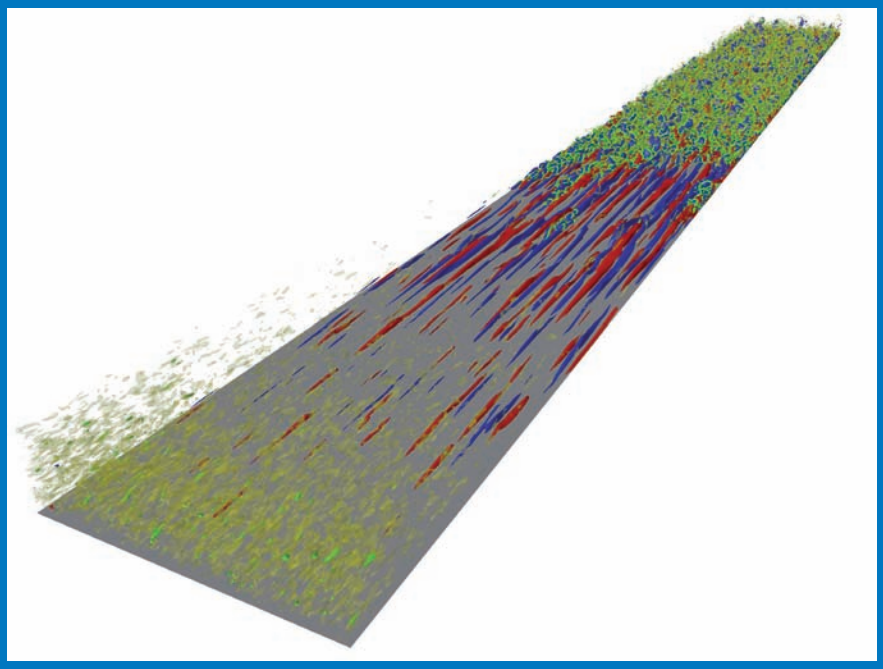
KTH Engineering Sciences

SIMSON

A Pseudo-Spectral Solver for
Incompressible Boundary Layer Flows



MATTIAS CHEVALIER, PHILIPP SCHLATTER,
ANDERS LUNDBLADH AND DAN S. HENNINGSON

Technical Report
Stockholm, Sweden 2007

ISSN 0348-467x
ISBN 978-91-7178-838-2
TRITA-MEK 2007:07

KTH Mechanics
KTH, SE-100 44 Stockholm
www.kth.se

SIMSON

A Pseudo-**S**pectral Solver for **I**nco**M**pre**S**sible B**O**u**N**dary Layer Flows

Mattias Chevalier, Philipp Schlatter,
Anders Lundbladh and Dan S. Henningson

KTH Mechanics
TRITA-MEK 2007:07

This is software which is distributed freely on a limited basis; it comes with no guarantees whatsoever. Problems can be reported to henning@mech.kth.se, but no action is promised. If results obtained by using the programs included in the **Simson** distribution are published the authors would like an acknowledgment, *e.g.* in the form of referencing this code manual.

The front page image is a visualization of laminar-turbulent transition induced by ambient free-stream turbulence convected above a flat plate, *i.e.* so-called bypass transition (Brandt *et al.*, 2004). The large-eddy simulation used to generate the image (Schlatter *et al.*, 2006) is performed with Simson using the ADM-RT subgrid-scale model (Schlatter *et al.*, 2004), further postprocessed using the program **lambda2** and rendered using **OpenDX**. Low and high speed streaks are visualized with blue and red isocontours, respectively; green and yellow isocontours indicate the λ_2 vortex identification criterion (Jeong & Hussain, 1995). The flow is from lower left to upper right.

This user guide is compiled from revision 1047.

ISBN 978-91-7178-838-2

Contents

1 Introduction

This report is a part of **Simson** version 4.0.0, a software package that implements an efficient spectral integration technique to solve the Navier–Stokes equations for incompressible channel and boundary layer flows. The report describes how to configure, compile and install the software. Additionally, an introduction to the theory and the numerical details of the implementation is given.

The solver is implemented in Fortran 77/90. The original algorithm reported in Lundbladh *et al.* (1992a) solved the incompressible Navier–Stokes equations in a channel flow geometry. That algorithm has been reimplemented in a boundary layer version of the code reported in Lundbladh *et al.* (1999). That allowed simulations of the flow over a flat plate. To do this an artificial free-stream boundary condition was introduced, and for spatial simulations a fringe region technique similar to that of Bertolotti *et al.* (1992) was implemented. In **Simson** the channel and boundary layer solvers have been combined together with many different features developed over the years.

The code can be run either as a solver for direct numerical simulation (DNS) in which all length and time scales are resolved, or in large-eddy simulation (LES) mode where a number of different subgrid-scale models are available. The evolution of multiple passive scalars can also be computed. The code can be run with distributed or with shared memory parallelization through the Message Passing Interface (MPI) or OpenMP, respectively. The wall-parallel directions are discretized using Fourier series and the wall-normal direction using Chebyshev series. Time integration is performed using a third order Runge–Kutta method for the advective and forcing terms and a Crank–Nicolson method for the viscous terms. The basic numerical method is similar to the Fourier–Chebyshev method used by Kim *et al.* (1987). Further details about spectral discretizations and additional references are given in *e.g.* Canuto *et al.* (1988).

1.1 Contributions

The following people, in alphabetical order, have all made contributions during the development of the **Simson** code: Krister Alvelius, Shervin Bagheri, Stellan Berlin, Luca Brandt, Mattias Chevalier, Jaisig Choi, Dan Henningson, Astrid Herbst, Casper Hildings, Markus Högberg, Arne Johansson, John Kim, Ori Levin, Qiang Li, Anders Lundbladh, Linus Marstorp, Antonios Monokrousos, Philipp Schlatter, Lars-Uve Schrader, Martin Skote, Petra Wikström and Espen Åkervik.

1.2 Release notes

1.2.1 Version 4.0.0

The most prominent features of the **Simson** package are listed below.

- Fully spectral discretization with high accuracy
- Support for many different flow cases
  - Blasius boundary layer flow
  - The family of Falkner–Skan-Cooke boundary layer flows
  - Poiseuille flow
  - Couette flow
  - Asymptotic suction boundary layer flow
- Spatial and temporal flows
- Disturbance formulation and linearized equations
- Support for MPI and OpenMP parallelization
- Multiple passive scalars
- Selective frequency damping
- Low magnetic Reynolds number approximation (MHD)
- Various initial, inflow and boundary condition types

1.3 Published results

The channel and boundary layer codes have been used in a number of investigations. A selection of these publications are listed, chronologically, below. Note that some studies listed in the boundary layer section include also channel flow results.

1.3.1 Channel and Couette flow studies

1. Optimal secondary energy growth in a plane channel flow (Cossu *et al.* (2007) and some additional derivations in Chevalier (2004))
2. State estimation in wall-bounded flow systems, Part 1. Laminar flows (Hœpffner *et al.*, 2005)
3. Optimal control of bypass transition (Högberg *et al.*, 2000)
4. On stability of streamwise streaks and transition thresholds in plane channel flows (Reddy *et al.*, 1998)
5. Bypass transition and linear growth mechanisms (Henningson, 1995)
6. Ribbon induced oblique transition in plane Poiseuille flow (Elofsson & Lundbladh, 1994)
7. Bounds for threshold amplitudes in subcritical shear flows (Kreiss *et al.*, 1994)
8. Threshold amplitudes for transition in channel flows (Lundbladh *et al.*, 1994a)
9. Spatial evolution of disturbances in plane Poiseuille flow (Schmid *et al.*, 1994)
10. Growth of a localized disturbance in inviscidly stable shear flow (Lundbladh, 1993)
11. A mechanism for bypass transition from localized disturbances in wall-bounded shear flows (Henningson *et al.*, 1993)
12. Numerical simulation of spatial disturbance development in rotating channel flow (Lundbladh & Henningson, 1993)

13. Nonlinear energy density transfer during oblique transition in plane Poiseuille flow (Schmid & Henningson, 1993)
14. A new mechanism for rapid transition involving a pair of oblique waves (Schmid & Henningson, 1992)
15. Direct simulation of turbulent spots in plane Couette flow (Lundbladh & Johansson, 1991)
16. On the evolution of localized disturbances in laminar shear flows (Henningson *et al.*, 1990)
17. Subcritical transition in plane Poiseuille flow (Lu & Henningson, 1990)

1.3.2 Boundary layer flow studies

1. Linear feedback control and estimation applied to instabilities in spatially developing boundary layers (Chevalier *et al.*, 2007)
2. Turbulent spots in the asymptotic suction boundary layer (Levin & Henningson, 2007)
3. Large-eddy simulation of bypass transition (Schlatter *et al.*, 2006)
4. Early turbulent evolution of the Blasius wall jet (Levin *et al.*, 2006)
5. Periodic excitation of a turbulent separation bubble (Herbst & Henningson, 2006)
6. Transition thresholds in the asymptotic suction boundary layer (Levin *et al.*, 2005b)
7. A study of the Blasius wall jet (Levin *et al.*, 2005a)
8. Transition in boundary layers subject to free-stream turbulence (Brandt *et al.*, 2004)
9. Linear compensator control of a pointsource induced perturbation in a Falkner–Skan–Cooke boundary layer (Högberg *et al.*, 2003)
10. On the convectively unstable nature of optimal streaks in boundary layers (Brandt *et al.*, 2003)
11. Varicose instabilities in turbulent boundary layers (Skote *et al.*, 2002)
12. Transition of streamwise streaks in zero-pressure-gradient boundary layers (Brandt & Henningson, 2002)
13. Direct numerical simulation of a separated turbulent boundary layer (Skote & Henningson, 2002)
14. Linear optimal control applied to instabilities in spatially developing boundary layers (Högberg & Henningson, 2002)
15. Linear and nonlinear optimal control in spatial boundary layers (Chevalier *et al.*, 2002)
16. Optimal control of wall bounded flows (Högberg *et al.*, 2001)
17. A study of boundary layer receptivity to disturbances in the free stream (Berlin & Henningson, 1999)
18. Numerical and experimental investigation of oblique boundary layer transition (Berlin *et al.*, 1999)
19. The fringe region technique and the Fourier method used in the direct numerical simulation of spatially evolving viscous flows (Nordström *et al.*, 1999)
20. Secondary instability of cross-flow vortices in Falkner–Skan–Cooke boundary layers (Högberg & Henningson, 1998)
21. Linear and nonlinear development of localized disturbances in zero and adverse pressure gradient boundary-layers (Bech *et al.*, 1998)
22. Control of oblique transition by flow oscillations (Berlin *et al.*, 1998)

23. Direct numerical simulation of self-similar turbulent boundary layers in adverse pressure gradients (Skote *et al.*, 1998)
24. Simulations of laminar and transitional separation bubbles (Hildings, 1997)
25. Transition thresholds in boundary layer and channel flow (Schmid *et al.*, 1996)
26. Evaluation of Turbulence Models from Direct Numerical Simulations of Turbulent Boundary Layers (Lundbladh & Henningson, 1995)
27. Simulation of bypass transition in spatially evolving flows (Lundbladh *et al.*, 1994b)
28. Transition in Falkner–Skan–Cooke flow (Henningson & Lundbladh, 1994)
29. Spatial simulations of oblique transition (Berlin *et al.*, 1994)
30. A mechanism for bypass transition from localized disturbances in wall-bounded shear flows (Henningson *et al.*, 1993)
31. Simulation of the breakdown of localized disturbances in boundary layers (Lundbladh *et al.*, 1992b)

2 Installation

2.1 Prerequisites

2.1.1 Requirements

To compile and install **Simson** the following tools are required:

- A **Unix-like platform** (*e.g.* Linux, OSF1, Irix, AIX and Tru64).
- A **Fortran 90 compiler** (*e.g.* Intel ifort 10, Lahey f95 and PGI).
- **sh** (Bourne Shell)
- **GNU make** 3.79.1 or later
  (freely available at http://www.gnu.org/software/make/make.html)

2.1.2 Optional requirements

The following tools are optional:

- An **MPI implementation** (*e.g.* LAM, MPICH). Required to take advantage of the distributed-memory parallelization.
- An **OpenMP compiler**. Required to take advantage of shared-memory multiprocessor computers.
- **Matlab** for various scripts, plotting and postprocessing.
- **X-Windows** to use some of the postprocessing tools.
- **OpenDX** data format is supported for three-dimensional visualization.
- **EnSight Gold** data format is supported (*e.g.* **EnSight** and **ParaView**).
- **HDF4** data format is supported (*e.g.* **Matlab**).
- **Subversion** is used as the version control system for the development of **Simson**.

2.2 Directory structure

The **Simson** source code is usually supplied as a compressed tar file. This file should be uncompressed and unpacked in a chosen installation directory, for instance using the following Unix command:

```
gunzip simson-v4.0.0.tar.gz | tar xf -
```

After unpacking the **Simson** tarfile, the files and directories given in table 2.1 and 2.2 should have appeared.

File	Contents
config.mk	Parameter file read by all Makefiles
configure	Script that configures the system and stores it in **config.mk**
COPYRIGHT	Copyright and contact information document
par.f	Compile time parameter file regarding resolution *etc.*
README	How to configure, compile and install **Simson**
rules.mk	Generic Makefile build rules read by all Makefiles
todo.txt	A text file including things to correct/add/delete *etc.*

Table 2.1: Root directory files.

Directory	Contents
bla	Main program
bls	Program to generate initial velocity fields
cmp	Program to subtract and compare velocity fields
common	Common subroutines not shared by **bla**
config	Configure files
doc	Documentation
examples	Complete examples for different flow configurations
fou	Program to Fourier transform velocity fields in time
lambda2	Three-dimensional visualisation with OpenDX
matlab	Matlab scripts
pamp	Programs to plot amplitude data from amplitude files
pext1	Program to plot components from an extremum file
pxyst	Program to plot xy-statistics
rit	Program to plot solutions from complete velocity fields
rps	Program to plot planes
xys_add	Program to add xy-statistics

Table 2.2: Root directory contents.

2.3 Building **Simson**

The installation procedure given here is also found in the **README** file in the root directory. Building **Simson** requires three steps: configuring, compiling and installing.

2.3.1 Configuring

The main **Makefile** and the **Makefile.config** in the subdirectories should not be edited. They all fetch information from the **config.mk** located in the root directory. The configure script **configure**, located in the root directory, updates the information in the **config.mk** file based on analysis of the system and on information from a configuration file. Note that there exists a default **config.mk** file which can be directly edited which means that, for experienced users, it is not necessary to run the **configure** script. In each subdirectory there is also a **Makefile** that only uses locally defined parameters which could be useful, for example, when testing different compiler options.

The **configure** script requires Bourne shell to run. If Bourne shell is not in the default location (**/bin/sh**) the script will fail. In this case you should run

```
sh configure
```

instead of

```
./configure
```

The following sections describe the command-line options that can be given to **configure**.

2.3.1.1 `--help`

Type

```
./configure --help
```

for a complete listing of available options.

2.3.1.2 `--config`

The `--config` option determines from which file in the **config** directory the compiler and Makefile environment for your build should be fetched; *e.g.* to use the settings in **config/config.i686**, type

```
./configure --config=i686
```

If you give no `--config` option the default value is taken from the environment variable MACHINE. If MACHINE is not defined or empty the output of `uname -m` is used.

If there is no suitable file in **config** for your platform you may need to create a new **config** file. Use one of the existing files in **config** as template and save the file (*e.g.* as **config/config.custom**). Then configure with `--config=custom`. See the comments inside the **config** files for more details.

2.3.1.3 `--prefix`

The configure script also determines where **Simson** will be installed. This can be changed through the `--prefix` option; *e.g.* to put the **Simson** binaries in **$HOME/bin** type

```
./configure --prefix=\$HOME --config=i686
```

If you give no `--prefix` option the default path is the build directory itself where a directory based on the environment variable MACHINE will be created. If MACHINE is empty the result from `uname -m` is used instead.

2.3.1.4 `--program-suffix`

This option adds a suffix to all executable files. This may be convenient when installing several versions of **Simson** to the same location; *e.g.* to install a debug version with program suffix `_debug` type:

```
./configure --config=i686_debug --program-suffix=_debug
```

where the config file **config.i686_debug** contains the additional compiler flags required.

2.3.2 Compiling

Before starting to compile the code the resolution and some other compile-time parameters, specific for each flow case, have to be chosen. These parameters are all stored in the **par.f** file and examples of it reside in the root and **examples** directory. The root directory version of **par.f** is automatically distributed when typing `make all`. A detailed description of each parameter in **par.f** is found in section 7.1.

Note that the programs that depend on the resolution of the problem require a local copy of **par.f** and need to be recompiled for each change in **par.f**. The **par.f** in the root directory is easily distributed to these directories by writing

```
make dist
```

Now the **Simson** package can be compiled simply by writing

```
make
```

from the root directory. To ensure that all object files are compiled with the same flags and the same **par.f** file type

```
make all
```

which corresponds to

```
make clean
make dist
make
```

To compile the most commonly used programs (**bla**, **bls**, **rit**, **pxyst**) type

```
make allred
```

If you only want to build a single directory you can run make in that directory

```
cd rit
make -f Makefile.config
```

or

```
make rit.all
```

in the root directory. Note that in each subdirectory two makefiles exist, **Makefile** and **Makefile.config** which both can be used to compile the code in that subdirectory. The difference is that **Makefile.config** fetches compiling options from the **config.mk** file in the root directory whereas **Makefile** has local definitions of all options.

To only build a single Fortran file **filename.f** write

```
make filename.o
```

To remove all built object files and executable files, do

```
make clean
```

2.3.3 Compiling for parallel runs

After the appropriate changes in **par.f** (*i.e.* setting the variables **nproc** and/or **nthread** to the desired values), the flow solver **bla** can be compiled using

```
make mpi=yes
```

for MPI parallelization and using

```
make omp=yes
```

to activate the OpenMP directives. An experimental combination of both MPI and OpenMP is implemented (however requiring MPI2). The output from **bla** indicates whether OpenMP/MPI was used to compile. Note that only **bla** allows parallel runs. Therefore it is important to change the **nproc** and/or **nthread** parameters only in the **bla/par.f** file.

2.3.4 Installing

If the compilation finished without errors **Simson** can be installed with the command

```
make install
```

The binaries will be installed into the directory specified with the `--prefix` option to the **configure** script. To only install the contents of a subdirectory you may run

```
make install
```

in that directory. You can also compile and install a specific directory by writing

```
make bla.all
make bla.install
```

from the root directory. Alternatively, the executable files (*e.g.* **bla**, **bls** *etc.*) can be copied directly to another directory and run.

There is an **examples** directory located which contains parameter files for many of the different base flows and features that are available in the **Simson**. More information about the different cases can be found in appendix A.

3 Operation

The program **bla** reads a velocity field and necessary input files, steps the solution to a selected final time while producing log information on standard output and writes the final velocities to file. During the simulation **bla** may also output various velocity, pressure and scalar statistics, a file of the amplitude of specific wavenumbers, a file of extremum amplitudes, files with velocities in two dimensional planes at regular intervals in time and files containing complete 3D velocity fields at selected times. The simulation can be run with the pressure solver switched on to generate pressure fields corresponding to each velocity field that is written to file.

The program **bls** may be used to produce initial velocity fields including different types of disturbances.

The program **rit** performs postprocessing of 3D velocity fields into Tektronix or Postscript compatible graphics. Linear combinations (for example difference) of one or more 3D velocity fields can be computed with **cmp**, which can also calculate rms and maximum norm amplitudes of the result. This is useful when doing, for example, convergence checks. A set of complete velocity fields can also be analyzed through **fou** where they can be Fourier transformed in time. Three-dimensional visualization including the computation of the λ_2 vortex-identification criterion (Jeong & Hussain, 1995) is performed by **lambda2**.

Postprocessing of two dimensional planes is done by the program **rps** in a way similar to **rit**. Plots of amplitude files are generated by the programs **pamp1** and **pamp2**, which handle one and multiple amplitude files respectively. Wave amplitude files are plotted by the program **pampw** and **pampw2** and extremum amplitude files by **pext1**.

Statistics from simulations can be analyzed through **pxyst**, and to add statistical data sets of different runs **xys_add** can be used.

These programs along with the Fourier transform library **cvecfft_acc** and the plot library **plot1** form a completely self contained and portable system written in Fortran 77/90. Note that most of the main routines are written in Fortran 77 but that some Fortran 90 features have been used to ease the readability and flexibility of the code.

A set of example cases, for various base flows, is included in the **examples** directory. The examples are briefly explained in appendix A.

3.1 Preprocessing

3.1.1 Generating initial velocity fields with **fsc** and **bls**

An initial velocity field consists of a header and an array with the three components of velocity in Fourier space fulfilling the equation of continuity. The format of the file is described in section 7.6. The program **bls** may be used to generate an initial velocity field, consisting of a basic laminar flow and a range of different disturbances, for example, localized disturbance, waves and random noise. Different disturbances

can also be combined. The program **bls** can also generate passive scalar fields if needed.

The initial velocity field file has the same format as files generated by subsequent execution of the **bla** program so that it is possible to feed the initial velocity field to the postprocessing tools directly for examination.

For some flow types the laminar base flow profile is not given analytically (*e.g.* Blasius profile). For these cases, a velocity profile file must first be generated. The program **fsc** computes velocity profiles from the Blasius/Falkner–Skan/Falkner–Skan–Cooke family for both velocity and scalars and it requires an input parameter file **fsc.i**. The output file **fsc.dat** is needed by **bls** and **bla** and is described in section 7.17. It contains similarity boundary layer profiles computed from the laminar boundary layer flow equations for flow over a flat plate or a wedge. The program **bls** generates a temporal/spatial or parallel/non-parallel velocity field depending on the flow type parameter **fltype**. It is intended for batch execution and has no interactive input. The input comes from the file **bls.i**. The format of this file is given in section 7.3.

3.1.2 Generating non-similarity base flows

In boundary-layer cases where the streamwise free-stream velocity is not a power of the downstream distance, the boundary layer equations do not have a self similar solution. To generate a base flow for this situation **bls** is first used to generate a similarity flow field (without disturbances) which is a good approximation to the sought flow around the inflow boundary, *e.g.* a flow such that the boundary layer thickness and the acceleration are correct around the inflow boundary. Then this flow field can be advanced in time with **bla** to find a steady state using a streamwise free-stream velocity given in tabular form as a function of the downstream distance (see sections 7.4 and 7.15). The generated steady flow field can be input to **bls** and disturbances superimposed. The same flow field can be used to specify the baseflow to **bla** for subsequent simulations.

3.2 Running **bla**

The program is intended to be used in batch mode and so has no interactive input. The main configuration is done at compile time through changes in the file **par.f** (see section 7.1) and at runtime in **bla.i** (see section 7.4). Depending on choices made in **bla.i** other input files might be needed. An initial velocity field, which can be produced by the program **bls**, is always needed to start execution.

By default **bla** generates a number of output files depending on the base flow type and choices made in the file **bla.i**. A list of the most common input and output files can be found in table 3.1.

When the simulation has finished information about how much time that has been spent in different subroutines and in total can be found in the **step.out** file. During the simulation it contains information about how much time each time step takes.

A simulation can always be stopped by creating a file **stop.now** in the running directory. This will cause the time integration to stop at the end of the current time step. An output file and the statistics (if active) are written to file.

3.2.1 Running in serial

For a simple simulation, without any particular forcing added, only an initial velocity field and a runtime parameter file **bla.i** is needed. The generation of initial velocity fields is described in section 3.1 and how to construct a suitable **bla.i** is described in section 7.4. For a serial run, the **par.f** variables **nproc** and **nthread** need to be set to 1.

After compiling (see section 2.3.2) the simulation is started by typing

File	I/O	Contents
bla.i	I	Input parameter file for **bla**
bls.i	I	Input parameter file for **bls**
fsc.dat	I	Similarity solutions f, f', f'', g and g' ($+$ θ, θ' and θ'')
fsc.i	I	Input parameter file for **fsc**
history.out	O	Time history data
nodes.out	O	Information about how many nodes that were used
sgs.i	I	Input parameter file for LES mode
step.out	O	Timing information
.amp	O	Amplitude data for different wave number pairs
.p	O	Pressure fields
.stat	O	Statistics
.u	I/O	Velocity fields ($+$ θ)

Table 3.1: Files generated or required by **bla**, **bls** and **fsc**. This information is given in column two where *I* stands for input and *O* for output. Note that the scalar fields θ are included only if the option passive scalar is active.

```
./bla
```

where runtime information will be written to standard output.

3.2.2 Running in parallel with OpenMP

OpenMP is a parallelization strategy that works on shared memory machines. To run the code in OpenMP mode the code needs to be compiled for the maximum number of threads that are to be used (variable **nthread**, see section 2.3.3). It is also necessary to set the environmental variable OMP_NUM_THREADS to the number of threads in order to allocate them. Note that one can always use less threads than compiled for via the variable **nthread**.

A simulation that is compiled for four threads is thus started (on a system using **bash**) by typing

```
export OMP_NUM_THREADS=4
./bla
```

3.2.3 Running in parallel with MPI

MPI, on the other hand, is a parallelization method that works on distributed memory machines. To run the code in MPI mode the code needs to be compiled for the number of processors that are to be used (variable **nproc**, see section 2.3.3). How to start MPI simulations can vary depending on, for example, architecture, operating system and queuing system. On a Linux cluster, using a standard MPI implementation one can start a 16 processor simulation with

```
mpirun -np 16 ./bla
```

The **-np** flag value must match the **nproc** parameter in the **par.f** file.

3.2.4 Memory requirements

The memory requirement depends on the resolution of the simulation, whether additional features are included (pressure solver, passive scalars *etc.*), whether dealiasing in the y-direction is used and whether MPI parallelization is used.

The three dimensional storage is **(7+pressure+3×scalar)×nx×ny×nz** double precision numbers (*i.e.* 8 bytes), multiply by a factor of 3/2 for dealiasing in the y-direction and by 1/2 if z-symmetry is used. Most of the memory-consuming two-dimensional working arrays are allocated in the main program (*i.e.* **bla.f**) and passed to the respective subroutines. Some features can be turned off manually in **bla.f** at compile time to save memory (*e.g.* **iiles**, **iisfd**).

3.2.5 Performance tuning

The code itself has been written for maximum speed on vectorizing computers using a highly optimizing compiler.

The basic vector length is **nxp/2×(mby−1)+nx/2** in most of step 2 (where **nxp** is equal to **nx** without x-dealiasing and **nx×3/2** with x-dealiasing, and **nz×mby** in the x-transform, multiply the latter by 3/2 for z-dealiasing. The vector length in step 3 is **nx/2×mbz**. If these values are lower than what is needed to get a good performance, **mby** and **mbz** can be increased. Note however that in the present implementation **mby** and **mbz** are set to 1, which does not pose a serious problem on (super-)scalar architectures. It is however likely that support for general **mby** and **mbz** is included in later releases.

Both the OpenMP and MPI parallelization exploit coarse-grain parallelization. Step 2 and step 3 can each be divided on as many processors as there are boxes to process; typically this is no limitation. The code has been run in parallel mode on the Alliant FX-80 and FX-2800, the SGI Powerstation, Challenge and Power Challenge, the CRAY-2, J90, DEC alpha, Cray C90, IBM SP-3/4, IBM BlueGene, SGI Origin/Altix, and various Linux clusters. The typical speed-up is 3.5 – 3.8 for four processors using OpenMP. With MPI the parallel performance is heavily dependent on the speed of the interconnect.

3.3 Postprocessing

3.3.1 Postprocessing velocity files with **rit**

The program **rit** generates various graphs from a velocity field. The graphs can be generated in either Tektronix 4014 format or Postscript. When executed, **rit** prompts for an input file name. The file is read and the program offers a choice of various types of graphs. It is mainly intended for interactive execution and should be self explanatory.

It is possible to use **rit** in a batch environment by compiling it into an input program. This is run interactively to produce a file **ritin**, which is subsequently read by the batch code to produce the desired plots. Note that if plots in batch mode are produced to the screen the resulting Tektronix graphic characters will be written to the log file. To compile a batch program, set **imode** to 2 in the **rit.f** file and compile a second time with **imode** = 3 to get an input program. To get an interactive program **imode** should be left at 1.

