

Nektar++: Spectral/hp Element Framework

Version 5.2.0

User Guide

August 23, 2022

Department of Aeronautics, Imperial College London, UK Scientific Computing and Imaging Institute, University of Utah, USA ii

Contents

xi

Ι	Get	tting S	Started	1		
1	Inst	callation 2				
	1.1	Install	ing Debian/Ubuntu Packages	2		
	1.2	Install	ing Redhat/Fedora Packages	3		
	1.3	Install	ing from Source	3		
		1.3.1	Obtaining the source code	3		
		1.3.2	Linux	4		
		1.3.3	OS X	7		
		1.3.4	Windows	10		
		1.3.5	CMake Option Reference	14		
2	Ma	themat	tical Formulation	20		
	2.1	Backg	round	20		
	2.2	Metho	ds overview	21		
		2.2.1	The finite element method (FEM)	21		
		2.2.2	High-order finite element methods	21		
		2.2.3	The Galerkin formulation	23		
3	3 XML Session File		sion File	25		
	3.1	Geom	etry	26		
		3.1.1	Vertices	27		
		3.1.2	Edges	28		
		3.1.3	Faces	28		
		3.1.4	Element	28		
		3.1.5	Curved Edges and Faces	29		
		3.1.6	Composites	29		
		3.1.7	Domain	30		

Introduction

\mathbf{iv} Contents

3.2	Expansions		
3.3	Condi	$tions \ldots \ldots$	31
	3.3.1	Parameters	31
	3.3.2	Time Integration Scheme	31
	3.3.3	Solver Information	32
	3.3.4	Variables	33
	3.3.5	Global System Solution Algorithm	33
	3.3.6	Boundary Regions and Conditions	39
	3.3.7	Functions	42
	3.3.8	Quasi-3D approach	44
3.4	Filters	5	45
	3.4.1	Phase sampling	46
	3.4.2	Aerodynamic forces	47
	3.4.3	Benchmark	49
	3.4.4	Cell history points	49
	3.4.5	Checkpoint cell model	50
	3.4.6	Checkpoint fields	50
	3.4.7	Electrogram	51
	3.4.8	Error	52
	3.4.9	FieldConvert checkpoints	52
	3.4.10	History points	53
	3.4.11	Kinetic energy and enstrophy	55
	3.4.12	Mean values	56
	3.4.13	Modal energy	56
	3.4.14	Moving body	57
	3.4.15	Moving average of fields	57
	3.4.16	One-dimensional energy	58
	3.4.17	Reynolds stresses	58
	3.4.18	Time-averaged fields	59
	3.4.19	ThresholdMax	60
	3.4.20	ThresholdMin value	60
	3.4.21	Maximun/minimun fields	60
3.5	Forcin	g	61
	3.5.1	Absorption	52
	3.5.2	Body	52
	3.5.3	MovingReferenceFrame	52
	3.5.4	Programmatic	64
	3.5.5	Noise	64
3.6	Coupli	ing	65
	3.6.1	File	65
	3.6.2	Cwipi	66
3.7	Expres	ssions	67
	3.7.1	Variables and coordinate systems	68
	3.7.2	Performance considerations	73

Contents v	
------------	--

Π	II Preprocessing & Postprocessing 75		
4	Neł	cMesh	76
	4.1	Exporting a mesh from Gmsh	77
	4.2	Defining physical surfaces and volumes	77
	4.3	Converting the MSH to Nektar++ format	78
	4.4	NekMesh in NekPy	79
	4.5	NekMesh modules	81
		4.5.1 Input modules	82
		4.5.2 Output modules	84
		4.5.3 Extract surfaces from a mesh	86
		4.5.4 Negative Jacobian detection	87
		4.5.5 Spherigon patches	87
		4.5.6 Periodic boundary condition alignment	88
		4.5.7 Boundary layer splitting	90
		4.5.8 High-order cylinder generation	91
		4.5.9 Linearisation	92
		4.5.10~ Extracting interface between tetrahedra and prismatic elements	92
		4.5.11 Boundary identification	93
		4.5.12 Scalar function curvature	93
		4.5.13 Link Checking	93
		4.5.14 2D mesh extrusion \ldots	93
		4.5.15 Variational Optimisation	94
		4.5.16 r -adaptation	94
		4.5.17 Mesh projection	95
	4.6	Mesh generation	95
		4.6.1 Methodology	95
		4.6.2 Mesh generation manual	98
5	Fiel	ldConvert	103
	5.1	Basic usage	103
		5.1.1 Input formats	104
	5.2	Convert .fld / .chk files into Paraview, VisIt or Tecplot format	104
		5.2.1 Using the VTK library for output	105
	5.3	Convert field files between XML and HDF5 format	107
	5.4	Range option -r	107
	5.5	FieldConvert in NekPy	108
	5.6	FieldConvert modules -m	109
		5.6.1 Smooth the data: <i>C0Projection</i> module	111
		5.6.2 Calculate CFL number: CFL module	112
		5.6.3 Calculate Q-Criterion: <i>QCriterion</i> module	112
		5.6.4 Calculate λ_2 : <i>L2Criterion</i> module	112
		5.6.5 Add composite ID: <i>addcompositeid</i> module	113
		5.6.6 Add new field: <i>fieldfromstring</i> module	113
		5.6.7 Sum two .fld files: <i>addFld</i> module	113

vi Contents

	5.6.8	Combine two .fld files containing time averages: <i>combineAvg</i> module114
	5.6.9	Concatenate two files: <i>concatenate</i> module
	5.6.10	Count the number of DOF: <i>dof</i> module
	5.6.11	Equi-spaced output of data: equispaced output module
	5.6.12	Extract a boundary region: <i>extract</i> module
	5.6.13	Compute the gradient of a field: gradient module
	5.6.14	Convert HalfMode expansion to SingleMode for further processing:
		halfmodetofourier module
	5.6.15	Extract a plane from 3DH1D expansion: <i>homplane</i> module 116
	5.6.16	Stretch a 3DH1D expansion: <i>homstretch</i> module
	5.6.17	Inner Product of a single or series of fields with respect to a single
		or series of fields: <i>innerproduct</i> module
	5.6.18	Interpolate one field to another: <i>interpfield</i> module
	5.6.19	Interpolate scattered point data to a field: <i>interpoint data to fld</i>
		module
	5.6.20	Interpolate a field to a series of points: <i>interppoints</i> module 119
	5.6.21	Interpolate a set of points to another: <i>interpptstopts</i> module 121
	5.6.22	Isocontour extraction: <i>iscontour</i> module
	5.6.23	Show high frequency energy of the Jacobian: <i>jacobianenergy</i> module122
	5.6.24	Calculate mesh quality: <i>qualitymetric</i> module
	5.6.25	Evaluate the mean of variables on the domain: $mean$ module \ldots 123
	5.6.26	Extract mean mode of 3DH1D expansion: $meanmode \mod 1$ 123
	5.6.27	Project point data to a field: <i>pointdatatofld</i> module
	5.6.28	Print L2 and LInf norms: <i>printfldnorms</i> module
	5.6.29	Removes one or more fields from .fld files: $remove field \mod 1.125$
	5.6.30	Computes the scalar gradient: <i>scalargrad</i> module
	5.6.31	Scale a given .fld: <i>scaleinputfld</i> module
	5.6.32	Time-averaged shear stress metrics: <i>shear</i> module
	5.6.33	Stream function of a 2D incompressible flow: <i>streamfunction</i> module126
	5.6.34	Boundary layer height calculation: <i>surfdistance</i> module 126
	5.6.35	Calculate vorticity: <i>vorticity</i> module
	5.6.36	Computing the wall shear stress: wss module
	5.6.37	Calculating the shape function Φ for an SPM case: <i>phifile</i> module 127
	5.6.38	Interpolate values for a point array: <i>wallNormalData</i> module 128
	5.6.39	Manipulating meshes with FieldConvert
5.7	FieldC	onvert in parallel
5.8	Proces	sing large files in serial $\ldots \ldots 131$
	5.8.1	Using the <i>part-only</i> and <i>part-only-overlapping</i> options 131
	5.8.2	Using the <i>nparts</i> options
	5.8.3	Running in parallel with the <i>nparts</i> option

IIISolver Applications

6 Acoustic Solver

 $\mathbf{134}$

 $\mathbf{135}$

	6.1	Synop	sis	135
		6.1.1	Linearized Euler Equations	135
		6.1.2	Acoustic Perturbation Equations	136
	6.2	Usage		137
	6.3	Session	n file configuration	137
		6.3.1	Time Integration Scheme	138
		6.3.2	Solver Info	138
		6.3.3	Variables	138
		6.3.4	Functions	139
		6.3.5	Boundary Conditions	139
	6.4	Exam	ples	140
		6.4.1	Wave Propagation in a Sheared Base Flow	140
7	Adv	vection	-Diffusion-Reaction Solver	143
	7.1	Synop	sis	143
	7.2	Usage		144
	7.3	Session	n file configuration	144
		7.3.1	Time Integration Scheme	144
		7.3.2	Solver Info	144
		7.3.3	Parameters	145
		7.3.4	Functions	145
	7.4	Exam	ples	146
		7.4.1	1D Advection equation	146
		7.4.2	2D Helmholtz Problem	147
		7.4.3	Advection dominated mass transport in a pipe	150
		7.4.4	Unsteady reaction-diffusion systems	153
8	Car	diac E	lectrophysiology Solver	156
	8.1	Synop	sis	156
		8.1.1	Bidomain Model	156
		8.1.2	Monodomain Model	156
		8.1.3	Cell Models	157
	8.2	Usage		157
	8.3	Session	n file configuration	157
		8.3.1	Solver Info	157
		8.3.2	Parameters	158
		8.3.3	Functions	159
		8.3.4	Filters	159
		8.3.5	Stimuli	159
9	Cor	npressi	ible Flow Solver	162
	9.1	Synop	sis	162
		9.1.1	Euler equations	162
		9.1.2	Compressible Navier-Stokes equations	163
		9.1.3	Numerical discretisation	163

	9.2	Usage		. 164
	9.3	Session	file configuration	. 164
	9.4	Examp	\overline{les}	. 172
		9.4.1	Shock capturing	. 172
		9.4.2	Variable polynomial order	. 174
		9.4.3	De-Aliasing Techniques	. 175
		9.4.4	Implicit solver	. 176
10	Dur	nmy Se	olver	178
	10.1	Synops	sis	. 178
11	Inco	ompres	sible Navier-Stokes Solver	179
	11.1	Synops	sis	. 179
		11.1.1	Velocity Correction Scheme	. 179
		11.1.2	Immersed Boundary Methods: Smoothed Profile Method	. 186
		11.1.3	Direct solver (coupled approach)	. 190
		11.1.4	Linear Stability Analysis	. 191
		11.1.5	Steady-state solver using Selective Frequency Damping	. 194
	11.2	Usage		. 195
	11.3	Session	file configuration	. 195
		11.3.1	Solver Info	. 195
		11.3.2	Parameters	. 199
		11.3.3	Womersley Boundary Condition	. 200
		11.3.4	Forcing	. 201
		11.3.5	Filters	. 203
	11.4	Session	file configuration: Linear stability analysis	. 204
		11.4.1	Solver Info	. 204
		11.4.2	Parameters	. 205
		11.4.3	Functions	. 206
	11.5	Session	a file configuration: Steady-state solver	. 207
		11.5.1	Execution of the classical steady-state solver	. 207
		11.5.2	Execution of the adaptive steady-state solver	. 208
	11.6	Session	a file configuration: Coordinate transformations	. 209
		11.6.1	Solver Info	. 209
		11.6.2	Parameters	. 209
		11.6.3	Mapping	. 209
		11.6.4	Functions	. 210
		11.6.5	Boundary conditions	. 210
	11.7	Session	a file configuration: Adaptive polynomial order	. 211
		11.7.1	Solver Info	. 212
		11.7.2	Parameters	. 212
		11.7.3	Functions	. 212
		11.7.4	Restarting the simulation	. 213
	11.8	Advect	ing extra passive scalar fields	. 213
	11.9	Imposi	ng a constant flowrate	. 214

11.10	DExamples
	11.10.1 Kovasznay Flow 2D
	11.10.2 Kovasznay Flow 2D using high-order outflow boundary conditions 218
	11.10.3 Steady Kovasznay Oseen Flow using the direct solver
	11.10.4 Laminar Channel Flow 2D
	11.10.5 Laminar Channel Flow 3D
	11.10.6 Laminar Channel Flow Quasi-3D
	11.10.7 Turbulent Channel Flow
	11.10.8 Turbulent Pipe Flow
	11.10.9 Aortic Blood Flow
	11.10.16 inite-strip modeling of flow past flexible cables
	11.10.12D direct stability analysis of the channel flow
	11.10.12D adjoint stability analysis of the channel flow
	11.10.12D Transient Growth analysis of a flow past a backward-facing step 243
	11.10.14BiGlobal Floquet analysis of a of flow past a cylinder
12 Line	ear elasticity solver 253
12.1	Synopsis
	12.1.1 The linear elasticity equations
12.2	Usage
12.3	Session file configuration
	12.3.1 Solver Info
	12.3.2 Parameters
12.4	Examples
	12.4.1 L-shaped domain
	12.4.2 Boundary layer deformation
10 0 1	
13 Puls	Se Wave Solver 258
13.1	
13.2	Usage
13.3	Session file configuration
	13.3.1 Pulse Wave Solver mesh connectivity
	13.3.2 Time Integration Scheme
	13.3.3 Session Info
	13.3.4 Parameters
	13.3.5 Boundary conditions
	13.3.6 Functions
13.4	Examples
	13.4.1 Human Vascular Network
	13.4.2 Stented Artery
13.5	Further Information
13.6	Future Development
13.7	References

14 Shallow Water Solver

x Contents

14.1 Synopsis	276 276
14.2 Usage	277
14.3 Session file configuration	277
14.3.1 Time Integration Scheme	277
14.3.2 Solver Info	277
14.3.3 Parameters	277
14.3.4 Functions	277
14.4 Examples	278
14.4.1 Rossby modon case	278
15 Optimisation 15.1 Collections and MatrixFree operations 15.1.1 Automatic tuning and the –writeoptfile command line option 15.1.2 Manually selecting the COLLECTIONS section 15.1.3 Collection size	280 281 . 281 . 282 . 283 . 284
 15 Optimisation 15.1 Collections and MatrixFree operations	280 281 . 281 . 282 . 283 . 283 . 284 285
 15 Optimisation 15.1 Collections and MatrixFree operations 15.1.1 Automatic tuning and the -writeoptfile command line option 15.1.2 Manually selecting the COLLECTIONS section 15.1.3 Collection size 16 Command-line Options 17 Frequently Asked Questions 	280 281 281 282 283 284 285 287
 15 Optimisation 15.1 Collections and MatrixFree operations 15.1.1 Automatic tuning and the -writeoptfile command line option 15.1.2 Manually selecting the COLLECTIONS section 15.1.3 Collection size 16 Command-line Options 17 Frequently Asked Questions 17.1 Compilation and Testing 	280 281 . 281 . 282 . 283 . 283 . 284 285 287 . 287
 15 Optimisation 15.1 Collections and MatrixFree operations 15.1.1 Automatic tuning and the -writeoptfile command line option 15.1.2 Manually selecting the COLLECTIONS section 15.1.3 Collection size 16 Command-line Options 17 Frequently Asked Questions 17.1 Compilation and Testing 17.2 Usage 	280 281 . 281 . 282 . 283 . 283 . 284 285 287 . 287 . 289

Introduction

Nektar++ [6] is a tensor product based finite element package designed to allow one to construct efficient classical low polynomial order h-type solvers (where h is the size of the finite element) as well as higher p-order piecewise polynomial order solvers. The framework currently has the following capabilities:

- Representation of one, two and three-dimensional fields as a collection of piecewise continuous or discontinuous polynomial domains.
- Segment, plane and volume domains are permissible, as well as domains representing curves and surfaces (dimensionally-embedded domains).
- Hybrid shaped elements, i.e triangles and quadrilaterals or tetrahedra, prisms and hexahedra.
- Both hierarchical and nodal expansion bases.
- Continuous or discontinuous Galerkin operators.
- Cross platform support for Linux, Mac OS X and Windows.

The framework comes with a number of solvers and also allows one to construct a variety of new solvers.

Our current goals are to develop:

- Automatic auto-tuning of optimal operator implementations based upon not only h and p but also hardware considerations and mesh connectivity.
- Temporal and spatial adaption.
- Features enabling evaluation of high-order meshing techniques.

xii Introduction

For further information and to download the software, visit the Nektar++ website at http://www.nektar.info.

Part I Getting Started

CHAPTER **1**

Installation

Nektar++ is available in both a source-code distribution and as pre-compiled binary packages for a number of operating systems. We recommend using the pre-compiled packages if you wish to use the existing Nektar++ solvers for simulation and do not need to perform additional code development.

1.1 Installing Debian/Ubuntu Packages

Binary packages are available for current Debian/Ubuntu based Linux distributions. These can be installed through the use of standard system package management utilities, such as Apt, if administrative access is available.

1. Add the appropriate line for the Debian-based distribution to the end of the file /etc/apt/sources.list

Distribution	Repository
Debian 10.0 (buster)	deb http://www.nektar.info/debian-buster buster contrib
Debian 9.0 (stretch)	deb http://www.nektar.info/debian-stretch stretch contrib
Debian 8.0 (jessie)	deb http://www.nektar.info/debian-jessie jessie contrib
Ubuntu 14.04 (trusty)	<pre>deb http://www.nektar.info/ubuntu-trusty trusty contrib</pre>
Ubuntu 18.04 (bionic beaver)	deb http://www.nektar.info/ubuntu-bionic bionic contrib

2. Update the package lists

```
apt-get update
```

3. Install the required Nektar++ packages, or the complete suite with:

```
apt-get install nektar++
```

Any additional dependencies required for Nektar++ to function will be automatically installed. Tip Nektar++ is split into multiple packages for the different components of the software. A list of available Nektar++ packages can be found using: apt-cache search nektar++

1.2 Installing Redhat/Fedora Packages

Add a file to the directory /etc/yum.repos.d/nektar.repo with the following contents

[Nektar] name=nektar baseurl=<baseurl>

substituting **<baseurl>** for the appropriate line from the table below.

Distribution	<baseurl></baseurl>
Fedora 25	http://www.nektar.info/fedora/25/\$basearch



Note

The **\$basearch** variable is automatically replaced by Yum with the architecture of your system.

1.3 Installing from Source

This section explains how to build Nektar++ from the source-code package.

Nektar++ uses a number of third-party libraries. Some of these are required, others are optional. It is generally more straightforward to use versions of these libraries supplied pre-packaged for your operating system, but if you run into difficulties with compilation errors or failing regression tests, the Nektar++ build system can automatically build tried-and-tested versions of these libraries for you. This requires enabling the relevant options in the CMake configuration.

1.3.1 Obtaining the source code

There are two ways to obtain the source code for Nektar++:

• Download the latest source-code archive from the Nektar++ downloads page.

- 4 Chapter 1 Installation
 - Clone the git repository
 - Using anonymous access. This does not require credentials but any changes to the code cannot be pushed to the public repository. Use this initially if you would like to try using Nektar++ or make local changes to the code.

```
git clone https://gitlab.nektar.info/nektar/nektar.git nektar++
```

 Using authenticated access. This will allow you to directly contribute back into the code.

git clone git@gitlab.nektar.info:nektar/nektar.git nektar++



You can easily switch to using the authenticated access from anonymous access at a later date by running git remote set-url origin git@gitlab.nektar.info:nektar/nektar.git

1.3.2 Linux

1.3.2.1 Prerequisites

Nektar++ uses a number of external programs and libraries for some or all of its functionality. Some of these are *required* and must be installed prior to compiling Nektar++, most of which are available as pre-built *system* packages on most Linux distributions or can be installed manually by a *user*. Typically, the development packages, with a *-dev* or *-devel* suffix, are required to compile codes against these libraries. Others are optional and required only for specific features, or can be downloaded and compiled for use with Nektar++ *automatically* (but not installed system-wide).

		Installation			
Package	Req.	Sys.	User	Auto.	Note
C++ compiler	1	1			gcc, icc, etc, supporting C++11
$\mathrm{CMake} \geq 3.5.1$	1	✓	1		Ncurses GUI optional
BLAS	1	\checkmark	1	1	Or MKL, ACML, OpenBLAS
LAPACK	1	~	1	1	
Boost $>= 1.56$	1	1	1	\checkmark	Compile with iostreams
TinyXML	1	1	1	\checkmark	For reading XML input files
Scotch	1	1	\checkmark	1	Required for multi-level static con- densation, highly recommended
METIS		1	1	1	Alternative mesh partitioning
FFTW > 3.0		1	1	1	For high-performance FFTs
ARPACK > 2.0		1	1	\checkmark	For arnoldi algorithms
MPI		✓	1		For parallel execution (OpenMPI,
					MPICH, Intel MPI, etc)
GSMPI				1	For parallel execution
HDF5		1	1	1	For large-scale parallel I/O (requires
					CMake > 3.1)
OpenCascade CE		\checkmark	\checkmark	\checkmark	For mesh generation and optimisa-
					tion
PETSc		\checkmark	1	\checkmark	Alternative linear solvers
PT-Scotch		\checkmark	1	\checkmark	Required when MPI enabled
Tetgen		\checkmark	\checkmark	1	For 3D mesh generation
Triangle		\checkmark	1	\checkmark	For 2D mesh generation
VTK > 5.8		\checkmark	1		Not required to convert field output
					files to VTK, only mesh files

1.3.2.2 Quick Start

Open a terminal.

If you have downloaded the tarball, first unpack it:

```
tar -zxvf nektar++-5.2.0.tar.gz
```

Change into the **nektar++** source code directory

```
mkdir -p build && cd build
ccmake ../
make install
```

1.3.2.3 Detailed instructions

From a terminal:

- 6 Chapter 1 Installation
 - 1. If you have downloaded the tarball, first unpack it

```
tar -zxvf nektar++-5.2.0.tar.gz
```

2. Change into the source-code directory, create a build subdirectory and enter it

```
mkdir -p build && cd build
```

3. Run the CMake GUI and configure the build by pressing c

ccmake ../

- Select the components of Nektar++ (prefixed with NEKTAR_BUILD_) you would like to build. Disabling solvers which you do not require will speed up the build process.
- Select the optional libraries you would like to use (prefixed with NEKTAR_USE_)) for additional functionality.
- Select the libraries not already available on your system which you wish to be compiled automatically (prefixed with <code>THIRDPARTY_BUILD_</code>). Some of these will be automatically enabled if not found on your system.
- Choose the installation location by adjusting CMAKE_INSTALL_PREFIX. By default, this will be a dist subdirectory within the build directory, which is satisfactory for most users initially.

A full list of configuration options can be found in Section 1.3.5.

Note

Selecting THIRDPARTY_BUILD_ options will request CMake to automatically download thirdparty libraries and compile them within the *Nektar++* directory. If you have administrative access to your machine, it is recommended to install the libraries system-wide through your package-management system.

If you have installed additional system packages since running CMake, you may need to wipe your build directory and rerun CMake for them to be detected.

4. Press c to configure the build. If errors arise relating to missing libraries, review the THIRDPARTY_BUILD_ selections in the configuration step above or install the missing libraries manually or from system packages.

- 5. When configuration completes without errors, press c again until the option g to generate build files appears. Press g to generate the build files and exit CMake.
- 6. Compile the code

make install

During the build, missing third-party libraries will be automatically downloaded, configured and built in the Nektar++ build directory.



7. Test the build by running unit and regression tests.

ctest

1.3.3 OS X

1.3.3.1 Prerequisites

Nektar++ uses a number of external programs and libraries for some or all of its functionality. Some of these are *required* and must be installed prior to compiling Nektar++, most of which are available on *MacPorts* (www.macports.org) or can be installed manually by a *user*. Others are optional and required only for specific features, or can be downloaded and compiled for use with Nektar++ *automatically* (but not installed system-wide).

Note

To compile *Nektar++* on OS X, Apple's Xcode Developer Tools must be installed. They can be installed either from the App Store (only on Mac OS 10.7 and above) or downloaded directly from http://connect.apple.com/ (you are required to have an Apple Developer Connection account). Xcode includes Apple implementations of BLAS and LAPACK (called the Accelerate Framework).

		Installation			
Package	Req.	MacPorts	User	Auto.	Note
Xcode	1				Provides developer tools
$CMake \geq 3.5.1$	1	cmake	1		Ncurses GUI optional
BLAS	1				Part of Xcode
LAPACK	1				Part of Xcode
Boost $>= 1.56$	1	boost	1	1	Compile with iostreams
TinyXML	1	tinyxml	1	1	
Scotch	1	scotch	1	\checkmark	Required for multi-level static
					condensation, highly recom- mended
METIS		metis	1	1	Alternative mesh partitioning
FFTW > 3.0		fftw-3	✓	\checkmark	For high-performance FFTs
ARPACK > 2.0		arpack	1		For arnoldi algorithms
OpenMPI		openmpi			For parallel execution
GSMPI				1	For parallel execution
HDF5			1	\checkmark	For large-scale parallel I/O (re- quires CMake >3.1)
OpenCascade CE				1	For mesh generation and $opti-$
Openeaseade OL			•	•	misation
PETSc		petsc	1	1	Alternative linear solvers
PT-Scotch			1	1	Required when MPI enabled
Tetgen			1	1	For 3D mesh generation
Triangle			~	1	For 2D mesh generation
VTK > 5.8		vtk	~		Not required to convert field
					output files to VTK, only mesh
					files

Tip

CMake, and some other software, is available from MacPorts (http://macports.org) and can be installed using, for example,

sudo port install cmake

Package names are given in the table above. Similar packages also exist in other package managers such as Homebrew.

1.3.3.2 Quick Start

Open a terminal (Applications->Utilities->Terminal).

If you have downloaded the tarball, first unpack it:

tar -zxvf nektar++-5.2.0.tar.gz

Change into the **nektar++** source code directory

```
mkdir -p build && cd build
ccmake ../
make install
```

1.3.3.3 Detailed instructions

From a terminal (Applications->Utilities->Terminal):

1. If you have downloaded the tarball, first unpack it

```
tar -zxvf nektar++-5.2.0.tar.gz
```

2. Change into the source-code directory, create a build subdirectory and enter it

mkdir -p build && cd build

3. Run the CMake GUI and configure the build

ccmake ../

Use the arrow keys to navigate the options and ENTER to select/edit an option.

- Select the components of Nektar++ (prefixed with NEKTAR_BUILD_) you would like to build. Disabling solvers which you do not require will speed up the build process.
- Select the optional libraries you would like to use (prefixed with NEKTAR_USE_) for additional functionality.
- Select the libraries not already available on your system which you wish to be compiled automatically (prefixed with THIRDPARTY_BUILD_)
- Choose the installation location by adjusting CMAKE_INSTALL_PREFIX. By default, this will be a dist subdirectory within the build directory, which is satisfactory for most users initially.

A full list of configuration options can be found in Section 1.3.5.



Selecting THIRDPARTY_BUILD_ options will request CMake to automatically download thirdparty libraries and compile them within the *Nektar++* directory. If you have administrative access to your machine, it is recommended to install the libraries system-wide through MacPorts.

- 4. Press c to configure the build. If errors arise relating to missing libraries (variables set to NOTFOUND), review the THIRDPARTY_BUILD_ selections in the previous step or install the missing libraries manually or through MacPorts.
- 5. When configuration completes without errors, press c again until the option g to generate build files appears. Press g to generate the build files and exit CMake.
- 6. Compile the code

make install

During the build, missing third-party libraries will be automatically downloaded, configured and built in the Nektar++ build directory.

Tip If you have multiple processors/cores on your system, compilation can be significantly increased by adding the -jX option to make, where X is the number of simultaneous jobs to spawn. For example,

make -j4 install

7. Test the build by running unit and regression tests.

ctest

1.3.4 Windows

Windows compilation is supported but there are some complexities with building additional features on this platform at present. As such, only builds with a minimal amount of additional build packages are currently supported. These can either be installed by the user, or automatically as part of the build process. Support has recently been added for building with MPI on Windows. This enables parallel computations to be carried out with *Nektar++* on Windows where only sequential computations were previously supported.

		Installation		
Package	Req.	User	Auto.	Note
MS Visual Studio	1	1		2015, 2017 and 2019 known working
$CMake \geq 3.5.1$	\checkmark	1		3.16+ recommended, see info below
BLAS	\checkmark	1	\checkmark	
LAPACK	\checkmark	1	\checkmark	
Boost ≥ 1.61	\checkmark	1	\checkmark	Recommend installing from binaries
Microsoft MPI $\geq 10.1.2$		1		Required for parallel execution. In- stall both runtime and SDK



These instructions assume you are using a 64-bit version of Windows 10.

Note

Note

There have been issues with automatically building Boost from source as a third party dependency during the Nektar++ build when using MS Visual Studio 2015, 2017 and 2019. This should now be possible but it is, nonetheless, recommended you install a suitable version of Boost from binaries as detailed in the instructions below.

1.3.4.1 Detailed instructions

- 1. Install Microsoft Visual Studio 2019 (preferred), 2017 or 2015 (both known to work). This can be obtained from Microsoft free of charge by using their Community developer tools from https://visualstudio.microsoft.com/vs/community/.
- 2. Install CMake from http://www.cmake.org/download/. For building on Windows you are strongly recommended to use a recent version of CMake, e.g 3.16+. Minimum required CMake versions for building Nektar++ on Windows with Visual Studio are CMake 3.5.1+ (VS2015), 3.7+ (VS2017), or 3.15+ (VS2019). When prompted, select the option to add CMake to the system PATH.
- 3. (Optional) As highlighted above, it is possible to have Boost built from source as a third-party library during the Nektar++ build. However, it is currently recommended to install the Boost binaries that can be found at http://sourceforge.net/projects/boost/files/boost-binaries. By default these install into C:\local\boost_<version>. We recommend installing a specific version of the binaries depending on the version of Visual Studio you are using, these are known to be working with the Nektar++ build:

- For Visual Studio 2015, install boost 1.61 using the package boost_1_61_0msvc-14.0-64.exe from http://sourceforge.net/projects/boost/files/ boost-binaries/1.61.0/
- For Visual Studio 2017, install boost 1.68 using the package boost_1_68_0msvc-14.1-64.exe from http://sourceforge.net/projects/boost/files/ boost-binaries/1.68.0/
- For Visual Studio 2019, install boost 1.72 using the package boost_1_72_0msvc-14.2-64.exe from http://sourceforge.net/projects/boost/files/ boost-binaries/1.72.0/

If you use these libraries, you will need to:

- Add a BOOST_HOME environment variable. To do so, click the Start menu and type 'env', you should be presented with an "Edit the system environment variables" option. Alternatively, from the Start menu, navigate to Settings > System > About > System info (under Related Settings on the right hand panel), select Advanced System Settings, and in the Advanced tab click the Environment Variables button. In the System variables box, click New. In the New System Variable window, type BOOST_HOME next to Variable name and C:\local\<boost_dir> next to Variable value, where <boost_dir> corresponds to the directory that boost has been installed to, based on the boost version you have installed. (e.g. boost_1_61_0, boost_1_68_0, boost_1_72_0).
- 4. (Optional) Install Git for Windows from https://gitforwindows.org/ to use the development versions of Nektar++. You can accept the default set of components in the Select Components panel. When prompted, in the "Adjusting your PATH environment" panel, select the option "Git from the command line and also from 3rd-party software". You do not need to select the option to add Unix tools to the PATH.
- 5. If you've downloaded the source code archive (as described in Section 1.3.1), unpack nektar++-5.2.0.zip.

Note

Some Windows versions do not recognise the path of a folder which has ++ in the name. If you are not using Windows 10 and think that your Windows version cannot handle paths containing special characters, you should rename nektar++-5.2.0 to nektar-5.2.0.

6. Create a build directory within the nektar++-5.2.0 subdirectory. If you cloned the source code from the git repository, your Nektar++ subdirectory will be called nektar rather than nektar++-5.2.0

- 7. Open a Visual Studio terminal (*Developer Command Prompt for VS* [2015/2017/2019] or x64 Native Tools Command Prompt. From the Start menu, this can be found under Visual Studio [2015/2017/2019].
- 8. Change directory into the build directory and run CMake to generate the build files. You need to set the *generator* to the correct Visual Studio version using the -G switch on the command line, e.g. for VS2019:

```
cd C:\path\to\nektar\builds
cmake -G "Visual Studio 16 2019" ...
```

You can see the list of available generators using *cmake -help*. For VS2017 use "Visual Studio 15 2017 Win64" and for VS2015 use "Visual Studio 14 2015 Win64".

If you want to build a parallel version of *Nektar++* with MPI support, you need to add the *-DNEKTAR_USE_MPI=ON* switch to the cmake command, e.g.:

cmake -G "Visual Studio 16 2019" -D NEKTAR_USE_MPI=ON ..

Note

If you installed Boost binaries, as described above, you should ensure at this stage that the version of Boost that you installed has been correctly detected by CMake. You should see a number of lines of output from CMake saying - *Found boost <library name> library:* followed by paths to one or more files which should be located in the directory where you installed your Boost binaries. If you do not see this output, CMake has failed to detect the installed Boost libraries and the build process will instead try to build Boost from source as part of building Nektar++.

If you experience any issues with CMake finding pre-installed Boost, binaries ensure that you are working in a Visual Studio command prompt that was opened *after* you installed boost and set up the BOOST_HOME environment variable.

9. Assuming the configuration completes successfully and you see the message *Build* files have been written to: ..., you should now be ready to issue the build command:

msbuild INSTALL.vcxproj /p:Configuration=Release

To build in parallel with, for example, 12 processors, issue:

msbuild INSTALL.vcxproj /p:Configuration=Release /m:12

10. After the build and installation process has completed, the executables will be available in build\dist\bin.

14 Chapter 1 Installation

- 11. To use these executables, you need to modify your system PATH to include the bin directory and library directories where DLLs are stored. To do this, click the Start menu and type 'env', you should be presented with an "Edit the system environment variables" option. Alternatively, navigate to Settings > System > About > System info (under Related Settings on the right hand panel), select Advanced System Settings, and in the Advanced tab click the Environment Variables button. In the System Variables box, select Path and click Edit. Add the full paths to the following directories to the end of the list of paths shown in the "Edit environment variable" window:
 - nektar++-5.2.0\build\dist\lib\nektar++-5.2.0
 - nektar++-5.2.0\build\dist\bin
 - nektar++-5.2.0\ThirdParty
 - C:\local\boost_<boost_version>\<boost_lib_dir> where boost_<boost_version> is the directory where the boost binaries were installed to and <boost_lib_dir> is the name of the library directory within this location, e.g. lib64-msvc-14.2 or similar depending on the version of Boost binaries you installed.
- 12. To run the test suite, open a **new** command line window, change to the **build** directory, and then issue the command

ctest -C Release

1.3.5 CMake Option Reference

This section describes the main configuration options which can be set when building Nektar++. The default options should work on almost all systems, but additional features (such as parallelisation and specialist libraries) can be enabled if needed.

1.3.5.1 Components

The first set of options specify the components of the Nektar++ toolkit to compile. Some options are dependent on others being enabled, so the available options may change.

Components of the *Nektar++* package can be selected using the following options:

• NEKTAR_BUILD_DEMOS (Recommended)

Compiles the demonstration programs. These are primarily used by the regression testing suite to verify the Nektar++ library, but also provide an example of the basic usage of the framework.

• NEKTAR_BUILD_DOC

Compiles the Doxygen documentation for the code. This will be put in

\$BUILDDIR/doxygen/html

• NEKTAR_BUILD_LIBRARY (Required)

Compiles the Nektar++ framework libraries. This is required for all other options.

• NEKTAR_BUILD_PYTHON

Installs the Python wrapper to Nektar++. Requires running the following command after installing Nektar++ in order to install the Python package for the current user:

make nekpy-install-user

Alternatively, the Python package can be installed for all users by running the following command with appropriate priviledges:

make nekpy-install-system

• NEKTAR_BUILD_SOLVERS (Recommended)

Compiles the solvers distributed with the Nektar++ framework.

If enabling NEKTAR_BUILD_SOLVERS, individual solvers can be enabled or disabled. See Part III for the list of available solvers. You can disable solvers which are not required to reduce compilation time. See the NEKTAR_SOLVER_X option.

• NEKTAR_BUILD_TESTS (Recommended)

Compiles the testing program used to verify the Nektar++ framework.

• NEKTAR_BUILD_TIMINGS

Compiles programs used for timing Nektar++ operations.

• NEKTAR_BUILD_UNIT_TESTS

Compiles tests for checking the core library functions.

• NEKTAR_BUILD_UTILITIES

Compiles utilities for pre- and post-processing simulation data, including the mesh conversion and generation tool NekMesh and the FieldConvert post-processing utility.

• NEKTAR_SOLVER_X

Enable compilation of the 'X' solver.

• NEKTAR_UTILITY_X

Enable compilation of the 'X' utility.

16 Chapter 1 Installation

A number of ThirdParty libraries are required by Nektar++. There are also optional libraries which provide additional functionality. These can be selected using the following options:

• NEKTAR_USE_ARPACK

Build Nektar++ with support for ARPACK. This provides routines used for linear stability analyses. Alternative Arnoldi algorithms are also implemented directly in Nektar++.

• NEKTAR_USE_CCM

Use the ccmio library provided with the Star-CCM package for reading ccm files. This option is required as part of NekMesh if you wish to convert a Star-CCM mesh into the Nektar format. It is possible to read a Tecplot plt file from Star-CCM but this output currently needs to be converted to ascii format using the Tecplot package.

• NEKTAR_USE_CWIPI

Use the CWIPI library for enabling inter-process communication between two solvers. Solvers may also interface with third-party solvers using this package.

• NEKTAR_USE_FFTW

Build Nektar++ with support for FFTW for performing Fast Fourier Transforms (FFTs). This is used only when using domains with homogeneous coordinate directions.

• NEKTAR_USE_HDF5

Build *Nektar++* with support for HDF5. This enables input/output in the HDF5 parallel file format, which can be very efficient for large numbers of processes. HDF5 output can be enabled by using a command-line option or in the SOLVERINFO section of the XML file. This option requires that *Nektar++* be built with MPI support with NEKTAR_USE_MPI enabled and that HDF5 is compiled with MPI support.

• NEKTAR_USE_MESHGEN

Build the NekMesh utility with support for generating meshes from CAD geometries. This enables use of the OpenCascade Community Edition library, as well as Triangle and Tetgen.

• NEKTAR_USE_METIS

Build Nektar++ with support for the METIS graph partitioning library. This provides both an alternative mesh partitioning algorithm to SCOTCH for parallel simulations.

- NEKTAR_USE_MPI (Recommended)
 Build Nektar++ with MPI parallelisation. This allows solvers to be run in serial or parallel.
- NEKTAR_USE_PETSC

Build Nektar++ with support for the PETSc package for solving linear systems.

• NEKTAR_USE_PYTHON3 (Requires NEKTAR_BUILD_PYTHON)

Enables the generation of Python3 interfaces.

• NEKTAR_USE_SCOTCH (Recommended)

Build *Nektar++* with support for the SCOTCH graph partitioning library. This provides both a mesh partitioning algorithm for parallel simulations and enabled support for multi-level static condensation, so is highly recommended and enabled by default. However for systems that do not support SCOTCH build requirements (e.g. Windows), this can be disabled.

• NEKTAR_USE_SYSTEM_BLAS_LAPACK (Recommended)

On Linux systems, use the default BLAS and LAPACK library on the system. This may not be the implementation offering the highest performance for your architecture, but it is the most likely to work without problem.

• NEKTAR_USE_VTK

Build Nektar++ with support for VTK libraries. This is only needed for specialist utilities and is not needed for general use.

Note

The VTK libraries are not needed for converting the output of simulations to VTK format for visualization as this is handled internally.

The THIRDPARTY_BUILD_X options select which third-party libraries are automatically built during the *Nektar++* build process. Below are the choices of X:

• ARPACK

Library of iterative Arnoldi algorithms.

• BLAS_LAPACK

Library of linear algebra routines.

• BOOST

The *Boost* libraries are frequently provided by the operating system, so automatic compilation is not enabled by default. If you do not have Boost on your system, you can enable this to have Boost configured automatically.

18 Chapter 1 Installation

• CCMIO

I/O library for the Star-CCM+ format.

• CWIPI

Library for inter-process exchange of data between different solvers.

• FFTW

Fast-Fourier transform library.

• GSMPI

(MPI-only) Parallel communication library.

• HDF5

Hierarchical Data Format v5 library for structured data storage.

• METIS

A graph partitioning library used for mesh partitioning when Nektar++ is run in parallel.

• OCE

OpenCascade Community Edition 3D modelling library.

• PETSC

A package for the parallel solution of linear algebra systems.

• SCOTCH

A graph partitioning library used for mesh partitioning when Nektar++ is run in parallel, and reordering routines that are used in multi-level static condensation.

• TETGEN

3D tetrahedral meshing library.

• TINYXML

Library for reading and writing XML files.

• TRIANGLE

2D triangular meshing library.

There are also a number of additional options to fine-tune the build:

• NEKTAR_DISABLE_BACKUPS

By default, Nektar++ solvers and the FieldConvert utility will not overwrite any generated field files or output files they find an existing file with the same name.

Instead, the existing file will be either moved to a backup file or you will be prompted to overwrite them. If you do not want this behaviour, then enabling this option will cause all pre-existing output to be overwritten silently.

• NEKTAR_TEST_ALL

Enables an extra set of more substantial and long-running tests.

• NEKTAR_TEST_USE_HOSTFILE

By default, MPI tests are run directly with the **mpiexec** command together with the number of cores. If your MPI installation requires a hostfile, enabling this option adds the command line argument **-hostfile hostfile** to the command line arguments when tests are run with **ctest** or the **(Tester)** executable.

We have recently added explicit support to SIMD (Single Instruction Multiple Data) x86 instruction set extensions (i.e. AVX2, AVX512). Selected operators (the matrix free operators) utilize the SIMD types, if none of them is enabled these operators default to scalar types. The various extensions available are marked as advanced options (to visualize them in the cmake gui you need to press the t-button):

• NEKTAR_ENABLE_SIMD_AVX2

Enables 256 bit wide vector types and set the appropriate compiler flags (gcc only).

• NEKTAR_ENABLE_SIMD_AVX512

Enables 512 bit wide vector types and set the appropriate compiler flags (gcc only).

Note that if you are not configuring cmake for the first time, you need to delete the cached variable CMAKE_CXX_FLAGS in order for the appropriate flags to be set. Alternatively you can manually set the flag to target the appropriate architecture.

CHAPTER 2

Mathematical Formulation

2.1 Background

The spectral/hp element method combines the geometric flexibility of classical *h*-type finite element techniques with the desirable resolution properties of spectral methods. In this approach a polynomial expansion of order P is applied to every elemental domain of a coarse finite element type mesh. These techniques have been applied in many fundamental studies of fluid mechanics [44] and more recently have gained greater popularity in the modelling of wave-based phenomena such as computational electromagnetics [18] and shallow water problems [5] - particularly when applied within a Discontinuous Galerkin formulation.

There are at least two major challenges which arise in developing an efficient implementation of a spectral/hp element discretisation:

- implementing the mathematical structure of the technique in a digestible, generic and coherent manner, and
- designing and implementing the numerical methods and data structures in a matter so that both high- and low-order discretisations can be efficiently applied.

In order to design algorithms which are efficient for both low- and high-order spectral/hp discretisations, it is important clearly define what we mean with low- and high-order. The spectral/hp element method can be considered as bridging the gap between the high-order end of the traditional finite element method and low-order end of conventional spectral methods. However, the concept of high- and low-order discretisations can mean very different things to these different communities. For example, the seminal works by Zienkiewicz & Taylor [50] and Hughes list examples of elemental expansions only up to third or possibly fourth-order, implying that these orders are considered to be high-order for the traditional h-type finite element community. In contrast the text books of the spectral/hp element community typically show examples of problems ranging from a

low-order bound of minimally fourth-order up to anything ranging from 10^{th} -order to 15^{th} -order polynomial expansions. On the other end of the spectrum, practitioners of global (Fourier-based) spectral methods [16] would probably consider a 16^{th} -order global expansion to be relatively low-order approximation.

One could wonder whether these different definitions of low- and high-order are just inherent to the tradition and lore of each of the communities or whether there are more practical reasons for this distinct interpretation. Proponents of lower-order methods might highlight that some problems of practical interest are so geometrically complex that one cannot computationally afford to use high-order techniques on the massive meshes required to capture the geometry. Alternatively, proponents of high-order methods highlight that if the problem of interest can be captured on a computational domain at reasonable cost then using high-order approximations for sufficiently *smooth* solutions will provide a higher accuracy for a given computational cost. If one however probes even further it also becomes evident that the different communities choose to implement their algorithms in different manners. For example the standard h-type finite element community will typically uses techniques such as sparse matrix storage formats (where only the non-zero entries of a global matrix are stored) to represent a global operator. In contrast the spectral/hp element community acknowledges that for higher polynomial expansions more closely coupled computational work takes place at the individual elemental level and this leads to the use of elemental operators rather than global matrix operators. In addition the global spectral method community often make use of the tensor-product approximations where products of one-dimensional rules for integration and differentiation can be applied.

2.2 Methods overview

Here a review of some terminology in order to situate the spectral/hp element method within the field of the finite element methods.

2.2.1 The finite element method (FEM)

Nowadays, the finite element method is one of the most popular numerical methods in the field of both solid and fluid mechanics. It is a discretisation technique used to solve (a set of) partial differential equations in its equivalent variational form. The classical approach of the finite element method is to partition the computational domain into a mesh of many small subdomains and to approximate the unknown solution by piecewise linear interpolation functions, each with local support. The FEM has been widely discussed in literature and for a complete review of the method, the reader is also directed to the seminal work of Zienkiewicz and Taylor [50].

2.2.2 High-order finite element methods

While in the classical finite element method the solution is expanded in a series of linear basis functions, high-order FEMs employ higher-order polynomials to approximate the

22 Chapter 2 Mathematical Formulation

solution. For the high-order FEM, the solution is locally expanded into a set of P + 1 linearly independent polynomials which span the polynomial space of order P. Confusion may arise about the use of the term *order*. While the order, or *degree*, of the expansion basis corresponds to the maximal polynomial degree of the basis functions, the order of the method essentially refers to the accuracy of the approximation. More specifically, it depends on the convergence rate of the approximation with respect to mesh-refinement. It has been shown by Babuska and Suri [3], that for a sufficiently smooth exact solution $u \in H^k(\Omega)$, the error of the FEM approximation u^{δ} can be bounded by:

$$||u - u^{\delta}||_E \le Ch^P ||u||_k.$$

This implies that when decreasing the mesh-size h, the error of the approximation algebraically scales with the P^{th} power of h. This can be formulated as:

$$||u - u^{\delta}||_E = O(h^P).$$

If this holds, one generally classifies the method as a P^{th} -order FEM. However, for non-smooth problems, i.e. k < P + 1, the order of the approximation will in general be lower than P, the order of the expansion.

2.2.2.1 h-version FEM

A finite element method is said to be of h-type when the degree P of the piecewise polynomial basis functions is fixed and when any change of discretisation to enhance accuracy is done by means of a mesh refinement, that is, a reduction in h. Dependent on the problem, local refinement rather than global refinement may be desired. The h-version of the classical FEM employing linear basis functions can be classified as a first-order method when resolving smooth solutions.

2.2.2.2 p-version FEM

In contrast with the *h*-version FEM, finite element methods are said to be of *p*-type when the partitioning of domain is kept fixed and any change of discretisation is introduced through a modification in polynomial degree *P*. Again here, the polynomial degree may vary per element, particularly when the complexity of the problem requires local enrichment. However, sometimes the term *p*-type FEM is merely used to indicated that a polynomial degree of P > 1 is used.

2.2.2.3 hp-version FEM

In the hp-version of the FEM, both the ideas of mesh refinement and degree enhancement are combined.

2.2.2.4 The spectral method

As opposed to the finite element methods which builds a solution from a sequence of local elemental approximations, spectral methods approximate the solution by a truncated series of global basis functions. Modern spectral methods, first presented by Gottlieb and Orzag [16], involve the expansion of the solution into high-order orthogonal expansion, typically by employing Fourier, Chebyshev or Legendre series.

2.2.2.5 The spectral element method

Patera [37] combined the high accuracy of the spectral methods with the geometric flexibility of the finite element method to form the spectral element method. The multielemental nature makes the spectral element method conceptually similar to the above mentioned high-order finite element. However, historically the term spectral element method has been used to refer to the high-order finite element method using a specific nodal expansion basis. The class of nodal higher-order finite elements which have become known as spectral elements, use the Lagrange polynomials through the zeros of the Gauss-Lobatto(-Legendre) polynomials.

2.2.2.6 The spectral/hp element method

The spectral/hp element method, as its name suggests, incorporates both the multidomain spectral methods as well as the more general high-order finite element methods. One can say that it encompasses all methods mentioned above. However, note that the term spectral/hp element method is mainly used in the field of fluid dynamics, while the terminology p and hp-FEM originates from the area of structural mechanics.

2.2.3 The Galerkin formulation

Finite element methods typically use the Galerkin formulation to derive the weak form of the partial differential equation to be solved. We will primarily adopt the classical Galerkin formulation in combination with globally C^0 continuous spectral/hp element discretisations.

To describe the Galerkin method, consider a steady linear differential equation in a domain Ω denoted by

$$L(u) = f,$$

subject to appropriate boundary conditions. In the Galerkin method, the weak form of this equation can be derived by pre-multiplying this equation with a test function v and integrating the result over the entire domain Ω to arrive at: Find $u \in \mathcal{U}$ such that

$$\int_{\Omega} v L(u) d\boldsymbol{x} = \int_{\Omega} v f d\boldsymbol{x}, \quad \forall v \in \mathcal{V},$$

where \mathcal{U} and \mathcal{V} respectively are a suitably chosen trial and test space (in the traditional Galerkin method, one typically takes $\mathcal{U} = \mathcal{V}$). In case the inner product of v and $\mathbb{L}(u)$

can be rewritten into a bi-linear form a(v, u), this problem is often formulated more concisely as: Find $u \in \mathcal{U}$ such that

$$a(v,u) = (v,f), \quad \forall v \in \mathcal{V},$$

where (v, f) denotes the inner product of v and f. The next step in the classical Galerkin finite element method is the discretisation: rather than looking for the solution u in the infinite dimensional function space \mathcal{U} , one is going to look for an approximate solution u^{δ} in the reduced finite dimensional function space $\mathcal{U}^{\delta} \subset \mathcal{U}$. Therefore we represent the approximate solution as a linear combination of basis functions Φ_n that span the space \mathcal{U}^{δ} , i.e.

$$u^{\delta} = \sum_{n \in \mathcal{N}} \Phi_n \hat{u}_n.$$

Adopting a similar discretisation for the test functions v, the discrete problem to be solved is given as: Find \hat{u}_n $(n \in \mathcal{N})$ such that

$$\sum_{n \in \mathcal{N}} a(\Phi_m, \Phi_n) \hat{u}_n = (\Phi_m, f), \quad \forall m \in \mathcal{N}.$$

It is customary to describe this set of equations in matrix form as

$$A\hat{u}=\hat{f},$$

where $\hat{\boldsymbol{u}}$ is the vector of coefficients \hat{u}_n , \boldsymbol{A} is the system matrix with elements

$$\boldsymbol{A}[m][n] = a(\Phi_m, \Phi_n) = \int_{\Omega} \Phi_m L(\Phi_n) d\boldsymbol{x},$$

and the vector \hat{f} is given by

$$\hat{\boldsymbol{f}}[m] = (\Phi_m, f) = \int_{\Omega} \Phi_m f d\boldsymbol{x}.$$
CHAPTER **3**

XML Session File

The Nektar++ native file format is compliant with XML version 1.0. The root element is NEKTAR which contains a number of other elements which describe configuration for different aspects of the simulation. The required elements are shown below:

```
1 <NEKTAR>
2 <GEOMETRY>
3
     . . .
   </GEOMETRY>
4
5 <EXPANSIONS>
6
     . . .
   </EXPANSIONS>
7
    <CONDITIONS>
8
9
     . . .
10
    </CONDITIONS>
11
   . . .
12 </NEKTAR>
```

The different sub-elements can be split across multiple files, however each file must have a top-level NEKTAR tag. For example, one might store the geometry information separate from the remaining configuration in two separate files as illustrated below:

geometry.xml

```
      1
      <NEKTAR>

      2
      <GEOMETRY>

      3
      ...

      4
      </GEOMETRY>

      5
      </NEKTAR>
```

conditions.xml

```
1 <NEKTAR>

2 <CONDITIONS>

3 ...

4 </CONDITIONS>

5 <EXPANSIONS>
```

```
6 ....
7 </EXPANSIONS>
8 ...
9 </NEKTAR>
```

Note

When specifying multiple files, repeated first-level XML sub-elements are not merged. The sub-elements from files appearing later in the list will, in general, override those elements from earlier files.

For example, the NekMesh utility will produce a default (EXPANSIONS) element and blank (CONDITIONS) element. Specifying a custom-written XML file containing these sections *after* the file produced by NekMesh will override these defaults.

The exception to this rule is when an empty XML sub-element would override a non-empty XML sub-element. In this case the empty XML sub-element will be ignored. If the custom-written XML file containing **CONDITIONS** were specified before the file produced by **NekMesh**, the empty **CONDITIONS** tag in the latter file would be ignored.

3.1 Geometry

This section defines the mesh. It specifies a list of vertices, edges (in two or three dimensions) and faces (in three dimensions) and how they connect to create the elemental decomposition of the domain. It also defines a list of composites which are used in the Expansions and Conditions sections of the file to describe the polynomial expansions and impose boundary conditions.

The GEOMETRY section is structured as

```
1 <GEOMETRY DIM="2" SPACE="2">
2 <VERTEX> ... </VERTEX>
3 <EDGE> ... </EDGE>
4 <FACE> ... </FACE>
5 <ELEMENT> ... </FACE>
6 <CURVED> ... </CURVED>
7 <COMPOSITE> ... </COMPOSITE>
8 <DOMAIN> ... </DOMAIN>
9 </GEOMETRY>
```

It has two (required) attributes:

- (DIM) specifies the dimension of the expansion elements.
- **(SPACE)** specifies the dimension of the space in which the elements exist.

These attributes allow, for example, a two-dimensional surface to be embedded in a three-dimensional space.

Note

The attribute (PARTITION) may also appear in a partitioned mesh. However, this attribute should not be explicitly specified by the user.

The contents of each of the (VERTEX), (EDGE), (FACE), (ELEMENT) and (CURVED) sections may optionally be compressed and stored in base64-encoded gzipped binary form, using either little-endian or big-endian ordering, as specified by the (COMPRESSED) attribute to these sections. Currently supported values are:

- B64Z-LittleEndian: Base64 Gzip compressed using little-endian ordering.
- B64Z-BigEndian : Base64 Gzip compressed using big-endian ordering.

When generating mesh input files for Nektar++ using NekMesh, the binary compressed form will be used by default. To convert a compressed XML file into human-readable ASCII format use, for example:

NekMesh file.msh newfile.xml:xml:uncompress



The description in the remainder of this section explains how the **GEOMETRY** section is laid out in uncompressed ASCII format.

3.1.1 Vertices

Note

Vertices have three coordinates. Each has a unique vertex ID. In uncompressed form, they are defined within **VERTEX** subsection as follows:

```
1 <V ID="0"> 0.0 0.0 0.0 </V> ...
```

The **VERTEX** subsection has optional attributes which can be used to apply a transformation to the mesh:

```
(XSCALE), (YSCALE), (ZSCALE), (XMOVE), (YMOVE), (ZMOVE)
```

They specify scaling factors (centred at the origin) and translations to the vertex coordinates. For example, the following snippet

```
1 <VERTEX XSCALE="5">

2 <V ID="0"> 0.0 0.0 0.0 </V>

3 <V ID="1"> 1.0 2.0 0.0 </V>

4 </VERTEX>
```

defines two vertices with coordinates (0.0, 0.0, 0.0), (1.0, 2.0, 0.0).

All of these attributes can be arbitrary analytic expressions depending on pre- defined constants and parameters defined in the XML file and mathematical operations/functions of the latter. If omitted, default scaling factors 1.0, and translations of 0.0, are assumed.

3.1.2 Edges

Tip

The EDGES section is only necessary when DIM=2 or DIM=3 in the parent GEOMETRY element and may be omitted for one-dimensional meshes.

Edges are defined by two vertices. Each edge has a unique edge ID. In uncompressed form, they are defined in the file with a line of the form

1 <E ID="0"> 0 1 </E>

3.1.3 Faces



The **FACES** section is only necessary when **DIM=3** in the parent **GEOMETRY** element and may otherwise be omitted.

Faces are defined by three or more edges. Each face has a unique face ID. They are defined in the file with a line of the form

1 <T ID="0"> 0 1 2 </T> 2 <Q ID="1"> 3 4 5 6 </Q>

The choice of tag specified (T or Q), and thus the number of edges specified depends on the geometry of the face (triangle or quadrilateral).

3.1.4 Element

Elements define the top-level geometric entities in the mesh. Their definition depends upon the dimension of the expansion. For two-dimensional expansions, an element is defined by a sequence of three or four edges. For three-dimensional expansions, the element is defined by a list of faces. Elements are defined in the file with a line of the form

1 <T ID="0"> 0 1 2 </T> 2 <H ID="1"> 3 4 5 6 7 8 </H>

Again, the choice of tag specified depends upon the geometry of the element. The element tags are:

- S Segment
- T Triangle
- **Q** Quadrilateral
- A Tetrahedron
- P Pyramid
- R Prism
- (H) Hexahedron

3.1.5 Curved Edges and Faces



The CURVED section is only necessary if curved edges or faces are present in the mesh and may otherwise be omitted.

