g15.6)') "### Non-Dimensionalized time step 'dtp' is selected as
↪", dtp
         iKill = 0
      END IF
   END SUBROUTINE SetTimeStep

!-----------------------------------------------------------------------------!
   SUBROUTINE UpdateNonDimVars()
!-----------------------------------------------------------------------------!
!  Update Non-dimensionalized variables from dimensional variables
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: t, dt, y, u, dy, &
```

```
                                tp, dtp, yp, up, dyp, nu, distL, uTop, &
                                upExac, upExacSS, uExac, uExacSS

      IMPLICIT NONE
      REAL(KIND=wp) :: tau

      tau = distL / nu

      tp = t / tau
      yp = y / distL
      up = u / uTop
      dtp = dt / tau
      dyp = dy / distL
      upExac = uExac / uTop
      upExacSS = uExacSS / uTop

   END SUBROUTINE UpdateNonDimVars

!-----------------------------------------------------------------------------!
   SUBROUTINE UpdateDimVars()
!-----------------------------------------------------------------------------!
!  Update dimensionalized variables from Non-dimensional variables
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: t, dt, y, u, dy, &
                                tp, dtp, yp, up, dyp, nu, distL, uTop, &
                                upExac, upExacSS, uExac, uExacSS

      IMPLICIT NONE
      REAL(KIND=wp) :: tau

      tau = distL / nu

      t = tp * tau
      y = yp * distL
      u = up * uTop
      dt = dtp * tau
      dy = dyp * distL
      uExac = upExac * uTop
      uExacSS = upExacSS * uTop

   END SUBROUTINE UpdateDimVars

!-----------------------------------------------------------------------------!
   SUBROUTINE UpdateVelocity()
!-----------------------------------------------------------------------------!
!   Setup Tri-Diagonal matrix for solving Thomas Loop
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: jmax, dtp, dyp, theta, up

      IMPLICIT NONE
      INTEGER :: j

      REAL(KIND=wp) :: rr
      REAL(KIND=wp), DIMENSION(jmax) :: A, B, C, D

      rr = dtp / dyp**2
      DO j = 1, jmax
         IF( j == 1 .or. j == jmax ) THEN
```

```
            A(j) = 0.0_wp
            B(j) = 1.0_wp
            C(j) = 0.0_wp
            D(j) = up(j)
         ELSE
            A(j) = -rr * theta
            B(j) = 1.0_wp + 2.0_wp * rr * theta
            C(j) = -rr * theta
            D(j) = up(j) + rr * (1.0_wp - theta) * (up(j-1) - 2.0_wp*up(j) +up(j+1))
         END IF
      END DO

      ! Call Thomas method solver
      CALL SY(1, jmax, A, B, C, D)

      DO j = 1, jmax
         up(j) = D(j)
      END DO
   END SUBROUTINE UpdateVelocity

!-----------------------------------------------------------------------------!
   SUBROUTINE SY(IL,IU,BB,DD,AA,CC)
!-----------------------------------------------------------------------------!
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: IL, IU
      REAL(KIND=wp), DIMENSION(IL:IU), INTENT(IN) :: AA, BB
      REAL(KIND=wp), DIMENSION(IL:IU), INTENT(INOUT) :: CC, DD

      INTEGER :: LP, I, J
      REAL(KIND=wp) :: R

      LP = IL + 1

      DO I = LP, IU
         R = BB(I) / DD(I-1)
         DD(I) = DD(I) - R*AA(I-1)
         CC(I) = CC(I) - R*CC(I-1)
      ENDDO

      CC(IU) = CC(IU)/DD(IU)
      DO I = LP, IU
         J = IU - I + IL
         CC(J) = (CC(J) - AA(J)*CC(J+1))/DD(J)
      ENDDO
   END SUBROUTINE SY

!-----------------------------------------------------------------------------!
   SUBROUTINE CalSteadyExactSol()
!-----------------------------------------------------------------------------!
!  Calculate Steady State Solution: used at one time
!  USE Non-Dimensionalized variables only!
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: upExacSS, yp, jmax

      IMPLICIT NONE

      upExacSS = yp
      CALL UpdateDimVars
```

```
   END SUBROUTINE CalSteadyExactSol

!-----------------------------------------------------------------------------!
   SUBROUTINE CalRMSerrSteady()
!-----------------------------------------------------------------------------!
!  Calculate RMS error relative to the Steady-State exact solution
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: up, upExacSS, RMSerrSS, jmax

      IMPLICIT NONE
      INTEGER :: j
      REAL(KIND=wp) :: rr

      rr = 0.0_wp
      DO j = 2, jmax - 1
         rr = rr + (upExacSS(j) - up(j))**2
      END DO
      RMSerrSS = (rr / (jmax-2)) ** 0.5_wp

   END SUBROUTINE CalRMSerrSteady

!-----------------------------------------------------------------------------!
   SUBROUTINE CalRMSerrSteady()
!-----------------------------------------------------------------------------!
!  Calculate RMS error relative to the Steady-State exact solution
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: up, upExacSS, RMSerrSS, jmax

      IMPLICIT NONE
      INTEGER :: j
      REAL(KIND=wp) :: rr

      rr = 0.0_wp
      DO j = 2, jmax - 1
         rr = rr + (upExacSS(j) - up(j))**2
      END DO
      RMSerrSS = (rr / (jmax-2)) ** 0.5_wp

   END SUBROUTINE CalRMSerrSteady

!-----------------------------------------------------------------------------!
   SUBROUTINE CalUnSteadyExactSol()
!-----------------------------------------------------------------------------!
!  Calculate Steady State Solution: updated every time step
!-----------------------------------------------------------------------------!
      USE Parameters_m, ONLY: PI
      USE SimulationVars_m, ONLY: up, upExac, tp, yp, jmax

      IMPLICIT NONE

      upExac = yp + sin(PI * yp) * exp(-PI**2 * tp)
      CALL UpdateDimVars
   END SUBROUTINE CalUnSteadyExactSol

!-----------------------------------------------------------------------------!
   SUBROUTINE CalRMSerrUnSteady()
!-----------------------------------------------------------------------------!
!  Calculate RMS error relative to the Steady-State exact solution
```

```
!-----------------------------------------------------------------------------!
      USE SimulationVars_m, ONLY: up, upExac, RMSerrUS, jmax

      IMPLICIT NONE
      INTEGER :: j
      REAL(KIND=wp) :: rr

      rr = 0.0_wp
      DO j = 2, jmax - 1
         rr = rr + (upExac(j) - up(j))**2
      END DO
      RMSerrUS = (rr / (jmax-2)) ** 0.5_wp

   END SUBROUTINE CalRMSerrUnSteady
END MODULE SimulationSetup_m
```