3.3.2 Postprocessing velocity files with **cmp**

The program **cmp** is used for constructing linear combinations of velocity fields which is often used when comparing data. The program can also raise a velocity field to a given power.

3.3.3 Postprocessing plane files with **rps**

Planes saved during a simulation can be examined with the program **rps**.

3.3.4 Postprocessing velocity files with **fou**

When a number of velocity fields has been saved during a simulation, the program **fou** can be used to make Fourier transforms in both space and time.

3.3.5 Postprocessing amplitude files with **pamp1**, **pamp2**, **pampw**, **pampw2** and **pext1**

The programs **pamp1** and **pamp2** can be used to produce plots of the time history of various amplitudes from the amplitude files written by **bla**. The program **pamp1** works on one file and **pamp2** can plot one quantity from multiple files. The program **pext1** makes plots of time histories of extremum values (*i.e.* minimum and maximum values) of velocities and vorticities and the location of extrema. The programs **pampw** and **pampw2** similarly plot amplitudes of wave components (streamwise–spanwise Fourier mode) from one or multiple wave-amplitude files. The programs are intended to be self explanatory and prompt for input file names. Since the amplitude files are formatted and normally relatively small, no batch versions of these programs are available. The files contain no headers so that files from sequential runs of one flow case can be concatenated and then plotted to show the complete evolution of the amplitudes.

3.3.6 Postprocessing xy-statistics files with **pxyst**

To get good statistics of space developing flows with one homogeneous direction (spanwise), the data needs to be averaged in time. The plotting of time and spanwise averaged data saved to file is performed by **pxyst**. Note that these files have headers, and thus if statistical data need to be concatenated it has to be done with the program **xys_add**. The format of the statistics files is given in section 7.12.

The program **pxyst** generates plots both of the raw statistical data and of a number of derived quantities. It is also possible to generate various special plots of the mean flow, such as boundary layer thicknesses and skin friction.

There is an initial option to filter data, which applies to the raw data, before computing other quantities. There is also an option to filter data prior to producing plots, the filter is then applied to the derived quantity. The results of the two filtering processes may differ. In both cases the filter is applied in the streamwise direction.

4 Theory

4.1 Derivation of the velocity–vorticity formulation

The starting point for the theoretical aspects and derivations are the non-dimensionalized incompressible Navier–Stokes equations in a rotating reference frame, here written in tensor notation,

$$\frac{\partial u_i}{\partial t} = -\frac{\partial p}{\partial x_i} + \epsilon_{ijk} u_j (\omega_k + 2\Omega_k) - \frac{\partial}{\partial x_i}\left(\frac{1}{2} u_j u_j\right) + \frac{1}{Re}\nabla^2 u_i + F_i \,, \tag{4.1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \,, \tag{4.2}$$

with different initial and boundary conditions depending on the flow geometry.

The first equation represents conservation of momentum and the second equation represents the incompressibility of the fluid. Here $(x_1, x_2, x_3) = (x, y, z)$ denotes the streamwise, normal and spanwise coordinates, $(u_1, u_2, u_3) = (u, v, w)$ the respective velocities, $(\omega_1, \omega_2, \omega_3) = (\chi, \omega, \vartheta)$ the corresponding vorticities, and p the pressure. The streamwise and spanwise directions will alternatively be termed horizontal directions. The angular velocity of the coordinate frame around axis k is denoted Ω_k. In practice the most often used case is rotation around the spanwise axis, thus let $\Omega = \Omega_3$ be the rotation number. The body force $\mathbf{F} = (F_1, F_2, F_3)$ is used for numerical purposes that will be further discussed in sections 4.2.1 and 4.2.2. It can also be used to introduce disturbances in the flow. The definition of the Reynolds number varies with the flow type. For boundary layer flows it is defined as

$$Re_{\mathrm{bl}} = \frac{U_\infty \delta^*}{\nu} \,, \tag{4.3}$$

where U_∞ is the undisturbed streamwise free-stream velocity at $x = 0$ and $t = 0$, δ^* is the displacement thickness of the undisturbed streamwise velocity at $x = 0$ and $t = 0$, and ν is the kinematic viscosity. For channel flow the Reynolds number is defined as

$$Re_{\mathrm{cl}} = \frac{U_{\mathrm{cl}} y_L}{2\nu} \,, \tag{4.4}$$

where U_{cl} is the (laminar) centerline velocity. For Couette flow the Reynolds number is defined as

$$Re_{\mathrm{co}} = \frac{U_{\mathrm{co}} y_L}{2\nu} \,, \tag{4.5}$$

where U_{co} is defined as half of the velocity difference between the walls.

The size of the solution domain in physical space is x_L, y_L and z_L in the streamwise, normal and spanwise directions respectively. For channel and Couette flow $y_L = 2h = 2$ with h being the channel half-width.

A Poisson equation for the pressure can be obtained by taking the divergence of the momentum equations,

$$\nabla^2 p = \frac{\partial H_i}{\partial x_i} - \nabla^2\left(\frac{1}{2} u_j u_j\right) \,, \tag{4.6}$$

where

$$H_i = \epsilon_{ijk} u_j (\omega_k + 2\Omega_k) + F_i \,. \tag{4.7}$$

Application of the Laplace operator to the momentum equations for the normal velocity yields an equation for that component through the use of equations (4.2) and (4.6). One finds

$$\frac{\partial \nabla^2 v}{\partial t} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2} \right) H_2 - \frac{\partial}{\partial y} \left(\frac{\partial H_1}{\partial x} + \frac{\partial H_3}{\partial z} \right) + \frac{1}{Re} \nabla^4 v \,. \tag{4.8}$$

This equation can, for numerical purposes, be written as a system of two second order equations

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= h_v + \frac{1}{Re} \nabla^2 \phi \,, \\ \nabla^2 v &= \phi \,, \end{aligned} \tag{4.9}$$

where

$$h_v = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2} \right) H_2 - \frac{\partial}{\partial y} \left(\frac{\partial H_1}{\partial x} + \frac{\partial H_3}{\partial z} \right) . \tag{4.10}$$

An equation for the normal vorticity can be found by taking the curl of the momentum equations. The second component of that equation reads

$$\frac{\partial \omega}{\partial t} = h_\omega + \frac{1}{Re} \nabla^2 \omega \,, \tag{4.11}$$

where

$$h_\omega = \frac{\partial H_1}{\partial z} - \frac{\partial H_3}{\partial x} \,. \tag{4.12}$$

Note that the equations for ϕ, v and ω have similar form, and can thus be solved using the same numerical routine. Once the normal velocity v and the normal vorticity ω have been calculated the other velocity components can be found from the incompressibility constraint and the definition of the normal vorticity.

4.2 Forcing

4.2.1 Temporal simulations

4.2.1.1 Channel flow

Laminar and turbulent flow between two plates with periodic boundary conditions in both wall-parallel directions is constantly loosing kinetic energy due to friction at the walls. Consequently, a forcing needs to be employed in order to keep the flow from decelerating. Two possibilities are implemented: Fixed mass-flow rate and fixed pressure gradient. In the latter case, a constant pressure gradient is imposed in the streamwise (x) direction to exactly balance the mean shear stress at the wall,

$$F_1 = -\frac{\mathrm{d}p_w}{\mathrm{d}x} = \tau_w = \frac{1}{Re_{\mathrm{cl}}} \left. \frac{\mathrm{d}U}{\mathrm{d}y} \right|_{\mathrm{wall}} \,. \tag{4.13}$$

Note that dimensional quantities are omitted and the non-dimensionalization is based on equation (4.4), *i.e.* the channel half-width and the laminar centerline velocity. The corresponding friction Reynolds number Re_τ based on the friction velocity u_τ is then given by

$$Re_\tau = Re_{\mathrm{cl}} \sqrt{-\frac{\mathrm{d}p_w}{\mathrm{d}x}} \,. \tag{4.14}$$

On the other hand, with a fixed mass-flow rate the above pressure gradient is adapted at every time step in order to keep the bulk velocity u_b constant,

$$\frac{1}{2h} \int_{-h}^{h} U(y) \mathrm{d}y = \frac{1}{2} \int_{-1}^{1} (1 - y^2) \mathrm{d}y = \frac{2}{3} = u_b \,. \tag{4.15}$$

The laminar centerline Reynolds number $Re_{\rm cl}$ and the bulk Reynolds number Re_b are thus related through $Re_{\rm cl} = (3/2)Re_b$. The following relations are useful:

$$\text{laminar channel flow} \qquad Re_\tau = \sqrt{2Re_{\rm cl}} = \sqrt{3Re_b}\,, \tag{4.16}$$

$$\text{turbulent channel flow} \qquad Re_\tau \approx 0.116 Re_{\rm cl}^{0.88} \approx 0.166 Re_b^{0.88}\,. \tag{4.17}$$

4.2.1.2 Boundary-layer flow

A localized disturbance or wave of relatively short wavelength which travels downstream in a slowly growing boundary layer is surrounded by a boundary layer of almost constant thickness which grows slowly in time. This forms the basis of the temporal simulation technique.

Following the ideas of Spalart & Yang (1987) we assume that the boundary layer streamwise velocity is $U(x,y)$ and introduce a reference point $x_r = x_0 + ct$ at which we evaluate the mean flow and where c is a reference speed. We now assume that the undisturbed boundary layer in the vicinity of the disturbance has the velocity distribution $\{U(y,t) = U(x_r,y), V(y,t) = 0, W(y)\}$. Since the boundary layer is now parallel (as there is no dependence on x), it is possible to apply periodic boundary conditions in the horizontal directions. However, whereas $U(x,y)$ (with the corresponding V given by continuity) is a solution to the Navier–Stokes equations or at least the boundary layer equations, this is not true for $\{U(y,t), V(y,t)\}$. Thus to ensure the correct development of the boundary layer profile over extended periods of time it is necessary to add a (weak since $1/Re$ when $c = 0$) forcing to balance the streamwise and spanwise momentum equations,

$$\begin{aligned} F_1 &= \frac{\partial U(y,t)}{\partial t} - \frac{1}{Re}\frac{\partial^2 U(y,t)}{\partial y^2} = c\frac{\partial U(x,y)}{\partial x} - \frac{1}{Re}\frac{\partial^2 U(x,y)}{\partial y^2}\,, \\ F_2 &= 0\,, \\ F_3 &= -\frac{1}{Re}\frac{W_\infty}{U_\infty}\frac{\partial^2 W(y)}{\partial y^2}\,, \end{aligned} \tag{4.18}$$

where the right hand side should be evaluated at the reference coordinate x_r and U_∞ and W_∞ represent the streamwise and spanwise freestream velocity components. The reference speed should be chosen as the group speed of the wave or the propagation speed of the localized disturbance for best agreement with a spatially developing flow. To fully justify the periodic boundary conditions in the case of a wave train, the wave itself should be slowly developing.

4.2.2 Spatial simulations

The best numerical model of a physical boundary layer, which is usually developing in the downstream direction rather than in time, is a spatial formulation. To retain periodic boundary conditions, which is necessary for the Fourier discretization, a fringe region is added downstream of the physical domain, similar to that described by Bertolotti *et al.* (1992). In the fringe region disturbances are damped and the flow returned to the desired inflow condition. This is accomplished by the addition of a volume force which only increases the execution time of the algorithm by a few percent.

The forcing has the form

$$\mathbf{F} = \lambda(x)(\boldsymbol{\mathcal{U}} - \mathbf{u})\,, \tag{4.19}$$

where $\lambda(x)$ is a non-negative fringe function which is significantly non-zero only within the fringe region. The flow field $\boldsymbol{\mathcal{U}} = \{\mathcal{U}, \mathcal{V}, \mathcal{W}\}$ is the same flow field used for the boundary conditions, which also contains the desired flow solution in the fringe. The streamwise velocity component is calculated as

$$\mathcal{U}(x,y) = U(x,y) + (U(x+x_L,y) - U(x,y))\, S\left(\frac{x - x_{\rm blend}}{\Delta_{\rm blend}}\right), \tag{4.20}$$

where $U(x, y)$ is normally a solution to the boundary layer equations. Here x_{blend} and Δ_{blend} are chosen so that the prescribed flow, within the fringe region, smoothly changes from the outflow velocity of the physical domain to the desired inflow velocity, *i.e.* a blending between inflow and outflow. The step function S is given in (4.22). The wall-normal component $\mathcal{V}$ is then calculated from the equation of continuity, and the spanwise velocity $\mathcal{W}$ is set to zero for simulations where the mean flow is two dimensional. For three dimensional boundary layers $\mathcal{W}$ is computed from a boundary layer solution in fashion analogous to that for $\mathcal{U}$. This choice of $\boldsymbol{\mathcal{U}}$ ensures that for the undisturbed laminar boundary layer the decrease in thickness is completely confined to the fringe region, thus minimizing the upstream influence. A forced disturbance to the laminar flow can be given as inflow condition if that disturbance is included in $\boldsymbol{\mathcal{U}}$.

A convenient form of the fringe function λ is as follows

$$\lambda(x) = \lambda_{\max}\left[S\left(\frac{x - x_{\text{start}}}{\Delta_{\text{rise}}}\right) - S\left(\frac{x - x_{\text{end}}}{\Delta_{\text{fall}}} + 1\right)\right]. \tag{4.21}$$

Here $\lambda_{\max}$ is the maximum strength of the damping, x_{start} to x_{end} the spatial extent of the region where the damping function is non-zero and Δ_{rise} and Δ_{fall} the rise and fall distance of the damping function. The smooth "step" function $S(x)$ rises from zero for negative x to one for $x \geq 1$. We have used the following form for S, which has the advantage of having continuous derivatives of all orders,

$$S(x) = \begin{cases} 0\,, & x \leq 0\,, \\ 1/\left(1 + e^{(1/(x-1)+1/x)}\right), & 0 < x < 1\,, \\ 1\,, & x \geq 1\,. \end{cases} \tag{4.22}$$

An example of a fringe function is depicted in figure 4.1.

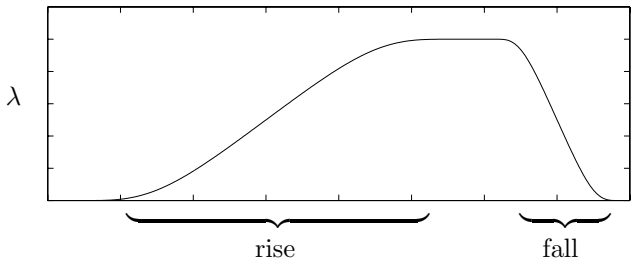


Figure 4.1: Schematic picture of the fringe region. For this fringe the sum of the rise and fall is the same as the total length.

To achieve maximum damping both the total length of the fringe and $\lambda_{\max}$ have to be tuned. The actual shape of $\lambda(x)$ is less important for the damping but it should have its maximum closer to x_{end} than to x_{start}. The damping is also strongly influenced by the resolution of the disturbance that should be damped.

In summary the main parameters determining the damping properties of the fringe forcing are

- Length of fringe (L)
- Strength of fringe (λ)
- Shape of fringe
- Resolution of simulation
- Influence of blending (for boundary layer flows)

An investigation on how the fringe parameters effect the disturbance in the fringe can be found in Lundbladh *et al.* (1999) and example cases can be found in the **examples** directory. Generally however one usually has to try a few different sets of

fringe parameters when setting up a new flow case and make sure that the critical disturbances are damped out satisfactory.

For maximum computational efficiency the simulated flow has to be considered when the fringe parameters are tuned. Assuming that the achieved damping is sufficient, a short fringe reduces the box length and therefore the required CPU time per iteration. However, if the flow gradients introduced in the fringe region are larger than those in the physical domain that may decrease the time step and consequently increase the necessary number of iterations. Note that the boundary layer growth causes outflow through the free-stream boundary. The streamwise periodicity requires that all that fluid reenters in the fringe region.

Analysis of the Navier–Stokes equations with a fringe forcing term yields an additional part of the disturbance associated with the pressure with a decay independent of λ. For a boundary layer, this solution decays appreciably over a downstream distance equal to the boundary layer thickness, and thus the fringe region must be some factor (say 10 to 30) times this thickness to get a large decay factor, see Nordström *et al.* (1999).

4.3 Boundary conditions

The boundary conditions in the horizontal directions are periodic which hold for all flows implemented. The wall-normal boundary conditions however may differ between the different flow types; additionally, inhomogeneous boundary conditions may be specified (*e.g.* wall blowing and suction).

4.3.1 Poiseuille flow

The natural no-slip boundary conditions on both walls, located at $y = \pm 1$, read

$$v|_{\text{wall}} = 0\,, \quad \left.\frac{\partial v}{\partial y}\right|_{\text{wall}} = 0\,, \quad \omega|_{\text{wall}} = 0\,. \tag{4.23}$$

For disturbance generation and control by blowing and suction through the wall, an arbitrary time dependent velocity distribution,

$$\begin{aligned} v|_{y=1} &= v_{\text{BS}_u}(x,z,t)\,, \\ v|_{y=-1} &= v_{\text{BS}_l}(x,z,t)\,, \end{aligned} \tag{4.24}$$

can be used. Several types of control signals are available.

4.3.2 Couette flow

The boundary conditions for Couette flow impose constant streamwise velocity, positive on the upper wall and negative on the lower wall, and zero wall-normal and spanwise velocity components.

4.3.3 Boundary layer flow

The boundary conditions in the horizontal directions are periodic but one needs to specify boundary conditions at the wall and in the free-stream, to solve equations (4.9) and (4.11).

For disturbance generation and control by blowing and suction through the wall, an arbitrary time dependent velocity distribution,

$$v|_{y=0} = v_{\text{BS}}(x,z,t)\,, \tag{4.25}$$

can be used.

The flow is assumed to extend to an infinite distance perpendicular to the wall. However, the discretization discussed below can only handle a finite domain. Therefore, the flow domain is truncated and an artificial boundary condition is applied in the free-stream at a wall-normal position y_L.

The Dirichlet boundary condition, defined as

$$\mathbf{u}_{y=y_L} = \boldsymbol{\mathcal{U}}_{y=y_L}\,, \tag{4.26}$$

is the simplest one, where $\boldsymbol{\mathcal{U}}(x,y)$ is a base flow that is normally chosen as a Falkner–Skan–Cooke (FSC) flow. An arbitrary pressure gradient, to for instance create a separation bubble, can be imposed by choosing $\boldsymbol{\mathcal{U}}$ accordingly.

The desired flow solution generally contains a disturbance and that will be forced to zero by the Dirichlet condition. This introduces an error compared to the exact solution for which the boundary condition is applied at an infinite distance from the wall. The error may result in increased damping for disturbances in the boundary layer.

Some improvement can be achieved by using a Neumann condition,

$$\left.\frac{\partial \mathbf{u}}{\partial y}\right|_{y=y_L} = \left.\frac{\partial \boldsymbol{\mathcal{U}}}{\partial y}\right|_{y=y_L}\,. \tag{4.27}$$

This condition can be shown to be stable if there is outflow at the boundary or the inflow is weaker than $O(1/Re)$. This restriction is usually fulfilled if the boundary is placed on a sufficiently large distance from the wall, so that the disturbance velocity is small.

A generalization of the boundary condition used by Malik *et al.* (1985) allows the boundary to be placed closer to the wall. It is an asymptotic condition that decreases the error further and it reads

$$\left.\left(\frac{\partial \hat{\mathbf{u}}}{\partial y} + |k|\hat{\mathbf{u}}\right)\right|_{y=y_L} = \left.\left(\frac{\partial \hat{\boldsymbol{\mathcal{U}}}}{\partial y} + |k|\hat{\boldsymbol{\mathcal{U}}}\right)\right|_{y=y_L}\,, \tag{4.28}$$

where $\hat{(\cdot)}$ denotes the horizontal Fourier transform with respect to the horizontal coordinates, $k^2 = \alpha^2 + \beta^2$ and α and β are the horizontal wavenumbers (see equation (5.10)). Thus this condition is most easily applied in Fourier space. The boundary condition exactly represents a potential flow solution decaying away from the wall. It is essentially equivalent to requiring that the vorticity is zero at the boundary. Thus, it can be applied immediately outside the vortical part of the flow.

4.3.4 Asymptotic suction boundary layer flow

The asymptotic suction boundary layer (ASBL) is an analytical solution to the Navier–Stokes equations when uniform wall-normal suction, with velocity $-v_s$, is applied at the wall. It can be written as

$$\mathbf{U} = (1 - e^{-y}, -v_s, 0)\,. \tag{4.29}$$

The analytical solution allows the displacement thickness to be calculated exactly, $\delta^* = \nu/v_s^*$ and the Reynolds number to be expressed as the velocity ratio, $Re_{\text{asbl}} = U_\infty/v_s^*$, where $-v_s^*$ is the dimensional suction velocity.

The constant suction requirement at the wall is removed by subtracting the Navier–Stokes equations by v_s which gives the following modified form of equation 4.7

$$H_1 = (v - v_s)\vartheta - w\omega\,, \quad H_3 = u\omega - (v - v_s)\chi\,, \tag{4.30}$$

where $(\chi, \omega, \vartheta)$ denotes the vorticity vector. Note that no additional forcing is needed to keep the base flow parallel. More details about the ASBL and its implementation in **Simson** can be found in Levin *et al.* (2005b).

4.3.5 Surface roughness

The spectral framework of **bla** requires uniform boundaries. Thus, roughness elements on the surface of the plate cannot be included in terms of grid geometry, but they need to be modeled by modified boundary conditions. A projection method is employed here, as originally proposed in Choudhari & Streett (1992) and Crouch (1992). A Taylor expansion of fourth order is used to project the no-slip conditions along the desired bump contour onto the lowest grid plane $y = 0$, where y is the wall-normal coordinate,

$$v_i|_{y=0} \approx \underbrace{v_i(h)}_{\equiv 0} - h\frac{\partial v_i}{\partial y}\bigg|_{y=0} - \quad ... \quad - \frac{h^4}{4!}\frac{\partial^4 v_i}{\partial y^4}\bigg|_{y=0} \quad , \quad i = 1,2,3 \tag{4.31}$$

where $h = h(x,z)$ denotes the contour function of the bump and v_i can be either the base-flow profile or the instantaneous velocity. In the present implementation the bump function h is localized in the chordwise direction x (step function, equation (4.22)) and periodic along the span z.

The wall-normal velocity derivatives in equation (4.31) are obtained from the base-flow profiles. The representation of the no-slip conditions along the desired roughness contour can be improved by updating the bump conditions regularly. Then, the velocity derivatives of equation (4.31) are taken from the current flow field.

4.3.6 Jet in crossflow

To model a jet discharging normal to a crossflow either a top-hat jet profile or a parabolic jet profile can be imposed. The former boundary condition models fluid ejected from a turbulent pipe, whereas the latter models a laminar pipe flow. The main parameters (set in **bla.f**) are the position of the jet orifice $(x_{\rm jet}, z_{\rm jet})$, the jet diameter $D_{\rm jet}$ and the ratio

$$R_{\rm jet} = \frac{\bar{v}_j}{U_\infty} \tag{4.32}$$

of the bulk velocity $\bar{v}_j$ from the jet orifice to the cross-flow velocity.

The first boundary condition is a top-hat profile given by,

$$v_j(r) = \frac{1}{2}[1 - \tanh(5(r - 1/r))]\,, \tag{4.33}$$

where $r = 2\sqrt{x^2 + z^2}/(\alpha D_{\rm jet})$. In this case, the diameter of the jet is defined by the bulk velocity, by choosing $\alpha = 0.9919$ such that the bulk velocity equals the maximum velocity, $\bar{v}_j = v_{j,\max}$. The second boundary condition is a parabolic profile multiplied with a Gaussian function,

$$v_j(r) = (1 - r^2)e^{(-(r/0.7)^4)}\,. \tag{4.34}$$

For this boundary condition the relation between the bulk and the maximum velocity is $\bar{v}_j \approx v_{j,\max}/3$.

Furthermore, for the actual implementation in the numerical code, the net mass flux is set to zero via an artificial uniform suction applied at the lower wall, *i.e.* the $((0,0)$-mode of the v-component. This uniform suction velocity can however be subtracted from the velocity fields and statistics.

4.4 Initial conditions

To start a temporal or spatial simulation a laminar initial flow field is required. For boundary layer flows it is constructed from similarity solutions of the boundary layer equations and for Poiseuille and Couette flows it can be given analytically.

In the boundary layer flow favorable and adverse pressure gradients can be accounted for as well as the effect of a sweep. To obtain the family of FSC similarity solutions it

is assumed that the chordwise outer-streamline velocity obeys the power law $U^*_\infty = U^*_0(x^*/x^*_0)^m$ and that the spanwise velocity W^*_∞ is constant. Here U^*_0 is the free-stream velocity at the beginning of the computational box and the asterisks $(*)$ denote dimensional quantities. Note that the Blasius profile is a special case of FSC with zero cross flow component and pressure gradient. If the similarity variable η is chosen as

$$\eta(y^*) = y^* \sqrt{\frac{m+1}{2}\frac{U^*_\infty}{2\nu x^*}}$$

one can derive the following self-similar boundary layer profiles,

$$f''' + ff'' + \beta_h(1 - f'^2) = 0\,,$$
$$g'' + fg' = 0\,,$$

where the Hartree parameter β_h relates to the power law exponent m as $\beta_h = 2m/(m+1)$. The accompanying boundary conditions are

$$f = f' = g = 0 \quad \text{for} \quad \eta = 0\,,$$
$$f' \to 1\,, \quad g \to 1 \quad \text{as} \quad \eta \to \infty\,.$$

The complete derivation can be found in *e.g.* Schlichting (1979) and Cooke (1950). From the FSC similarity solutions, one constructs the nondimensional velocity profiles

$$U(y) = f'(\eta(y))\,, \tag{4.35a}$$
$$W(y) = \frac{W_\infty}{U_\infty} g(\eta(y))\,, \tag{4.35b}$$

for a fixed x and where $y = y^*/\delta^*$. The velocity profiles (4.35a) and (4.35b) are then used when constructing the base flow.

The initial similarity solution for the passive scalar is found from solving

$$\theta'' = -Pr\, f\, \theta' + Pr\, m_1(2 - \beta_h) f' \theta\,,$$

subject to the boundary conditions

$$\theta = 1 \quad \text{for} \quad \eta = 0\,,\ \theta = 0 \quad \text{as} \quad \eta \to \infty\,.$$

Here, Pr is the Prandtl (or Schmidt) number, and m_1 is the power-law exponent for the scalar distribution along the streamwise direction. Note that $m_1 = 0$ defines a constant scalar value at the wall (*i.e.* $\theta = 1$), whereas $m_1 = 1/2$ leads to a similarity solution with constant wall-normal derivative (*i.e.* isoflux boundary condition).

Type	Description
−2	Temporal Falkner–Skan–Cooke BL
−1	Temporal Falkner–Skan BL
0	No base flow (currently not in use)
1	Temporal Poiseuille
2	Temporal Couette
3	Temporal Blasius BL
4	Spatial Poiseuille
5	Spatial Couette
6	Spatial Blasius BL
7	Spatial Falkner–Skan BL
8	Spatial Falkner–Skan–Cooke BL
9	Parallel Blasius/Falkner–Skan/Falkner–Skan–Cooke BL with fringe

Table 4.1: Available base flow types.

4.5 Disturbance formulation and linearized solver

Introducing a decomposition of the flow variables into a base flow and a disturbance part, $u_i = U_i + u_i'$ leads to an evolution equation for u_i' due to equation (4.1)

$$\begin{aligned}\frac{\partial u_i'}{\partial t} = &- \frac{\partial p'}{\partial x_i} + \epsilon_{ijk}(u_j'\omega_k' + U_j\omega_k' + u_j'W_k) \\ &- \frac{\partial}{\partial x_i}\left(\frac{1}{2}u_j'u_j' + U_ju_j'\right) + \frac{1}{Re}\nabla^2 u_i' + F_i - G_i\,.\end{aligned} \tag{4.36}$$

Here, the rotation rate Ω_k is neglected; W_k denotes the vorticity field corresponding to the base flow U_i. The base flow is assumed to satisfy the stationary Navier–Stokes equations,

$$0 = -\frac{\partial P}{\partial x_i} + \epsilon_{ijk}U_jW_k - \frac{\partial}{\partial x_i}\left(\frac{1}{2}U_jU_j\right) + \frac{1}{Re}\nabla^2 U_i + G_i\,. \tag{4.37}$$

Additionally, both U_i and u_i' are supposed to satisfy the continuity constraint. G_i can be considered the residual of the base flow when put into the Navier–Stokes equations (4.1), and it will include the fringe force $\lambda(\mathcal{U}_i - U_i)$ for spatial simulations. It also includes additional (unknown) contributions if the base flow does not satisfy the Navier–Stokes equations, *e.g.* when using the boundary-layer equations to compute U_i. For simulations using the disturbance formulation it is therefore suggested to always use a converged solution of the Navier–Stokes equations (*i.e.* the converged steady result of a simulation) as a base flow.