For mesh elements with curved edges and/or curved faces, a separate entry is used to describe the control points for the curve. Each curve has a unique curve ID and is associated with a predefined edge or face. The total number of points in the curve (including end points) and their distribution is also included as attributes. The control points are listed in order, each specified by three coordinates. Curved edges are defined in the file with a line of the form

```
1 <E ID="3" EDGEID="7" TYPE="PolyEvenlySpaced" NUMPOINTS="3">
2 0.0 0.0 0.0 0.5 0.5 0.0 1.0 0.0 0.0
3 </E>
```

Note

In the compressed form, this section contains different sub-elements to efficiently encode the high-order curvature data. This is not described further in this document.

3.1.6 Composites

Composites define collections of elements, faces or edges. Each has a unique composite ID associated with it. All components of a composite entry must be of the same type. The syntax allows components to be listed individually, using ranges, or a mixture of the two. Examples include

1 <C ID="0"> T[0-862] </C> 2 <C ID="1"> E[61-67,69,70,72-74] </C> The composites can also optionally contain a name which is then used in the multi-block VTK output to label the block descriptively rather than by ID, for example

```
1 <C NAME="Main domain" ID="0"> T[0-862] </C>
2 <C NAME="Walls" ID="1"> E[61-67,69,70,72-74] </C>
```

3.1.7 Domain

This tag specifies composites which describe the entire problem domain. It has the form of

1 <DOMAIN> C[0] </DOMAIN>

3.2 Expansions

This section defines the polynomial expansions used on each of the defined geometric composites and variables. Expansion entries specify the number of modes and the expansion type, or a full list of data of basis type, number of modes, points type and number of points. The short-hand version has the following form

```
1 <E COMPOSITE="C[0]" NUMMODES="5" FIELDS="u" TYPE="MODIFIED" />
```

or, if we have more then one variable we can apply the same basis to all using

```
1 <E COMPOSITE="C[0]" NUMMODES="5" FIELDS="u,v,p" TYPE="MODIFIED" />
```

The expansion basis can also be specified in detail as a combination of one-dimensional bases, and thus the user is able to, for example, increase the quadrature order. For tet elements this takes the form:

```
1 <E COMPOSITE="C[0]"
2 BASISTYPE="Modified_A,Modified_B,Modified_C"
3 NUMMODES="3,3,3"
4 POINTSTYPE="GaussLobattoLegendre,GaussRadauMAlpha1Beta0,GaussRadauMAlpha2Beta0"
5 NUMPOINTS="4,3,3"
6 FIELDS="u" />
```

and for prism elements:

```
1 <E COMPOSITE="C[1]"
2 BASISTYPE="Modified_A,Modified_B"
3 NUMMODES="3,3,3"
4 POINTSTYPE="GaussLobattoLegendre,GaussLobattoLegendre,GaussRadauMAlpha1Beta0"
5 NUMPOINTS="4,4,3"
6 FIELDS="u" />
```

The expansions can be defined with a list of $\langle E \rangle$ elements (e.g., to represent different polynomial orders for different variables or to address different composites). The user can define a default expansion field by entering $\langle E \rangle$ tags without the FIELDS attribute. The default expansion is used to define any variables not explicitly listed in the $\langle E \rangle$

entries. In the following example, the default expansion is used to define the expansions for the composites C[0], C[1] and C[2]:

```
1 <E COMPOSITE="C[0-2]" NUMMODES="5" TYPE="MODIFIED" />
2 <E COMPOSITE="C[3]" NUMMODES="4" TYPE="MODIFIED" FIELDS="u,v"/>
3 <E COMPOSITE="C[3]" NUMMODES="3" TYPE="MODIFIED" FIELDS="p"/>
```

The expansions of each field should be defined only once for each composite.

3.3 Conditions

This section of the file defines parameters and boundary conditions which define the nature of the problem to be solved. These are enclosed in the **CONDITIONS** tag.

3.3.1 Parameters

Numerical parameters may be required by a particular solver (for instance time-integration or physical parameters), or may be arbitrary and only used for the purpose of simplifying the problem specification in the session file (e.g. parameters which would otherwise be repeated in the definition of an initial condition and boundary conditions). All parameters are enclosed in the (PARAMETERS) XML element.

```
1 <PARAMETERS>
2 ...
3 </PARAMETERS>
```

A parameter may be of integer or real type and may reference other parameters defined previous to it. It is expressed in the file as

1 <P> [PARAMETER NAME] = [PARAMETER VALUE] </P>

For example,

1 <P> NumSteps = 1000 </P> 2 <P> TimeStep = 0.01 </P> 3 <P> FinTime = NumSteps*TimeStep </P>

A number of pre-defined constants may also be used in parameter expressions, for example PI. A full list of supported constants is provided in Section 3.7.1.2.

3.3.2 Time Integration Scheme

These specify properties to define the parameters specific to the time integration scheme to be used. The parameters are specified as XML elements and require a string corresponding to the time-stepping method and the order, and optionally the variant and free parameters. For example,

```
1 <TIMEINTEGRATIONSCHEME>
2 <METHOD> IMEX </METHOD>
```

```
3 <VARIANT> DIRK </VARIANT>
```

```
4 <ORDER> 2 </ORDER>
5 <FREEPARAMETERS> 2 3 </FREEPARAMETERS>
6 </TIMEINTEGRATIONSCHEME>
```

For additional details on the different time integration schemes refer to the developer's guide.

3.3.3 Solver Information

These specify properties to define the actions specific to solvers, typically including the equation to solve and the projection type. The property/value pairs are specified as XML attributes. For example,

```
      1 <SOLVERINFO>

      2 <I PROPERTY="EQTYPE"</td>
      VALUE="UnsteadyAdvection"
      />

      3 <I PROPERTY="Projection"</td>
      VALUE="Continuous"
      />

      4 </SOLVERINFO>
```

Boolean-valued solver properties are specified using **True** or **False**. The list of available solvers in Nektar++ can be found in Part III.

3.3.3.1 Drivers

Drivers are defined under the **CONDITIONS** section as properties of the **SOLVERINFO** XML element. The role of a driver is to manage the solver execution from an upper level.

The default driver is called **Standard** and executes the following steps:

- 1. Prints out on screen a summary of all the conditions defined in the input file.
- 2. Sets up the initial and boundary conditions.
- 3. Calls the solver defined by SolverType in the SOLVERINFO XML element.
- 4. Writes the results in the output (.fld) file.

In the following example, the driver **Standard** is used to manage the execution of the incompressible Navier-Stokes equations:

```
1
   <TIMEINTEGRATIONSCHEME>
\mathbf{2}
      <METHOD> IMEX </METHOD>
3
      <ORDER> 2 </ORDER>
  </TIMEINTEGRATIONSCHEME>
4
5
6 <SOLVERINFO>
      <I PROPERTY="EQTYPE"
                                            VALUE="UnsteadyNavierStokes"
                                                                                />
7
      <I PROPERTY="SolverType"
                                            VALUE="VelocityCorrectionScheme"
                                                                                1>
8
      <I PROPERTY="Projection"
                                            VALUE="Galerkin"
9
                                                                                />
      <I PROPERTY="Driver"
                                             VALUE="Standard"
                                                                                />
10
11 </SOLVERINFO>
```

If no driver is specified in the session file, the driver **Standard** is called by default. Other drivers can be used and are typically focused on specific applications. As described in Sec. 11.3.1 and 11.4.1, the other possibilities are:

- ModifiedArnoldi computes of the leading eigenvalues and eigenmodes using modified Arnoldi method.
- Arpack computes of eigenvalues/eigenmodes using Implicitly Restarted Arnoldi Method (ARPACK).
- **SteadyState** uses the Selective Frequency Damping method (see Sec. 11.1.5) to obtain a steady-state solution of the Navier-Stokes equations (compressible or incompressible).

3.3.4 Variables

These define the number (and name) of solution variables. Each variable is prescribed a unique ID. For example a two-dimensional flow simulation may define the velocity variables using

1 <VARIABLES> 2 <V ID="0"> u </V> 3 <V ID="1"> v </V> 4 </VARIABLES>

3.3.5 Global System Solution Algorithm

Many Nektar++ solvers use an implicit formulation of their equations to, for instance, improve timestep restrictions. This means that a large matrix system must be constructed and a global system set up to solve for the unknown coefficients. There are several approaches in the spectral/hp element method that can be used in order to improve efficiency in these methods, as well as considerations as to whether the simulation is run in parallel or serial. Nektar++ opts for 'sensible' default choices, but these may or may not be optimal depending on the problem under consideration.

This section of the XML file therefore allows the user to specify the global system solution parameters, which dictates the type of solver to be used for any implicit systems that are constructed. This section is particularly useful when using a multi-variable solver such as the incompressible Navier-Stokes solver, as it allows us to select different preconditioning and residual convergence options for each variable. As an example, consider the block defined by:

```
1 <GLOBALSYSSOLNINFO>
2 <V VAR="u,v,w">
3 <I PROPERTY="GlobalSysSoln" VALUE="IterativeStaticCond" />
4 <I PROPERTY="Preconditioner" VALUE="LowEnergyBlock"/>
5 <I PROPERTY="IterativeSolverTolerance" VALUE="1e-8"/>
6 </V>
7 <V VAR="p">
```

```
      8
      <I PROPERTY="GlobalSysSoln"</td>
      VALUE="IterativeStaticCond" />

      9
      <I PROPERTY="Preconditioner"</td>
      VALUE="FullLinearSpaceWithLowEnergyBlock"/>

      10
      <I PROPERTY="IterativeSolverTolerance"</td>
      VALUE="1e-6"/>

      11

      12
```

The above section specifies that the variables u, v, w should use the IterativeStaticCond global solver alongside the LowEnergyBlock preconditioner and an iterative tolerance of 10^{-8} on the residuals. However the pressure variable p is generally stiffer: we therefore opt for a more expensive FullLinearSpaceWithLowEnergyBlock preconditioner and a larger residual of 10^{-6} . We now outline the choices that one can use for each of these parameters and give a brief description of what they mean.

Defaults for all fields can be defined by setting the equivalent property in the SOLVERINFO section. Parameters defined in this section will override any options specified there.

3.3.5.1 GlobalSysSoln options

Nektar++ presently implements four methods of solving a global system:

- **Direct** solvers construct the full global matrix and directly invert it using an appropriate matrix technique, such as Cholesky factorisation, depending on the properties of the matrix. Direct solvers **only** run in serial.
- **Iterative** solvers instead apply matrix-vector multiplications repeatedly, using the conjugate gradient method, to converge to a solution to the system. For smaller problems, this is typically slower than a direct solve. However, for larger problems it can be used to solve the system in parallel execution.
- Xxt solvers use the XX^T library to perform a parallel direct solve. This option is only available if the NEKTAR_USE_MPI option is enabled in the CMake configuration.
- **PETSc** solvers use the PETSc library, giving access to a wide range of solvers and preconditioners. See section 3.3.5.4 below for some additional information on how to use the PETSc solvers. This option is only available if the NEKTAR_USE_PETSC option is enabled in the CMake configuration.

Warning

Both the **Xxt** and **PETSc** solvers are considered advanced and are under development – either the direct or iterative solvers are recommended in most scenarios.

These solvers can be run in one of three approaches:

- The **Full** approach constructs the global system based on all of the degrees of freedom contained within an element. For most of the Nektar++ solvers, this technique is not recommended.
- The **StaticCond** approach applies a technique called *static condensation* to instead construct the system using only the degrees of freedom on the boundaries of the elements, which reduces the system size considerably. This is the **default option in parallel**.
- **MultiLevelStaticCond** methods apply the static condensation technique repeatedly to further reduce the system size, which can improve performance by 25-30% over the normal static condensation method. It is therefore the **default option in serial**. Note that whilst parallel execution technically works, this is under development and is likely to be slower than single-level static condensation: this is therefore not recommended.

The GlobalSysSoln option is formed by combining the method of solution with the approach: for example IterativeStaticCond or PETScMultiLevelStaticCond.

3.3.5.2 Preconditioner options

Preconditioners can be used in the iterative and PETSc solvers to reduce the number of iterations needed to converge to the solution. There are a number of preconditioner choices, the default being a simple Jacobi (or diagonal) preconditioner, which is enabled by default. There are a number of choices that can be enabled through this parameter, which are all generally discretisation and dimension-dependent:

Name	Dimensions	Discretisations
Null	All	All
Diagonal	All	All
FullLinearSpace	2/3D	CG
LowEnergyBlock	3D	CG
Block	2/3D	All
FullLinearSpaceWithDiagonal	All	CG
FullLinearSpaceWithLowEnergyBlock	2/3D	CG
FullLinearSpaceWithBlock	2/3D	CG

For a detailed discussion of the mathematical formulation of these options, see the developer guide.

3.3.5.3 SuccessiveRHS options

The SuccessiveRHS option can be used in the iterative solver only, to attempt to reduce the number of iterations taken to converge to a solution. It stores a number of previous solutions or right-hand sides, dictated by the setting of the SuccessiveRHS option, to give a better initial guess for the iterative process. This method is better than any linear extrapolation method.

It can be activated by setting

```
1 <GLOBALSYSSOLNINFO>
\mathbf{2}
      <V VAR="u,v,w">
          <I PROPERTY="GlobalSysSoln"
                                                VALUE="IterativeStaticCond" />
3
4
          <I PROPERTY="Preconditioner"</pre>
                                              VALUE="LowEnergyBlock"/>
          <I PROPERTY="SuccessiveRHS" VALUE="8" />
\mathbf{5}
          <I PROPERTY="IterativeSolverTolerance"</pre>
6
                                                      VALUE="1e-4"/>
      </V>
7
      <V VAR="p">
8
9
          <I PROPERTY="GlobalSysSoln"
                                                  VALUE="IterativeStaticCond" />
          <I PROPERTY="Preconditioner"
                                                 VALUE="LowEnergyBlock"/>
10
          <I PROPERTY="SuccessiveRHS"
                                                VALUE="8" />
11
12
          <I PROPERTY="IterativeSolverTolerance"</pre>
                                                      VALUE="1e-4"/>
13
      </V>
14 </GLOBALSYSSOLNINFO>
```

or

1 <PARAMETERS> 2 <P> SuccessiveRHS = 8 </P> 3 </PARAMETERS>

The typical value of SuccessiveRHS is ≤ 10 .

The linear problem to be solved is

$$Ax = b, (3.1)$$

here x and b are both column vectors. There are a sequence of already solved linear problems

$$Ax_n = b_n, n = 1, 2, \dots, J.$$
(3.2)

Assume x_n are all linearly independent. In the successive right-hand method (see [13]), the best approximation to x is

$$\tilde{x} = \sum_{n=1}^{J} \alpha_n x_n \tag{3.3}$$

which is found by minimizing the object function

$$Q_1 = (A(\tilde{x} - x))^T A(\tilde{x} - x), \qquad (3.4)$$

or

$$Q_2 = (\tilde{x} - x)^T A (\tilde{x} - x).$$
(3.5)

If Q_1 is used, the projection bases are $e_m = b_m, m = 1, 2, ..., J$. Using

$$(A(\tilde{x} - x))^T = \sum_{m=1}^J \alpha_m e_m^T - b^T,$$
(3.6)

there is

$$Q_1 = \sum_{m=1}^{J} \sum_{n=1}^{J} \alpha_m \alpha_n e_m^T b_n - 2 \sum_{m=1}^{J} \alpha_m e_m^T b + b^T b.$$
(3.7)

To minimize Q_1 , there should be $\partial Q_1/\partial \alpha_m = 0, m = 1, 2, ..., J$. The corresponding linear problem is

$$M(\alpha_1, \alpha_2, ..., \alpha_J)^T = (e_1^T b, e_2^T b, ..., e_J^T b)^T,$$
(3.8)

with symmetric positive definite coefficient matrix $M_{mn} = e_m^T b_n$. In this case, both the solutions x_m and the right-hand sides b_m need to be stored.

If Q_2 is used, A should be a symmetric positive definite matrix, as those encountered in the Poisson equation and the Helmholtz equation. Here, the projection bases are $\hat{e}_m = x_m, m = 1, 2, ..., J$. Using

$$(\tilde{x} - x)^T = \sum_{m=1}^J \alpha_n \hat{e}_m^T - x^T,$$
(3.9)

there is

$$Q_2 = \sum_{m=1}^{J} \sum_{n=1}^{J} \alpha_m \alpha_n \hat{e}_m^T b_n - 2 \sum_{m=1}^{J} \alpha_m \hat{e}_m^T b + x^T b.$$
(3.10)

To minimize Q_2 , there should be $\partial Q_2/\partial \alpha_m = 0, m = 1, 2, ..., J$. The corresponding linear problem is

$$M(\alpha_1, \alpha_2, ..., \alpha_J)^T = (\hat{e}_1^T b, \hat{e}_2^T b, ..., \hat{e}_J^T b)^T,$$
(3.11)

with symmetric positive definite coefficient matrix $M_{mn} = \hat{e}_m^T b_n$. In this case, only the solutions x_m need to be stored.

The formulations of Q_1 version and Q_2 version are the same, except the difference of projection bases. By default, Q_2 is used as the object function. If you want to use Q_1 instead, you can assign a negative value to SuccessiveRHS:

```
1 <I PROPERTY="SuccessiveRHS" VALUE="-8" />
```

or

1 <P> SuccessiveRHS = -8 </P>

In the original paper of Fischer (1998) [13], the Gram-Schmidt orthogonal process is applied to the projection bases, this method is very stable and avoids the calculation of M^{-1} . However, when the memory space is full, this approach makes it hard to decide which basis should be overwritten. In our implementation, we just store the normalized

right-hand sides or old solutions, i.e. $e_m^T b_m = 1$ or $\hat{e}_m^T b_m = 1$, and overwrite the oldest ones. The linear problem (3.8) or (3.11) is solved using a direct method.

To make sure M is positive definite, when a new basis e_{J+1} arrives, we test the following condition to decide whether or not to accept it,

$$r = (-y^T \tilde{M}^{-1}, 1) \begin{pmatrix} \tilde{M} & y \\ y^T & 1 \end{pmatrix} \begin{pmatrix} -\tilde{M}^{-1}y \\ 1 \end{pmatrix} = 1 - y^T \tilde{M}^{-1}y \ge \varepsilon > 0.$$
(3.12)

Assuming e_J is the oldest basis, there are $\tilde{M}_{mn} = M_{mn}, y_m = e_m^T b_{J+1}, (m, n = 1, 2, ..., J - 1).$

3.3.5.4 PETSc options and configuration

The PETSc solvers, although currently experimental, are operational both in serial and parallel. PETSc gives access to a wide range of alternative solver options such as GMRES, as well as any packages that PETSc can link against, such as the direct multi-frontal solver MUMPS.

Configuration of PETSc options using its command-line interface dictates what matrix storage, solver type and preconditioner should be used. This should be specified in a .petscrc file inside your working directory, as command line options are not currently passed through to PETSc to avoid conflict with *Nektar++* options. As an example, to select a GMRES solver using an algebraic multigrid preconditioner, and view the residual convergence, one can use the configuration:

```
-ksp_monitor
-ksp_view
-ksp_type gmres
-pc_type gamg
```

Or to use MUMPS, one could use the options:

```
-ksp_type preonly
-pc_type lu
-pc_factor_mat_solver_package mumps
-mat_mumps_icntl_7 2
```

A final choice that can be specified is whether to use a *shell* approach. By default, *Nektar++* will construct a PETSc sparse matrix (or whatever matrix is specified on the command line). This may, however, prove suboptimal for higher order discretisations. In this case, you may choose to use the *Nektar++* matrix-vector operators, which by default use an assembly approach that can prove faster, by setting the PETScMatMult SOLVERINFO option to Shell:

```
1 <I PROPERTY="PETScMatMult" VALUE="Shell" />
```

The downside to this approach is that you are now constrained to using one of the Nektar++ preconditioners. However, this does give access to a wider range of Krylov methods than are available inside Nektar++ for more advanced users.

3.3.6 Boundary Regions and Conditions

Boundary conditions are defined by two XML elements. The first defines the boundary regions in the domain in terms of composite entities from the **GEOMETRY** section of the file. Each boundary region has a unique ID and are defined as,

```
1 <BOUNDARYREGIONS>
2   <B ID=[id]> [composite-list] </B>
3   ...
4 </BOUNDARYREGIONS>
```

For example,

```
1 <BOUNDARYREGIONS>

2 <B ID="0"> C[2] </B>

3 <B ID="1"> C[3] </B>

4 </BOUNDARYREGIONS>
```

The boundary regions can also optionally contain a name which is then used in the multi-block VTK output to label the block descriptively rather than by ID, for example

```
1 <BOUNDARYREGIONS>
2 <B ID="0" NAME="Wall"> C[2] </B>
3 <B ID="1" NAME="Farfield"> C[3] </B>
4 </BOUNDARYREGIONS>
```

The second XML element defines, for each variable, the condition to impose on each boundary region, and has the form,

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="[regionID]">
3 <[type1] VAR="[variable1]" VALUE="[expression1]" />
4 ...
5 <[typeN] VAR="[variableN]" VALUE="[expressionN]" />
6 </REGION>
7 ...
8 </BOUNDARYCONDITIONS>
```

There should be precisely one **REGION** entry for each **B** entry defined in the **BOUNDARYREGION** section above. For example, to impose a Dirichlet condition on both variables for a domain with a single region,

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="u" VALUE="sin(PI*x)*cos(PI*y)" />
4 <D VAR="v" VALUE="sin(PI*x)*cos(PI*y)" />
5 </REGION>
6 </BOUNDARYCONDITIONS>
```

Boundary condition specifications may refer to any parameters defined in the session file. The **REF** attribute corresponds to a defined boundary region. The tag used for each variable specifies the type of boundary condition to enforce.

3.3.6.1 Dirichlet (essential) condition

Dirichlet conditions are specified with the **D** tag.

Projection	Homogeneous support	Time-dependent support	Dimensions
CG	Yes	Yes	1D, 2D and 3D
DG	Yes	Yes	1D, 2D and 3D
HDG	Yes	Yes	1D, 2D and 3D

Example:

```
1 <!-- homogeneous condition -->
2 <D VAR="u" VALUE="0" />
3 <!-- inhomogeneous condition -->
4 <D VAR="u" VALUE="x^2+y^2+z^2" />
5 <!-- time-dependent condition -->
6 <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="x+t" />
```

3.3.6.2 Neumann (natural) condition

Neumann conditions are specified with the \mathbb{N} tag.

Projection	Homogeneous support	Time-dependent support	Dimensions
CG	Yes	Yes	1D, 2D and 3D
DG	No	No	1D, 2D and 3D
HDG	?	?	?

Example:

```
1 <!-- homogeneous condition -->
2 <N VAR="u" VALUE="0" />
3 <!-- inhomogeneous condition -->
4 <N VAR="u" VALUE="x^2+y^2+z^2" />
5 <!-- time-dependent condition -->
6 <N VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="x+t" />
7 <!-- high-order pressure boundary condition (for IncNavierStokesSolver) -->
8 <N VAR="u" USERDEFINEDTYPE="H" VALUE="0" />
```

3.3.6.3 Periodic condition

Periodic conditions are specified with the **P** tag.

Projection	Homogeneous support	Dimensions
CG	Yes	1D, 2D and 3D
DG	No	2D and $3D$

```
Example:
```

```
1 <BOUNDARYREGIONS>
2 <B ID="0"> C[1] </B>
3 <B ID="1"> C[2] </B>
4 </BOUNDARYREGIONS>
5
6 <BOUNDARYCONDITIONS>
    <REGION REF="0">
\overline{7}
     <P VAR="u" VALUE="[1]" />
8
9
   </REGION>
   <REGION REF="1">
10
    <P VAR="u" VALUE="[0]" />
11
   </REGION>
12
13 </BOUNDARYCONDITIONS>
```

Periodic boundary conditions are specified in a significantly different form to other conditions. The **VALUE** property is used to specify which **BOUNDARYREGION** is periodic with the current region in square brackets.

Caveats:

- A periodic condition must be set for "'both"' boundary regions; simply specifying a condition for region 0 or 1 in the above example is not enough.
- The order of the elements inside the composites defining periodic boundaries is important. For example, if 'C[0]' above is defined as edge IDs '0,5,4,3' and 'C[1]' as '7,12,2,1' then edge 0 is periodic with edge 7, 5 with 12, and so on.
- For the above reason, the composites must also therefore be of the same size.
- In three dimensions, care must be taken to correctly align triangular faces which are intended to be periodic. The top (degenerate) vertex should be aligned so that, if the faces were connected, it would lie at the same point on both triangles.
- It is possible specify periodic boundaries that are related by a rotation about a cartesian axis. In three-dimensions it is necessary to specify the rotational arguments to allow the orientation of each periodic face to be determined. This is not required in two-dimensions. An example of how two periodic boundaries are related by a rotation about the x-axis of PI/6 is shown below. The last number specifies an optional tolerance to which the rotation is considered as equivalent (default value is 1e 8).

```
1 <BOUNDARYREGIONS>
2 <B ID="0">C[1] </B>
3 <B ID="1">C[2] </B>
4 </BOUNDARYREGIONS>
5
6 <BOUNDARYCONDITIONS>
7 <REGION REF="0">
8 <P VAR="u" USERDEFINEDTYPE="Rotated:x:PI/6:1e-6" VALUE="[1]" />
```

```
9 </REGION>
10 <REGION REF="1">
11  <P VAR="u" USERDEFINEDTYPE="Rotated:x:-PI/6:1e-6" VALUE="[0]" />
12  </REGION>
13 </BOUNDARYCONDITIONS>
```

3.3.6.4 Time-dependent boundary conditions

Time-dependent boundary conditions may be specified through setting the USERDEFINEDTYPE attribute and using the parameter t where the current time is required. For example,

```
1 <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="sin(PI*(x-t))" />
```

3.3.6.5 Boundary conditions from file

Boundary conditions can also be loaded from file. The following example is from the Incompressible Navier-Stokes solver,

```
1 <REGION REF="1">
2 <D VAR="u" FILE="Test_ChanFlow2D_bcsfromfiles_u_1.bc" />
3 <D VAR="v" VALUE="0" />
4 <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
5 </REGION>
```

Boundary conditions can also be loaded simultaneously from a file and from an expression (currently only implemented in 3D). For example, in the scenario where a spatial boundary condition is read from a file, but needs to be modulated by a time-dependent expression:

In the case where both **VALUE** and **FILE** are specified, the values are multiplied together to give the final value for the boundary condition.

3.3.7 Functions

Finally, multi-variable functions such as initial conditions and analytic solutions may be specified for use in, or comparison with, simulations. These may be specified using expressions (<E>) or imported from a file (<F>) using the Nektar++ FLD file format

```
1 <FUNCTION NAME="ExactSolution">
2  <E VAR="u" VALUE="sin(PI*x-advx*t))*cos(PI*(y-advy*t))" />
3 </FUNCTION>
4 <FUNCTION NAME="InitialConditions">
5  <F VAR="u" FILE="session.rst" />
6 </FUNCTION>
```

A restart file is a solution file (in other words an .fld renamed as .rst) where the field data is specified. The expansion order used to generate the .rst file must be the same as that for the simulation. .pts files contain scattered point data which needs to be interpolated to the field. For further information on the file format and the different interpolation schemes, see section 5.6.19. All filenames must be specified relative to the location of the .xml file.

With the additional argument TIMEDEPENDENT="1", different files can be loaded for each timestep. The filenames are defined using boost::format syntax where the step time is used as variable. For example, the function Baseflow would load the files U0V0_1.0000000E-05.fld, U0V0_2.0000000E-05.fld and so on.

For .pts files, the time consuming computation of interpolation weights is only performed for the first timestep. The weights are stored and reused in all subsequent steps, which is why all consecutive .pts files must use the same ordering, number and location of data points.

Other examples of this input feature can be the insertion of a forcing term,

```
1 <FUNCTION NAME="BodyForce">
2  <E VAR="u" VALUE="0" />
3  <E VAR="v" VALUE="0" />
4 </FUNCTION>
5 <FUNCTION NAME="Forcing">
6  <E VAR="u" VALUE="-(Lambda + 2*PI*PI)*sin(PI*x)*sin(PI*y)" />
7 </FUNCTION>
```

or of a linear advection term

```
1 <FUNCTION NAME="AdvectionVelocity">
2 <E VAR="Vx" VALUE="1.0" />
3 <E VAR="Vy" VALUE="1.0" />
4 <E VAR="Vz" VALUE="1.0" />
5 </FUNCTION>
```

3.3.7.1 Remapping variable names

Note that it is sometimes the case that the variables being used in the solver do not match those saved in the FLD file. For example, if one runs a three-dimensional incompressible Navier-Stokes simulation, this produces an FLD file with the variables (u, v), v and p. If we wanted to use this velocity field as input for an advection velocity, the advection-diffusion-reaction solver expects the variables (vx), (vy) and (vz). We can manually specify this mapping by adding a colon to the filename, indicating the variable names in the target file that align with the desired function variable names. This gives a definition such as:

```
1 <FUNCTION NAME="AdvectionVelocity">
2 <F VAR="Vx,Vy,Vz" FILE="file.fld:u,v,w" />
3 </FUNCTION>
```

There are some caveats with this syntax:

• The same number of fields must be defined for both the **VAR** attribute and in the comma-separated list after the colon. For example, the following is not valid:

```
1 <FUNCTION NAME="AdvectionVelocity">
2 <F VAR="Vx,Vy,Vz" FILE="file.fld:u" />
3 </FUNCTION>
```

• This syntax is not valid with the wildcard operator (*), so one cannot write for example:

```
1 <FUNCTION NAME="AdvectionVelocity">
2 <F VAR="*" FILE="file.fld:u,v,w" />
3 </FUNCTION>
```

3.3.7.2 Time-dependent file-based functions

With the additional argument TIMEDEPENDENT="1", different files can be loaded for each timestep. The filenames are defined using boost::format syntax where the step time is used as variable. For example, the function Baseflow would load the files U0V0_1.0000000E-05.fld, U0V0_2.0000000E-05.fld and so on.

```
1 <FUNCTION NAME="Baseflow">
2 <F VAR="U0,V0" TIMEDEPENDENT="1" FILE="U0V0_%14.8R.fld" />
3 </FUNCTION>
```

Section 3.7 provides the list of acceptable mathematical functions and other related technical details.

3.3.8 Quasi-3D approach

To generate a Quasi-3D appraoch with Nektar++ we only need to create a 2D or a 1D mesh, as reported above, and then specify the parameters to extend the problem to a 3D case. For a 2D spectral/hp element problem, we have a 2D mesh and along with the parameters we need to define the problem (i.e. equation type, boundary conditions, etc.). The only thing we need to do, to extend it to a Quasi-3D approach, is to specify some additional parameters which characterise the harmonic expansion in the third direction. First we need to specify in the solver information section that that the problem will be extended to have one homogeneous dimension; here an example

```
1 <SOLVERINFO>
2 ...
3 <I PROPERTY="HOMOGENEOUS" VALUE="1D" />
4 </SOLVERINFO>
```

then we need to specify the parameters which define the 1D harmonic expansion along the z-axis, namely the homogeneous length (LZ) and the number of modes in the

/>

homogeneous direction (HomModesZ). HomModesZ corresponds also to the number of quadrature points in the homogenous direction, hence on the number of 2D planes discretized with a spectral/hp element method.

```
1 <PARAMETERS>

2 ...

3 <P> HomModesZ = 4 </P>

4 <P> LZ = 1.0 </P>

5 </PARAMETERS>
```

In case we want to create a Quasi-3D approach starting from a 1D spectral/hp element mesh, the procedure is the same, but we need to specify the parameters for two harmonic directions (in Y and Z direction). For Example,

```
1 <SOLVERINFO>
\mathbf{2}
   . . .
   <I PROPERTY="HOMOGENEOUS"
3
                                           VALUE="2D"
4 </SOLVERINFO>
5 <PARAMETERS>
6
    . . .
    <P> HomModesY
                                </P>
                       = 10
7
    <P> LY
                                </P>
                       = 6.5
8
    <P> HomModesZ
                       = 6
                                </P>
9
10
    <P> LZ
                        = 2.0
                                </P>
11 </PARAMETERS>
```

By default the operations associated with the harmonic expansions are performed with the Matrix-Vector-Multiplication (MVM) defined inside the code. The Fast Fourier Transform (FFT) can be used to speed up the operations (if the FFTW library has been compiled in ThirdParty, see the compilation instructions). To use the FFT routines we need just to insert a flag in the solver information as below:

1	<solverinfo></solverinfo>		
2			
3	<i <="" property="HOMOGENEOUS" td=""><td>VALUE="2D"</td><td>/></td></i>	VALUE="2D"	/>
4	<i <="" property="USEFFT" td=""><td>VALUE="FFTW"</td><td>/></td></i>	VALUE="FFTW"	/>
5			

The number of homogeneous modes has to be even. The Quasi-3D approach can be created starting from a 2D mesh and adding one homogeneous expansion or starting form a 1D mesh and adding two homogeneous expansions. Not other options available. In case of a 1D homogeneous extension, the homogeneous direction will be the z-axis. In case of a 2D homogeneous extension, the homogeneous directions will be the y-axis and the z-axis.

3.4 Filters

Filters are a method for calculating a variety of useful quantities from the field variables as the solution evolves in time, such as time-averaged fields and extracting the field variables at certain points inside the domain. Each filter is defined in a (FILTER) tag inside a **FILTERS** block which lies in the main **NEKTAR** tag. In this section we give an overview of the modules currently available and how to set up these filters in the session file.

Here is an example **FILTER**:

A filter has a name – in this case, FilterName – together with parameters which are set to user-defined values. Each filter expects different parameter inputs, although where functionality is similar, the same parameter names are shared between filter types for consistency. Numerical filter parameters may be expressions and so may include session parameters defined in the PARAMETERS section.

Some filters may perform a large number of operations, potentially taking up a significant percentage of the total simulation time. For this purpose, the parameter IO_FiltersInfoSteps is used to set the number of steps between successive total filter CPU time stats are printed. By default it is set to 10 times IO_InfoSteps. If the solver is run with the verbose -v flag, further information is printed, detailing the CPU time of each individual filter and percentage of time integration.

In the following we document the filters implemented. Note that some filters are solverspecific and will therefore only work for a given subset of the available solvers.

3.4.1 Phase sampling

Note

This feature is currently only supported for filters derived from the FieldConvert filter: AverageFields, MovingAverage, ReynoldsStresses.

When analysing certain time-dependent problems, it might be of interest to activate a filter in a specific physical phase and with a certain period (for instance, to carry out phase averaging). The simulation time can be written as $t = m\mathcal{T} + n_{\mathcal{T}}\mathcal{T}$, where *m* is an integer representing the number of periods \mathcal{T} elapsed, and $0 \leq n_{\mathcal{T}} \leq 1$ is the phase. This feature is not a filter in itself and it is activated by adding the parameters below to the filter of interest:

Option name	Required	Default	Description
PhaseAverage	1		Feature activation
PhaseAveragePeriod	1		Period \mathcal{T}
PhaseAveragePhase	\checkmark		Phase $n_{\mathcal{T}}$.

For instance, to activate phase averaging with a period of $\mathcal{T} = 10$ at phase $n_{\mathcal{T}} = 0.5$:

Since this feature monitors $n_{\mathcal{T}}$ every SampleFrequency, for best results it is recommended to set SampleFrequency = 1.

The maximum error in sampling phase is $n_{\mathcal{T},\text{tol}} = \frac{\Delta t}{2\mathcal{T}}$. SampleFrequency, which is displayed at the beginning of the simulation if the solver is run with the verbose $-\mathbf{v}$ option.

The number of periods elapsed is calculated based on simulation time. Caution is therefore recommended when modifying time information in the restart field, because if the new time does not correspond to the same phase, the feature will produce erroneous results.

3.4.2 Aerodynamic forces

Note

This filter is only supported for the incompressible Navier-Stokes solver.

This filter evaluates the aerodynamic forces along a specific surface. The forces are projected along the Cartesian axes and the pressure and viscous contributions are computed in each direction.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which the
			forces are written.
Frequency	X	1	Number of timesteps after which output is
			written.
Boundary	1	-	Boundary surfaces on which the forces are
			to be evaluated.
PivotPoint	X	$(0. \ 0. \ 0.)$	Pivot Point around which the Moments are
			calculated. If Doesn't explicitly defined
			by user, the moments will be calculated
			around the point $\mathbf{X} = (0, 0, 0)$

An example is given below:

```
1 <FILTER TYPE="AeroForces">
2 <PARAM NAME="OutputFile">DragLift</PARAM>
3 <PARAM NAME="OutputFrequency">10</PARAM>
4 <PARAM NAME="Boundary"> B[1,2] </PARAM>
5 <PARAM NAME="PivotPoint"> 1. 0.1 0. </PARAM>
6 </FILTER>
```

During the execution a file named DragLift.fce will be created and the value of the aerodynamic forces on boundaries 1 and 2, defined in the GEOMETRY section, will be output every 10 time steps.

3.4.2.1 Extension for the SPM formulation



This filter is only supported for the incompressible Navier-Stokes solver with the Smoothed Profile Method.

The lack of physical boundaries in the Smoothed Profile Method requires an alternative formulation to calculate the aerodynamic forces. Since the method imposes the boundary geometry by adding an impulse to the flow proportional to the difference between the flow velocity and the expected velocity of the immersed bodies, the forces in this filter are computed by integrating this difference where $\Phi \neq 0$ [26]:

$$\frac{\mathbf{F}^n}{\rho} = \frac{1}{\Delta t} \int_{\Omega} \Phi^{n+1} (\mathbf{u}^* - \mathbf{u}^n_p) \, \mathrm{d}\Omega$$

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which the forces are written.
Frequency	X	1	Number of timesteps after which the output is written.
StartTime	X	0	Forces before this instant are not written to the output.

For instance, a block like the following

```
1 <FILTER TYPE="AeroForcesSPM">
2 <PARAM NAME="OutputFile"> Forces </PARAM>
3 <PARAM NAME="OutputFrequency"> 10 </PARAM>
4 <PARAM NAME="StartTime"> 50.0 </PARAM>
5 </FILTER>
```

will generate a file called **Forces.fce** with the values of the forces in the (X, Y, Z) directions, that must be scaled with the density of the fluid to get the real values. It is important to remark that the computed values are the sum of all the boundaries defined by Φ .

3.4.3 Benchmark

Note

This filter is only supported for the Cardiac Electrophysiology Solver.

Filter **Benchmark** records spatially distributed event times for activation and repolarisation (recovert) during a simulation, for undertaking benchmark test problems.

```
1 <FILTER TYPE="Benchmark">
2 <PARAM NAME="ThresholdValue"> -40.0 </PARAM>
3 <PARAM NAME="InitialValue"> 0.0 </PARAM>
4 <PARAM NAME="OutputFile"> benchmark </PARAM>
5 <PARAM NAME="StartTime"> 0.0 </PARAM>
6 </FILTER>
```

- **ThresholdValue** specifies the value above which tissue is considered to be depolarised and below which is considered repolarised.
- InitialValue specifies the initial value of the activation or repolarisation time map.
- OutputFile specifies the base filename of activation and repolarisation maps output from the filter. This name is appended with the index of the event and the suffix '.fld'.
- **StartTime** (optional) specifies the simulation time at which to start detecting events.

3.4.4 Cell history points



Note

This filter is only supported for the Cardiac Electrophysiology Solver.

Filter **CellHistoryPoints** writes all cell model states over time at fixed points. Can be used along with the **HistoryPoints** filter to record all variables at specific points during a simulation.

- OutputFile specifies the filename to write history data to.
- OutputFrequency specifies the number of steps between successive outputs.
- Points lists coordinates at which history data is to be recorded.

3.4.5 Checkpoint cell model

Note This filter is only supported for the Cardiac Electrophysiology Solver.

Filter CheckpointCellModel checkpoints the cell model. Can be used along with the Checkpoint filter to record complete simulation state and regular intervals.

- OutputFile (optional) specifies the base filename to use. If not specified, the session name is used. Checkpoint files are suffixed with the process ID and the extension '.chk'.
- OutputFrequency specifies the number of timesteps between checkpoints.

3.4.6 Checkpoint fields

The checkpoint filter writes a checkpoint file, containing the instantaneous state of the solution fields at at given timestep. This can subsequently be used for restarting the simulation or examining time-dependent behaviour. This produces a sequence of files, by default named session_*.chk, where * is replaced by a counter. The initial condition is written to session_0.chk. Existing files are not overwritten, but renamed to e.g. session_0.bak0.chk. In case this file already exists, too, the chk-file is renamed to session_0.bak*.chk and so on.



Note

This functionality is equivalent to setting the $\fboxtimesticetonetate{lll_CheckSteps}$ parameter in the session file.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which
OutputFrequency	1	-	the checkpoints are written. Number of timesteps after which output is written.

For example, to output the fields every 100 timesteps we can specify:

3.4.7 Electrogram

Note This filter is only supported for the Cardiac Electrophysiology Solver.

Filter Electrogram computes virtual unipolar electrograms at a prescribed set of points.

```
1 <FILTER TYPE="Electrogram">
2  <PARAM NAME="OutputFile"> session </PARAM>
3  <PARAM NAME="OutputFrequency"> 1 </PARAM>
4  <PARAM NAME="Points">
5      0.0     0.5     0.7
6      1.0     0.5     0.7
7      2.0     0.5     0.7
8  </PARAM>
9 </FILTER>
```

- OutputFile (optional) specifies the base filename to use. If not specified, the session name is used. The extension '.ecg' is appended if not already specified.
- OutputFrequency specifies the number of resolution of the electrogram data.
- **Points** specifies a list of coordinates at which electrograms are desired. *They must not lie within the domain.*

3.4.8 Error

This filter produces a file containing the time-evolution of the L_2 and L_{inf} errors. By default this file is called **session.err** where **session** is the session name.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which
			the errors are written.
OutputFrequency	×	1	Number of timesteps after which output
			is written.

An example syntax is given below:

3.4.9 FieldConvert checkpoints

This filter applies a sequence of FieldConvert modules to the solution, writing an output file. An output is produced at the end of the simulation into $session_fc.fld$, or alternatively every M timesteps as defined by the user, into a sequence of files $session_*_fc.fld$, where * is replaced by a counter.

Module options are specified as a colon-separated list, following the same syntax as the **FieldConvert** command-line utility (see Section 5).

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session.fld	Output filename. If no extension is provided, it is assumed as .fld
OutputFrequency	×	NumSteps	Number of timesteps after which output is written, M .
Modules	X		FieldConvert modules to run, sepa- rated by a white space.

As an example, consider:

```
1 <FILTER TYPE="FieldConvert">
2  <PARAM NAME="OutputFile">MyFile.vtu</PARAM>
```

```
3 <PARAM NAME="OutputFrequency">100</PARAM>
4 <PARAM NAME="Modules"> vorticity isocontour:fieldid=0:fieldvalue=0.1 </PARAM>
5 </FILTER>
```

This will create a sequence of files named MyFile_*_fc.vtu containing isocontours. The result will be output every 100 time steps.

3.4.10 History points

The history points filter can be used to evaluate the value of the fields in specific points of the domain as the solution evolves in time. By default this produces a file called **session.his**. For each timestep, and then each history point, a line is output containing the current solution time, followed by the value of each of the field variables. Commented lines are created at the top of the file containing the location of the history points and the order of the variables.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which
			the checkpoints are written.
OutputFrequency	×	1	Number of timesteps after which output
			is written.
OutputOneFile	X	true	If OutputOneFile is true or yes, only
			one file is generated for all points and
			the whole history. Otherwise, a separate
			file is created for each time snapshot.
OutputPlane	X	-1	If the simulation is homogeneous, the
			plane on which to evaluate the history
			point. If set, all points will be reset to
			lie on this plane.
WaveSpace	×	false	If the simulation is homogeneous and
			WaveSpace is true or yes, the Fourier
			coefficient on the nearest plane will be
			output.
Points	×	-	A list of the history points.
line	×	-	npts,x0,y0,x1,y1 in 2D or
			npts,x0,y0,z0,x1,y1,z1 in 3D. Set the
			history points as npts equispaced points
			between $(x0,y0,z0)$ and $(x1,y1,z1)$.
plane	×	-	npts1,npts2,x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y
			Set the history points as npts1 * npts2
			equispaced points on a plane.
			npts1,npts2 is the number of eq-
			uispaced points in each direction and
			$(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2)$ and
			(x_3, y_3, z_3) define the plane of points
			specified in a clockwise or anticlockwise
			direction. Be careful about the size
			of history points file when using this
			option.
box	×	-	npts1,npts2,npts3,xmin,xmax,ymin,ymax,zmin,
			Set the history points as
			npts1 * npts2 * npts3 equispaced
			points in a box. npts1,npts2,npts3
			is the number of equispaced points in
			each direction and $(x_{min}, y_{min}, z_{min})$
			and $(x_{max}, y_{max}, z_{max})$ define the limits
			of the box of points. Be careful about
			the size of history points file when using
			this option.

One of Points, line, plane or box must be specified.

For example, to output the value of the solution fields at three points (1, 0.5, 0), (2, 0.5, 0) and (3, 0.5, 0) into a file TimeValues.his every 10 timesteps, we use the syntax:

```
1 <FILTER TYPE="HistoryPoints">
      <PARAM NAME="OutputFile">TimeValues</PARAM>
\mathbf{2}
3
      <PARAM NAME="OutputFrequency">10</PARAM>
      <PARAM NAME="Points">
4
         1 0.5 0
\mathbf{5}
          2 0.5 0
6
          3 0.5 0
7
      </PARAM>
8
9 </FILTER>
```

or

3.4.11 Kinetic energy and enstrophy



Note

This filter is only supported for the incompressible and compressible Navier-Stokes solvers in three dimensions.

The purpose of this filter is to calculate the kinetic energy and enstrophy

$$E_k = \frac{1}{2\mu(\Omega)} \int_{\Omega} \|\mathbf{u}\|^2 \, dx, \qquad \mathcal{E} = \frac{1}{2\mu(\Omega)} \int_{\Omega} \|\omega\|^2 \, dx$$

where $\mu(\Omega)$ is the volume of the domain Ω . This produces a file containing the timeevolution of the kinetic energy and enstrophy fields. By default this file is called **session.eny** where **session** is the session name.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session.eny	Output file name to which the en-
OutputFrequency	1	-	ergy and enstrophy are written. Number of timesteps at which out- put is written.

To enable the filter, add the following to the **FILTERS** tag:

3.4.12 Mean values

This filter calculates time-evolution of the averaged velocity components over the flow domain and can be used to track flow rates.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which averaged values are written.
OutputFrequency	X	1	Number of timesteps after which output is written.

An example syntax is given below:

3.4.13 Modal energy

Note

ň

This filter is only supported for the incompressible Navier-Stokes solver.

This filter calculates the time-evolution of the kinetic energy. In the case of a two- or three-dimensional simulation this is defined as

$$E_k(t) = \frac{1}{2} \int_{\Omega} \|\mathbf{u}\|^2 \, dx$$

However if the simulation is written as a one-dimensional homogeneous expansion so that

$$\mathbf{u}(\mathbf{x},t) = \sum_{k=0}^{N} \mathbf{\hat{u}}_k(t) e^{2\pi i k \mathbf{x}}$$

then we instead calculate the energy spectrum

$$E_k(t) = \frac{1}{2} \int_{\Omega} \|\mathbf{\hat{u}}_k\|^2 \, dx.$$

Note that in this case, each component of $\hat{\mathbf{u}}_k$ is a complex number and therefore N = (HomModesZ)/2 lines are output for each timestep. This is a particularly useful tool in examining turbulent and transitional flows which use the homogeneous extension. In either case, the resulting output is written into a file called (session.mdl) by default.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which
OutputFrequency	×	1	the energy spectrum is written. Number of timesteps after which output is written.

An example syntax is given below:

3.4.14 Moving body

8

Note

This filter is only supported for the Quasi-3D incompressible Navier-Stokes solver, in conjunction with the MovingBody forcing.

This filter MovingBody is encapsulated in the forcing module to evaluate the aerodynamic forces along the moving body surface. It is described in detail in section 11.3.4.1

3.4.15 Moving average of fields

This filter computes the exponential moving average (in time) of fields for each variable defined in the session file. The moving average is defined as:

$$\bar{u}_n = \alpha u_n + (1 - \alpha)\bar{u}_{n-1}$$

with $0 < \alpha < 1$ and $\bar{u}_1 = u_1$.

The same parameters of the time-average filter are supported, with the output file in the form session_*_movAvg.fld. In addition, either α or the time-constant τ must be defined. They are related by:

$$\alpha = \frac{t_s}{\tau + t_s}$$

where t_s is the time interval between consecutive samples.

As an example, consider:

This will create a file named MyMovingAverage_movAvg.fld with a moving average sampled every 10 time steps. The averaged field is however only output every 100 time steps.

3.4.16 One-dimensional energy

This filter is designed to output the energy spectrum of one-dimensional elements. It transforms the solution field at each timestep into a orthogonal basis defined by the functions

$$\psi_p(\xi) = L_p(\xi)$$

where L_p is the *p*-th Legendre polynomial. This can be used to show the presence of, for example, oscillations in the underlying field due to numerical instability. The resulting output is written into a file called **session.eny** by default. The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which
OutputFrequency	×	1	the energy spectrum is written. Number of timesteps after which output is written.

An example syntax is given below:

3.4.17 Reynolds stresses

Note This filter is only supported for the incompressible Navier-Stokes solver. This filter is an extended version of the time-average fields filter (see Section 3.4.18). It outputs not only the time-average of the fields, but also the Reynolds stresses. The same parameters supported in the time-average case can be used, for example:

By default, this filter uses a simple average. Optionally, an exponential moving average can be used, in which case the output contains the moving averages and the Reynolds stresses calculated based on them. For example:

3.4.18 Time-averaged fields

This filter computes time-averaged fields for each variable defined in the session file. Time averages are computed by either taking a snapshot of the field every timestep, or alternatively at a user-defined number of timesteps N. An output is produced at the end of the simulation into <code>session_avg.fld</code>, or alternatively every M timesteps as defined by the user, into a sequence of files <code>session_*_avg.fld</code>, where ***** is replaced by a counter. This latter option can be useful to observe statistical convergence rates of the averaged variables.

This filter is derived from FieldConvert filter, and therefore support all parameters available in that case. The following additional parameter is supported:

Option name	Required	Default	Description
SampleFrequency	X	1	Number of timesteps at which the average is calculated, N .
RestartFile	×		Restart file used as initial average. If no extension is provided, it is assumed as .fld

As an example, consider:

```
1 <FILTER TYPE="AverageFields">
```

2 <PARAM NAME="OutputFile">MyAverageField</PARAM>

```
3 <PARAM NAME="RestartFile">MyRestartAvg.fld</PARAM>
4 <PARAM NAME="OutputFrequency">100</PARAM>
5 <PARAM NAME="SampleFrequency"> 10 </PARAM>
6 </FILTER>
```

This will create a file named MyAverageField.fld averaging the instantaneous fields every 10 time steps. The averaged field is however only output every 100 time steps.

3.4.19 ThresholdMax

The threshold value filter writes a field output containing a variable m, defined by the time at which the selected variable first exceeds a specified threshold value. The default name of the output file is the name of the session with the suffix $_max.fld$. Thresholding is applied based on the first variable listed in the session by default.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	$session_max.fld$	Output filename to which the threshold times are written.
ThresholdVar	×	first variable name	Specifies the variable on which the threshold will be applied.
ThresholdValue	1	-	Specifies the threshold value.
InitialValue	1	-	Specifies the initial time.
StartTime	X	0.0	Specifies the time at which to
			start recording.

An example is given below:

```
1 <FILTER TYPE="ThresholdMax">
2 <PARAM NAME="OutputFile"> threshold_max.fld </PARAM>
3 <PARAM NAME="ThresholdVar"> u </PARAM>
4 <PARAM NAME="ThresholdValue"> 0.1 </PARAM>
5 <PARAM NAME="InitialValue"> 0.4 </PARAM>
6 </FILTER>
```

which produces a field file threshold_max.fld.

3.4.20 ThresholdMin value

Performs the same function as the ThresholdMax filter (see Section ??) but records the time at which the threshold variable drops below a prescribed value.

3.4.21 Maximun/minimun fields

This filter computes the local (pointwise) maximum or minimum field for each variable, including the addition variables for compressible flow solver, over the simulation.
Option name	Required	Default	Description
MaxOrMin	✓	max	Type of fields to compute. It can
			be either max or min.
RestartFile	×	∓ 9999	Restart file used for initial com-
			parison. If it is not provided,
			-9999 will be used for the max
			filter while 9999 will be used for
			the min filter. Output filename
			to which the threshold times are
			written.
OutputFile	X	$session_max.fld$	Output filename to which the
			threshold times are written.
OutputFrequency	X	1	Number of timesteps after which
			output is written.
SampleFrequency	X	1	Number of timesteps after which
			the fields are used to compute
			\max/\min .

The following parameters are supported:

An example is given below:

```
1 <FILTERS>
    <FILTER TYPE="MaxMinFields">
2
       <PARAM NAME="MaxOrMin">max</PARAM>
3
       <PARAM NAME="RestartFile">Baseflow.fld</PARAM>
4
       <PARAM NAME="OutputFile">DisturbedFields</PARAM>
\mathbf{5}
        <PARAM NAME="OutputFrequency">100</PARAM>
6
\overline{7}
       <PARAM NAME="SampleFrequency">1</PARAM>
8
     </FILTER>
9
   </FILTERS>
```

3.5 Forcing

An optional section of the file allows forcing functions to be defined. These are enclosed in the **FORCING** tag. The forcing type is enclosed within the **FORCE** tag and expressed in the file as:

```
1 <FORCE TYPE="[NAME]">
2 ...
3 </FORCE>
```

The force type can be any one of the following.

62 Chapter 3 XML Session File

3.5.1 Absorption

This force type allows the user to apply an absorption layer (essentially a porous region) anywhere in the domain. The user may also specify a velocity profile to be imposed at the start of this layer, and in the event of a time-dependent simulation, this profile can be modulated with a time-dependent function. These velocity functions and the function defining the region in which to apply the absorption layer are expressed in the **CONDITIONS** section, however the name of these functions are defined here by the **COEFF** tag for the layer, the **REFFLOW** tag for the velocity profile, and the **REFFLOWTIME** for the time-dependent function.

If a list of **BOUNDARYREGIONS** is specified, the distance to these regions is available as additional variable (\mathbf{r}) in the definition of the **COEFF** function:

3.5.2 Body

This force type specifies the name of a body forcing function expressed in the **CONDITIONS** section.

3.5.3 MovingReferenceFrame

This force type allows the solution of incompressilbe Navier-Stokes in moving frame of reference. The moving frame is attached the to body and can have translational, rotational or both motions. Although the Navier-Stokes equations are solved in a moving reference frame, our formulation is based on the absolute velocity and pressure (in inertial frame). However, note that these absolute velocities and any other vector quantities are expressed using the coordinate basis of the moving frame. Further, note that if you are using the FilterAeroForces, the force vector (F_x, F_y, F_z) is automatically converted and output in the inertial frame (ground reference frame).

To use this formulation the user need to specify the force type inside the **FORCING** tag as follwos:

Here we are required to provide at least one function for this force type which can be a function that defines the linear velocity of the reference frame or a function that defines the angular velocity of reference frame or both. In the case of rotating frame, i.e. when we are prescribing the angular velocity of reference frame, we can provide a coordinate of **PivotPoint**, around which the frame is rotating. If no pivot point provided, the orgin of coordinates in the moving reference frame will be used as the pivot point. Note that the frame velocities (both linear and angular velocities) must be defined in the inertial stationary frame of reference, i.e. ground reference frame (and expressed using the basis of inertial stationary frame), however, the Poivot point is in the moving reference frame.

Examples of linear and angular velocity funcitons together with their usage in the Forcing is shown below:

```
1 <CONDITIONS>
2
3 <FUNCTION NAME="LinVel">
      <E VAR="u" VALUE="2*sin(PI*t)" />
4
      <E VAR="v" VALUE="0.1" />
5
     <E VAR="w" VALUE="0" />
6
7 </FUNCTION>
8
9 <FUNCTION NAME="AngVel">
    <E VAR="Omega_x" VALUE="0" />
10
      <E VAR="Omega_y" VALUE="O" />
11
      <E VAR="Omega_z" VALUE="0.3*cos(2*PI*t)" />
12
13 </FUNCTION>
14
15 </CONDITIONS>
16
17 <FORCING>
18
      <FORCE TYPE="MovingReferenceFrame">
19
          <LinearVelocity> LinVel <LinearVelocity/>
20
          <AngularVelocity> AngVel <AngularVelocity/>
21
          <PivotPoint> 0.2 0.0 0.0 <PivotPoint/>
22
      </FORCE>
23
24
25 </FORCING>
```

The moving frame functions defines the velocity of the body frame observed in the inertial reference frame

$$\mathbf{u}_{frame} = \mathbf{u}_0 + \mathbf{\Omega} \times (\mathbf{x} - \mathbf{x}_0)$$

. This means that these functions (such as the LinVel and AngVel in the above example) are defined and expressed in the stationary inertial frame (ground frame).

Here, $\mathbf{u}_0 = (\mathbf{u}, \mathbf{v}, \mathbf{w})$ is the translational velocity, $\mathbf{\Omega} = (\text{Omega}_x, \text{Omega}_y, \text{Omega}_z)$ is the angular velocity. $\mathbf{x}_0 = (0.2, 0.0, 0.0)$ is the rotation pivot and it is fixed in the body frame. Translational motion is allowed for all dimensions while rotational motion is currently restricted to z (omega_z) for 2D, 3DH1D and full 3D simulaitons.