For spatial simulations, the terms F_i and G_i are given by the fringe forcing (4.19). Then, $G_i = \lambda(\mathcal{U}_i - U_i)$ and $F_i = \lambda(\mathcal{U}_i - u_i)$ and thus the combined forcing is simply damping the disturbance velocity to zero in the fringe region,

$$F_i - G_i = \lambda(x)u_i'\,. \tag{4.38}$$

The linearized Navier–Stokes equations are obtained from equation (4.36) by neglecting the nonlinear terms $u_j'\omega_k'$ and $u_j'u_j'$.

4.6 Pressure solver

By expressing the Navier–Stokes equations in the form of equations (4.8) and (4.11), the pressure does not need to be taken into account. However, it might be of interest to solve for this quantity as well as the velocity components. The pressure can, for example, be used for detecting regions of rapid motion in a turbulent boundary layer.

The Poisson equation for the pressure, equation (4.6), can be written as

$$\nabla^2(p + E) = \frac{\partial H_i}{\partial x_i}\,, \tag{4.39}$$

where $E = \frac{1}{2}u_iu_i$ and H_i as defined in (4.7). Note that the term F_i does not contain any disturbances in the fringe region for the spatial simulations and is zero for the temporal boundary layer. This equation has a similar form as the equations for ϕ, v and ω and can thus be solved using the same numerical routine.

The boundary conditions at the wall ($y = 0$) and at the upper boundary ($y = y_L$) are derived from the normal component of the Navier–Stokes equations. The boundary condition with non-zero wall velocities becomes

$$\left.\frac{\partial}{\partial y}(p + E)\right|_{y=0} = \left.\left(\frac{1}{Re}\nabla^2 v + \epsilon_{ijk}u_j(\omega_k + 2\Omega_k) - \frac{\partial v}{\partial t}\right)\right|_{y=0}. \tag{4.40}$$

The term $\partial v/\partial t$ is included for the case of flow control like blowing/suction from the wall and is approximated with first-order backward differences. For a wall with zero velocities the boundary condition becomes

$$\left.\frac{\partial}{\partial y}(p + E)\right|_{y=0} = \left.\frac{1}{Re}\frac{\partial^2 v}{\partial y^2}\right|_{y=0}. \tag{4.41}$$

At $y = y_L$, for boundary layer flows, the boundary condition becomes

$$\frac{\partial}{\partial y}(p+E)\bigg|_{y=y_L} = \left(\frac{1}{Re}\nabla^2 v + \epsilon_{ijk}u_j(\omega_k + 2\Omega_k) + \lambda(x)(\mathcal{V}-v) - \frac{\partial v}{\partial t}\right)\bigg|_{y=y_L}, \quad (4.42)$$

where $\lambda(x)$ is the fringe function described in section 4.2.2.

For wavenumber zero the boundary condition (4.42) is automatically fulfilled if boundary condition (4.40) is fulfilled. It is required by the compatibility condition

$$\int_0^{y_L} \frac{dH_2}{dy}dy = \frac{\partial}{\partial y}(p+E)|_{y=y_L} - \frac{\partial}{\partial y}(p+E)|_{y=0}\,, \quad (4.43)$$

which comes from the integration of equation (4.39). A second boundary condition for p itself is needed at $y = 0$ and this is chosen to be $p = 0$. The mean pressure at the wall cannot be determined and $p = 0$ at the wall is a reference pressure. It is not reasonable to choose $p = 0$ at $y = y_L$ because the location of the free-stream is arbitrary chosen for numerical purposes.

It might seem to be a better approach to rewrite equation (4.6) as

$$\nabla^2 p = -\frac{\partial u_i}{\partial x_j}\frac{\partial u_j}{\partial x_i} + \frac{\partial}{\partial x_i}(2\epsilon_{ijk}u_j\Omega_k) + \frac{\partial F_i}{\partial x_i}\,, \quad (4.44)$$

and solve for the pressure directly. The solution to equation (4.44) turns out to be sensitive to the values of the velocities at the upper boundary. When using different boundary conditions for the velocities, the solutions are slightly different, hence the pressure will be different. The sensitivity comes from the fact that the derivation in the normal direction in Chebyshev space is dependent on the coefficients in all the collocation points. These coefficients change when transforming back and forth to physical space. Thus the derivations must be, for consistency, performed at the same time, with no transformations between them. These problems are avoided by solving for the pressure plus energy as in equation (4.39).

If turbulent statistics involving pressure are being calculated during a simulation, the pressure is calculated in those time steps where the sampling occurs.

Note that at the moment the pressure calculation is only possible if the full Navier–Stokes equations are solved, *i.e.* evaluating the pressure in disturbance formulation (see previous section 4.5) is not supported.

4.7 Passive scalar

The equation to solve for a passive scalar is

$$\frac{\partial \theta}{\partial t} = -u_i\frac{\partial \theta}{\partial x_i} + \frac{1}{Re\,Pr}\nabla^2\theta \quad (4.45)$$

subject to appropriate boundary conditions. The molecular Prandtl number (fluid dependent) is denoted Pr and the combination $Pe = Re\,Pr$ is the Péclet number. The various terms in equation (4.45) are advanced in time in a similar manner as the corresponding terms in the Navier–Stokes equations (see chapter 5). Boundary conditions for θ need to be given at both the lower wall and the upper boundary (free-stream or wall).

4.8 Selective frequency damping

The selective frequency damping method (SFD) can be used to obtain steady solutions of the Navier–Stokes equations by time marching. This is possible, even for unstable flow configurations, due to an additional forcing term on the right-hand side which

damps all temporally fluctuating parts of the solutions. Further details are given in Åkervik *et al.* (2006). The forcing term reads

$$-\chi(u_i - \overline{u}_i)\,, \tag{4.46}$$

where u_i is the velocity and $\overline{u}_i$ a corresponding *temporally* low-pass filtered flow field. The filtered field is obtained using the differential form of an exponential filter leading to an evolution equation

$$\frac{\partial \overline{u}_i}{\partial t} = \frac{u_i - \overline{u}_i}{\Delta}\,, \tag{4.47}$$

with the filter width Δ. If SFD is enabled, the forcing term (4.46) is explicitely added to the governing equations (4.1), and the evolution equation for the filtered solution (4.47) is solved via the same explicit Runge-Kutta scheme.

4.9 Large-eddy simulation

In large-eddy simulation (LES), the *filtered* Navier–Stokes equations are solved. The application of the primary LES filter is denoted by an overbar, *i.e.* $\overline{u}_i$, $\overline{p}$ and $\overline{\theta}$ for the filtered velocities, pressure and scalar, respectively. Note that this primary LES filter is usually the implicit grid filter due to the lower resolution employed for LES calculations. The governing equations for the resolved (filtered) quantities read

$$\begin{aligned}
\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} &= -\frac{\partial \overline{p}}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 \overline{u}_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}}{\partial x_j}\,, \\
\frac{\partial \overline{u}_j}{\partial x_j} &= 0\,, \\
\frac{\partial \overline{\theta}}{\partial t} + \overline{u}_j \frac{\partial \overline{\theta}}{\partial x_j} &= \frac{1}{Re\,Pr}\nabla^2 \overline{\theta} - \frac{\partial q_j}{\partial x_j}\,.
\end{aligned} \tag{4.48}$$

The unclosed subgrid-scale (SGS) stresses and scalar flux are

$$\begin{aligned}
\tau_{ij} &= \overline{u_i u_j} - \overline{u}_i\,\overline{u}_j\,, \\
q_j &= \overline{u_j \theta} - \overline{u}_j \overline{\theta}\,.
\end{aligned} \tag{4.49}$$

4.9.1 Dynamic Smagorinsky model

The eddy-viscosity ansatz due to Boussinesq relates the deviatoric part of the SGS stresses to the resolved strain rate,

$$\tau_{ij} - \frac{\delta_{ij}}{3}\tau_{kk} = \tau^*_{ij} = -2\nu_t \overline{S}_{ij}\,, \tag{4.50}$$

with the strain rate

$$\overline{S}_{ij} = \frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)\,. \tag{4.51}$$

Within the eddy-diffusivity ansatz, the scalar flux is approximated as being proportional to the mean scalar gradient,

$$q_j = -k_t \frac{\partial \overline{\theta}}{\partial x_j} \tag{4.52}$$

with $k_t = \nu_t / Pr_t$. The turbulent Prandtl number Pr_t is approximately a constant of order one. A standard value is $Pr_t = 0.6$.

The Smagorinsky model (Smagorinsky, 1963) gives the eddy viscosity as follows

$$\nu_t = C\Delta^2 |\overline{S}| \quad \text{with } |\overline{S}| = (2\overline{S}_{ij}\overline{S}_{ij})^{1/2}\,. \tag{4.53}$$

The model coefficient $C = C_S^2$ can be determined dynamically using the well-known dynamic procedure (Germano *et al.*, 1991; Lilly, 1992). Commonly, a two-dimensional spectral cutoff filter at $\omega_c = \pi/2$ acting in the wall-parallel directions is employed

as a test filter. The coefficient C predicted by the dynamic procedure is known to fluctuate strongly in both space and time. Therefore, in the present implementation, a spanwise averaging is performed, *i.e.* $C = C(x, y, t)$. Additionally, a clipping can be employed, either requiring a positive model coefficient $C = \max(C, 0)$, or positive total viscosity $\nu_t = \max(\nu_t, -\nu)$. As common practice, the LES computations are performed on the coarse (*i.e.* non-dealiasing) grid.

4.9.2 High-pass filtered Smagorinsky model

Recent developments in LES have shown that computing the SGS model forces based on the small-scale fluctuations of the flow alone can be very successful and lead to accurate results. This idea was first put forward in the variational multiscale (VMS) approach, where a separation between large and small scales is performed using hierarchical basis function and the modeling is restricted to only the small scales. A physical-space formulation of the VMS method using spatial filters was used in Stolz *et al.* (2005). The high-pass filtered (HPF) eddy-viscosity model, using a high-pass filter H, is written as

$$\tau_{ij} - \frac{1}{3}\tau_{kk}\delta_{ij} = 2\nu_t^{\mathrm{HPF}} S_{ij}(H * \overline{u})\,, \tag{4.54}$$

with both $S_{ij}(H * \overline{u})$ and the eddy viscosity computed from the high-pass filtered velocity,

$$\nu_t^{\mathrm{HPF}} = C\overline{\Delta}^2 |S(H * \overline{u})|\,. \tag{4.55}$$

The symbol $*$ stands for convolution in physical space. This model is denoted as small-"small" since the eddy viscosity ν_t^{HPF} is computed from the small scales, and is applied to the Navier–Stokes equation using the small-scale strain rate. Due to nonlinearity, however, the model influence (4.54) is not restricted to the small part of the governing equations.

A consistent dynamic procedure for the HPF model (4.54), (4.55) has been introduced in Bruhn (2006). Essentially, one can follow a similar way as in the procedure by Germano *et al.* (1991), however, the residual stresses are now taken from their respective HPF formulation.

The high-pass filters employed are based on the filter G defined in Stolz *et al.* (2001). The filter definition for non-equidistant grids, *e.g.* in the wall-normal direction, assures that all moments in physical space up to second order are vanishing. The cutoff wavenumber ω_c is defined by $\hat{G}(\omega_c) = 1/2$ with the hat indicating a Fourier transform (transfer function). Usually, $\omega_c = 2\pi/3$ is a preferred value. Based on G, related high-pass filters can readily be constructed by $H_N = (I - G)^{N+1}$. The three-dimensional filters are derived from the one-dimensional filters by a tensor product. H_N is at least of order $r(N + 1)$ with r being the order of G. The latter is at least $r = 3$ on non-equidistant grids.

4.9.3 Relaxation-term model (ADM-RT)

Whereas the Smagorinsky model is based on the eddy-viscosity assumption, the ADM-RT model acts on the velocity components directly. The model employs the relaxation term used in the context of the approximate deconvolution model (ADM) (Stolz *et al.*, 2001). It has been shown in *e.g.* Schlatter (2005) that for spectral simulations the deconvolution operation applied in the ADM approach is not necessary. Therefore, the SGS force due to the ADM-RT model is given by (Schlatter *et al.*, 2004)

$$\frac{\partial \tau_{ij}}{\partial x_j} = \chi H_N * \overline{u}_i\,. \tag{4.56}$$

χ is the model coefficient which can be set to a constant value herein motivated by previous studies showing little dependency of the results on the actual value of the coefficient (see *e.g.* Schlatter (2005)). A good guess for the model coefficient can be obtained by relating it to the time step of the integration, *i.e.* $\chi \propto 1/\Delta t$, which itself

is related to a physically meaningful quantity (advection of small scales) by the CFL stability condition.

The relaxation term $\chi H_N * \overline{u}_i$ is proportional to the small-scale velocity fluctuations in the flow field. Therefore, it will damp out these oscillations leading to a drain of kinetic energy from the smallest resolved scales. It can be shown that the relaxation term has a similar effect as an explicit filtering of the solution every $(\chi\Delta t)^{-1}$ time steps.

The ADM-RT model proved to be accurate and robust in predicting transitional and turbulent incompressible flows with spectral methods (Schlatter *et al.*, 2004; Schlatter, 2005). Note that the relaxation-term model is related to the spectral vanishing viscosity approach. Due to the high-order filter H_N with a cutoff frequency of $\omega_c \approx 0.86\pi$ only the smallest represented eddies are affected, whereas the larger, energy-carrying scales are not directly influenced by the model contributions.

Similar to the HPF model, a dynamic procedure for the model coefficient χ has been proposed in Bruhn (2006).

4.10 Magneto-Hydrodynamics (MHD)

In the limit of low magnetic Reynolds number ($Re_m \ll 1$), Moreau (1998), the induced magnetic field is very small when compared to an externally imposed magnetic field $\mathbf{B_0}$, and the electric current $\mathbf{J}$ is then given by the Ohm's law,

$$\mathbf{J} = \sigma \ (-\nabla\Phi + \mathbf{u} \times \mathbf{B_0}) \,, \tag{4.57}$$

where σ is the electrical conductivity, Φ the electric potential, ($E = -\nabla\Phi$), E is the electric field, and $\mathbf{u}$ is the velocity field. Using a characteristic magnetic field B the relevant non-dimensional quantities are $\mathbf{b_0} = \mathbf{B_0}/B$, $\mathbf{j} = \mathbf{J}/(\sigma U B)$ and $\phi = \Phi\,\delta^*/(UB)$, with the velocity scale U. Then, the following equations for an electrically conducting incompressible fluid in the limit of $Re_m \ll 1$ are obtained:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2\mathbf{u} + N\ (\mathbf{j} \times \mathbf{b_0}) \,, \tag{4.58}$$

$$\mathbf{j} = -\nabla\phi + \mathbf{u} \times \mathbf{b_0} \,, \tag{4.59}$$

$$\nabla^2\phi = \nabla\cdot(\mathbf{u} \times \mathbf{b_0}) = \mathbf{b_0}\cdot\omega \,, \tag{4.60}$$

where Eq. (4.60) is a consequence of $\nabla\cdot\mathbf{j} = 0$ and is obtained by taking the divergence of Eq. (4.59). Here $\omega = \nabla \times \mathbf{u}$, p the dimensionless pressure and $Re = U\delta^*/\nu$ is the hydrodynamic Reynolds number. Moreover, incompressibility imposes $\nabla\cdot\mathbf{u} = 0$.

The traditional choice $B = |\mathbf{B_0}|$ has been adopted for the characteristic magnetic field so that $\mathbf{b_0}$ is a unit vector in the direction of the magnetic field. The Stuart number (or interaction parameter), $N = \sigma B^2\delta^*/\rho U$, is the ratio of the electromagnetic force to the inertial force and it is related to the Hartmann number by $Ha = \sqrt{Re\,N}(= B\delta^*\sqrt{\sigma/\rho\nu})$ where ρ is the density. Equations (4.58)–(4.60) are referred to as the low-Re_m MHD equations.

At present, in channel geometry the walls are assumed to be non-conducting. Therefore, the boundary conditions for the current density and the electrical potential are

$$j_y|_{\text{wall}} = -\left.\frac{\partial\phi}{\partial y}\right|_{\text{wall}} = 0 \,, \tag{4.61}$$

where j_y is the wall-normal component of the current density.

5 Numerical method

The temporal and spatial discretizations are described in detail in the following sections.

5.1 Temporal discretization

The time advancement is carried out by one out of two semi-implicit schemes. We illustrate them on the equation

$$\frac{\partial \psi}{\partial t} = G + L\psi\,, \tag{5.1}$$

which is of the same form as equation (4.9) and (4.11). Here ψ represents ϕ or ω, G contains the (non-linear) advective, rotation and forcing terms and depends on all velocity and vorticity components. Operator L represents the (linear) diffusion and is discretized implicitly using the second order accurate Crank–Nicolson (CN) scheme. Operator G is discretized explicitly by a low storage third order three or four stage Runge–Kutta (RK3) scheme. These time discretizations may be written in the following manner

$$\psi^{n+1} = \psi^n + a_n G^n + b_n G^{n-1} + (a_n + b_n)\left(\frac{L\psi^{n+1} + L\psi^n}{2}\right), \tag{5.2}$$

where the constants a_n and b_n are chosen according to the explicit scheme used and G and L are assumed to have no explicit dependence on time. The two possibilities for the RK3 schemes are shown in the table 5.1. Note that the schemes have three or four stages which imply that a full physical time step is only achieved every three or four iterations. The time used for the intermediate stages are given by $t = t + c_n$, where c_n is given in table 5.1.

To obtain some insight into the properties of these discretizations they will be applied to the two dimensional linearized Burgers' equation with a system rotation. The eigenvalue analysis yields a necessary condition for stability which must be augmented

	$a_n/\Delta t^n$	$b_n/\Delta t^n$	$c_n/\Delta t^n$
RK3	8/15	0	0
3-stage	5/12	−17/60	8/15
	3/4	−5/12	2/3
RK3	8/17	0	0
4-stage	17/60	−15/68	8/17
	5/12	−17/60	8/15
	3/4	−5/12	2/3

Table 5.1: Time stepping coefficients.

by an experimental verification. Putting the equation into the form of equation (5.1) yields

$$\begin{aligned}
\psi &= \begin{bmatrix} u \\ w \end{bmatrix}, \\
G &= \begin{bmatrix} u_0\partial/\partial x + w_0\partial/\partial z & 2\Omega \\ -2\Omega & u_0\partial/\partial x + w_0\partial/\partial z \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix}, \\
L &= \frac{1}{Re}\begin{bmatrix} \partial^2/\partial x^2 + \partial^2/\partial z^2 & 0 \\ 0 & \partial^2/\partial x^2 + \partial^2/\partial z^2 \end{bmatrix},
\end{aligned} \tag{5.3}$$

where u_0 and w_0 represent the flow field around which the linearization was made. System (5.3) can be seen as an approximation to equation (4.1). The dependence of ψ on both the streamwise and spanwise coordinate directions have been included in order to indicate how multiple dimensions enter into the stability considerations.

We will for simplicity use Fourier discretization in the spatial directions. The Chebyshev method acts locally as a transformed Fourier method and thus the stability properties derived here can be applied with the local space step. An exception to this occurs at the end points where the transformation is singular. It can be shown that the Chebyshev method is more stable there. A numerical study of a one-dimensional advection equation using the Chebyshev discretization yields that the upper limit of its spectrum along the imaginary axis is about 16 times lower than the simple application of the results from the Fourier method. This allows a corresponding increase of the time step when the stability is limited by the wall-normal velocity at the free-stream boundary.

Fourier transforming in x- and z-directions yields

$$\hat{\psi}_t = \begin{bmatrix} i\alpha u_0 + i\beta w_0 & 2\Omega \\ -2\Omega & i\alpha u_0 + i\beta w_0 \end{bmatrix} \hat{\psi} - \frac{\alpha^2+\beta^2}{Re}\hat{\psi}\,, \tag{5.4}$$

where α and β are the wavenumbers in the x- and z-directions, respectively. This equation can be diagonalized to give the equation,

$$\hat{u}_t = i(\alpha u_0 + \beta w_0 \pm 2\Omega)\hat{u} + \frac{\alpha^2+\beta^2}{Re}\hat{u}\,. \tag{5.5}$$

We assume that the absolute stability limit will first be reached for the largest wavenumbers of the discretization $\alpha_{\max}$ and $\beta_{\max}$, which corresponds to a wavelength of $2\Delta x$ and $2\Delta z$, respectively, where Δx and Δz are the discretization step lengths in physical space. The following parameters are useful for our analysis,

$$\begin{aligned}
\mu &= \Delta t(2|\Omega_k| + (\alpha_{\max}|u_0| + \beta_{\max}|w_0|)) \\
&= \Delta t\left(2|\Omega_k| + \pi\left(\frac{|u_0|}{\Delta x} + \frac{|w_0|}{\Delta z}\right)\right), \\
\lambda &= \frac{1}{Re}\Delta t(\alpha_{\max}^2 + \beta_{\max}^2) \\
&= \frac{1}{Re}\pi^2\Delta t\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta z^2}\right).
\end{aligned} \tag{5.6}$$

The parameter μ is usually called the spectral CFL number, in analogy with the stability theory for finite difference equations. Henceforth it will be termed simply the CFL number. Using the RK3–CN time discretization we have the following three stages in each time step for the model equation (5.5),

$$\begin{aligned}
\hat{u}^{n+1} &= \hat{u}^n + i\mu a_1\hat{u}^n - \frac{\lambda}{2}a_1(\hat{u}^{n+1} + \hat{u}^n)\,, \\
\hat{u}^{n+2} &= \hat{u}^{n+1} + i\mu(a_2\hat{u}^{n+1} + b_2\hat{u}^n) - \frac{\lambda}{2}(a_2+b_2)(\hat{u}^{n+2} + \hat{u}^{n+1})\,, \\
\hat{u}^{n+3} &= \hat{u}^{n+2} + i\mu(a_3\hat{u}^{n+2} + b_3\hat{u}^{n+1}) - \frac{\lambda}{2}(a_3+b_3)(\hat{u}^{n+3} + \hat{u}^{n+2})\,.
\end{aligned} \tag{5.7}$$

The absolute stability regions, *i.e.* the regions where all solutions to the above difference equations are bounded in the μ – λ plane, can now be found by calculating

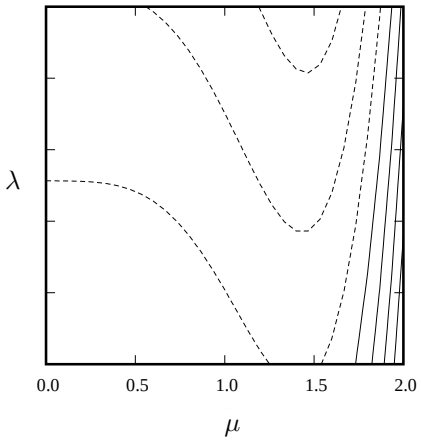


Figure 5.1: Contours of constant amplification factor for the RK3-CN method. Contour spacing is 0.05 with dashed lines indicating that the amplification factor is below unity.

the roots of the associated characteristic polynomials. Contours of constant absolute values of the roots, for the RK3-CN method, are given in figure 5.1. Note that higher values of λ (lower Reynolds number) stabilizes the method, *i.e.* increases the CFL number μ that is allowed for an absolutely stable solution. In the limit of infinite Reynolds number the RK3-CN method approaches the limit $\sqrt{3}$, a result which also can be arrived at through the standard analysis of the RK3 scheme alone. The analysis for the four stage method is analogous and the stability limit is $\sqrt{8}$.

If the time advancement scheme (5.2) is applied to equations (4.9) and (4.11) we find (for the moment disregarding the boundary conditions),

$$\begin{aligned}\left(1-\frac{a_n+b_n}{2Re}\nabla^2\right)\phi^{n+1} &= \left(1+\frac{a_n+b_n}{2Re}\nabla^2\right)\phi^n + a_n h_v^n + b_n h_v^{n-1}\,,\\ \nabla^2 v^{n+1} &= \phi^{n+1}\,,\end{aligned} \tag{5.8}$$

and

$$\left(1-\frac{a_n+b_n}{2Re}\nabla^2\right)\omega^{n+1} = \left(1+\frac{a_n+b_n}{2Re}\nabla^2\right)\omega^n + a_n h_\omega^n + b_n h_\omega^{n-1}\,. \tag{5.9}$$

5.2 Horizontal discretization – Fourier expansions

The discretization in the horizontal directions uses a Fourier series expansion which assumes that the solution is periodic.

The streamwise and spanwise dependence of each variable can then be written

$$u(x,z) = \sum_{l=-(\frac{N_x}{2}-1)}^{\frac{N_x}{2}-1}\ \sum_{m=-(\frac{N_z}{2}-1)}^{\frac{N_z}{2}-1} \hat{u}(\alpha_l,\beta_m)e^{(i(\alpha_l x+\beta_m z))}\,, \tag{5.10}$$

where $\alpha_l = 2\pi l/x_L$ and $\beta_m = 2\pi m/z_L$, and N_x and N_z are the number of Fourier modes included in the respective directions. Note that the indices on the discrete wavenumbers α and β are sometimes left out for notational convenience and that $k^2 = \alpha^2 + \beta^2$.

5.2.1 Normal velocity and normal vorticity equations

Expanding the dependent variables of equation (5.8) in Fourier series gives

$$\begin{aligned}\left(1-\frac{a_n+b_n}{2Re}(D^2-k^2)\right)\hat{\phi}^{n+1} &= \left(1+\frac{a_n+b_n}{2Re}(D^2-k^2)\right)\hat{\phi}^{n} \\ &\quad + a_n\hat{h}_v^n + b_n\hat{h}_v^{n-1}\,, \\ (D^2-k^2)\hat{v}^{n+1} &= \hat{\phi}^{n+1}\,,\end{aligned} \tag{5.11}$$

where D signifies a derivative in the normal direction. Note that the above equations are two linear constant coefficient second order ordinary differential equations in y. A similar equation can also be derived from equation (5.9). These three equations can be written as follows

$$\begin{aligned}(D^2-\lambda^2)\hat{\phi}^{n+1} &= \hat{f}_v^n\,, \\ (D^2-k^2)\hat{v}^{n+1} &= \hat{\phi}^{n+1}\,, \\ (D^2-\lambda^2)\hat{\omega}^{n+1} &= \hat{f}_\omega^n\,,\end{aligned} \tag{5.12}$$

where

$$\begin{aligned}\lambda^2 &= k^2 + 2Re/(a_n+b_n)\,, \\ \hat{f}_v^n &= \hat{p}_v^n - \frac{2Re\,a_n}{a_n+b_n}\hat{h}_v^n\,, \\ \hat{f}_\omega^n &= \hat{p}_\omega^n - \frac{2Re\,a_n}{a_n+b_n}\hat{h}_\omega^n\,,\end{aligned} \tag{5.13}$$

and

$$\begin{aligned}\hat{p}_v^n &= -\left(D^2-\lambda^2+\frac{4Re}{a_n+b_n}\right)\hat{\phi}^n - \frac{2Re\,b_n}{a_n+b_n}\hat{h}_v^{n-1} \\ &= -\hat{f}_v^{n-1} - \left(\frac{2Re}{a_{n-1}+b_{n-1}}+\frac{2Re}{a_n+b_n}\right)\hat{\phi}^n - \frac{2Re\,b_n}{a_n+b_n}\hat{h}_v^{n-1}, \\ \hat{p}_\omega^n &= -\left(D^2-\lambda^2+\frac{4Re}{a_n+b_n}\right)\hat{\omega}^n - \frac{2Re\,b_n}{a_n+b_n}\hat{h}_\omega^{n-1} \\ &= -\hat{f}_\omega^{n-1} - \left(\frac{2Re}{a_{n-1}+b_{n-1}}+\frac{2Re}{a_n+b_n}\right)\hat{\omega}^n - \frac{2Re\,b_n}{a_n+b_n}\hat{h}_\omega^{n-1}\,.\end{aligned} \tag{5.14}$$

We will denote the quantities $\hat{p}_\omega^n$ and $\hat{p}_v^n$ the partial right hand sides of the equations.