Finally, note that when using MovingReferenceFrame force type, for any open part of the computational domain that the user specifies the velocity, such as inlet and free stream boundary conditions, the USERDEFINEDTYPE="MovingFrameDomainVel" tag should be used for all of velocity components. For example if boundary ID=2 is the inlet with Uinfx and Uinfy the values of inlet velocities defined as parameters, the boundary condition for this boundary becomes:

for the wall boundary conditions on the surface of the body, we need to use MovingFrameWall tag as shown below:

The outlet and pressure boundary conditions are the same as before.

3.5.4 Programmatic

This force type allows a forcing function to be applied directly within the code, thus it has no associated function.

```
1 <FORCE TYPE="Programmatic">
2 </FORCE>
```

3.5.5 Noise

This force type allows the user to specify the magnitude of a white noise force. Optional arguments can also be used to define the frequency in time steps to recompute the noise (default is never) and the number of time steps to apply the noise (default is the entire simulation).

3.6 Coupling

Nektar++ Solvers can be run in parallel with third party applications and other Nektar++ solvers, where run-time data exchange is enabled by the coupling interface. The interface is configured in the (COUPLING) tag as

```
1 <COUPLING TYPE="[type]" NAME="[name]">
2 <I PROPERTY="SendSteps" VALUE="1" />
3 <I PROPERTY="SendVariables" VALUE="uOS,vOS" />
4 <I PROPERTY="ReceiveSteps" VALUE="1" />
5 <I PROPERTY="ReceiveVariables" VALUE="uOR,vOR" />
6 ...
7 </COUPLING>
```

The coupling type may be any of the types described later in this section, while the name can be chosen arbitrarily. Inside each coupling block, the send and receive frequencies are defined by the **SendSteps** and **ReceiveSteps** parameters, respectively. Which variables are to be sent or received is specified by the **SendVariables** and **ReceiveVariables**. By default, the send and receive frequencies is set to zero, which disables the corresponding exchange in this coupling. An empty **SendVariables** or **ReceiveVariables** list has the same effect.

Option name	Required	Default	Description
SendSteps	X	0	Frequency (in steps) at which fields are sent. Sending is disabled if set to zero.
SendVariables	X	<empty></empty>	Comma-separated list of sent variables. Sending is disabled if the list is empty.
ReceiveSteps	×	0	Frequency (in steps) at which fields are received. Receiving is disabled if set to zero.
ReceiveVariables	X	<empty></empty>	Comma-separated list of received vari- ables. Receiving is disabled if the list is empty.

3.6.1 File

This coupling type allows the user to exchange fields at run time by reading from and writing to files. Besides the basic parameters which define the exchanged variables and the exchange frequency, the file coupling type requires the SendFileName and ReceiveFunction parameters to be set. The Coupling name is not used for this type and can be ignored.

66 Chapter 3 XML Session File

```
6 <I PROPERTY="ReceiveVariables" VALUE="uOR,vOR" />
7 <I PROPERTY="ReceiveFunction" VALUE="CouplingIn" />
8 </COUPLING>
```

SendFileName specifies a file name template to write the field data to. Currently, only .pts files are supported and the file is only created once fully written, avoiding race conditions between sender and receiver. Receiving is implemented by evaluating a session function specified in the ReceiveFunction parameter. The coupling waits for the file given in the receive function to appear.

Option name	Required	Default	Description
SendFileName	(•)	-	File name where the sent fields should be written to. Required if sending is enabled. Time dependent file names are supported.
ReceiveFunction	(✓)	-	Function to evaluate to obtain the re- ceived fields.Required if receiving is en- abled.

3.6.2 Cwipi



Note

The Cwipi coupling is only available when Nektar++ is compiled with OpenMPI and CWIPI

The Cwipi coupling uses CWIPI¹ to facilitate real time data exchange over MPI. See [24] for details. All data transfers are non-blocking to minimize the computational overhead. The interface must be enabled with the command line option **-cwipi** and a unique application name, e.g:

```
DummySolver --cwipi 'Dummy1' Dummy_3DCubeCwipi_1.xml
```

CWIPI uses the names of the current application and the coupling to identify two peers in cosimulation setups. The name of the remote application must be provided by the **RemoteName** parameter. Unlike the File-type coupling, a linear interpolation in time is applied to the received fields if non-unity values are set for **ReceiveSteps**.

```
1 <COUPLING NAME="coupling1" TYPE="Cwipi">
2 <I PROPERTY="RemoteName" VALUE="Dummy1" />
3 <I PROPERTY="SendSteps" VALUE="1" />
4 <I PROPERTY="SendVariables" VALUE="uOS,vOS" />
5 <I PROPERTY="SendMethod" VALUE="Evaluate" />
```

¹http://sites.onera.fr/cwipi/

```
6<I PROPERTY="ReceiveSteps"</th>VALUE="1" />7<I PROPERTY="ReceiveVariables"</td>VALUE="uOR,vOR" />8<I PROPERTY="Oversample"</td>VALUE="5" />9<I PROPERTY="FilterWidth"</td>VALUE="10E-3" />10<I PROPERTY="NotLocMethod"</td>VALUE="Extrapolate" />11
```

Additional options which define the coupling include SendMethod, the method used to retrieve the physical values at the locations requested by the remote application. Available options are NearestNeighbour, Shepard and the default Evaluate. The last option directly evaluates the expansions using a backward transform, giving superior accuracy at acceptable computational cost.

When using non-conforming domains, the current application might request values outside of the computational domain of the remote application. How to handle these not-located points is specified by the NotLocMethod parameter. When set to keep, the point value is not altered. With Extrapolate, the nearest neighbor value of the current application is used. Note that this can be very inefficient when using many MPI ranks.

Option name	Required	Default	Description
RemoteName	✓	-	Name of the remote application.
SendMethod	X	Evaluate	Specifies how to evaluate fields be-
			fore sending. Available options are
			NearestNeighbour, Shepard and
			Evaluate).
Oversample	×	0	Receive fields at a higher (or lower) num-
			ber of quadrature points before filtering
			to avoid aliasing.
FilterWidth	X	0	Apply a spatial filter of a given filter width
			to the received fields. Disabled when set
			to zero.
NotLocMethod	X	keep	Specifies how not located points in non-
			conformal domains are handled. Possible
			values are keep and Extrapolate.

3.7 Expressions

This section discusses particulars related to expressions appearing in Nektar++. Expressions in Nektar++ are used to describe spatially or temporally varying properties, for example

- velocity profiles on a boundary
- some reference functions (e.g. exact solutions)

68 Chapter 3 XML Session File

which can be retrieved in the solver code.

Expressions appear as the content of **VALUE** attribute of

- parameter values;
- boundary condition type tags within <REGION> subsection of <BOUNDARYCONDITIONS>,
 e.g. <D>, <N> etc;
- expression declaration tag <E> within <FUNCTION> subsection.

The tags above declare expressions as well as link them to one of the field variables declared in **<EXPANSIONS>** section. For example, the declaration

```
1 <D VAR="u" VALUE="sin(PI*x)*cos(PI*y)" />
```

registers expression $\sin(\pi x)\cos(\pi y)$ as a Dirichlet boundary constraint associated with field variable \mathbf{u} .

Enforcing the same velocity profile at multiple boundary regions and/or field variables results in repeated re-declarations of a corresponding expression. Currently one cannot directly link a boundary condition declaration with an expression uniquely specified somewhere else, e.g. in the *<FUNCTION>* subsection. However this duplication does not affect an overall computational performance.

3.7.1 Variables and coordinate systems

Declarations of expressions are formulated in terms of problem space-time coordinates. The library code makes a number of assumptions to variable names and their order of appearance in the declarations. This section describes these assumptions.

Internally, the library uses 3D global coordinate space regardless of problem dimension. Internal global coordinate system has natural basis (1,0,0),(0,1,0),(0,0,1) with coordinates (x), y and (z). In other words, variables (x), y and (z) are considered to be first, second and third coordinates of a point ((x), (y), (z)).

Declarations of problem spatial variables do not exist in the current XML file format. Even though field variables are declarable as in the following code snippet,

```
1 <VARIABLES>

2 <V ID="0"> u </V>

3 <V ID="1"> v </V>

4 </VARIABLES>
```

there are no analogous tags for space variables. However an attribute SPACE of <GEOMETRY> section tag declares the dimension of problem space. For example,

```
1 <GEOMETRY DIM="1" SPACE="2"> ...
2 </GEOMETRY>
```

specifies 1D flow within 2D problem space. The number of spatial variables presented in expression declaration should match space dimension declared via <a href="https://www.ceencommutation.ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-space-ceencommutation-ceencommutatio-ceencommutation-ceencommutation-ceencommutatio-ceencommutation-ceencommutatio-ceencommutatio-ceencommutatio-ceencommutati

The library assumes the problem space also has natural basis and spatial coordinates have names (x), (y) and (z).

Problem space is naturally embedded into the global coordinate space: each point of

- 1D problem space with coordinate x is represented by 3D point (x,0,0) in the global coordinate system;
- 2D problem space with coordinates (x,y) is represented by 3D point (x,y,0) in the global coordinate system;
- 3D problem space with coordinates (x,y,z) has the same coordinates in the global space coordinates.

Currently, there is no way to describe rotations and translations of problem space relative to the global coordinate system.

The list of variables allowed in expressions depends on the problem dimension:

- For 1D problems, expressions must make use of variable (x) only;
- For 2D problems, expressions should make use of variables (x) and (y) only;
- For 3D problems, expressions may use any of variables (x), (y) and (z).

Violation of these constraints yields unpredictable results of expression evaluation. The current implementation assigns magic value -9999 to each dimensionally excessive spacial variable appearing in expressions. For example, the following declaration

```
<GEOMETRY DIM="2" SPACE="2"> ...
1
2
    </GEOMETRY> ...
3
    <CONDITIONS> ...
      <BOUNDARYCONDITIONS>
4
         <REGION REF="0">
5
           <D VAR="u" VALUE="x+y+z" /> <D VAR="v" VALUE="sin(PI*x)*cos(PI*y)" />
6
7
         </REGION>
       </BOUNDARYCONDITIONS>
8
9
    </CONDITIONS>
10
```

results in expression x + y + z being evaluated at spatial points $(x_i, y_i, -9999)$ where x_i and y_i are the spacial coordinates of boundary degrees of freedom. However, the library behaviour under this constraint violation may change at later stages of development (e.g., magic constant 0 may be chosen) and should be considered unpredictable.

70 Chapter 3 XML Session File

Another example of unpredictable behaviour corresponds to wrong ordering of variables:

```
<GEOMETRY DIM="1" SPACE="1"> ...
1
    </GEOMETRY> ...
\mathbf{2}
3
    <CONDITIONS> ...
4
      <BOUNDARYCONDITIONS>
         <REGION REF="0">
5
           <D VAR="u" VALUE="sin(y)" />
6
7
         </REGION>
       </BOUNDARYCONDITIONS>
8
9
    . . .
    </CONDITIONS>
10
```

Here one declares 1D problem, so Nektar++ library assumes spacial variable is \mathbf{x} . At the same time, an expression sin(y) is perfectly valid on its own, but since it does not depend on \mathbf{x} , it will be evaluated to constant sin(-9999) regardless of degree of freedom under consideration.

3.7.1.1 Time dependence

Variable t represents time dependence within expressions. The boundary condition declarations need to add an additional property USERDEFINEDTYPE="TimeDependent" in order to flag time dependency to the library.

3.7.1.2 Syntax of expressions

Analytic expressions are formed of

- brackets (). Bracketing structure must be balanced.
- real numbers: every representation is allowed that is correct for boost::lexical_cast<double>(),
 e.g.

1 1.2, 1.2e-5, .02

• mathematical constants

3.7 Expressions 71

Identifier	Meaning	Real Value			
	Fundamental constants				
Ε	Natural Logarithm	2.71828182845904523536			
PI	π	3.14159265358979323846			
GAMMA	Euler Gamma	0.57721566490153286060			
DEG	deg/radian	57.2957795130823208768			
PHI	golden ratio	1.61803398874989484820			
Derived constants					
LOG2E	$\log_2 e$	1.44269504088896340740			
LOG10E	$\log_{10} e$	0.43429448190325182765			
LN2	$\log_e 2$	0.69314718055994530942			
PI_2	$\frac{\pi}{2}$	1.57079632679489661923			
PI_4	$\frac{\overline{\pi}}{4}$	0.78539816339744830962			
1_PI	$\frac{1}{\pi}$	0.31830988618379067154			
2_PI	$\frac{2}{\pi}$	0.63661977236758134308			
2_SQRTPI	$\frac{2}{\sqrt{\pi}}$	1.12837916709551257390			
SQRT2	$\sqrt{2}$	1.41421356237309504880			
$SQRT1_2$	$\frac{1}{\sqrt{2}}$	0.70710678118654752440			

- parameters: alphanumeric names with underscores, e.g. GAMMA_123, GaM123_45a_,
 _gamma123 are perfectly acceptable parameter names. However parameter name
 cannot start with a numeral. Parameters must be defined with <PARAMETERS>...</PARAMETERS>.
 Parameters play the role of constants that may change their values in between of expression evaluations.
- variables (i.e., x, y, z and t)
- unary minus operator (e.g. -x)
- binary arithmetic operators +, -, *, /, ^ Powering operator allows using real exponents (it is implemented with std::pow() function)
- boolean comparison operations <, <=, >, >=, == evaluate their sub-expressions to real values 0.0 or 1.0.
- mathematical functions of one or two arguments:

_

Identifier	Meaning
abs(x)	absolute value $ x $
asin(x)	inverse sine $\arcsin x$
acos(x)	inverse cosine $\arccos x$
ang(x,y)	computes polar coordinate $\theta = \arctan(y/x)$ from (x, y)
atan(x)	inverse tangent $\arctan x$
atan2(y,x)	inverse tangent function (used in polar transformations)
ceil(x)	round up to nearest integer $\lceil x \rceil$
cos(x)	$\cos n e \cos x$
cosh(x)	hyperbolic cosine $\cosh x$
exp(x)	exponential e^x
fabs(x)	absolute value (equivalent to abs)
floor(x)	rounding down $\lfloor x \rfloor$
log(x)	logarithm base e , $\ln x = \log x$
log10(x)	logarithm base 10, $\log_{10} x$
rad(x,y)	computes polar coordinate $r = \sqrt{x^2 + y^2}$ from (x, y)
sin(x)	sine $\sin x$
sinh(x)	hyperbolic sine $\sinh x$
sqrt(x)	square root \sqrt{x}
tan(x)	tangent $\tan x$
tanh(x)	hyperbolic tangent $\tanh x$

These functions are implemented by means of the cmath library: http://www.cplusplus.com/reference/clibrary/cmath/. Underlying data type is double at each stage of expression evaluation. As consequence, complex-valued expressions (e.g. $(-2)^0.123$) get value nan (not a number). The operator $\hat{}$ is implemented via call to std::pow() function and accepts arbitrary real exponents.

• random noise generation functions. Currently implemented is awgn(sigma) - Gaussian Noise generator, where σ is the variance of normal distribution with zero mean. Implemented using the boost::mt19937 random number generator with boost variate generators (see http://www.boost.org/libs/random)

3.7.1.3 Examples

Some straightforward examples include

- Basic arithmetic operators: (0.5*0.3164/(3000^0.25))
- Simple polynomial functions: (y*(1-y))
- Use of values defined in PARAMETERS section: -2*Kinvis*(x-1)
- More complex expressions involving trigonometric functions, parameters and constants: (LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)

• Boolean operators for multi-domain functions: (y<0)*sin(y) + (y>=0)*y

3.7.2 Performance considerations

Processing expressions is split into two stages:

- parsing with pre-evaluation of constant sub-expressions,
- evaluation to a number.

Parsing of expressions with their partial evaluation take place at the time of setting the run up (reading an XML file). Each expression, after being pre-processed, is stored internally and quickly retrieved when it turns to evaluation at given spatial-time point(s). This allows to perform evaluation of expressions at a large number of spacial points with minimal setup costs.

3.7.2.1 Pre-evaluation details

Partial evaluation of all constant sub-expressions makes no sense in using derived constants from table above. This means, either make use of pre-defined constant LN10² or straightforward expression log10(2)² results in constant 5.3018981104783980105 being stored internally after pre-processing. The rules of pre-evaluation are as follows:

- constants, numbers and their combinations with arithmetic, analytic and comparison operators are pre-evaluated,
- appearance of a variable or parameter at any recursion level stops pre-evaluation of all upper level operations (but doesn't stop pre-evaluation of independent parallel sub-expressions).

For example, declaration

1 <D VAR="u" VALUE="exp(-x*sin(PI*(sqrt(2)+sqrt(3))/2)*y)" />

results in expression $\exp(-x*(-0.97372300937516503167)*y)$ being stored internally: sub-expression $\sin(PI*(sqrt(2)+sqrt(3))/2)$ is evaluated to constant but appearance of x and y variables stops further pre-evaluation.

Grouping predefined constants and numbers together helps. Its useful to put brackets to be sure your constants do not run out and become factors of some variables or parameters.

Expression evaluator does not do any clever simplifications of input expressions, which is clear from example above (there is no point in double negation). The following subsection addresses the simplification strategy.

74 Chapter 3 XML Session File

3.7.2.2 Preparing expression

The total evaluation cost depends on the overall number of operations. Since evaluator is not making simplifications, it worth trying to minimise the total number of operations in input expressions manually.

Some operations are more computationally expensive than others. In an order of increasing complexity:

- +, -, <, >, <=, >=, ==,
- *, /, abs, fabs, ceil, floor,
- ^, sqrt, exp, log, log10, sin, cos, tan, sinh, cosh, tanh, asin, acos, atan.

For example,

- x*x is faster than x^2 it is one double multiplication vs generic calculation of arbitrary power with real exponents.
- $(x+\sin(y))^2$ is faster than $(x+\sin(y))*(x+\sin(y))$ sine is an expensive operation. It is cheaper to square complicated expression rather than compute it twice and add one multiplication.
- An expression exp(-41*((x+(0.3*cos(2*PI*t)))^2 + (0.3*sin(2*PI*t))^2)) makes use of 5 expensive operations (exp, sin, cos and power ^ twice) while an equivalent expression exp(-41*(x*x+0.6*x*cos(2*PI*t) + 0.09)) uses only 2 expensive operations.

If any simplifying identity applies to input expression, it may worth applying it, provided it minimises the complexity of evaluation. Computer algebra systems may help.

3.7.2.3 Vectorized evaluation

Expression evaluator is able to calculate an expression for either given point (its spacetime coordinates) or given array of points (arrays of their space-time coordinates, it uses SoA). Vectorized evaluation is faster then sequential due to a better data access pattern. Some expressions give measurable speedup factor 4.6. Therefore, if you are creating your own solver, it worth making vectorized calls.

Part II

Preprocessing & Postprocessing

 $_{\rm CHAPTER}$ 4

NekMesh

NekMesh is a utility bundled with Nektar++ which has two purposes:

- allow foreign mesh file formats to be converted into Nektar++'s XML format;
- aide in the generation of high-order meshes through a series of supplied processing modules.

Note

ň

NekMesh replaces a previous utility called MeshConvert. This change is to reflect the fact that the program no longer only converts and manipulates meshes but can now also generate them from a CAD definition. This mesh generator is in an early stage of development and as such is disabled by default. For the time being those not using the mesh generator can use NekMesh as they would have used MeshConvert, none of the functionality or methodology has changed.

There is also some limited support for other output formats. We begin by running through a basic example to show how a mesh can be converted from the widely-used mesh-generator **Gmsh** to the XML file format.

Note

The default since January 2016 is to output the .xml files in a compressed form where the VERTEX, EDGES, FACES, ELEMENTS and CURVED information is compressed into binary format which is then converted into base64. This is identified for each section by the attribute COMPRESSED="B64Z-LittleEndian". To output in ascii format add the module option ":xml:uncompress" to the .xml file, i.e.

NekMesh file.msh newfile.xml:xml:uncompress

4.1 Exporting a mesh from Gmsh

To demonstrate how NekMesh works, we will define a simple channel-like 3D geometry. First, we must define the Gmsh geometry to be used. The Gmsh definition is given below, and is visualised in figure 4.1.

```
1 Point(1) = {-1, 0, 0, 1.0};
2 Point(2) = {-0.3, 0, 0, 1.0};
3 Line(3) = {1, 2};
4 s[] = Extrude {0, 0, 7} {
5 Line{3}; Layers{5}; Recombine;
6 };
7 v[] = Extrude {{0, 0, 1}, {0, 0, 0}, Pi} {
8 Surface{s[1]}; Layers{10}; Recombine;
9 };
```

Whilst a full tutorial on Gmsh is far beyond the scope of this document, note the use of the Recombine argument. This allows us to generate a structured hexahedral mesh; remove the first Recombine to generate a prismatic mesh and both occurances to generate a tetrahedral mesh. Increasing the Layers numbers refines the mesh in the radial and azimuthal direction respectively.

4.2 Defining physical surfaces and volumes



Figure 4.1 Geometry definition in Gmsh (left) and resulting high-order mesh visualised in ParaView (right).

In order for us to use the mesh, we need to define the physical surfaces which correspond to the inflow, outflow and walls so that we can set appropriate boundary conditions. The numbering resulting from the extrusions in this case is not straightforward. In the graphical interface, select **Geometry > Physical Groups > Add > Surface**, and then hover over each of the surfaces which are shown by the dashed gray lines. The numbering will be revealed in the toolbar underneath the geometry as a ruled surface. In this case:

- Walls: surfaces 7, 8, 28, 29.
- Inflow: surface 16.

• Outflow: surface 24.

We also need to define the physical volumes, which can be done in a similar fashion. For this example, there is only one volume having ID 1. Adding these groups to the end of the .geo file is very straightforward:

1 Physical Volume(0) = {1}; 2 Physical Surface(1)= {7,8,28,29}; 3 Physical Surface(2) = {16}; 4 Physical Surface(3) = {24};

Either choose the option File->Save Mesh or, assuming this is saved in a file named test.geo, run the command

gmsh -3 test.geo

which will produce the resulting MSH file test.msh. One can generate a high-order mesh by specifying the order on the command line, for example

gmsh -3 -order 6 test.geo

will generate a sixth-order mesh. Note that you will need to use a current version of Gmsh in order to do this, most likely from subversion.

4.3 Converting the MSH to Nektar++ format

Assuming that you have compiled Nektar++ according to the compilation instructions, run the command

NekMesh test.msh test.xml

to generate the XML file.

Note

8

This file contains only the geometry definition (and a default **EXPANSIONS**) definition). In order to use this mesh, a **CONDITIONS** section must be supplied detailing the solver and parameters to use.

To validate the mesh visually, we can use a utility such as Paraview or VisIt. To do this, we can use the **FieldConvert** command using:

FieldConvert test.xml test.vtu

which generates an unstructured VTK file test.vtu.

It is possible that, when the high-order information was inserted into the mesh by Gmsh, invalid elements are generated which self intersect. In this case, the Jacobian of the mapping defining the curvature will have negative regions, which will generate warnings such as:

```
Warning: Level 0 assertion violation
3D deformed Jacobian not positive (element ID = 48) (first vertex ID = 105)
```

This tells you the element ID that is invalid, and the ID of the first vertex of the element. Whilst a resulting simulation may run, the results may not be valid because of this problem, or excessively large amounts of time may be needed to solve the resulting linear system.

4.4 NekMesh in NekPy

The Python interface allows the user to instantiate input, output, and process modules by calling the static Create method of the InputModule, ProcessModule, and OutputModule, register configuration options, and process them. Consider the following example:

```
1 import sys
2 from NekPy.NekMesh import Mesh, ProcessModule, OutputModule
3
4 \text{ mesh} = \text{Mesh}()
5 \text{ mesh.expDim} = 3
6 \text{ mesh.spaceDim} = 3
7 \text{ mesh.nummode} = 5
8 mesh.verbose = True
9
10 # Load the CAD file
11 ProcessModule.Create("loadcad", mesh, \
                      filename="input.stp", verbose=True).Process()
12
13 # Load the octree
14 ProcessModule.Create("loadoctree", mesh, mindel=0.04,\
15
                       maxdel=0.2, eps=0.02).Process()
16 # Create a surface mesh
17 ProcessModule.Create("surfacemesh", mesh).Process()
18 # Output a 2D manifold mesh
19 mesh.expDim = 2
20 # Create a high-order surface
21 ProcessModule.Create("hosurface", mesh).Process()
22 # Dump out elemental Jacobians
23 ProcessModule.Create("jac", mesh, list=True).Process()
24 # Dump out the surface mesh.
25 OutputModule.Create("xml", mesh, test=True,\
                      outfile="output.xml").Process()
26
```

After importing the Mesh, ProcessModule, and OutputModule classes, first we create a Mesh object by calling the constructor. This object will be shared by the modules. Then we manipulate the Mesh object by creating different ProcessModules, at the end we write out the result into a xml file using an OutputModule. The configuration options for a given module are passed to the static Create method of the InputModule, ProcessModule, and OutputModule. This creates the corresponding module and the modules can be processed immediately after instantiation. Note that the first parameter of the Create method has to be the key for a given module, the second is the previously created Mesh object. The remaining keyword arguments can specify additional parameters for a module.

The Python interface allows the user to create new modules by inheriting from one of the possible base classes (InputModule, ProcessModule, OutputModule).

The following is a simple example when we inherit from the InputModule and override the Process method:

```
1 import sys
2 from NekPy.LibUtilities import ShapeType
3 import NekPy.NekMesh as NekMesh
4 import numpy as np
5
6 # StructuredGrid creates a 2D structured grid of triangles.
7 class StructuredGrid(NekMesh.InputModule):
     def __init__(self, mesh):
8
         super(StructuredGrid, self).__init__(mesh)
9
         self.mesh.spaceDim = 2
10
         self.mesh.expDim = 2
11
12
         # Define some configuration options for this module.
         self.AddConfigOption("nx", "2", "Number of points in x direction")
13
         self.AddConfigOption("ny", "2", "Number of points in y direction")
14
         self.AddConfigOption("lx", "0", "Lower-left x-coordinate")
15
         self.AddConfigOption("rx", "0", "Upper-right x-coordinate")
16
         self.AddConfigOption("ly", "0", "Lower-left y-coordinate")
17
         self.AddConfigOption("ry", "0", "Upper-right y-coordinate")
18
         self.AddConfigOption("compid", "0", "Composite ID")
19
20
     def Process(self):
21
         # Get the input variables from our configuration options.
22
         coord_1x = self.GetFloatConfig("lx")
23
         coord_1y = self.GetFloatConfig("ly")
24
         coord_2x = self.GetFloatConfig("rx")
25
26
         coord_2y = self.GetFloatConfig("ry")
27
         nx = self.GetIntConfig("nx")
28
         ny = self.GetIntConfig("ny")
         compID = self.GetIntConfig("compid")
29
         x_points = np.linspace(coord_1x, coord_2x, nx)
30
         y_points = np.linspace(coord_1y, coord_2y, ny)
31
32
33
         nodes = []
         id_cnt = 0
34
35
```

```
for y in range(ny):
36
37
             tmp = []
38
             for x in range(nx):
39
                 tmp.append(NekMesh.Node(id_cnt, x_points[x], y_points[y],
                      (0.0)
                 id_cnt += 1
40
41
             nodes.append(tmp)
42
          self._create_triangles(nodes, nx, ny, compID)
43
          # Call the Module functions to create all of the edges, faces and
44
          # composites.
          self.ProcessVertices()
45
          self.ProcessEdges()
46
          self.ProcessFaces()
47
          self.ProcessElements()
48
49
          self.ProcessComposites()
50
51
      def _create_triangles(self, nodes, nx, ny, compID):
52
          . . .
53
54 # Register our TestInput module with the factory.
55 NekMesh.Module.Register(
      NekMesh.ModuleType.Input, "StructuredGrid", StructuredGrid)
56
57
58 if __name__ == '__main__':
      # Create a 'pipeline' of the input and output modules.
59
      mesh = NekMesh.Mesh()
60
61
      # First, call our input module's create function from the NekMesh
62
          factory.
63
      NekMesh.InputModule.Create(
64
          "StructuredGrid", mesh,
         nx = sys.argv[1], ny = sys.argv[2], lx = sys.argv[3],
65
         ly = sys.argv[4], rx = sys.argv[5], ry = sys.argv[6],
66
          compid = sys.argv[7], shape = sys.argv[8]).Process()
67
68
69
      # Then ensure there's no negative Jacobians.
      NekMesh.ProcessModule.Create("jac", mesh, list=True).Process()
70
71
72
      # Finally, output the resulting file
73
      NekMesh.OutputModule.Create(
          "xml", mesh, test=True, outfile=sys.argv[9]).Process()
74
```

4.5 NekMesh modules

NekMesh is designed to provide a pipeline approach to mesh generation. To do this, we break up tasks into three different types. Each task is called a *module* and a chain of modules specifies the pipeline.

- Input modules read meshes in a variety of formats;
- Processing modules modify meshes to aide in generation processes;

• Output modules write meshes in a variety of formats.

The figure below depicts how these might be coupled together to form a pipeline: On the



Figure 4.2 Illustrative pipeline of the NekMesh process.

command line, we would define this as:

NekMesh -m process1 -m process2 input.msh output.xml

Process modules can also have parameters passed to them, that can take arguments, or not.

NekMesh -m process1:p1=123:booleanparam input.msh output.xml

To list all available modules use the -1 command line argument:

```
Available classes:
Input: dat:
Reads Tecplot polyhedron ascii format converted from Star CCM (.dat).
...
```

and then to see the options for a particular module, use the -p command line argument:

```
Options for module detect:
vol: Tag identifying surface to process.
```

Note

Module names change when you use the -p option. Input modules should be preceded by (in:), processing modules by proc: and output modules by (out:).

4.5.1 Input modules

Input and output modules use file extension names to determine the correct module to use. Not every module is capable of reading high-order information, where it exists. The table below indicates support currently implemented.

Format	Extension	High-order	Notes
Gmsh	msh	✓	Only reads nodes, elements and physical
			groups (which are mapped to composites).
			File format versions 2.x and 4.x currently
		_	supported.
Nektar	rea	\checkmark	Reads elements, fluid boundary conditions.
			Most curve types are unsupported: high-
			order information must be defined in an
		_	accompanying .hsf file.
Nektar++	xml	\checkmark	Fully supported.
PLY	ply	×	Reads only the ASCII format
Semtex	sem	\checkmark	Reads elements and boundary conditions.
			In order to read high-order information,
			<pre>run meshpr session.sem > session.msh</pre>
			and place in the same directory as the
			session file.
Star-CCM+	dat	×	Star outputs plt file which currently needs
			to be coverted to ascii using Tecplot.
			Reads mesh only, only support for quads
			and triangles (2D) and hexes, prisms,
			tetrahedra $(3D)$.
Star-CCM+	ccm	×	Reads start ccm format. Reads mesh only,
			only support for quads and triangles (2D)
			and hexes, prisms, tetrahedra (3D). Re-
			quires NEKTAR_USE_CCM option to
			be activated in cmake and then requires
			ccmio library to be compiled by user.
VTK	vtk	×	Experimental support. Only ASCII trian-
			gular data is supported.

Note that you can override the module used on the command line. For example, Semtex session files rarely have extensions. So for a session called pipe-3d we can convert this using the syntax

NekMesh pipe-3d:sem pipe-3d.xml

The NekMesh input module also has an option to re-process all composites. By default only composites are reprocessed. However when extracting a smaller mesh from a larger mesh definition by redefining the composite of volumetric elements you can force the edges and/or faces to be reprocessed removing the definition of any edges and/or faces not required in the smaller mesh. This option is called processall and has the syntax

NekMesh input.xml:xml:processall output.xml

Typically, mesh generators allow physical surfaces and volumes to contain many element types; for example a cube could be constructed from a mixture of hexes and prisms. In Nektar++, a composite can only contain a single element type. Whilst the converter will attempt to preserve the numbering of composites from the original mesh type, sometimes a renumbering will occur when a domain contains many element types. For example, for a domain with the tag **150** containing quadrilaterals and triangles, the Gmsh reader will print a notification along the lines of:

Multiple elements in composite detected; remapped: - Tag 150 => 150 (Triangle), 151 (Quadrilateral)

The resulting file therefore has two composites of IDs (150) and (151) respectively, containing the triangular and quadrilateral elements of the original mesh. We note there is one exception to this convention in three-dimensional meshes where a face composite can contain both triangular and quadrilateral elements.

Typically a NekMesh call requires both an input and output module to be called, however, by specifying the output file name or file extension as **stdout** no output file will be created. This option is typically used for mesh statistics processing or inspecting composite values etc. An example call would be:

NekMesh input.xml stdout

or

NekMesh input.xml name.stdout

4.5.2 Output modules

The following output formats are supported:

Format	Extension	High-order	Notes
Gmsh	msh	1	High-order hexes, quads, tetrahedra and tri- angles are supported up to arbitrary order. Prisms supported up to order 4, pyramids
Nektar++ HDF5 VTK	xml nekg vtk	√ √ X	up to order 1. Most functionality supported. Most functionality supported. Outputs mesh only, supports line segments in 1D, triangles and quadrilaterals in 2D and tetrahedra, hexahedra, prisms and pyra- mids in 3D. The VTK legacy format and XML format, both compressed and uncom- pressed, are supported. Requires NEK- TAR_USE_VTK option to be activated in cmake.

Note that for Gmsh, it is highly likely that you will need to experiment with the source code in order to successfully generate meshes since robustness is not guaranteed.

The default for xml and vtk is into binary data which has been converted into base64. If you wish to see an ascii output you need to specify the output module option uncompress. For the uncompressed xml the user can execute:

NekMesh Mesh.msh output.xml:uncompress

If the user wishes to obtain the vtk output in the legacy format, the output module option legacy should be specified by executing:

NekMesh Mesh.xml output.vtk:vtk:legacy

Finally, both the Gmsh and Nektar++ output modules support an **order** parameter, which allows you to generate a mesh of a uniform polynomial order. This is used in the same manner as the above, so that the command

NekMesh Mesh.msh output.msh:msh:order=7

will generate an order 7 Gmsh mesh. In the rest of these subsections, we discuss the various processing modules available within NekMesh.

It is possible to use FieldConvert to extract a smaller region of a mesh from a larger mesh using the "-r xmin,xmax,ymin,ymax,zmin" range option, e.g.

FieldConvert -r xmin,xmax,ymin,ymax,zmin,zmax bigMesh.xml smallMesh.xml

However this will not provide a composite of he faces on the boundary if they were not part of the original boundary composites of the original bigMesh.xml. These can however be recovered by an output module option chkbndcomp in NekMesh. To do this one should specify

NekMesh smallMesh.xml newSmallMesh.xml:xml:chkbndcomp

this will then add a composite for any face that is on the boundary and not part of an existing boundary condition and put it into a composite with *id=9999*. This capability is currently only set up for 3D meshes

4.5.2.1 HDF5 format

NekMesh and all solvers within Nektar++ - along with subsequent FieldConvert modules - also support the HDF5 format. This allows for faster loading of geometries and meshes within each solver - and is a significant improvement over the XML format. HDF5 is recommended input format for any larger cases.

Converting from XML to HDF5 is a simple task that only requires the one NekMesh command:

NekMesh XMLMesh.xml HDF5Mesh.nekg

This will create two files (HDF5Mesh.xml) and (HDF5Mesh.nekg) which are both needed in the same directory to run the simulation. An additional flag in the session file is required, ensuring it is placed before the expansion list being:

1 <GEOMETRY DIM="3" SPACE="3" HDF5FILE="HDF5Mesh.nekg" />

HDF5 also has the additional advantage of ensuring the mesh and session file are split - which allows for easy ammending of the session file - whilst allowing for use of FieldCovnert modules that require only 1 XML input file - rather than having to concatenate the session and mesh XML files. Solvers and any FieldConvert modules can be run by referencing only the session file after the (GEOMETRY) tag is included.

4.5.3 Extract surfaces from a mesh

Often one wants to visualise surfaces of a 3D mesh, or extract the values of variables on the surface and visualise them. To support this, NekMesh can extract two-dimensional surfaces which can be visualised using **FieldConvert** in order to extract the value of a 3D field on a given surface.

As an example, we can extract composite surfaces 2 and 3-5 from a mesh using the **extract** module:

```
NekMesh -m extract:surf=2,3-5 Mesh.xml output.xml
```

If you also wish to have the boundaries of the extracted surface detected add the detectbnd option

NekMesh -m extract:surf=2,3-5:detectbnd Mesh.xml output.xml

which will produce new composites for the extracted boundary.

4.5.4 Negative Jacobian detection

To detect elements with negative Jacobian determinant, use the jac module:

NekMesh -m jac Mesh.xml output.xml

To get a detailed list of elements which have negative Jacobians, one may use the list option:

NekMesh -m jac:list Mesh.xml output.xml

and to extract the elements for the purposes of visualisation within the domain, use the **extract** boolean parameter:

NekMesh -m jac:extract Mesh.xml MeshWithNegativeElements.xml

To turn off curvature associated with negative jacobians one can try to use the **linearise** module:

NekMesh -m linerise:invalid Mesh.xml output.xml

This option will remove all high order curvature on all element types with singular jacobians.

4.5.5 Spherigon patches

Where high-order information is not available (e.g. when using meshes from imaging software), various techniques can be used to apply a smoothing to the high-order element. In NekMesh we use *spherigons*, a kind of patch used in the computer graphics community used for efficiently smoothing polygon surfaces.

Spherigons work through the use of surface normals, where in this sense 'surface' refers to the underlying geometry. If we have either the exact or approximate surface normal at each given vertex, spherigon patches approximate the edges connecting two vertices by arcs of a circle. In NekMesh we can either approximate the surface normals from the linear elements which connect to each vertex (this is done by default), or supply a file which gives the surface normals.

To apply spherigon patches on two connected surfaces 11 and 12 use the following command:

```
NekMesh -m spherigon:surf=11,12 \
MeshWithStraighEdges.xml MeshWithSpherigons.xml
```

If the two surfaces "11" and "12" are not connected, or connect at a sharp edge which is C^0 continuous but not C^1 smooth, use two separate instances of the spherigon module.

```
NekMesh -m spherigon:surf=11 -m spherigon:surf=12 \
MeshWithStraighEdges.xml MeshWithSpherigons.xml
```

This is to avoid the approximated surface normals being incorrect at the edge.

If you have a high-resolution mesh of the surfaces 11 and 12 in ply format it can be used to improve the normal definition of the spherigons. Run:

NekMesh -m spherigon:surf=11,12:usenormalfile=Surf_11-12_Mesh.ply \
 MeshWithStraighEdges.xml MeshWithSpherigons.xml

This can be useful, for example, when meshing the Leading edge of an airfoil. Starting from a linear mesh (left figure) the spherigon patches curve the surface elements producing leading edge closer to the underlying geometry:



Figure 4.3 (a) Leading edge without spherigons, (b) Leading edge with spherigons

4.5.6 Periodic boundary condition alignment

When using periodic boundary conditions, the order of the elements within the boundary composite determines which element edges are periodic with the corresponding boundary composite.

To facilitate this alignment, NekMesh has a periodic alignment module which attempts to identify pairs of mutually periodic edges. Given two surfaces surf1 and surf2, which for example correspond to the physical surface IDs specified in Gmsh, and an axis which defines the periodicity direction, the following command attempts to reorder the composites:

NekMesh -m peralign:surf1=11:surf2=12:dir=y \
 -m peralign:surf1=13:surf2=14:dir=z Mesh.xml Mesh_aligned.xml

Here the surfaces with IDs 11 and 12 will be aligned normal to the y-axis and the surfaces 13 and 14 will be aligned normal to the z-axis.

Note that this command cannot perform magic – it assumes that any given edge or face lying on the surface is periodic with another face on the opposing surface, that there are the same number of elements on both surfaces, and the corresponding edge or face is the same size and shape but translated along the appropriate axis.

When using periodic boundary conditions that are rotationally aligned the following rotational options should be applied:

NekMesh -m peralign:surf1=11:surf2=12:dir=x:rot=PI/6 Mesh.xml Mesh_aligned.xml

where **rot** specifies the rotation angle in radians from surf1 to surf2 about the axis specified by **dir** (i.e. the "x" axis in this example).

The rotation/translation is assumed to be exact within a relative tolerance. An optional factor, which is used to scale the tolerance, **tolfact** can also be specified. The default tolerance factor is 4, and it needs to be $tolfact \ge 1$. For example:

```
NekMesh -m peralign:surf1=11:surf2=12:dir=x:rot=PI/6:tolfact=100 \
Mesh.xml Mesh_aligned.xml
```

In 3D, where prismatic or tetrahedral elements are connected to one or both of the surfaces, additional logic is needed to guarantee connectivity in the XML file. In this case we append the **orient** parameter:

NekMesh -m peralign:surf1=11:surf2=12:dir=y:orient input.dat output.xml

Note

One of the present shortcomings of **orient** is that it throws away all high-order information and works only on the linear element. This can be gotten around if you are just doing e.g. spherigon patches by running this **peralign** module before the **spherigon** module.

4.5.7 Boundary layer splitting

Often it is the case that one can generate a coarse boundary layer grid of a mesh. NekMesh has a method for splitting prismatic and hexahedral elements into finer elements based on the work presented in [33] and [34]. You must have a prismatic mesh that is *O*-type – that is, you can modify the boundary layer without modifying the rest of the mesh.

Given n layers, and a ratio r which defines the relative heights of elements in different layers, the method works by defining a geometric progression of points

$$x_k = x_{k-1} + ar^k, \quad a = \frac{2(1-r)}{1-r^{n+1}}$$

in the standard segment [-1, 1]. These are then projected into the coarse elements to construct a sequence of increasingly refined elements, as depicted in figure 4.4.



Figure 4.4 Splitting Ω_{st} and applying the mapping χ^e to obtain a high-order layer of prisms from the macro-element.

To split a prism boundary layer on surface 11 into 3 layers with a growth rate of 2 and 7 integration points per element use the following command:

NekMesh -m bl:surf=11:layers=3:r=2:nq=7 MeshWithOnePrismLayer.xml MeshWith3PrismsLayers.xml



Figure 4.5 (a) LE with Spherigons but only one prism layer for resolving the boundary layer, (b) LE with Spherigons with 3 growing layers of prisms for better resolving the boundary layer.

Note

You can also use an expression in terms of coordinates (x, y, z) for r to make the ratio spatially varying; e.g. r=sin(x). In this case the function should be sufficiently smooth to prevent the elements self-intersecting.

4.5.8 High-order cylinder generation

Generating accurate high-order curved geometries in Gmsh is quite challenging. This module processes an existing linear cylindrical mesh, with axis aligned with the z-coordinate axis, to generate accurate high-order curvature information along the edges.

NekMesh -m cyl:surf=2:r=1.0:N=5 LinearCylinder.xml HighOrderCylinder.xml

The module parameters are:

- **surf**: Surface on which to apply curvature. This should be the outer surface of the cylinder.
- **r**: Radius of the cylinder.
- N: Number of high-order points along each element edge.

Note

The module could also be used to apply curvature along the interior of a hollow cylinder. However, there are no checks to ensure the resulting elements are not self-intersecting.

4.5.9 Linearisation

The ability to remove all the high-order information in a mesh can be useful at times when mesh generation is tricky or impossible in the presence of curvature

To do this in NekMesh use the command:

```
NekMesh -m linearise:all high-order-mesh.xml linear-mesh.xml
```

The output will contain only the linear mesh information, all curved information is removed. Alternatively

```
NekMesh -m linearise:invalid high-order-mesh.xml linear-mesh.xml
```

attempts to remove curvature from elements only where necessary. This is a simple algorithm that removes curvature from invalid elements and repeats until all elements are valid. Either all or invalid must be specified.

- all: remove curvature from all elements.
- invalid: remove curvature from invalid elements.
- **prismonly**: consider only prisms when removing curvature. This is useful in the presence of a prismatic boundary layer.

4.5.10 Extracting interface between tetrahedra and prismatic elements

When the mesh is three-dimensional and comprised of a prismatic boundary layer with tetrahedra in the interior of the domain, this module extracts the prismatic elements only, and constructs a boundary region for the interface between the tetrahedra and prisms. This is useful in, for example, the study of aortic flows, where the prismatic boundary layer can be extracted and refined to study unsteady advection-diffusion problems on a more refined grid inside the boundary layer.

To use this module you therefore use the command:

```
NekMesh -m extracttetprisminterface input.xml output.xml
```

There are no configuration options for this module, as it is highly specific to a certain class of meshes.

4.5.11 Boundary identification

Some mesh formats lack the ability to identify boundaries of the domain they discretise. NekMesh has a rudimentary boundary identification routine for conformal meshes, which will create a composite of edges (2D) or faces (3D) which are connected to precisely one element. This can be done using the detect module:

NekMesh -m detect volume.xml volumeWithBoundaryComposite.xml

4.5.12 Scalar function curvature

This module imposes curvature on a surface given a scalar function z = f(x, y). For example, if on surface 1 we wish to apply a surface defined by a Gaussian $z = \exp[-(x^2 + y^2)]$ using 7 quadrature points in each direction, we may issue the command

NekMesh -m scalar:surf=1:nq=7:scalar=exp\(x*x+y*y\) mesh.xml deformed.xml

Note

This module makes no attempt to apply the curvature to the interior of the domain. Elements must therefore be coarse in order to prevent self-intersection. If a boundary layer is required, one option is to use this module in combination with the splitting module described earlier.

4.5.13 Link Checking

It is quite possible that a mesh contains some sort of hanging entity or element connectivity error. The check link module is a fast check that, a) elements are correctly connected and b) the boundary entities (composites) match the interior domain:

NekMesh -m linkcheck mesh.xml mesh2.xml

This module should be added to the module chain if the user suspects there may be a mesh issue. The module will print a warning if there is a connectivity error.

4.5.14 2D mesh extrusion

This module allows a 2D mesh, quads, triangles or both, to be extruded in the z direction to make a simple 3D mesh made of prisms and hexahedra. It is also capable of extruding the high-order curvature within the 2D mesh. The module requires two parameters:

```
NekMesh -m extrude:layers=n:length=l 2D.xml 3D.xml
```

length which determines how long the z extrusion will be and layers, the number of elements in the z direction.

4.5.15 Variational Optimisation

This module can correct invalid and improve the quality of elements in high-order meshes by applying curvilinear deformation to the interiors of domains. It achieves this by solving a solid mechanics system which, using variational calculus has been cast is a non-linear energy optimisation problem. It is basis of the work in [46].

It works by considering the boundary (curved) mesh entities to be fixed and moving the interior nodes to a lower energy configuration. This new configuration in most scenarios is a higher quality mesh. The energy is evaluated depending on which functional is chosen. We find hyperleastic to be the most reliable but it can also model the mesh and a linearelastic solid as well as functionals based on the Winslow equation and the distortion method proposed by Roca et al. [14].

There are a large number of options which can be viewed using the help function but the basic usage is:

NekMesh -m varopti:type inital.xml optimised.xml

where type can be hyperelastic, linearelastic, winslow or roca.

4.5.16 *r*-adaptation

This module can deform an existing mesh by using the variational optimiser presented above. A file must be provided that contains a list of points and a scaling value for each of them. This scaling factor is then used to target an element size based on the initial size of the element. Scaling values are interpolated throughout the domain based on the interpolation method of the main library. The file should look like

where the first three columns are x, y, z and the last column is the scaling factor.

The call is identical to the variational optimisation module above:

```
NekMesh -m varopti:type:scalingfile=file.txt:subiter=x inital.xml adapted.
xml
```

where **subiter** is an additional parameter to the variational optimiser that defines the frequency at which individual elements update their target scaling based on their latest location in the domain. **subiter** should be a scalar and is the number of steps between updates. It is often recommended to run *r*-adaptation on a linear mesh for stability and

performance reasons. Note also that the mesh must have CAD information in order for nodes to slide on curves and surfaces.

4.5.17 Mesh projection

This module can take any linear mesh, providing that it is a close representation of the CAD and project the boundary of the mesh onto the CAD. This will curve the surface of the mesh. The method has a number of failsafes ensuring that even bad CAD or poor linear meshes should be able to be curved to some degree. If the method encounters an issue, such as the linear mesh being a large distance from the CAD, it will simply leave that element straight sided. A well made CAD model and accurate linear mesh should be curved with little issue.

The module needs to be informed of the CAD file to project the mesh to and the order at which to curve the surface:

NekMesh -m projectcad:file=cadfile.step:order=x inital.xml optimised.xml

4.6 Mesh generation

In addition to the functionality described previously, NekMesh is capable of generating high-order meshes directly from a CAD definition. By default this functionality is not activated, a user wishing to utilise the mesh generation capability of NekMesh must compile Nektar++ with the NEKTAR_USE_MESHGEN option on. As well as compiling the relevant routines into NekMesh it will also download a number of other packages which are required.

The most critical dependancy of the mesh generation routines is OpenCascade which powers the CAD engine. NekMesh is capable of finding and using existing installations of OpenCascade 6.8 or OCE 0.17. If either are not present on the installation machine NekMesh will install OCE 0.17 from source. This is a very big installation and will take some time so it is advised that the user ensures OpenCascade is available on the machine.

As with all tasks within NekMesh the mesh generation capability exists as its own separate module which is of type Input. Due to the vast amount of code associated with the generation of high-order meshes and the comparatively small nature of modules in the NekMesh program a new library has been created for Nektar++ called NekMeshUtils, which contains all the core routines and classes for the NekMesh mesh format as well as a series of classes for the generation of meshes. This library also contains the CAD API for Nektar++ which is used to generate the meshes.

4.6.1 Methodology

This section outlines the approach taken by NekMesh to generate high-order meshes. To simplify the sometimes very complicated high-order mesh generation processes in other

programs, NekMesh executes all the stages required to produce a high-order mesh in one single pipeline which once started requires no interaction from the user. In broad terms these stages are:

- Specification of the element sizes in the mesh,
- Coarse linear mesh generation of the domain,
- Generation of optimised high-order surface on the geometric boundary,

and are outlined in more detail in the following sections.

4.6.1.1 CAD Interaction

At the core of all the ideas in the NekMesh generator is that the final mesh is a high quality representation of the underlying geometry. As such all of the entities in the mesh must know where they are located with respect to the CAD and the system to be able to query any geometric information at any point in the domain easily and with accuracy. To handle this NekMesh has been interfaced with the third-party suite of CAD libraries called OpenCascade. In its normal state OpenCascade is a very large collection of libraries with tens of thousands of functions which are simply not needed for our purposes, because of this its installation is a very arduous and long process. Combine this with the fact that there are dozens of versions and types of OpenCascade, such as OpenCascade Community Edition, it is simply impossible for NekMesh to use already existing OpenCascade installations on a given machine. To solve these issues, when installing Nektar++ with the mesh generator it will download pre-compiled binaries for the relevant OS and link against those, any previously installed versions of OpenCascade will not be searched for and therefore ignored. To reduce the massively complex libraries in OpenCascade down to a manageable set of functions to be used in NekMesh a set of interface classes have been created which act as buffer between it and Nektar++. These CAD classes mean that development of mesh generation routines is significantly easier and in the future Nektar++ developers will be able to utilise CAD information in all aspects of the framework without having to learn OpenCascade. Another advantage with this approach is that adding support for other CAD engines, as well as OpenCascade, in the future should be relativity simple and will not require the rewriting of any of the NekMesh code.

4.6.1.2 Automatic specification of the mesh

One of the key challenges of generating a high-order mesh is the creation of a suitable coarse linear mesh. It is quite difficult for a user to define a full set spacings over a whole domain which will produce a good quality especially when aiming for coarseness. This is tackled in NekMesh with a system for automatically defining a set of smooth and coarse mesh spacings throughout the whole domain. This is achieved using an octree description of the domain. The domain is recursively subdivided into octants which each describe
a small portion of the domain. The level to which the domain subdivides is based on the curvature of the geometric boundary. Higher curvature regions will subdivide to a finer level allowing for increased control on the mesh specification and smoothness. The geometric curvature is then related to a mesh sizing parameter and propagated throughout the domain ensuring a smooth mesh. For those unfamiliar with octrees, it is best to think of it as a non-conforming hexahedral mesh

4.6.1.3 Linear Mesh Generation

The first challenge mentioned in the previous section is addressed with the NekMesh approach to linear mesh generation. Primarily because of the difficulties in interfacing existing linear mesh generators for high-order applications the decision was made to include a bespoke linear mesh generator within the program. Compared with the mesh generators included in commercial packages this linear mesh generator takes the quite unconventional and more historic approach in building the mesh in a bottom up fashion from 0D to 3D. Using this approach means it is possible to guarantee a level of boundary conformity which direct to 3D approaches cannot at the desired level of coarseness. In this approach, first mesh nodes are placed on the vertices of the CAD model (0D), then the curves in the CAD are meshed in 1D using the vertex nodes as boundaries, then the surfaces are meshed in their 2D parameter plane using the curve meshes as boundaries and finally the 3D volume is meshed using the surface mesh as the boundary to complete the linear mesh. In NekMesh, to achieve greater robustness, the 2D mesh generation library Triangle is used and the TetGen library for the 3D. Both of which are highly developed Delaunay based mesh generators. As with all additional libraies in Nektar++these are automatically downloaded and installed if needed.

4.6.1.4 High-order Surface Generation

Addition of the high-order nodes to and the curving of the mesh is very open problem, no high-order mesh generator has solved this and while the methods used in NekMesh are not 100% full-proof, the system currently in place can create good quality high-order curved meshes with a reasonable robustness. This area will receive the greatest level of development in the future. The most critical part of defining the high-order mesh is the addition of high-order nodes on the geometric surface. The mesh generator must achieve the greatest level of geometric accuracy as it can otherwise it will greatly affect the final flow solutions. If the linear surface triangulation is taken to be fixed during this process, the problem can be addressed in a element by element fashion. If the high-order nodes are placed by simply using an affine mapping to the CAD surface and back the resulting high-order triangle will inherit the same distortions as the CAD surface. To solve this NekMesh uses a system node location optimisation in the parameter plane of the CAD surface to ensure the high-order triangles have as little distortion as possible while remaining exactly on the geometric surface. To do this the system models the high-order edges and triangles as a network of springs with an associated spring energy which is minimised using a multidimensional Newton type optimisation procedure with a Gauss-Seidel matrix solver.

98 Chapter 4 NekMesh

4.6.1.5 Mesh Correction

Due to the fact that, for the time being, no consideration is given to the curving of mesh interior entities explicitly in the mesh generation process, the curving the geometric surface can produce meshes with invalid elements, especially in the case of Euler type (Tetrahedra only) meshes. Three strategies exist within Nektar++ to correct these elements. Firstly removing the curvature, by removing the curvature of invalid elements they become valid. However this has the massive downside of compromising the geometric accuracy of the mesh but is quick and effective, this can be enacted using the command:

NekMesh -m linearise:invalid invalidMesh.xml validMesh.xml

An alternative to this is to use the linear elastic solver within Nektar++ to deform the mesh interior entities. Its use is very computationally expensive, as with all PDE solvers, and is also not particularly robust. It can be used with the set of commands outlined in the FieldConvert deform and displacement modules and the section on the Linear Elastic Solver.

The final and possibly most useful approach is to use the Variational Optimisation module to curve the interior of the domain. This is explained in 4.5.15.

4.6.2 Mesh generation manual

The mesh generation is executed with the command:

NekMesh session.mcf mesh.xml

where session.mcf is a mesh configuration file which contains all the options and parameters needed for mesh generation. Below is an example of a simple example which generates a 2D NACA wing.

```
<NEKTAR>
1
           <MESHING>
2
3
               <INFORMATION>
4
                   <I PROPERTY="CADFile"
                                              VALUE="6412"
                                                             />
\mathbf{5}
                    <I PROPERTY="MeshType" VALUE="2D"
6
                                                             />
               </INFORMATION>
7
8
9
               <PARAMETERS>
                                             VALUE="0.01"
10
                   <P PARAM="MinDelta"</pre>
                                                             1>
                                             VALUE="1.0"
11
                    <P PARAM="MaxDelta"
                                                             />
                    <P PARAM="EPS"
                                              VALUE="0.1"
12
                                                             1>
                    <P PARAM="Order"
                                              VALUE="4"
                                                             1>
13
14
                    <!-- 2D Domain !-->
15
                    <P PARAM="Xmin"
                                              VALUE="-1.0"
                                                             />
16
                    <P PARAM="Ymin"
                                              VALUE="-2.0"
17
                                                            />
                                              VALUE="3.0"
                    <P PARAM="Xmax"
                                                             />
18
```

```
      19
      <P PARAM="Ymax"</td>
      VALUE="2.0" />

      20
      <P PARAM="AOA"</td>
      VALUE="15.0" />

      21
      </PARAMETERS>

      22

      23
      </MESHING>

      24
      </NEKTAR>
```

In all cases the mesh generator needs two pieces of information and four parameters. It firstly needs to know the CAD file with which to work. In the example above this is listed as a 4 digit number, this is because the mesh generator is equiped with a NACA wing generator. In all other cases this parameter would be the name of a CAD file (in either STEP or GEO format). Secondly, what type of mesh to make, the options are (EULER) and (BndLayer) for 3D meshes and (2D) and (2DBndLayer) for 2D meshes. In the case of **EULER**) the mesh will be made with only tetrahedra. For **BndLayer** the mesh generator will attempt to insert a single macro prism layer onto the geometry surface. This option requires additional parameters. This is similar for the 2D scenarios. The automatic mesh specification system requires three parameters to build the specification of a smooth, curvature refined mesh. Firstly (MinDelta) which is the size of the smallest element to be found in the final mesh. Secondly (MaxDelta) which is the maximum size of an element in the mesh and lastly (EPS) which is a sensitivity to curvature parameter with a range $1 \ge \varepsilon > 0$ which heuristically controls the size of the elements for a given degree of curvature on the geometric surface. (Order) is the polynomial order of the mesh to be generated. When generating a boundary layer mesh a few additional parameters must be given. An example is shown.