5.2.2 Horizontal velocities and wavenumber zero

Having obtained $\hat{v}$ and $\hat{\omega}$ we can find $\hat{u}$ and $\hat{w}$ using equation (4.2) and the definition of the normal vorticity component, both transformed to Fourier space. We find

$$\hat{u} = \frac{i}{k^2}(\alpha D\hat{v} - \beta\hat{\omega})\,, \tag{5.15a}$$

$$\hat{w} = \frac{i}{k^2}(\alpha\hat{\omega} + \beta D\hat{v})\,. \tag{5.15b}$$

Similarly, we can find the streamwise and spanwise component of vorticity in terms of $\hat{\omega}$ and $\hat{\phi}$,

$$\hat{\chi} = \frac{i}{k^2}(\alpha D\hat{\omega} + \beta\hat{\phi})\,, \tag{5.16a}$$

$$\hat{\vartheta} = \frac{-i}{k^2}(\alpha\hat{\phi} + \beta D\hat{\omega})\,. \tag{5.16b}$$

These relations give the streamwise and spanwise components of velocity and vorticity for all wavenumber combinations, except when both α and β are equal to zero. In that case we have to use some other method to find $\hat{u}_0$, $\hat{w}_0$, $\hat{\chi}_0$ and $\hat{\vartheta}_0$ (the zero subscript indicates that $k = 0$). The appropriate equations are found by taking the

horizontal average of the first and the third component of equation (4.1). Due to the periodic boundary condition all horizontal space derivatives cancel out, *i.e.*,

$$\begin{aligned} \frac{\partial u_0}{\partial t} &= H_1 + \frac{1}{Re}\frac{\partial^2 u_0}{\partial y^2}\,, \\ \frac{\partial w_0}{\partial t} &= H_3 + \frac{1}{Re}\frac{\partial^2 w_0}{\partial y^2}\,. \end{aligned} \tag{5.17}$$

After a time discretization we find,

$$\begin{aligned} (D^2-\lambda^2)\hat{u}_0^{n+1} &= \hat{f}_{01}^n\,, \\ (D^2-\lambda^2)\hat{w}_0^{n+1} &= \hat{f}_{03}^n\,, \end{aligned} \tag{5.18}$$

where

$$\hat{f}_{0i}^n = \hat{p}_{0i}^n - \frac{2Re\,a_n}{a_n+b_n}\hat{H}_{0i}^n\,, \tag{5.19}$$

and

$$\begin{aligned} \hat{p}_{0i}^n &= -\left(D^2-\lambda^2+\frac{4Re}{a_n+b_n}\right)\hat{u}_{0i}^n - \frac{2Re\,b_n}{a_n+b_n}\hat{H}_{0i}^{n-1} \\ &= -\hat{f}_{0i}^{n-1} - \left(\frac{2Re}{a_{n-1}+b_{n-1}}+\frac{2Re}{a_n+b_n}\right)\hat{u}_{0i}^n - \frac{2Re\,b_n}{a_n+b_n}\hat{H}_{0i}^{n-1}\,. \end{aligned} \tag{5.20}$$

Here the 0 index in $\hat{H}_{0i}$ refers to the zero wavenumber in both horizontal directions. Note that the above system contains the same type of equations as the system (5.12), and can thus be solved using the same numerical algorithm. Once $\hat{u}_0$ and $\hat{w}_0$ are calculated, the streamwise and spanwise components of vorticity for $k=0$ can be found as follows

$$\begin{aligned} \hat{\chi}_0 &= D\hat{w}_0\,, \\ \hat{\vartheta}_0 &= -D\hat{u}_0\,. \end{aligned} \tag{5.21}$$

5.2.3 Solution procedure with boundary conditions

A problem with the above equations is that the boundary conditions do not apply to the quantities for which we have differential equations. To remedy this, each of the equations can be solved for a particular solution with homogeneous boundary conditions. Then a number of homogeneous solutions with non-homogeneous boundary conditions are found for the same equations. Finally the boundary conditions are fulfilled by a suitable linear combination of particular and homogeneous solutions. Explicitly we proceed as follows:

For all $k=\sqrt{\alpha^2+\beta^2}\neq 0$ and each of the two symmetries (symmetric and antisymmetric with respect to reflections around $y=y_L/2$) we solve:

$$(D^2-\lambda^2)\hat{\phi}_p^{n+1} = \hat{f}_v^{n+1}\,, \qquad \hat{\phi}_p^{n+1}(y_L)=0\,, \tag{5.22a}$$

$$(D^2-k^2)\hat{v}_p^{n+1} = \hat{\phi}_p^{n+1}\,, \qquad \hat{v}_p^{n+1}(y_L) = \begin{cases} \frac{\hat{v}_{\mathrm{BS}}}{2} & \text{symmetric}\,, \\ -\frac{\hat{v}_{\mathrm{BS}}}{2} & \text{antisymmetric}\,, \end{cases} \tag{5.22b}$$

$$(D^2-\lambda^2)\hat{\phi}_h^{n+1} = 0\,, \qquad \hat{\phi}_h^{n+1}(y_L)=1\,, \tag{5.22c}$$

$$(D^2-k^2)\hat{v}_{ha}^{n+1} = \hat{\phi}_h^{n+1}\,, \qquad \hat{v}_{ha}^{n+1}(y_L)=0\,, \tag{5.22d}$$

$$(D^2-k^2)\hat{v}_{hb}^{n+1} = 0\,, \qquad \hat{v}_{hb}^{n+1}(y_L)=1\,, \tag{5.22e}$$

$$(D^2-\lambda^2)\hat{\omega}_p^{n+1} = \hat{f}_\omega^{n+1}\,, \qquad \hat{\omega}_p^{n+1}(y_L)=0\,, \tag{5.22f}$$

$$(D^2-\lambda^2)\hat{\omega}_h^{n+1} = 0\,, \qquad \hat{\omega}_h^{n+1}(y_L)=1\,, \tag{5.22g}$$

where the subscripts p, h, ha and hb indicate the particular and the homogeneous parts. The variable $\hat{v}_{\mathrm{BS}}$ is only non-zero for cases with blowing and suction through the wall. Note that only one boundary condition is needed for each second order equation since the assumption of symmetry (or antisymmetry) takes care of the other.

The particular solution of the wall-normal velocity, $\hat{v}_p^{n+1}(y_L)$, is zero at the upper boundary when the symmetric and antisymmetric solutions are added and all the other solutions are zero at $y = 0$. Equations (5.22c) and (5.22g) have zero right hand sides and the same boundary conditions. The solution coefficients are therefore identical and so are also their symmetric and antisymmetric coefficients. Thus, four calls of the equation solver can be reduced to one.

To fulfill the remaining boundary conditions one first constructs $\hat{v}_{p1}$, $\hat{v}_{h1}$ and $\hat{v}_{h2}$,

$$\begin{aligned}
\hat{v}_{p1}^{n+1} &= \hat{v}_p^{n+1} + C_{p1}\hat{v}_{ha}^{n+1}\,, & \hat{v}_{p1}^{n+1}(y_L) &= 0\,, & \hat{v}_{p1}^{n+1}(0) &= v_{\mathrm{BS}}/2\,,\\
\hat{v}_{h1}^{n+1} &= \hat{v}_{ha}^{n+1} \Big/ \left.\frac{\partial \hat{v}_{ha}}{\partial y}\right|_{y=y_L}\,, & \hat{v}_{h1}^{n+1}(y_L) &= 0\,, & \hat{v}_{h1}^{n+1}(0) &= 0\,,\\
\hat{v}_{h2}^{n+1} &= \hat{v}_{hb}^{n+1} + C_{h2}\hat{v}_{ha}^{n+1}\,, & \hat{v}_{h2}^{n+1}(y_L) &= 1\,, & \hat{v}_{h2}^{n+1}(0) &= 0\,,
\end{aligned} \tag{5.23}$$

where C_{p1} and C_{h2} are chosen to fulfill the boundary condition $\partial v/\partial y = 0$ at the lower wall for each of the two symmetries of $\hat{v}_{p1}$ and $\hat{v}_{h2}$. As the symmetric and antisymmetric parts of $\partial \hat{v}_{h1}/\partial y$ cancel at the lower wall their sum v_{h1} fulfills $\partial v_{h1}/\partial y = 0$.

Now the solutions (v_{p1}, ω_p), $(v_{h1}, \omega = 0)$, $(v_{h2}, \omega = 0)$ and $(v = 0, \omega_h)$ fulfill all the physical boundary conditions at the lower wall. The total normal velocity is then given by

$$\hat{v}^{n+1} = \hat{v}_{p1}^{n+1} + C_{v1}\hat{v}_{h1}^{n+1} + C_{v2}\hat{v}_{h2}^{n+1}\,, \tag{5.24}$$

and the vorticity by

$$\hat{\omega}^{n+1} = \hat{\omega}_p^{n+1} + C_\omega \hat{\omega}_h^{n+1}\,, \tag{5.25}$$

where C_{v1}, C_{v2} and C_ω are chosen such that the boundary conditions at the upper boundary are fulfilled. The u and w velocities are found from the definition of the normal vorticity and the incompressibility constraint.

In general we have to find u and w first to evaluate the boundary conditions. Thus with the C's unknown we find:

$$\begin{aligned}
\hat{u}^{n+1} &= \hat{u}_{p1}^{n+1} + C_{v1}\hat{u}_{h1}^{n+1} + C_{v2}\hat{u}_{h2}^{n+1} + C_\omega \hat{u}_h^{n+1}\,,\\
\hat{w}^{n+1} &= \hat{w}_{p1}^{n+1} + C_{v1}\hat{w}_{h1}^{n+1} + C_{v2}\hat{w}_{h2}^{n+1} + C_\omega \hat{w}_h^{n+1}\,,
\end{aligned} \tag{5.26}$$

where (u_{p1}, w_{p1}), (u_{h1}, w_{h1}), (u_{h2}, w_{h2}) and (u_h, w_h) are found from (v_{p1}, ω_p), $(v_{h1}, \omega = 0)$, $(v_{h2}, \omega = 0)$ and $(v = 0, \omega_h)$ using equation (5.15a) and (5.15b).

Assuming the boundary conditions are linear we can write them as:

$$L_i(\hat{u}, \hat{v}, \hat{w}) = \hat{D}_i\,, \quad i = 1, 2, 3\,. \tag{5.27}$$

Here L_i is the linear operator for the ith boundary condition. This can include derivatives in the wall-normal direction. The operator may also depend on the wave number (for example when the boundary condition contains horizontal derivatives). Note that the expression for evaluation L_i may include $\hat{\omega}$ as this is equivalent to horizontal derivatives. The right hand side $\hat{D}_i$ is the data for the boundary condition, the most common form of which is either zero (homogeneous boundary conditions) or the operator L_i applied to a base flow.

Finally inserting the expressions (5.24) and (5.26) into equation (5.27) and moving all terms containing the particular solution to the right hand side, we get a three by three linear system of equations which is easily solved to find the C's.

For $k = 0$ we solve

$$\begin{aligned}
(D^2 - \lambda^2)\hat{u}_{p0}^{n+1} &= \hat{f}_{01}^{n}\,, & \hat{u}_{p0}^{n+1}(0) &= u_{\mathrm{low}}\,, & \hat{u}_{p0}^{n+1}(y_L) &= u_{\mathrm{upp}}\,,\\
(D^2 - \lambda^2)\hat{w}_{p0}^{n+1} &= \hat{f}_{03}^{n}\,, & \hat{w}_{p0}^{n+1}(0) &= w_{\mathrm{low}}\,, & \hat{w}_{p0}^{n+1}(y_L) &= w_{\mathrm{upp}}\,,\\
(D^2 - \lambda^2)\hat{u}_{h0}^{n+1} &= 0\,, & \hat{u}_{h0}^{n+1}(0) &= 0\,, & \hat{u}_{h0}^{n+1}(y_L) &= 2\,,\\
(D^2 - \lambda^2)\hat{w}_{h0}^{n+1} &= 0\,, & \hat{w}_{h0}^{n+1}(0) &= 0\,, & \hat{w}_{h0}^{n+1}(y_L) &= 2\,,
\end{aligned} \tag{5.28}$$

where u_{low}, u_{upp}, w_{low} and w_{upp} denote the lower and upper wall velocities. Computations in a moving reference frame can increase the time step. If the boundary

condition at the upper wall is in the form of Dirichlet type (specified velocity) then

$$\begin{aligned} \hat{u}_0 &= \hat{u}_{p0}\,, \\ \hat{w}_0 &= \hat{w}_{p0}\,. \end{aligned} \tag{5.29}$$

For other types of upper wall boundary conditions we find the complete solution from

$$\begin{aligned} \hat{u}_0 &= \hat{u}_{p0} + C_u \hat{u}_{h0}\,, \\ \hat{w}_0 &= \hat{w}_{p0} + C_w \hat{w}_{h0}\,, \end{aligned} \tag{5.30}$$

where C_u and C_w are chosen so that $\hat{u}_0$ and $\hat{w}_0$ fulfill the boundary conditions.

The above equations are all in Fourier space, where the non-linear terms h_v, h_ω, H_1 and H_3 become convolution sums. These sums can be efficiently calculated by transforming the velocities and vorticities using FFTs to physical space, where they are evaluated using pointwise products.

5.3 Normal discretization – Chebyshev expansion

The typical equation derived above is a second order constant coefficient ordinary differential equation of the form

$$(D^2 - \kappa)\hat{\psi} = \hat{f}\,, \quad \hat{\psi}(0) = \gamma_{-1}\,, \quad \hat{\psi}(y_L) = \gamma_1\,. \tag{5.31}$$

For boundary layer flows first map the interval $[0, y_l]$ to $[-1, 1]$ by setting $y' = 2y/y_L - 1$. Then

$$(D'^2 - \nu)\hat{\psi} = \hat{f}\,, \quad \hat{\psi}(-1) = \gamma_{-1}\,, \quad \hat{\psi}(1) = \gamma_1\,, \tag{5.32}$$

where $\nu = \kappa y_L^2/4$. In the following we have for simplicity dropped the prime.

This equation can be solved accurately if the dependent variable $\hat{\psi}$, its second derivatives, the right hand side $\hat{f}$ and the boundary conditions are expanded in Chebyshev series, *i.e.*,

$$\hat{\psi}(y) = \sum_{j=0}^{N_y} \tilde{\psi}_j T_j(y)\,, \tag{5.33a}$$

$$D^2\hat{\psi}(y) = \sum_{j=0}^{N_y} \tilde{\psi}_j^{(2)} T_j(y)\,, \tag{5.33b}$$

$$\hat{f}(y) = \sum_{j=0}^{N_y} \tilde{f}_j T_j(y)\,, \tag{5.33c}$$

$$\hat{\psi}(1) = \sum_{j=0}^{N_y} \tilde{\psi}_j = \gamma_1\,, \tag{5.33d}$$

$$\hat{\psi}(-1) = \sum_{j=0}^{N_y} (-1)^j \tilde{\psi}_j = \gamma_{-1}\,, \tag{5.33e}$$

$$D\hat{\psi}(1) = \sum_{j=1}^{N_y} j^2 \tilde{\psi}_j = \delta_1\,, \tag{5.33f}$$

$$D\hat{\psi}(-1) = \sum_{j=1}^{N_y} j^2 (-1)^{j+1} \tilde{\psi}_j = \delta_{-1}\,, \tag{5.33g}$$

where T_j are the Chebyshev polynomial of order j and N_y the highest order of polynomial included in the expansion. If the Chebyshev expansions are used in equation (5.32), together with the orthogonality properties of the Chebyshev polynomials, we find the following relation between the coefficients

$$\tilde{\psi}_j^{(2)} - \nu\tilde{\psi}_j = \tilde{f}_j\,, \quad j = 0, ..., N_y\,. \tag{5.34}$$

By writing the Chebyshev functions as cosines and using well known trigonometric identities, one finds relations between the Chebyshev coefficients of $\hat{\psi}$ and those of its derivative that can be used for differentiation and integration (Canuto *et al.* (1988))

$$\tilde{\psi}_j^{(p)} = \sum_{\substack{m=j+1 \\ m+j \text{ odd}}}^{N_y} m\tilde{\psi}_m^{(p-1)}\,, \qquad j = 1,\ldots,N_y\,, \tag{5.35a}$$

$$\tilde{\psi}_j^{(p-1)} = \frac{1}{2j}(c_{j-1}\tilde{\psi}_{j-1}^{(p)} - \tilde{\psi}_{j+1}^{(p)})\,, \qquad j = 1,\ldots,N_y\,, \tag{5.35b}$$

where the superscript p indicates the order of the derivative and $c_j = 2$ for $j = 0$ and $c_j = 1$ for $j > 0$. In the first differentiation relation one observes that an error in the highest order coefficients of $\tilde{\psi}^{(p-1)}$ influences all coefficients of its derivative $\tilde{\psi}^{(p)}$. This problem is what is supposed to be avoided by the Chebyshev integration method discussed below. In the second relation we assume that $\tilde{\psi}_j^{(p)} = 0$ for $j > N_y$ and note that $\tilde{\psi}_0^{(p-1)}$ is an integration constant needed when the function $\hat{\psi}^{(p-1)}$ is found by integrating $\hat{\psi}^{(p)}$. Note also that the integration procedure introduces a truncation error, since an integration of a Chebyshev polynomial would result in a polynomial of one degree higher. The coefficient $\tilde{\psi}_{N_y+1}^{(p-1)}$ which would have multiplied T_{N_y+1} is in the present truncation set to zero.

If the relations (5.35a) and (5.35b) are used together with relation (5.34) a system of equations can be derived for either coefficients $\tilde{\psi}_j$ or $\tilde{\psi}_j^{(2)}$. The second approach, called the Chebyshev integration method (CIM), was proposed by Greengard (1991) to avoid the ill conditioned process of numerical differentiation in Chebyshev space. It was implemented in the original channel code by Lundbladh *et al.* (1992a) and is also included in the present implementation. However, it has been found that using this method, subtle numerical instabilities occur in some cases and it is therefore recommended to solve for the coefficients of the function itself, $\tilde{\psi}_j$. Such a Chebyshev tau method (CTM), almost identical to that used by Kim, Moin & Moser, is also implemented and is so far found to be stable. Note that the instabilities have occurred only a few times and that the results otherwise are the same for the two methods.

5.3.1 Chebyshev tau method – CTM

If the recursion relation (5.35a) is used to express equations (5.34) in the coefficients $\tilde{\psi}_j$, one arrives at the system of equations (5.36 below). A more detailed derivation can be found in Canuto *et al.* (1988), but observe the sign errors therein. We have

$$\begin{aligned} &-\frac{c_{j-2}\nu}{4j(j-1)}\tilde{\psi}_{j-2} + \left(1 + \frac{\nu\beta_j}{2(j^2-1)}\right)\tilde{\psi}_j - \frac{\nu}{4j(j+1)}\tilde{\psi}_{j+2} \\ &= \frac{c_{j-2}}{4j(j-1)}\tilde{f}_{j-2} - \frac{\beta_j}{2(j^2-1)}\tilde{f}_j + \frac{\beta_{j+2}}{4j(j+1)}\tilde{f}_{j+2}\,, \quad j = 2,\ldots,N_y \end{aligned} \tag{5.36}$$

where

$$\beta_j = \begin{cases} 1 & 0 \le j \le N_y - 2\,, \\ 0 & j > N_y - 2\,. \end{cases} \tag{5.37}$$

Note that the even and odd coefficients are uncoupled. Since a Chebyshev polynomial with an odd index is an odd function, and vice versa, the decoupling of the systems of equations is just a result of the odd and even decoupling of equation (5.32) itself. The same can be achieved for the boundary conditions (5.33d)–(5.33g) if they are added and subtracted,

$$\begin{aligned} \sum_{\substack{j=0 \\ j \text{ even}}}^{N_y} \tilde{\psi}_j &= \frac{\gamma_1 + \gamma_{-1}}{2}\,, \quad \sum_{\substack{j=1 \\ j \text{ odd}}}^{N_y} \tilde{\psi}_j = \frac{\gamma_1 - \gamma_{-1}}{2}\,, \\ \sum_{\substack{j=2 \\ j \text{ even}}}^{N_y} j^2\tilde{\psi}_j &= \frac{\delta_1 - \delta_{-1}}{2}\,, \quad \sum_{\substack{j=1 \\ j \text{ odd}}}^{N_y} j^2\tilde{\psi}_j = \frac{\delta_1 + \delta_{-1}}{2}\,. \end{aligned} \tag{5.38}$$

These boundary conditions together with the equations (5.36) constitute a linear system of N_y+1 equations that can be solved for the coefficients $\tilde{\psi}_j$ $(j=0,\ldots,N_y)$. The structure of the equations involving the even coefficients forms a tridiagonal system and so does the equation for the odd coefficients. The boundary conditions fill the top row of both systems and make the systems only quasi-tridiagonal, but it only takes $16N_y$ operations to solve both systems.

The system (5.36) has in fact been truncated to only contain N_y-1 equations and two equations have been replaced by boundary conditions. That truncation introduces what is usually called the tau error. In solution algorithms that solve for the three velocity components of the Navier–Stokes equations and the pressure, the coupling between the equations for the velocities and that for the pressure require corrections of the tau error (Kleiser & Schumann (1980)). We have chosen to eliminate the pressure in the Navier–Stokes equations and solve for the normal velocity and the normal vorticity. As those equations do not couple in the same way, we do not have to correct the tau error.

5.3.2 Chebyshev integration method – CIM

Instead of solving for the coefficients $\tilde{\psi}_j$, the CIM solves for the coefficients of the Chebyshev series for the second derivative, $\tilde{\psi}^{(2)}_j$. The major advantage is supposed to come in the calculation of derivatives of the solution $\hat{\psi}$. Derivatives are needed in the calculation of the remaining velocities and vorticities using equations (5.15a)–(5.16b). In the CIM the second derivative is already calculated and the first derivative and the function itself can be found by the numerically well conditioned process of integration.

If the relations (5.35b) are used to write (5.34) in terms of $\tilde{\psi}^{(2)}_j$ the result is the following system of equations,

$$
\begin{gathered}
\tilde{\psi}^{(2)}_0 - \nu\tilde{\psi}_0 = \tilde{f}_0\,, \quad j=0\,,\\
\tilde{\psi}^{(2)}_1 - \nu(\tilde{\psi}^{(1)}_0 - \frac{1}{8}\tilde{\psi}^{(2)}_1 + \frac{1}{8}\tilde{\psi}^{(2)}_3 = \tilde{f}_1\,, \quad j=1\,,\\
\tilde{\psi}^{(2)}_j - \nu\frac{1}{4j}\\
\left(\frac{c_{j-2}\tilde{\psi}^{(2)}_{j-2}}{j-1} - \tilde{\psi}^{(2)}_j\left(\frac{1}{j-1}+\frac{1}{j+1}\right) + \frac{\tilde{\psi}^{(2)}_{j+2}}{j+1}\right) = \tilde{f}_j\,, \quad 2\le j\le N_y-2\,,\\
\tilde{\psi}^{(2)}_{N_y-1} - \nu\frac{1}{4(N_y-1)}\\
\left(\frac{\tilde{\psi}^{(2)}_{N_y-3}}{N_y-2} - \tilde{\psi}^{(2)}_{N_y-1}\left(\frac{1}{N_y-2}+\frac{1}{N_y}\right)\right) = \tilde{f}_{N_y-1}\,, \quad j=N_y-1\,,\\
\tilde{\psi}^{(2)}_{N_y} - \nu\frac{1}{4N_y(N_y-1)}(\tilde{\psi}^{(2)}_{N_y-2} - \tilde{\psi}^{(2)}_{N_y}) = \tilde{f}_{N_y}\,, \quad j=N_y\,.
\end{gathered}
\tag{5.39}
$$

The equations for odd and even coefficients decouple and so do the boundary conditions on the form (5.38). However, we now need to rewrite them with the aid of (5.34) to contain the coefficients of $\tilde{\psi}^{(2)}$ that we are now solving for. We find that the first sum in (5.38) takes the form,

$$
\begin{gathered}
\tilde{\psi}_0 + \tilde{\psi}^{(1)}_0 + \frac{1}{4}\tilde{\psi}^{(2)}_0 - \frac{1}{12}\tilde{\psi}^{(2)}_1 - \frac{7}{48}\tilde{\psi}^{(2)}_2 +\\
\sum_{j=3}^{N_y-2}\frac{3\tilde{\psi}^{(2)}_j}{(j-2)(j-1)(j+1)(j+2)} -\\
\frac{(N_y-6)\tilde{\psi}^{(2)}_{N_y-1}}{4(N_y-3)(N_y-2)N_y} - \frac{\tilde{\psi}^{(2)}_{N_y}}{2(N_y-2)(N_y-1)N_y} = \gamma_1\,.
\end{gathered}
\tag{5.40}
$$

Thus, the solution of equation (5.32) is found by solving the system of equations for the second derivative of $\tilde{\psi}$ (5.39) together with the boundary conditions (5.40) and

the corresponding one at $y = -1$. We now have two more equations than for the tau method and the solution to the full system is a set of $N_y + 1$ coefficients of the second derivative and the two integration constants $\tilde{\psi}_0^{(1)}$ and $\tilde{\psi}_0^{(2)}$ representing the zeroth order Chebyshev coefficient of $D\hat{\psi}$ and $\hat{\psi}$ itself, respectively. The function $\hat{\psi}$ is then found by two integrations, which in Chebyshev space can easily be constructed using the relations (5.35b). The same quasi-tridiagonal form of the equation systems for the odd and even coefficients appears as for the CTM and the same solution routine can be used.

5.3.3 Integration correction

When the solution for $\hat{\psi}^{(2)}$ is found by the CIM and integrated to obtain $\hat{\psi}^{(1)}$ and $\hat{\psi}$ the same truncation is used for both the derivatives and $\hat{\psi}$ itself. They are all represented with $N_y + 1$ non-zero Chebyshev coefficients. This means that the truncations are not compatible, since the derivative of a function represented as a finite Chebyshev series should have one coefficient less than the function itself. For example, if the coefficients $\tilde{\psi}_j$ are used to construct those for the derivative, using the recurrence relation (5.35a), the result will not be the same as the coefficients $\tilde{\psi}_j^{(1)}$. There will be a slight difference in half of the coefficients for the derivative, the size depending on the magnitude of the coefficient $\tilde{\psi}_{N_y}$. The expression for the difference can be derived as follows. We write $\hat{\psi}$ explicitly using the coefficients $\tilde{\psi}_j^{(1)}$ and the relation (5.35b),

$$\hat{\psi} = \tilde{\psi}_0 T_0 + \sum_{j=1}^{N_y-1} \frac{1}{2j}(c_{j-1}\tilde{\psi}_{j-1}^{(1)} - \tilde{\psi}_{j+1}^{(1)})T_j + \frac{1}{2N_y}\tilde{\psi}_{N_y-1}^{(1)} T_{N_y} \,. \tag{5.41}$$

Now (5.35a) is applied to the Chebyshev coefficients in (5.41) to calculate the derivative $D\hat{\psi}$. Let $\tilde{\psi}_j^D$ be its new coefficients. We find that these new coefficients will not equal $\tilde{\psi}_j^{(1)}$ and the following relation is found between them,

$$\begin{aligned}
\tilde{\psi}_j^D &= \frac{2}{c_j} \sum_{\substack{q=j+1 \\ q+j \text{ odd}}}^{N_y} (c_{q-1}\tilde{\psi}_{q-1}^{(1)} - \tilde{\psi}_{q+1}^{(1)}) + \frac{1}{c_j}\tilde{\psi}_{N_y-1}^{(1)} = \tilde{\psi}_j^{(1)} \,, \qquad q + N_y \ \text{ odd} \,, \\
\tilde{\psi}_j^D &= \frac{2}{c_j} \sum_{\substack{q=j+1 \\ q+j \text{ odd}}}^{N_y} (c_{q-1}\tilde{\psi}_{q-1}^{(1)} - \tilde{\psi}_{q+1}^{(1)}) = \tilde{\psi}_j^{(1)} - \frac{1}{c_j}\tilde{\psi}_{N_y}^{(1)} \,, \qquad q + N_y \ \text{ even} \,.
\end{aligned} \tag{5.42}$$

Thus, we have a method of correcting the coefficients $\tilde{\psi}_j^{(1)}$ so that they represent $D\hat{\psi}$ with the same truncation as $\tilde{\psi}_j$ represent $\hat{\psi}$. A similar correction can be derived for the coefficients $\tilde{\psi}_j^{(2)}$ of the second derivative. After some algebra we find,

$$\begin{aligned}
\tilde{\psi}_j^{D^2} &= \tilde{\psi}_j^{(2)} - \frac{1}{c_j}\left(1 + \frac{(N_y-1)^2 - j^2}{4N_y}\right)\tilde{\psi}_{N_y-1}^{(2)} \,, \qquad j + N_y \ \text{ odd} \,, \\
\tilde{\psi}_j^{D^2} &= \tilde{\psi}_j^{(2)} - \frac{1}{c_j}\tilde{\psi}_{N_y}^{(2)} \,, \qquad\qquad\qquad\qquad\qquad\quad j + N_y \ \text{ even} \,,
\end{aligned} \tag{5.43}$$

where $\tilde{\psi}_j^{D^2}$ are the corrected Chebyshev coefficients for $D^2\hat{\psi}$.