1	<nektar></nektar>			
2	<meshing></meshing>			
3				
4	<information></information>			
5	<i <="" property="CADFile" td=""><td>VALUE="6412"</td><td>/></td><td></td></i>	VALUE="6412"	/>	
6	<i <="" property="MeshType" td=""><td>VALUE="2DBndLayer"</td><td>/></td><td></td></i>	VALUE="2DBndLayer"	/>	
$\overline{7}$				
8				
9	<parameters></parameters>			
10	<pre><p <="" param="MinDelta" pre=""></p></pre>	VALUE="0.01"	/>	
11	<pre><p <="" param="MaxDelta" pre=""></p></pre>	VALUE="1.0"	/>	
12	<pre><p <="" param="EPS" pre=""></p></pre>	VALUE="0.1"	/>	
13	<pre><p <="" param="Order" pre=""></p></pre>	VALUE="4"	/>	
14				
15	Boundary layer !			
16	<pre><p <="" param="BndLayerSurfaces" pre=""></p></pre>	VALUE="5,6"	/>	
17	<pre><p <="" param="BndLayerThickness" pre=""></p></pre>	VALUE="0.03"	/>	
18	<pre><p <="" param="BndLayerLayers" pre=""></p></pre>	VALUE="4"	/>	
19	<pre><p <="" param="BndLayerProgression" pre=""></p></pre>	VALUE="2.0"	/>	
20				
21	2D Domain !			
22	<pre><p <="" param="Xmin" pre=""></p></pre>	VALUE="-1.0"	/>	
23	<pre><p <="" param="Ymin" pre=""></p></pre>	VALUE="-2.0"	/>	
24	<pre><p <="" param="Xmax" pre=""></p></pre>	VALUE="3.0"	/>	
25	<pre><p <="" param="Ymax" pre=""></p></pre>	VALUE="2.0"	/>	
26	PARAM="AOA"	VALUE="15.0"	/>	

100 Chapter 4 NekMesh

27 </PARAMETERS> 28 29 </MESHING> 30 </NEKTAR>

A list of the CAD surfaces which will have a prism generated on is described by **BndLayerSurfaces** and the thickness of the boundary to aim for is **BndLayerThickness**. The mesh generator has been created with a range of error messages to aid in debugging. If you encounter an error and the mesh generator fails, run NekMesh with the verbose -v flag and send the stdout with the .mcf and .step files to d.moxey@exeter.ac.uk. Without the feedback this functionality cannot improve.

4.6.2.1 Handling 3D geometries with voids

Although NekMesh supports the definition of 3D geometries that contain voids – for example, a sphere contained within a cube – at present it does require the definition of a point per-void that lies strictly on the interior of the void. This is so that tetrahedra on the interior of the void can be removed before the mesh is generated. For example, if one defines a geometry where two spheres of radius 1, centred at (0,0,0) and (2,0,0), were contained within a larger domain, the void points can be specified through the **(VOIDPOINTS)** tag in the MCF as follows:

```
<NEKTAR>
1
2
            <MESHING>
3
                <!-- other parameters here... -->
4
                 <VOIDPOINTS>
                      <V> 0 0 0 </V>
\mathbf{5}
                      <V> 2 0 0 </V>
6
                 </VOIDPOINTS>
\overline{7}
8
          </MESHING>
       </NEKTAR>
9
```

4.6.2.2 GEO format

Recent developments have been made to facilitate the generation of meshes from simple 2D and 3D geometries. The GEO file format, used by Gmsh, is a popular option that allows the user to script geometrical and meshing operations without the need of a GUI. A simplified reader has been implemented in NekMesh for 2D and 3D geometries. Although very basic this reader may be extended in the future to cover a wider range of geometrical features.

For a full description of the GEO format the user should refer to Gmsh's documentation. The following commands are currently supported:

- // and /* */ (i.e. comments)
- Point

- Line
- Spline (through points)
- BSpline (i.e. a Bézier curve)
- Ellipse (arc): as defined in Gmsh's OpenCASCADE kernel, the first point defines the start of the arc, the second point the centre and the fourth point the end. The third point is not used. The start point along with the centre point form the major axis and the minor axis is then computed so that the end point falls onto the arc. The major axis must always be greater or equal to the minor axis.
- **Circle** (arc): the circle is a special case of the ellipse where the third point is skipped. The distances between the start and end points and the centre point must be equal or an error will be thrown.
- Line Loop
- Plane Surface
- Ruled Surface or, in newer versions of Gmsh, Surface
- Surface Loop
- Volume

At the present time, NekMesh does not support the full scripting capabilities of the GEO format, but the evaluation of simple variables is supported. The used GEO files should be a straightforward succession of entity creations (see list above). This should however allow for the creation of quite a wide range of 2D and 3D geometries by transformation of arbitrary curves into generic splines and arcs.

4.6.2.3 CCM format

For very complex 3D geometries with manifold bodies, Nektar++ can take advantage of the meshing tool embedded in the Star-CCM package. NekMesh projects linear meshes generated in Star-CCM on the CAD definition to generate high-order meshes. Nektar++ must be built NEKTAR_USE_CCM in conjunction with the ccmio package to use this functionality. Projection is performed by specifying the option projectcad.

The supported element types for the linear mesh are :

- Tetrahedron
- Prism
- Pyramid

102 Chapter 4 NekMesh

Therefore it is recommended to use the Surface Remesher, the Tetrahedral Mesher and the Prism Layer Mesher in Star-CCM. The CAD Projection and Generate Only Standard Prismatic Cells options should be activated. The prismatic layer for the linear mesh in Star-CCM should not be splitted e.g. there should be only one layer with the total thickness of the boundary layer (see figure 4.6. Splitting will then be perfomed directly by NekMesh on surfaces specified by the argument surf and the number of layers (layers argument) and growth rate (r argument) are specified during the mesh conversion process with the bl option. Surfaces correspond to the regions created in Star-CCM and the reference index is displayed when running NekMesh with verbosity (-v option).

NekMesh -m projectcad:file=cadFile.STEP:order=meshOrder -m bl:surf=surf1, surf2,surf3:r=growthRate:layers=nLayers linearMesh.ccm outputMesh.xml



The command linearise can be added to correct bad elements.

Figure 4.6 Linear mesh generated in Star-CCM (visualised with ParaView) showing the one-layer prismatic cells layer

CHAPTER 5

FieldConvert

FieldConvert is a utility embedded in Nektar++ with the primary aim of allowing the user to convert the Nektar++ output binary files (.chk and .fld) into formats which can be read by common visualisation and post-processing software, primarily Paraview/VisIt (in unstructured VTK .vtu format) or Tecplot/VisIt (in ASCII .dat or binary .plt formats). FieldConvert also allows the user to manipulate the Nektar++ output binary files by using some additional modules which can be called with the option -m which stands for module. Note that another flag, -r (which stand for r ange) allows the user to specify a sub-range of the domain on which the conversion or manipulation of the Nektar++ output binary files will be performed.

Almost all of the FieldConvert functionalities can be run in parallel if Nektar++ is compiled using MPI (see the installation documentation for additional info on how to implement Nektar++ using MPI).¹

5.1 Basic usage

FieldConvert expects at least one input specification (such as a session file and its corresponding field file) and one output specification. These are specified on the command line as

FieldConvert in1.xml in2.fld out.dat

These can be combined with a processing module by adding the -m command line option. There can be more than one module specified, and they can appear anywhere in the command line arguments, although the order of execution is inferred from their order in the command line. For example, the command

FieldConvert in1.xml -m module1 in2.fld -m module2 out.dat

¹Modules that do not have parallel support will be specified in the appropriate section.

causes in1.xml and in2.fld to be read, followed by the module1 processing module, the module2 processing module, and finally output to the out.dat Tecplot file.

5.1.1 Input formats

FieldConvert supports XML and FLD-format files as produced by Nektar++. It also supports the reading of data files from two external spectral element codes: $Semtex^2$ and $Nek5000^3$. These files can be directly converted to Nektar++ format files by using the command

FieldConvert input.fld output.fld

Note that even though the .fld extension is typically associated with Nektar++ files, FieldConvert can automatically identify Semtex and Nek5000 input field files.

To use these files in a simulation, or to post-process the results of a simulation, an appropriate mesh must also be defined in the Nektar++ XML format. NekMesh can be used to convert these input files to XML, as outlined in section 4.

5.2 Convert .fld / .chk files into Paraview, VisIt or Tecplot format

To convert the *Nektar++* output binary files (.chk and .fld) into a format which can be read by two common visualisation softwares: Paraview (.vtu format), VisIt (.vtu format) or Tecplot (.dat or .plt format) the user can run the following commands:

• Paraview or VisIt (.vtu format)

FieldConvert test.xml test.fld test.vtu

• Tecplot (.dat format)

FieldConvert test.xml test.fld test.dat

• Tecplot or VisIt(.plt format)

```
FieldConvert test.xml test.fld test.plt
```

where FieldConvert is the executable associated to the utility FieldConvert, test.xml is the session file and test.vtu, test.dat, test.plt are the desired format outputs, either Paraview, VisIt or Tecplot formats.

 $^{^{2} \}rm http://users.monash.edu.au/\ bburn/semtex.html$

³https://nek5000.mcs.anl.gov

When converting to .dat or .plt format, it is possible to enable output with double precision, which is more accurate but requires larger disk space. For example, double precision output in plt. format can be produced with the command:

FieldConvert test.xml test.fld test.plt:plt:double



5.2.1 Using the VTK library for output

While the basic VTK output detailed above (.vtu unstructured grid files) works without the third party VTK library, Nektar++ can also optionally be linked with the VTK library. By enabling VTK library support the creation of the VTK output files is offloaded to the VTK library rather than being manually written by Nektar++.

If you want to build a version of *Nektar++* with integrated VTK library support, you need to turn on the **NEKTAR_USE_VTK** switch in cmake. This will then look for a compatible VTK library on the system, VTK version 8.0.0 or higher is recommended for full functionality because this is the minimum version which allows for high-order VTK output. Versions older than this are supported but the high-order VTK output will be disabled.

Using the VTK library automatically enables file compression on the output .vtu file which can substantially reduce file sizes. This can be turned off with the command line option uncompress when using FieldConvert.

FieldConvert session.xml field.fld output.vtu:vtu:uncompress

If Nektar++ has been compiled with NEKTAR_USE_VTK enabled then there are certain situations in which the legacy VTK output (manually writing rather than using the VTK library) must be used. The current VTK library output only supports 1D, 2D, 3D and 3DH1D (Quasi-3D) simulations. Any simulations with expansions other than those mentioned, such as 2DH2D will need to be converted using the legacy option to invoke the manual Nektar++ VTK output.

FieldConvert session.xml field.fld output.vtu:vtu:legacy

The VTK library also adds two new output types that can be specified by the command line when choosing to output a .vtu format file, these are detailed below.

5.2.1.1 High-order output

High-order output can be enabled using the option highorder which first runs the mesh through the equispacedoutput module to get a truly equispaced mesh and then constructs a .vtu file using arbitrary-order Lagrange polynomials. This gives a more precise representation of the curvature of elements and allows for polynomial refinement using non-linear subdivisions. This should result in smaller output files as refinement of the mesh geometry can be computed on the fly. This allows for faster postprocessing in software such as Paraview due to the reduced memory requirements. High-order output is currently only implemented for non-homogenous expansions and does not support pyramid elements. A minimum VTK version of 8.0.0 is required for the high-order output, any VTK versions older than this will result in the high-order output being disabled.

FieldConvert session.xml field.fld output.vtu:vtu:highorder

5.2.1.2 Multi-block output

Multi-block output can be enabled using the option multiblock which splits the mesh in to separate blocks, one block for each domain, boundary, and any other composite. This allows each block to be toggled in the visualisation software independently.

Multi-block output uses the composite/boundary/domain names defined in the session file to label each block, so the ID numbers above will be replaced by the name if specified. It is also necessary to include the **CONDITIONS** section in the session file supplied to **FieldConvert** because this is where the boundary definitions are stored. Multi-block output is currently only implemented for non-homogenous expansions i.e. 1D, 2D, and 3D.

FieldConvert session.xml conditions.xml field.fld output.vtu:vtu:multiblock

An example multi-block tree structure is shown in figure 5.1, with the default output on the left and the output with the optional defined names on the right.

5.2.1.3 Improved FieldConvert filter

An additional advantage of using the VTK library is that when a filter is set up with **FieldConvert** to output a .vtu file, the mesh is now cached inbetween timesteps and only the new expansion data is written each filter call. This can provide significant speedups by keeping the geometry in memory. An example filter specifying a high-order and uncompressed .vtu output is shown below:



Figure 5.1 An example of the tree structure for a multi-block VTK output file. With the default IDs on the left and the optional names specified on the right.

5.3 Convert field files between XML and HDF5 format

When Nektar++ is compiled with HDF5 support, solvers can select the format used for output of .fld files. FieldConvert can be used to convert between these formats using an option on the .fld output module. For example, if in.fld is stored in the default XML format, it can be converted to HDF5 format by issuing the command

```
FieldConvert in.fld out.fld:fld:format=Hdf5
```

5.4 Range option -r

The Fieldconvert range option \neg allows the user to specify a sub-range of the mesh (computational domain) by using an additional flag, \neg (which stands for \neg ange and either convert or manipulate the *Nektar++* output binary files. Taking as an example the conversion of the *Nektar++* binary files (.chk or .fld) shown before and wanting to convert just the 2D sub-range defined by $-2 \le x \le 3$, $-1 \le y \le 2$ the additional flag \neg can be used as follows:

• Paraview or VisIt (.vtu format)

FieldConvert -r -2,3,-1,2 test.xml test.fld test.vtu

• Tecplot (.dat format)

FieldConvert -r 2,3,-1,2 test.xml test.fld test.dat

where $-\mathbf{r}$ defines the range option of the FieldConvert utility, the two first numbers define the range in x direction and the the third and fourth number specify the y range. A sub-range of a 3D domain can also be specified. For doing so, a third set of numbers has to be provided to define the z range.

5.5 FieldConvert in NekPy

The Python interface allows the user to instantiate input, output, and process modules by calling the static Create method of the InputModule, ProcessModule, and OutputModule, register configuration options, and run them. For example, consider the command:

```
FieldConvert -n 10 -m wss:bnd=0 session.xml field.fld field_wss.fld
```

A Python script performing the same task is given below.

```
1 import sys
2 from NekPy.FieldUtils import *
3
4 field = Field(sys.argv, output_points=10)
5
6 InputModule.Create("xml", field, "session.xml").Run()
7 InputModule.Create("fld", field, "field.fld").Run()
8 ProcessModule.Create("wss", field, bnd="0").Run()
9 OutputModule.Create("fld", field, "field_wss.fld").Run()
```

The key points are that the FieldConvert command line options, in this case output-points, are passed to the constructor of Field. The configuration options for a given module are passed to the static Create method of the InputModule, ProcessModule, and OutputModule. This creates the corresponding module and the modules can be run immediately after instantiation. Note that the first parameter of the Create method has to be the key for a given module, the second is the field variable and for input and output modules the remaining arguments may identify the input and output files for a given module. Optionally we can explicitly specify the file type of an input module using the "infile" keyword and the "outfile" keyword for output modules.

```
1 import sys
2 from NekPy.FieldUtils import *
3
4 field = Field(sys.argv, output_points=10)
5
```

5.6 FieldConvert modules -m 109

```
6 InputModule.Create("xml", field, infile={"xml": "session.xml"}).Run()
7 InputModule.Create("fld", field, infile={"fld": "field.fld"}).Run()
8 ProcessModule.Create("wss", field, bnd="0").Run()
9 OutputModule.Create("fld", field, outfile="field_wss.fld").Run()
```

It can also emulate the functionality of FieldConvert when using the **nparts** option in the following way. Here **session_xml** is a directory containing the mesh partitioned into 2.

```
1 import sys
2 from NekPy.FieldUtils import *
3
4 field = Field(sys.argv, nparts=2, forceoutput=True, error=True)
5
6 inputxml = InputModule.Create("xml", field, "session_xml")
7 inputfld = InputModule.Create("fld", field, "field.fld")
8 processwss = ProcessModule.Create("wss", field, bnd="2")
9 outputfld = OutputModule.Create("fld", field, "field_wss.fld")
10
11 for part in range(2):
         field.NewPartition(sys.argv, part)
12
         inputxml.Run()
13
         inputfld.Run()
14
15
         processwss.Run()
         outputfld.Run()
16
17
18 OutputModule.Create("info", field, "field_wss_b0.fld", nparts=2).Run()
```

The number of partitions is looped over, with NewPartition called at the start of each. When using OutputModule, the info module must be used in order to obtain the correct result.

5.6 FieldConvert modules -m

FieldConvert allows the user to manipulate the *Nektar++* output binary files (.chk and .fld) by using the flag <u>m</u> (which stands for <u>m</u>odule). Specifically, FieldConvert has these additional functionalities

- 1. COProjection: Computes the C0 projection of a given output file;
- 2. CFL: Computes the CFL number over the solution domain for Incompressible flow
- 3. QCriterion: Computes the Q-Criterion for a given output file;
- 4. L2Criterion: Computes the Lambda 2 Criterion for a given output file;
- 5. addcompositeid: Adds the composite ID of an element as an additional field;
- 6. fieldfromstring: Modifies or adds a new field from an expression involving the existing fields;

- 7. (addFld): Sum two .fld files;
- 8. **combineAvg**: Combine two *Nektar++* binary output (.chk or .fld) field file containing averages of fields (and possibly also Reynolds stresses) into single file;
- concatenate: Concatenate a Nektar++ binary output (.chk or .fld) field file into single file (deprecated);
- 10. dof: Count the total number of DOF;
- 11. equispacedoutput: Write data as equi-spaced output using simplices to represent the data for connecting points;
- 12. extract : Extract a boundary field;
- 13. gradient : Computes gradient of fields;
- 14. halfmodetofourier: Convert (HalfMode) expansion to SingleMode for further processing;
- 15. homplane: Extract a plane from 3DH1D expansions;
- 16. (homstretch): Stretch a 3DH1D expansion by an integer factor;
- 17. **innerproduct**: take the inner product between one or a series of fields with another field (or series of fields).
- 18. **interpfield**: Interpolates one field to another, requires fromxml, fromfld to be defined;
- 19. **interpointdatatofld**: Interpolates given discrete data using a finite difference approximation to a fld file given an xml file;
- 20. **interpoints**: Interpolates a field to a set of points. Requires fromfid, fromxml to be defined, and a topts, line, plane or box of target points;
- 21. **interpptstopts**: Interpolates a set of points to another. Requires a topts, line, plane or box of target points;
- 22. **isocontour**: Extract an isocontour of "fieldid" variable and at value "fieldvalue". Optionally "fieldstr" can be specified for a string definition or "smooth" for smoothing;
- 23. jacobianenergy: Shows high frequency energy of Jacobian;
- 24. qualitymetric: Evaluate a quality metric of the underlying mesh to show mesh quality;
- 25. mean: Evaluate the mean of variables on the domain;
- 26. (meanmode): Extract mean mode (plane zero) of 3DH1D expansions;

- 27. pointdatatofld: Given discrete data at quadrature points project them onto an expansion basis and output fld file;
- 28. printfldnorms: Print L2 and LInf norms to stdout;
- 29. (removefield): Removes one or more fields from .fld files;
- 30. scalargrad: Computes scalar gradient field;
- 31. scaleinputfld: Rescale input field by a constant factor;
- 32. shear: Computes time-averaged shear stress metrics: TAWSS, OSI, transWSS, TAAFI, TACFI, WSSG;
- 33. streamfunction: Calculates stream function of a 2D incompressible flow.
- 34. **surfdistance**: Computes height of a prismatic boundary layer mesh and projects onto the surface (for e.g. y^+ calculation).
- 35. vorticity: Computes the vorticity field.
- 36. wss: Computes wall shear stress field.
- 37. [phifile: Computes the Φ function representing a body defined in an .stl file. Useful for the Smoothed Profile Method solver.
- 38. **wallNormalData**: Interpolate values for a set of points in the wall-normal direction (for e.g. extract boundary layer profiles).

The module list above can be seen by running the command

FieldConvert -1

In the following we will detail the usage of each module.

5.6.1 Smooth the data: C0Projection module

To smooth the data of a given .fld file one can use the COProjection module of Field-Convert

FieldConvert -m COProjection test.xml test.fld test-COProj.fld

where the file test-COProj.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

The option localtoglobalmap will do a global gather of the coefficients and then scatter them back to the local elements. This will replace the coefficients shared between two

elements with the coefficients of one of the elements (most likely the one with the highest id). Although not a formal projection it does not require any matrix inverse and so is very cheap to perform.

The option usexmlbcs will enforce the boundary conditions specified in the input xml file.

The option helmsmoothing=L will perform a Helmholtz smoothing projection of the form

$$\left(\nabla^2 - \left(\frac{2\pi}{L}\right)^2\right)\hat{u}^{new} = -\left(\frac{2\pi}{L}\right)^2\hat{u}^{orig}$$

which can be interpreted in a Fourier sense as smoothing the original coefficients using a low pass filter of the form

$$\hat{u}_{k}^{new} = \frac{1}{(1+k^{2}/K_{0}^{2})}\hat{u}_{k}^{orig}$$
 where $K_{0} = \frac{2\pi}{L}$

and so L is the length scale below which the coefficients values are halved or more. Since this form of the Helmholtz operator is not possitive definite, currently a direct solver is necessary and so this smoother is mainly of use in two-dimensions.

5.6.2 Calculate CFL number: CFL module

This module calculates the CFL number over the domain. It currently only supports the solution results from the Incompressible flow simulaitons, i.e. the outputs of **IncNavierStokesSolver**. To Estimate the *CFL* number, the user can run

FieldConvert -m CFL test.xml test.fld test-cfl.fld

where the file **test-cfl.fld** can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.3 Calculate Q-Criterion: QCriterion module

To perform the Q-criterion calculation and obtain an output data containing the Qcriterion solution, the user can run

FieldConvert -m QCriterion test.xml test.fld test-QCrit.fld

where the file test-QCrit.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.4 Calculate λ_2 : *L2Criterion* module

To perform the λ_2 vortex detection calculation and obtain an output data containing the values of the λ_2 eigenvalue, the user can run

FieldConvert -m L2Criterion test.xml test.fld test-L2Crit.fld

where the file test-L2Crit.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.5 Add composite ID: addcompositeid module

When dealing with a geometry that has many surfaces, we need to identify the composites to assign boundary conditions. To assist in this, FieldConvert has a addcompositeid module, which adds the composite ID of every element as a new field. To use this we simply run

FieldConvert -m addcompositeid mesh.xml out.dat

In this case, we have produced a Tecplot file which contains the mesh and a variable that contains the composite ID. To assist in boundary identification, the input file mesh.xml should be a surface XML file that can be obtained through the NekMesh extract module (see section 4.5.3).

5.6.6 Add new field: *fieldfromstring* module

To modify or create a new field using an expression involving the existing fields, one can use the fieldfromstring module of FieldConvert

```
FieldConvert -m fieldfromstring:fieldstr="x+y+u":fieldname="result" \
file1.xml file2.fld file3.fld
```

In this case **fieldstr** is a required parameter describing a function of the coordinates and the existing variables, and **fieldname** is an optional parameter defining the name of the new or modified field (the default is newfield). **file3.fld** is the output containing both the original and the new fields, and can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.7 Sum two .fld files: addFld module

To sum two .fld files one can use the (addFld) module of FieldConvert

```
FieldConvert -m addfld:fromfld=file1.fld:scale=-1 file1.xml file2.fld \
file3.fld
```

In this case we use it in conjunction with the command **scale** which multiply the values of a given .fld file by a constant **value**. **file1.fld** is the file multiplied by **value**, **file1.xml** is the associated session file, **file2.fld** is the .fld file which is summed to **file1.fld** and finally **file3.fld** is the output which contain the sum of the two .fld

files. **file3.fld** can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.8 Combine two .fld files containing time averages: combineAvg module

To combine two .fld files obtained through the AverageFields or ReynoldsStresses filters, use the combineAvg module of FieldConvert

```
FieldConvert -m combineAvg:fromfld=file1.fld file1.xml file2.fld \
file3.fld
```

file3.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.9 Concatenate two files: concatenate module

To concatenate **file1.fld** and **file2.fld** into **file-conc.fld** one can run the following command

FieldConvert file.xml file1.fld file2.fld file-conc.fld

where the file file-conc.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt. The concatenate module previously used for this purpose is not required anymore, and will be removed in a future release.

5.6.10 Count the number of DOF: dof module

To count the number of DOF in a solution file, one can run the following command

FieldConvert -m dof file.xml file.fld out.stdout

5.6.11 Equi-spaced output of data: equispaced output module

This module interpolates the output data to a truly equispaced set of points (not equispaced along the collapsed coordinate system). Therefore a tetrahedron is represented by a tetrahedral number of poinst. This produces much smaller output files. The points are then connected together by simplices (triangles and tetrahedrons).

FieldConvert -m equispacedoutput test.xml test.fld test.dat

or

FieldConvert -m equispacedouttput test.xml test.fld test.vtu

Note



Currently this option is only set up for triangles, quadrilaterals, hexahedrons, tetrahedrons and prisms.

5.6.12 Extract a boundary region: *extract* module

The boundary region of a domain can be extracted from the output data using the following command line

```
FieldConvert -m extract:bnd=2 test.xml \
    test.fld test-boundary.fld
```

The option bnd specifies which boundary region to extract. Note this is different to NekMesh where the parameter surf is specified and corresponds to composites rather boundaries. If bnd is not provided, all boundaries are extracted to different fields. The output will be placed in test-boundary_b2.fld. If more than one boundary region is specified the extension _b0.fld, _b1.fld etc will be outputted. To process this file you will need an xml file of the same region. This can be generated using the command:

NekMesh -m extract:surf=5 test.xml test_b0.xml

The surface to be extracted in this command is the composite number and so needs to correspond to the boundary region of interest. Finally to process the surface file one can use

FieldConvert test_b0.xml test_b0.fld test_b0.dat

This will obviously generate a Tecplot output if a .dat file is specified as last argument. A .vtu extension will produce a Paraview or VisIt output.

5.6.13 Compute the gradient of a field: gradient module

To compute the spatial gradients of all fields one can run the following command

FieldConvert -m gradient test.xml test.fld test-grad.fld

where the file file-grad.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.14 Convert HalfMode expansion to SingleMode for further processing: halfmodetofourier module

To obtain full Fourier expansion form a (HalfMode) result, use the comand:

FieldConvert -m halfmodetofourier:realmodetoimag=value1,value2 file.xml half_mode_file.fld single_mode_file.fld

In linear stability analysis with halfmode expansion, the variable w, whose variable number is 2, is a imaginary mode but it is output as a real mode in the eigenvector field file. By setting option <code>realmodetoimag=2</code>, w can be transformed to a imaginary mode correctly.

5.6.15 Extract a plane from 3DH1D expansion: homplane module

To obtain a 2D expansion containing one of the planes of a 3DH1D field file, use the command:

```
FieldConvert -m homplane:planeid=value file.xml file.fld file-plane.fld
```

If the option wavespace is used, the Fourier coefficients corresponding to planeid are obtained. The command in this case is:

FieldConvert -m homplane:wavespace:planeid=value file.xml \
 file.fld file-plane.fld

The output file file-plane.fld can be processed in a similar way as described in section 5.2 to visualise it either in Tecplot or in Paraview.

5.6.16 Stretch a 3DH1D expansion: homstretch module

To stretch a 3DH1D expansion in the z-direction, use the command:

FieldConvert -m homstretch:factor=value file.xml file.fld file-stretch.fld

The number of modes in the resulting field can be chosen using the command-line parameter --output-points-hom-z.

The output file <u>file-stretch.fld</u> can be processed in a similar way as described in section 5.2 to visualise it either in Tecplot or in Paraview.

5.6.17 Inner Product of a single or series of fields with respect to a single or series of fields: *innerproduct* module

You can take the inner product of one field with another field using the following command:

```
FieldConvert -m innerproduct:fromfld=file1.fld file2.xml file2.fld \
out.stdout
```

This command will load the file1.fld and file2.fld assuming they both are spatially defined by files.xml and determine the inner product of these fields. The input option fromfld must therefore be specified in this module.

Optional arguments for this module are **fields** which allow you to specify the fields that you wish to use for the inner product, i.e.

```
FieldConvert -m innerproduct:fromfld=file1.fld:fields="0,1,2" file2.xml \
file2.fld out.stdout
```

will only take the inner product between the variables 0,1 and 2 in the two fields files. The default is to take the inner product between all fields provided.

Additional options include multifldids and allfromflds which allow for a series of fields to be evaluated in the following manner:

```
FieldConvert -m innerproduct:fromfld=file1.fld:multifldids="0-3"\
file2.xml file2.fld out.stdout
```

will take the inner product between a file names field1_0.fld, field1_1.fld, field1_2.fld and field1_3.fld with respect to field2.fld.

Analogously including the options allfromflds, i.e.

```
FieldConvert -m innerproduct:fromfld=file1.fld:multifldids="0-3":\
allfromflds file2.xml file2.fld out.stdout
```

Will take the inner product of all the from fields, i.e. field1_0.field1_1.field1_2.fid and field1_3.fid with respect to each other. This option essentially ignores file2.fid. Only the unique inner products are evaluated so if four from fields are given only the related trianuglar number $4 \times 5/2 = 10$ of inner products are evaluated.

This option can be run in parallel.

5.6.18 Interpolate one field to another: *interpfield* module

To interpolate one field to another, one can use the following command:

```
FieldConvert -m interpfield:fromxml=file1.xml:fromfld=file1.fld \
    file2.xml file2.fld
```

This command will interpolate the field defined by file1.xml and file1.fld to the new mesh defined in file2.xml and output it to file2.fld. The fromxml and fromfld must be specified in this module. In addition there are two optional ar-

guments clamptolowervalue and clamptouppervalue which clamp the interpolation between these two values. Their default values are -10,000,000 and 10,000,000.

If the fromfld is a 3DH1D field and uses HalfMode expansion, you can use realmodetoimag= $n_1, n_2, ..., n_m$ to transform the $n_1, n_2, ..., n_m$ th variables from real modes to imaginary modes. Variable index starts from 0.



Tip

This module can run in parallel where the speed is increased not only due to using more cores but also, since the mesh is split into smaller sub-domains, the search method currently adopted performs faster.

5.6.19 Interpolate scattered point data to a field: *interppointdatatofid* module

To interpolate discrete point data to a field, use the interpoint data to fid module:

FieldConvert -m interppointdatatofld:frompts=file1.pts file1.xml file1.fld

or alternatively for csv data:

FieldConvert -m interppointdatatofld:frompts=file1.csv file1.xml file1.fld

This command will interpolate the data from file1.pts (file1.csv) to the mesh and expansions defined in file1.xml and output the field to file1.fld. The file file.pts must be of the form:

```
1 <?rml version="1.0" encoding="utf-8" ?>
2 <NEKTAR>
3 <POINTS DIM="1" FIELDS="a,b,c">
4     1.0000 -1.0000 1.0000 -0.7778
5     2.0000 -0.9798 0.9798 -0.7980
6     3.0000 -0.9596 0.9596 -0.8182
7     4.0000 -0.9394 0.9394 -0.8384
8 </POINTS>
9 </NEKTAR>
```

where DIM="1" FIELDS="a,b,c specifies that the field is one-dimensional and contains three variables, a, b, and c. Each line defines a point, while the first column contains its x-coordinate, the second one contains the a-values, the third the b-values and so on. In case of n-dimensional data, the n coordinates are specified in the first n columns accordingly. An equivalent csv file is:

x, a, b, c 1.0000,-1.0000,1.0000,-0.7778 2.0000,-0.9798,0.9798,-0.7980 3.0000,-0.9596,0.9596,-0.8182 4.0000,-0.9394,0.9394,-0.8384

In order to interpolate 1D data to a nD field, specify the matching coordinate in the output field using the interproord argument:

This will interpolate the 1D scattered point data from 1D-file1.pts to the y-coordinate of the 3D mesh defined in 3D-file1.xml. The resulting field will have constant values along the x and z coordinates. For 1D Interpolation, the module implements a quadratic scheme and automatically falls back to a linear method if only two data points are given. A modified inverse distance method is used for 2D and 3D interpolation. Linear and quadratic interpolation require the data points in the .pts-file to be sorted by their location in ascending order. The Inverse Distance implementation has no such requirement.

5.6.20 Interpolate a field to a series of points: interpoints module

You can interpolate one field to a series of given points using the following command:

```
FieldConvert -m interppoints:fromxml=file1.xml:fromfld=\
    file1.fld:topts=file2.pts file2.dat
```

This command will interpolate the field defined by file1.xml and file1.fld to the points defined in file2.pts and output it to file2.dat. The fromxml and fromfld must be specified in this module. The format of the file file2.pts is of the same form as for the *interpointdatatofld* module:

```
1 <?rml version="1.0" encoding="utf-8" ?>
2 <NEKTAR>
3 <POINTS DIM="2" FIELDS="">
4           0.0 0.0
5           0.5 0.0
6           1.0 0.0
7           </POINTS>
8 </NEKTAR>
```

Similar to the *interppointdatatofid* module, the .pts file can be interchanged with a .csv file (the output can also be written to .csv):

x, y 0.0,0.0 0.5,0.0

1.0,0.0

There are three optional arguments clamptolowervalue, clamptouppervalue and defaultvalue the first two clamp the interpolation between these two values and the third defines the default value to be used if the point is outside the domain. Their default values are -10,000,000, 10,000,000 and 0.

In addition, instead of specifying the file file2.pts, a module list of the form

```
FieldConvert -m interppoints:fromxml=file1.xml:fromfld= \
    file1.fld:line=npts,x0,y0,x1,y1 file2.dat
```

can be specified where **npts** is the number of equispaced points between (x_0, y_0) to (x_1, y_1) . This also works in 3D, by specifying (x_0, y_0, z_0) to (x_1, y_1, z_1) .

An extraction of a plane of points can also be specified by

```
FieldConvert -m interppoints:fromxml=file1.xml:fromfld=file1.fld:\
    plane=npts1,npts2,x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3 file2.dat
```

where **npts1,npts2** is the number of equispaced points in each direction and (x_0, y_0, z_0) , (x_1, y_1, z_1) , (x_2, y_2, z_2) and (x_3, y_3, z_3) define the plane of points specified in a clockwise or anticlockwise direction.

In addition, an extraction of a box of points can also be specified by

```
FieldConvert -m interppoints:fromxml=file1.xml:fromfld=file1.fld:\
    box=npts1,npts2,npts3,xmin,xmax,ymin,ymax,zmin,zmax file2.dat
```

where npts1,npts2,npts3 is the number of equispaced points in each direction and $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$ define the limits of the box of points.

There is also an additional optional argument cp=p0,q which adds to the interpolated fields the value of $c_p = (p - p0)/q$ and $c_{p0} = (p - p0 + 0.5u^2)/q$ where p0 is a reference pressure and q is the free stream dynamics pressure. If the input does not contain a field "p" or a velocity field "u,v,w" then cp and cp0 are not evaluated accordingly.

If the fromfld is a 3DH1D field and uses HalfMode expansion, you can use $realmodetoimag=n_1, n_2, ..., n_m$ to transform the $n_1, n_2, ..., n_m$ th variables from real modes to imaginary modes. Variable index starts from 0.

Note This module runs in parallel for the line, plane and box extraction of points.

5.6.21 Interpolate a set of points to another: *interpptstopts* module

You can interpolate one set of points to another using the following command:

```
FieldConvert file1.pts -m interpptstopts:topts=file2.pts file2.dat
```

This command will interpolate the data in <u>file1.pts</u> to a new set of points defined in <u>file2.pts</u> and output it to <u>file2.dat</u>.

Similarly to the *interppoints* module, the target point distribution can also be specified using the line, plane or box options. The optional arguments clamptolowervalue, clamptouppervalue, defaultvalue and cp are also supported with the same meaning as in *interppoints*.

One useful application for this module is with 3DH1D expansions, for which currently the *interpoints* module does not work. In this case, we can use for example

```
FieldConvert file1.xml file1.fld -m interpptstopts:\
    plane=npts1,npts2,x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3 \
    file2.dat
```

With this usage, the *equispacedoutput* module will be automatically called to interpolate the field to a set of equispaced points in each element. The result is then interpolated to a plane by the *interpolatopts* module.

Note This module does not work in parallel.

5.6.22 Isocontour extraction: *iscontour* module

Extract an isocontour from a field file. This option automatically take the field to an equispaced distribution of points connected by linear simplicies of triangles or tetrahedrons. The linear simplices are then inspected to extract the isocontour of interest. To specify the field **fieldid** can be provided giving the id of the field of interest and **fieldvalue** provides the value of the isocontour to be extracted.

FieldConvert -m isocontour:fieldid=2:fieldvalue=0.5 test.xml test.fld \

test-isocontour.dat

Alternatively fieldstr="u+v" can be specified to calculate the field u + v and extract its isocontour. You can also specify fieldname="UplusV" to define the name of the isocontour in the .dat file, i.e.

```
FieldConvert -m isocontour:fieldstr="u+v":fieldvalue=0.5:\
    fieldname="UplusV" test.xml test.fld test-isocontour.dat
```

Optionally smooth can be specified to smooth the isocontour with default values smoothnegdiffusion =0.495, smoothnegdiffusion =0.5 and smoothiter =100. This option typically should be used with the globalcondense option which removes multiply defined verties from the simplex definition which arise as isocontour are generated element by element. The smooth option preivously automatically called the globalcondense option but this has been depracated since it is now possible to read isocontour files directly and so it is useful to have these as separate options.

In addition to the smooth or globalcondense options you can specify removesmallcontour =100 which will remove separate isocontours of less than 100 triangles.



Currently this option is only set up for triangles, quadrilaterals, tetrahedrons and prisms.

5.6.23 Show high frequency energy of the Jacobian: *jacobianenergy* module

FieldConvert -m jacobianenergy file.xml file.fld jacenergy.fld

The option topmodes can be used to specify the number of top modes to keep.

The output file jacenergy.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.24 Calculate mesh quality: qualitymetric module

The qualitymetric module assesses the quality of the mesh by calculating a per-element quality metric and adding an additional field to any resulting output. This does not require any field input, therefore an example usage looks like

```
FieldConvert -m qualitymetric mesh.xml mesh-with-quality.dat
```

Two quality metrics are implemented that produce scalar fields Q:

- By default a metric outlined in [14] is produced, where all straight sided elements have quality Q = 1 and Q < 1 shows the deformation between the curved element and the straight-sided element. If Q = 0 then the element is invalid. Note that Q varies over the volume of the element but is not guaranteed to be continuous between elements.
- Alternatively, if the scaled option is passed through to the module, then the scaled Jacobian

$$J_s = \frac{\min_{\xi \in \Omega_{\rm st}} J(\xi)}{\max_{\xi \in \Omega_{\rm st}} J(\xi)}$$

(i.e. the ratio of the minimum to maximum Jacobian of each element) is calculated. Again Q = 1 denotes an ideal element, but now invalid elements are shown by Q < 0. Any elements with Q near zero are determined to be low quality.

5.6.25 Evaluate the mean of variables on the domain: mean module

To evaluate the mean of variables on the domain one can use the mean module of FieldConvert

FieldConvert -m mean file1.xml file2.fld out.stdout

This module does not create an output file which is reinforced by the out.stdout option. The integral and mean for each field variable are then printed to the stdout.

5.6.26 Extract mean mode of 3DH1D expansion: meanmode module

To obtain a 2D expansion containing the mean mode (plane zero in Fourier space) of a 3DH1D field file, use the command:

FieldConvert -m meanmode file.xml file.fld file-mean.fld

The output file <u>file-mean.fld</u> can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot or in Paraview or VisIt.

5.6.27 Project point data to a field: *pointdatatofld* module

To project a series of points given at the same quadrature distribution as the .xml file and write out a .fld file use the pointdatatofld module:

```
FieldConvert -m pointdatatofld:frompts=file.pts file.xml file.fld
```

This command will read in the points provided in the file.pts and assume these are given at the same quadrature distribution as the mesh and expansions defined in file.xml and output the field to file.fld. If the points do not match an error will be dumped.

The file **file.pts** which is assumed to be given by an interpolation from another source is of the form:

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <NEKTAR>
3 <POINTS DIM="3" FIELDS="p">
  1.70415 -0.4
                  -0 0182028
                                     -0.106893
4
  1.70415 -0.395683 -0.0182028
                                     -0.106794
5
  1.70415 -0.3875 -0.0182028
                                     -0.106698
6
  1.70415 -0.379317 -0.0182028
                                     -0.103815
7
8
  </POINTS>
9 </NEKTAR>
```

where DIM="3" FIELDS="p specifies that the field is three-dimensional and contains one variable, p. Each line defines a point, the first, second, and third columns contains the x, y, z-coordinate and subsequent columns contain the field values, in this case the p-value So in the general case of n-dimensional data, the n coordinates are specified in the first n columns accordingly followed by the field data. Alternatively, the file.pts can be interchanged with a csv file.

The default argument is to use the equispaced (but potentially collapsed) coordinates which can be obtained from the command.

FieldConvert file.xml file.dat

In this case the pointdatatofid module should be used without the <u>-noequispaced</u> option. However this can lead to problems when peforming an elemental forward projection/transform since the mass matrix in a deformed element can be singular as the equispaced points do not have a sufficiently accurate quadrature rule that spans the polynomial space. Therefore it is advisable to use the set of points given by

FieldConvert -- noequispaced file.xml file.dat

which produces a set of points at the gaussian collapsed coordinates.

Finally the option **setnantovalue=0** can also be used which sets any nan values in the interpolation to zero or any specified value in this option.

5.6.28 Print L2 and LInf norms: *printfldnorms* module

FieldConvert -m printfldnorms test.xml test.fld out.stdout

This module does not create an output file which is reinforced by the out.stdout option. The L2 and LInf norms for each field variable are then printed to the stdout.

5.6.29 Removes one or more fields from .fld files: removefield module

This module allows to remove one or more fields from a .fld file:

```
FieldConvert -m removefield:fieldname="u,v,p" test.xml test.fld test-
removed.fld
```

where the file test-removed.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt. The lighter resulting file speeds up the postprocessing of large files when not all fields are required.

5.6.30 Computes the scalar gradient: *scalargrad* module

The scalar gradient of a field is computed by running:

FieldConvert -m scalargrad:bnd=0 test.xml test.fld test-scalgrad.fld

The option bnd specifies which boundary region to extract. Note this is different to NekMesh where the parameter surf is specified and corresponds to composites rather boundaries. If bnd is not provided, all boundaries are extracted to different fields. To process this file you will need an xml file of the same region.

5.6.31 Scale a given .fld: *scaleinputfld* module

To scale a .fld file by a given scalar quantity, the user can run:

```
FieldConvert -m scaleinputfld:scale=value test.xml test.fld test-scal.fld
```

The argument scale=value rescales of a factor value test.fld by the factor value. The output file file-scal.fld can be processed in a similar way as described in section 5.2 to visualise the result either in Tecplot, Paraview or VisIt.

5.6.32 Time-averaged shear stress metrics: shear module

Time-dependent wall shear stress derived metrics relevant to cardiovascular fluid dynamics research can be computed using this module. They are

- TAWSS: time-averaged wall shear stress;
- OSI: oscillatory shear index;
- transWSS: transverse wall shear stress;
- TACFI: time-averaged cross-flow index;
- TAAFI: time-averaged aneurysm formation index;

• |WSSG|: wall shear stress gradient.

To compute these, the user can run:

```
FieldConvert -m shear:N=value:fromfld=test_id_b0.fld \
    test.xml test-multishear.fld
```

The argument **N** and **fromfld** are compulsory arguments that respectively define the number of **fld** files corresponding to the number of discrete equispaced time-steps, and the first **fld** file which should have the form of **test_id_b0.fld** where the first underscore in the name marks the starting time-step file ID.

The input .fld files are the outputs of the *wss* module. If they do not contain the surface normals (an optional output of the *wss* modle), then the *shear* module will not compute the last metric, [WSSG].

5.6.33 Stream function of a 2D incompressible flow: streamfunction module

The streamfunction module calculates the stream function of a 2D incompressible flow, by solving the Poisson equation

$$\nabla^2 \psi = -\omega$$

where ω is the vorticity. Note that this module applies the same boundary conditions specified for the y-direction velocity component $\overline{\mathbf{v}}$ to the stream function, what may not be the most appropriate choice.

To use this module, the user can run

FieldConvert -m streamfunction test.xml test.fld test-streamfunc.fld

where the file (test-streamfunc.fld) can be processed in a similar way as described in section 5.2.

5.6.34 Boundary layer height calculation: surfdistance module

The surface distance module computes the height of a boundary layer formed by quadrilaterals (in 2D) or prisms and hexahedrons (in 3D) and projects this value onto the surface of the boundary, in a similar fashion to the extract module. In conjunction with a mesh of the surface, which can be obtained with NekMesh, and a value of the average wall shear stress, one potential application of this module is to determine the distribution of y^+ grid spacings for turbulence calculations.

To compute the height of the prismatic layer connected to boundary region 3, the user can issue the command:

FieldConvert -m surfdistance:bnd=3 input.xml output.fld

Note that no .fld file is required, since the mesh is the only input required in order to calculate the element height. This produces a file output_b3.fld, which can be visualised with the appropriate surface mesh from NekMesh.

5.6.35 Calculate vorticity: vorticity module

To perform the vorticity calculation and obtain an output data containing the vorticity solution, the user can run

FieldConvert -m vorticity test.xml test.fld test-vort.fld

where the file test-vort.fld can be processed in a similar way as described in section 5.2.

5.6.36 Computing the wall shear stress: wss module

To obtain the wall shear stres vector and magnitude, the user can run:

FieldConvert -m wss:bnd=0:addnormals=1 test.xml test.fld test-wss.fld

The option bnd specifies which boundary region to extract. Note this is different to NekMesh where the parameter surf is specified and corresponds to composites rather boundaries. If bnd is not provided, all boundaries are extracted to different fields. The addnormals is an optional command argument which, when turned on, outputs the normal vector of the extracted boundary region as well as the shear stress vector and magnitude. This option is off by default. To process the output file(s) you will need an xml file of the same region.

The output fields are wall shear stress in streamwise direction (Shear_s), normal direction (Shear_n) and magnitude of the wall shear stress vector (Shear_mag) for 2D simulations. For 3D simulations, wall shear stress in the spanwise direction (Shear_z) is also included in the output fileds. Note that the streamwise and normal directions are local at each point on the corresponding boundary.

5.6.37 Calculating the shape function Φ for an SPM case: *phifile* module

Note

This module is in experimental phase and only runs in serial. When reading 3D geometries from .stl files, errors may occur if triangles are placed exactly at 90°.

This FieldConvert module converts a binary .stl CAD file into .fld or .vtu/.dat files that can be used as inputs for the Smoothed Profile Method solver. Running the command:

FieldConvert -m phifile:file=geom.stl:scale=value session.xml geom.fld

will generate an output file geom.fld with all the information required to define a shape function Φ representing the geometry specified in geom.stl. The option scale sets the value of ξ as it is described in the Synopsis chapter of the Incompressible N-S solver. If the output file gets the extension .vtu or .dat, the module will produce a graphical representation of the shape function; however, the recommended way to proceed is to generate an .fld file and then, use FieldConvert to obtain the .vtu or .dat files if needed for visualisation purposes:

FieldConvert -m phifile:file=geom.stl:scale=value session.xml geom.fld FieldConvert session.xml geom.fld geom.vtu

This module can also be used to produce a .fld and .vtu / .dat file of shapes defined directly in the session file through an analytical expression. In this case, the commands simplify to:

FieldConvert -m phifile session.xml geom.fld FieldConvert session.xml geom.fld geom.vtu

The algorithm computes an octree for the triangles that define the 3D object in the .stl file, and then loops over all the nodes of the computational mesh looking for the shortest distance to the 3D object. Since this module is currently in experimental phase, it only runs in serial and therefore its performance in computing the shape function from the .stl file is limited, especially in 3D cases. In addition to this, it is recommended to make sure that the angles between triangles are not strictly equal to 90° , since the algorithm will probably fail to find the real Φ function.

5.6.38 Interpolate values for a point array: wallNormalData module

To obtain the values of an array of points in the wall-normal direction, such as boundary layer profiles, the user can run:

```
FieldConvert -m wss:bnd=0:xorig="0.5,0,0":projDir="0,1,0":maxDist=0.1:\
    distH=0.1:nptsH=0.01:d=0.2 test.xml test.fld test.pts
```

The option bnd specifies the target boundary region, to which the input point specified by xorig will be projected as the first point in the interpolation points array. The projection direction is specified by projDir, which does not have to be a unit vector but will be automatically normalized. The input point and projetion direction must be set carefully to make sure a projected point can be found on the target boundary. In cases with curved boundaries on which multiple projected points might exist, the user can use maxDist to limit the maximum projection distance between the input point to the projected point.

After the projected point is found, this module will compute the wall-normal direction at the point, and an array of points will be set in this direction. The number of points is specified by <code>nptsH</code>, and the parameter d controls the distribution of the points as follows

$$h(\xi) = H\left\{1 - \frac{\tanh\left[(1-\xi)atanh(\sqrt{1-\delta})\right]}{\sqrt{1-\delta}}\right\}$$

where ξ is equally-spaced parametric parameter varying from 0 to 1; H is the distance between the first point (on the booundary) and the last point, which is specified distH; δ is given by d. This distribution is employed for $0 < \delta \leq 0.95$ while $\delta > 0.95$ for evenly-spaced distribution.

Finally, this module will interpolate the values at the points from test.fld, and save the result in test.pts.



Figure 5.2 (a) Project the input point onto the boundary, (b) Array of points to be interpolated.

5.6.39 Manipulating meshes with FieldConvert

FieldConvert has support for two modules that can be used in conjunction with the linear elastic solver, as shown in chapter 12. To do this, FieldConvert has an XML output module, in addition to the Tecplot and VTK formats.

The **deform** module, which takes no options, takes a displacement field and applies it to the geometry, producing a deformed mesh:

FieldConvert -m deform input.xml input.fld deformed.xml

The displacement module is designed to create a boundary condition field file. Its intended use is for mesh generation purposes. It can be used to calculate the displacement between the linear mesh and a high-order surface, and then produce a fld file, prescribing the displacement at the boundary, that can be used in the linear elasticity solver.

Presently the process is somewhat convoluted and must be used in conjunction with NekMesh to create the surface file. However the bash input below describes the procedure. Assume the high-order mesh is in a file called mesh.xml, the linear mesh is mesh-linear.xml that can be generated by removing the CURVED section from mesh.xml, and that we are interested in the surface with ID 123.

```
# Extract high order surface
NekMesh -m extract:surf=123 mesh.xml mesh-surf-curved.xml
# Use FieldConvert to calculate displacement between two surfaces
FieldConvert -m displacement:id=123:to=mesh-surf-curved.xml \
    mesh-linear.xml mesh-deformation.fld
# mesh-deformation.fld is used as a boundary condition inside the
# solver to prescribe the deformation conditions.xml contains
# appropriate Nektar++ parameters (mu, E, other BCs, ...)
LinearElasticSolver mesh-linear.xml conditions.xml
# This produces the final field mesh-linear.fld which is the
# displacement field, use FieldConvert to apply it:
FieldConvert-g -m deform mesh-linear.xml mesh-linear.fld mesh-deformed.xml
```

5.7 FieldConvert in parallel

To run FieldConvert in parallel the user needs to compile Nektar++ with MPI support and can employ the following command

mpirun -np <nprocs> FieldConvert test.xml test.fld test.dat

mpirun -np <nprocs> FieldConvert test.xml test.fld test.plt

or

mpirun -np <nprocs> FieldConvert test.xml test.fld test.vtu

replacing <nprocs> with the number of processors. For the .dat and .plt outputs the current version will proudce a single output file. However it is also sometimes useful to produce multiple output files, one for each partition, and this can be done by using the writemultiplefiles option, i.e.

For the .vtu format multiple files will by default be produced of the form test_vtu/P0000000.vtu, test_vtu/P0000002.vtu. For this format an additional file called test.pvtu is written out which allows for parallel reading of the individual .vtu files.

FieldConvert functions that produce a .fld file output will also be created when running in parallel. In this case when producing a .fld file a directory called test.fld (or the specified output name) is created with the standard parallel field files placed within the directory.

5.8 Processing large files in serial

When processing large files, it is not always convenient to run in parallel but process each parallel partition in serial, for example when interpolating a solution field from one mesh to another or creating an output file for visualization.

5.8.1 Using the part-only and part-only-overlapping options

Loading full file1.xml can be expensive if the file1.xml is already large. So instead you can pre-partition the file using the using the **_part-only** option. So the command

FieldConvert --part-only 10 file.xml file.fld

will partition the mesh into 10 partitions and write each partition into a directory called file_xml. If you enter this directory you will find partitioned XML files P000000.xml, P0000001.xml, ..., P0000009.xml which can then be processed individually as outlined above.

There is also a *-part-only-overlapping* option, which can be run in the same fashion.