When the horizontal components of the velocity and vorticity are found using the relations (5.15a) to (5.16b), we need $\hat{\phi}$, $D\hat{v}$ and $D\hat{\omega}$. The above corrections are therefore needed in order for the velocity and vorticity fields to exactly satisfy the incompressibility constraint (4.2). Note that an error in the highest Chebyshev coefficients will, by the above correction scheme, affect all other coefficients of the first and second derivative. This is exactly what was supposed to be avoided by the integration method.

The CTM and CIM methods are equally efficient and give the same results with the exception of a few, very rare, cases. We have found that numerical instabilities may

occur when the wall-normal resolution is very low and the velocity and vorticity fields are not divergence free. We have also found that it in those cases is enough to make the vorticity divergence free to stabilize the calculations. With the integration correction or the CTM method, both the velocity and vorticity are completely divergence free. However, for one channel flow case so far, and more frequently in the boundary layer, a numerical instability occurs with the integration correction but not without.

Fortunately the instability causes the calculation to blow up in a few time steps and before that the results are the same as for a stable version of the code. With sufficient wall-normal resolution (which is required anyhow) and without the integration correction the boundary layer code has been found completely reliable. The CTM method is, however, to prefer.

6 Implementation

In implementing the flow solver significant effort has been put into portability, flexibility and computational efficiency. The main language is standard Fortran 77 with fixed source format. However, to ease readability, some Fortran 90 features have also been used making Fortran 90 compilers a requirement. In the main programs no dynamic memory allocation has been used, *i.e.* all the sizes of arrays are known at compile time. The demands on the data structure have forced an encapsulation of the access to the main storage which requires some attention. Also the vectorization and the need to process suitably large chunks of data at a time adds complexity in exchange for execution speed.

The complete **Simson** package is written in single precision, *i.e.* with plain `real` and `complex` declarations. However, in most cases there is a need to run the code in double precision, *i.e.*, with at least 10–12 digit precision. For most compilers there are options that automatically changes single precision statements to double precision like the `-r8` flag for the Intel compiler.

6.1 Program structure of **bla**

The program **bla** has been divided into subroutines each with specific tasks. The main program steps the time and calculates the adaptive time step. The subroutines **nonlinbl** and **linearbl** carry out the main part of the algorithm aided by smaller subroutines for integration, equation solving *etc.* The FFTs are taken from **cvecfft_acc** which was developed specifically for the simulation codes but is an independent package of vectorizable Fourier and Chebyshev transforms.

Step 1 reads input files, initializes the FFTs and calculates the partial right hand sides needed to start the time stepping loop and computes the base flow. In the main time stepping loop the data needs to be stepped through twice. First slicing in xz-planes to calculate the FFTs and the pointwise product for non-linear terms, step 2, and second in xy-planes to calculate the normal Chebyshev transforms and solve the system of equations for the new velocities and vorticities, step 3. Step 4 stores the final velocity field. The most important subroutines are briefly described in the following sections.

6.1.1 Step 1, Initialization

In step 1 the flow solver is initialized:

- **mpi_*** subroutines initialize the MPI parallelization and are only called if compiled with MPI.
- **OMP_*** subroutines initialize the OpenMP parallelization and are only called if compiled with OpenMP.
- **ppar** prints the contents of the parameter file **par.f** to standard output as a check of which size of problem the image is compiled for.

- **rparambl** reads the file **bla.i** that contains the runtime parameters, in particular the input and output filenames and the final time to which the simulation is to be run, cf. section 7.4.
- **rdiscbl** reads the resolution, the computational box size and a few parameters defining the flow from the velocity file. The velocities are then read from the file and put into the main storage positions 1–3. If the resolution of the image and the file do not correspond, this is printed on standard output and the program stops execution. The check can be disabled by the **varsiz** flag in the **bla.i** file in which case the field is extended by zero-padding or truncated to fit the image resolution.
- **rescale** rescales all input data that requires it (boundary layer simulations only) from boundary layer scaling to the channel flow scaling which is used internally, see appendix B.
- **preprbl** calculates wavenumbers and collocation points and initializes the FFTs.
- **rbla** reads similarity solution from **fsc.dat**. Used by, for example, the fringe forcing.
- **fshift** computes a Galilei transformation which can be used to increase the maximum stable time step by shifting the streamwise and/or spanwise velocity by a constant.
- **rwavebl** reads the profile of forcing waves to be introduced in the fringe region.
- **rwavesbl** a more general wave forcing than **rwavebl** introduced in the fringe region.
- **rstreakbl** reads streak data to be introduced in the fringe region.
- **bflow** gets from file or computes the base flow used by the fringe forcing and some free-stream boundary conditions.
- **blfou** computes the streamwise Fourier transform of the base flow.
- **getdt** calculates the initial time step to get a CFL number equal to the cflmax value. The subroutine is only used if the time stepping is adaptive.
- **rparamles** reads parameters for LES.
- **init_filter** initializes LES filters.
- **prhs** calculates the initial partial right hand sides $\hat{p}_\phi$, $\hat{p}_\omega$, $\hat{p}_{01}$, $\hat{p}_{03}$ and places the first two in positions 6 and 7 of the main storage. The streamwise and spanwise vorticities are also calculated and put into positions 4 and 5 of the main storage.

Some initial parameter values for the time stepping mechanism are prepared in the main program and output files are opened.

6.1.2 Step 2, Computations in physical space

Step 2 includes the first half of the time integration loop:

- **wplbl_serial** / **wplbl** writes data to 2D plane files.
- **blshift** shifts the base flow and boundary conditions to be aligned with the computational domain when a Galilean transform is used, *i.e.* if the lower wall is moving.
- **gtrip** generates a random force flow trip.
- **boxxys** computes the spanwise and time averaged statistics for one xz-box.
- **wxys** writes statistics to file.
- **wdiscbl** writes intermediate velocity field to file if required.
- **ffun** computes the fringe forcing.

- **nonlinbl** calculates H_i as pointwise products in physical space and stores them in position 1 to 3 of the main storage. It also computes the volume forcing and adds it to H_i. As the main storage is in Fourier–physical space, cf. section 6.2.2, the velocities and vorticities must be transformed back to physical space before the product can be formed. Likewise the products H_i must be transformed to Fourier space before storing them. The velocity rms amplitudes are computed in Fourier–physical space. The maximum CFL number and the extrema of the velocities are calculated from the velocities in physical space.
- **wampbl** the amplitudes are written to file.
- **wextbl** the extremum amplitudes are written to file.
- **wcflbl** updates the CFL time-step restriction which is used if adaptive time stepping is enabled.
- **cwallbc** sets the boundary condition at the wall.

6.1.3 Step 3, Computations in Fourier–Chebyshev space

Step 3 includes the second half of the time integration loop:

- **linearbl** transforms the non-linear products computed in step 2 into Chebyshev space and constructs the complete right hand sides for the evolution equations. The Chebyshev-tau or Chebyshev-integration method is used to solve for the evolution variables from a set of tridiagonal equations. The chosen boundary conditions are applied. All velocities and vorticities are constructed and partial right hand sides are computed for the next time step. Finally the velocities and vorticities are transformed back to physical space in the y-direction. The velocities are stored into positions 1 to 3, the streamwise and spanwise vorticity into 4 and 5 and the partial right hand sides into 6 and 7 of the main storage.
- **nonlinp** computes the terms $H_{1,1} + H_{3,3}$ and H_2 and stores them in position 4 and 5. The energy E is calculated and stored in position 8.
- **linearp** computes the linear and non-linear parts of the boundary conditions and the sum $H_{1,1} + H_{2,2} + H_{3,3}$. The equation for the pressure is solved and the streamwise and spanwise vorticity are recalculated. Pressure is stored in position 8 of the main storage. Note that the pressure is only computed if the **pressure** parameter is equal to one in the **par.f** file.

For selected times the 3D velocity data is written to file. Time is incremented and execution is continued with the next time step from step 2 if the final time **tmax** or any other break criterion (maximum integration steps, maximum wall-clock time *etc.*) is not reached.

6.1.4 Step 4, Output

Step 4 writes the final output to files after the time integration has finished:

- **wxys** the final values of xy-statistics are written to file.
- **wdiscbl** handles the output of a velocity field to an external file.
- **wdiscp** the pressure is written to an external file if a valid pressure solution is available and **pressure** is set to one.
- **wplbl** the planes are written to file.

6.2 Data structure

As the size of a problem is explicitly compiled into the program, the memory allocation is for the most part static. Some effort has been put into minimizing not only the three dimensional storage but also the two dimensional arrays.

6.2.1 Complex numbers and FFTs

Most of the algorithm works with quantities in Fourier space. They are in general in complex form which requires storage of both real and imaginary parts. Although Fortran has the capability of automatically handling complex numbers most compilers produce inefficient code for this, especially for mixed real and complex expressions. Moreover Fortran stores complex numbers with alternating real and imaginary parts, which causes a severe performance loss for vector fetches on certain computers as the stride will be even. To circumvent this, it was decided to store all complex quantities in double arrays, one for real and one for imaginary parts. As the algorithm neither includes general complex-complex multiplications nor divisions this did not add very much code.

6.2.2 Main storage, boxes, drawers, and planes

The size of the three-dimensional main storage has already been introduced in section 3.2.4 on page 13. It is

$$(7 + \mathbf{pressure} + 3 \times \mathbf{scalar}) \times \mathbf{nx} \times \mathbf{ny} \times \mathbf{nz}$$

double precision numbers (*i.e.* 8 bytes), multiply by a factor of 3/2 for dealiasing in the y-direction and by 1/2 if z-symmetry is used. If MPI is used the main 3D storage is reduced by the factor **nproc**.

The access to the main storage can be rearranged to optimize the performance on, for example, vector machines. If the three dimensional storage is divided into xz- and xy-planes the largest common element between these is a single vector in the x-direction, a *pencil* containing **nx** words. In order to increase this number, planes are combined into a *box* consisting of an integer number of adjacent planes *e.g.*, an xy-box holds **mbz** xy-planes and an xz-box holds **mby** xz-planes. The intersection between an xy- and an xz-box then holds **mby**×**mbz** pencils, which is called a *drawer*. Most subroutines are made to handle a box rather than a plane at a time, with the additional advantage that the vector length increases by a factor of **mby** or **mbz**. Note that currently **mby** and **mbz** are set to one and cannot be changed.

The variables in the main storage are in Fourier–physical format, *i.e.*, the axes are α, physical y and β, except for the partial right hand sides $\hat{p}_v$ and $\hat{p}_\omega$, which are stored in Fourier–Chebyshev space.

The main storage is accessed box-wise by the routines **getxy**, **putxy**, **getxz** and **putxz**; in the MPI parallel version the routines **getpxz** and **putpxz** replace the latter two.

The two dimensional storage for step 2 (with storage for calculating statistics) is

$$(16 + 4 \times \mathbf{scalar} + 3 \times \mathbf{pressure}) \times \mathbf{nx} \times \mathbf{nz} \times \mathbf{mby} \times \mathbf{nthread}$$

double precision numbers; multiply by a factor 3/2 each for dealiasing in the x and z-directions and by 1/2 for z-symmetry. For step 3 storage is

$$(23.5 + 11.5 \times \mathbf{scalar}) \times \mathbf{nx} \times \mathbf{ny} \times \mathbf{mbz} \times \mathbf{nthread}$$

double precision numbers; multiply by 3/2 for dealiasing in the y-direction. The storage for step 2 and step 3 overlaps so that the total two-dimensional storage is equal to the maximum of the requirement for step 2 and step 3.

6.2.3 Naming conventions

Some naming conventions have been used for variable names. Greek letters have been replaced by abbreviations. In the case a variable is complex it has been replaced by two variables with the last letters `r` and `i`, for the real and imaginary parts respectively. An example of this is `pomyr` which is the real part of the array $\hat{p}^n_\omega$. Note

that the superscripts n *etc.* and the hat symbol are generally left out. When needed for distinction they are replaced by suffixes, *e.g.* a^{n+1} becomes `anp1`. The component indices 1, 2 and 3 in, *e.g.*, H_1 are usually found as the last index of the array. Instead numbers in the array names are used to distinguish between the same variable when represented by two different arrays in step 2 and step 3. Normal derivatives are denoted by prefixes `d` and `d2`. Sometimes a `b` is used for `box`, *e.g.*, `bbeta` is the wavenumber beta vector expanded to correspond to other box sized arrays.

Note that wall-normal resolution N_y used in this report corresponds to **ny−1** in the code.

6.2.4 The oddball wavenumbers

When transforming data between Fourier and physical space, one wavenumber, called the Nyquist wavenumber $N/2$ or the oddball, needs special attention. This can be understood if one considers the following scenario. If $u(x)$ is a real function sampled at N grid points, two modes in the finite Fourier series expansion $\hat{u}_k$ (in the positive k half-plane) will be real, $\hat{u}_0$ and $\hat{u}_{N/2}$ and the rest will be complex, $\hat{u}_{1,\ldots,N/2-1}$, so that the information (N real numbers) is conserved. Consider calculating the first derivative of $u(x)$, denoted by $f(x)$, which of course will be real. Transforming again the real function $f(x)$ to Fourier space will give, again, two real Fourier coefficients $\hat{f}_0$ and $\hat{f}_{N/2}$. If one takes the derivative of $u(x)$ in Fourier space, $\hat{f}_n = ik_n\hat{u}_n$ then $\hat{f}_{N/2}$ will be purely imaginary. This is however not possible if f is supposed to be real. Therefore, one sets $\hat{f}_{N/2} = 0$ regardless of $\hat{u}_{N/2}$. The consequence is that oscillation in the numerical representation of a function at the Nyquist frequency should be removed from the data to be consistent with computed spectral derivatives. If the oddball modes are not filtered, their effect can propagate towards lower frequencies and thus contaminate the accuracy of a whole simulation.

The above consideration directly transfers to multiple dimensions, see figure 6.1 for a two-dimensional example. The oddball mode is always associated with the mode $N/2$ in even directions of N. In consequence there is less information in spectral representation than in physical space. Note that for N odd there is no oddball mode and, consequently, a direct correspondence between physical and spectral space exists.

6.3 Parallelization

The code supports both parallelization based on shared memory using OpenMP and distributed memory using MPI. Both variants give good speed-up on a variety of computers. For machines supporting both MPI and OpenMP it is recommended to compare both parallelizations, however, OpenMP is expected to be slightly more efficient.

The parallelization (both OpenMP and MPI) is designed in such a way that exactly the same operations in the same order are performed on the data. This implies that – provided that the compiler generates similar code – results of a serial run are binary the same as the ones obtained from a parallel calculation. This applies in particular to the velocity and pressure fields, statistics files, two-point correlations and saved planes.

6.3.1 OpenMP

The OpenMP parallelization is implemented in a straight-forward way around the loops calling **nonlinbl**, **linearbl**, **nonlinp** and **linearp**. Additionally, one loop in **ffun** and the computation of statistics in **boxxys** is also parallelized. These parallel parts are the ones corresponding to the majority of the work performed during time integration. Note that most of the two-dimensional working arrays have an additional dimension corresponding to the number of threads used. Thereby it is assured that

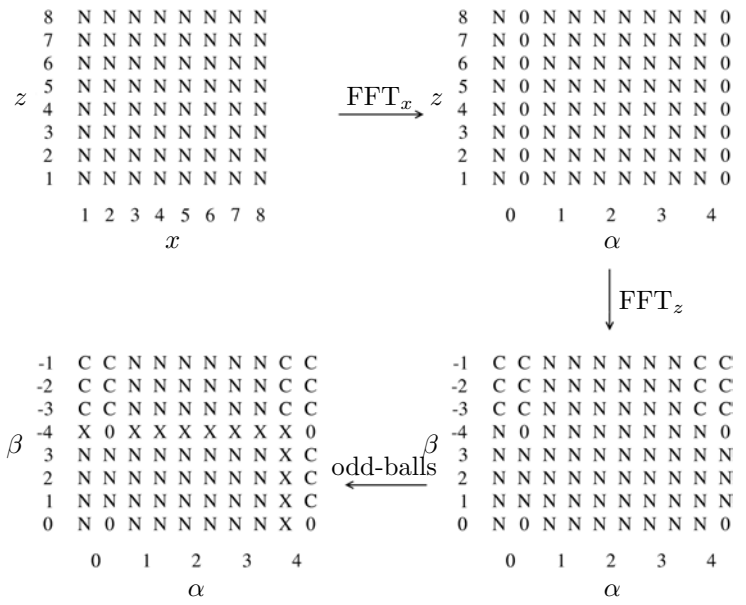


Figure 6.1: Schematic figure of the 2D transformation from physical to Fourier space and the oddball. A domain with $N = 8$ in two dimensions is first transformed from physical space with a total of 64 numbers on a $N \times N = 8 \times 8 = 64$ grid (`N` indicates an arbitrary number) in the x- and z-directions. In spectral space, the information is conserved, *i.e.* 64 independent numbers on a $N \times (N + 2) = 8 \times 10 = 80$ grid (`C` indicates redundant data due to complex conjugate and 0 means zero). To make sure that the resulting field in spectral space also corresponds to a realizable field, the oddball wavenumbers have to be set to zero (marked with `X`). Thus, we are left with 49 independent numbers.

these work arrays are thread-private. Moreover, the first threaded use of these work arrays is done by the correct thread, such that for certain processor architectures independent memory busses are assigned.

There is no strict upper limit on the number of OpenMP threads. However, the number of planes in both the wall-normal direction and spanwise direction is given by **nyp** and **nz**, respectively. Therefore these numbers can be considered natural upper limits for the number of threads that could still lead to a performance increase.

6.3.2 MPI

In the present MPI parallelization, the main storage **ur**, **ui** of the code is distributed among the processors along the spanwise z direction. Communication is therefore necessary to evaluate the nonlinear terms in physical space. Hence, at the beginning of the subroutines **nonlinbl** and **nonlinp**, a global data transpose has to be performed. This is accomplished by accessing the main storage via the functions **getpxz** and **putpxz**, which internally take care of collecting the relevant data of a xz-plane from all processors. Note that **getpxz** and **putpxz** need to be called from all processors at the same time, and that these functions always operate on **nproc** consecutive wall-parallel planes.

The global communication in **getpxz** and **putpxz** is implemented in two different ways. On the one hand, the standard `MPI_ALLTOALL` library function can be used. However, to fit with the data structure of the code, user-defined data types involving `MPI_TYPE_STRUCT` and `MPI_UB` have to be used. On the other hand, a hand-written alternate version is also included, which is based on explicit point-to-point send and receive statements. Both version of **getpxz** and **putpxz** essentially perform similarly and exhibit approximately the same memory requirement. The hand-written version

is chosen as the standard for the present release. Further details on the implementation of the MPI communication can be found in Alvelius & Skote (2000). Note that this reference is based on an older version of the code, not supporting all the features of the new code release.

The main storage is distributed among the processors. Therefore, the memory footprint is also reduced by employing multiple MPI processes. However, since each process needs to have a certain amount of two- and one-dimensional work arrays, the decrease in memory requirement is not linear. Note that all the large permanent storage is allocated/declared in the main Fortran function **bla**, which allows a simple estimate on the memory required by each process.

The distribution of the data along the spanwise direction imposes the natural upper limit of the possible processors to be **nz**. Moreover, **nz** needs to be evenly divisible by the number of processors, **nproc**. The code has been shown to scale essentially linearly up to **nz**/**2** processors on various distributed-memory architectures.

7 File formats

This section contains the formats of the most important input and output files that are used by the **Simson** programs.

7.1 Compile time parameter file **par.f**

The parameter file **par.f** needs to be adjusted to fit the numerical experiment at hand before starting to compile **Simson**. Below follows a short description of these parameters.

The number of spectral modes in each direction is set by the parameters **nx**, **ny** and **nz**. The following restrictions apply: **nx** and **ny−1** must be even and factorable by 2, 3 and 5, **nz** must be factorable by 2, 3, 5 and at least 2. Note that **ny** is the number of Chebyshev polynomials and thus is equal to $N_y + 1$ used in chapter 4.

Dealiasing, *i.e.* padding to remove aliasing errors, can be switched on (1) or off (0) independently for each direction by the flags **nfxd**, **nfyd** and **nfzd**. If dealiasing in the respective direction is used **nx**, **ny−1** must be divisible by 4, and **nz** must be divisible by 2.

There is an option to run 2 1/2 dimensional simulations, *i.e.*, simulations of flow in a two dimensional geometry with all three velocity components non-zero, which is sometimes called the infinite swept flow (two dimensional flow is a special case of this). In this case the spanwise parameters should be set to **nz** = 1, **nfzd** = 0 and the limitations on **nz** given above do not apply.

The parameter **nthread** determines the maximum number of OpenMP threads the code should be compiled for. The parameter **nproc** specifies the number of processors that should be used for the MPI parallelization. Note that it is possible to combine OpenMP and MPI parallelization (experimental).

To collect run-time statistical data during a simulation, several parameters are used to define the storage requirements of these data sets. The maximum number of two-dimensional $x - y$ velocity statistics collected is given by **nxys** (at the moment, 96 statistics are implemented, see section 7.12. The number of statistics involving the scalars is given by **nxysth** (maximum 35). Note that these statistics are taken for each scalar separately. The parameter **mcorr** sets the maximum number of spanwise two-point correlations. The maximum number of time series is given by **mser**. The parameter **msamp** is the size of the array storing the time-series during run-time between writing them out to disk and it is therefore required to be at least **ixyss**/**ixys** (see section 7.4).

To allow for simultaneous calculation of velocities and pressure, the parameter **pressure** should be set to 1. The temperature can also be computed as a passive scalar field by setting **scalar** to 1 or higher. By setting **scalar** larger than one, multiple passive scalar fields with individual Prandtl numbers and scalar boundary conditions can be computed simultaneously.

```
.
.
c
c     Number of spectral modes
c
      parameter (nx=128,ny=121,nz=128)
c
c     Number of processors (MPI)
c
      parameter (nproc=1)
c
c     Number of threads (OpenMP)
c
      parameter (nthread=1)
c
c     Statistics
c
      parameter (nxys  =  96)
      parameter (nxysth=  35)
      parameter (mcorr =  30)
      parameter (mser  =  20)
      parameter (msamp =  16)
c
c     Pressure (0/1)
c
      parameter (pressure=1)
c
c     Passive scalars
c
      parameter (scalar=0)
c
c     Dealiasing flags
c
      parameter (nfxd=1,nfyd=0,nfzd=1)
c
c     Number of waves for freestream OS-eigenmodes
c
      parameter (osnf=250)
c
c     Number of points in base flow
c
      parameter (mbla=20001)
.
.
```

Table 7.1: An example of the user specific part of the compile time parameter file **par.f**.

All other parameters in the **par.f** file are computed and should not be changed manually. Note that most subroutines must be recompiled after changing **par.f**. An example of the user relevant part of a **par.f** file is given in table 7.1.

7.2 Runtime parameter file **fsc.i**

The file **fsc.i** is formatted and sequential. Comments can be put after data on lines not containing character input. This file is read when running **fsc** in order to create a similarity solution for boundary layer flows.

1. **m** Power law exponent; *real*.
2. **n** Wall-normal resolution; *integer*.
3. **eps** Convergence criterion; *real*.
4. **ymax** Box height; *real*.
5. **scalar** The number of scalars. If non-zero, items a) and b) are repeated for each passive scalar; *integer*.
   a) **pr** Prandtl number; *real*.
   b) **m1** Scalar exponent; *real*. Note that $\mathbf{m1} = 0$ corresponds to an isothermal boundary condition, whereas $\mathbf{m1} = 0.5$ can be used for a iso-flux boundary condition.

7.3 Runtime parameter file **bls.i**

The file **bls.i** is formatted and sequential. Comments can be put after the data on lines not containing character input. For boundary layer flows all input is non-dimensionalized with the displacement thickness and the free-stream velocity at $x = 0$ and $t = 0$. This file is read when running **bls** to generate initial velocity fields, see also section 3.1.1. Contents line by line:

1. **namnin** Optional input velocity field file name; *character*80*.
   The base flow can be given in the form of an input velocity field file.
2. **namnut** Output velocity file name; *character*80*.
3. **re** The Reynolds number; *real*.
4. **xlb** The length of the computational box; *real*.
   The streamwise extent of the box must for spatially developing flows include the length of the fringe region, which is typically set to 30–100 displacement thicknesses.
5. **h2** The height of the computational box; *real*.
   In case of boundary layer flow the vertical extent of the box must include the whole boundary layer. Depending on the choice of free-stream boundary condition, the box may include only the boundary layer or a few times more. The sufficiency of the box height may be investigated through numerical experiments.
6. **zlb** The width of the computational box; *real*.
7. **fltype** Base flow type; *integer*.
   See table 4.1 on page 24 for a complete list of flow types.
   a) If **fltype** $= -1$ or ≥ 7: **rlam** The acceleration exponent of the velocity in the free-stream; *real*.
   b) If **fltype** $= -2$ or ≥ 8: **spanv** The spanwise free-stream velocity; *real*.
   c) If **fltype** ≥ 4: **bstart** The streamwise position of the start of the blending of the base flow; *real*.
   d) If **fltype** ≥ 4: **blength** The length of base flow blending region; *real*.

   The base flow can either be parallel or spatially developing. The parallel base flows currently span Poiseuille, Couette, Blasius, Falkner–Skan and FSC corresponding to **fltype** $= 1, 2, 3, -1$ and -2 respectively. The spatially developing base flow can be either Poiseuille (**fltype** $= 4$), Couette (**fltype** $= 5$), Blasius (**fltype** $= 6$), Falkner–Skan (**fltype** $= 7$), Falkner–Skan–Cooke (**fltype** $= 8$) or temporal flow with fringe region (**fltype** $= 9$). For the three latter cases the acceleration exponent **rlam** for the streamwise free-stream velocity must be given (*i.e.* m in $U = Cx^m$). For Falkner–Skan–Cooke (swept wedge) flow the spanwise velocity **spanv** in the free-stream must be specified. Note that the spanwise direction is parallel to the leading edge of the wedge for this case, and that the spanwise free-stream velocity is constant. For spatially developing flows the base flow from the upstream and the downstream end are blended in the fringe region. The start and blending length must be specified. Typically the start is given as a negative number *i.e.*, the distance upstream of the inflow boundary where the blend starts is given (see section 4.2.2).
8. **pert** Flag to generate flow field without base flow. Used for simulations in perturbation mode; *logical*.
9. **ushift** The Galilei shift velocity. Set to 0 for no shift; *real*.
10. **locdi** Flag to generate a localized disturbance; *logical*.
    a) **ditype** The type of disturbance, only useful values 1 to 4; *integer*.
    b) **amp** The amplitude of a localized disturbance; *real*.
    c) **theta** The rotation angle **theta** of the localized disturbance in radians; *real*.

d) **xscale** The streamwise scale of the disturbance; *real*.

e) **xloc0** Origin of the disturbance in x-direction; *real*.

f) **yscale** The wall-normal scale of the disturbance; *real*.

g) **zscale** The spanwise scale of the disturbance; *real*.

h) **ipoly** The wall-normal distribution of the disturbance, only useful values 1 to 4; *integer*.

The **ditype** determines the type of disturbance. See **bls.i** for more information. The localized disturbance example (**ditype** = 1) is governed by the amplitude, the rotation angle, the length and spanwise scale. The rotation angle is the angle by which the spanwise symmetric disturbance is rotated about the y-axis. The x-scale and the z-scale of the disturbance are given to be applied to the disturbance before rotation. The form of the disturbance is given in a coordinate system aligned with the disturbance:

$$\begin{aligned} u' &= 0\,, \\ v &= \frac{\partial \psi}{\partial z}\,, \\ w' &= -\frac{\partial \psi}{\partial y}\,, \\ \psi &= \mathrm{amp}\frac{x'}{x_{\mathrm{sc}}}\frac{z'}{z_{\mathrm{sc}}}p\left(\frac{y}{y_{\mathrm{sc}}}\right)e^{-\left(\frac{x'}{x_{\mathrm{sc}}}\right)^2-\left(\frac{z'}{z_{\mathrm{sc}}}\right)^2}\,, \end{aligned} \tag{7.1}$$

where $p(s)$ is determined by **ipoly**, see **bls.i**. The relation between the disturbance aligned velocities and coordinates (with $'$) and the computational box aligned ones is:

$$\begin{aligned} x &= x'\cos(\theta) + z'\sin(\theta)\,, \\ z &= -x'\sin(\theta) + z'\cos(\theta)\,, \\ u &= w'\sin(\theta)\,, \\ w &= w'\cos(\theta)\,. \end{aligned} \tag{7.2}$$

11. **gaussian** Flag to generate a Gaussian shaped disturbance; *logical*.

    a) **amp** Disturbance amplitude; *real*.

    b) **y0** Wall-normal peak location; *real*.

    c) **yscale1** Wall-normal scaling factor; *real*.

    d) **walfa** Streamwise wavelength; *real*.

    e) **wbeta** Spanwise wavelength; *real*.

12. **waves** Flag to generate a pair of oblique waves; *logical*.

    a) **energy** Energy density of the waves; *real*.

    b) **ystart** The lowest y-value of non-zero wave amplitude; *real*.

    c) **yend** The largest y-value of non-zero wave amplitude; *real*.

    d) **yrise** The switch distance from zero to max wave amplitude; *real*.

    e) **yfall** The highest y-value of non-zero wave amplitude; *real*.

    f) **walfa** Streamwise wave number of the waves; *real*.

    g) **wbeta** Spanwise wave number of the waves; *real*.

13. **os** Flag to use tabulated eigenmodes; *logical*.

14. **specm** Spectral space mode; *logical*.

    a) **amp** Amplitude; *real*.

    b) **y0** Wall-normal peak location; *real*.

    c) **yscale1** Wall-normal scaling factor; *real*.

    d) **nalfa** Streamwise wavenumber; *real*.

e) **nbeta** Spanwise wavenumber; *real*.

15. **pertfromfile** Read perturbation from file; *logical*.

    a) **initcond_u** File name of u-component; *character*80*.

    b) **initcond_v** File name of v-component; *character*80*.

    c) **nalfa** Streamwise wavenumber; *real*.

    d) **nbeta** Spanwise wavenumber; *real*.

16. **noise** Flag to add noise; *logical*.

    a) **ed** The mean energy density of the noise; *real*.

    b) **nxn** The maximum streamwise wavenumber of the noise, should be $\leq$ **nx**/**2**; *integer*.

    c) **nyn** The number of vertical Stokes modes in the noise, should be even, $<$ **ny**$\times$**2**/**3**; *integer*.

    d) **nzn** The maximum spanwise wavenumber of the noise, should be odd, $<$ **nz**; *integer*.

    e) **seed** A random number seed in the range -700000 to -1; *integer*.

    The noise is in the form of Stokes modes, *i.e.*, eigenmodes of the flow operator without the convective term. They fulfill the equation of continuity and the boundary condition of vanishing velocity at the lower and upper boundaries. Although the actual boundary condition may allow a non-zero amplitude at the free-stream boundary the restriction of zero amplitude for the noise doesn't have a large impact in practice.
    If noise is used the mean energy density must be given along with the number of wave numbers to be randomized for each direction. In the wall-normal direction the number of Stokes modes to be randomized is given. The same noise will be generated for the same setting of this seed, if the physical size of the simulation box is unchanged. In particular the resolution can be changed without affecting the noise, as long as the number of grid points is sufficient to resolve the noise modes. This is useful for convergence studies.

7.4 Runtime parameter file **bla.i**

The file **bla.i** is formatted and sequential. Comments can be put after data on lines not containing character input or on separate lines if the line begins with **#**. Contents line by line:

1. **date** Version of the **bla.i** file. The version has to match the version of **bla**. The format of the version is YYYYMMDD; The current version is dated 20070716. *integer*.
2. **namnin** Input velocity file name; *character*80*.
3. **namnut** Output velocity file name; *character*80*.
4. **tmax** The final simulation time; *real*.
5. **maxit** The maximum number of iterations to simulate; *integer*.
6. **cpumax** The maximum CPU time in seconds; *real*.
7. **wallmax** The maximum wall clock time in seconds; *real*.
   The input and output file names and the final time **tmax** determine the scope of the simulation, in addition setting the maximum number of iterations puts a limit on the number of iterations to be taken through the main time step loop. The latter parameter is useful with variable time stepping in a batch environment to ensure that the execution terminates before running out of execution time. If the maximum number of iterations is used before the final time is reached the execution will terminate normally by saving the present velocity field to the output velocity file. Note that the physical time step consists

of three or four iterations. The execution will only stop after completing an integer number of physical time steps. If adaptive time stepping is used the program will adjust the final four time steps so that it reaches exactly the final time. You can also control maximum execution time by giving either the maximum CPU time (**cpumax**) or maximum wall clock time (**wallmax**) for batch jobs. If one of the stop criteria is not to be used, specify a negative number.

8. **write_inter** Write intermediate velocity field whenever statistics are written to disk (*i.e.* every **ixyss** steps, see below); *logical*.

9. **dt** The time step length; *real*.
dt is the length of the time step, if it is set ≤ 0 the adaptive time stepping is used. The time step is regulated to keep the CFL number close to **cflmax**, which is set by the next parameter.

10. **nst** The number of stages in the time discretization; *integer*.
The parameter **nst** selects between the different formulas for the explicit time discretization (3 three stage Runge–Kutta, 4 four stage Runge–Kutta). The 4 stage Runge–Kutta method is about 20% more efficient than the 3 stage version.

11. **cflmaxin** The maximum CFL number; *real*.
The maximum convective stability limit is $\sqrt{3}$ for the three stage Runge–Kutta and $\sqrt{8}$ for the four stage Runge–Kutta. When using a fringe region the time step is also limited by the numerical stability for the damping term, this is $1.75/\mathbf{fmax}$ for the three stage RK and $1.96/\mathbf{fmax}$ for the four stage RK (**fmax** is the max strength of the fringe region, see below). If **dt** is set < 0 then $-\mathbf{dt}$ is used as an additional limit on the variable time step. The parameter **cflmaxin** is a factor indicating with fraction of the maximum stability limit should be used for dynamic time-stepping. A value of about 0.8 is recommended.

12. **xl** The new box length. If lower than the length read from the velocity field, the velocity field length will be used; *real*.

13. **varsiz** Flag to allow read of a velocity field of different size than the code is compiled for; *logical*.
If **varsiz** is set true the program may start from an input field of a different resolution than the program is compiled for. The spectral coefficients are padded with zeroes or truncated to achieve a spectrally accurate interpolation. However, the resolution cannot be reduced in the normal direction as the truncated field in general will not fulfill the equation of continuity and the boundary conditions.

14. **rot** The angular velocity of the coordinate frame around the z-axis. For non-rotating flows it should be set to zero; *real*.

15. **cflux** Constant mass flux; *logical*. Only effective for channel flow (**fltype** $= 1$).
If **cflux** is false:

    a) **retau** Target Reynolds number Re_τ; *real*. The pressure gradient is specified such that the required skin friction is obtained.

16. **pert** Perturbation equations; *logical*. Solve the Navier–Stokes equations in perturbation form (4.36). The base flow should be a converged solution to the Navier–Stokes equations, *i.e.* preferably a steady simulation result. If **pert** is true:

    a) **lin** Linear simulation; *logical*.

17. **ibc** The free-stream boundary condition number; *integer*.
A number of these boundary conditions makes the numerical scheme unstable. Among the stable boundary conditions, the most used for boundary-layer cases are number 101 and 110. A complete list of available conditions are given in table 7.2 on page 63. See also section 4.3 for more information about the different boundary conditions. Note that for channel flow and Couette flow **ibc** $= 0$ should be chosen.

18. Boundary conditions for passive scalars. The following parameters are read **scalar** times.
    a) **tbc** The boundary condition number (from 0 to 3); *integer*.
    b) If **tbc** = 0
        i. **theta0_low** Value of scalar at lower boundary; *real*.
        ii. **theta0_upp** Value of scalar at upper boundary; *real*.
    c) If **tbc** = 1
        i. **dtheta0_low** Derivative of scalar at lower boundary; *real*.
        ii. **theta0_upp** Value of scalar at upper boundary; *real*.
    d) If **tbc** = 2
        i. **theta0_low** Value of scalar at lower boundary; *real*.
        ii. **dtheta0_upp** Derivative of scalar at upper boundary; *real*.
    e) If **tbc** = 3
        i. **dtheta0_low** Derivative of scalar at lower boundary; *real*.
        ii. **dtheta0_upp** Derivative of scalar at upper boundary; *real*.
        iii. **theta0_upp** Value of scalar at upper boundary; *real*.
19. **cim** Flag to use Chebyshev integration method. If false the tau method is used; *logical*. If **cim** is true:
    a) **icorr** Flag to use integration correction; *logical*.
    The combination of using integration correction and boundary conditions other than of Dirichlet type may lead to numerical instability. The flag is normally set false.
20. **gall** Flag to compute and use a Galilei transformation in both streamwise and spanwise direction to increase the maximum stable time step; *logical*.
    Presently only supported for temporal flows. The output (velocity fields, statistics) are corrected for the shifted walls, *i.e.* the shift velocity is added to the velocity components, and the flow field is periodically shifted by the appropriate value.
21. **suction** Flag to use constant suction at lower wall; *logical*. If **suction** is true:
    a) **asbl** Asymptotic suction boundary layer; *logical*. If **asbl** is false:
        i. **vsuc** Suction rate; *real*.
22. **spat** Flag to perform spatial simulation; *logical*.
    If **spat** is false **bla** program performs a temporal simulation. For spatial simulations a number of parameters specifying the fringe region must be given, see section 4.2.2. Items a) through l) are read only if **spat** is true.
    a) **tabfre** Tabulated free-stream velocity; *logical*.
    To use a tabulated free-stream velocity the flag **tabfre** is set true. The format of the free-stream velocity file is given in section 7.15. If **tabfre** is true:
        i. **namfre** Name of file containing free-stream velocity table; *character*80*.
    b) **rbfl** Flag to use a 3D flow field as a base flow; *logical*.
    To use a 3D flow file to define the base flow the flag **rbfl** is set true. The format of the 3D flow file is given in section 7.6. If **rbfl** is true:
        i. **nambfl** Name of file containing a 3D base flow; *character*80*.
    c) **fmax** Maximum strength of the fringe region; *real*.
    d) **fstart** x-position of the start of the fringe region; *real*.
    e) **fend** x-position of the end of the fringe region; *real*.
    f) **frise** The distance from the start of the fringe region to the first point of maximum damping; *real*.

g) **ffall** The distance from the last point of maximum damping to the end of the fringe region; *real*.

h) **ampob** The amplitude of oblique waves forced in the fringe; *real*.
A pair of oblique waves can be generated in the fringe region by setting **ampob** non-zero. The format of the waveform file **wave.dat** is given in section 7.16.

i) **amp2d** The amplitude of two dimensional Tollmien–Schlichting wave forced in the fringe; *real*.

j) **osmod** Flag to use free-stream turbulence modes; *logical*. If **osmod** is true:

i. **osdamp**; *logical*. Whether or not to account for the spatial growth of the amplitudes in the fringe region.

ii. **osamp** The amplitude of free-stream turbulence modes forced in the fringe; *real*.

iii. **osfil** The name of the file containing the free-stream turbulence modes; *real*.

k) **streak** Generate streaks (one or two) in the fringe;*logical*. If **streak** is true:

i. **str_nam1** Input file; *character*80*.
ii. **ampst(1)** Amplitude; *real*.
iii. **tsmoo(1)** Length of smooth turn on; *real*.
iv. **tsmoo(2)** Length of smooth turn off; *real*.
v. **tsmst(1)** Start of smooth turn on; *real*.
vi. **tsmend(1)** Start of smooth turn off; *real*.
vii. **iampst** Initial amplitude of streak; *real*.
viii. **str_nam2** Input file; *character*80*.
ix. **ampst(2)** Amplitude; *real*.
x. **tsmoo(3)** Length of smooth turn on; *real*.
xi. **tsmoo(4)** Length of smooth turn off; *real*.
xii. **tsmst(2)** Start of smooth turn on; *real*.
xiii. **tsmend(2)** Start of smooth turn off; *real*.
xiv. **phist** Relative phase($/\pi$) of streaks; *real*.

l) **waves** Generate waves in the fringe; *logical*. If **waves** is true:

i. **waamp** Amplitude; *real*.
ii. **wamoo** Smoothing; *real*.
iii. **wamst** Start time; *real*.
iv. **waiamp** Initial amplitude; *real*.

If **spat** is false:

a) **cdev** The reference speed for the parallel boundary layer growth; *real*.
For temporal simulations **cdev** must be set to the reference speed of the boundary layer growth, see section 4.2.1.2.

23. **sgs** To read the file **sgs.i**, used to enable LES mode; *logical*.

24. **isfd** To use selective frequency damping (0=off, 1=on); *integer*. If **isfd** is 1:

a) **sfdzero** Start from zero filtered field; *logical*. If true:

i. **namnut_sfd** Initial filtered field; *character*80*.

b) **sfd_delta** Temporal filter width; *real*.

c) **sfd_chi** Temporal relaxation parameter; *real*.

25. **imhd** To enable a magnetic field (MHD) (0=off, 1=on); *integer*. If **imhd** is 1:

a) **b0(1)** Component of the magnetic field in x-direction; *real*.

b) **b0(2)** Component of the magnetic field in y-direction; *real*.

c) **b0(3)** Component of the magnetic field in z-direction; *real*.

d) **mhd_n** Strength of the magnetic field. The direction of the magnetic field **b0(1)**, **b0(2)**, **b0(3)** will be normalized to unity; *real*.

26. **loctyp** to generate a localized volume force disturbance; *integer*.
The parameter **loctyp** can take values from 1 to 8. Various disturbances can be created. See **locf.f** for more information. The different values of **loctyp** each requires a distinct number of parameters in the file **bla.i**, see **rparambl.f** for more information. As an example, a localized volume force disturbance to generate wave packets is created by setting **loctyp** to 1.

a) **ampx** Max amplitude of the localized volume force disturbance in x-direction; *real*.

b) **ampy** Max amplitude of the localized volume force disturbance in y-direction; *real*.

c) **ampz** Max amplitude of the localized volume force disturbance in z-direction; *real*.

d) **xscale** Length scale of the localized volume force disturbance in x-direction; *real*.

e) **xloc0** Origin of the localized volume force disturbance in x-direction; *real*.

f) **yscale** Length scale of the localized volume force disturbance in y-direction; *real*.

g) **zscale** Length scale of the localized volume force disturbance in z-direction; *real*.

h) If **zscale** < 0: **lskew** The obliqueness of waves of the localized volume force disturbance; *real*.

i) **tscale** Time scale of the localized volume force disturbance; *real*.
If **loctyp** is 1, the form of the localized disturbance is:

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} \mathrm{amp}_x \\ \mathrm{amp}_y \\ \mathrm{amp}_z \end{bmatrix} e^{-(y/y_{\mathrm{sc}})^2} g(x,z) f(t)\,, \tag{7.3}$$

where

$$\begin{aligned} z_{\mathrm{sc}} > 0 \quad & g(x,z) = e^{-((x-x_{\mathrm{loc0}})/x_{\mathrm{sc}})^2-(z/z_{\mathrm{sc}})^2}\,, \\ z_{\mathrm{sc}} < 0 \quad & g(x,z) = \cos(2\pi(z-x_{\mathrm{lskew}})/z_{\mathrm{sc}}) e^{-((x-x_{\mathrm{loc0}})/x_{\mathrm{sc}})^2}\,, \end{aligned} \tag{7.4}$$

and

$$\begin{aligned} t_{\mathrm{sc}} > 0 \quad & f(t) = e^{-(t/t_{\mathrm{sc}})^2}\,, \\ t_{\mathrm{sc}} < 0 \quad & f(t) = S(-t/t_{\mathrm{sc}}))\,, \\ t_{\mathrm{sc}} = 0 \quad & f(t) = 1\,, \end{aligned} \tag{7.5}$$

and

$$S(x) = \begin{cases} 0\,, & x \le 0\,, \\ 1/(1+e^{(1/(x-1)+1/x)})\,, & 0 < x < 1\,, \\ 1\,, & x \ge 1\,. \end{cases} \tag{7.6}$$

27. **tripf** Flag to generate a random "sand paper" volume force trip strip running in the spanwise direction; *logical*. If **tripf** is true:

a) **tamps** Maximum steady amplitude of the trip; *real*.

b) **tampt** Maximum time varying amplitude of the trip; *real*.

c) **txsc** Streamwise length scale of the trip; *real*.

d) **tx0** Streamwise origin of the trip; *real*.

e) **tysc** Wall-normal length scale of the trip; *real*.

f) **nzt** Number of Fourier modes in the spanwise direction of the trip; *integer*.

g) **tdt** Time interval between change of the time dependent part of the trip; *real*.

h) **seed** Negative number in the range −700000 to −1 to initialize the random number generator for the trip; *integer*.

The trip force can be used to generate turbulence or noise at lower amplitude levels to test the stability of a boundary layer or flow structure. The trip has a steady amplitude **tamps** and a time dependent amplitude **tampt** which allow both steady and time varying trips to be generated. The volume force has one continuous time derivative and is independent of the time discretization. The random numbers are generated such that if the random number **seed** and other trip parameters are unchanged, the same trip forces are generated. This is true even if the simulation is split into two or more runs. For every run beyond the first the random number generator is run forward to the correct state. The form of the volume force, which is directed normal to the wall, is as follows:

$$F_2 = \exp(\{(x - t_{x0})/t_{\rm xsc}\}^2 - (y/t_{\rm ysc})^2) f(z,t)\,, \tag{7.7}$$

where

$$f(z,t) = t_{\rm amps} g(z) + t_{\rm ampt}((1 - b(t))h^i(z) + b(t)h^{i+1}(z))\,, \tag{7.8}$$

and

$$\begin{aligned} i &= {\rm int}(t/t_{dt})\,, \\ b(t) &= 3p^2 - 2p^3\,, \\ p &= t/t_{dt} - i\,. \end{aligned} \tag{7.9}$$

Here $g(z)$ and $h_i(z)$ are Fourier series of unit amplitude with **nzt** random coefficients. The trip forcing generates noise with a uniform distribution over all frequencies lower than the cutoff frequency corresponding to $2\pi/t_{dt}$.

28. **wbci** Boundary conditions at wall; *integer*.
wbci can be set from −2 to 2. If **wbci** is not equal to zero, additional parameters must be provided. See **rparambl.f**, **rparamwallrough.f** and **cwallbc.f**. The example below is for **wbci** set to 1.

a) **amp** Max amplitude of the localized blowing/suction; *real*.

b) **damp** Damp amplitude. No effect if less than one; *real*.

c) **xstart** Start position of disturbance; *real*.

d) **xend** End position of disturbance; *real*.

e) **xrise** Rise length of disturbance; *real*.

f) **xfall** Fall length of disturbance; *real*.

g) **zbet** Spanwise variation; *real*.

h) **tomeg** Time variation; *real*.

The form of this boundary condition is

$$v|_{y=0} = {\rm amp} f(x) \cos(z_{\rm bet} z) \sin(t_{\rm omeg} t)\,, \tag{7.10}$$

where

$$f(x) = S\left(\frac{x - x_{\rm start}}{x_{\rm rise}}\right) - S\left(\frac{x - x_{\rm end}}{x_{\rm fall} + 1}\right), \tag{7.11}$$

and $S(x)$ is given by equation (7.6).
If **wbci** is set to −1 the wall-roughness model is turned on, see section 4.3.5. Then, an extra input file containing the relevant parameters has to be supplied:

a) **wallroughinput** Name of input file for wall roughness; *character*80*.

Details about the required parameters for the roughness model can be found in **rparamwallrough.f** and in section 7.18.
If **wbci** is set to −2 a jet profile is imposed, see section 4.3.6 for more details.

29. **icfl** Number of time iterations between calculation of the CFL number; *integer*. If the CFL number is computed each iteration this adds a few percent to the execution time, but since it is used to regulate the time step it should not be computed too sparsely, preferably every complete time step, *i.e.* **icfl** = **nst**.

30. **iamp** Number of time iterations between calculation of rms amplitudes; *integer*. If **iamp** > 0 items a) and b) are read.

    a) **namamp** Output file for rms amplitudes; *character*80*.

    b) **fileurms** Write u-rms velocities; *logical*.

    As for the CFL number continuous calculation of the amplitude costs a number of percent in execution speed. If **iamp** $= 0$ no amplitudes will be calculated and no amplitude file will be written. To get the correct time accuracy **iamp** should be an integer multiple of **nst**.

31. **longli** Flag to generate amplitude for each horizontal plane (y-value). Applies both to rms amplitudes and wave component amplitudes.
    If **longli** is set true the program will produce y-dependent statistics and write these to the amplitude files, both for the global statistics and statistics by wavenumber. The statistics files can become quite large if the flag is set true.

32. **iext** Number of time iterations between calculation of extremum amplitudes; *integer*.
    To get the correct time accuracy **iext** should be an integer multiple of **nst**. If **iext** $= 0$ no extremum amplitudes will be calculated. If **iext** > 0 item a) is read.

    a) **namext** Output file for extremum amplitudes; *character*80*.

33. **ixys** Number of time iterations between calculation of xy-statistics; *integer*.
    The statistics can be analyzed with **pxyst**. The statistics generated and the output file format are described in section 7.12. The file is written to every **ixyss** iterations, overwriting older data. To get the correct time accuracy **ixys** should be an integer multiple of **nst**. If **ixys** > 0 items a) through e) are read.

    a) **namxys** Output file for xy-statistics; *character*80*.

    b) **ixyss** Number of time iterations between saving of xy-statistics data to file; *integer*. If **write_inter** is true, then also an intermediate velocity field is saved to the file **end.uu** (this name can be changed in **rparambl.f**). A crashed simulation can then be safely restarted from the saved intermediate field without corrupting statistics taken during the run, since the statistics and the velocity field are written out at the same time instant.

    c) **txys** Time to start accumulation of xy-statistics; *real*.

    d) **corrf** Flag to save spanwise two-point correlations; *logical*. The statistics generated and the output file format are described in section 7.13. If **corrf** is true:

        i. **corrnam** Output file for two-point correlations; *character*80*.

        ii. **ncorr** Number of positions to save two-point correlations; *integer*.
        The list is then read from file **two-point.dat**; the format of this input file is a simple text file with at least **ncorr** lines specifying on each line the x and y coordinates and the quantity to save. The possible quantities are the velocities, pressure, scalar correlations and some selected correlations of derivatives (see file **boxxys.f** for details).

    e) **serf** Flag to save time histories at given coordinates; *logical*. The statistics generated and the output file format are described in section 7.14. If **serf** is true:

        i. **namser** Output file for time histories; *character*80*.

        ii. **nser** Number of positions to save time histories; *integer*.
        The list is then read from the input file **probe.dat**; the format of this input file is a simple text file with at least **nser** lines specifying

on each line the x, y and z coordinates and an integer representing the quantity to save. The possible quantities are the velocities, pressure, scalar, and the various velocity and scalar derivatives (see file **boxxys.f** for details).

34. **msave** The number of complete velocity fields to be saved at specific times. If **msave** > 0, items a) and b) are repeated for each file; *integer*.

    a) **tsave** The time for which to save a field; *real*.

    b) **nmsave** The name of the velocity file; *character*80*.

    msave is the number of velocity fields to be saved, maximum **nsave** (set as a parameter in **bla.f**). If larger than zero the times and names of the files to be saved must be given. If the adaptive time stepping is used **bla** automatically adjusts the time step to reach exactly the desired times. For fixed time step the save is done at the nearest time.

35. **mwave** The number of wavenumber amplitudes to be saved. If **mwave** > 0, item b) is repeated for each wavenumber; *integer*.

    a) **namwav** The name of the wave amplitude file; *character*80*.

    b) **kx kz** The streamwise wavenumber as multiples of the fundamental wavenumber $2\pi/x_L$, the spanwise wavenumber as multiple of the fundamental wavenumber $2\pi/z_L$; both *integers*.

    mwave sets the number of specific wavenumbers to calculate amplitudes for. For each wave, the x and z wavenumbers must be specified as integers to be multiplied by $2\pi/x_L$ and $2\pi/z_L$ respectively. The wavenumbers are counted in the physical way for positive and negative **kz** and **kx** zero and up, not in the way of the internal storage. The wave amplitudes are calculated for each of the six velocities and vorticities at intervals set by the **iamp** value.

36. **npl** The number of planes to be continuously saved during the simulation. If **npl** > 0, items b) through e) are repeated for each plane; *integer*.

    a) **ipl** The saving interval for planes in number of iteration; *integer*.

    b) **tpl(i,1)** The type of plane to be saved, 1 for xy and 2 for xz; *integer*.

    c) **tpl(i,2)** The variable to be saved, *i.e.* 1 for u, 2 for v, 3 for w; *integer*.

    d) **cpl** The coordinate for which to save the plane; *real*.

    e) **nampl** The name of the file in which to save the planes; *character*80*.

    npl is the number of 2D planes to be saved every **ipl** iterations during the simulation. To get the correct time accuracy **ipl** should be an integer multiple of **nst**. These files can be visualized with **rps**. The format is described in section 7.11 below. The MPI version does currently not support yz-planes.

7.5 Runtime LES parameter file **sgs.i**

The file **sgs.i** is formatted and sequential. Comments can be put after data on lines not containing character input. This file is read when running LES. Contents line by line:

1. **date** Date to show which version of the **sgs.i** file that is needed; the current version is dated 20060909; *integer*.

2. **iles** Type of SGS model used; *integer*.
   If **iles** = 0: no SGS model is used, *i.e.* a DNS is performed.
   If **iles** = 1: the ADM-RT model is used:

    a) **cutoff_inv** Inverse cutoff wavenumber of the primary filter; *real*.

    b) **iord** Order of the high-pass filter; *integer*.

ibc	Free-stream boundary condition types
0	$u = v = w = 0$
1	$Du = Dv = Dw = 0$
2	$D^2u = D^2v = D^2w = 0$
3	$D^3u = D^3v = D^3w = 0$
10	$Du + ku = Dv + kv = Dw + kw = 0$
11	$D^2u + kDu = D^2v + kDv = D^2w + kDw = 0$
12	$D^3u + kD^2u = D^3v + kDv = D^3w + kD^2w = 0$
20	$Dv + kv = D^2v + kDv = \omega = 0$
100	$u = U$, $v = V$, $w = W$
101	$Du = DU$, $Dv = DV$, $Dw = DW$
110	$Du + ku = DU + kU$, $Dv + kv = DV + kV$, $Dw + kw = DW + kW$
120	$Dv + kv = DV + kV$, $D^2v + kDv = D^2V + kDV$, $\omega = 0$
130	$u = U$, $Du = DU$, $w = W$
140	$u = U$, $Dv = DV$, $w = W$
150	$u = U$, $Du - vx = 0$, $Dw = 0$

Table 7.2: Free-stream boundary conditions. Here u, v, w are the solution velocities and U, V, W are the base flow velocities. The velocity derivative normal to the boundary is indicated by D and k denotes the modulus of the horizontal wavenumber ($k^2 = \alpha^2 + \beta^2$).

c) **chi** Model coefficient χ, $\chi = 0$ activates the dynamic procedure; *real*.