```
FieldConvert --part-only-overlapping 10 file.xml file.fld
```

In this mode, the mesh is partitioned into 10 partitions in a similar manner, but the elements at the partition edges will now overlap, so that the intersection of each partition

with its neighbours is non-empty. This is sometime helpful when, for example, producing a global isocontour which has been smoothed. Applying the smoothed isocontour extraction routine with the <code>-part-only</code> option will produce a series of isocontour where there will be a gap between partitions, as the smoother tends to shrink the isocontour within a partition. using the <code>-part-only-overlapping</code> option will still yield a shrinking isocontour, but the overlapping partitions help to overlap the partition boundaries.

5.8.2 Using the *nparts* options

If you have a partitioned directory either from a parallel run or using the <code>-part-only</code> option you can now run the <code>FieldConvert</code> option using the <code>nparts</code> command line option, that is

FieldConvert --nparts 10 file1_xml:xml file1.fld file1.vtu

Note the form file1_xml:xml option tells the code it is a parallel partition which should be treated as an xml type file. the argument of nparts should correpsond to the number of partitions used in generating the file1_xml directory. This will create a parallel vtu file as it processes each partition.

Another example is to interpolate file1.fld from one mesh file1.xml to another file2.xml. If the mesh files are large we can do this by partitioning file2.xml into 10 (or more) partitions to generate the file_xml directory and interpolating each partition one by one using the command:

Note that internally the routine uses the range option so that it only has to load the part of file1.xml that overlaps with each partition of file2.xml. The resulting output will lie in a directory called file2.fld, with each of the different parallel partitions in files with names P0000000.fld, P000001.fld, ..., P0000009.fld. In previous versions of FieldConvert it was necessary to generate an updated Info.xml file but in the current version it should automatically be updating this file.

5.8.3 Running in parallel with the *nparts* option

The examples above will process each partition serially which may now take a while for many partitions. You can however run this option in parallel using a smaller number of cores than the nparts.

For the example of creating a vtu file above you can use 4 processor concurrently wiht the command line:
mpirun -n 4 FieldConvert --nparts 10 file1_xml:xml file1.fld file1.vtu

Obviously the executable will have to have been compiled with the MPI option for this to work.

Part III Solver Applications

CHAPTER **6**

Acoustic Solver

6.1 Synopsis

The aim of the AcousticSolver is to predict acoustic wave propagation. Through the application of a splitting technique, the flow-induced acoustic field is totally decoupled from the underlying hydrodynamic field.

6.1.1 Linearized Euler Equations

The Linearized Euler Equations (LEE) are obtained by linearizing the Euler Equations about a mean flow state $(\bar{\rho}, \bar{c^2}, \bar{u})$. Hence, they describe the evolution of perturbations $(p^{a}, \rho^{a}, \bar{\rho}u^{a})$ around this state. In conservative form, the LEE are given as:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}_1}{\partial x_1} + \frac{\partial \boldsymbol{F}_2}{\partial x_2} + \frac{\partial \boldsymbol{F}_3}{\partial x_3} + \boldsymbol{C}\boldsymbol{U} = \boldsymbol{W}$$
(6.1)

with

$$\boldsymbol{U} = \begin{bmatrix} p^{a} \\ \rho^{a} \\ \overline{p} u_{1}^{a} \\ \overline{p} u_{2}^{a} \\ \overline{p} u_{2}^{a} \\ \overline{p} u_{3}^{a} \\ \overline{p} u_{1}^{a} + \overline{u}_{1} \rho^{a} \\ \overline{p} u_{1}^{a} + \overline{u}_{1} \rho^{a} \\ \overline{p} u_{1}^{a} \overline{u}_{1} + p^{a} \\ \overline{p} u_{2}^{a} \overline{u}_{1} \\ \overline{p} u_{3}^{a} \overline{u}_{1} \end{bmatrix}, \quad \boldsymbol{F}_{2} = \begin{bmatrix} \overline{p} u_{2}^{a} \overline{c^{2}} + \overline{u}_{2} p^{a} \\ \overline{p} u_{2}^{a} \overline{u}_{2} \rho^{a} \\ \overline{p} u_{3}^{a} \overline{u}_{2} \\ \overline{p} u_{3}^{a} \overline{u}_{2} \end{bmatrix}, \quad \boldsymbol{F}_{3} = \begin{bmatrix} \overline{p} u_{3}^{a} \overline{c^{2}} + \overline{u}_{3} p^{a} \\ \overline{p} u_{3}^{a} \overline{u}_{3} \rho^{a} \\ \overline{p} u_{3}^{a} \overline{u}_{3} \\ \overline{p} u_{2}^{a} \overline{u}_{3} + p^{a} \end{bmatrix}, \quad (6.3)$$

$$\boldsymbol{C} = \begin{bmatrix} (\gamma - 1) \frac{\partial \overline{u}_{k}}{\partial x_{k}} & 0 & \frac{1}{p} (1 - \gamma) \frac{\partial \overline{p}}{\partial x_{1}} & \frac{1}{p} (1 - \gamma) \frac{\partial \overline{p}}{\partial x_{2}} & \frac{1}{p} (1 - \gamma) \frac{\partial \overline{p}}{\partial x_{3}} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \overline{u}_{k} \frac{\partial \overline{u}_{1}}{\partial x_{k}} & \frac{\partial \overline{u}_{1}}{\partial x_{1}} & \frac{\partial \overline{u}_{1}}{\partial x_{2}} & \frac{\partial \overline{u}_{1}}{\partial x_{2}} \\ 0 & \overline{u}_{k} \frac{\partial \overline{u}_{2}}{\partial x_{k}} & \frac{\partial \overline{u}_{2}}{\partial x_{1}} & \frac{\partial \overline{u}_{2}}{\partial x_{2}} & \frac{\partial \overline{u}_{2}}{\partial x_{3}} \\ 0 & \overline{u}_{k} \frac{\partial \overline{u}_{3}}{\partial x_{k}} & \frac{\partial \overline{u}_{3}}{\partial x_{1}} & \frac{\partial \overline{u}_{3}}{\partial x_{2}} & \frac{\partial \overline{u}_{3}}{\partial x_{2}} & \frac{\partial \overline{u}_{3}}{\partial x_{3}} \\ \end{bmatrix} . \quad (6.4)$$

By default, the source term vector \boldsymbol{W} is zero and has to be specified by an appropriate forcing.

6.1.2 Acoustic Perturbation Equations

The acoustic perturbation equations (APE-1/APE-4) proposed by Ewert and Schroeder [12] assure stable aeroacoustic simulations. These equations are similar to the LEE, but account for acoustic perturbations exclusively. The AcousticSolver implements the APE-1/4 type operator:

$$\frac{\partial p^{\mathbf{a}}}{\partial t} + \overline{c^2} \nabla \cdot \left(\overline{\rho} \boldsymbol{u}^{\mathbf{a}} + \overline{\boldsymbol{u}} \frac{p^{\mathbf{a}}}{\overline{c^2}} \right) = \dot{\omega}_{\mathbf{c}}$$
(6.5a)

$$\frac{\partial \boldsymbol{u}^{\mathrm{a}}}{\partial t} + \nabla \left(\overline{\boldsymbol{u}} \cdot \boldsymbol{u}^{\mathrm{a}} \right) + \nabla \left(\frac{p^{\mathrm{a}}}{\overline{\rho}} \right) = \dot{\boldsymbol{\omega}}_{\mathrm{m}} , \qquad (6.5\mathrm{b})$$

where $(\bar{\boldsymbol{u}}, \bar{c^2}, \bar{\rho})$ represents the base flow and $(\boldsymbol{u}^{\rm a}, p^{\rm a})$ the acoustic perturbations. Similar to the LEE, the acoustic source terms $\dot{\omega}_{\rm c}$ and $\dot{\boldsymbol{\omega}}_{\rm m}$ are by default zero and must be specified e.g. by an appropriate forcing. This way, e.g. the APE-1, APE-4 [12] or revised APE equations [15] can be obtained. Expressed as hyperbolic conservation law, the APE-1/4 operator reads:

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}_1}{\partial x_1} + \frac{\partial \boldsymbol{F}_2}{\partial x_2} + \frac{\partial \boldsymbol{F}_3}{\partial x_3} = \boldsymbol{W}$$
(6.6)

with

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{p}^{\mathrm{a}} \\ \boldsymbol{u}_{1}^{\mathrm{a}} \\ \boldsymbol{u}_{2}^{\mathrm{a}} \\ \boldsymbol{u}_{3}^{\mathrm{a}} \end{bmatrix}, \qquad (6.7)$$

$$\boldsymbol{F_1} = \begin{bmatrix} \overline{\rho}\overline{c^2}u_1^{\mathrm{a}} + p^{\mathrm{a}}\overline{u}_1 \\ \overline{u}_ju_j^{\mathrm{a}} + p^{\mathrm{a}}/\overline{\rho} \\ 0 \\ 0 \end{bmatrix}, \ \boldsymbol{F_2} = \begin{bmatrix} \overline{\rho}\overline{c^2}u_2^{\mathrm{a}} + p^{\mathrm{a}}\overline{u}_2 \\ 0 \\ \overline{u}_ju_j^{\mathrm{a}} + p^{\mathrm{a}}/\overline{\rho} \\ 0 \end{bmatrix}, \ \boldsymbol{F_3} = \begin{bmatrix} \overline{\rho}\overline{c^2}u_3^{\mathrm{a}} + p^{\mathrm{a}}\overline{u}_3 \\ 0 \\ 0 \\ \overline{u}_ju_j^{\mathrm{a}} + p^{\mathrm{a}}/\overline{\rho} \end{bmatrix}.$$
(6.8)

6.2 Usage

AcousticSolver session.xml

6.3 Session file configuration

Parameters

Under this section it is possible to set the parameters of the simulation.

```
1 <PARAMETERS>
  <P>> TimeStep
                     = 1e-05 /P>
2
3
   <P> NumSteps
                     = 1000
                              /P>
  <P> FinTime
                     = 0.01
                              /P>
4
  <P> I0_CheckSteps = 100
                              /P>
5
  <P> I0_InfoSteps = 10
                              /P>
6
  I0_CFLSteps
                     = 10
                              /P>
7
8 </PARAMETERS>
```

- TimeStep is the time-step we want to use;
- FinTime is the final physical time at which we want our simulation to stop;
- NumSteps is the equivalent of FinTime but instead of specifying the physical final time we specify the number of time-steps;
- IO_CheckSteps sets the number of steps between successive checkpoint files;
- **IO_InfoSteps** sets the number of steps between successive info stats are printed to screen;
- **IO_CFLSteps** sets the number of steps between successive Courant number stats are printed to screen;

6.3.1 Time Integration Scheme

```
1 <TIMEINTEGRATIONSCHEME>
```

- 2 <METHOD> RungeKutta </METHOD>
- 3 <VARIANT> SSP </VARIANT>
- 4 <ORDER> 3 </ORDER>
- 5 </TIMEINTEGRATIONSCHEME>
- Method is the time-integration method. Note that only an explicit discretisation is supported.
- Order is the order of the time-integration method.
- Variant is the variant of the time-integration method (variables for Runga Kutta: Blank, SSP).

6.3.2 Solver Info

```
      1 <SOLVERINFO>

      2 <I PROPERTY="EQType"</td>
      VALUE="APE"
      />

      3 <I PROPERTY="Projection"</td>
      VALUE="DisContinuous"
      />

      4 <I PROPERTY="UpwindType"</td>
      VALUE="LaxFriedrichs"
      />

      5 </SOLVERINFO>
```

- EQType is the tag which specify the equations we want solve:
 - (APE) Acoustic Perturbation Equations (variables: (p,u,v,w));
 - (LEE) Linearized Euler Equations (variables: p,rho,rhou,rhov,rhow).
- **Projection** is the type of projection we want to use. Currently, only **DisContinuous** is supported.
- AdvectionType is the advection operator. Currently, only WeakDG (classical DG in weak form) is supported.
- UpwindType is the numerical interface flux (i.e. Riemann solver) we want to use for the advection operator (see [24] for the implemented formulations):

```
- Upwind;
```

- LaxFriedrichs;

6.3.3 Variables

For the APE operator, the acoustic pressure and velocity perturbations are solved, e.g.:

```
1 <VARIABLES>

2 <V ID="0"> p </V>

3 <V ID="1"> u </V>

4 <V ID="2"> v </V>

5 <V ID="3"> w </V>

6 </VARIABLES>
```

The LEE use a conservative formulation and introduce the additional density perturbation:

```
1 <VARIABLES>

2 <V ID="0"> p </V>

3 <V ID="1"> rho </V>

4 <V ID="2"> rhou </V>

5 <V ID="3"> rhov </V>

6 <V ID="4"> rhow </V>

7 </VARIABLES>
```

6.3.4 Functions

- BaseFlow Baseflow $(\overline{\rho}, \overline{c^2}, \overline{u})$ defined by the variables rho0, c0sq, u0, v0, w0 for APE and $(\overline{\rho}, \overline{c^2}, \overline{u}, \gamma)$ defined by rho0, c0sq, u0, v0, w0, gamma for LEE.
- InitialConditions

6.3.5 Boundary Conditions

In addition to plain Dirichlet and Neumann boundary conditions, the AcousticSolver features a slip-wall boundary condition, a non-reflecting boundary and a white noise boundary condition.

• Rigid (Slip-) Wall Boundary Condition, e.g. for APE:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3   <D VAR="p" USERDEFINEDTYPE="Wall" VALUE="0" />
4   <D VAR="u" USERDEFINEDTYPE="Wall" VALUE="0" />
5   <D VAR="v" USERDEFINEDTYPE="Wall" VALUE="0" />
6   <D VAR="w" USERDEFINEDTYPE="Wall" VALUE="0" />
7 </REGION>
8 </BOUNDARYCONDITIONS>
```

This BC imposes zero wall-normal perturbation velocity in a way that is more robust than using a Dirichlet boundary condition directly.

• Non-Reflecting Boundary Condition, e.g. for APE:

The Riemann-Invariant BC approximates a non-reflecting (r.g. Farfield) boundary condition by setting incoming invariants to zero.

• White Noise Boundary Condition, e.g. for APE:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3   <D VAR="p" USERDEFINEDTYPE="Wall" VALUE="10" />
4   <D VAR="u" USERDEFINEDTYPE="Wall" VALUE="10" />
5   <D VAR="v" USERDEFINEDTYPE="Wall" VALUE="10" />
6   <D VAR="w" USERDEFINEDTYPE="Wall" VALUE="10" />
7 </REGION>
8 </BOUNDARYCONDITIONS>
```

The white noise BC imposes a stochastic, uniform pressure at the boundary. The implementation uses a Mersenne-Twister pseudo random number generator to generate white Gaussian noise. The standard deviation σ of the pressure is specified by the **VALUE** attribute.

6.4 Examples

6.4.1 Wave Propagation in a Sheared Base Flow

In this section we explain how to set up a simple, 2D simulation of aeroacoustics in Nektar++. We will study the propagation of an acoustic wave in the simple case of a sheared base flow, i.e. $\overline{\boldsymbol{u}} = [300 \tanh(20x_2), 0]^{\mathrm{T}}$, $\overline{c^2} = (341 \,\mathrm{m/s})^2$, $\overline{\rho} = 1.204 \,\mathrm{kg/m^3}$. The geometry consists of 64 quadrilateral elements.

6.4.1.1 Input file

We require a discontinuous Galerkin projection and use an explicit fourth-order Runge-Kutta time integration scheme. We therefore set the following time integration schem and solver information:

```
1 <TIMEINTEGRATIONSCHEME>
      <METHOD> RungeKutta </METHOD>
2
     <ORDER> 4 </ORDER>
3
4 </TIMEINTEGRATIONSCHEME>
5
6 <SOLVERINFO>
   <I PROPERTY="EQType"
                                        VALUE="APE"/>
7
     <I PROPERTY="Projection"
                                         VALUE="DisContinuous"/>
8
     <I PROPERTY="UpwindType"
9
                                         VALUE="LaxFriedrichs"/>
10 </SOLVERINFO>
```

To maintain numerical stability we must use a small time-step. Finally, we set the density, heat ratio and ambient pressure.

```
1 <PARAMETERS>

2 <P> TimeStep = 1e-05 </P>

3 <P> NumSteps = 1000 </P>

4 <P> FinTime = TimeStep*NumSteps </P>

5 <P> IO_CheckSteps = 10 </P>

6 <P> IO_InfoSteps = 10 </P>

7 </PARAMETERS>
```

The initial condition and the base flow field are specified by the **Baseflow** and **InitialConditions** functions, respectively:

```
1 <FUNCTION NAME="Baseflow">
2
   <E VAR="u0" VALUE="300 * tanh(2*y/0.1)"/>
3
   <E VAR="v0" VALUE="0"/>
4
   <E VAR="c0sq" VALUE="1.4 * Pinfinity / Rho0"/>
   <E VAR="rho0" VALUE="Rho0"/>
5
6 </FUNCTION>
7 <FUNCTION NAME="InitialConditions">
8 <E VAR="p" VALUE="0"/>
   <E VAR="u" VALUE="0"/>
9
10 <E VAR="v" VALUE="0"/>
11 </FUNCTION>
```

At all four boundaries the (RiemannInvariantBC) condition is used:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3  <D VAR="p" USERDEFINEDTYPE="RiemannInvariantBC"/>
4  <D VAR="u" USERDEFINEDTYPE="RiemannInvariantBC"/>
5  <D VAR="v" USERDEFINEDTYPE="RiemannInvariantBC"/>
6  </REGION>
7 </BOUNDARYCONDITIONS>
```

The system is excited via an acoustic source term $\dot{\omega}_{c}$, which is modeled by a field forcing as:

```
1 <FORCING>
2 <FORCE TYPE="Field">
3 <FIELDFORCE> Source <FIELDFORCE/>
4 </FORCE>
5 </FORCING>
```

and the corresponding function

```
1 <FUNCTION NAME="Source">
2  <E VAR="p" VALUE="100 * 2*PI*5E2 * cos(2*PI*5E2 * t) * exp(-32*(x^2+y^2))"/>
3  <E VAR="u" VALUE="0"/>
4  <E VAR="v" VALUE="0"/>
5 </FUNCTION>
```

6.4.1.2 Running the code

AcousticSolver Test_pulse.xml

6.4.1.3 Results

Fig. 6.1 shows the acoustic source term, the velocity and the acoustic pressure and velocity perturbations at a single time step.



Figure 6.1 Acoustic source term, base flow velocity, acoustic pressure and acoustic velocity perturbations.

CHAPTER 7

Advection-Diffusion-Reaction Solver

7.1 Synopsis

The ADRSolver is designed to solve partial differential equations of the form:

$$\alpha \frac{\partial u}{\partial t} + \lambda u + \nu \nabla u + \epsilon \nabla \cdot (D \nabla u) = f$$
(7.1)

in either discontinuous or continuous projections of the solution field. For a full list of the equations which are supported, and the capabilities of each equation, see the table below.

Equation to solve	EquationType	Dimensions	Projections
u = f	Projection	All	Continuous/Discontinuous
$\nabla^2 u = 0$	Laplace	All	Continuous/Discontinuous
$\nabla^2 u = f$	Poisson	All	Continuous/Discontinuous
$\nabla^2 u + \lambda u = f$	Helmholtz	All	Continuous/Discontinuous
$\epsilon \nabla^2 u + \mathbf{V} \nabla u = f$	SteadyAdvectionDiffusion	2D only	Continuous/Discontinuous
$\epsilon \nabla^2 u + \lambda u = f$	SteadyDiffusionReaction	2D only	Continuous/Discontinuous
$\epsilon \nabla^2 u + \mathbf{V} \nabla u + \lambda u = f$	SteadyAdvectionDiffusionReaction	2D only	Continuous/Discontinuous
$\frac{\partial u}{\partial t} + \mathbf{V} \nabla u = f$	UnsteadyAdvection	All	Continuous/Discontinuous
$\frac{\partial u}{\partial t} = \epsilon \nabla^2 u$	UnsteadyDiffusion	All	Continuous/Discontinuous
$\frac{\partial u}{\partial t} = \epsilon \nabla^2 u + R(u)$	UnsteadyReactionDiffusion	All	Continuous
$\frac{\partial u}{\partial t} + \mathbf{V} \nabla u = \epsilon \nabla^2 u$	UnsteadyAdvectionDiffusion	All	Continuous/Discontinuous
$\frac{\partial u}{\partial t} + u\nabla u = 0$	UnsteadyInviscidBurger	1D only	Continuous/Discontinuous

Table 7.1 Equations supported by the ADRSolver with their capabilities.

144 Chapter 7 Advection-Diffusion-Reaction Solver

7.2 Usage

ADRSolver session.xml

7.3 Session file configuration

The type of equation which is to be solved is specified through the EquationType SOLVERINFO option in the session file. This can be set as in table 7.1. At present, the Steady non-symmetric solvers cannot be used in parallel.

7.3.1 Time Integration Scheme

• **TimeIntegrationScheme**: The following types of time integration schemes have been tested with each solver:

EqType	Explicit	Diagonally Implicit	IMEX	Implicit
UnsteadyAdvection	\checkmark			
UnsteadyDiffusion	\checkmark	\checkmark		
UnsteadyReactionDiffusion			\checkmark	
UnsteadyAdvectionDiffusion			\checkmark	
UnsteadyInviscidBurger	\checkmark			

7.3.2 Solver Info

The solver info are listed below:

- **Eqtype**: This sets the type of equation to solve, according to the table above.
- **Projection**: The Galerkin projection used may be either:
 - Continuous for a C0-continuous Galerkin (CG) projection.
 - Discontinuous for a discontinous Galerkin (DG) projection.
- **DiffusionAdvancement**: This specifies how to treat the diffusion term. This will be restricted by the choice of time integration scheme:
 - Explicit Requires the use of an explicit time integration scheme.
 - Implicit Requires the use of a diagonally implicit, IMEX or Implicit scheme.
- AdvectionAdvancement: This specifies how to treat the advection term. This will be restricted by the choice of time integration scheme:

- Explicit Requires the use of an explicit or IMEX time integration scheme.
- Implicit Not supported at present.
- AdvectionType: Specifies the type of advection:
 - NonConservative (for CG only).
 - WeakDG (for DG only).
- DiffusionType:
 - LDG (The penalty term is proportional to an optional parameter LDGc11) which is by default set to one; proportionality to polynomial order can be manually imposed by setting the parameter (LDGc11) equal to p^2).

• UpwindType:

- Upwind.

7.3.3 Parameters

The following parameters can be specified in the **PARAMETERS** section of the session file:

- **epsilon**: sets the diffusion coefficient ϵ . *Can be used* in: SteadyDiffusionReaction, SteadyAdvectionDiffusionReaction, UnsteadyDiffusion, UnsteadyAdvectionDiffusion. *Default value*: 0.
- d00, d11, d22: sets the diagonal entries of the diffusion tensor D.
 Can be used in: UnsteadyDiffusion
 Default value: All set to 1 (i.e. identity matrix).
- lambda: sets the reaction coefficient λ.
 Can be used in: SteadyDiffusionReaction, Helmholtz, SteadyAdvectionDiffusionReaction
 Default value: 0.

7.3.4 Functions

The following functions can be specified inside the **CONDITIONS** section of the session file:

- AdvectionVelocity: specifies the advection velocity V.
- InitialConditions: specifies the initial condition for unsteady problems.
- Forcing : specifies the forcing function f.

7.4 Examples

Example files for the ADRSolver are provided in solvers/ADRSolver/Examples

7.4.1 1D Advection equation

In this example, it will be demonstrated how the Advection equation can be solved on a one-dimensional domain.

7.4.1.1 Advection equation

We consider the hyperbolic partial differential equation:

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0, \tag{7.2}$$

where f = au is the advection flux.

7.4.1.2 Input file

The input for this example is given in the example file Advection1D.xml

The geometry section defines a 1D domain consisting of 10 segments. On each segment an expansion consisting of 4 Lagrange polynomials on the Gauss-Lobotto-Legendre points is used as specified by

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" FIELDS="u" TYPE="GLL_LAGRANGE_SEM" NUMMODES="4"/>
3 </EXPANSIONS>
```

Since we are solving the unsteady advection problem, we must specify this in the solver information. We also choose to use a discontinuous flux-reconstruction projection and use a Runge-Kutta order 4 time-integration scheme.

1	<timeintegrationscheme></timeintegrationscheme>			
2	<pre><method> RungeKutta </method></pre>			
3	<order> 4 </order>			
4				
5				
6	<solverinfo></solverinfo>			
7	<i <="" property="EQTYPE" td=""><td>VALUE="UnsteadyAdvection"</td><td>/></td><td></td></i>	VALUE="UnsteadyAdvection"	/>	
8	<i <="" property="Projection" td=""><td>VALUE="DisContinuous"</td><td>/></td><td></td></i>	VALUE="DisContinuous"	/>	
9	<pre><i <="" pre="" property="AdvectionType"></i></pre>	VALUE="FRDG"	/>	
10	<i <="" property="UpwindType" td=""><td>VALUE="Upwind"</td><td>/></td><td></td></i>	VALUE="Upwind"	/>	
11				

We choose to advect our solution for 20 time units with a time-step of 0.01 and so provide the following parameters

1	<p></p>	FinTime	=	20	
2	<p></p>	TimeStep	=	0.01	
3	<p></p>	NumSteps	=	FinTime/TimeStep	

We also specify the advection velocity. We first define dummy parameters

1	<p></p>	advx	=	1	
2	<p></p>	advy	=	0	

and then define the actual advection function as

Two boundary regions are defined, one at each end of the domain, and periodicity is enforced

```
1 <BOUNDARYREGIONS>
     <B ID="0"> C[1] </B>
2
     <B ID="1"> C[2] </B>
3
4 </BOUNDARYREGIONS>
5
6 <BOUNDARYCONDITIONS>
    <REGION REF="0">
7
        <P VAR="u" VALUE="[1]" />
8
9
     </REGION>
10
     <REGION REF="1">
       <P VAR="u" VALUE="[0]" />
11
12
     </REGION>
13 </BOUNDARYCONDITIONS>
```

Finally, we specify the initial value of the solution on the domain

```
1 <FUNCTION NAME="InitialConditions">
2  <E VAR="u" VALUE="exp(-20.0*x*x)" />
3 </FUNCTION>
4
5 <FUNCTION NAME="ExactSolution">
6  <E VAR="u" VALUE="exp(-20.0*x*x)" />
7 </FUNCTION>
```

7.4.1.3 Running the code

ADRSolver Advection1D.xml

To visualise the output, we can convert it into either TecPlot or VTK formats

```
FieldConvert Advection1D.xml Advection1D.fld Advection1D.dat
FieldConvert Advection1D.xml Advection1D.fld Advection1D.vtu
```

7.4.2 2D Helmholtz Problem

In this example, it will be demonstrated how the Helmholtz equation can be solved on a two-dimensional domain.

148 Chapter 7 Advection-Diffusion-Reaction Solver

7.4.2.1 Helmholtz equation

We consider the elliptic partial differential equation:

$$\nabla^2 u + \lambda u = f \tag{7.3}$$

where ∇^2 is the Laplacian and λ is a real positive constant.

7.4.2.2 Input file

The input for this example is given in the example file Helmholtz2D_modal.xml

The geometry for this problem is a two-dimensional octagonal plane containing both triangles and quadrilaterals. Note that a mesh composite may only contain one type of element. Therefore, we define two composites for the domain, while the rest are used for enforcing boundary conditions.

```
1 <COMPOSITE>

2 <C ID="0">Q[22-47] </C>

3 <C ID="1">T[0-21] </C>

4 <C ID="2">E[0-1] </C>

5 .

6 .

7 <C ID="10">E[84,75,69,62,51,40,30,20,6] </C>

8 </COMPOSITE>

9

10 <DOMAIN> C[0-1] </DOMAIN>
```

For both the triangular and quadrilateral elements, we use the modified Legendre basis with 7 modes (maximum polynomial order is 6).

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="7" FIELDS="u" TYPE="MODIFIED" />
3 <E COMPOSITE="C[1]" NUMMODES="7" FIELDS="u" TYPE="MODIFIED" />
4 </EXPANSIONS>
```

Only one parameter is needed for this problem. In this example $\lambda = 1$ and the Continuous Galerkin Method is used as projection scheme to solve the Helmholtz equation, so we need to specify the following parameters and solver information.

```
1 <PARAMETERS>
2 <P> Lambda = 1 </P>
3 </PARAMETERS>
4
5 <SOLVERINFO>
6 <I PROPERTY="EQTYPE" VALUE="Helmholtz" />
7 <I PROPERTY="Projection" VALUE="Continuous" />
8 </SOLVERINFO>
```

All three basic boundary condition types have been used in this example: Dirichlet, Neumann and Robin boundary. The boundary regions are defined, each of which corresponds to one of the edge composites defined earlier. Each boundary region is then assigned an appropriate boundary condition.

```
1 <BOUNDARYREGIONS>
      <B ID="0"> C[2] </B>
\mathbf{2}
3
4
      <B ID="8"> C[10] </B>
5
6 </BOUNDARYREGIONS>
7
8 <BOUNDARYCONDITIONS>
9
      <REGION REF="0">
10
         <D VAR="u" VALUE="sin(PI*x)*sin(PI*y)" />
11
      </REGION>
      <REGION REF="1">
12
          <R VAR="u" VALUE="sin(PI*x)*sin(PI*y)-PI*sin(PI*x)*cos(PI*y)"
13
             PRIMCOEFF="1" />
14
      </REGION>
15
      <REGION REF="2">
16
          <N VAR="u" VALUE="(5/sqrt(61))*PI*cos(PI*x)*sin(PI*y)-</pre>
17
                              (6/sqrt(61))*PI*sin(PI*x)*cos(PI*y)" />
18
      </REGION>
19
20
         .
21
22 </BOUNDARYCONDITIONS>
```

We know that for $f = -(\lambda + 2\pi^2)sin(\pi x)cos(\pi y)$, the exact solution of the two-dimensional Helmholtz equation is $u = sin(\pi x)cos(\pi y)$. These functions are defined specified to initialise the problem and verify the correct solution is obtained by evaluating the L_2 and L_{inf} errors.

```
1 <FUNCTION NAME="Forcing">
2  <E VAR="u" VALUE="-(Lambda + 2*PI*PI)*sin(PI*x)*sin(PI*y)" />
3 </FUNCTION>
4
5 <FUNCTION NAME="ExactSolution">
6  <E VAR="u" VALUE="sin(PI*x)*sin(PI*y)" />
7 </FUNCTION>
```

7.4.2.3 Running the code

ADRSolver Test_Helmholtz2D_modal.xml

This execution should print out a summary of input file, the L_2 and L_{inf} errors and the time spent on the calculation.

7.4.2.4 Post-processing

Simulation results are written in the file Helmholtz2D_modal.fld. We can choose to visualise the output in Gmsh

```
FieldConvert Helmholtz2D_modal.xml Helmholtz2D_modal.fld Helmholtz2D_modal. vtu
```

which generates the file Helmholtz2D_modal.vtu which can be visualised and is shown in Fig. 7.1



Figure 7.1 Solution of the 2D Helmholtz Problem.

7.4.3 Advection dominated mass transport in a pipe

The following example demonstrates the application of the ADRsolver for modelling advection dominated mass transport in a straight pipe. Such a transport regime is encountered frequently when modelling mass transport in arteries. This is because the diffusion coefficient of small blood borne molecules, for example oxygen or adenosine triphosphate, is very small $O(10^{-10})$.

7.4.3.1 Background

The governing equation for modelling mass transport is the unsteady advection diffusion equation:

$$\frac{\partial u}{\partial t} + v\nabla u + \epsilon\nabla^2 u = 0$$

For small diffusion coefficient, ϵ , the transport is dominated by advection and this leads to a very fine boundary layer adjacent to the surface which must be captured in order to get a realistic representation of the wall mass transfer processes. This creates problems not only from a meshing perspective, but also numerically where classical oscillations are observed in the solution due to under-resolution of the boundary layer. The Graetz-Nusselt solution is an analytical solution of a developing mass (or heat) transfer boundary layer in a pipe. Previously this solution has been used as a benchmark for the accuracy of numerical methods to capture the fine boundary layer which develops for high Peclet number transport (the ratio of advection to diffusion). The solution is derived based on the assumption that the velocity field within the mass transfer boundary layer is linear i.e. the Schmidt number (the relative thickness of the momentum to mass transfer boundary layer) is sufficiently large. The analytical solution for the non-dimensional mass transfer at the wall is given by:

$$Sh(z) = \frac{2^{4/3} (PeR/z)^{1/3}}{g^{1/3} \Gamma(4/3)},$$

where z is the streamwise coordinate, R the pipe radius, $\Gamma(4/3)$ an incomplete Gamma function and Pe the Peclet number given by:

$$Pe = \frac{2UR}{\epsilon}$$

In the following we will numerically solver mass transport in a pipe and compare the calculated mass transfer at the wall with the Graetz-Nusselt solution. The Peclet number of the transport regime under consideration is 750000, which is physiologically relevant.

7.4.3.2 Input file

The geometry under consideration is a pipe of radius, R = 0.5 and length l = 0.5



Figure 7.2 Pipe.

Since the mass transport boundary layer will be confined to a very small layer adjacent to the wall we do not need to mesh the interior region, hence the mesh consists of a layer of ten prismatic elements over a thickness of 0.036R. The elements progressively grow over the thickness of domain.

152 Chapter 7 Advection-Diffusion-Reaction Solver

In this example we utilise heterogeneous polynomial order, in which the polynomial order normal to the wall is higher so that we avoid unphysical oscillations, and hence the incorrect solution, in the mass transport boundary layer. To do this we specify explicitly the expansion type, points type and distribution in each direction as follows:

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]"
3 NUMMODES="3,5,3"
4 BASISTYPE="Modified_A,Modified_B"
5 NUMPOINTS="4,6,3"
6 POINTSTYPE="GaussLobattoLegendre,GaussLobattoLegendre,GaussRadauMAlpha1Beta0"
7 FIELDS="u" />
8 </EXPANSIONS>
```

The above represents a quadratic polynomial order in the azimuthal and streamwise direction and 4th order polynomial normal to the wall for a prismatic element.

We choose to use a continuous projection and an first-order implicit-explicit timeintegration scheme. The DiffusionAdvancement and AdvectionAdvancement parameters specify how these terms are treated.

1	<timeintegrationscheme></timeintegrationscheme>		
2	<method> IMEX </method>		
3	<pre><order> 1 </order></pre>		
4			
5			
6	<solverinfo></solverinfo>		
7	<i <="" property="EQTYPE" td=""><td>VALUE="UnsteadyAdvectionDiffusion"</td><td>/></td></i>	VALUE="UnsteadyAdvectionDiffusion"	/>
8	<i <="" property="Projection" td=""><td>VALUE="Continuous"</td><td>/></td></i>	VALUE="Continuous"	/>
9	<pre><i <="" pre="" property="DiffusionAdvancement"></i></pre>	VALUE="Implicit"	/>
10	<pre><i <="" pre="" property="AdvectionAdvancement"></i></pre>	VALUE="Explicit"	/>
11	<i <="" property="GlobalSysSoln" td=""><td>VALUE="IterativeStaticCond"</td><td>/></td></i>	VALUE="IterativeStaticCond"	/>
12			

We integrate for a total of 30 time units with a time-step of 0.0005, necessary to keep the simulation numerically stable.

1	<p></p>	TimeStep = 0.0005	
2	<p></p>	FinalTime = 30	
3	<p></p>	NumSteps = FinalTime/TimeStep	

The value of the ϵ parameter is $\epsilon = 1/Pe$

```
1 <P> epsilon = 1.33333e-6 </P>
```

The analytical solution represents a developing mass transfer boundary layer in a pipe. In order to reproduce this numerically we assume that the inlet concentration is a uniform value and the outer wall concentration is zero; this will lead to the development of the mass transport boundary layer along the length of the pipe. Since we do not model explicitly the mass transfer in the interior region of the pipe we assume that the inner wall surface concentration is the same as the inlet concentration; this assumption is valid based on the large Peclet number meaning the concentration boundary layer is confined to the region in the immediate vicinity of the wall. The boundary conditions are specified as follows in the input file:

```
1 <BOUNDARYREGIONS>
     <B ID="0"> C[3] </B> <!-- inlet -->
2
     <B ID="1"> C[4] </B> <!-- outlet -->
3
     <B ID="2"> C[2] </B> <!-- outer surface -->
4
     <B ID="3"> C[5] </B> <!-- inner surface -->
\mathbf{5}
6 </BOUNDARYREGIONS>
7
8 <BOUNDARYCONDITIONS>
   <REGION REF="0">
9
     <D VAR="u" VALUE="1" />
10
   </REGION>
11
12
    <REGION REF="1">
13
     <N VAR="u" VALUE="0" />
14
   </REGION>
    <REGION REF="2">
15
     <D VAR="u" VALUE="0" />
16
   </REGION>
17
   <REGION REF="3">
18
     <D VAR="u" VALUE="1" />
19
   </REGION>
20
21 </BOUNDARYCONDITIONS>
```

The velocity field within the domain is fully devqeloped pipe flow (Poiseuille flow), hence we can define this through an analytical function as follows:

```
1 <FUNCTION NAME="AdvectionVelocity">
2 <E VAR="Vx" VALUE="0" />
3 <E VAR="Vy" VALUE="0" />
4 <E VAR="Vz" VALUE="2.0*(1-(x*x+y*y)/0.25)" />
5 </FUNCTION>
```

We assume that the initial domain concentration is uniform everywhere and the same as the inlet. This is defined by,

```
1 <FUNCTION NAME="InitialConditions">
2 <E VAR="u" VALUE="1" />
3 </FUNCTION>
```

7.4.3.3 Results

To compare with the analytical expression we numerically calculate the concentration gradient at the surface of the pipe. This is then plotted against the analytical solution by extracting the solution along a line in the streamwise direction, as shown in Fig. 7.3.

7.4.4 Unsteady reaction-diffusion systems

Reaction-diffusion systems are prevalent in a number of areas relating to the modelling of various physical phenomena, and are particularly prevalent in the study of chemical



Figure 7.3 Concentration gradient at the surface of the pipe.

interactions and pattern formation. The ADRSolver supports the solution of a single-variable system

$$\frac{\partial u}{\partial t} = \epsilon \nabla^2 u x + R(u)$$

where the diffusion coefficient ϵ and reaction term R(u) are defined using the session file.

7.4.4.1 Numerical restrictions

The reaction-diffusion system is only supported in a selected configuration, which is mostly defined inside the **SOLVERINFO** block:

- use of a continuous Galerkin discretisation;
- use an implicit-explicit (IMEX) timestepping scheme, such as IMEXOrder3;

This naturally leads to the following **TIMEINTEGRATIONSCHEME** and **SOLVERINFO** configuration:

```
1 <TIMEINTEGRATIONSCHEME>
      <METHOD> IMEX </METHOD>
2
      <ORDER> 3 </ORDER>
3
4 </TIMEINTEGRATIONSCHEME>
\mathbf{5}
6 <SOLVERINFO>
      <I PROPERTY="EQTYPE"
                                             VALUE="UnsteadyReactionDiffusion" />
7
                                                                                  1>
      <I PROPERTY="Projection"
                                             VALUE="Continuous"
8
      <I PROPERTY="DiffusionAdvancement"
                                             VALUE="Implicit"
                                                                                  />
9
10 </SOLVERINFO>
```

Further to this, the reaction term R(u) is imposed by the definition of a body forcing function. For example, the reaction term R(u) = 0.1u may be defined using the function:

Note in particular the use of the **EVARS** (equation variables) attribute, which permits the definition of this function in terms of the scalar variable u. This function should be used together with an appropriate **FORCING** block (as described in section 3.5):

```
1 <FORCING>

2 <FORCE TYPE="Body">

3 <BODYFORCE> BodyForce </BODYFORCE>

4 </FORCE>

5 </FORCING>
```

An example of a simple unsteady reaction-diffusion problem can be found in the **Tests** directory in the session file **ReactionDiffusion2D.xml**.

CHAPTER **8**

Cardiac Electrophysiology Solver

8.1 Synopsis

The CardiacEPSolver is used to model the electrophysiology of cardiac tissue, specifically using the monodomain or bidomain model. These models are continuum models and represent an average of the electrical activity over many cells. The system is a reactiondiffusion system, with the reaction term modeling the flow of current in and out of the cells using a separate set of ODEs.

8.1.1 Bidomain Model

The Bidomain model is given by the following PDEs,

$$g_{ix}\frac{\partial^2 V_i}{\partial x^2} + g_{iy}\frac{\partial^2 V_i}{\partial y^2} = \chi \left[C_m \frac{\partial (V_i - V_e)}{\partial t} + G_m (V_i - V_e) \right]$$
$$g_{ex}\frac{\partial^2 V_e}{\partial x^2} + g_{ey}\frac{\partial^2 V_e}{\partial y^2} = -\chi \left[C_m \frac{\partial (V_i - V_e)}{\partial t} + G_m (V_i - V_e) \right].$$

However, when solving numerically, one often rewrites these equations in terms of the transmembrane potential and extracellular potential,

$$\chi \left[C_m \frac{\partial V_m}{\partial t} + J_{ion} \right] = g_{ex} \frac{\partial^2 V_e}{\partial x^2} + g_{ey} \frac{\partial^2 V_e}{\partial y^2}$$
$$(g_{ix} + g_{ex}) \frac{\partial^2 V_e}{\partial x^2} + (g_{iy} + g_{ey}) \frac{\partial^2 V_e}{\partial y^2} = -g_{ix} \frac{\partial^2 V_m}{\partial x^2} - g_{iy} \frac{\partial^2 V_m}{\partial y^2}$$

8.1.2 Monodomain Model

In the case where the intracellular and extracellular conductivities are proportional, that is $g_{ix} = kg_{ex}$ for some k, then the above two PDEs can be reduced to a single PDE:

$$\chi \left[C_m \frac{\partial V_m}{\partial t} + J_{ion} \right] = \nabla \cdot (\sigma \nabla V_m)$$

8.1.3 Cell Models

The action potential of a cardiac cell can be modelled at either a biophysical level of detail, including a number of transmembrane currents, or as a phenomenological model, to reproduce the features of the action potential, with fewer variables. Each cell model will include a unique system of ODEs to represent the gating variables of that model.

A number of ionic cell models are currently supported by the solver including:

- Courtemanche, Ramirez, Nattel, 1998
- Luo, Rudy, 1991
- ten Tusscher, Panfilov, 2006 (epicardial, endocardial and mid-myocardial variants)

Phenomological cell models are also supported:

- Aliev-Panfilov
- Fitzhugh-Nagumo

It is important to ensure that the units of the voltage and currents from the cell model are consistent with the units expected by the tissue level solver (monodomain/bidomain). We will show as an example the Courtemanche, Ramirez, Nattel, 1998 human atrial model.

The monodomain equation:

$$\chi \left[C_m \frac{\partial V_m}{\partial t} + J_{ion} \right] = \nabla \cdot (\sigma \nabla V_m)$$

8.2 Usage

CardiacEPSolver session.xml

8.3 Session file configuration

8.3.1 Solver Info

- Eqtype Specifies the PDE system to solve. The following values are supported:
 - Monodomain: solve the monodomain equation.
 - BidomainRoth: solve the bidomain equations using the Roth formulation.
- CellModel Specifies the cell model to use. Available cell models are

158 Chapter 8 Cardiac Electrophysiology Solver

Value	Description	No. of Var.	Ref.
AlievPanfilov	Phenomological	1	[2]
CourtemancheRamirezNatte198	Human atrial	20	[27]
FitzHughNagumo			
Fox02			
LuoRudy91	Mammalian ventricular	7	[25]
PanditGilesDemir03			
TenTusscher06	Human ventricular	18	[45]
Winslow99			

- **Projection** Specifies the Galerkin projection type to use. Only **Continuous** has been extensively tested.
- TimeIntegrationScheme Specifies the time integration scheme to use for advancing the PDE system. This must be an IMEX scheme. Suitable choices are: IMEX Order 1,2,3, IMEX, Variant dirk, Order 3, Free Parameters 3 4. The cell model state variables are time advanced using Forward Euler for the ion concentrations, and Rush-Larsen for the cell model gating variables.
- DiffusionAdvancement Specifies whether the diffusion is handled implicitly or explicitly in the time integration scheme. The current code only supports Implicit integration of the diffusion term. The cell model is always integrated explicitly.

8.3.2 Parameters

The following parameters can be specified in the **PARAMETERS** section of the session file. Example values are taken from [11].

- Chi sets the surface-to-volume ratio (Units: mm⁻¹). Example: $\chi = 140$ mm⁻¹
- Cm sets the specific membrane capacitance (Units: $\mu F mm^{-2}$). Example: $C_m = 0.01 \mu F mm^{-2}$
- Substeps sets the number of substeps taken in time integrating the cell model for each PDE timestep. Example: 4
- d_min, d_max, o_min, o_max specifies a bijective map to assign conductivity values σ to intensity values μ when using the IsotropicConductivity function. The intensity map is first thresholded to the range $[d_{\min}, d_{\max}]$ and then the conductivity is calculated as

$$\sigma = \frac{o_{\max} - o_{\min}}{d_{\max} - d_{\min}} (1 - \mu) + o_{\min}$$

8.3.3 Functions

The following functions can be specified inside the **CONDITIONS** section of the session file. If both are specified, the effect is multiplicative. Example values are taken from [11].

• IsotropicConductivity specifies the conductivity σ of the tissue. Example: $\sigma = 0.13341 \text{ mS mm}^{-1}$, based on $\sigma = \frac{\sigma_i \sigma_e}{\sigma_i + \sigma_e}$, $\sigma_i = 0.17$, $\sigma_e = 0.62 \text{ mS mm}^{-1}$

The variable name to use is **intensity** since the conductivity may be derived from late-Gadolinium enhanced MRA imaging. Example specifications are

```
1 <E VAR="intensity" VALUE="0.13341" />
2 <F VAR="intensity" FILE="scarmap.con" />
```

where scarmap.con is a Nektar++ field file containing a variable intensity describing the conductivity across the domain.

• AnisotropicConductivity specifies the conductivity σ of the tissue.

8.3.4 Filters

The following filters are supported exclusively for the cardiac EP solver. Further filters from section 3.4 are also available for this solver.

- Benchmark (section 3.4.3)
- CellHistoryPoints (section 3.4.4)
- CheckpointCellModel (section 3.4.5)
- Electrogram (section 3.4.7)

8.3.5 Stimuli

Electrophysiological propagaion is initiated through the stimulus current I_{ion} . The **STIMULI** section describes one or more regions of stimulus and the time-dependent protocol with which they are applied.

```
1 <STIMULI>
2 ...
3 </STIMULI>
```

A number of stimulus types are available

8.3.5.1 Stimulus types

• StimulusRect stimulates a cuboid-shaped region of the domain, specified by two coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) . An additional parameter specifies the

"smoothness" of the boundaries of the region; higher values produce a sharper boundary. Finally, the maximum strength of the stimulus current is specified in $\mu A/mm^3$

```
1 <STIMULUS TYPE="StimulusRect" ID="0">
2 <p_x1> -15.24 </p_x1>
3 <p_y1> 14.02 </p_y1>
4 <p_z1> 6.87 </p_z1>
5 <p_x2> 12.23 </p_x2>
6 <p_y2> 16.56 </p_y2>
7 <p_z2> 8.88 </p_z2>
8 <p_is> 100.00 </p_is>
9 <p_strength> 50.0 </p_strength>
10 </STIMULUS>
```

• **StimulusCirc** stimulates a spherical region of the domain, as specified by a centre and radius. The smoothness and strength parameters are also specified as for 'StimulusRect'.

```
1 <STIMULUS TYPE="StimulusCirc" ID="0">
2 <p_x1> -15.24 </p_x1>
3 <p_y1> 14.02 </p_y1>
4 <p_z1> 6.87 </p_z1>
5 <p_r1> 12.23 </p_r1>
6 <p_is> 100.00 </p_is>
7 <p_strength> 50.0 </p_strength>
8 </STIMULUS>
```

8.3.5.2 Protocols

A protocol specifies the time-dependent function indicating the strength of the stimulus and one such **PROTOCOL** section should be included within each **STIMULUS**. This can be expressed as one of:

• ProtocolSingle a single stimulus is applied at a given start time and for a given duration

```
1 <PROTOCOL TYPE="ProtocolSingle">
2 <START> 0.0 </START>
3 <DURATION> 2.0 </DURATION>
4 </PROTOCOL>
```

• **ProtocolS1** a train of pulses of fixed duration applied at a given start time and with a given cycle length.

```
1 <PROTOCOL TYPE="ProtocolS1">

2 <START> 0.0 </START>

3 <DURATION> 2.0 </DURATION>

4 <S1CYCLELENGTH> 300.0 </S1CYCLELENGTH>

5 <NUM_S1> 5 </NUM_S1>

6 </PROTOCOL>
```

• **ProtocolS1S2** same as 'ProtocolS1' except with an additional single pulse applied at a different cycle length at the end of the train of S1 pulses.

1 <PROTOCOL TYPE="ProtocolS1S2">
2 <START> 0.0 </START>
3 <DURATION> 2.0 </DURATION>
4 <S1CYCLELENGTH> 300.0 </S1CYCLELENGTH>
5 <NUM_S1> 5 </NUM_S1>
6 <S2CYCLELENGTH> 100.0 </S2CYCLELENGTH>
7 </PROTOCOL>

CHAPTER 9

Compressible Flow Solver

9.1 Synopsis

The CompressibleFlowSolver allows us to solve the unsteady compressible Euler and Navier-Stokes equations for 1D/2D/3D problems using a discontinuous representation of the variables. In the following we describe both the compressible Euler and the Navier-Stokes equations.

9.1.1 Euler equations

The Euler equations can be expressed as a hyperbolic conservation law in the form

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_i}{\partial x} + \frac{\partial \mathbf{g}_i}{\partial y} + \frac{\partial \mathbf{h}_i}{\partial z} = 0, \qquad (9.1)$$

where **q** is the vector of the conserved variables, $\mathbf{f}_i = \mathbf{f}_i(\mathbf{q})$, $\mathbf{g}_i = \mathbf{g}_i(\mathbf{q})$ and $\mathbf{h}_i = \mathbf{h}_i(\mathbf{q})$ are the vectors of the inviscid fluxes

where ρ is the density, u, v and w are the velocity components in x, y and z directions, p is the pressure and E is the total energy. In this work we considered a perfect gas law for which the pressure is related to the total energy by the following expression

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2 + w^2), \qquad (9.3)$$

where γ is the ratio of specific heats.

9.1.2 Compressible Navier-Stokes equations

The Navier-Stokes equations include the effects of fluid viscosity and heat conduction and are consequently composed by an inviscid and a viscous flux. They depend not only on the conserved variables but also, indirectly, on their gradient. The second order partial differential equations for the three-dimensional case can be written as:

$$\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} + \frac{\partial \mathbf{h}}{\partial z} = 0, \qquad (9.4)$$

where **q** is the vector of the conserved variables, $\mathbf{f} = \mathbf{f}(\mathbf{q}, \nabla(\mathbf{q})), \mathbf{g} = \mathbf{g}(\mathbf{q}, \nabla(\mathbf{q}))$ and $\mathbf{h} = \mathbf{h}(\mathbf{q}, \nabla(\mathbf{q}))$ are the vectors of the fluxes which can also be written as:

$$\mathbf{f} = \mathbf{f}_i - \mathbf{f}_v, \mathbf{g} = \mathbf{g}_i - \mathbf{g}_v, \mathbf{h} = \mathbf{h}_i - \mathbf{h}_v, \tag{9.5}$$

where \mathbf{f}_i , \mathbf{g}_i and \mathbf{h}_i are the inviscid fluxes of Eq. (9.2) and \mathbf{f}_v , \mathbf{g}_v and \mathbf{h}_v are the viscous fluxes which take the following form:

$$\mathbf{f}_{v} = \left\{ \begin{array}{c} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{zx} \\ u\tau_{xx} + v\tau_{yx} + w\tau_{zx} + kT_{x} \end{array} \right\}, \quad \mathbf{g}_{v} = \left\{ \begin{array}{c} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{zy} \\ u\tau_{xy} + v\tau_{yy} + w\tau_{zy} + kT_{y} \end{array} \right\}, \quad (9.6)$$
$$\mathbf{h}_{v} = \left\{ \begin{array}{c} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + kT_{z} \end{array} \right\},$$

where τ_{xx} , τ_{xy} , τ_{xz} , τ_{yx} , τ_{yx} , τ_{yy} , τ_{zx} , τ_{zy} and τ_{zz} are the components of the stress tensor¹

$$\tau_{xx} = 2\mu \left(u_x - \frac{u_x + v_y + w_z}{3} \right), \quad \tau_{yy} = 2\mu \left(v_y - \frac{u_x + v_y + w_z}{3} \right),$$

$$\tau_{zz} = 2\mu \left(w_z - \frac{u_x + v_y + w_z}{3} \right), \quad \tau_{xy} = \tau_{yx} = \mu (v_x + u_y),$$

$$\tau_{yz} = \tau_{zy} = \mu (w_y + v_z), \quad \tau_{zx} = \tau_{xz} = \mu (u_z + w_x).$$

(9.7)

where μ is the dynamic viscosity calculated using the Sutherland's law and k is the thermal conductivity.

9.1.3 Numerical discretisation

In Nektar++ the spatial discretisation of the Euler and of the Navier-Stokes equations is projected in the polynomial space via a discontinuous projection. Specifically we make use either of the discontinuous Galerkin (DG) method or the Flux Reconstruction (FR)

¹Note that we use Stokes hypothesis $\lambda = -2/3$.

164 Chapter 9 Compressible Flow Solver

approach. In both the approaches the physical domain Ω is divided into a mesh of N nonoverlapping elements Ω_e and the solution is allowed to be discontinuous at the boundary between two adjacent elements. Since the Euler as well as the Navier-Stokes equations are defined locally (on each element of the computational domain), it is necessary to define a term to couple the elements of the spatial discretisation in order to allow information to propagate across the domain. This term, called numerical interface flux, naturally arises from the discontinuous Galerkin formulation as well as from the Flux Reconstruction approach.

For the advection term it is common to solve a Riemann problem at each interface of the computational domain through exact or approximated Riemann solvers. In Nektar++ there are different Riemann solvers, one exact and nine approximated. The exact Riemann solver applies an iterative procedure to satisfy conservation of mass, momentum and energy and the equation of state. The left and right states are connected either with the unknown variables through the Rankine-Hugoniot relations, in the case of shock, or the isentropic characteristic equations, in the case of rarefaction waves. Across the contact surface, conditions of continuity of pressure and velocity are employed. Using these equations the system can be reduced to a non-linear algebraic equation in one unknown (the velocity in the intermediate state) that is solved iteratively using a Newton method. Since the exact Riemann solver gives a solution with an order of accuracy that is related to the residual in the Newton method, the accuracy of the method may come at high computational cost. The approximated Riemann solvers are simplifications of the exact solver.

Concerning the diffusion term, the coupling between the elements can be achieved by using local discontinuous Galerkin (LDG) approach, interior penalty method or five different FR diffusion terms.

The boundary conditions are also implemented by exploiting the numerical interface fluxes just mentioned. For a more detailed description of the above the interested reader can refer to [7] and [30].

9.2 Usage

CompressibleFlowSolver session.xml

9.3 Session file configuration

In the following we describe the session file configuration. Specifically we consider the sections under the tag **<CONDITIONS>** in the session (.xml) file.

Parameters

Under this section it is possible to set the parameters of the simulation.

1	<paran< th=""><th>IETERS></th><th></th><th></th><th></th></paran<>	IETERS>			
2	<p></p>	TimeStep	=	0.000001	
3	<p></p>	FinTime	=	1.0	
4	<p></p>	NumSteps	=	FinTime/TimeStep	
5	<p></p>	IO_CheckSteps	=	5000	
6	<p></p>	IO_InfoSteps	=	1	
$\overline{7}$	<p></p>	Gamma	=	1.4	
8	<p></p>	pInf	=	101325	
9	<p></p>	rhoInf	=	1.225	
10	<p></p>	GasConstant	=	287.058	
11	<p></p>	TInf	=	pInf/(287.058*rhoInf)	
12	<p></p>	Twall	=	pInf/(287.058*rhoInf)+15.0	
13	<p></p>	uInf	=	147.4	
14	<p></p>	vInf	=	0.0	
15	<p></p>	wInf	=	0.0	
16	<p></p>	mu	=	1e-5	
17	<p></p>	Pr	=	0.72	
18	<p></p>	${\tt thermalConductivity}$	=	0.02	
19	<p></p>	IO_Timer_Level	=	3	
20	<td>METERS></td> <td></td> <td></td> <td></td>	METERS>			

- TimeStep is the time-step we want to use.
- FinTime is the final physical time at which we want our simulation to stop.
- NumSteps is the equivalent of FinTime but instead of specifying the physical final time we specify the number of time-steps.
- **IO_CheckSteps** sets the number of steps between successive checkpoint files. No checkpoint file is written if it is set to 0.
- **IO_InfoSteps** sets the number of steps between successive info stats are printed to screen.
- Gamma ratio of the specific heats. Default value = 1.4.
- **pInf** farfield pressure (i.e. p_{∞}). Default value = 101325 *Pa*.
- **rhoInf** farfield density (i.e. ρ_{∞}). Default value = 1.225 Kg/m³.
- **GasConstant** universal gas contant. Default value = $287.058 \ JKg^{-1}K^{-1}$.
- (TInf) farfield temperature (i.e. T_{∞}). Default value = 288.15 K.
- Twall temperature at the wall when isothermal boundary conditions are employed (i.e. T_w). Default value = 300.15K.
- **uInf** farfield X-component of the velocity (i.e. u_{∞}). Default value = 0.1 m/s.
- vInf farfield Y-component of the velocity (i.e. v_{∞}). Default value = 0.0 m/s.
- (wInf) farfield Z-component of the velocity (i.e. w_{∞}). Default value = 0.0 m/s.

- mu dynamic viscosity (i.e. μ_{∞}). Default value = 1.78e-05 *Pas*.
- (Pr) Prandtl number. Default value = 0.72.
- thermalConductivity thermal conductivity (i.e. κ_{∞}). This can be set as an alternative to **Pr**, in which case the Prandtl number is calculated from κ_{∞} (it is only possible to set one of them). By default, this is obtained from the Prandtl number.
- CFL is the CFL number (explicit and implicit solvers).
- **CFLGrowth** is the growing CFL (explicit and implicit solvers).
- CFLEnd is the maximum value of the CFL number (explicit and implicit solvers).
- Timer_IO_Level defines the amount of timer information that is printed after the solver is finished. The default value is -1, which disables output. By selecting a value between 0 and 2, more detailed timer information is printed.

Time Integration Scheme

Under this section it is possible to set the time integration scheme information.