If **iles** = 2: the standard Smagorinsky model is used.

a) **cs** Model coefficient C_S, $C_S = 0$ activates the dynamic procedure; *real*.

b) If **cs** = 0: **ineg** Determines clipping behavior of the dynamic procedure; *integer*. **ineg** = 0, no clipping, **ineg** = 1, clipping to positive C_S, **ineg** = 2, clipping to positive total viscosity.

If **iles** = 3: the high-pass filtered Smagorinsky model is used.

a) **cutoff_inv** Inverse cutoff wavenumber of the primary filter; *real*.

b) **iord** Order of the low-pass filter; *integer*.

c) **cs** Model coefficient C_S, $C_S = 0$ activates the (consistent) dynamic procedure; *real*.

d) If **cs** = 0: **ineg** Determines clipping behavior of the dynamic procedure; *integer*. **ineg** = 0, no clipping, **ineg** = 1, clipping to positive C_S, **ineg** = 2, clipping to positive total viscosity.

7.6 Velocity file

Format of a 3D velocity file. The format is used for any 3D input or output from **bls**, **bla** and **cmp**. The file is unformatted, sequential. This file can be visualized by **rit**.

Record 1: Reynolds number; *real*, Poiseuille (true) or Couette flow (false); *logical*, x_L; *real*, z_L; *real*, the time for this field; *real*, the length by which the box has been shifted to the right since time zero; *real*, Prandtl number; *real*, m_1 power law exponent; *real*. The last two parameters are repeated for each scalar field.

Record 2: Number of spectral modes in the x-direction; *integer*, the number of points in the physical y-direction; *integer*, the number of spectral modes in the z-direction reduced for symmetry; *integer*, 0/1 no z-symmetry/z-symmetry; *integer*.

Record 3: Flow type **fltype**; *integer*, displacement thickness expressed in half box heights **dstar**; *real*.

Record 4: If **fltype** ≥ 4: start of blending region **bstart**, length of blending region **blength**, if **fltype** ≥ 7: acceleration exponent of streamwise free-stream velocity **rlam**, spanwise free-stream velocity **spanv**. If **fltype** $= -1$: **rlam** and if **fltype** $= -2$ **rlam** and **spanv**. For other values of **fltype** this record is omitted.

Record 5: The u, v, w-velocities in Fourier x, z and physical y space. One record contains **nx/2** complex coefficients in normal Fortran format. The records are stored in `y`, `z`, `i` order with `y` varying the fastest and `i` the slowest. The number of points in the y-direction is **nyp** and the number of points in the z-direction is **nzc**. Total number of records **nyp**×**nzc**×**3**.

Record 6 - ... The θ_i scalar distribution in Fourier x, z and physical y space (similar to the velocities). The number of records depends on the number of scalars at compile time.

7.7 Pressure file

Format of a 3D pressure file. The format is the same as for the velocity file, except the last record which contains only the pressure. This file can be visualized by **rit**.

7.8 Amplitude file

Formatted, sequential. To be analyzed with the tools **pamp1** and **pamp2**. The rms-levels are averages over the physical box. For each time three records are saved:

1: Time; *real*, $u_{\rm rms}$; *real*, $v_{\rm rms}$; *real*, $w_{\rm rms}$; *real*, $\chi_{\rm rms}$; *real*, $\omega_{\rm rms}$; *real*, $\vartheta_{\rm rms}$; *real*, ω^2/k^2; *real*, $DUuv$; *real*, energy for wavenumber zero; *real*, h^+, *i.e.* the box half-height in wall units; *real*.

If **longli** is true then for each time the above is followed by statistics by y-plane in descending y-coordinate order as follows:

2: Mean squared streamwise velocity without Blasius base flow; *real*, mean squared normal velocity; *real*, mean squared spanwise velocity; *real*, mean squared streamwise vorticity; *real*, mean squared normal vorticity; *real*, mean squared spanwise vorticity without Blasius base flow; *real*, mean squared vorticity squared over wavenumber square average, no (0,0); *real*, Reynolds stress average; *real*, mean streamwise disturbance velocity squared; *real*, mean spanwise disturbance velocity squared; *real*.

7.9 Wave amplitude file

Formatted, sequential. To be analyzed with the tools **pampw** and **pampw2**. The data in this file is in internal scaling. For each time the following data are saved:

1: Time; *real*, number of waves saved; *integer*, number of points in the y-direction; *integer*, Reynolds number; *real*, fundamental wavenumber in the x-direction; *real*, fundamental wavenumber in the z-direction; *real*, flag **longli**.

2: The wavenumber α as multiples of the fundamental $2\pi/x_L$; *integer*, the wavenumber β as multiple of the fundamental $2\pi/z_L$; *integer*, $u_{\rm rms}$; *real*, $v_{\rm rms}$; *real*, $w_{\rm rms}$; *real*, $\omega_{\rm rms}$; *real*.
Item 2 is repeated for each wave.

If **longli** is true then for each time the above is followed by statistics by y-plane in descending y-coordinate order as follows:

3: If the wavenumber is zero: $\hat{u}$ for each y-plane (with the imaginary part zero), otherwise $\hat{v}$ for each y-plane; *complex*.

4: If the wavenumber is zero: $\hat{w}$ for each y-plane (with the imaginary part zero), otherwise $\hat{\omega}$ for each y-plane; *complex*.

Items 3 and 4 are repeated for each wave.

7.10 Extremum file

Formatted, sequential. For each time the following data are saved:

1: Time; *real*.

2: Min $u - U_{\text{laminar}}$; *real*, x-coordinate for this minimum; *real*.

3: y-coordinate; *real*, z-coordinate; *real*.

4 - 5: Same for min v.

6 - 7: Same for min w.

8 - 9: Same for min χ.

10 - 11: Same for min ω.

12 - 13: Same for min ϑ.

14 - 15: Same for min $\vartheta - \vartheta_{\text{laminar}}$.

16 - 29: Same as 2 through 15 but for maximum.

7.11 Plane velocity file

Unformatted, sequential. This file can be visualized by **rps**.

Record 1: Reynolds number; *real*, **.false.** (this is to be backward compatible with channel flow files); *logical*, x_L; *real*, z_L; *real*, the time for this field; *real*, the length by which the box has been shifted to the right since time zero; *real*.

Record 2: Number of spectral modes in the x-direction; *integer*, the number of points in the physical y-direction; *integer*, the number of spectral modes in the z-direction reduced for symmetry; *integer*, 0/1 no z-symmetry/z-symmetry; *integer*.

Record 3: The type of plane, 1 for xy-plane, 2 for xz-plane; *integer*, the variable number, *i.e.*, 1 for u, 2 for v, 3 for w; *integer*, the coordinate of the plane; *real*, flow type **fltype**; *integer*, displacement thickness **dstar** expressed in half box heights; *real*.

Record 4: Time; *real*, the length by which the boxed has been shifted to the right since time zero; *real*.

Record 5: The velocity array in physical space; xy-planes are **nx**×**nyp** with x varying the fastest; xz-planes are **nx**×**nz** for the non-symmetric case and **nx**×**(nz/2+1)** for the symmetric case with x varying the fastest.

Record 4 and 5 are repeated for each time the plane is saved.

7.12 xy-statistics file

Unformatted, sequential. This file can be visualized by **pxyst**.

Record 1: Reynolds number; *real*, **.false.** (this is to be backward compatible with channel flow files); *logical*, x_L; *real*, z_L; *real*, the time for this field; *real*, the length by which the boxed has been shifted to the right since time zero; *real*. *real*, Prandtl number; *real*, power-law exponent for the scalar; *real*. The last two variables are repeated for each scalar.

Record 2: **A**; *character*, **mhd_n**; *real*, **b0**; *real*. Strength and direction of a possible magnetic field (MHD).

Record 3: Number of spectral modes in the x-direction; *integer*, the number of points in the physical y-direction; *integer*, the number of spectral modes in the z-direction reduced for symmetry; *integer*, 0/1 no z-symmetry/z-symmetry; *integer*.

Record 4: Flow type **fltype**; *integer*, displacement thickness **dstar** expressed in half box heights; *real*.

Record 5: If **fltype** ≥ 4: start of blending region **bstart**; *real*, length of blending region **blength**; *real*, acceleration exponent of streamwise free-stream velocity **rlam**; *real*, spanwise free-stream velocity **spanv**; *real*. If **fltype** < 0 acceleration exponent of streamwise free-stream velocity **rlam**; *real*. For other values of **fltype** this record is omitted.

Record 6: Sum of the length of the time steps at which statistics have been sampled **sumw**; *real*, number of statistics calculated **nxys**; *integer*, number of statistics calculated involving the scalars **nxysth**; *integer*, the number of scalars; *integer*. At the time of writing this user guide, **nxys** = 96 velocity statistics and **nxysth** = 35 scalar statistics are implemented.

Record 7 - 6+nxys: Each record contains a **nx**×**nyp** plane of statistics with the x-index varying the fastest. The statistics are averaged over time and the z-direction.

Velocity statistics 96 velocity statistics are implemented and described below. For details of the implementation see file **boxxys.f**.

Record 7 - 12: u, v, w, u^2, v^2, w^2.

Record 13 - 18: ω_1, ω_2, ω_3, ω_1^2, ω_2^2, ω_3^2.

Record 19 - 22: uv, uw, vw.

Record 22 - 24: $u(x)u(x+1)$, $v(x)v(x+1)$, $w(x)w(x+1)$ (*i.e.* one point separation auto correlations, x counted cyclically). Note that these quantities are not implemented anymore, *i.e.* records 22–30 are zero for the time being.

Record 25 - 27: $u(y)u(y+1)$, $v(y)v(y+1)$, $w(y)w(y+1)$.

Record 28 - 30: $u(z)u(z+1)$, $v(z)v(z+1)$, $w(z)w(z+1)$ (z counted cyclically).

Record 31: $R\epsilon_{11} = u_x^2 + u_y^2 + u_z^2$, ϵ_{ij} is the dissipation tensor.

Record 32: $R\epsilon_{22} = v_x^2 + v_y^2 + v_z^2$.

Record 33: $R\epsilon_{33} = w_x^2 + w_y^2 + w_z^2$.

Record 34: $R\epsilon_{12} = u_x v_x + u_y v_y + u_z v_z$.

Record 35: $R\epsilon_{13} = u_x w_x + u_y w_y + u_z w_z$.

Record 36: $R\epsilon_{23} = v_x w_x + v_y w_y + v_z w_z$.

Record 37 - 48: p, p^2, pu, pv, pw, pux, pvy, pwz, puy, pvz, pwx, puz.

Record 49 - 50: Minimum and maximum disturbance u.

Record 51: Square strain rate $S_{ij}S_{ij}$.

Record 52 - 57: Strain-rate tensor S_{ij}.

Record 58: LES model coefficient $C = C_S^2$.

Record 59: SGS dissipation $\tau_{ij}S_{ij}$.

Record 60 - 65: SGS stress tensor τ_{ij}.

Record 66: Eddy viscosity ν_t.

Record 67 - 68: Forward and backward scatter.

Record 69 - 71: Velocity skewness.

Record 72 - 78: Triple velocity correlations.

Record 79 - 80: pvx, pwy.

Record 81 - 83: Velocity flatness.

Record 84 - 85: Pressure skewness and flatness.

Record 86 - 88: $\tau_{ij}u_j$.

Record 89 - 90: Electric potential ϕ, ϕ^2.

Record 91 - 99: j_i, j_i^2, j_1j_2, j_1j_3, j_2j_3.

Record 100 - 102: ϕj_i.

Scalar statistics 35 scalar statistics are implemented and described below. For details of the implementation see file **boxxys.f**. The scalar statistics are repeated for each scalar.

Record 103 - 104: θ, θ^2.

Record 105 - 107: $u_i\theta$.

Record 108 - 110: $u_i\theta^2$.

Record 111 - 113: $\partial\theta/\partial x_i\partial\theta/\partial x_i$ (no summation).

Record 114 - 117: θp correlations.

Record 118 - 123: Velocity^2-θ correlations.

Record 124 - 132: $u_j\partial\theta/\partial x_i$.

Record 133 - 135: θ-velocity dissipation.

Record 136 - 137: Scalar skewness and flatness.

7.13 Two-point correlation file

Unformatted, sequential.

Record 1: Reynolds number (negative); *real*, **.false.** (this is to be backward compatible with channel flow files); *logical*, x_L; *real*, z_L; *real*, the time for this field; *real*, the length by which the boxed has been shifted to the right since time zero; *real*, Prandtl number; *real*, power-law exponent for the scalar; *real*. The last two parameters are repeated **scalar** times. *real*. The last two variables are repeated for each scalar.

Record 2: Number of spectral modes in the x-direction; *integer*, the number of points in the physical y-direction; *integer*, the number of spectral modes in the z-direction reduced for symmetry; *integer*, 0/1 no z-symmetry/z-symmetry; *integer*.

Record 3: Flow type **fltype**; *integer*, displacement thickness expressed in half box heights; *real*.

Record 4: If **fltype** ≥ 4: start of blending region **bstart**; *real*, length of blending region **blength**; *real*. For other values of **fltype** this record is omitted.

Record 5: Sum of the length of the time steps at which statistics have been sampled **sumw**; *real*, the number of scalars; *integer*.

Record 6: **ncorr** Number of two-point correlations; *integer*.

Record 7: Coordinates of the two-point correlation: x and y position; *real*, x and y index; *integer*, x and y position actually used (*i.e.* the coordinates of the grid point where the correlations are evaluated); *real*, quantities for the correlation; *integer* (see **boxxys.f** for a list of possible values).

Record 8: Mean of the two quantities involved, and mean squared of the two quantities; *real*.

Record 9: Two-point correlation data, total $\mathbf{n}\times\mathbf{z}$ *real* numbers.

Record 10: Records 7–9 are repeated **ncorr** times for each two-point correlation.

7.14 Time-series file

Formatted, sequential. Time history of data at given coordinates is appended to an existing file.

appended record: time; *real*. **nser** numbers corresponding to the instantaneous values of the quantities specified in the input file **probe.dat**; *real*.

7.15 Free-stream velocity table file

Formatted, sequential. This file is read if **tabfre** is true in **bla.i** and it specifies the free-stream velocity distribution for boundary layer flows.

1: n number of table entries

2 - (n+1): xtab streamwise coordinate; *real*, **utab** free-stream velocity; *real*.

7.16 Forced wave file **wave.dat**

Formatted, sequential.

1: rew Reynolds number of wave (not used by **bla**); *real*.

2: alfaw the streamwise, **betaw** the spanwise wavenumber of the wave; both *real*.

3: eig the eigenvalue of the wave, the real part of which is used as the angular frequency of the wave; *complex*.

4 - (n+3): n Chebyshev coefficients of the mode shape of the normal velocity, of which the first **nyp** are used. If there are not enough coefficients they are padded by zeros; *complex*.

7.17 Base flow profile file **fsc.dat**

The file **fsc.dat** is unformatted and sequential. It is generated by **fsc** located in the **bls** directory. The basic flow profile is needed for flows that require a non-analytic base flow (*e.g.* Blasius and FSC). The current version of **bla** is not generating the file if it does not exist.

Record 1: Similarity coordinate $\eta_{\rm start}$; *real*, $\eta_{\rm end}$; *real*, number of grid points n; *integer*, power-law exponent m for the velocities; *real*, displacement thickness in units of η; *real*, number of scalars; *integer*.

Record 2 For all scalars if **scalar** > 0: Prandtl number Pr; *real*, power-law exponents m_1; *real* for all scalars (this record is not present for **scalar** $= 0$).

Records 2+scalar - n+1+scalar: The similarity coordinate η, similarity solutions f, f', f'', f''', g, g', g'', θ, θ', θ'' (the last three variables are repeated according to the number of scalars).

7.18 Surface-roughness file

Formatted, sequential.

1. **updat** If true the boundary conditions for the velocity at $y = 0$ due to roughness are re-computed, using the current flow field; *logical*.

a) **every** Update interval, *i.e.* the roughness conditions are re-calculated every **every**th full time step. Note that no update is performed during the first 50 time steps to allow for the flow to adjust to the new boundary conditions. For high roughness elements the code might become unstable; increase **every** in this case; *integer*.

b) **monitor** If true a formatted file is created containing the modified boundary conditions at $(x_{\max}, 0, z_{\max})$ over the iteration number **it**, *i.e.* at the location of highest roughness $h_{\max} = h(x_{\max}, z_{\max})$; *logical*.

i. **monfile** Name of file containing modified boundary condition; *character*.

2. **v_wall** If true the projection method is also applied to the wall-normal velocity component v; *logical*.
3. **taylor4** If true the Taylor series used to project the roughness no–slip conditions onto the plane $y = 0$ is truncated after the 4^{th}–order term; otherwise after the linear term; *logical*.
4. **h_rough** Maximum roughness height; *real*.
5. **hstart** Chordwise location at which the roughness element starts; *real*.
6. **hend** Chordwise location at which the roughness element ends; *real*.
7. **hrise** Chordwise width of the rising roughness flank; *real*.
8. **hfall** Chordwise width of the falling roughness flank; *real*.
9. **zfunc** Spanwise distribution of the roughness function. If 0: Roughness contour constant (sinusoidal) along the span z; *integer*.

a) If **zfunc** is 1: **qq** Number of periods within the span; *integer*.

10. **rghfil** If true a formatted file is generated, containing the roughness contour over the chordwise coordinate x at maximum height $h(x, z_{\max})$; *logical*.

a) **roughfile** Name of roughness contour file; *character*80*.

Bibliography

Alvelius, K. & Skote, M. 2000 The performance of a spectral simulation code for turbulence on parllel computers with distributed memory. Technical Report TRITA-MEK 2000:17, Royal Institute of Techmology, Stockholm.

Bech, K. H., Henningson, D. S. & Henkes, R. A. W. M. 1998 Linear and nonlinear development of localized disturbances in zero and adverse pressure gradient boundary-layers. *Phys. Fluids*, **10**(6), 1405–1418. URL http://link.aip.org/link/?PHF/10/1405/1.

Berlin, S. & Henningson, D. S. 1999 A new nonlinear mechanism for receptivity of free-stream disturbances. *Phys. Fluids*, **11**(12), 3749–3760. URL http://link.aip.org/link/?PHF/11/3749/1.

Berlin, S., Kim, J. & Henningson, D. S. 1998 Control of oblique transition by flow oscillations. Technical Report TRITA-MEK 1998:6, Royal Institute of Techmology, Stockholm.

Berlin, S., Lundbladh, A. & Henningson, D. S. 1994 Spatial simulations of oblique transition. *Phys. Fluids*, **6**, 1949–1951.

Berlin, S., Wiegel, M. & Henningson, D. S. 1999 Numerical and experimental investigation of oblique boundary layer transition. *J. Fluid Mech.*, **393**, 23–57.

Bertolotti, F. P., Herbert, T. & Spalart, P. R. 1992 Linear and nonlinear stability of the Blasius boundary layer. *J. Fluid Mech.*, **242**, 441–474.

Brandt, L., Cossu, C., Chomaz, J.-M., Huerre, P. & Henningson, D. S. 2003 On the convectively unstable nature of optimal streaks in boundary layers. *J. Fluid Mech.*, **485**, 221–242.

Brandt, L. & Henningson, D. S. 2002 Transition of streamwise streaks in zero-pressure-gradient boundary layers. *J. Fluid Mech.*, **472**, 229–262.

Brandt, L., Schlatter, P. & Henningson, D. S. 2004 Transition in boundary layers subject to free-stream turbulence. *J. Fluid Mech.*, **517**, 167–198.

Bruhn, T. 2006 *Large-eddy simulation of zero-pressure gradient turbulent boundary layers*. Master's thesis, KTH Mechanics, Stockholm, Sweden.

Canuto, C., Hussaini, M. Y., Quarteroni, A. & Zang, T. A. 1988 *Spectral Methods in Fluids Dynamics*. Springer.

Chevalier, M. 2004 *Feedback and Adjoint Based Control in Boundary Layer Flows*. Ph.D. thesis, Royal Institute of Technology, Department of Mechanics, Stockholm, Sweden.

Chevalier, M., Hœpffner, J., Åkervik, E. & Henningson, D. S. 2007 Linear feedback control and estimation applied to instabilities in spatially developing boundary layers. *J. Fluid Mech.*, **588**, 163–187.

Chevalier, M., Högberg, M., Berggren, M. & Henningson, D. S. 2002 Linear and nonlinear optimal control in spatial boundary layers. AIAA Paper 2002–2755.

Choudhari, M. & Streett, C. L. 1992 A finite reynolds-number approach for the prediction of boundary-layer receptivity in localized regions. *Phys. Fluids A*, **4**, 2495–2514.

Collins-Sussman, B., Fitzpatrick, B. W. & Pilato, C. M. Version Control with Subversion. http://svnbook.red-bean.com.

Cooke, J. C. 1950 The boundary layer of a class of infinite yawed cylinders. *Proc. Camb. Phil. Soc.*, **46**, 645–648.

Cossu, C., Chevalier, M. & Henningson, D. S. 2007 Optimal secondary energy growth in a plane channel flow. *Phys. Fluids*, **19**(5), 058107. URL http://link.aip.org/link/?PHFLE6/19/058107/1.

Crouch, J. D. 1992 Localized receptivity of boundary layers. *Phys. Fluids A*, **4**, 1408–1414.

Elofsson, P. A. & Lundbladh, A. 1994 Ribbon induced oblique transition in plane poiseulle flow. In Henningson, D. S. (Editor), *Bypass transition - Proceedings from a mini-workshop*, pages 29–41. KTH. TRITA-MEK Technical Report 1994:14, Royal Institute of Technology, Stockholm, Sweden.

Germano, M., Piomelli, U., Moin, P. & Cabot, W. H. 1991 A dynamic subgrid-scale eddy viscosity model. *Phys. Fluids A*, **3**(7), 1760–1765.

Greengard, L. 1991 Spectral integration and two-point boundary value problems. *SIAM J. Numer. Anal.*, **28**, 1071–1080.

Henningson, D. S. 1995 Bypass transition and linear growth mechanisms. In Benzi, R. (Editor), *Advances in turbulence V*, pages 190–204. Kluwer Academic Publishers.

Henningson, D. S., Johansson, A. V. & Lundbladh, A. 1990 On the evolution of localized disturbances in laminar shear flows. In Arnal, D. & Michel, R. (Editors), *Laminar-Turbulent Transition*, pages 279–284. Springer-Verlag.

Henningson, D. S. & Lundbladh, A. 1994 Transition in Falkner–Skan–Cooke flow. *Bull. Am. Phys. Soc.*, **39**, 1930.

Henningson, D. S., Lundbladh, A. & Johansson, A. V. 1993 A mechanism for bypass transition from localized disturbances in wall-bounded shear flows. *J. Fluid Mech.*, **250**, 169–207.

Herbst, A. H. & Henningson, D. S. 2006 Periodic Influence of Periodic Excitation on a Turbulent Separation Bubble. *Flow, Turbulence and Combustion*, **76**(1), 1–21.

Hildings, C. 1997 Simulations of laminar and transitional separation bubbles. Technical report, Department of Mechanics, The Royal Institute of Technology, Stockholm, Sweden.

Hœpffner, J., Chevalier, M., Bewley, T. R. & Henningson, D. S. 2005 State estimation in wall-bounded flow systems, Part 1. Laminar flows. *J. Fluid Mech.*, **534**, 263–294.

Högberg, M. & Henningson, D. S. 1998 Secondary instability of cross-flow vortices in Falkner–Skan–Cooke boundary layers. *J. Fluid Mech.*, **368**, 339–357.

Högberg, M., Chevalier, M., Berggren, M. & Henningson, D. S. 2001 Optimal control of wall bounded flows. Scientific Report FOI-R--0182--SE, Computational Aerodynamics Department, Aeronautics Division, FOI.

Högberg, M., Chevalier, M. & Henningson, D. S. 2003 Linear compensator control of a pointsource induced perturbation in a Falkner–Skan–Cooke boundary layer. *Phys. Fluids*, **15**(8), 2449–2452.

Högberg, M. & Henningson, D. S. 2002 Linear optimal control applied to instabilities in spatially developing boundary layers. *J. Fluid Mech.*, **470**, 151–179.

Högberg, M., Henningson, D. S. & Berggren, M. 2000 Optimal control of bypass transition. In C., D. (Editor), *Advances in turbulence VIII*.

Jeong, J. & Hussain, F. 1995 On the identification of a vortex. *J. Fluid Mech.*, **285**, 69–94.

Kim, J., Moin, P. & Moser, R. 1987 Turbulence statistics in fully developed channel flow. *J. Fluid Mech.*, **177**, 133–166.

Kleiser, L. & Schumann, U. 1980 Treatment of incompressibility and boundary conditions in 3-d numerical spectral simulations of plane channel flows. In Hirschel, E. H. (Editor), *Proc. 3rd GAMM Conf. Numerical Methods in Fluid Mechanics*, pages 165–173. Vieweg. Braunschweig.

Kreiss, G., Lundbladh, A. & Henningson, D. S. 1994 Bounds for threshold amplitudes in subcritical shear flows. *J. Fluid Mech.*, **270**, 175–198.

Levin, O., Chernoray, V., Löfdahl, L. & Henningson, D. S. 2005a A study of the blasius wall jet. *J. Fluid Mech.*, **539**, 313–347.

Levin, O., Davidsson, E. N. & Henningson, D. S. 2005b Transition thresholds in the asymptotic suction boundary layer. *Physics of Fluids*, **17**(11), 114104. URL http://link.aip.org/link/?PHF/17/114104/1.

Levin, O. & Henningson, D. S. 2007 Turbulent spots in the asymptotic suction boundary layer. *J. Fluid Mech.*, **584**, 397–413.

Levin, O., Herbst, A. H. & Henningson, D. S. 2006 Early turbulent evolution of the Blasius wall jet. *J. Turbulence*, **7**(68), 1–17.

Lilly, D. K. 1992 A proposed modification of the Germano subgrid-scale closure method. *Phys. Fluids A*, **4**(3), 633–635.

Lu, Q. & Henningson, D. S. 1990 Subcritical transition in plane Poiseuille flow. *Bull. Am. Phys. Soc.*, **35**, 2288.

Lundbladh, A. 1993 Growth of a localized disturbance in inviscidly stable shear flow. TRITA-MEK 1993:4, Royal Institute of Technology, Stockholm, Sweden.

Lundbladh, A., Berlin, S., Skote, M., Hildings, C., Choi, J., Kim, J. & Henningson, D. S. 1999 An Efficient Spectral Method for Simulations of Incompressible Flow over a Flat Plate. Technical Report TRITA-MEK 1999:11, Department of Mechanics, Royal Institute of Technology, KTH.

Lundbladh, A. & Henningson, D. S. 1993 Numerical simulation of spatial disturbance development in rotating channel flow. FFA-TN 1993-30, Aeronautical Research Institute of Sweden, Bromma.

Lundbladh, A. & Henningson, D. S. 1995 Evaluation of turbulence models from direct numerical simulations of turbulent boundary layers. FFA-TN 1995-09, Aeronautical Research Institute of Sweden, Bromma.

Lundbladh, A., Henningson, D. S. & Johansson, A. V. 1992a An efficient spectral integration method for the solution of the Navier–Stokes equations. FFA-TN 1992-28, Aeronautical Research Institute of Sweden, Bromma.

Lundbladh, A., Henningson, D. S. & Reddy, S. C. 1994a Threshold amplitudes for transition in channel flows. In Hussaini, M. Y., Gatski, T. B. & Jackson, T. L. (Editors), *Transition, Turbulence and Combustion, Volume I*, pages 309–318. Kluwer, Dordrecht.

Lundbladh, A. & Johansson, A. V. 1991 Direct simulation of turbulent spots in plane Couette flow. *J. Fluid Mech.*, **229**, 499–516.