```
1 <TIMEINTEGRATIONSCHEME>
2 <METHOD> RungeKutta </METHOD>
3 <VARIANT> SSP </VARIANT>
4 <ORDER> 3 </ORDER>
5 </TIMEINTEGRATIONSCHEME>
```

• **TimeIntegrationScheme** is the time-integration scheme we want to use. There are implicit and explicit schemes for the Compressible flow solver. For an explicit discretization, the time-integration schemes supported are as follows

Name	<method></method>	<pre><variant></variant></pre>	<order></order>
Forward Euler	ForwardEuler	-	1
Runge Kutta 2 - ${\rm SSP}$	RungeKutta	SSP	2
Runge Kutta 3 - SSP	RungeKutta	SSP	3
Runge Kutta 4	ClassicalRungeKutta	-	4
Runge Kutta 5	RungeKutta	-	5

For an implicit discretization, the time-integration schemes available are

Name	<method></method>	<pre> <variant></variant></pre>	<pre><order></order></pre>
Backward Euler	BackwardEuler	-	1
Backward Differentiation Formula Implicit	BDFImplicit	-	1
Backward Differentiation Formula Implicit	BDFImplicit	-	2
Singly Diagonally Implicit Runge Kutta	DIRK	-	2
Singly Diagonally Implicit Runge Kutta	DIRK	-	3
Singly Diagonally Implicit Runge Kutta	DIRK	ES5	3
Singly Diagonally Implicit Runge Kutta	DIRK	-	4
Singly Diagonally Implicit Runge Kutta	DIRK	ES5	4

Solver info

Under this section it is possible to set the solver information.

1	<sol< td=""><td>/ERINFO></td><td></td><td></td></sol<>	/ERINFO>		
2	<i< td=""><td>PROPERTY="EQTYPE"</td><td>VALUE="NavierStokesImplicitCFE"</td><td>/></td></i<>	PROPERTY="EQTYPE"	VALUE="NavierStokesImplicitCFE"	/>
3	<i< td=""><td>PROPERTY="Projection"</td><td>VALUE="DisContinuous"</td><td>/></td></i<>	PROPERTY="Projection"	VALUE="DisContinuous"	/>
4	<i< td=""><td>PROPERTY="TimeIntegrationMethod"</td><td>VALUE="DIRKOrder2"</td><td>/></td></i<>	PROPERTY="TimeIntegrationMethod"	VALUE="DIRKOrder2"	/>
5	<i< td=""><td>PROPERTY="AdvectioType"</td><td>VALUE="WeakDG"</td><td>/></td></i<>	PROPERTY="AdvectioType"	VALUE="WeakDG"	/>
6	<i< td=""><td>PROPERTY="DiffusionType"</td><td>VALUE="InteriorPenalty"</td><td>/></td></i<>	PROPERTY="DiffusionType"	VALUE="InteriorPenalty"	/>
7	<i< td=""><td>PROPERTY="UpwindType"</td><td>VALUE="Roe"</td><td>/></td></i<>	PROPERTY="UpwindType"	VALUE="Roe"	/>
8	<i< td=""><td>PROPERTY="ProblemType"</td><td>VALUE="General"</td><td>/></td></i<>	PROPERTY="ProblemType"	VALUE="General"	/>
9	<i< td=""><td>PROPERTY="ViscosityType"</td><td>VALUE="Constant"</td><td>/></td></i<>	PROPERTY="ViscosityType"	VALUE="Constant"	/>
10	<i< td=""><td>PROPERTY="EquationOfState"</td><td>VALUE="IdealGas"</td><td>/></td></i<>	PROPERTY="EquationOfState"	VALUE="IdealGas"	/>
11	<i< td=""><td>PROPERTY="Driver"</td><td>VALUE="Standard"</td><td>/></td></i<>	PROPERTY="Driver"	VALUE="Standard"	/>
12	<td>LVERINFO></td> <td></td> <td></td>	LVERINFO>		

- EQType is the tag which specify the equations we want solve: Explicit discretization in time:
 - NavierStokesCFE) (Compressible Navier-Stokes equations).
 - EulerCFE (Compressible Euler equations).
 - [IsentropicVortex] (Isentropic vortex test case).
 - RinglebFlow (Ringleb flow test case).

Implicit discretization in time:

- NavierStokesImplicitCFE (Compressible Navier-Stokes equations).
- EulerImplicitCFE (Compressible Euler equations).
- **Projection** is the type of projection we want to use:
 - DisContinuous.

Note that the Continuous projection is not supported in the Compressible Flow Solver.

- 168 Chapter 9 Compressible Flow Solver
 - AdvectionType is the advection operator we want to use.
 - WeakDG (classical DG in weak form).
 - **FRDG** (Flux-Reconstruction recovering nodal DG scheme).
 - **FRSD** (Flux-Reconstruction recovering a spectral difference (SD) scheme).
 - FRHU (Flux-Reconstruction recovering Huynh (G2) scheme).
 - FRcmin (Flux-Reconstruction with $c = c_{min}$).
 - FRcinf (Flux-Reconstruction with $c = \infty$).

Note that only WeakDG is fully supported, the other operators work only with quadrilateral elements (2D or 2.5D).

- DiffusionType is the diffusion operator we want to use for the Navier-Stokes equations:
 - **LDGNS** (LDG with primitive variables. The penalty term is inversely proportional to the element size, proportional to the local viscosity for the momentum equations and to the thermal conductivity for the energy equation, and proportional to an optional parameter **LDGNSc11** which is by default set to one; proportionality to polynomial order can be manually imposed by setting the parameter (LDGNSc11) equal to p^2).
 - (LFRDGNS) (Flux-Reconstruction recovering nodal DG scheme).
 - (LFRSDNS) (Flux-Reconstruction recovering a spectral difference (SD) scheme).
 - (LFRHUNS) (Flux-Reconstruction recovering Huynh (G2) scheme).
 - **LFRcminNS** (Flux-Reconstruction with $c = c_{min}$).
 - **LFRcinfNS** (Flux-Reconstruction with $c = \infty$).
 - InteriorPenalty (Symmetric interior penalty method).

Note that only (LDGNS) and (InteriorPenalty) are fully supported, the other operators work only with quadrilateral elements (2D or 2.5D).

- UpwindType is the numerical interface flux (i.e. Riemann solver) we want to use for the advection operator:
 - (AUSMO).
 - AUSM1.
 - (AUSM2).
 - AUSM3.
 - Average.
 - ExactToro.
 - (HLL).
- HLLC.
- LaxFriedrichs.
- Roe.
- ViscosityType is the viscosity type we want to use:
 - Constant (Constant viscosity).
 - Variable (Variable viscosity through the Sutherland's law).
- EquationOfState allows selecting an equation of state for accounting for non-ideal gas behaviour:
 - IdealGas (default option).
 - VanDerWaals (requires additional parameters Tcrit) and Pcrit).
 - RedlichKwong (requires additional parameters Tcrit) and Pcrit).
 - (PengRobinson) (requires additional parameters (Tcrit), (Pcrit) and (AcentricFactor)).
- **Driver** specifies the type of problem to be solved:
 - (Standard) (default option to solve the unsteady equations).
 - SteadyState (uses the Selective Frequency Damping method (see Sec. 11.1.5) to obtain a steady-state solution of the Navier-Stokes equations (explicit or implicit)).
- ShockCaptureType specifies the type of operator to be used for shock capturing:
 - NonSmooth add a Laplacian operator to apply artificial diffusion (see Sec. 9.4.1.1).
 - Physical add artificial viscosity to the physical viscosity.
- ShockSensorType specifies the sensor type of shock capturing to be used:
 - Modal (default) use a modal sensor to identify where to add viscosity (see Sec. 9.4.1.2).
 - (Dilatation) use a dilatation sensor to identify where to add viscosity.
- DucrosSensor apply a Ducros [10] (anti-vorticity) filter to the shock sensor:
 - (On)
 - (0ff)
- Smoothing apply a smoothing filter to the shock sensor:
 - (CO) smooth the artificial viscosity to be a continuous field.

170 Chapter 9 Compressible Flow Solver

Boundary conditions

In this section we can specify the boundary conditions for our problem. First we need to define the variables under the section (VARIABLES). For a 1D problem we have:

```
1 <VARIABLES>

2 <V ID="0"> rho </V>

3 <V ID="1"> rhou </V>

4 <V ID="2"> E </V>

5 </VARIABLES>
```

For a 2D problem we have

1 <VARIABLES> 2 <V ID="0"> rho </V> 3 <V ID="1"> rhou </V> 4 <V ID="2"> rhov </V> 5 <V ID="3"> E </V 6 </VARIABLES>

For a 3D problem we have:

1 <VARIABLES> 2 <V ID="0"> rho </V> 3 <V ID="1"> rhou </V> 4 <V ID="2"> rhov </V> 5 <V ID="3"> rhow </V> 6 <V ID="4"> E </V> 7 </VARIABLES>

After having defined the variables depending on the dimensions of the problem we want to solve, it is necessary to specify the boundary regions on which we want to define the boundary conditions:

```
1 <BOUNDARYREGIONS>
2 <B ID="0"> C[100] </B>
3 </BOUNDARYREGIONS>
```

Finally we can specify the boundary conditions on the regions specified under **BOUNDARYREGIONS**. Note that the no-slip, isothermal, wall boundary condition requires Twall to be specified. In the following are some examples for a 2D problem:

• Slip wall boundary conditions:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="rho" USERDEFINEDTYPE="Wall" VALUE="0" />
4 <D VAR="rhou" USERDEFINEDTYPE="Wall" VALUE="0" />
5 <D VAR="rhou" USERDEFINEDTYPE="Wall" VALUE="0" />
6 <D VAR="E" USERDEFINEDTYPE="Wall" VALUE="0" />
7 </REGION>
8 </BOUNDARYCONDITIONS>
```

• No-slip wall boundary conditions:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="rho" USERDEFINEDTYPE="WallViscous" VALUE="0" />
4 <D VAR="rhou" USERDEFINEDTYPE="WallViscous" VALUE="0" />
5 <D VAR="rhov" USERDEFINEDTYPE="WallViscous" VALUE="0" />
6 <D VAR="E" USERDEFINEDTYPE="WallViscous" VALUE="0" />
7 </REGION>
8 </BOUNDARYCONDITIONS>
```

• Adiabatic wall boundary conditions:

In some cases we need to excite perturbations inside the boundary layer. This can be achieved by setting part of the wall as a disturbance strip. In the no-slip/adiabatic wall boundary conditions, if the **VALUE** is not exact "0" but any expression, which can be time-dependent, the value of the expression will be added to the ghost state of what it should be in the input boundary conditions, and then generate a non-zero flux through the Riemann solver. The following is an example to set a disturbance strip of amplitude A, frequency f, and in the range of $[x_0, x_0 + len]$ on a flat plate.

1	<boundarycondition< th=""><th>15></th></boundarycondition<>	15>
2	<region ref="0"></region>	>
3	<d <="" th="" var="rho"><th>USERDEFINEDTYPE="WallAdiabatic" VALUE="0" /></th></d>	USERDEFINEDTYPE="WallAdiabatic" VALUE="0" />
4	<d <="" th="" var="rhou"><th>USERDEFINEDTYPE="WallAdiabatic" VALUE="0" /></th></d>	USERDEFINEDTYPE="WallAdiabatic" VALUE="0" />
5	<d <="" td="" var="rhov"><td>USERDEFINEDTYPE="WallAdiabatic"</td></d>	USERDEFINEDTYPE="WallAdiabatic"
6		VALUE="A*sin(2*PI*f*t)*sin(2*PI*(x-x0)/len)
7		*(x>=x0)*(x<=(x0+len))" />
8	<d <="" th="" var="E"><th>USERDEFINEDTYPE="WallAdiabatic" VALUE="0" /></th></d>	USERDEFINEDTYPE="WallAdiabatic" VALUE="0" />
9		
10	<td>DNS></td>	DNS>

• Farfield boundary conditions (including inviscid characteristic boundary conditions):

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3  <D VAR="rho" VALUE="rhoInf" />
4  <D VAR="rhou" VALUE="rhoInf*uInf" />
5  <D VAR="rhov" VALUE="rhoInf*vInf" />
6  <D VAR="E"
7  VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)"/>
8  </REGION>
9 </BOUNDARYCONDITIONS>
```

• Pressure outflow boundary conditions:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="rho" USERDEFINEDTYPE="PressureOutflow" VALUE="0" />
4 <D VAR="rhou" USERDEFINEDTYPE="PressureOutflow" VALUE="0" />
5 <D VAR="rhov" USERDEFINEDTYPE="PressureOutflow" VALUE="0" />
6 <D VAR="E" USERDEFINEDTYPE="PressureOutflow" VALUE="0" />
7 </REGION>
8 </BOUNDARYCONDITIONS>
```

where **pOut** is the target static pressure at the boundary.

Initial conditions and exact solution

Under the two following sections it is possible to define the initial conditions and the exact solution (if existent).

```
1 <FUNCTION NAME="InitialConditions">
2
   <E VAR="rho" VALUE="rhoInf"/>
   <E VAR="rhou" VALUE="rhoInf*uInf"</pre>
                                        />
3
   <E VAR="rhov" VALUE="rhoInf*vInf" />
4
   <E VAR="E"
5
   VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)"/>
6
7 </FUNCTION>
8
9 <FUNCTION NAME="ExactSolution">
10 <E VAR="rho" VALUE="rhoInf"
                                          />
   <E VAR="rhou" VALUE="rhoInf*uInf"
                                        />
11
   <E VAR="rhov" VALUE="rhoInf*vInf"
                                        />
12
   <E VAR="E"
13
   VALUE="pInf/(Gamma-1)+0.5*rhoInf*(uInf*uInf+vInf*vInf)"/>
14
15 </FUNCTION>
```

9.4 Examples

9.4.1 Shock capturing

Compressible flows can be characterised by abrupt changes in flow variables within the flow domain often referred to as shocks. These discontinuities can lead to numerical instabilities (Gibbs phenomena). This problem is prevented by locally adding a diffusion term to the equations to damp the numerical oscillations.

9.4.1.1 Non-smooth artificial viscosity model

For the non-smooth artificial viscosity model the added artificial viscosity is constant in each element and discontinuous between the elements. The Euler system is augmented by an added Laplacian term on right hand side of equation 9.1 [38]. The diffusivity of the system is controlled by a variable viscosity coefficient ε . For consistency ε is proportional to the element size and inversely proportional to the polynomial order. Finally, from physical considerations ε needs to be proportional to the maximum characteristic speed of the problem. The final form of the artificial viscosity is

$$\varepsilon = \varepsilon_0 \frac{h}{p} \lambda_{max} S, \tag{9.8}$$

where S is a sensor.

To enable the non-smooth viscosity model, the following line has to be added to the **SOLVERINFO** section:

```
1 <SOLVERINFO>
2 <I PROPERTY="ShockCaptureType" VALUE="NonSmooth" />
3 <SOLVERINFO>
```

9.4.1.2 Modal sensor

As shock sensor, a modal resolution-based indicator is used

$$s_e = \log_{10} \left(\frac{\langle q - \tilde{q}, q - \tilde{q} \rangle}{\langle q, q \rangle} \right), \tag{9.9}$$

where $\langle \cdot, \cdot \rangle$ represents a L^2 inner product, q and \tilde{q} are the full and truncated expansions of a state variable (in our case density)

$$q(x) = \sum_{i=1}^{N(P)} \hat{q}_i \phi_i, \quad \tilde{q}(x) = \sum_{i=1}^{N(P-1)} \hat{q}_i \phi_i, \quad (9.10)$$

then the constant element-wise sensor is computed as follows

$$S_{\varepsilon} = \begin{cases} 0 & \text{if } s_e < s_0 - \kappa \\ \frac{1}{2} \left(1 + \sin \frac{\pi(s_e - s_0)}{2\kappa} \right) & \text{if } |s_e - s_0| \le \kappa \\ 1 & \text{if } s_e > s_0 + \kappa \end{cases}$$
(9.11)

where $s_0 = s_{\kappa} - 4.25 \log_{10}(p)$.

The modal sensor is enabled by default and it can be explicitly set adding the following line to the **SOLVERINFO** section:

```
1 <SOLVERINFO>

2 <I PROPERTY="ShockSensorType" VALUE="Modal" />

3 <SOLVERINFO>
```

The diffusivity and the sensor can be controlled by the following parameters:

1	<parameters></parameters>		
2	<p> Skappa</p>	= -1.3	
3	<p> Kappa</p>	= 0.2	
4	<p> mu0</p>	= 1.0	
5			



Figure 9.1 (a) Steady state solution for M = 0.8 flow at $\alpha = 1.25^{\circ}$ past a NACA 0012 profile, (b) Artificial viscosity (ε) distribution

9.4.2 Variable polynomial order

A sensor based *p*-adaptive algorithm is implemented to optimise the computational cost and accuracy. The DG scheme allows one to use different polynomial orders since the coupling between different elements are determined by common numerical fluxes and there is no further coupling between the elements. Furthermore, the initial *p*-adaptive algorithm uses the same sensor as the shock capturing algorithm to identify the smoothness of the local solution so it rather straightforward to implement both algorithms at the same time.

The polynomial order in each element can be adjusted based on the sensor value that is obtained. Initially, a converged solution is obtained after which the sensor in each element is calculated. Based on the determined sensor value and the pre-defined sensor thresholds, it is decided to increase, decrease or maintain the degree of the polynomial approximation in each element and a new converged solution is obtained.

$$p_{e} = \begin{cases} p_{e} - 1 & \text{if } s_{e} > s_{ds} \\ p_{e} + 1 & \text{if } s_{sm} < s_{e} < s_{ds} \\ p_{e} & \text{if } s_{fl} < s_{e} < s_{sm} \\ p_{e} - 1 & \text{if } s_{e} < s_{fl} \end{cases}$$
(9.12)

For now, the threshold values s_e , s_{ds} , s_{sm} and s_{fl} are determined empirically by looking at the sensor distribution in the domain. Once these values are set, two .txt files are outputted, one that has the composites called VariablePComposites.txt and one with the expansions called VariablePExpansions.txt. These values have to copied into a new .xml file to create the adapted mesh.

9.4.3 De-Aliasing Techniques

Aliasing effects, arising as a consequence of the nonlinearity of the underlying problem, need to be address to stabilise the simulations. Aliasing appears when nonlinear quantities are calculated at an insufficient number of quadrature points. We can identify two types of nonlinearities:

- PDE nonlinearities, related to the nonlinear and quasi-linear fluxes.
- Geometrical nonlinearities, related to the deformed/curves meshes.

We consider two de-aliasing strategies based on the concept of consistent integration:

- Local dealiasing: It only targets the PDE-aliasing sources, applying a consistent integration of them locally.
- Global dealiasing: It targets both the PDE and the geometrical-aliasing sources. It requires a richer quadrature order to consistently integrate the nonlinear fluxes, the geometric factors, the mass matrix and the boundary term.

Since Nektar++ tackles separately the PDE and geometric aliasing during the projection and solution of the equations, to consistently integrate all the nonlinearities in the compressible NavierStokes equations, the quadrature points should be selected based on the maximum order of the nonlinearities:

$$Q_{min} = P_{exp} + \frac{max(2P_{exp}, P_{geom})}{2} + \frac{3}{2}$$
(9.13)

where Q_{min} is the minimum required number of quadrature points to exactly integrate the highest-degree of nonlinearity, P_{exp} being the order of the polynomial expansion and P_{geom} being the geometric order of the mesh. Bear in mind that we are using a discontinuous discretisation, meaning that aliasing effect are not fully controlled, since the boundary terms introduce non-polynomial functions into the problem.

To enable the global de-aliasing technique, modify the number of quadrature points by:

```
1 <E COMPOSITE="[101]"
2 BASISTYPE="Modified_A,Modified_A"
3 NUMMODES="7,7"
4 POINTSTYPE="GaussLobattoLegendre,GaussLobattoLegendre"
5 NUMPOINTS="14,14"
6 FIELDS="rho,rhou,rhov,E"
7 />
```

where **NUMMODES** corresponds to P+1, where P is the order of the polynomial used to approximate the solution. **NUMPOINTS** specifies the number of quadrature points.

176 Chapter 9 Compressible Flow Solver

9.4.4 Implicit solver

In this example, we solve a compressible flow past a circular cylinder using an implicit discontinuous Galerkin compressible flow solver as shown in figure 9.2. For the implicit time-integration schemes, TimeStep or CFL can be adopted to control the time step. For the case of using CFL, the CFL number can grow from CFL to CFLEnd by a ratio of (CFLGrowth) to adjust the time step at different stages of the simulation.

The CFL number, growing CFL number and the maximum value of the CFL number are controlled by the following parameters as also described in section 9.3

1	<parameters></parameters>		
2	<p> CFL</p>	= 0.1	
3	<p> CFLGrowth</p>	= 1.1	
4	<p> CFLEnd</p>	= 2.0	
5			

In this case, the numerical simulation starts from a CFL number of 0.1 and grows by a ratio of 1.1 up to a maximum value of CFL number of 2.0. Note that the CFLEnd parameter may assume higher values, depending on the strategy adopted. In addition, there is no need to define the TimeStep parameter, since the time step size is calculated based on the CFL number in each time step.

Since we are solving an implicit time-integration scheme, we must specify to the solver information the EQType which corresponds to an implicit solver. It should be noted that currently the CFS solver only supports NavierStokesImplicitCFE and EulerImplicitCFE.

```
1 <PARAMETERS>
2 <I PROPERTY="EQTYPE" VALUE="NavierStokesImplicitCFE" />
3 <I PROPERTY="TimeIntegrationMethod" VALUE="DIRKOrder2" />
4 </PARAMETERS>
```

There are some other parameters controlling the performance of the implicit solver. NonlinIterTolRelativeL2 determines the convergence tolerance of the nonlinear system solver relative to the initial nonlinear system residual. NekNonlinSysMaxIterations is the maximum iteration number of the nonlinear system solver. LinSysRelativeTolInNonlin determines the convergence tolerance of linear system solver in each nonlinear iteration. NekLinSysMaxIterations is the maximum iteration number of the linear system solver in each nonlinear iteration. LinSysMaxStorage determines the maximum number of variable vector allowed to store in the linear system solver. Specifically for GMRES solver, the GMRES solver will be restarted if NekLinSysMaxIterations is larger than LinSysMaxStorage and thus LinSysMaxStorage determines the storage consumption of the GMRES solver. Regarding the parameters to control the preconditioners in GMRES, PreconMatFreezNumb specifies the number of time steps to freeze the preconditioning matrices, in other words, the preconditioning matrices will be updated based on the number of time steps provided by the user. PreconItsStep determines the number of preconditioning iterations to calculate the preconditioned vector. The default parameters are listed below.

1	<paran< th=""><th>1ETERS></th><th></th><th></th><th></th></paran<>	1ETERS>			
2	<p></p>	NonlinIterTolRelativeL2	=	1 E -3	
3	<p></p>	NekNonlinSysMaxIterations	=	10	
4	<p></p>	${\tt LinSysRelativeTolInNonlin}$	=	5.0 <mark>E</mark> -2	
5	<p></p>	NekLinSysMaxIterations	=	30	
6	<p></p>	LinSysMaxStorage	=	30	
$\overline{7}$	<p></p>	PreconMatFreezNumb	=	200	
8	<p></p>	PreconItsStep	=	7	
9	<th>AMETERS></th> <th></th> <th></th> <th></th>	AMETERS>			

Here, we choose to solve the compressible Navier-Stokes equations and use the 2nd order Singly Diagonally Implicit Runge–Kutta (SDIRK) method.



Figure 9.2 Laminar flow past a circular cylinder at Re = 200 and M = 0.2.

Chapter 10

Dummy Solver

10.1 Synopsis

The Dummy solver does not solve any equation systems but only serves to exchange fields with other solvers and applications. It is intended for demonstrating and testing the coupling implementations only.

CHAPTER **11**

Incompressible Navier-Stokes Solver

11.1 Synopsis

A useful tool implemented in Nektar++ is the incompressible Navier Stokes solver that allows one to solve the governing equation for viscous Newtonians fluids governed by:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}$$
(11.1a)

$$\nabla \cdot \mathbf{u} = 0 \tag{11.1b}$$

where V is the velocity, p is the specific pressure (including density) and ν the kinematic viscosity.

11.1.1 Velocity Correction Scheme

The first approach uses a splitting/projection method where the velocity system and the pressure are typically decoupled. Splitting schemes are typically favoured for their numerical efficiency since the velocity and pressure are handled independently, requiring the solution of three (in two dimensions) elliptic systems of rank N (opposed to a single system of rank 3N solved in the Stokes problem). However, a drawback of this approach is the splitting scheme error which is introduced when decoupling the pressure and the velocity system, although this can be made consistent with the overall temporal accuracy of the scheme by appropriate discretisation of the pressure boundary conditions.

11.1.1.1 High order splitting scheme

In the original approach a stiffly-stable time integration was proposed in the work of Karniadakis, Israeli and Orszag [20]. This was then later fully analysed in the work of Guermond and Shen [17].

Briefly, high order splitting scheme was originally proposed in three steps involving explicit advection of the non-linear terms, followed by the solution of the pressure Poisson system and finally solving a Helmholtz problem to enforce the viscous terms and velocity boundary conditions. In the following however we briefly formulate this scheme as a two steps using a formulation outline by Guermond and Shen.

1. In the first step we formulate a weak pressure Poisson problem by taking the inner product over the solution domain Ω of equation (11.1a) with respect to the gradient of the test basis, ∇q , i.e.

$$\int_{\Omega} \nabla q \cdot \frac{\partial \mathbf{u}}{\partial t} + \int_{\Omega} \nabla q \cdot \mathbf{N}(\mathbf{u}) = -\int_{\Omega} \nabla q \cdot \nabla p + \int_{\Omega} \nabla q \cdot \nu \nabla^2 \mathbf{u}$$
(11.2)

where $N(\mathbf{u}) = \mathbf{u} \cdot \nabla \mathbf{u}$. We recall that the term $\int_{\Omega} \nabla q \cdot \nabla p$ is the weak approximation to the Laplacian operator for pressure. To decouple this term from the velocity system a few steps are necessary. Using the identity

$$abla^2 \mathbf{u} = -
abla imes
abla imes \mathbf{u} +
abla (
abla \cdot \mathbf{u})$$

we can enforce the divergence to be zero by setting the last term to zero. If we now integrate the 1st, 2nd and last term in equation (11.2) by parts we can obtain the weak pressure equation

$$\int_{\Omega} \nabla q \cdot \nabla p^{n+1} = \int_{\Omega} q \,\nabla \cdot \left(\frac{\partial \mathbf{u}^{n+1}}{\partial t} + \mathbf{N}(\mathbf{u})^{n+1} \right) - \int_{\partial \Omega} q \left(\frac{\partial \mathbf{u}^{n+1}}{\partial t} + \mathbf{N}(\mathbf{u})^{n+1} + \nu \nabla \times \nabla \times \mathbf{u}^{n+1} \right) \cdot \mathbf{n}$$
(11.3)

where $\partial\Omega$ is the boundary of the domain and we have used the factor that $\nabla \cdot (\nabla \times \nabla \times \mathbf{u}) = 0$. To get the final form of the weak pressure Poisson equation we can use a backward approximation of the time derivative to obtain

$$\frac{\partial \mathbf{u}^{n+1}}{\partial t} \simeq \frac{\gamma_0 \tilde{\mathbf{u}}^{n+1} - \hat{\mathbf{u}}}{\Delta t}$$
(11.4)

where $\tilde{\mathbf{u}}^{n+1}$ is an intermediate velocity upon which to decouple the system we impose that $\nabla \cdot \tilde{\mathbf{u}}^{n+1} = 0$ and

$$\gamma_0 = \begin{cases} 1, & \text{if } J = 1 \\ \frac{3}{2}, & \text{if } J = 2 \end{cases} \quad \hat{\mathbf{u}} = \begin{cases} u^n, & \text{if } J = 1 \\ 2u^n - \frac{1}{2}u^{n-1}, & \text{if } J = 2. \end{cases}$$

Finally we introduce a consistent extrapolation for the non-linear terms and the curl of vorticity terms of the form:

$$\mathbf{N}^{*,n+1} = \begin{cases} \mathbf{N}^n, & \text{if } J = 1\\ 2\mathbf{N}^n - \mathbf{N}^{n-1}, & \text{if } J = 2 \end{cases}$$

A similar extrapolation can be used on the curl-curl term to end up with the final weak pressure approximation

$$\int_{\Omega} \nabla q \cdot \nabla p^{n+1} = \int_{\Omega} q \,\nabla \cdot \left(-\frac{\hat{\mathbf{u}}}{\Delta t} + \mathbf{N}(\mathbf{u})^{*,n+1} \right) - \int_{\partial\Omega} q \left(\frac{\partial \mathbf{u}}{\partial t}^{n+1} + \mathbf{N}(\mathbf{u})^{*,n+1} + \nu (\nabla \times \nabla \times \mathbf{u})^{*,n+1} \right) \cdot \mathbf{n} \quad (11.5)$$

We note this can be recast into an equivalent strong form of the pressure Poisson equation of the form

$$\nabla^2 p^{n+1} = \nabla \cdot \left(\frac{\hat{\mathbf{u}}}{\Delta t} - \mathbf{N}^{*,n+1}\right)$$
(11.6)

with consistent Neumann boundary conditions prescribed as

$$\frac{\partial p}{\partial n}^{n+1} = -\left[\frac{\partial \mathbf{u}}{\partial t}^{n+1} + \nu(\nabla \times \nabla \times \mathbf{u})^{*,n+1} + \mathbf{N}^{*,n+1}\right] \cdot \mathbf{n}$$
(11.7)

2. The second step is discretise equation (11.1a) at time level n + 1, use the pressure at n + 1 from the first step and solve for the velocity \mathbf{u}^{n+1} .

In this step now approximate the time derivative using

$$\frac{\partial \mathbf{u}^{n+1}}{\partial t} \simeq \frac{\gamma_0 \mathbf{u}^{n+1} - \hat{\mathbf{u}}}{\Delta t}$$
(11.8)

which leads us to the Helmholtz problem

$$\left(\Delta - \frac{\gamma_0}{\nu \Delta t}\right) \mathbf{u}^{n+1} = -\left(\frac{\gamma_0}{\nu \Delta t}\right) \hat{\mathbf{u}} + \frac{1}{\nu} \nabla p^{n+1}$$
(11.9)

This scheme is activated in the SolverInfo section with the [SolverType] specification:

```
1 <I PROPERTY="SolverType" VALUE="VelocityCorrectionScheme"/>
```

11.1.1.2 Velocity Correction Scheme with a Weak Pressure formulation

As presented in the previous section in the work of Guermond and Shen [17] and subsequent work they formulate the pressure in a weak rather than strong form to obtain the pressure Poisson system. Therefore if we take the inner product of equation (11.1a) with respect to the gradient of the test space, ∇q , we obtain equation (11.2)

We again make the approximation

$$\frac{\partial \mathbf{u}^{n+1}}{\partial t} \simeq \frac{\gamma_0 \tilde{\mathbf{u}}^{n+1} - \hat{\mathbf{u}}}{\Delta t}.$$
(11.10)

However this time we only integrate by parts the last term and do not integrate the non-linear term by parts. However we still need to enforce the condition that $\nabla \cdot \tilde{\mathbf{u}}^{n+1} = 0$

and so we also integrate just this part of the time derivate by parts to arrive at a weak pressure system of the form:

$$\int_{\Omega} \nabla q \cdot \nabla p^{n+1} + \frac{\gamma_0}{\Delta t} \int_{\partial \Omega_0} q \tilde{\mathbf{u}}^{n+1} \cdot \mathbf{n} = \int_{\Omega} \nabla q \cdot (\frac{\hat{u}}{\Delta t} - \mathbf{N}^{\star, n+1}) \\ - \int_{\partial \Omega_d \bigcup \partial \Omega_0} q \, \nu (\nabla \times \nabla \times \mathbf{u})^{\star, n+1} \cdot \mathbf{n} + \frac{\gamma_0}{\Delta t} \int_{\partial \Omega_d} q \mathbf{w}^{n+1} \cdot \mathbf{n} \quad (11.11)$$

where $\partial \Omega_d$ is the Dirichlet boundary conditions for the velocity and $\partial \Omega_0$ is the outflow boundary.

This scheme is activated in the SolverInfo section with the SolverType specification:

```
1 <I PROPERTY="SolverType" VALUE="VCSWeakPressure"/>
```

11.1.1.3 Specifying pressure boundary conditions

In order to specify the pressure boundary conditions given by equation (11.7) or for the equivalent conditions in the VCSWeakPressure scheme the USERDEFINEDTYPE condition "H" can be used. Therefore a zero velocity wall boundary condition on boundary region 0 in two-dimensions can be specified as

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="u" VALUE="0" />
4 <D VAR="v" VALUE="0" />
5 <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
6 </REGION>
7 </BOUNDARYCONDITIONS>
```

11.1.1.4 Outflow boundary conditions

The most straightforward outflow condition is to specify fully developed conditions of $\nabla \mathbf{u}^{n+1} \cdot \mathbf{n} = 0$ and p = 0 which can be specified as

```
      1
      <BOUNDARYCONDITIONS>

      2
      <REGION REF="0">

      3
      <N VAR="u" VALUE="0" />

      4
      <N VAR="v" VALUE="0" />

      5
      <D VAR="p" VALUE="0" />

      6
      </REGION>

      7
      </BOUNDARYCONDITIONS>
```

However when energetic vortices pass through an outflow region one can experience instabilities as identified by the work of Dong, Karnidakis and Chryssostomidis [9]. In this paper they suggest to impose a pressure Dirichlet outflow condition of the form

$$p^{n+1} = \nu \mathbf{n} \cdot \nabla \mathbf{u}^{*,n+1} \cdot \mathbf{n} - \frac{1}{2} | \mathbf{u}^{*,n+1} |^2 S_o(\mathbf{n} \cdot \mathbf{u}^{*,n+1}) + \mathbf{f}_b^{n+1} \cdot \mathbf{n}$$
(11.12)

with a step function defined by $S_o(n \cdot \mathbf{u}) = \frac{1}{2}(1 - \tanh \frac{n \cdot \mathbf{u}}{\mathbf{u}_0 \delta})$, where \mathbf{u}_0 is the characteristic velocity scale and δ is a non-dimensional positive constant chosen to be sufficiently small. \mathbf{f}_b is the forcing term in this case the analytical conditions can be given but if these are not known explicitly, it is set to zero, i.e. $\mathbf{f}_b = 0$. (see the test KovaFlow_m8_short_HOBC.xml for a non-zero example). Note that in the paper [9] they define this term as the negative of what is shown here so that it could be use used to impose a default pressure values. This does however mean that the forcing term is imposed through the velocity components u, v by specifying the entry (VALUE) (An example can be found in ChanFlow_m3_VCSWeakPress_ConOBC.xml). For the velocity component one can specify

$$\nabla \mathbf{u}^{n+1} \cdot \mathbf{n} = \frac{1}{\nu} \Big[p^{n+1} \mathbf{n} + \frac{1}{2} \mid \mathbf{u}^{*,n+1} \mid^2 S_o(\mathbf{n} \cdot \mathbf{u}^{*,n+1}) - \nu (\nabla \cdot \mathbf{u}^{*,n+1}) \mathbf{n} - \mathbf{f}_b^{n+1} \Big] \quad (11.13)$$

This condition can be enforced using the (USERDEFINEDTYPE) "HOutflow", i.e.

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <REGION REF="0">
4 <N VAR="u" USERDEFINEDTYPE="HOutflow" VALUE="0" />
5 <N VAR="v" USERDEFINEDTYPE="HOutflow" VALUE="0" />
5 <N VAR="p" USERDEFINEDTYPE="HOutflow" VALUE="0" />
6 </REGION>
7 </BOUNDARYCONDITIONS>
```

Note that in the moving body work of Bao et al. [4] some care must be made to identify when the flow over the boundary is incoming or outgoing and so a modification of the term

$$\frac{1}{2}\mid \mathbf{u}^{*,n+1}\mid^2 S_o(\mathbf{n}\cdot\mathbf{u}^{*,n+1})$$

is replaced with

$$\frac{1}{2}\left((\theta+\alpha_2)\mid \mathbf{u}^{*,n+1}\mid^2 + (1-\theta+\alpha_1)(\mathbf{u}^{*,n+1}\cdot\mathbf{n})\mathbf{u}^{*,n+1}\right)S_o(\mathbf{n}\cdot\mathbf{u}^{*,n+1})$$

where the default values are given by $\theta = 1, \alpha_1 = 0, \alpha_2 = 0$ and these values can be set through the parameters OutflowBC_theta, OutflowBC_alpha1 and OutflowBC_alpha2.

Dong has also suggested convective like outflow conditions in [8] which can be enforced through a Robin type specification of the form

$$\frac{\partial \mathbf{u}^{n+1}}{\partial n} + \frac{\gamma_0 D_0}{\Delta t} \mathbf{u}^{n+1} = \frac{1}{\nu} \Big[\mathbf{f}^{n+1} + \mathbf{E}(\mathbf{n}, \mathbf{u}^{*, n+1}) + p^{n+1} \mathbf{n} - \nu (\nabla \cdot \mathbf{u}^{*, n+1}) \mathbf{n} \Big] + \frac{D_0}{\Delta t} \hat{\mathbf{u}} \quad (11.14)$$

$$\frac{\partial p^{n+1}}{\partial n} + \frac{1}{\nu D_0} p^{n+1} = -\left(-\nu (\nabla \times \nabla \times \mathbf{u})^{*,n+1} + \mathbf{N}^{*,n+1}\right) \cdot \mathbf{n}$$
$$-\frac{1}{\nu D_0} \left[\mathbf{f}^{n+1} + \mathbf{E}(\mathbf{n}, \mathbf{u}^{*,n+1} + p^{n+1}\mathbf{n} - \nu (\nabla \cdot \mathbf{u}^{*,n+1})\mathbf{n}\right] \quad (11.15)$$

1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <R VAR="u" USERDEFINEDTYPE="HOutflow" VALUE="0" PRIMCOEFF="D0/TimeStep"/>
4 <R VAR="v" USERDEFINEDTYPE="HOutflow" VALUE="0" PRIMCOEFF="D0/TimeStep"/>
5 <R VAR="p" USERDEFINEDTYPE="HOutflow" VALUE="0" PRIMCOEFF="1.0/(D0*Kinvis)"/>
6 </REGION>
7 </BOUNDARYCONDITIONS>

11.1.1.5 Substepping/subcycling the Velocity Correction Scheme

It is possible to use different time steps in the velocity correction scheme using a substepping (also known as subcycling) [41] or auxiliary semi-Lagrangian approach [49]. Originally the scheme was proposed by Maday, Patera and Ronquist who referred to as an operator-integration-factor splitting method [28]



Figure 11.1 Schematic representation of the substepping approach. (a) Making an explicit time step the hyperbolic solution, travelling with a speed a, can be understood as being related to the solution at point x_d (the departure point). (b) Making smaller explicit time steps we can evaluate the solution $\phi(x_d)$ at the departure point and then use this value to make a semi-Lagrangian discretisation of the implicit components usually associated with diffusion.

A schematic of the approach can be understood from figure 11.1.1.5 where we observe that smaller time steps can be used for the explicit advection steps whilst a larger overall time step is adopted for the more expensive implicit solve for the diffusion operator. More details of the implementation can be found in [49] and [41]. In the following sections we outline the parameters that can be used to set up this scheme. Since the explicit part is advanced using a DG scheme it is necessary to use a Mixed_CG_Discontinuous expansion with this option. Note



Some examples of the substepping scheme can be found in the regression tests directory under \$NEKHOME/Solver/IncNavierStokesSolver/Tests/ directory: KovaFlow_SubStep_2order.xml, Hex_Kovasnay_SubStep.xml and Tet_Kovasnay_SubStep.xml.

11.1.1.6 Approximation spaces for the velocity correction scheme

For well resolved simulations it appears that often using the same polynomial space for the pressure and velocity does give suitable answer but this does not satisfy the so-called LBB or inf-sup condition. Therefore, it is potentially better to specify an equivalent of the Taylor Hood approximation and use one higher polynomial order for velocity than the pressure with a continuous expansion. To specify this type of expansion you can use an expansion section of the form:

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="8" FIELDS="u,v" TYPE="MODIFIED" />
3 <E COMPOSITE="C[0]" NUMMODES="7" FIELDS="p" TYPE="MODIFIEDQUADPLUS1" />
4 </EXPANSIONS>
```

In the above example the "u,v" fields are specified to have a polynomial order of 7 using a modified expansion. Implicitly this form of the expansion definition uses a quadrature order of 9. The above definition then also uses a modified expansion for pressure but of polynomial order 6. Since currently for this solver to run we need to use a consistent quadrature order for both the velocity and pressure fields we specify the **MODIFIEDQUADPLUS1** to tell the solver to use an additional quadrature point and therefore also use 9 quadrature points in each 1D direction for the pressure.

In other cases it is sometimes useful to run with an even higher quadrature order, for example to handle highly deformed elements where the Jacobian is represented by a polynomial expansion. This can be done by using a more detailed definition of the expansion of the form:

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" BASISTYPE="Modified_A,Modified_B" NUMMODES="8,8"
2 POINTSTYPE="GaussLobattoLegendre,GaussRadauMAlpha1Beta0" NUMPOINTS="9,8"
3 FIELDS="u,v" />
3 <E COMPOSITE="C[0]" BASISTYPE="Modified_A,Modified_B" NUMMODES="7,7"
POINTSTYPE="GaussLobattoLegendre,GaussRadauMAlpha1Beta0" NUMPOINTS="9,8"
FIELDS="p" />
4 </EXPANSIONS>
```

In this example we have specified an 8th order expansion for "u,v" and a 7th order expansion for "p". The BasisType is given as "Modified_A, Modified_B" which is for a triangular expansion (note that for a quadrilateral expansion it would have been "Modified_A,Modified_A") and so the number of quadrature points in this case is 9 in the first direction which uses Gauss-Lobatto-Legendre points but only 8 in the second

direction since this uses a Gauss-Radau formula with $\alpha = 1, \beta = 0$ weights (see [21] for details on why).

11.1.2 Immersed Boundary Methods: Smoothed Profile Method

The usual way to solve any PDE requires the definition of a well defined domain where the solution is to be determined. Thus, for complex geometries, the meshing process may get cumbersome and, in any case, the solver will have to the meshing process may get cumbersome, and likely struggle to avoid the presence of highly deformed and skewed elements. In addition, when solving cases with moving boundaries, the mesh has to be updated every time step to follow the shape of the boundaries, leading to a very resource and time-consuming simulation that limits the capabilities of the solver.

Immersed Boundary Methods may be very useful in these situations, where the definition of the boundaries requires a very complex mesh to reach convergence. The main idea behind them is the use of a forcing term in the incompressible Navier-Stokes equations in such a way that the mesh does not necessarily follow the boundaries. The solution in the regions falling outside the boundaries is simply that of the boundaries, forcing the flow to behave as if there were a real object even if the mesh does not represent it. The method presented here is an adaptation of the Smoothed Profile Method [35] extended to a high-order semi-implicit splitting scheme [26, 47]. This method ensures that the no-slip, no-penetration and incompressibility constraints are mathematically enforced. Starting from the incompressible Navier-Stokes equations, the term $\mathbf{f_s}$ is added to the right hand side:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} + \mathbf{f_s}$$
(11.16a)

$$\nabla \cdot \mathbf{u} = 0 \tag{11.16b}$$

The definition of this term depends on the method but, for the Smoothed Profile Method (SPM), it is related to a *shape function* $\Phi(\mathbf{x}, t)$ valued 0 in the fluid domain and 1 outside. It is usually defined as:

$$\Phi(\mathbf{x},t) = -\frac{1}{2} \left[\tanh\left(\frac{d(\mathbf{x},t)}{\xi}\right) - 1 \right], \qquad (11.17)$$

being ξ a scaling factor [47] and $d(\mathbf{x}, t)$ a function representing the distance to the boundary (positive inside the body, negative inside the fluid). If the case to be simulated includes more than one immersed boundaries, the final shape function is calculated by adding the individual ones as long as they do not overlap:

$$\Phi = \sum_{i} \Phi_i \tag{11.18}$$



Figure 11.2 Definition of the shape function Φ close to the boundary of an immersed body.

The approach followed during the implementation in Nektar++ is an extension of the Velocity Correction Scheme, using the final velocity obtained with this method as an intermediate velocity to determine the value of the forcing term. The initial equation (11.16a) is slightly modified and integrated in time by means of a *stiffly-stable* scheme and, then, split into different smaller parts that are solved separately:

$$\frac{\gamma_{0}\mathbf{u}^{n+1} - \sum_{q=0}^{J-1} \alpha_{q} \mathbf{u}^{n-q}}{\Delta t} = -\sum_{q=0}^{J-1} \beta_{q} (\mathbf{u} \cdot \nabla \mathbf{u})^{n-q} - \nabla (p^{*} + p_{p})^{n+1} + \nu \nabla^{2} \mathbf{u}^{n+1} + \mathbf{f}_{s}^{n+1} + \mathbf{f}_{s}^{n+1}$$
(11.19)

$$\frac{\tilde{\mathbf{u}} - \sum_{q=0}^{J-1} \alpha_q \, \mathbf{u}^{n-q}}{\Delta t} = -\sum_{q=0}^{J-1} \beta_q \, (\mathbf{u} \cdot \nabla \mathbf{u})^{n-q} + \mathbf{f}^{n+1}, \qquad (11.20a)$$

$$\frac{\hat{\mathbf{u}} - \tilde{\mathbf{u}}}{\Delta t} = -\nabla p^{*n+1},\tag{11.20b}$$

$$\frac{\gamma_0 \mathbf{u}^* - \hat{\mathbf{u}}}{\Delta t} = \nu \nabla^2 \mathbf{u}^{n+1}, \qquad (11.20c)$$

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \gamma_0 \mathbf{u}^*}{\Delta t} = -\nabla p_p^{n+1} + \mathbf{f_s}^{n+1}$$
(11.20d)

The new term $\mathbf{f_s}$ is defined as follows:

$$\mathbf{f_s}^{n+1} = \frac{\gamma_0 \Phi^{n+1} (\mathbf{u_p}^{n+1} - \mathbf{u}^*)}{\Delta t},$$
 (11.21)

where α_q , β_q and γ_0 are coefficients of the stiffly-stable time integration method and $\mathbf{u_p}$ is the velocity of the points that lay outside the boundaries. Thus, the new term is just an acceleration proportional to the difference between the expected and the intermediate velocity, forcing the flow to follow the shapes defined by Φ and $\mathbf{u_p}$. Some transformations in these expressions lead to the final SPM equations:

1. Advection and external forces:

$$\frac{\tilde{\mathbf{u}} - \sum_{q=0}^{J-1} \alpha_q \, \mathbf{u}^{n-q}}{\Delta t} = \sum_{q=0}^{J-1} \beta_q \left[-\mathbf{u} \cdot \nabla \mathbf{u} + \mathbf{f} \right]^{n-q}$$
(11.22)

2. Pressure:

$$\nabla^2 p^* = \nabla \cdot \left(\frac{\tilde{\mathbf{u}}}{\Delta t}\right),\tag{11.23a}$$

$$\frac{\partial p}{\partial \mathbf{n}}^* = -\sum_{q=0}^{J-1} \beta_q \left[-\mathbf{u} \cdot \nabla \mathbf{u} + \nu (\nabla \times \nabla \times \mathbf{u}) - \mathbf{f} \right]^{n-q} \cdot \mathbf{n}$$
(11.23b)

3. Viscous effects:

$$\left(\nabla^2 - \frac{\gamma_0}{\nu\Delta t}\right)\mathbf{u}^* = \frac{\mathbf{\hat{u}}}{\nu\Delta t} \tag{11.24}$$

4. SPM pressure:

$$\nabla^2 p_p = \nabla \cdot \frac{\gamma_0 \Phi^{n+1} (\mathbf{u_p}^{n+1} - \mathbf{u}^*)}{\Delta t}, \qquad (11.25a)$$

$$\frac{\partial p_p}{\partial \mathbf{n}} = \frac{\gamma_0 \Phi^{n+1}(\mathbf{u_p}^{n+1} - \mathbf{u}^*)}{\Delta t} \cdot \mathbf{n}$$
(11.25b)

5. SPM force:

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \gamma_0 \mathbf{u}^*}{\Delta t} = \frac{\gamma_0 \Phi^{n+1} (\mathbf{u_p}^{n+1} - \mathbf{u}^*)}{\Delta t} - \nabla p_p \tag{11.26}$$

Since the term ∇p_p in the last equation may induce a velocity slightly different to $\mathbf{u_p}$ inside the bodies, it may be changed to $(1 - \Phi)\nabla p_p$, cancelling this effect but adding some compressibility to the flow [26]:

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \gamma_0 \mathbf{u}^*}{\Delta t} = \frac{\gamma_0 \Phi^{n+1} (\mathbf{u_p}^{n+1} - \mathbf{u}^*)}{\Delta t} - (1 - \Phi) \nabla p_p \tag{11.27}$$

11.1.2.1 Input files

The session file follows the same rules as any other incompressible Navies-Stokes solver. However, there are some additional parameters that must be supplied when using this approach. First, the property SolverType must be set to SmoothedProfileMethod, while the immersed boundaries are defined in a function called ShapeFunction. Besides, the property ForceBoundary can be set to True if equation (11.27) is preferred over (11.26). Thus, the (TIMEINTEGRATIONSCHEME) and (SOLVERINFO) section could be similar to:

```
1 <TIMEINTEGRATIONSCHEME>
2
      <METHOD> IMEX </METHOD>
3
      <ORDER> 3 </ORDER>
4 </TIMEINTEGRATIONSCHEME>
5
6 <SOLVERINFO>
     <I PROPERTY="SolverType"
                                           VALUE="SmoothedProfileMethod" />
\overline{7}
8
     <I PROPERTY="ForceBoundary"
9
                                            VALUE="True"
                                                                            />
10 </SOLVERINFO>
```

In addition, somewhere in the **CONDITIONS** section this function must appear:

As a brief guideline, to define a cylinder of radius 1 and center at the point (0,0) according to expression (11.17), the "Phi" field in the ShapeFunction function should be:

```
1 <E VAR="Phi" USERDEFINEDTYPE="TimeDependent" VALUE="-0.5*(tanh((rad(x,y)-1.0)
/0.04)-1.0)" />
```

where the scaling coefficient has been set to 0.04. The variable names are compulsory, being Phi the shape of the bodies, and Up, Vp and Wp functions representing the velocity field inside them. The attribute USERDEFINEDTYPE is compulsory only if the functions depend on time, when it must be set to "TimeDependent".

For immersed boundaries with geometries that cannot be represented by means of analytical functions, an .stl binary file can be supplied as well. However, the geometry file must be first converted to .fld format with the phifile module of FieldConvert. The simplest way to proceed is by issuing the command:

FieldConvert -m phifile:file=geom.stl:scale=value session.xml geom.fld

where the value of scale corresponds to the coefficient ξ in equation (11.17). More details can be found in section 5.6.37. In any case, it is important to remark that this functionality is still under development.

In the ShapeFunction block of the session file, the line <E VAR="Phi" ... /> indicates that the immersed bodies are defined by the function introduced in VALUE, while a line like the following:

1 <F VAR="Phi" FILE="geometry.fld" />

must be used when the Φ field is defined in an .stl file previously converted to .fld format. In this case, the solver only supports non-moving geometries and the attribute USERDEFINEDTYPE="TimeDependent", if specified, will not be used.

11.1.3 Direct solver (coupled approach)

The second approach consists of directly solving the matrix problem arising from the discretization of the Stokes problem. The direct solution of the Stokes system introduces the problem of appropriate spaces for the velocity and the pressure systems to satisfy the inf-sup condition and it requires the solution of the full velocity-pressure system. However, if a discontinuous pressure space is used then all but the constant mode of the pressure system can be decoupled from the velocity. When implementing this approach with a spectral/hp element discretization, the remaining velocity system may then also be statically condensed to decouple the so called interior elemental degrees of freedom, reducing the Stokes problem to a smaller system expressed on the elemental boundaries. The direct solution of the Stokes problem provides a very natural setting for the solution of the pressure system which is not easily dealt with in a splitting scheme. Further, the solution of the full coupled velocity system allows the introduction of a spatially varying viscosity, which arise for non-Newtonian flows, with only minor modifications.

Note

The coupled solver is only supported for two-dimensional or quasi-3D problems, and only using a direct solver (e.g. DirectStaticCond) which prevents its use in parallel.

We consider the weak form of the Stokes problem for the velocity field $\boldsymbol{u} = [u, v]^T$ and the pressure field p:

$$(\nabla\phi, \nu\nabla \boldsymbol{u}) - (\nabla \cdot \phi, p) = (\phi, \boldsymbol{f})$$
(11.28a)

$$(q, \nabla \cdot \boldsymbol{u}) = 0 \tag{11.28b}$$

where the components of A, B and C are $\nabla \phi_b, \nu \nabla u_b, \nabla \phi_b, \nu \nabla u_i$ and $\nabla \phi_i, \nu \nabla u_i$ and the components D_b and D_i are $q, \nabla u_b$ and $q, \nabla u_i$. The indices b and i refer to the degrees of freedom on the elemental boundary and interior respectively. In constructing the system we have lumped the contributions form each component of the velocity field into matrices A, B and C. However, we note that for a Newtonian fluid the contribution from each field is decoupled. Since the interior degrees of freedom of the velocity field do not overlap, the matrix C is block diagonal and to take advantage of this structure we can statically condense out the C matrix to obtain the system:

$$\begin{bmatrix} A - BC^{-1}B^T & D_b^T - BC^{-1}D_i & 0\\ D_b - D_i^T C^{-1}B^T & -D_i^T C^{-1}D_i & 0\\ B^T & D_i & C \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_b \\ p \\ \boldsymbol{u}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_b - BC^{-1}\boldsymbol{f}_i \\ -D_i^T C^{-1}\boldsymbol{f}_i \\ \boldsymbol{f}_i \end{bmatrix}$$
(11.29)

To extend the above Stokes solver to an unsteady Navier-Stokes solver we first introduce the unsteady term, $\partial \boldsymbol{u}/\partial t$, into the Stokes problem. This has the principal effect of modifying the weak Laplacian operator $\nabla \phi, \nu \nabla \boldsymbol{u}$] into a weak Helmholtz operator $\nabla \phi, \nu \nabla \boldsymbol{u}) - \lambda(\phi, \boldsymbol{u})$ where λ depends on the time integration scheme. The second modification requires the explicit discretisation of the non-linear terms in a similar manner to the splitting scheme and this term is then introduced as the forcing term \boldsymbol{f} . For more details see [1, 42].

11.1.4 Linear Stability Analysis

Hydrodynamic stability is an important part of fluid-mechanics that has a relevant role in understanding how an unstable flow can evolve into a turbulent state of motion with chaotic three-dimensional vorticity fields and a broad spectrum of small temporal and spatial scales. The essential problems of hydrodynamic stability were recognised and formulated in 19th century, notably by Helmholtz, Kelvin, Rayleigh and Reynolds.

Conventional linear stability assumes a normal representation of the perturbation fields that can be represented as independent wave packets, meaning that the system is selfadjoint. The main aim of the global stability analysis is to evaluate the amplitude of the eigenmodes as time grows and tends to infinity. However, in most industrial applications, it is also interesting to study the behaviour at intermediate states that might affects significantly the functionality and performance of a device. The study of the transient evolution of the perturbations is seen to be strictly related to the non-normality of the linearised Navier-Stokes equations, therefore the normality assumption of the system is no longer assumed. The eigenmodes of a non-normal system do not evolve independently and their interaction is responsible for a non-negligible transient growth of the energy. Conventional stability analysis generally does not capture this behaviour, therefore other techniques should be used.

A popular approach to study the hydrodynamic stability of flows consists in performing a direct numerical simulation of the linearised Navier-Stokes equations using iterative methods for computing the solution of the associated eigenproblem. However, since linearly stable flows could show a transient increment of energy, it is necessary to extend this analysis considering the combined effect of the direct and adjoint evolution operators. This phenomenon has noteworthy importance in several engineering applications and it is known as transient growth.

In Nektar++ it is then possible to use the following tools to perform stability analysis:

• direct stability analysis;

- adjoint stability analysis;
- transient growth analysis;

11.1.4.1 Direct stability analysis

The equations that describe the evolution of an infinitesimal disturbance in the flow can be derived decomposing the solution into a basic state (\mathbf{U}, p) and a perturbed state $\mathbf{U} + \varepsilon \mathbf{u}'$ with $\varepsilon \ll 1$ that both satisfy the Navier-Stokes equations. Substituting into the Navier-Stokes equations and considering that the quadratic terms $\mathbf{u}' \cdot \nabla \mathbf{u}'$ can be neglected, we obtain the linearised Navier-Stokes equations:

$$\frac{\partial \mathbf{u}'}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{u}' + \mathbf{u}' \cdot \nabla \mathbf{U} = -\nabla p + \nu \nabla^2 \mathbf{u}' + \mathbf{f}$$
(11.30a)

$$\nabla \cdot \mathbf{u}' = 0 \tag{11.30b}$$

The linearised Navier-Stokes equations are identical in form to the non-linear equation, except for the non-linear advection term. Therefore the numerical techniques used for solving Navier-Stokes equations can still be applied as long as the non-linear term is substituted with the linearised one. It is possible to define the linear operator that evolved the perturbation forward in time:

$$\mathbf{u}'(\mathbf{x},t) = \mathcal{A}(\mathbf{U})\mathbf{u}'(\mathbf{x},0) \tag{11.31}$$

Let us assume that the base flow \mathbf{U} is steady, then the perturbations are autonomous and we can assume that:

$$\mathbf{u}'(\mathbf{x},t) = \mathbf{q}'(\mathbf{x}) \exp(\lambda t) \quad \text{where } \lambda = \sigma + i\omega$$
 (11.32)

Then we obtain the associated eigenproblem:

$$\mathcal{A}(\mathbf{U})\mathbf{q}' = \lambda \mathbf{q}' \tag{11.33}$$

The dominant eigenvalue determines the behaviour of the flow. If the real part is positive then there exist exponentially growing solutions. Conversely, if all the eigenvalues have negative real part then the flow is linearly stable. If the real part of the eigenvalue is zero, it is a bifurcation point.

11.1.4.2 Adjoint Stability Analysis

The adjoint of a linear operator is one of the most important concepts in functional analysis and it plays an important role in understanding transition to turbulence. Let us write the linearised Navier-Stokes equation in a compact form:

$$\mathcal{H}\mathbf{q} = 0 \quad \text{where} \quad \mathcal{H} = \left(\frac{-\partial_t - (\mathbf{U} \cdot \nabla) + (\nabla \mathbf{U}) \cdot + \frac{1}{Re} \nabla^2 \mid -\nabla}{\nabla \cdot \mid 0} \right)$$
(11.34)

The adjoint operator \mathcal{H}^* is defined as:

$$\langle \mathcal{H}\mathbf{q}, \mathbf{q} \rangle = \langle \mathbf{q}, \mathcal{H}^* \mathbf{q}^* \rangle \tag{11.35}$$

Integrating by parts and employing the divergence theorem, it is possible to express the adjoint equations:

$$-\frac{\partial \mathbf{u}^*}{\partial t} + (\mathbf{U} \cdot \nabla)\mathbf{u}^* + (\nabla \mathbf{U})^T \cdot \mathbf{u}^* = -\nabla p^* + \frac{1}{Re}\nabla^2 \mathbf{u}$$
(11.36a)

$$\nabla \cdot \mathbf{u}^* = 0 \tag{11.36b}$$

The adjoint fields are in fact related to the concept of **receptivity**. The value of the adjoint velocity at a point in the flow indicates the response that arises from an unsteady momentum source at that point. The adjoint pressure and the adjoint stream function play instead the same role for mass and vorticity sources respectively. Therefore, the adjoint modes can be seen as a powerful tool to understand where to act in order to ease/inhibit the transition.

11.1.4.3 Transient Growth Analysis

Transient growth is a phenomenon that occurs when a flow that is linearly stable, but whose perturbations exhibit a non-negligible transient response due to regions of localised convective instabilities. This situation is common in many engineering applications, for example in open flows where the geometry is complex, producing a steep variation of the base flow. Therefore, the main question to answer is if it exists a bounded solution that exhibit large growth before inevitably decaying. Let us introduce a norm to quantify the size of a perturbation. It is physically meaningful to use the total kinetic energy of a perturbation on the domain Ω . This is convenient because it is directly associated with the standard-L2 inner product:

$$\mathcal{A}(\tau)\mathbf{v} = \sigma \mathbf{u}, \quad \|\mathbf{u}\| = 1 \tag{11.37}$$

where $\sigma = \|\mathbf{u}'(\tau)\|$. This is no other than the singular value decomposition of $\mathcal{A}(\tau)$. The phenomenology of the transient growth can be explained considering the non-normality of the linearised Navier-Stokes evolution operator. This can be simply understood using the simple geometric example showed in figure 11.1.4.3. Let us assume a unit-length vector \mathbf{f} represented in a non-orthogonal basis .This vector is defined as the difference of the nearly collinear vectors Φ_1 and Φ_2 . With the time progression, the component of these two vectors decrease respectively by 20% and 50%. The vector \mathbf{f} increases substantially in length before decaying to zero. Thus, the superposition of decaying non-orthogonal eigenmode can produce in short term a growth in the norm of the perturbations.



Figure 11.3 Geometric interpretation of the transient growth. Adapted from Schmid, 2007

11.1.5 Steady-state solver using Selective Frequency Damping

To compute linear stability analysis, the choice of the base flow, around which the system will be linearised, is crucial. If one wants to use the steady-state solution of the Navier-Stokes equations as base flow, a steady-state solver is implemented in Nektar++. The method used is the encapsulated formulation of the Selective Frequency Damping method [19]. Unstable steady base flows can be obtained with this method. The SFD method is based on the filtering and control of unstable temporal frequencies within the flow. The time continuous formulation of the SFD method is

$$\begin{cases} \dot{q} = NS(q) - \chi(q - \bar{q}), \\ \dot{\bar{q}} = \frac{q - \bar{q}}{\Delta}. \end{cases}$$
(11.38)

where q represents the problem unknown(s), the dot represents the time derivative, NS represents the Navier-Stokes equations, $\chi \in \mathbb{R}_+$ is the control coefficient, \bar{q} is a filtered version of q, and $\Delta \in \mathbb{R}_+^*$ is the filter width of a first-order low-pass time filter. The steady-state solution is reached when $q = \bar{q}$.

The convergence of the method towards a steady-state solution depends on the choice of the parameters χ and Δ . They have to be carefully chosen: if they are too small, the instabilities within the flow can not be damped; but if they are too large, the method may converge extremely slowly. If the dominant eigenvalue of the flow studied is known (and given as input), the algorithm implemented can automatically select parameters that ensure a fast convergence of the SFD method. Most of the time, the dominant eigenvalue is not know, that is why an adaptive algorithm that adapts χ and Δ all along the solver execution is also implemented.

Note that this method can not be applied for flows with a pure exponential growth of the instabilities (*e.g.* jet flow within a pipe). In other words, if the frequency of the dominant eigenvalue is zero, then the SFD method is not a suitable tool to obtain a steady-state solution.

11.2 Usage

```
IncNavierStokesSolver session.xml
```

11.3 Session file configuration

In the following the possible options are shown for the incompressible Navier-Stokes. The Expansion section for an incompressible flow simulation can be set as for other solvers regardless of the projection type. Here an example for a 3D simulation (for 2D simulations the specified fields would be just (u,v,p)).

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="6" FIELDS="u,v,w,p" TYPE="MODIFIED" />
3 </EXPANSIONS>
```

In case of a simulation using the Direct Solver we need to set FIELDS=u, v as the pressure expansion order will be automatically set to fulfil the inf-sup condition. Possible choices for the expansion (TYPE) are:

Basis	TYPE
Modal	MODIFIED
Nodal	GLL_LAGRANGE
Nodal SEM	GLL_LAGRANGE_SEM

11.3.1 Solver Info

The following parameters can be specified in the (SOLVERINFO) section of the session file:

• EqType : sets the kind of equations we want to solve on the domain as:

```
1 <I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes"/>
```

Possible values are:

Equations	EQTYPE	Dim.	Projections	Alg.
Steady Stokes (SS)	SteadyStokes	All	CG	VCS
Steady Oseen (SO)	SteadyOseen	All	CG	DS
Unsteady Stokes (US)	UnsteadyStokes	All	CG	VCS
Steady Linearised NS (SLNS)	${\tt SteadyLinearisedNS}$	All	CG	DS
Unsteady Linearised NS (ULNS)	UnsteadyLinearisedNS	All	CG	VCS,DS
Unsteady NS (UNS)	UnsteadyNavierStokes	All	CG, CG-DG	VCS

• SolverType : sets the scheme we want to use to solve the set of equations as

1 <I PROPERTY="SolverType" VALUE="VelocityCorrectionScheme"/>

Possible values are:

Algorithm	SolverType	Dimensions	Projections
Velocity Correction Scheme (VCS)	VelocityCorrectionScheme	2D, Quasi-3D, 3D	CG, CG-DG
Smoothed Profile Method (SPM)	${\tt SmoothedProfileMethod}$	2D, Quasi-3D, 3D	CG, CG-DG
VCS with weak pressure	VCSWeakPressure	2D, Quasi-3D, 3D	CG, CG-DG
Direct solver	CoupledLinearisedNS	2D, Quasi-3D	CG

• **Driver**: this specifies the type of problem to be solved:

Driver	Description	Dimensions	Projections
Standard	Time integration of the equations	All	CG, DG
SteadyState	Steady-state solver (see Sec. 11.1.5)	All	CG, DG

• Projection: sets the Galerkin projection type as

1 <I PROPERTY="Projection" VALUE="Continuous"/>

Possible values are:

Galerkin Projection	Projection	Dimensions	Equations	Algorithms
Continuous (CG)	Continuous	All	All	All
Discontinuous (DG)	DisContinuous	All		
Mixed CG and DG (CG-DG)	Mixed_CG_Discontinuous	2D,3D	just UNS	VCS-substepping

• TimeIntegrationScheme : sets the time integration as

1 <TIMEINTEGRATIONSCHEME>

- 2 <METHOD> IMEX </METHOD>
- 3 <ORDER> 2 </ORDER>
- 4 </TIMEINTEGRATIONSCHEME>

Possible values are

Time-Integration Scheme	Method	Order	Dimensions	Equations	Projections
IMEX Order 1	IMEX	1	all	US, UNS	CG
IMEX Order 2	IMEX	2	all	US, UNS	CG
IMEX Order 3	IMEX	3	all	US, UNS	CG
Backward Euler	BackwardEuler	1	all	US, UNS	CG-DG
BDF Order 1	BDFImplicit	1	all	US, UNS	CG-DG
BDF Order 2	BDFImplicit	2	all	US, UNS	CG-DG

• Extrapolation: Specify the extrapolation method (standard or substepping) to be used in velocity correction scheme. Essentially this activates the sub-stepping routine which requires the mixed CG-DG projection

1 <I PROPERTY="Extrapolation" VALUE="SubStepping"/>

Possible values are **SubStepping** or **Standard** with "Standard" being the default value if nothing is specified.

• SubStepIntScheme: choose the substep DG time integration scheme so that a different order schems can be used as compared to the overal time integraiton scheme.

1 <I PROPERTY="SubStepIntScheme" VALUE="RungeKutta2_ImprovedEuler"/>

Possible values are

Time-Integration Scheme	SubStepIntScheme			
ForwardEuler, Order 1	ForwardEuler, Order 1			
RungeKutta, Order 2	RungeKutta, Variant SSP, Order 2			

This option is useful if you wish to use an overall scheme that is first order accurate for example with TimeIntegrationScheme as BDFImplicit Order 1 but using a second order RungeKutta, Variant SSP, Order 2 for greater stability in the substep.

• GlobalSysSoln: sets the approach we use to solve the the linear systems of the type Ax = b appearing in the solution steps, such as the Poisson equation for the pressure in the splitting-scheme. It can be set as

1 <I PROPERTY="GlobalSysSoln" VALUE="IterativeStaticCond"/>

Possible values are

System solution	GlobalSysSoln	Parallel
Direct Solver (DS)	DirectFull	just quasi-3D
DS with Static Condensation	DirectStaticCond	just Quasi-3D
DS with Multilevel Static Condensation	DirectMultiLevelStaticCond	just Quasi-3D
Iterative Solver (IS)	IterativeFull	just Quasi-3D
IS with Static Condensation	IterativeStaticCond	quasi-3D
IS with Multilevel Static Condensation	${\tt Iterative MultiLevel Static Cond}$	quasi-3D

Default values are (DirectMultiLevelStaticCond) in serial and (IterativeStaticCond) in parallel.

• **SmoothAdvection**: activates a stabilization technique which smooths the advection term using the pressure inverse mass matrix. It can be used just in combination with nodal expansion basis for efficiency reasons.

1 <I PROPERTY="SmoothAdvection" VALUE="True"/>

• SpectralVanishingViscosity: activates a stabilization technique which increases the viscosity on the modes with the highest frequencies.

```
1 <I PROPERTY="SpectralVanishingViscosity" VALUE="True"/>
```

In a Quasi-3D simulation, this will affect both the Fourier and the spectral/hp expansions. To activate them independently, use SpectralVanishingViscositySpectralHP and SpectralVanishingViscosityHomo1D.

There are three spectral vanishing viscosity kernels available:

SVV Kernel	SpectralVanishingViscosity
Exponential Kernel Power Kernel	True PowerKernel
DG Kernel	DGKernel

The Exponential kernel is based on the work of Maday et al. [29], its extension to 2D can be found in [22]. A diffusion coefficient can be specified which defines the base magnitude of the viscosity; this parameter is scaled by h/p. SVV viscosity is activated for expansion modes greater than the product of the cut-off ratio and the expansion order. The Power kernel is a smooth function with no cut-off frequency; it focusses on a narrower band of higher expansion modes as the polynomial order increases. The cut-off ratio parameter for the Power kernel corresponds to the power ratio, see Moura et al. [31]. The DG-Kernel is an attempt to match the dissipation of CG-SVV to DG schemes of lower expansion orders. This kernel does not require any parameters although the diffusion coefficient can still be modified.

• GJPStabilisation: activates the gradient jump penalty[32] stabilization technique. The GJP is less dissipative than the spectral vanishing viscousity (SVV) stabilisation and gives a better turbulent structures specially when the simulaiton is underresolved or in other words when the polynomial order is low. It is expected that with increasing the polynomial order and approaching towards a resolved solution both SVV and GJP stabilisation solutions approaches each other. Since the GJP is less dissipative than the SVV, it means that it probably requires a smaller Δt for the stable solution. Further, GJP only affects the spectral/hp discretisations and doesn't have any stabilisation effect on the Fourier discritisation. This means that in the Quasi-3D simulations, if GJP is activated we still needs to use the SpectralVanishingViscousityHomo1D to stablise the solution in the spanwise direction which is discretised using the Fourier method. For the 2D and full 3D problems, GJP stabilisation should suffices though.

1 <I PROPERTY="GJPStabilisation" VALUE="SemiImplicit"/>

Another option for the GJP stabilisation is to use Explicit for its value instead of SemiImplicit. If the Explicit value is set, there is an additional option that can be set for the explicit GJP stabilisation in the solver info as follows.

1 <I PROPERTY="GJPNormalVelocity" VALUE="True"/>

Using the GJPNormalVelocity, the GJP formulation will uses the average velocity normal to the edge/face as the velocity scaling of jump term.

• **DEALIASING**: activates the 3/2 padding rule on the advection term of a Quasi-3D simulation.

```
1 <I PROPERTY="DEALIASING" VALUE="True"/>
```

• SPECTRALHPDEALIASING: activates the spectral/hp dealiasing to stabilize the simulation. This method is based on the work of Kirby and Sherwin [7].

```
1 <I PROPERTY="SPECTRALHPDEALIASING" VALUE="True" />
```

11.3.2 Parameters

The following parameters can be specified in the **PARAMETERS** section of the session file:

- TimeStep: sets the timestep for the integration in time formula.
- NumSteps: sets the number of time-steps.
- IO_CheckSteps : sets the number of steps between successive checkpoint files.
- **IO_InfoSteps**: sets the number of steps between successive info stats are printed to screen.
- Kinvis: sets the cinematic viscosity coefficient formula.
- SubStepCFL: sets the CFL safety limit for the sub-stepping algorithm (default value = 0.5).
- MinSubSteps: perform a minimum number of substeps in sub-stepping algorithm (default is 1).
- MaxSubSteps: perform a maxmimum number of substeps in sub-stepping algorithm otherwise exit (default is 100).

- SVVCutoffRatio: sets the ratio of Fourier frequency not affected by the SVV technique (default value = 0.75, i.e. the first 75% of frequency are not damped).
- **SVVDiffCoeff**: sets the SVV diffusion coefficient (default value = 0.1 (Exponential and Power kernel), 1 (DG-Kernel)).
- GJPJumpScale: This is a parameter that scales all of the jump terms in the GJP stabilisation formulation. The default value GJPJumpScale=1.
- **I0_CFLWriteFld**: sets a treshold value for the CFL number. If CFL exceeds this value, the flow field is written to file (only once). This is useful for debugging purposes, allowing to visually inspect a flow field that is becoming unstable.
- IO_CFLWriteFldNumSteps: sets the number of timesteps after which IO_CFLWriteFld becomes operational. This avoids writing the flow field at the beginning of a simulation when initialising a new geometry.

11.3.3 Womersley Boundary Condition

It is possible to define the time-dependent Womersley velocity profile for pulsatile flow in a pipe. The modulation of the velocity profile is based on the desired peak or centerline velocity which can be represented by a Fourier expansion $U_{max} = A(\omega_n)e^{i\omega_n t}$ where A are the Fourier modes and ω the frequency. The womersely solution is then defined as:

$$w(r,t) = A_0(1 - (r/R)^2) + \sum_{n=1}^N \tilde{A}_n [1 - \frac{J_0(i^{3/2}\alpha_n r/R)}{J_0(i^{3/2}\alpha)}]e^{i\omega_n t}$$

where the womensley number α is defined:

$$\alpha_n = R \sqrt{\frac{2\pi n}{T\nu}}$$

and $\tilde{A_n}$ (n = 1 : N) are the Fourier coefficients scaled in the following way:

$$\tilde{A}_n = 2A_n / [1 - \frac{1}{J_0(i^{3/2}\alpha)}]$$

The Womersley velocity profile is implemented in the following way:

```
1 <REGION REF="0">
2 <D VAR="u" USERDEFINEDTYPE="Womersley:WomParams.xml" VALUE="0" />
3 <D VAR="v" USERDEFINEDTYPE="Womersley:WomParams.xml" VALUE="0" />
4 <D VAR="w" USERDEFINEDTYPE="Womersley:WomParams.xml" VALUE="0" />
5 <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
6 </REGION>
```

A file containing the Fourier coefficients, \tilde{A} , must be in the directory where the solver is called from. The name of the file is defined by the string given in the attribute **USERDEFINEDTYPE** after the ":" and contains the real and imaginary coefficients. This file has the format

```
1 <NEKTAR>
    <WOMERSLEYBC>
2
      <WOMPARAMS>
3
        <W PROPERTY="Radius" VALUE="0.5" />
4
        <W PROPERTY="Period" VALUE="1.0" />
5
        <W PROPERTY="axisnormal" VALUE="0.0,0.0,1.0" />
6
        <W PROPERTY="axispoint" VALUE="0.0,0.0,0.0" />
7
      </WOMPARAMS>
8
9
      <FOURIERCOEFFS>
10
        <F ID="0"> 0.600393641193.
                                        0.0
                                                             </F>
11
        <F ID="1"> -0.277707172935, 0.0767582715413
<F ID="2"> -0.0229953131146, 0.0760936232478
                                                             </F>
12
                                                             </F>
13
14
        <F ID="3"> 0.00858135174058, 0.017089888642
                                                             </F>
15
        <F ID="4"> 0.0140332527651, 0.0171575122496
                                                             </F>
16
         <F ID="5"> 0.0156970122129,
                                        -0.00547357750345 </F>
         <F ID="6"> 0.00473626554238, -0.00498786519876 </F>
17
         <F ID="7"> 0.00204434981523, -0.00614566561937 </F>
18
         <F ID="8"> -0.000274697215201, 0.000153571881197 </F>
19
         <F ID="9"> -0.000148037910774, 2.68919619581e-05 </F>
20
      </FOURIERCOEFFS>
21
    </WOMERSLEYBC>
22
23 </NEKTAR>
```

Each value of \tilde{A} is provided in the **FOURIERCOEFFS** section and provided as separate entries containing the real and imaginary components, i.e. the mean component provided above is 0.600393641193, 0.0.

Similarly in the (WOMPARAMS) section the key parameters of the boundary condition are also provided as:

- **(RADIUS)** is the radius of the boundary.
- **PERIOD** is the cycle time period,
- (AXISNORMAL) defines the normal direction to the boundary,
- (AXISPOINT) defines a coordinate in the boundary centre,

11.3.4 Forcing

11.3.4.1 MovingBody

Note

This force type is only supported for the Quasi-3D incompressible Navier-Stokes solver.

This force type allows the user to solve the interaction system of an incompressible fluid flowing past a flexible moving bodies [36]. By this forcing function, one can eliminate the difficulty of moving mesh by using body-fitted coordinates, so that an additional acceleration term(i.e., forcing term) is introduced to the momentum equations by the non-inertial transform from the deformed and moving coordinate system to non-deformed and stationary one.

1 <FORCE TYPE="MovingBody">
2 </FORCE>

Available options of the motion type for the moving body include free, constrained and forced vibrations, which can be specified in the **TIMEINTEGRATIONSCHEME** and **SOLVERINFO** section. The free type of motion allows the body to move in both streamwise and crossflow directions, while the constrained type limits the motion only in the crossflow direction. For the forced type, the vibration profiles of the body should be specified as a given function or read from input file in **MovingBody** section. For example:

1	<timei< th=""><th>NTEGRATIONSCHEME></th><th></th><th></th></timei<>	NTEGRATIONSCHEME>						
2	<method> IMEX </method>							
3	<pre><order> 2 </order></pre>							
4								
5								
6	6 <solverinfo></solverinfo>							
7	<i< td=""><td>PROPERTY="EQTYPE"</td><td>VALUE="UnsteadyNavierStokes"</td><td>/></td></i<>	PROPERTY="EQTYPE"	VALUE="UnsteadyNavierStokes"	/>				
8	<i< td=""><td>PROPERTY="SolverType"</td><td>VALUE="VelocityCorrectionScheme"</td><td>/></td></i<>	PROPERTY="SolverType"	VALUE="VelocityCorrectionScheme"	/>				
9	<i< td=""><td>PROPERTY="EvolutionOperator"</td><td>VALUE="SkewSymmetric"</td><td>/></td></i<>	PROPERTY="EvolutionOperator"	VALUE="SkewSymmetric"	/>				
10	<i< td=""><td>PROPERTY="Projection"</td><td>VALUE="Galerkin"</td><td>/></td></i<>	PROPERTY="Projection"	VALUE="Galerkin"	/>				
11	<i< td=""><td>PROPERTY="GlobalSysSoln"</td><td>VALUE="DirectStaticCond"</td><td>/></td></i<>	PROPERTY="GlobalSysSoln"	VALUE="DirectStaticCond"	/>				
12	<i< td=""><td>PROPERTY="HOMOGENEOUS"</td><td>VALUE="1D"</td><td>/></td></i<>	PROPERTY="HOMOGENEOUS"	VALUE="1D"	/>				
13	<i< td=""><td>PROPERTY="USEFFT"</td><td>VALUE="FFTW"</td><td>/></td></i<>	PROPERTY="USEFFT"	VALUE="FFTW"	/>				
14	<i< td=""><td>PROPERTY="VibrationType"</td><td>VALUE="FREE"</td><td>/></td></i<>	PROPERTY="VibrationType"	VALUE="FREE"	/>				
15								

A moving body type boundary condition should be specified in **BOUNDARYCONDITIONS** for the velocities on the moving body,

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3  <D VAR="u" USERDEFINEDTYPE="MovingBody" VALUE="0" />
4  <D VAR="v" USERDEFINEDTYPE="MovingBody" VALUE="0" />
5  <D VAR="w" VALUE="0" />
6  <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
7  </REGION>
8 </BOUNDARYCONDITIONS>
```

For the simulation of low mass ratio, there is an option to activate fictitious mass method for stabilizing explicit coupling between the fluid solver and structural dynamic solver. Here we need to specify the values of fictitious mass and damping in **PARAMETERS**, for example,

1 <SOLVERINFO>

```
2 <I PROPERTY="FictitiousMassMethod" VALUE="True" />

3 </SOLVERINFO>

4 <PARAMETERS>

5 <P> FictDamp = 1000 </P>

6 <P> FictMass = 1.5 </P>

7 </PARAMETERS>
```

A filter called MovingBody is encapsulated in this module to evaluate the aerodynamic forces along the moving body surface. The forces for each computational plane are projected along the Cartesian axes and the pressure and viscous contributions are computed in each direction.

The following parameters are supported:

Option name	Required	Default	Description
OutputFile	X	session	Prefix of the output filename to which the forces are written.
Frequency	X	1	Number of timesteps after which output is written.
Boundary	\checkmark	-	Boundary surfaces on which the forces are to be evaluated.

To enable the filter, add the following to the **FORCE**) tag::

```
1 <FORCE TYPE="MovingBody">
2 <PARAM NAME="OutputFile">DragLift</PARAM>
3 <PARAM NAME="OutputFrequency">10</PARAM>
4 <PARAM NAME="Boundary"> B[0] </PARAM>
5 </FORCE>
```

During the execution a file named DragLift.fce will be created and the value of the aerodynamic forces on boundary 0, defined in the GEDMETRY section, will be output every 10 time steps.evaluates the aerodynamic forces along the moving body surface. The forces for each computational plane are projected along the Cartesian axes and the pressure and viscous contributions are computed in each direction.

Also, to use this module a **MAPPING** needs to be specified, as described in section 11.6. In the case of free and constrained motion presented here, the functions defined by the mapping act as initial conditions. Also, when using the MovingBody forcing, it is not necessary to set the **TIMEDEPENDENT** property of the mapping.

11.3.5 Filters

The following filters are supported exclusively for the incompressible Navier-Stokes solver. Further filters from section 3.4 are also available for this solver.

- Aerodynamic forces (section 3.4.2)
- Aerodynamic forces SPM (section 3.4.2.1)
- Kinetic energy and enstrophy (section 3.4.11)
- Mean value (section 3.4.12)
- Modal energy (section 3.4.13)
- Moving body (section 3.4.14)
- Reynolds stresses (section 3.4.17)

11.4 Session file configuration: Linear stability analysis

Stability analyses of incompressible flow involves solving the linearised Navier-Stokes equations

$$\frac{\partial \mathbf{u}'}{\partial t} + \mathcal{L}(\mathbf{U}, \mathbf{u}') = -\nabla p + \nu \nabla^2 \mathbf{u}',$$

where \mathcal{L} is a linear operator, its adjoint form, or both. The evolution of the linearised Navier-Stokes operator, which evolves a solution from an initial state to a future time t, can be written as

$$u(t) = \mathcal{A}(t)u(0).$$

The adjoint evolution operator is denoted as \mathcal{A}^* . This section details the additional configuration options, in addition to the standard configuration options described earlier, relating to performing this task.

11.4.1 Solver Info

• Eqtype: sets the type of equation to solve. For linear stability analysis this must be set to

Equation Type	Dimensions	Projections	Algorithms
UnsteadyNavierStokes	2D, Quasi- $3D$	Continuous	VCS,DS

- EvolutionOperator: sets the choice of the evolution operator:
 - Nonlinear (standard non-linear Navier-Stokes equations).
 - Direct $(\mathcal{A} \text{linearised Navier-Stokes equations}).$
 - Adjoint (\mathcal{A}^* adjoint Navier-Stokes equations).
 - **TransientGrowth** ($\mathcal{A}^*\mathcal{A}$ transient growth evolution operator).
- Driver: specifies the type of problem to be solved:
 - (Standard) (normal time integration of the equations)
 - ModifiedArnoldi (computations of the leading eigenvalues and eigenmodes using modified Arnoldi method)
 - Arpack (computations of eigenvalues/eigenmodes using Implicitly Restarted Arnoldi Method (ARPACK)).
- ArpackProblemType: types of eigenvalues to be computed (for Driver Arpack only)
 - LargestMag (eigenvalues with largest magnitude).
 - SmallestMag (eigenvalues with smallest magnitude).
 - LargestReal (eigenvalues with largest real part).
 - SmallestReal (eigenvalues with smallest real part).
 - LargestImag (eigenvalues with largest imaginary part).
 - SmallestIma (eigenvalues with smallest imaginary part).
- Homogeneous: specifies the Fourier expansion in a third direction (optional)
 - (1D) (Fourier spectral method in z-direction).
- ModeType: this specifies the type of the quasi-3D problem to be solved.
 - MultipleMode (stability analysis with multiple modes, HomModesZ sets number of modes).
 - SingleMode (BiGlobal Stability Analysis: full-complex mode. Overrides HomModesZ to 1.).
 - HalfMode (BiGlobal Stability Analysis: half-complex mode u.Re v.Re w.Im p.Re).

Note

For visualization of Homogeneous results with FieldConvert you can use --output-points-hom-z to set output number of modes to a desired value. To process results obtained with HalfMode you can convert to SingleMode using FieldConvert module halfmodetofourier.

11.4.2 Parameters

The following parameters can be specified in the PARAMETERS section of the session file:

• kdim: sets the dimension of the Krylov subspace κ . Can be used with: ModifiedArnoldi and Arpack. Default value: 16.

- evtol: sets the tolerance of the iterative eigenvalue algorithm. Can be used with: ModifiedArnoldi) and Arpack. Default value: 1×10^{-6} .
- nvec: sets the number of converged eigenvalues sought. Can be used with: ModifiedArnoldi and Arpack. Default value: 2.
- **nits**: sets the maximum number of Arnoldi iterations to attempt. Can be used with: ModifiedArnoldi and Arpack. Default value: 500.
- **realShift**: provide a real shift to the direct solver eigenvalue problem by the specified value to improve convergence. Can be used with: **Arpack** only.
- imagShift: provide an imaginary shift to the direct solver eigenvalue problem by the specified value to improve convergence. Can be used with: Arpack only.
- LZ: sets the length in the spanswise direction L_z . Can be used with Homogeneous set to 1D. Default value: 1.
- HomModesZ: sets the number of planes in the homogeneous directions. Can be used with Homogeneous set to 1D and ModeType set to MultipleModes.
- **N_start**: sets the start number of temporal slices for Floquet stability analysis. Default value: 0.
- N_skip: sets the number skip of temporal slices for Floquet stability analysis. Default value: 1.
- N_slices: sets the number of temporal slices for Floquet stability analysis. Files sequence N_start, N_start + N_skip, ..., N_start + N_skip * (N_slices-1) will be loaded.
- BaseFlow_interporder: sets the interpolation order of temporal slices for Floquet stability analysis. If BaseFlow_interporder < 2, the baseflow is taken as periodic and trigonometric functions are used for interpolation. It should be noted that the file $N_{\text{start}} + N_{\text{skip}} * N_{\text{slices}}$ is at t = period and should not be loaded for the periodic case. If BaseFlow_interporder >= 2, the flow is taken as aperiodic and Lagrange interpolation is used. In this case, the file $N_{\text{start}} + N_{\text{skip}} * (N_{\text{slices-1}})$ is at t = period and should be loaded. Default value: 0.
- period: sets the time span (if BaseFlow_interporder >= 2) or period (if BaseFlow_interporder < 2) of the base flow. Default value: 1.

11.4.3 Functions

When using the direct solver for stability analysis it is necessary to specify a Forcing function "StabilityCoupledLNS" in the form:

```
1 <FORCING>
2 <FORCE TYPE="StabilityCoupledLNS">
3 </FORCE>
4 </FORCING>
```

This is required since we need to tell the solver to use the existing field as a forcing function to the direct matrix inverse as part of the Arnoldi iteration.



Note

Examples of the set up of the direct solver stability analysis (and other incompressible Navier-Stokes solvers) can be found in the regression test directory NEKTAR/solvers/IncNavierStokesSolver/Tests. See for example the files PPF_R15000_ModifiedArnoldi_Shift.tst and PPF_R15000_3D.xml noting that some parameters are specified in the .tst files.

11.5 Session file configuration: Steady-state solver

In this section, we detail how to use the steady-state solver (that implements the selective frequency damping method, see Sec. 11.1.5). Two cases are detailed here: the execution of the classical SFD method and the *adaptive* SFD method, where the control coefficient χ and the filter width Δ of the SFD method are updated all along the solver execution. For the second case, the parameters of the SFD method do not need to be defined by the user (they will be automatically calculated all along the solver execution) but several session files must be defined in a very specific way.

11.5.1 Execution of the classical steady-state solver

11.5.1.1 Solver Info

The definition of Eqtype, TimeIntegrationScheme and Projection is similar as what is explained in 11.4.1. The use of the steady-state solver is enforced through the definition of the Driver which has to be SteadyState. EvolutionOperator does not need to be defined to run the unadapted SFD method (by default, it is set to Nonlinear).

11.5.1.2 Parameters

The following parameters can be specified in the PARAMETERS section of the session file:

- Kinvis: sets the kinematic viscosity ν . It is typically 1/Re if both the characteristic velocity and characteristic length are chosen to be 1.
- ControlCoeff: sets the control coefficient χ of the SFD method. Default value: 1.
- FilterWidth: sets the filter width Δ of the SFD method. Default value: 2.

- GrowthRateEV and FrequencyEV: if the growth rate and the frequency of the dominant eigenvalue are known, they can be given given as input and the code will automatically select the optimum parameters χ and Δ (and overwrite the values of ControlCoeff and GrowthRateEV) that may be given in the session file)
- **TOL**: sets the tolerance of the SFD method. The code will run until $||q \bar{q}||_{inf} < TOL$. Default value: 10^{-8} .

Note that for the steady-state solver, the parameter NumSteps is not taken into account. The solver will run until a steady-state solution is found and not for a pre-defined number of time steps.

11.5.2 Execution of the adaptive steady-state solver

Running the adaptive selective frequency damping method requires to set up the session files in a very specific manner. First, the Geometry section must be in a separated archive file. If the test case studied is called "Session", the mesh file must be called Session.xml.gz (the linux command "gzip" can be used to obtain this file).

The requirements for the file Session.xml are similar as for the ones for the classical SFD method. The Geometry section being removed and placed in Session.xml.gz. This file defines the properties of the nonlinear problem solved (*i.e.* the flow for which we want a steady-state). Also, the SOLVERINFO section must contain the line:

1 <I PROPERTY="EvolutionOperator" VALUE="AdaptiveSFD" />

The adaptive SFD method used is coupled with a stability analysis method. Then kdim, nvec, evtol and nits should be defined into the PARAMETERS section of Session.xml. If not, these parameters will take the default values presented in Sec. 11.4.

The goal of running the stability analysis is to evaluate the dominant eigenvalue of a "partially converged" steady base flow. This approximation is then used by the steady-state solver to select a control coefficient χ and a filter width Δ then ensure a fast convergence towards a steady-state solution.

To define the linear stability problem, another file, that must be called Session_LinNS.xml, has to be defined. This file must be an exact copy/paste of Session.xml, only three things have to be modified:

- 1. The boundary conditions must be modified to be homogeneous (*i.e.* equal to zero) at all inflow boundaries.
- 2. A non-zero function InitialConditions has to be defined.
- 3. A random function **BaseFlow** has to be defined (it will be overwritten all along the solver execution). We recommend it to be a copy of **InitialConditions**.

Once these three files (the Geometry in Session.xml.gz, the nonlinear problem definition in Session.xml and the homogeneous linear problem in Session_LinNS.xml) are correctly defined, the adaptive SFD method must be executed using:

IncNavierStokesSolver Session.xml.gz Session.xml

11.6 Session file configuration: Coordinate transformations

This section describes how to include a coordinate transformation to the solution of the incompressible Navier-Stokes equations. In some cases, this approach allows a slightly deformed geometry to be mapped into a geometry with a homogeneous direction, which can be treated using a quasi-3D method. It is also useful for problems with a moving body, where otherwise a moving mesh would have to be employed.

11.6.1 Solver Info

To activate the mapping technique, SolverType needs to be set as:

```
1 <I PROPERTY="SolverType" VALUE="VCSMapping" />
```

Also, there are other optional properties in the (SolverInfo) section:

```
1 <I PROPERTY="MappingImplicitPressure" VALUE="TRUE"/> <!-- Default = FALSE -->
2 <I PROPERTY="MappingImplicitViscous" VALUE="TRUE"/> <!-- Default = FALSE -->
3
4 <I PROPERTY="MappingNeglectViscous" VALUE="FALSE"/> <!-- Default = FALSE -->
```

the first two options determine if the pressure and viscous terms resulting from the coordinate transformation are treated implicitly using an iterative procedure. If the last option is set to true, the viscous terms in the mapping are not computed. This leads to a faster solution, but the effect on the results need to be determined for the specific case. By default, all mapping terms are computed and treated explicitly.

11.6.2 Parameters

When treating the mapping terms implicitly, the following parameters can be set:

```
1 <P> MappingPressureTolerance= 1e-8 </P> <!-- Default = 1e-12 -->2 <P> MappingViscousTolerance= 1e-8 </P> <!-- Default = 1e-12 -->3 <P> MappingPressureRelaxation= 0.9 </P> <!-- Default = 1.0 -->4 <P> MappingViscousRelaxation= 0.9 </P> <!-- Default = 1.0 -->
```

They determine the tolerance of the iterative solution of the equations, and a relaxation parameter which can improve the numerical stability of the method.

11.6.3 Mapping

The particular transformation employed is specified by:

where (TIMEDEPENDENT) indicates if the transformation varies with time.

The available values for **TYPE**, and the transformations they represent, are:

Mapping type	\bar{x}	$ar{y}$	\overline{z}
Translation	x + f(t)	y + g(t)	z + h(t)
XofZ	x + f(z, t)	y	z
XofXZ	f(x, z, t)	y	z
XYofZ	x + f(z, t)	y + g(z, t)	z
XYofXY	f(x, y, t)	g(x, y, t)	z
General	f(x, y, z, t)	g(x, y, z, t)	h(x, y, z, t)

where $(\bar{x}, \bar{y}, \bar{z})$ are the Cartesian (physical) coordinates and (x, y, z) are the transformed coordinates. Note that for quasi-3D problems, the z coordinate cannot be transformed.

11.6.4 Functions

The function **COORDS** (and **VEL** for time dependent mappings) indicated in the **MAPPING** section need to be defined, for example as:

```
1
         <FUNCTION NAME="Mapping">
              <E VAR="x" VALUE="x + cos(PI*z)" />
2
              <E VAR="y" VALUE="y + cos(2*PI*t)" />
3
         </FUNCTION>
4
5
         <FUNCTION NAME="MappingVel">
6
              <E VAR="vx" VALUE="0.0" />
7
              <E VAR="vy" VALUE="-1.0*2*PI*sin(2*PI*t)" />
8
         </FUNCTION>
9
```

the transformation defined by these functions need to be valid (non-zero Jacobian). By default, any component of **COORDS** that is not specified is taken as a trivial transformation, e.g. $\bar{x} = x$, and any velocity not specified is considered to be zero.

11.6.5 Boundary conditions

In case of a time-dependent mapping, a moving body boundary condition is available:

```
1 <BOUNDARYCONDITIONS>

2 <REGION REF="0">

3 <D VAR="u" USERDEFINEDTYPE="MovingBody" VALUE="0" />

4 <D VAR="v" USERDEFINEDTYPE="MovingBody" VALUE="0" />
```

```
5 <D VAR="w" VALUE="0" />
6 <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
7 </REGION>
8 </BOUNDARYCONDITIONS>
```

When using the MovingBody boundary condition, the Dirichlet condition is relative to the boundary, while the regular Dirichlet boundary condition is taken in an absolute sense.

All Dirichlet boundary conditions are specified in the Cartesian (physical) space, and are automatically transformed to the computational frame of reference.

Note

Examples of the use of mappings can be found in the test directory NEKTAR/solvers/IncNavierStokesSolver/Tests. See for example the files KovaFlow_3DH1D_P8_16modes_Mapping-implicit.xml and CylFlow_Mov_mapping.xml.

11.7 Session file configuration: Adaptive polynomial order

An adaptive polynomial order procedure is available for 2D and Quasi-3D simulations. This procedure consists of the following steps:

- Advance the equations for a determined number of time steps
- Use the sensor defined in equation 9.9 to estimate an error measure (the variable used in the sensor can be specified). The error is defined here as the square of the sensor.
- Use the error to determine if the order in each element should be increased by one, decreased by one, or left unaltered.
- Project the solution in each element to the new polynomial order and use it as an initial condition to restart the equation, repeating all steps a given number of times.

It is important to note that restarting the simulation after the refinement can be an expensive operation (in a typical case 200 times the cost of a single time step). Therefore, the number of steps between successive refinements needs to be carefully chosen, since if this value is too low the procedure becomes inefficient, while if it is too high the refinement might not capture accurately structures that are convected by the flow.

11.7.1 Solver Info

The use of the adaptive polynomial order procedure is enforced through the definition of the Driver which has to be Adaptive.

11.7.2 Parameters

The following parameters can be specified in the PARAMETERS section of the session file:

- NumSteps: when using the adaptive order procedure, this parameter determines the number of time steps between successive refinements.
- NumRuns: this parameter defines the number of times the sequence of solving the equation and refining is performed. Therefore, the total number of time steps in the simulation is $NumSteps \times NumRuns$.
- AdaptiveMaxModes: sets the maximum number of modes (in each direction) that can be used in an element during the adaptive procedure. The solution will not be refined beyond this point, even if the error is higher than the tolerance. Default value: 12.
- AdaptiveMinModes: sets the minimal number of modes (in each direction) that can be used in an element during the adaptive procedure. Default value: 4.
- AdaptiveUpperTolerance: defines a constant tolerance. The polynomial order in an element is increased whenever the error is higher than this value. This can be replaced by a spatially-varying function, as described below. Default value: 10^{-6} .
- AdaptiveLowerinModes: defines a constant tolerance. The polynomial order in an element is decreased whenever the error is lower than this value. This can also be replaced by a spatially-varying function. Default value: 10^{-8} .
- AdaptiveSensorVariable: integer defining which variable will be considered when calculating the error. For example, if this parameter is set to 1 in the Incompressible Navier-Stokes Solver, the error will be estimated using the v velocity. Default value: 0.

11.7.3 Functions

Spatially varying tolerances can be specified by defining the functions AdaptiveLowerinModes and/or AdaptiveUpperTolerance. In this case, the tolerance in an element is taken as the average of the function in the quadrature points of the element. If these functions are defined, the values set for the tolerances in the PARAMETERS section are ignored.

11.7.4 Restarting the simulation

The simulation can be restarted using the final distribution of polynomial orders obtained from the adaptive procedure by setting the expansions as

```
1 <EXPANSIONS>
2 <F FILE="restartfile.fld" />
3 </EXPANSIONS>
```

note that this will only affect the polynomial order. The initial condition still needs to be set correctly, and does not need to come from the same file used for the expansions.

11.8 Advecting extra passive scalar fields

In some cases, it might be useful to advect passive scalar fields with the velocity obtained from the solution of the Navier-Stokes equation. For example, in study of mass transfer or heat transfer problems where getting analytical expression for advection velocity is not possible, the transport (advection-diffusion) equation needs to be solved along with the Navier-Stokes equation to get the scalar concentration or temperature distribution in the flow field.

In the input file, the extra field variables that are being advected need to be defined after the variables representing the velocity components. The pressure needs to be at the end of the list. For example, for a 2D simulation the expansions and variables would be defined as

```
1 <EXPANSIONS>
      <E COMPOSITE="C[0]" NUMMODES="5" FIELDS="u,v,c1,c2,p" TYPE="MODIFIED" />
2
3 </EXPANSIONS>
4
5 <VARIABLES>
     <V ID="0"> u </V>
6
      <V ID="1"> v </V>
7
      <V ID="2"> c1 </V>
8
      <V ID="3"> c2 </V>
9
      <V ID="4"> p </V>
10
11 </VARIABLES>
```

where u, v are the velocity components, c1 and c2 are extra fields that are being advected and p is the pressure.

In addition, diffusion coefficients for each extra variable can be specified by adding a function (DiffusionCoefficient)

Boundary conditions for the extra fields are set up in the same way as the velocity and pressure

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="u" VALUE="0" />
4 <D VAR="v" VALUE="0" />
5 <D VAR="c1" VALUE="1" />
6 <D VAR="c2" VALUE="1" />
7 <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
8 </REGION>
9 </BOUNDARYCONDITIONS>
```

It should be noted that if the diffusion coefficient is too small, the transport equation becomes advection dominated. This leads to small grid spacing required to resolve all physical scales associated with the transport equation (the ratio of resolution required for transport to Navier Stokes equation scales with $Sc^{3/4}$, where Sc is the Schmidt number = kinematic viscosity/diffusion coefficient). Hence, small diffusion coefficient might lead to spurious oscillations if the mesh spacing is not small enough.

11.9 Imposing a constant flowrate

In some simulations, it may be desirable to drive the flow by fixing a value of the volumetric flux through a boundary surface. A common use case for this is a channel flow, where the inflow and outflow are treated using periodic boundary conditions, requiring a use of either a body force or a flowrate condition to drive the flow. Often, the use of a body force is sufficient, but in some cases (e.g. transitional or turbulent simulations), it may be difficult to determine the correct body force to use in order to attain a specific Reynolds number. The incompressible solver supports the use of an alternative forcing whereby the volumetric flux,

$$Q(\mathbf{u}) = \frac{1}{\mu(R)} \int_R \mathbf{u} \cdot d\mathbf{s},$$

through a user-defined surface R of area $\mu(R)$ is kept constant. This is supported for standard two- and three-dimensional simulations, where R is a boundary region, as well as three-dimensional homogeneous simulations. In the latter case, the forcing can be imposed either in the homogeneous direction (in the x - y plane) or perpendicular to it (in the z direction).

The flowrate correction works by taking each timestep's velocity field \mathbf{u}^n , and computing a scalar α so that the corrected flow

$$\tilde{\mathbf{u}}^n = \mathbf{u}^n + \alpha \mathbf{u}_s$$

has the desired flowrate. Here, \mathbf{u}_s is a linear Stokes solution that is calculated once at the start of the simulation, so that the condition is not expensive to implement.

To enable flowrate corrections, three things must be defined in the session file:

• The Flowrate parameter in the parameters section, which defines the desired value of the volumetric flux $Q(\mathbf{u})$ through the reference region. To set a flux per unit surface of Q = 1 we would therefore define:

```
1 <PARAMETERS>
2 <P> Flowrate = 1.0 </P>
3 </PARAMETERS>
```

• A boundary condition must be tagged with the **Flowrate** user-defined type to define the reference region *R*. For example, a 2D channel flow with periodic boundary conditions might use the following arrangement:

```
1 <BOUNDARYCONDITIONS>
\mathbf{2}
    <REGION REF="0">
      <P VAR="u" VALUE="[1]" USERDEFINEDTYPE="Flowrate" />
3
      <P VAR="v" VALUE="[1]" />
4
      <P VAR="p" VALUE="[1]" />
\mathbf{5}
    </REGION>
6
    <REGION REF="1">
7
      <P VAR="u" VALUE="[0]" />
8
      <P VAR="v" VALUE="[0]" />
9
      <P VAR="p" VALUE="[0]" />
10
   </REGION>
11
12 </BOUNDARYCONDITIONS>
```

• a FlowrateForce function with components ForceX, ForceY and ForceZ that defines the direction in which the forcing will be applied. This should be a unit vector (i.e. of magnitude 1) and constant (i.e. not dependent on x, y, z or t). As an example, to impose a force in the x-direction we specify:

```
1 <FUNCTION NAME="FlowrateForce">
2 <E VAR="ForceX" VALUE="1.0" />
3 <E VAR="ForceY" VALUE="0.0" />
4 <E VAR="ForceZ" VALUE="0.0" />
5 </FUNCTION>
```

Importantly, note that in homogeneous simulations where the forcing is in the z-direction only the **Flowrate** parameter should be specified, and the reference area R is taken to be the homogeneous plane.

Optionally, the IO_FlowSteps parameter can be defined. If set to a non-zero integer, this produces a file session.prs which records the value of α used in the flowrate correction, every IO_FlowSteps steps.

11.10 Examples

11.10.1 Kovasznay Flow 2D

This example demonstrates the use of the velocity correction o solve the 2D Kovasznay flow at Reynolds number Re = 40. In the following we will numerically solve for the two dimensional velocity and pressure fields with steady boundary conditions.

11.10.1.1 Input file

The input for this example is given in the example file KovaFlow_m8.xml. The mesh consists of 12 quadrilateral elements.

We will use a 7th-order polynomial expansions (N = 8 modes) using the modified Legendre basis and therefore require the following expansion definition.

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="6" FIELDS="u,v,p" TYPE="MODIFIED" />
3 </EXPANSIONS>
```

We next specify the time integration scheme and solver information for our problem. In particular, we select the velocity correction scheme formulation, using a continuous Galerkin projection. For this scheme, an implicit-explicit time-integration scheme must be used and we choose one of second order.

```
1 <TIMEINTEGRATIONSCHEME>
      <METHOD> IMEX </METHOD>
2
      <ORDER> 2 </ORDER>
3
4 </TIMEINTEGRATIONSCHEME>
5
6 <SOLVERINFO>
    <I PROPERTY="EQTYPE"
                                        VALUE="UnsteadyNavierStokes"
                                                                          />
7
     <I PROPERTY="SolverType"
                                         VALUE="VelocityCorrectionScheme"
                                                                         />
8
     <I PROPERTY="AdvectionForm"
                                         VALUE="Convective"
                                                                          1>
9
     <I PROPERTY="Projection"
                                         VALUE="Galerkin"
                                                                          />
10
11 </SOLVERINFO>
```

The key parameters are listed below. Since the problem is unsteady we prescribe the time step and the total number of time steps. We also know the required Reynolds number, but we must prescribe the kinematic viscosity to the solver. We first define a *dummy* parameter for the Reynolds number, and then define the kinematic viscosity as the inverse of this. The value of λ is used when defining the boundary conditions and exact solution. Note that (PI) is a pre-defined constant.

```
1 <PARAMETERS>
2 <P> TimeStep
                   = 0.001 </P>
  <P> NumSteps
                  = 100
3
                            </P>
  <P> Re
                   = 40
                            </P>
4
                 = 1/Re
  <P> Kinvis
                           </P>
\mathbf{5}
  <P> LAMBDA
                    = 0.5*Re-sqrt(0.25*Re*Re+4*PI*PI)</P>
6
7 </PARAMETERS>
```

We choose to impose a mixture of boundary condition types as defined below.

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3  <REGION REF="0">
3  <D VAR="u" VALUE="1-exp(LAMBDA*x)*cos(2*PI*y)" />
4  <D VAR="v" VALUE="(LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)" />
5  <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
6  </REGION>
```

```
7
      <REGION REF="1">
         <D VAR="u" VALUE="1-exp(LAMBDA*x)*cos(2*PI*y)" />
8
         <D VAR="v" VALUE="(LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)" />
9
         <D VAR="p" VALUE="0.5*(1-exp(2*LAMBDA*x))" />
10
    </REGION>
11
    <REGION REF="2">
12
         <N VAR="u" VALUE="0" />
13
         <D VAR="v" VALUE="0" />
14
         <N VAR="p" VALUE="0" />
15
     </REGION>
16
17 </BOUNDARYCONDITIONS>
```

Initial conditions are obtained from the file KovaFlow_m8.rst, which is a Nektar++ field file. This is the output of an earlier simulation, renamed with the extension (rst) to avoid being overwritten, and is used in this case to reduce the integration time necessary to obtain the steady flow.

```
1 <FUNCTION NAME="InitialConditions">
2  <F FILE="KovaFlow_m8.rst" />
3 </FUNCTION>
```

Note the use of the (\mathbf{E}) tag to indicate the use of a file. In contrast, the exact solution is prescribed using analytic expressions which requires the use of the (\mathbf{E}) tag.

```
1 <FUNCTION NAME="ExactSolution">
2  <E VAR="u" VALUE="1-exp(LAMBDA*x)*cos(2*PI*y)" />
3  <E VAR="v" VALUE="(LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)" />
4  <E VAR="p" VALUE="0.5*(1-exp(2*LAMBDA*x))" />
5 </FUNCTION>
```

11.10.1.2 Running the simulation

Launch the simulation using the following command

IncNavierStokesSolver KovaFlow_m8.xml

After completing the prescribed 100 time-steps, the difference between the computed solution and the exact solution will be displayed. The actual mantissas may vary slightly, but the overall magnitude should be as shown.

```
L 2 error (variable u) : 3.75296e-07
L inf error (variable u) : 5.13518e-07
L 2 error (variable v) : 1.68897e-06
L inf error (variable v) : 2.23918e-06
L 2 error (variable p) : 1.46078e-05
L inf error (variable p) : 5.18682e-05
```

The output of the simulation is written to KovaFlow_m8.fld. This can be visualised by converting it to a visualisation format. For example, to use ParaView, convert the output into VTK format using the tility.

```
FieldConvert KovaFlow.xml KovaFlow.fld KovaFlow.vtu
```

The result should look similar to that shown in Figure 11.4.



Figure 11.4 Velocity profiles for the Kovasznay Flow (2D).

11.10.2 Kovasznay Flow 2D using high-order outflow boundary conditions

In this example, we solve the same case of 2D Kovasznay flow on severely-truncated computational domain but using a high-order outflow boundary condition, which is much more accurate and robust for unbounded flows [9]. The solver information and parameters used here are similar to the previous one. What only we need to modify in the input file is just the boundary condition type upon the outlet region shown as below

```
1 <BOUNDARYCONDITIONS>
      <REGION REF="0">
2
           <D VAR="u" VALUE="1-exp(KovLam*x)*cos(2*PI*y)" />
3
           <D VAR="v" VALUE="(KovLam/2/PI)*exp(KovLam*x)*sin(2*PI*y)" />
4
           <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
\mathbf{5}
6
      </REGION>
      <REGION REF="1">
7
           <N VAR="u" USERDEFINEDTYPE="HOutflow"</pre>
8
                 VALUE="-Kinvis*KovLam*exp(KovLam*x)*cos(2*PI*y)
9
                   - 0.5*(1-exp(2*KovLam*x))-0.5*(((1-exp(KovLam*x)*cos(2*PI*y))
10
                   *(1-exp(KovLam*x)*cos(2*PI*y))+(KovLam/(2*PI)*exp(KovLam*x)
11
                   *sin(2*PI*y))*(KovLam/(2*PI)*exp(KovLam*x)*sin(2*PI*y))))
12
                   *(0.5*(1.0-tanh((1-exp(KovLam*x)*cos(2*PI*y))*20)))" />
13
           <N VAR="v" USERDEFINEDTYPE="HOutflow"</pre>
14
                 VALUE="Kinvis*KovLam*KovLam/(2*PI)*exp(KovLam*x)*sin(2*PI*y)" />
15
```

```
<D VAR="p" USERDEFINEDTYPE="HOutflow"</pre>
16
                 VALUE="-Kinvis*KovLam*exp(KovLam*x)*cos(2*PI*y)
17
                   - 0.5*(1-exp(2*KovLam*x)) -0.5*(((1-exp(KovLam*x)*cos(2*PI*y))
18
                   *(1-exp(KovLam*x)*cos(2*PI*v))+(KovLam/(2*PI)*exp(KovLam*x)
19
                   *sin(2*PI*y))*(KovLam/(2*PI)*exp(KovLam*x)*sin(2*PI*y))))
20
                   *(0.5*(1.0-tanh((1-exp(KovLam*x)*cos(2*PI*y))*20)))" />
21
22
      </REGION>
23
      <REGION REF="2">
          <N VAR="u" VALUE="0" />
24
          <D VAR="v" VALUE="0" />
25
          <N VAR="p" VALUE="0" />
26
      </REGION>
27
28 </BOUNDARYCONDITIONS>
```

We note that in this example the "VALUE" property is set based on the analytic solution but this is not typically known and so often a VALUE of zero will be specified.

Instead of loading an initial condition from a specified file, we initialized the flow fields in this example by using following expressions

```
1 <FUNCTION NAME="InitialConditions">
2  <E VAR="u" VALUE="(1-exp(KovLam*x)*cos(2*PI*y))" />
3  <E VAR="v" VALUE="(KovLam/(2*PI))*exp(KovLam*x)*sin(2*PI*y)" />
4  <E VAR="p" VALUE="0.5*(1-exp(2*KovLam*x))" />
5 </FUNCTION>
```

11.10.2.1 Running the simulation

We then launch the simulation by the same solver as that in the previous example

```
IncNavierStokesSolver KovaFlow_m8_short_HOBC.xml
```

The solution with errors displayed as below

```
L 2 error (variable u) : 2.51953e-08
L inf error (variable u) : 9.56014e-09
L 2 error (variable v) : 1.10694e-08
L inf error (variable v) : 9.47464e-08
L 2 error (variable p) : 5.59175e-08
L inf error (variable p) : 2.93085e-07
```

The physical solution visualized in velocity profiles is also illustrated in Figure 11.5.

11.10.3 Steady Kovasznay Oseen Flow using the direct solver

In this example, we instead compute the steadhy Kovasznay Oseen flow using the direct solver. In contrast to the velocity correction scheme in which we time-step the solution to the final time, the direct solver computes the solution with a single solve.



Figure 11.5 Velocity profiles for the Kovasznay Flow in truncated domain (2D).

11.10.3.1 Input file

We can begin with the same input file as for the previous example, but with the following modifications. For reference, the modified version is provided in the example <code>Dseen_m8.xml</code>.

In the solver information, we must instead select the Steady-Oseen equation type and choose to use the coupled linearised Navier-Stokes

```
1 <I PROPERTY="EQTYPE" VALUE="SteadyOseen" />
2 <I PROPERTY="SolverType" VALUE="CoupledLinearisedNS" />
```

Note

Since we are using a coupled system, we are not solving for the pressure. We should therefore remove all references to the variable **p** in the session. In particular, it should be removed from the **EXPANSIONS**, **VARIABLES**, **BOUNDARYCONDITIONS** and **FUNCTIONS** sections of the file.

Instead of loading an initial condition from file, we can simply prescribe a zero field.

We must also provide an advection velocity.

11.10.3.2 Running the simulation

Run the simulation using

IncNavierStokesSolver Oseen_m8.xml

The resulting flow field should match the solution from the previous example.

11.10.4 Laminar Channel Flow 2D

In this example, we will simulate the flow through a channel at Reynolds number 1 with fixed boundary conditions.

11.10.4.1 Input file

The input file for this example is given in ChanFlow_m3_SKS.xml. The geometry is a square channel with height and length D = 1, discretised using four quadrilateral elements. We use a quadratic expansion order, which is sufficient to capture the quadratic flow profile. In this example, we choose to use the skew-symmetric form of the advection term. This is chosen in the solver information section:

1 <I PROPERTY="EvolutionOperator" VALUE="SkewSymmetric" />

A first-order time integration scheme is used and we set the time-step and number of time integration steps in the parameters section. We also prescribe the kinematic viscosity $\nu = 1/Re = 1$.

Boundary conditions are defined on the walls (region 0) and at the inflow (regions 1) as Dirichlet for the velocity field and as high-order for the pressure. At the outflow the velocity is left free using Neumann boundary conditions and the pressure is pinned to zero.

```
1 <BOUNDARYCONDITIONS>
      <REGION REF="0">
2
          <D VAR="u" VALUE="0" />
3
          <D VAR="v" VALUE="0" />
4
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
\mathbf{5}
      </REGION>
6
      <REGION REF="1">
7
          <D VAR="u" VALUE="y*(1-y)" />
8
          <D VAR="v" VALUE="0" />
9
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
10
      </REGION>
11
      <REGION REF="2">
12
          <N VAR="u" VALUE="0" />
13
          <N VAR="v" VALUE="0" />
14
          <D VAR="p" VALUE="0" />
15
      </REGION>
16
17 </BOUNDARYCONDITIONS>
```

Initial conditions are set to zero. The exact solution is a parabolic profile with a pressure gradient dependent on the Reynolds number. This is defined to allow verification of the calculation.

11.10.4.2 Running the solver

```
IncNaverStokesSolver ChanFlow_m3_SKS.xml
```

The error in the solution should be displayed and be close to machine precision

```
L 2 error (variable u) : 4.75179e-16
L inf error (variable u) : 3.30291e-15
L 2 error (variable v) : 1.12523e-16
L inf error (variable v) : 3.32197e-16
L 2 error (variable p) : 1.12766e-14
L inf error (variable p) : 7.77156e-14
```

The solution should look similar to that shown in Figure 11.6.



Figure 11.6 Pressure and velocity profiles for the laminar channel flow (2D).

11.10.5 Laminar Channel Flow 3D

We now solve the incompressible Navier-Stokes equations on a three-dimensional domain. In particular, we solver the three-dimensional equivalent of the previous example. We will also solve the problem in parallel. Note In ord

In order to run the example, you must have a version of Nektar++ compiled with MPI. This is the case for the packaged binary distributions.

11.10.5.1 Input file

The input file for this example is given in **Tet_channel_m8_par.xml**. In this example we use tetrahedral elements, indicated by the **A** element tags in the geometry section. All dimensions have length D = 1. We will use a 7th-order polynomial expansion. Since we now have three dimensions, and therefore three velocity components, the expansions section is now

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="8" FIELDS="u,v,w,p" TYPE="MODIFIED" />
3 </EXPANSIONS>
```

The solver information and parameters are similar to the previous example. Boundary conditions must now be defined on the six faces of the domain. Flow is prescribed in the z-direction through imposing a Poiseulle profile on the inlet and side walls. The outlet is zero-Neumann and top and bottom faces impose zero-Dirichlet conditions.