Lundbladh, A., Johansson, A. V. & Henningson, D. S. 1992b Simulation of the breakdown of localized disturbances in boundary layers. Proceedings of the 4th European Turbulence Conference, Delft, The Netherlands.

Lundbladh, A., Schmid, P. J., Berlin, S. & Henningson, D. S. 1994b Simulation of bypass transition in spatially evolving flows. Proceedings of the AGARD Symposium on Application of Direct and Large Eddy Simulation to Transition and Turbulence, AGARD-CP-551.

Malik, M. R., Zang, T. A. & Hussaini, M. Y. 1985 A spectral collocation method for the Navier–Stokes equations. *J. Comp. Phys.*, **61**, 64–88.

Moreau, R. 1998 *Magnetohydrodynamics*. Kluwer.

Nordström, J., Nordin, N. & Henningson, D. S. 1999 The fringe region technique and the Fourier method used in the direct numerical simulation of spatially evolving viscous flows. *SIAM J. Sci. Comp.*, **20**(4), 1365–1393.

Reddy, S. C., Schmid, P. J., Bagget, P. & Henningson, D. S. 1998 On stability of streamwise streaks and transition thresholds in plane channel flows. *J. Fluid Mech.*, **365**, 269–303.

Schlatter, P. 2005 *Large-eddy simulation of transition and turbulence in wall-bounded shear flow*. Ph.D. thesis, ETH Zürich, Switzerland, Diss. ETH No. 16000. Available online from `http://e-collection.ethbib.ethz.ch`.

Schlatter, P., Brandt, L. & Henningson, D. S. 2006 Large-eddy simulation of bypass transition. In *6th European Fluid Mechanics Conference, Stockholm, Sweden*.

Schlatter, P., Stolz, S. & Kleiser, L. 2004 LES of transitional flows using the approximate deconvolution model. *Int. J. Heat Fluid Flow*, **25**(3), 549–558.

Schlichting, H. 1979 *Boundary-Layer Theory*. Springer, seventh edition.

Schmid, P. J. & Henningson, D. S. 1992 A new mechanism for rapid transition involving a pair of oblique waves. *Phys. Fluids A*, **4**, 1986–1989.

Schmid, P. J. & Henningson, D. S. 1993 Nonlinear energy density transfer during oblique transition in plane Poiseuille flow. Technical Report TRITA-MEK 1993:5, Royal Institute of Technology, Stockholm.

Schmid, P. J., Lundbladh, A. & Henningson, D. S. 1994 Spatial evolution of disturbances in plane Poiseuille flow. In Hussaini, M. Y., Gatski, T. B. & Jackson, T. L. (Editors), *Transition, Turbulence and Combustion, Volume I*, pages 287–297. Kluwer, Dordrecht.

Schmid, P. J., Reddy, S. C. & Henningson, D. S. 1996 Transition thresholds in boundary layer and channel flow. In Gavrilakis, S., Machiels, L. & Monkewitz, P. A. (Editors), *Advances in Turbulence VI*, pages 381–384. Kluwer Academic Publishers.

Skote, M., Haritonidis, J. H. & Henningson, D. S. 2002 Varicose instabilities in turbulent boundary layers. *Physics of Fluids*, **14**(7), 2309–2323. URL `http://link.aip.org/link/?PHF/14/2309/1`.

Skote, M., Henkes, R. A. W. M. & Henningson, D. S. 1998 Direct numerical simulation of self-similar turbulent boundary layers in adverse pressure gradients. *Flow, Turbulence and Combustion*, **60**, 47–85.

Skote, M. & Henningson, D. S. 2002 Direct numerical simulation of a separated turbulent boundary layer. *J. Fluid Mech.*, **471**, 107–136.

Smagorinsky, J. 1963 General circulation experiments with the primitive equations. *Mon. Weath. Rev.*, **91**(3), 99–164.

Spalart, P. R. & Yang, K. 1987 Numerical study of ribbon induced transition in blasius flow. *J. Fluid Mech.*, **178**, 345–365.

Stolz, S., Adams, N. A. & Kleiser, L. 2001 An approximate deconvolution model for large-eddy simulation with application to incompressible wall-bounded flows. *Phys. Fluids*, **13**(4), 997–1015.

Stolz, S., Schlatter, P. & Kleiser, L. 2005 High-pass filtered eddy-viscosity models for large-eddy simulations of transitional and turbulent flow. *Phys. Fluids*, **17**, 065103. URL http://link.aip.org/link/?PHFLE6/17/065103/1.

Åkervik, E., Brandt, L., Henningson, D. S., Hœpffner, J., Marxen, O. & Schlatter, P. 2006 Steady solutions of the Navier–Stokes equations by selective frequency damping. *Phys. Fluids*, **18**, 068102. URL http://link.aip.org/link/?PHFLE6/18/068102/1.

A Examples

For each example case in the **examples** directory all the needed parameter files are included and each case is briefly described in this appendix. For some example cases there is also a Bash script **run.sh** that shows how to run the case in detail.

A.1 Temporal channel and Blasius boundary layer flow

The bypass transition from localized disturbances is studied in Henningson *et al.* (1993). To illustrate transitional flows, two examples from that publication are included in **temporal-channel-laminar** and **temporal-blasius** respectively.

For the channel flow case, at $Re = 3000$, the amplitude parameter for the localized perturbation is set to $\epsilon = 0.0001$ which guarantees a linear evolution. Due to dispersive effects and the damping of the normal modes the amplitudes decrease by about one order of magnitude between $t = 10$ and 40. During this time a strong streamwise shear has developed around the symmetry plane $z = 0$.

For the boundary layer flow, at $Re_{\delta^*} = 950$, a different wall-normal distribution of the localized perturbation is used and the amplitude is set to $\epsilon = 0.001$. However the main flow features in the channel and boundary layer flows remain the same.

A.2 Temporal turbulent channel flow at $Re_\tau = 180$

Turbulent channel flow with periodic boundary conditions in both the streamwise and spanwise direction is provided as an example in **temporal-channel-retau180**. The Reynolds number based on the laminar centerline velocity $Re_{\mathrm{cl}} = 4200$, corresponding to a bulk Reynolds number $Re_b = 2800$. The initial file is created with either **bls.i.noise** (creating the parabolic profile with superimposed small-amplitude noise) or **bls.i.localised** (laminar channel profile with a localized disturbance). The flow is then integrated using **bla.i** with either constant mass flux or setting the pressure gradient to a constant value corresponding to $Re_\tau = 180$ (for further details on the forcing see Section 4.2.1.1). Note that two-dimensional statistics in a xy-plane are taken although the x-direction is homogeneous; the spanwise average is taken directly in **pxyst**. The resolution in **par.f** is not sufficient for a fully resolved direct numerical simulation; for a DNS at least 128 grid points in each direction should be employed.

A.3 Temporal Couette flow with turbulent spots

An example with Couette flow is provided in **temporal-couette-spot**. A localized perturbation is developing in time until a turbulent spot has formed. The Reynolds number $Re_{\mathrm{co}} = 375$ is defined as in equation 4.5 and the box is large enough to fit the perturbation for even longer times than is used in the example ($t_{\mathrm{end}} = 90$).

The horizontal extent of the spot has an elliptical character, with an aspect ratio that increases with increasing Reynolds number. With increasing time however the streamwise elongation becomes less pronounced. For slightly lower Reynolds numbers the spot cannot be sustained. This and similar flow configurations were studied in Lundbladh & Johansson (1991).

A.4 Temporal Falkner–Skan–Cooke boundary layer flow

This test case, located in **temporal-fsc**, is based on a temporal FSC boundary layer flow with the Reynolds number $Re = 337.9$ and a free-stream cross-flow velocity component $W_\infty = 1.44232U_\infty(x = 0)$ and a favorable pressure gradient $m = 0.34207$ as defined in section 4.4.

The initial disturbance used here is the unstable eigenfunction associated with the eigenvalue $c = -0.15246 + i0.0382$ that appears at $\alpha = 0.25$ and $\beta = -0.25$. This corresponds to an inviscid instability due to an inflection point in the boundary layer flow. Note that for a boundary layer flow with a three-dimensional velocity profile one can always find inflection points.

The disturbance is generated by **bls** based on the information given in the file **os.i**. In the example bash script **run.sh** this disturbance is integrated in time to guarantee that a velocity field that fulfills the Navier–Stokes equations is generated. The final velocity field is saved and the time stamp is set to zero. A new simulation is started with the converged velocity field as initial condition. The perturbation energy grows exponentially in time. However if the simulation is run long enough nonlinear saturation will cause the growth rate to decrease.

This disturbance has been used in previous studies, for example, Högberg & Henningson (2002) and Chevalier *et al.* (2007) where it was used as a test case to verify control and estimator algorithms.

A.5 Asymptotic suction boundary layer flow

The example case **temporal-asbl** shows how to set up an asymptotic suction boundary layer flow with a localized perturbation included. The purpose of the steady suction is to stabilize the flow and delay/prohibit transition to turbulence. Detailed simulations based on this implementation can be found in Levin & Henningson (2007).

A.6 Spatial Blasius boundary layer flow

There are three examples included which all highlight some specific feature of the **Simson** package for spatially developing flows. The linear and nonlinear evolution of a Tollmien–Schlichting (TS) wave is simulated in the example **spatial-blasius-TS**. The inlet Reynolds number based on the displacement thickness and free-stream velocity is set to $Re_{\delta_0^*} = 300$ or $Re_x = (300/1.721)^2 = 30387$ in a two-dimensional ($\mathbf{nz} = 1$) computational box. A harmonic forcing located at $Re_x \approx 60386$ ($x = 100$) with nondimensional frequency $F = 10^6\omega/Re_{\delta_0^*} = 120$ acting in the wall-normal direction is introduced, forcing the TS-waves; branch I is at $Re_x \approx 150000$ ($x \approx 666$) and branch II at $Re_x = 387000$ ($x \approx 1071$). The streamwise rms-amplitude of the TS-waves at branch I is approximately 0.76%.

A sample result displaying the evolution of the streamwise fluctuation maximum $u_{\mathrm{rms,max}}$ (option -71 in **pxyst**) is given in figure A.1, where the simulation data are compared to the results obtained from linear PSE (parabolized stability equations). It can be seen in figure A.1a) that the direct nonlinear solution of the Navier–Stokes equations (**bls.i.full** and **bla.i.full**) gives exactly the same result as the one computed nonlinearly in disturbance formulation (**bls.i.dist** and **bla.i.pert**), provided that the corresponding baseflow is a converged solution (*i.e.* obtained using **bls.i.full**

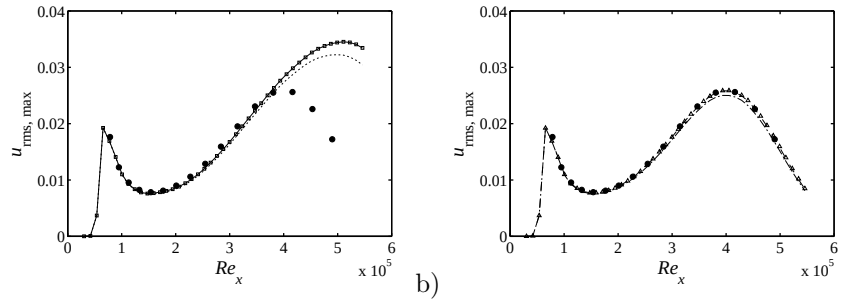


Figure A.1: a) Nonlinear and b) linear development of a two-dimensional TS-wave in a spatially developing Blasius boundary layer (example **spatial-blasius-TS**). Wall-normal maximum of the streamwise rms fluctuation $u_{\mathrm{rms,max}}$ is shown as a function of the downstream distance Re_x. ——— Direct nonlinear solution; □ nonlinear solution in disturbance formulation using a converged baseflow; ····· nonlinear solution in disturbance formulation using the Blasius solution as baseflow. – – – – Linearized disturbance formulation using a converged baseflow; Δ rescaled full solution at $10\times$ lower amplitude; —·— linearized disturbance formulation using the Blasius solution as baseflow. • Results from linear parabolized stability equations (PSE).

and **bla.i.baseflow**). The growth rate is considerably underestimated if the Blasius solution (*i.e.* a solution to the boundary layer equations) is used as a baseflow (**bls.i.pert** and **bla.i.magic-forcing**). On the other hand, due to nonlinear saturation the nonlinear development is considerably different from the linear development depicted in figure A.1b): The linearized solution around a converged baseflow (**bla.i.pert-linear**) and the rescaled direct solution coincide with the linear PSE solution, whereas the linear solution around the Blasius baseflow is again underestimating the growth of the TS-wave.

An example of a spatially evolving, zero-pressure-gradient turbulent boundary layer with various passive scalars is provided in **spatial-blasius-scalars**. The inflow is located at $Re_{\delta_0^*} = 450$. Turbulent statistics, spanwise two-point correlations and time series at selected coordinates of various quantities are generated as output. The laminar flow close to the inlet plane is disturbed via a trip force (see equation (7.7) in Section 7.4) located at $x = 10$. The Prandtl numbers Pr for the scalars are all set to 0.71, and different boundary conditions have been chosen at the wall and in the freestream: Dirichlet-Dirichlet, Neumann-Dirichlet, Dirichlet-Neumann and Neumann-Neumann. Note that the spatial resolution in **par.f** is not sufficient for a fully-resolved direct numerical simulation.

A similar turbulent boundary layer with large-eddy simulation (LES) is given as example in **spatial-blasius-les**. In **sgs.i** the Smagorinsky eddy-viscosity model is turned on together with an adaptive determination of the model coefficient (dynamic Smagorinsky model).

A.7 Spatial Falkner–Skan–Cooke boundary layer flow

In the example **spatial-fsc** the flow from appendix A.4 is studied in a spatial setting. Traveling cross-flow vortices are generated, in an upstream location, by applying a volume force (trip forcing) built as a sum of Stokes modes with time-varying amplitudes and stochastic coefficients.

The vortices merge and split and form complicated patterns. The time average of the disturbance energy is plotted in figure A.2 which shows the exponential growth of the traveling vortices. The fringe forcing starts at $x = 350$ and lowers the perturbation energy and makes the base flow periodic.

The same flow case was studied in Högberg & Henningson (1998), Högberg & Henningson (2002) and Chevalier *et al.* (2007). In the two latter studies this flow configu-

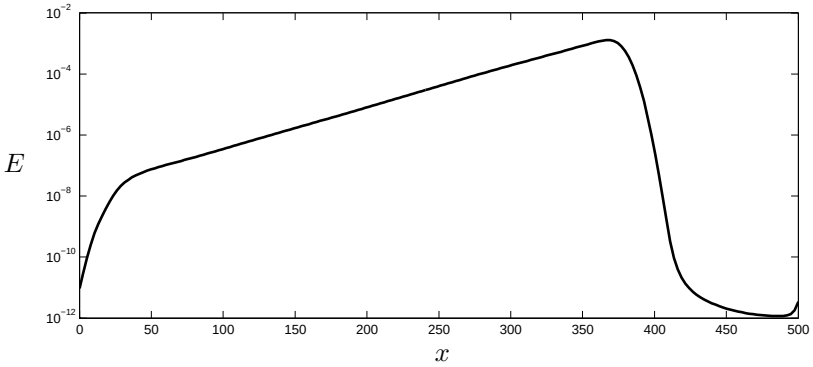


Figure A.2: Time average of energy integrated in the z-direction for simulation of traveling cross-flow vortices.

ration served as an example to illustrate the effectiveness of control and compensator algorithms respectively, where blowing and suction on the lower wall was used to decrease the disturbance energy growth.

B Scaling of variables

For all the boundary layer flows the scaling, for all parameters, is based on the displacement boundary layer thickness and free-stream velocity at $t = 0$, $x = 0$ for the reference or base flow. However, internally in the simulation code **bla** the implementation uses a scaling based on the half box height (the external and internal velocity scale is the same). This means that all external data must be rescaled when read into the program, and the reverse scaling applied on output. If we let **dstar** be the displacement thickness expressed in half box heights, then the following scaling relationships hold:

$$\begin{aligned} \text{time}_{\text{int}} &= \text{time}_{\text{ext}} \times \text{dstar}\,, \\ \text{length}_{\text{int}} &= \text{length}_{\text{ext}} \times \text{dstar}\,, \\ \text{velocity}_{\text{int}} &= \text{velocity}_{\text{ext}}\,, \\ \text{vorticity}_{\text{int}} &= \text{vorticity}_{\text{ext}}/\text{dstar}\,, \\ \text{force}_{\text{int}} &= \text{force}_{\text{ext}}/\text{dstar}\,. \end{aligned} \tag{B.1}$$

All formatted input and output files except the wave amplitude file use external scaling, whereas the unformatted files and the wave amplitude file use internal scaling.

C Subversion quickstart

C.1 Introduction

To ease the development of **Simson** the version control system **Subversion** has been used. **Subversion** works by keeping all source files including their history in a database called a repository where the data is stored in an efficient manner.

Each developer has their own private working copy of the files. Each working copy directory contains a special directory named **.svn** where some administrative files are stored. The developer decides when and how to synchronize the working copy with the repository.

Read and write access to the repository can be specified to certain directories for certain users if needed. This allows developers, responsible for different parts of the code, to work independently.

In this appendix follows a short description on how to use **Subversion** and the most common commands. The entire **Subversion** book/manual can be found in Collins-Sussman *et al.*.

C.2 Getting started

C.2.1 Creating a working copy root directory

The following command creates a new working directory **simson** and puts the latest source code from the **Simson** trunk repository there:

```
svn checkout https://www2.mech.kth.se/svn/simson/trunk simson
```

The `checkout` subcommand is only used when you want to create a new private working copy from scratch.

Now you can edit the files in the **simson** directory. Changes in this directory do not affect other developers. The local **.svn** directories contain administrative files that **Subversion** uses to keep track of your files and should not be changed.

C.2.2 The most commonly used Subversion subcommands

C.2.2.1 General work flow

This is the most often used **Subversion** commands and the general work flow goes through the following items. Each command is described in the following sections.

Update your working copy:

- svn update

Make changes:

- svn add
- svn delete
- svn copy
- svn move

Examine your changes:

- svn status
- svn diff
- svn revert
- tkdiff (svn-aware graphical diff frontend)

Merge others' changes into your working copy:

- svn update (IMPORTANT!)
- svn resolved

Commit your changes:

- svn commit

Note that it is crucial that you update your copy of the repository before you commit your changes. Otherwise changes committed by others between the previous update of your copy and now could be lost.

C.2.2.2 Getting help

If you want a list of all **Subversion** subcommands, use `help`:

```
svn help
```

Type `svn help subcommand` for help on a specific subcommand.

Most subcommands take file and/or directory arguments, recursing on the directories. If no arguments are supplied, it will recurse on the current directory (inclusive) by default. Available subcommands are (aliases are given within parentheses):

```
add
blame (praise, annotate, ann)
cat
checkout (co)
cleanup
commit (ci)
copy (cp)
delete (del, remove, rm)
diff (di)
export
help (?, h)
import
info
list (ls)
log
merge
mkdir
move (mv, rename, ren)
propdel (pdel, pd)
propedit (pedit, pe)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
resolved
revert
status (stat, st)
switch (sw)
update (up)
```

C.2.2.3 Examining and comparing working copy with the repository

The most commonly used subcommand reports what files you have modified in your working directory among other things:

```
svn status
```

The `status` subcommand and many other subcommands works recursively on directories (by default the current working directory). If you want the status of a named directory or a single file you add the name of that file or directory. For instance:

```
svn status mydirectory/myfile.f
```

There is an important extra flag to the `status` subcommand: `-u`. The `-u` flag is used to display pending updates from the repository.

```
svn status -u -v mydirectory/myfile.f
```

To view the history of a file or directory, use the `log` subcommand. This shows who changed the file, when they did it and the log message they provided to describe their changes.

```
svn log
```

The `diff` subcommand examines the difference between your working copy and the repository more in detail. The output is similar to the output of the Unix command `diff`; *e.g.*

```
svn diff afortranfile.f
```

A graphical frontend to `diff` is the freely available `tkdiff`, which is svn-aware. For example,

```
tkdiff afortranfile.f
```

displays all local changes of the given file since the last update, whereas

```
tkdiff -r100 afortranfile.f
```

shows all the changes compared to revision 100 in the repository.

C.2.2.4 Undoing changes to the working copy

If you want to undo your changes to some file or directory in your working copy there is a convenient way. Use `revert`:

```
svn revert myfile.f
```

After the `revert` subcommand has finished the working copy has the same state as when you did `update` or `checkout` the last time.

The `revert` subcommand may also be applied to several files at once, or an entire directory. Because the `revert` subcommand reverts to the mirror-copy kept in the **`.svn`** directories it works even when there is no network connection to the repository, such as when working with a laptop.

C.2.2.5 Updating working copy

The `update` subcommand updates your working copy and the **.svn** directories with the latest changes to the repository.

All files in your current working directory and below are updated. If you have also changed the contents of your working copy the repository changes are merged with your changes.

```
svn update
```

Again, you may update a single file or directory by naming it:

```
svn update mydirectory/myfile.f
```

Note that you can run `status -u` to get a list of all updates available in the repository. This is a good way to predict what `update` will do.

The `update` subcommand is also used when you want to retrieve an old version of some file(s) from the repository; *e.g.* to get the repository version of `myfile.f` as it was on the 6:th of October 2004 you write:

```
svn update -r '{20041006}' mydirectory/myfile.f
```

In some cases `update` will fail to merge changes to the repository with your local changes. If `update` or `status` lists a file with a "C" in front this means that you need to merge the reported file(s) manually.

C.2.2.6 Committing modified files into the repository

The `commit` subcommand updates the repository with your changes.

```
svn commit -m 'New adiabatic boundary condition added'
```

This commits all files in the current directory and below recursively. You may also commit a single file or directory by naming it:

```
svn commit mydirectory/somefile.f -m 'Fixed memory leak problem'
```

Note that `commit` will safely fail if you do not have write access to all corresponding files in the repository.

C.2.3 Other useful subcommands

C.2.3.1 Commands for moving, removing, and adding files

If you add or delete files from your working copy you need to explicitly tell **Subversion** about this. The subcommands `add`, `move`, `copy` and `delete` are used for this. See `svn help xxx` for help on these subcommands. The `rename` subcommand is an alias for `move`.

As usual, the repository is not changed when you apply these commands on your working copy. When you do `svn commit` the repository changed.

C.2.3.2 Adding the keywords to a new file

Subversion can substitute certain information directly into the files. This is done by putting keywords inside the file. When you add a new file for version control, put the following lines in the beginning of that file:

```
$HeadURL: https://www2.mech.kth.se/svn/simson/trunk/doc/simson-user-guide-v4.0.tex $
$LastChangedDate: 2007-12-07 13:05:07 +0100 (fr, 07 dec 2007) $
$LastChangedBy: mattias@MECH.KTH.SE $
$LastChangedRevision: 1053 $
```

You must tell **Subversion** which keywords to look for. This is done through the following command (on one line):

```
svn propset svn:keywords 'LastChangedBy LastChangedDate
LastChangedRevision HeadURL' filename
```

Another important keyword is the ignore tag. This is set/changed for the current directory through the command

```
svn propedit svn:ignore .
```

and then editing the file names in the editor window. In general, properties of a specific file or directory are displayed by

```
svn proplist <FILE>
svn propget <PROPERTY> <FILE>
```

C.2.3.3 Looking inside the repository

Because the **Subversion** repository is stored in a database file on a remote server machine you cannot look at the repository files using Unix/Linux `ls` command. Instead you must use the **Subversion** subcommand `ls`.

```
svn ls https://www2.mech.kth.se/svn/simson/releases
```

Note that many **Subversion** subcommands that take a working copy directory as an argument may also take a repository URL as an argument.

C.2.3.4 Exporting source code

To export the source code from a **Subversion** working copy into a source code tree without the **.svn** directories there is a convenient command:

```
svn export mydirectory simson-export
```

This creates a new directory **simson-export** with your source code in it. The only thing `export` does is to copy all files recursively from your working copy (or from a repository URL) into a new directory, omitting all **.svn** directories.

C.2.4 Manually merging a conflict

When `svn status` reports a file or directory with a "C" in front of the filename this means that there is a conflict that **Subversion** cannot resolve. This could happen if you and another developer have simultaneously changed the same lines in the same file. When the other developer commits his changes and you update your working copy **Subversion** finds the conflict which you have to resolve manually.

To resolve a conflict simply edit the conflicting file and then save it. To help you out, **Subversion** has written both your modifications and the modifications from the repository into the file. In addition, **Subversion** has saved your file and the repository's file in your working copy directory (**.mine** and **.Rxxx** respectively). Note that you need to remove the two additional files before you can commit your file.

C.2.5 Branches

Branches are versions of any file or directory that were forked from another version of the same file or directory and are subsequently developed further independent of the original version. A branch can *e.g.* include additional features that are not (yet) meant to be part of the original file/directory version. Branches are kept in the directory **branches**. To create a branch, simply copy the base version to the branch directory, *e.g.* from the **trunk/bla** directory

```
svn copy trunk/bla branches/bla-branch
```

followed by a commit of the new branch via

```
svn commit trunk/bla-branch -m'Created new branch bla-branch, based on revision XXX'
```

It is very important to state in the commit comment from which revision the branch was created. **Subversion** will not keep track of that essential information. Note that the commit will create a new revision which will overwrite the revision information in the newly copied files.

The main advantage of using branches comes from the fact that it is possible to keep a branch copy updated with respect to the original version (*i.e.* the one the branch was based upon). To update a branch, use

```
svn merge -rXXX:YYY trunk/bla/bla.f branches/bla-branch/bla.f
```

The two revision `XXX` and `YYY` are essential: They specify that all the changes in **trunk/bla/bla.f** made between revisions `XXX` and `YYY` should be included in **branches/bla-branch/bla.f**. A merge will probably create some conflicts which have to be resolved manually. As mentioned above, when committing the merged version, it is essential to include in the comment information about the revision numbers the merge was based on. **Subversion** will not keep track of this information. Therefore, use a comment similar to

```
svn commit -m'merged <files> with trunk revisions from XXX to YYY' <files>
```

In this way, for a subsequent merge, the logs can be used to determine up to which revision number a merge has already been performed.

C.2.6 Private configuration file

After running `svn` for the first time you will have a directory in your home directory named **.subversion**. This directory contains the file **.subversion/config** can be edited to customize **Subversion** behavior.

C.3 Examples

Below follows common examples of **Subversion** subcommands.

Create a new working copy root directory

```
svn checkout https://www2.mech.kth.se/svn/simson/trunk Simson
```

List your modifications to the working copy.

```
svn status somedirectory
```

List modifications to the repository and predict what `update` would do.

```
svn status -u somedirectory
```

Update your working copy with modifications to the repository.

```
svn update somefile.f90
```

Update the repository with modifications to your working copy of a file.

```
svn commit -m 'Your log-message here' somefile.f90
```

Get an old version of a file from the repository.

```
svn update -r '{20041006}' somefile.f90
```

Undo modifications to the working copy.

```
svn revert somefile.f90
```

List history of a file.

```
svn log somefile.f90
```

List details of your modifications to a working copy file.

```
svn diff somefile.f90
```

List differences between your working copy and an old repository version of a file.

```
svn diff -r '{20041006}' somefile.f90
```

List differences between two old repository versions of a file.

```
svn diff -r '{20041006}:{20040930}' somefile.f90
```

List differences between two old repository version r110 and r120 of a file.

```
svn diff -r 110:120 somefile.f90
```

List files in the repository.

```
svn ls https://www2.mech.kth.se/svn/simson/
```

Rename a file/directory.

```
svn mv somefile.f90 newname.f90
```

Put a new file/directory under version control.

```
svn add somefile.f90
```

Remove a file/directory.

```
svn delete somefile.f90
```

Export a source code tree for distribution to a user/customer.

```
svn export https://www2.mech.kth.se/svn/simson/releases/mydir mydir
```

View who has written specific parts of a file

```
svn blame somefile.f90
```

To resurrect an earlier deleted file (from revison 972)

```
svn cp -r972 https://www2.mech.kth.se/svn/simson/trunk/myfile myfile
```

To merge changes in trunk version of bla from revision 687 to 691 to one branch copy of bla

```
svn merge -r687:691 bla/bla.f ../branches/bla2dparallel/bla.f
```

And finally to get help

```
svn help
```