```
1 <BOUNDARYREGIONS>
                              <!-- Inlet -->
      <B ID="0"> C[1] </B>
2
                              <!-- Outlet -->
      <B ID="1"> C[6] </B>
3
      <B ID="2"> C[2] </B>
                              <!-- Wall -->
4
                              <!-- Wall left -->
      <B ID="3"> C[3] </B>
5
      <B ID="4"> C[4] </B>
                              <!-- Wall -->
6
7
      <B ID="5"> C[5] </B>
                              <!-- Wall right -->
8 </BOUNDARYREGIONS>
9
10 <BOUNDARYCONDITIONS>
    <REGION REF="0">
11
         <D VAR="u" VALUE="0" />
12
          <D VAR="v" VALUE="0" />
13
         <D VAR="w" VALUE="v*(1-v)" />
14
         <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
15
     </REGION>
16
      <REGION REF="1">
17
          <N VAR="u" VALUE="0" />
18
          <N VAR="v" VALUE="0" />
19
          <N VAR="w" VALUE="0" />
20
21
          <D VAR="p" VALUE="0" />
22
     </REGION>
      <REGION REF="2">
23
         <D VAR="u" VALUE="0" />
24
          <D VAR="v" VALUE="0" />
25
          <D VAR="w" VALUE="0" />
26
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
27
     </REGION>
28
      <REGION REF="3">
29
          <D VAR="u" VALUE="0" />
30
```

```
31
          <D VAR="v" VALUE="0" />
          <D VAR="w" VALUE="v*(1-y)" />
32
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
33
      </REGION>
34
      <REGION REF="4">
35
          <D VAR="u" VALUE="0" />
36
          <D VAR="v" VALUE="0" />
37
          <D VAR="w" VALUE="0" />
38
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
39
      </REGION>
40
      <REGION REF="5">
41
          <D VAR="u" VALUE="0" />
42
          <D VAR="v" VALUE="0" />
43
          <D VAR="w" VALUE="y*(1-y)" />
44
45
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
46
      </REGION>
47 </BOUNDARYCONDITIONS>
```

Initial conditions and exact solutions are also prescribed.

11.10.5.2 Running the solver

To run the solver in parallel, we use the mpirun command.

```
mpirun -np 2 IncNaverStokesSolver Tet_channel_m8_par.xml
```

The expected results are shown in Figure 11.7.



Figure 11.7 Pressure and velocity profiles for the laminar channel flow (full 3D).

11.10.6 Laminar Channel Flow Quasi-3D

For domains where at least one direction is geometrically homogeneous, a more efficient discretisation is to use a pure spectral discretisation, such as a Fourier expansion, in these directions. We use this approach to solve the same problem as in the previous example. We reuse the two-dimensional spectral/hp element mesh from the nd couple this with a Fourier expansion in the third component.

11.10.6.1 Input file

The input file for this example is ChanFlow_3DH1D_MVM.xml. We indicate that we are coupling the spectral/hp element domain with a pure spectral expansion using the following solver information

```
1 <I PROPERTY="HOMOGENEOUS" VALUE="1D"/>
```

We must also specify parameters to describe the particular spectral expansion

1 <P> HomModesZ = 20 </P> 2 <P> LZ = 1.0 </P>

The parameter (HomModesZ) specifies the number of Fourier modes to use in the homogeneous direction. The (LZ) parameter specifies the physical length of the domain in that direction.

Note

This example uses an in-built Fourier transform routine. Alternatively, one can use the FFTW library to perform Fourier transforms which typically offers improved performance. This is enabled using the following solver information

1 <I PROPERTY="USEFFT" VALUE="FFTW"/>

As with the spectral/hp element mesh consists of four quadrilateral elements with a second-order polynomial expansion. Since our domain is three-dimensional we have to now include the third velocity component

1 <E COMPOSITE="C[0]" NUMMODES="3" FIELDS="u,v,w,p" TYPE="MODIFIED" />

The remaining parameters and solver information is similar to previous examples.

Boundary conditions are specified as for the two-dimensional case (except with the addition of the third velocity component) since the side walls are now implicitly periodic. The initial conditions and exact solution are prescribed as for the fully three-dimensional case.

11.10.6.2 Running the solver

```
IncNaverStokesSolver ChanFlow_3DH1D_MVM.xml
```

The results can be post-processed and should match those of the fully three-dimensional case as shown in Figure 11.7.

11.10.7 Turbulent Channel Flow

In this example we model turbulence in a three-dimensional square channel at a Reynolds number of 2000.

Note

This example requires the FFTW Fast-Fourier transform library to be selected when compiling Nektar++.

11.10.7.1 Input file

The input file for this example is TurbChFl_3DH1D.xml. The geometry makes used of the homogeneous extension discussed in the previous example. The channel has height D = 2 and length $L = 4\pi$ and is discretised using quadratic quadrilateral elements in the spectral/hp element plane and a Fourier basis in the third coordinate direction. The elements are non-uniformly distributed so as to best capture the flow features with fewest degrees of freedom and is shown in Figure 11.8.



Figure 11.8 Mesh used for the turbulent channel flow.

The spanwise length of the channel is set using the LZ parameter and discretised with 32 Fourier modes by setting the value of (HomModesZ).

1	<p></p>	HomModesZ	=	32	
2	<p></p>	LZ	=	4* PI /3	

A second-order IMEX scheme is used for time-integration scheme is used with a time-step of 0.0001. The length of the simulation is 1 time-unit (10,000 steps).

Periodicity is naturally enforced in the spanwise direction, so boundary conditions need only be provided for the upper and lower walls, inlet and outlet as denoted by the following (BOUNDARYREGIONS).

1 <BOUNDARYREGIONS>
2 <B ID="0"> C[1] //walls
3 <B ID="1"> C[2] //inflow
4 <B ID="2"> C[3] //outflow
5 </BOUNDARYREGIONS>

In this example, we will use a body force to drive the flow and so, in addition to the spanwise periodicity, enforce periodicity in the streamwise direction of the spectral/hp element mesh. This is achieved by imposing the following boundary conditions

```
<REGION REF="1">
1
2
     <P VAR="u" VALUE="[2]" />
     <P VAR="v" VALUE="[2]" />
3
   <P VAR="w" VALUE="[2]" />
\mathbf{4}
    <P VAR="p" VALUE="[2]" />
5
6 </REGION>
7 <REGION REF="2">
    <P VAR="u" VALUE="[1]" />
8
     <P VAR="v" VALUE="[1]" />
9
   <P VAR="w" VALUE="[1]" />
10
    <P VAR="p" VALUE="[1]" />
11
12 </REGION>
```

Here, we use \mathbf{P} to denote the boundary type is periodic, and the value in square brackets denotes the boundary region to which the given boundary is periodic with. In this case regions 1 and 2 are denoted periodic with each other.

A streamwise plug-profile initial condition is prescribed such that u = 1 everywhere, except the wall boundaries. The body force requires two components in the XML file. The first specifies the type of forcing to apply and appears directly within the **NEKTAR** tag.

```
1 <FORCING>
2 <FORCE TYPE="Body">
3 <BODYFORCE> BodyForce </BODYFORCE>
4 </FORCE>
5 </FORCING>
```

The second defines the BodyForce function which will be used and is located within the CONDITIONS section,

```
1 <FUNCTION NAME="BodyForce">
2   <E VAR="u" VALUE="2*Kinvis" />
3   <E VAR="v" VALUE="0" />
4   <E VAR="w" VALUE="0" />
5 </FUNCTION>
```

To improve numerical stability, we also enable dealising of the advection term. This uses additional points to perform the quadrature and then truncates the higher-order terms when projecting back onto the polynomial space, thereby removing spurious oscillations. It is enabled by setting the solver information tag

```
1 <I PROPERTY="DEALIASING" VALUE="True" />
```

This feature is only available when using the FFTW library is used, so we enable this using

```
1 <I PROPERTY="USEFFT" VALUE="FFTW" />
```

11.10.7.2 Running the solver

To run the solver, we use the following command

```
IncNaverStokesSolver TurbChFl_3DH1D.xml
```

The result after transition has occurred is illustrated in Figure 11.9.



Figure 11.9 Velocity profile of the turbulent channel flow (quasi-3D).

11.10.8 Turbulent Pipe Flow

In this example we simulate flow in a pipe at Reynolds number 3000 using a mixed spectral/hp element and Fourier discretisation. The Fourier expansion is used in the streamwise direction in this case and the spectral/hp elements are used to capture the circular cross-section.

11.10.8.1 Input File

The circular pipe has diameter D = 1, the 2D mesh is composed of 64 quadrilateral elements and the streamwise direction is discretised with 128 Fourier modes. An illustrative diagram of the discretisation is given in Figure 11.10.



Figure 11.10 Domain for the turbulent pipe flow problem.

The input file for this example is **Pipe_turb.xml**. We use 7th-order lagrange polynomials through the Gauss-Lobatto-Legendre points for the quadrilateral expansions.

```
1 <E COMPOSITE="C[0]" NUMMODES="8" FIELDS="u,v,w,p" TYPE="GLL_LAGRANGE_SEM" />
```

We set the Fourier options, as in the previous example, except using 128 modes and a length of 5 non-dimensional units. A small amplitude noise is also added to the initial condition, which is a plug profile, to help stimulate transition. Since the streamwise direction is the Fourier direction, we must necessarily use a body force to drive the flow.

11.10.8.2 Running the solver

In this example we will run the solver in parallel. Due to the large number of Fourier modes and relatively few elements, it is more efficient to parallelise in the streamwise direction. We can specify this by providing an additional flag to the solver, <code>-npz</code>. This indicates the number of partitions in the z-coordinate. In this example, we will only run two processes. We therefore would specify <code>-npz 2</code> to ensure parallelisation only occurs in the Fourier direction.

To improve the efficiency of the solver further, we would prefer to solve the Helmholtz problems within the spectral/hp element planes using a direct solver (since no parallelisation is necessary). The default when running in parallel is to use an iterative solver, so we explicitly specify the type of algorithm to use in the session file solver information:

```
1 <I PROPERTY="GlobalSysSoln" VALUE="DirectStaticCond" />
```

The solver can now be run as follows

```
mpirun -np 2 IncNavierStokesSolver --npz 2 Pipe_turb.xml
```

When the pipe transitions, the result should look similar to Figure 11.11.



Figure 11.11 Velocity profile of the turbulent pipe flow (quasi-3D).

11.10.9 Aortic Blood Flow

The following example demonstrates the application of the incompressible Navier-Stokes solver using the Velocity Correction Scheme algorithm for modelling viscid Newtonian blood flow in a region of a rabbit descending thoracic aorta with intercostal branch pairs. Such studies are necessary to understand the effect local blood flow changes have on cardiovascular diseases such as atherosclerosis.

In the following we will numerically solve for the three dimensional velocity and pressure field for steady boundary conditions. The Reynolds number under consideration is 300, which is physiologically relevant.

Geometry

The geometry under consideration is a segment of a rabbit descending aorta with two pairs of intercostal arteries branching off. The inlet has a diameter D = 3.32mm.

In order to capture the physics of the flow in the boundary layer, a thin layer consisting of prismatic elements is created adjacent to the surface, and curved using spherigons. The interior consist of tetrahedral elements.

Input parameters



Figure 11.12 Reduced region of rabbit descending thoracic aorta.



Figure 11.13 Surface mesh indicating curved surface elements at a branch location.

11.10.9.1 Expansion:

In this example we will use a fourth order polynomial expansion. There are two composites defined here since we have both prismatic and tetrahedral elements.

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="5" TYPE="MODIFIED" FIELDS="u,v,w,p" />
3 <E COMPOSITE="C[1]" NUMMODES="5" TYPE="MODIFIED" FIELDS="u,v,w,p" />
4 </EXPANSIONS>
```

11.10.9.2 Time Integration Scheme information:

```
1 <TIMEINTEGRATIONSCHEME>
```

- 2 <METHOD> IMEX </METHOD>
- 3 <ORDER> 1 </ORDER>
- 4 </TIMEINTEGRATIONSCHEME>

11.10.9.3 Solver information:

1 <SOLVERINFO>

2 3 4

<i <="" property="SolverType" th=""><th>VALUE="VelocityCorrectionScheme"</th><th>/></th></i>	VALUE="VelocityCorrectionScheme"	/>
<i <="" property="EQTYPE" td=""><td>VALUE="UnsteadyNavierStokes"</td><td>/></td></i>	VALUE="UnsteadyNavierStokes"	/>
<i <="" property="AdvectionForm" td=""><td>VALUE="Convective"</td><td>/></td></i>	VALUE="Convective"	/>

```
5 <I PROPERTY="Projection" VALUE="Galerkin" />
6 </SOLVERINFO>
```

11.10.9.4 Parameters:

Since we are prescribing a Reynolds number of 300, and to simplify the problem definition, we set the mean inlet velocity to 1, this allows us to define the kinematic viscosity as $\nu = \frac{UD}{Re} = \frac{3.32}{300} = 1/90.36.$

1	<parame< th=""><th>TERS></th><th></th><th></th><th></th></parame<>	TERS>			
2	<p></p>	TimeStep	=	0.0005	
3	<p></p>	NumSteps	=	1600	
4	<p></p>	IO_CheckSteps	=	200	
5	<p></p>	IO_InfoSteps	=	50	
6	<p></p>	Kinvis	=	1.0/90.36	
$\overline{7}$	<th>ETERS></th> <th></th> <th></th> <th></th>	ETERS>			

11.10.9.5 Boundary conditions:

For the purpose of this example a blunted inlet velocity profile has been prescribed. Ideally to obtain more significant results, the velocity profile at the inlet would be obtained from previous simulations on the complete rabbit aorta (including aortic root, aortic arch, and descending aorta with all 5 pairs of intercostal arteries), where a blunted profile at the aortic root is a better representation of reality.

Dirichlet boundary conditions are imposed for the velocity at the inlet, as well as on the wall to account for the no-slip condition. Neumann boundary conditions are imposed for the velocity field at the outlets where fully developed flow is imposed.

```
1 <BOUNDARYREGIONS>
      <B ID="0"> C[2] </B>
\mathbf{2}
                                      <!-- Inlet -->
      <B ID="1"> C[3,4,5,6] </B>
3
                                      <!-- intercostal outlets -->
      <B ID="2"> C[7] </B>
                                      <!-- outlet -->
4
      <B ID="3"> C[8] </B>
                                      <!-- wall -->
\mathbf{5}
6 </BOUNDARYREGIONS>
7
8 <BOUNDARYCONDITIONS>
      <REGION REF="0">
9
          <D VAR="u" VALUE="0.024" />
10
          <D VAR="v" VALUE="-0.064" />
11
          <D VAR="w" VALUE="-0.998" />
12
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
13
14
      </REGION>
      <REGION REF="1">
15
          <N VAR="u" VALUE="0" />
16
          <N VAR="v" VALUE="0" />
17
          <N VAR="w" VALUE="0" />
18
          <D VAR="p" VALUE="0" />
19
      </REGION>
20
      <REGION REF="2">
21
          <N VAR="u" VALUE="0" />
22
          <N VAR="v" VALUE="0" />
23
```

```
<N VAR="w" VALUE="0" />
24
          <D VAR="p" VALUE="0" />
25
      </REGION>
26
      <REGION REF="3">
27
          <D VAR="u" VALUE="0" />
28
          <D VAR="v" VALUE="0" />
29
          <D VAR="w" VALUE="0" />
30
          <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
31
      </REGION>
32
33 </BOUNDARYCONDITIONS>
```

11.10.9.6 Functions:

```
1 <FUNCTION NAME="InitialConditions">

2 <E VAR="u" VALUE="0" />

3 <E VAR="v" VALUE="0" />

4 <E VAR="w" VALUE="0" />

5 <E VAR="p" VALUE="0" />

6 </FUNCTION>
```

11.10.9.7 Results

We can visualise the internal velocity field by applying a volume render filter in ParaView.



Figure 11.14 The solved-for velocity field.

It is possible to visualise the wall shear stress distribution by running the FldAddWSS utility.

11.10.10 finite-strip modeling of flow past flexible cables

As a computationally efficient model, strip theory-based modeling technique has been proposed previously to predict vortex-induced vibration (VIV) for higher Reynolds number flows. In the strip theory-based model, the fluid flow solution is obtained on a series of 2D computational planes (also called as "strips") along the riser's axis direction. These strips then are coupled with each other through structural dynamic model of the riser, and then VIV response prediction is achieved by the strip-structure interactions.In



Figure 11.15 Non-dimensional wall shear stress distribution.

the 2D strip theory, it is assumed that the flow is purely two-dimensional without spanwise correlation, which allows the problems to be split into various 2D planes. A consequence of 2D strip solution under this assumption is that it is unable to reflect the influence of spanwise wake turbulence on the structural dynamics. In order to overcome this shortcoming, we proposed a new module in the framework of Nektar++, in which a spanwise scale is locally allocated to each one of the strips, so that the spanwise velocity correlation is reconstructed in the flow field within each strips. In particular, this model lets the fluid domain to be divided in N strips with thickness ratio of L_z/D and evenly distributed along the spanwise (z) direction. The gap between the neighboring strips, represented by L_g , satisfies relation $L_c = N(L_z + L_g)$. Since the strip in this model has finite scale in the z-direction, we named it as finite strip to distinguish from traditional 2D strip plane. Next, the flow dynamics within each individual strips are modeled by viscous incompressible Navier-Stokes equations, while a tensioned beam model is employed to govern the dynamics of the flexible structures. In this example, we will show how to perform a finite-strip model to predict the vortex-induced vibration responses of flexible cables. Let us consider a vortex-induced vibration of a slender cable with an aspect ratio of $L_z/D=4\pi$, which is immersed in uniform flows at Re=100.

11.10.10.1 Input File

The cable with a mass ratio (defined as the ratio of the total oscillating mass to the mass of displaced fluid) of 1 has diameter D = 1, the 2D mesh is composed of 284 quadrilateral elements. The spanwise direction is split in 16 strips with thickness ratio of $L_c/D=\pi/8$ and one pair of complex Fourier modes for each one of the strips. We will use a sixth order polynomial expansion for the spectral element and the input file for this example is CylFlow_HomoStrip.xml.

1 <E COMPOSITE="C[73]" NUMMODES="6" TYPE="MODIFIED" FIELDS="u,v,w,p" />

To use the finite strip routines we need just to insert a flag of "HomoStrip" in the solver information as below, in addition, we need to specify the types of vibration and support ends for the cables. In this case, the vibration type is specified as VALUE="CONSTRAINED", which means that the cable's vibration is constrained only in the crossflow direction. Other options include VALUE="FREE" and "FORCED", respectively corresponding to the free vibrations in both streamwise and crossflow directions and forced vibration by specified functions given in input file. For the support ends of the cable, another option of VALUE="PINNED-PINNED" is available for the simulations, which satisfies the condition of zero values of displacements on the support ends.

11.10.10.2 Solver information:

```
1<SOLVERINFO>2<I PROPERTY="HomoStrip"</td>VALUE="True"3<I PROPERTY="VibrationType"</td>VALUE="CONSTRAINED"4<I PROPERTY="SupportType"</td>VALUE="FREE-FREE"5</SOLVERINFO>
```

11.10.10.3 Parameters

All the simulation parameters are specified in the section as follows.

```
<PARAMETERS>
1
        <P> LZ
                          = PI/8
                                             </P> <!--thickness ratio-->
2
                                             </P> <!--aspect ratio-->
        <P>LC
                          = 4*PI
3
        <P> A
                          = 0.025
                                             </P>
4
        <P> omega
                          = 1.0
                                             </P>
5
        <P> PROC_Z
                          = 16
                                             </P>
6
        <P> Strip_Z
                          = 16
                                             </P> <!--number of the strips-->
7
        <P> DistStrip
                          = PI/4
                                             </P> <!--distance of the strips-->
8
9
        StructStiff
                          = 0.02
                                             </P>
10
        <P> StructRho
                          = 2.0
                                             </P>
11
        <P> CableTension = 8.82
                                             </P>
        <P> BendingStiff = 0.0
12
                                             </P>
                          = 0.0
        <P> FictDamp
                                             </P>
13
        <P> FictMass
                          = 3.0
                                              </P>
14
    </PARAMETERS>
15
16
```

11.10.10.4 Running the solver

In this example we will run the solver in parallel. We can specify the number of the strips by providing an additional flag to the solver, -nsz. In this example, we will run 16 strips, therefore it would be specified as -nsz 16. The solver can now be run as follows

mpirun -np 16 IncNavierStokesSolver CylFlow_HomoStrip.xml --npz 16 --nsz 16 The simulation results are illustrated in spanwise vorticity contours in Figure 11.16. The wake response of the cable appears as standing wave pattern in the earlier stage and then it transitions into travelling wave response, as shown in this figure.



Figure 11.16 Spanwise vorticity contours in standing wave and travelling wave patterns predicted in finite strip modeling.

11.10.11 2D direct stability analysis of the channel flow

In this example, it will be illustrated how to perform a direct stability analysis using Nektar++. Let us consider a canonical stability problem, the flow in a channel which is confined in the wall-normal direction between two infinite plates (Poiseuille flow) at Reynolds number 7500. This problem is a particular case of the stability solver for the IncNavierStokesSolver.

11.10.11.1 Background

We consider the linearised Incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}'}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{u}' + \mathbf{u}' \cdot \nabla \mathbf{U} = -\nabla p + \nu \nabla^2 \mathbf{u}' + \mathbf{f}$$
(11.39a)

$$\nabla \cdot \mathbf{u}' = 0 \tag{11.39b}$$

We are interested to compute the leading eigenvalue of the system using the Arnoldi method.

11.10.11.2 Geometry

The geometry under consideration is a 2D channel.

11.10.11.3 Mesh Definition

In the GEOMETRY section, the dimensions of the problem are defined. Then, the coordinates (XSCALE, YSCALE, ZSCALE) of each vertices of each element are specified. As this input file defines a two-dimensional problem: ZSCALE = 0.

Edges can now be defined by two vertices.

In the ELEMENT section, the tag T and Q define respectively triangular and quadrilateral element. Triangular elements are defined by a sequence of three edges and quadrilateral elements by a sequence of four edges.

```
1
          <ELEMENT>
               <Q ID="0">
                                                 3 </Q>
2
                              0
                                 1
                                           2
3
               . . .
               <Q ID="47"> 107 108
                                        109
                                                 95 </Q>
4
\mathbf{5}
          </ELEMENT>
6
```

Finally, collections of elements are listed in the COMPOSITE section and the DOMAIN section specifies that the mesh is composed by all the triangular and quadrilateral elements. The other composites will be used to enforce boundary conditions.

```
1
                 <COMPOSITE>
2
               <C ID="0"> Q[0-47] </C>
3
               <C ID="1"> E[17,31,44,57,70,83,96,109,0,19,32,45,58,71,84,97] </C> //
      wall
               <C ID="2"> E[3,6,9,12,15,18] </C>//inflow
4
               <C ID="3"> E[98,100,102,104,106,108] </C> //outflow
\mathbf{5}
          </COMPOSITE>
6
          <DOMAIN> C[0] </DOMAIN>
\overline{7}
8 </GEOMETRY>
9
```

11.10.11.4 Expansion

This section defines the polynomial expansions used on each composites. For this example we will use a 10th order polynomial, i.e. P = 11.

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="11" FIELDS="u,v,p" TYPE="MODIFIED" />
3 </EXPANSIONS>
4
```

11.10.11.5 Time Integration Scheme

For this scheme, an implicit-explicit time-integration scheme is used and we choose one of first order.

```
1 <TIMEINTEGRATIONSCHEME>

2 <METHOD> IMEX </METHOD>

3 <ORDER> 1 </ORDER>

4 </TIMEINTEGRATIONSCHEME>

5
```

11.10.11.6 Solver Info

In this example the EvolutionOperator must be Direct to consider the linearised Navier-Stokes equations and the Driver was set up to ModifiedArnoldi for the solution of the eigenproblem.

```
      1
      <SOLVERINFO>

      2
      <I PROPERTY="SolverType"</td>
      VALUE="VelocityCorrectionScheme" />

      3
      <I PROPERTY="EQTYPE"</td>
      VALUE="UnsteadyNavierStokes" />

      4
      <I PROPERTY="EvolutionOperator"</td>
      VALUE="Direct"
      />

      5
      <I PROPERTY="Projection"</td>
      VALUE="Galerkin"
      />

      6
      <I PROPERTY="Driver"</td>
      VALUE="ModifiedArnoldi"
      />

      7
      </SOLVERINFO>
      8
```

11.10.11.7 Parameters

All the stability parameters are specified in this section.

```
1 <PARAMETERS>
2 <P> TimeStep
       <P> TimeStep = 0.002 </P>
<P> NumSteps = 500 </P>
3
      <P> IO_CheckSteps = 1000 
4
      <P> IO_InfoSteps = 10 </P>
5
      <P> Re = 7500 </P>
<P> Kinvis =1./Re </P>
<P> kdim =16 </P>

6
7
8
      <P> nvec
                          =2 </P>
9
                     =1e-5</P>
=5000 </P>
10
      <P> evtol
       <P> nits
11
12 </PARAMETERS>
13
```

11.10.11.8 Boundary Conditions

```
1 <BOUNDARYREGIONS>
2
              <B ID="0"> C[1] </B>
3
              <B ID="1"> C[2] </B>
              <B ID="2"> C[3] </B>
4
          </BOUNDARYREGIONS>
\mathbf{5}
6
\overline{7}
          <BOUNDARYCONDITIONS>
8
              <REGION REF="0">
                  <D VAR="u" VALUE="0" />
9
                   <D VAR="v" VALUE="0" />
10
                   <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
11
              </REGION>
12
               <REGION REF="1">
13
                   <P VAR="u" VALUE="[2]" />
14
                   <P VAR="v" VALUE="[2]" />
15
16
                   <P VAR="p" VALUE="[2]" />
17
               </REGION>
18
               <REGION REF="2">
                  <P VAR="u" VALUE="[1]" />
19
                   <P VAR="v" VALUE="[1]" />
20
                   <P VAR="p" VALUE="[1]" />
21
               </REGION>
22
           </BOUNDARYCONDITIONS>
23
24
```

11.10.11.9 Function

We need to set up the base flow that can be specified as a function BaseFlow. In case the base flow is not analytical, it can be generated by means of the Nonlinear evolution operator using the same mesh and polynomial expansion. The initial guess is specified in the InitialConditions functions and can be both analytical or a file. In this example it is read from a file.

11.10.11.10 Mask region

If the ModifiedArnoldi driver is used, a subregion of the computational domain can be used to calculate the eigenvalues and eigenvectors by defining an unmask function. This can be used to improve the accuracy and to accelerate convergence if the computational domain is large and contains very coarse elements.

```
1 <FUNCTION NAME="Unmask0">
2 <E VAR="CO" VALUE=" x" />
3 <E VAR="C1" VALUE="-x+1." />
```

```
<E VAR="C2" VALUE=" y+1.5" />
4
      <E VAR="C3" VALUE="-y+1.5" />
5
6 </FUNCTION>
7
8 <FUNCTION NAME="Unmask1">
      <E VAR="CO" VALUE=" x" />
9
      <E VAR="C1" VALUE="-x+1." />
10
      <E VAR="C2" VALUE=" y+1.5" />
11
      <E VAR="C3" VALUE="-y+1.5" />
12
13 </FUNCTION>
14
15 . . .
```

Each function UnmaskO selects elements whose center satisfies C0>0 and C1>0 and C2>0, and so on. More than one unmasked function can be defined, and the finally selected region is the union of all unmasked regions.

If the unmask function is defined, both the unmasked eigenvector and masked eigenvector (with __masked) will be output.

11.10.11.11 Usage

IncNavierStokesSolver ChanStability.xml

11.10.11.12 Results

The stability simulation takes about 250 iterations to converge and the dominant eigenvalues (together with the respective eigenvectors) will be printed. In this case it was found $\lambda_{1,2} = 1.000224 \times e^{\pm 0.24984i}$. Therefore, since the magnitude of the eigenvalue is larger than 1, the flow is absolutely unstable. It is possible to visualise the eigenvectors using the post-processing utilities. The figure shows the profile of the two eigenmode component, which shows the typical Tollmien - Schlichting waves that arise in viscous boundary layers.

11.10.12 2D adjoint stability analysis of the channel flow

In this example, it will be illustrated how to perform an adjoint stability analysis using Nektar++. Let us consider a canonical stability problem, the flow in a channel which is confined in the wall-normal direction between two infinite plates (Poiseuille flow) at Reynolds number 7500

11.10.12.1 Background

We consider the equations:

$$-\frac{\partial \mathbf{u}^*}{\partial t} + (\mathbf{U} \cdot \nabla)\mathbf{u}^* + (\nabla \mathbf{U})^T \cdot \mathbf{u}^* = -\nabla p^* + \frac{1}{Re}\nabla^2 \mathbf{u}$$
(11.40a)






Figure 11.18

$$\nabla \cdot \mathbf{u}^* = 0 \tag{11.40b}$$

We are interested in computing the leading eigenvalue of the system using the Arnoldi method.

11.10.12.2 Geometry & Mesh

The geometry and mesh are the same ones used for the direct stability analysis in the previous example.

11.10.12.3 Time Integration Scheme

For this scheme, an implicit-explicit time-integration scheme is used and we choose one of first order.

```
1 <TIMEINTEGRATIONSCHEME>

2 <METHOD> IMEX </METHOD>

3 <ORDER> 1 </ORDER>

4 </TIMEINTEGRATIONSCHEME>

5
```

11.10.12.4 Solver Info

1

This sections defines the problem solved. In this example the EvolutionOperator must be Adjoint to consider the adjoint Navier-Stokes equations and the Driver was set up to ModifiedArnoldi for the solution of the eigenproblem.

```
2 <SOLVERINFO>

      3
      <I PROPERTY="SolverType"</td>
      VALUE="VelocityCorrectionScheme" />

      4
      <I PROPERTY="EQTYPE"</td>
      VALUE="UnsteadyNavierStokes" />

      5
      <I PROPERTY="EvolutionOperator"</td>
      VALUE="Adjoint"
      />

      6
      <I PROPERTY="Projection"</td>
      VALUE="Galerkin"
      />

      7
      <I PROPERTY="Driver"</td>
      VALUE= "ModifiedArnoldi"
      />

                <I PROPERTY="Proof"
<I PROPERTY="Driver"
   8 </SOLVERINFO>
   9 \end{subequations}
  10
 11 \textbf{Parameters}
 12
                      \begin{lstlisting}[style=XMLStyle]
 13
 14 <PARAMETERS>

      15
      <P> TimeStep
      = 0.002 </P>

      16
      <P> NumSteps
      = 500 </P>

      17
      <P> IO_CheckSteps = 1000</P>

      18
      <P> IO_InfoSteps = 10

      <P> IO_CheckSteps = 10

      19
      <P> Re
      = 7500
      </P>

      20
      <P> Kinvis
      =1./Re
      </P>

      21
      <P> kdim
      =16
      </P>

      22
      <P> nvec
      =2
      </P>

      23
      <P> evtol
      =1e-5
      =1e-5

      24
      <P> nits
      =5000

      25

      =16

 25 </PARAMETERS>
  26
```

11.10.12.5 Boundary Conditions

```
1 <BOUNDARYREGIONS>
                 <B ID="0"> C[1] </B>
2

        3
        <B ID="1">C[2]

        4
        <B ID="2">C[3]

        5
        </BOUNDARYREGIONS>

3
                 <B ID="1"> C[2] </B>
                 <B ID="2"> C[3] </B>
6
          <BOUNDARYCONDITIONS>
7
                <REGION REF="0">
8
                    <D VAR="u" VALUE="0" />
9
                       <D VAR="v" VALUE="0" />
10
                       <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
11
                  </REGION>
12
                  <REGION REF="1">
13
```

```
<P VAR="u" VALUE="[2]" />
14
                   <P VAR="v" VALUE="[2]" />
15
                   <P VAR="p" VALUE="[2]" />
16
17
               </REGION>
               <REGION REF="2">
18
                   <P VAR="u" VALUE="[1]" />
19
                   <P VAR="v" VALUE="[1]" />
20
                   <P VAR="p" VALUE="[1]" />
21
               </REGION>
22
           </BOUNDARYCONDITIONS>
23
24
```

11.10.12.6 Functions

We need to set up the base flow that can be specified as a function BaseFlow. In case the base flow is not analytical, it can be generated by means of the Nonlinear evolution operator using the same mesh and polynomial expansion.

```
1 <FUNCTION NAME="BaseFlow">
2 <F VAR="u,v,p" FILE="ChanStability.bse" />
3 </FUNCTION>
4
```

The initial guess is specified in the InitialConditions functions and can be both analytical or a file. In this example it is read from a file.

```
1 <FUNCTION NAME="InitialConditions">
2 <F VAR="u,v,p" FILE="ChanStability.rst" />
3 </FUNCTION>
4
```

11.10.12.7 Usage

IncNavierStokesSolver ChanStability_adj.xml

11.10.12.8 Results

The equations will then be evolved backwards in time (consistently with the negative sign in front of the time derivative) and the leading eigenvalues will be computed after about 300 iterations. It is interesting to note that their value is the same one computed for the direct problem, but the eigenmodes present a different shape.

11.10.13 2D Transient Growth analysis of a flow past a backward-facing step

In this section it will be described how to perform a transient growth stability analysis using Nektar++. Let us consider a flow past a backward-facing step (figure 11.21). This is an important case because it allows us to understand the effects of separation caused by abrupt changes in the geometry and it is a common geometry in several studies of flow control and turbulence of separated flow.



Figure 11.19



Figure 11.20





11.10.13.1 Background

Transient growth analysis allows us to study the presence of convective instabilities that can arise in stable flows. Despite the fact that these instabilities will decay for a long time (due to the stability of the flow), they can produce significant increases in the energy of perturbations. The phenomenon of transient growth is associated with the non-normality of the linearised Navier-Stokes equations and it consists in computing the perturbation that leads to the highest energy growth for a fixed time horizon.

11.10.13.2 Input Parameters

In the GEOMETRY section, the dimensions of the problem are defined. Then, the coordinates (XSCALE, YSCALE, ZSCALE) of each vertices are specified. As this input file defines a two-dimensional problem: ZSCALE = 0.

Edges can now be defined by two vertices.

```
1 <EDGE>

2 <E ID="0"> 0 1 </E>

3 ....

4 <E ID="828"> 399 394 </E>

5 </EDGE>

6
```

In the ELEMENT section, the tag T and Q define respectively triangular and quadrilateral element. Triangular elements are defined by a sequence of three edges and quadrilateral elements by a sequence of four edges.

```
1 <ELEMENT>
2
         <T ID="0">
                          0
                                 1
                                        2 </T>
3
          . . .
          <T ID="209"> 333
                                        332 </T>
4
                                 314
          <Q ID="210"> 334
                                 335
                                        336
                                                0 </Q>
\mathbf{5}
6
          . . .
          <Q ID="429"> 826
                                 827
7
                                        828
                                              818 </Q>
     </ELEMENT>
8
9
```

Finally, collections of elements are listed in the COMPOSITE section and the DOMAIN section specifies that the mesh is composed by all the triangular and quadrilateral elements. The other composites will be used to enforce boundary conditions.

```
1 <COMPOSITE>
2 <C ID="0"> T[0-209] </C>
```

246 Chapter 11 Incompressible Navier-Stokes Solver

```
3 <C ID="1">Q[210-429] </C>

4 <C ID="2">E[2-3,7,10,16,21,2,...,828] </C>

5 <C ID="3">E[821,823,825,827] </C>

6 <C ID="4">E[722,724,726,728] </C>

7 </COMPOSITE>

8

9 <DOMAIN> C[0,1] </DOMAIN>

10 </GEOMETRY>
```

11.10.13.3 Expansion

For this example we will use a 6th order polynomial, i.e. P = 7:

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="7" FIELDS="u,v,p" TYPE="MODIFIED" />
3 <E COMPOSITE="C[1]" NUMMODES="7" FIELDS="u,v,p" TYPE="MODIFIED" />
4 </EXPANSIONS>
5
```

11.10.13.4 Time Integration Scheme

For this scheme, an implicit-explicit time-integration scheme is used and we choose one of second order.

```
1 <TIMEINTEGRATIONSCHEME>

2 <METHOD> IMEX </METHOD>

3 <ORDER> 2 </ORDER>

4 </TIMEINTEGRATIONSCHEME>

5
```

11.10.13.5 Solver Info

This sections defines the problem solved. In this example the EvolutionOperator must be TransientGrowth and the Driver was set up to Arpack for the solution of the eigenproblem.

1 <SOLVERINFO>

```
<I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes"
<I PROPERTY="EvolutionOperator" VALUE="TransientGrowth"
<I PROPERTY="Projection" VALUE="Galerkin"
<I PROPERTY="COLUTETYDE" VALUE="VelocityCorrectionScheme" VelocityCorrectionScheme" VelocityC
2
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               />
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              />
3
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              />
 4
                           <I PROPERTY="SOLVERTYPE" VALUE="VelocityCorrectionScheme" /> <I PROPERTY="Driver" VALUE="Arpack" />
 5
                            <I PROPERTY="Driver"
                                                                                                                                                                                                                                                                                                            VALUE="Arpack"
 6
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         />
 7 <I PROPERTY="ArpackProblemType" VALUE="LargestMag"
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               />
 8 </SOLVERINFO>
  9
```

1	<parameters></parameters>				
2	FinalTime	= 0.1			
3	<p> TimeStep</p>	= 0.005			

4	<p></p>	NumSteps	=	FinalTime/TimeStep		
5	<p></p>	IO_CheckSteps	=	1/TimeStep		
6	<p></p>	IO_InfoSteps	=	1		
$\overline{7}$	<p></p>	Re	=	500		
8	<p></p>	Kinvis	=	1.0/Re		
9	<p></p>	kdim	=	4		
10	<p></p>	nvec	=	1		
11	<p></p>	evtol	=	1e-4		
12	<td>ETERS></td> <td></td> <td></td> <td></td> <td></td>	ETERS>				
13						

11.10.13.7 Boundary Conditions

```
1 <BOUNDARYREGIONS>
                                      <!-- Wall -->
<!-- Inlet -->
               <B ID="0"> C[2] </B>
\mathbf{2}
               <B ID="1"> C[3] </B>
3
                                      <!-- Outlet -->
4
               <B ID="2"> C[4] </B>
          </BOUNDARYREGIONS>
\mathbf{5}
6
          <BOUNDARYCONDITIONS>
7
               <REGION REF="0">
8
                   <D VAR="u" VALUE="0" />
9
                   <D VAR="v" VALUE="0" />
10
                   <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
11
               </REGION>
12
               <REGION REF="1">
13
                   <D VAR="u" VALUE="0" />
14
                   <D VAR="v" VALUE="0" />
15
                   <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
16
               </REGION>
17
               <REGION REF="2">
18
                   <D VAR="u" VALUE="0" />
19
                   <D VAR="v" VALUE="0" />
20
                   <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
21
               </REGION>
22
23
           </BOUNDARYCONDITIONS>
24
```

11.10.13.8 Functions

We need to set up the base flow that can be specified as a function BaseFlow. In case the base flow is not analytical, it can be generated by means of the Nonlinear evolution operator using the same mesh and polynomial expansion.

The initial guess is specified in the InitialConditions functions and in this case is read from a file.

```
1 <FUNCTION NAME="InitialConditions">
```

248 Chapter 11 Incompressible Navier-Stokes Solver

```
2 <F VAR="u,v,p" FILE="bfs_tg-AR.rst" />
3 </FUNCTION>
4
```

11.10.13.9 Usage

IncNavierStokesSolver bfs_tg-AR.xml

11.10.13.10 Results

The solution will be evolved forward in time using the operator \mathcal{A} , then backward in time through the adjoint operator \mathcal{A}^* . The leading eigenvalue is $\lambda = 3.236204$). This corresponds to the largest possible transient growth at the time horizon $\tau = 1$. The leading eigenmode is shown below. This is the optimal initial condition which will lead to the greatest growth when evolved under the linearised Navier-Stokes equations.



Figure 11.22

It is possible to visualise the transient growth plotting the energy evolution over time if the system is initially perturbed with the leading eigenvector. This analysis was performed for a time horizon $\tau = 60$. It can be seen that the energy grows in time reaching its maximum value at x = 24 and then decays, almost disappearing after 100 temporal units.

11.10.14 BiGlobal Floquet analysis of a of flow past a cylinder

In this example it will be described how to run a BiGlobal stability analysis for a timeperiodic base flow using Nektar++. Let us consider a flow past a circular cylinder at



Figure 11.23



Figure 11.24

 $Re=220~\mathrm{has}$ a 2D time-periodic wake that is unstable to a 3D synchronous "mode A" instability.



Figure 11.25

11.10.14.1 Background

The numerical solution of the fully three- dimensional linear eigenvalue problem is often computationally demanding and may not have significant advantages over performing a direct numerical simulation. Therefore, some simplifications are required; the most radical consist in considering that the base flow depends only on one spatial coordinate, assuming that the other two spatial coordinates are homogenous. While this method offers a good prediction for the instability of boundary layers, it is not able to predict the instability of Hagen-Poiseuille flow in a pipe at all Reynolds numbers. Between a flow that depends upon one and three-spatial directions, it is possible to consider a steady or time-periodic base flow depending upon two spatial directions and impose threedimensional disturbances that are periodic in the the third homogeneous spatial direction. This approach is known as BiGlobal stability analysis and it represents the extension of the classic linear stability theory; let us consider a base flow \mathbf{U} that is function of only two spatial coordinates: mathbfU(x, y, t). The perturbation velocity can **u'** can be expressed in a similar form, but with the dependence on the third homogeneous direction incorporated through the Fourier mode: $\mathbf{u}' = \hat{\mathbf{u}}'(x, y, t)e^{i\beta z}$, where $\beta = 2\pi/L$ and L is the length in the homogeneous direction.

11.10.14.2 Input parameters

In this example we use a mesh of 500 quadrilateral elements with a 6th order polynomial expansion. The base flow has been computed using the Nonlinear evolution operator with appropriate boundary conditions. From its profile, it was possible to determine the periodicity of the flow sampling the velocity profile over time. In order to reconstruct the temporal behaviour of the flow, 32 time slices were considered over one period. Using these data it is possible to set up the stability simulation for a specified β , for example $\beta = 1.7$. Let us note that while the base flow is 2D, the stability simulation that we are performing is 3D.

11.10.14.3 Expansion

In this example we will use a 6th order polynomial expansion, i.e. P = 7.

11.10.14.4 Parameters

1	<timein< th=""><th>NTEGRATIONSCHEME></th><th></th><th></th></timein<>	NTEGRATIONSCHEME>			
2	<method> IMEX </method>				
3	<01	RDER> 2			
4	<td>INTEGRATIONSCHEME></td> <td></td> <td></td>	INTEGRATIONSCHEME>			
5					
6	<solver< td=""><td>RINFO></td><td></td><td></td></solver<>	RINFO>			
$\overline{7}$	<i< td=""><td>PROPERTY="SolverType"</td><td>VALUE="VelocityCorrectionScheme"</td><td>/></td></i<>	PROPERTY="SolverType"	VALUE="VelocityCorrectionScheme"	/>	
8	<i< td=""><td>PROPERTY="EQTYPE"</td><td>VALUE="UnsteadyNavierStokes"</td><td>/></td></i<>	PROPERTY="EQTYPE"	VALUE="UnsteadyNavierStokes"	/>	
9	<i< td=""><td>PROPERTY="EvolutionOperator"</td><td>VALUE="Direct"</td><td>/></td></i<>	PROPERTY="EvolutionOperator"	VALUE="Direct"	/>	
10	<i< td=""><td>PROPERTY="Projection"</td><td>VALUE="Galerkin"</td><td>/></td></i<>	PROPERTY="Projection"	VALUE="Galerkin"	/>	
11	<i< td=""><td>PROPERTY="ModeType"</td><td>VALUE="HalfMode"</td><td>/></td></i<>	PROPERTY="ModeType"	VALUE="HalfMode"	/>	
12	<i< td=""><td>PROPERTY="Driver"</td><td>VALUE= "ModifiedArnoldi"</td><td>/></td></i<>	PROPERTY="Driver"	VALUE= "ModifiedArnoldi"	/>	
13	<i< td=""><td>PROPERTY="HOMOGENEOUS"</td><td>VALUE="1D"</td><td>/></td></i<>	PROPERTY="HOMOGENEOUS"	VALUE="1D"	/>	
14	<td>ERINFO></td> <td></td> <td></td>	ERINFO>			
15					

11.10.14.5 Functions

```
1 <FUNCTION NAME="BaseFlow">
2            <F VAR="u,v,p" FILE="cyinder_floq" />
3 </FUNCTION>
4
```

11.10.14.6 Usage

IncNavierStokesSolver session.xml

252 Chapter 11 Incompressible Navier-Stokes Solver

11.10.14.7 Results

The stability simulation takes about 20 cycles to converge and the leading eigenvalue is $\lambda = 1.2670$ with a growth rate $\sigma = 4.7694e - 02$. The figure below shows the profile of the magnitude of the eigenmode at z = 2.



Figure 11.26

CHAPTER **12**

Linear elasticity solver

12.1 Synopsis

The LinearElasticSolver is a solver for solving the linear elasticity equations in two and three dimensions. Whilst this may be suitable for simple solid mechanics problems, its main purpose is for use for mesh deformation and high-order mesh generation, whereby the finite element mesh is treated as a solid body, and the deformation is applied at the boundary in order to curve the interior of the mesh.

Currently the following equation systems are supported:

Value	Description
LinearElasticSystem	Solves the linear elastic equations. A multi-step variant of the elasticity solver.
	which breaks a given deformation into multiple steps, and applies the deformation to a mesh.

12.1.1 The linear elasticity equations

The linear elasticity equations model how a solid body deforms under the application of a 'small' deformation or strain. The formulation starts with the equilibrium of forces represented by the equation

$$\nabla \cdot \mathbf{S} + \mathbf{f} = \mathbf{0} \quad \text{in} \quad \Omega \tag{12.1}$$

where **S** is the stress tensor and **f** denotes a spatially-varying force. We further assume that the stress tensor **S** incorporates elastic and, optionally, thermal stresses that can be switched on to assist in mesh deformation applications. We assume these can be decomposed so that **S** is written as

 $\mathbf{S} = \mathbf{S}_e + \mathbf{S}_t,$

where the subscripts e and t denote the elastic and thermal terms respectively. We adopt the usual linear form of the elastic stress tensor as

$$\mathbf{S}_e = \lambda \mathrm{Tr}(\mathbf{E}) \,\mathbf{I} + \mu \mathbf{E},$$

where λ and μ are the Lamé constants, **E** represents the strain tensor, and **I** is the identity tensor. For small deformations, the strain tensor **E** is given as

$$\mathbf{E} = \frac{1}{2} \left(\nabla \mathbf{u} + \nabla \mathbf{u}^t \right) \tag{12.2}$$

where **u** is the two- or three-dimensional vector of displacements. The boundary conditions required to close the problem consist of prescribed displacements at the boundary $\partial\Omega$, i.e.

$$\mathbf{u} = \hat{\mathbf{u}} \quad \text{in } \partial\Omega. \tag{12.3}$$

We further express the Lamé constants in terms of the Young's modulus E and Poisson ratio ν as

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}.$$

The Poisson ratio, valid in the range $\nu < \frac{1}{2}$, is a measure of the compressibility of the body, and the Young's modulus E > 0 is a measure of its stiffness.

12.2 Usage

LinearElasticSolver [arguments] session.xml [another.xml] ...

12.3 Session file configuration

12.3.1 Solver Info

- EqType Specifies the PDE system to solve, based on the choices in the table above.
- Temperature Specifies the form of the thermal stresses to use. The choices are:
 - None: No stresses (default).
 - Jacobian: Sets $\mathbf{S}_t = \beta JI$, where β is a parameter defined in the parameters section, J is the elemental Jacobian determinant and I is the identity matrix.
 - Metric: A more complex term, based on the eigenvalues of the metric tensor. This can only be used for simplex elements (triangles and tetrahedra). Controlled again by the parameter β .
- BCType Specifies the type of boundary condition to apply when the IterativeElasticSystem is being used.

- Normal: The boundary conditions are split into NumSteps steps, as defined by a parameter in the session file (default).
- Repeat: As the geometry is updated, re-evaluate the boundary conditions.
 This enables, for example, a cirlce to be rotated continuously.

12.3.2 Parameters

The following parameters can be specified in the **PARAMETERS** section of the session file:

- nu: sets the Poisson ratio ν. Default value: 0.25.
- E: sets the Young's modulus E. Default value: 1.
- beta: sets the thermal stress coefficient β.
 Default value: 1.
- NumSteps: sets the number of steps to use in the case that the iterative elastic system is enabled. Should be greater than 0. *Default value*: 0.

12.4 Examples

12.4.1 L-shaped domain

The first example is the classic L-shaped domain, in which an exact solution is known, which makes it an ideal test case [23]. The domain is the polygon formed from the vertices

$$(-1, -1), (0, -2), (2, 0), (0, 2), (-1, -1), (0, 0).$$

The exact solution for the displacements is known in polar co-ordinates (r, θ) as

$$u_r(r,\theta) = \frac{r^{\alpha}}{2\mu} \left[C_1(C_2 - \alpha - 1)\cos((\alpha - 1)\theta) - (\alpha + 1)\cos((\alpha + 1)\theta) \right]$$
$$u_{\theta}(r,\theta) = \frac{r^{\alpha}}{2\mu} \left[(\alpha + 1)\sin((\alpha + 1)\theta) + C_1(C_2 + \alpha - 1)\sin((\alpha - 1)\theta) \right]$$

where $\alpha \approx 0.544483737...$ is the solution of the equation $\alpha \sin(2\omega) + \sin(2\omega\alpha) = 0$,

$$C_1 = -\frac{\cos((\alpha + 1)\omega)}{\cos((\alpha - 1)\omega)}, \qquad C_2 = 2\frac{\lambda + 2\mu}{\lambda + \mu}$$

with λ and μ being the Lamé constants and $\omega = 3\pi/4$. Boundary conditions are set to be the exact solution and $\mathbf{f} = \mathbf{0}$. The solution has a singularity at the origin, and so in order to test convergence *h*-refinement is required.

256 Chapter 12 Linear elasticity solver



Figure 12.1 Solution of the *u* displacement field for the L-shaped domain.

A simple example of how the linear elastic solver can be set up can be found in the Tests/L-shaped.xml session file in the linear elastic solver directory. A more refined domain with the obtained u solution is shown in figure 12.1. The solver can be run using the command:

LinearElasticSolver L-domain.xml

The obtained solution L-domain.fld can be applied to the mesh to obtain a deformed XML file using the deform module in FieldConvert:

FieldConvert -m deform L-domain.xml L-domain.fld L-domain-deformed.xml

12.4.2 Boundary layer deformation

In this example we use the iterative elastic system to apply a large deformation to a triangular boundary layer mesh of a square mesh $\Omega = [0, 1]^2$. At the bottom edge, we apply a Dirichlet condition $g = \frac{1}{2} \sin(\pi x)$ that is enforced by splitting it into N substeps, so that at each step we solve the system with the boundary condition $g^n(x) = g(x)/N$. The process is depicted in figure 12.2.

The setup is very straightforward. The geometry can be found inside the file Examples/bl-mesh.xml



Figure 12.2 Figures that show the initial domain (left), after 50 steps (middle) and final deformation of the domain (right).

and the conditions inside Examples/bl-conditions.xml. The solver can be set up using the following parameters, with NumSteps denoting N:

```
1 <SOLVERINFO>
2 <I PROPERTY="EQTYPE" VALUE="IterativeElasticSystem" />
3 </SOLVERINFO>
4
5 <PARAMETERS>
6 <P> nu = 0.3 </P>
7 <P> E = 1.0 </P>
8 <P> NumSteps = 100 </P>
9 </PARAMETERS>
```

To identify the boundary that we intend to split up into substeps, we must assign the **WALL** tag to our boundary regions:

```
1 <BOUNDARYCONDITIONS>
     <REGION REF="0">
2
       <D VAR="u" VALUE="0" USERDEFINEDTYPE="Wall" />
3
       <D VAR="v" VALUE="0.5*sin(PI*x)" USERDEFINEDTYPE="Wall" />
4
     </REGION>
5
6
     <REGION REF="1">
\overline{7}
       <D VAR="u" VALUE="0" />
       <D VAR="v" VALUE="0" />
8
     </REGION>
9
  </BOUNDARYCONDITIONS>
10
```

The solver can then be run using the command:

LinearElasticSolver bl-mesh.xml bl-conditions.xml

This will produce a series of meshes bl-mesh-%d.xml, where %d is an integer running between 0 and 100. If at any point the mesh becomes invalid, that is, a negative Jacobian is detected, execution will cease.

CHAPTER **13**

Pulse Wave Solver

13.1 Synopsis

1D modelling of the vasculature (arterial network) represents and insightful and efficient tool for tackling problems encountered in arterial biomechanics as well as other engineering problems. In particular, 3D modelling of the vasculature is relatively expensive. 1D modelling provides an alternative in which the modelling assumptions provide a good balance between physiological accuracy and computational efficiency. To describe the flow and pressure in this network we consider the conservation of mass and momentum applied to an impermeable, deformable tube filled with an incompressible fluid, the nonlinear system of partial differential equations presented in non-conservative form is given by

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{H} \frac{\partial \mathbf{U}}{\partial x} = \mathbf{S}$$
(13.1)
$$\mathbf{U} = \begin{bmatrix} U\\ A \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} U & A\\ \rho \frac{\partial P}{\partial A} & U \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} 0\\ \frac{1}{\rho} \left(\frac{f}{A} - s\right) \end{bmatrix}$$

in which A is the Area (related to pressure), x is the axial coordinate along the vessel, U(x,t) the axial velocity, P(x,t) is the pressure in the tube, ρ is the density and finally f the frictional force per unit length. The unknowns in Eq. 13.1 are U, A and P; hence, we must provide an explicit algebraic relationship to close this system. Typically, closure is provided by an algebraic relationship between A and P. For a thin, viscoelastic tube this is given by

$$P = P_0 + \beta \left(\sqrt{A} - \sqrt{A_0}\right) + \frac{\Gamma}{\sqrt{A}} \frac{\partial A}{\partial t}, \quad \beta = \frac{\sqrt{\pi}Eh}{(1 - \nu^2)A_0}, \quad \Gamma = \frac{2\sqrt{\pi}\varphi h}{3A_0}$$
(13.2)

where P_0 is the external pressure, A_0 is the initial cross-sectional area, E is the Young's modulus, h is the vessel wall thickness, ν is the Poisson's ratio, and φ is the wall viscosity. An empirical law has also been implemented that incorporates strain-stiffening through the parameter α [1]:

$$P = P_0 - \frac{\beta \sqrt{A_0}}{2\alpha} \ln\left[1 - \alpha \ln\left(\frac{A}{A_0}\right)\right] + \frac{\Gamma}{\sqrt{A}} \frac{\partial A}{\partial t}.$$
 (13.3)

Application of Riemann's method of characteristics to Eqs. (13.1) and (13.2) indicates that velocity and area are propagated through the system by forward and backward travelling waves. These waves are reflected and within the network by appropriate treatment of interfaces and boundaries. In the following, we will explain the usage of the blood flow solver on the basis of a single-artery problem and also on an arterial network consisting of 55 arteries.

13.2 Usage

To execute in serial one should type

```
PulseWaveSolver session.xml
```

the solver can also be run in parallel if compiled with MPI sing the command

mpirn -n 2 PulseWaveSolver session.xml

where in this example 2 processors would be used.

13.3 Session file configuration

13.3.1 Pulse Wave Solver mesh connectivity

Typically 1D arterial networks are made up of a connection of different base units: segments, bifurcations and merging junctions. The input format in the PulseWaveSolver means these connections are handle naturally from the mesh topology; hence care must be taken when designing the 1D domain. The figure below outlines the structure of a bifurcation, which is a common reoccurring structure in the vasculature.

To represent this topology in the xml file we specify the following vertices under the section (VERTEX) (the extents are: $-100 \ge x \le 100$ and $-100 \ge y \le 100$)

1 <VERTEX> 2 <V ID="0">-1.000e+02 0.000e+00 0.000e+00</V> 3 <V ID="1">-8.000e+01 0.000e+00 0.000e+00</V> 4 <V ID="2">-6.000e+01 0.000e+00 0.000e+00</V> 5 <V ID="3">-4.000e+01 0.000e+00 0.000e+00</V> 6 <V ID="4">-2.000e+01 0.000e+00 0.000e+00</V>



Figure 13.1 Model of bifurcating artery. The bifurcation is made of three domains and 15 vertices. Vertex V[0] is the inlet and vertices V[10] and V[15] the outlets.

```
<V ID="5"> 0.000e+00 0.000e+00 0.000e+00</V>
\overline{7}
8
    <V ID="6"> 2.000e+01 2.000e+01 0.000e+00</V>
9
   <V ID="7"> 4.000e+01 4.000e+01 0.000e+00</V>
10
    <V ID="8"> 6.000e+01 6.000e+01 0.000e+00</V>
11
    <V ID="9"> 8.000e+01 8.000e+01 0.000e+00</V>
12
    <V ID="10"> 1.000e+02 1.000e+02 0.000e+00</V>
13
14
15
    <V ID="11"> 2.000e+01 -2.000e+01 0.000e+00</V>
16
    <V ID="12"> 4.000e+01 -4.000e+01 0.000e+00</V>
    <V ID="13"> 6.000e+01 -6.000e+01 0.000e+00</V>
17
    <V ID="14"> 8.000e+01 -8.000e+01 0.000e+00</V>
18
    <V ID="15"> 1.000e+02 -1.000e+02 0.000e+00</V>
19
20 </VERTEX>
```

The elements from these vertices are then constructed under the section (ELEMENT) by defining

```
1 <ELEMENT>
  <!-- Parent artery -->
2
   <S ID="0"> 0
                   1 
3
   <S ID="1">
                    2 
4
               1
   <S ID="2">
                    3 
             2
5
   <S ID="3">
                    4 
6
               3
   <S ID="4">
               4
                    5 
7
   <!-- Daughter artery 1 -->
8
9
   <S ID="5">
             5
                    6 
10
   <S ID="6">
               6
                    7 
   <S ID="7">
                    8 
11
               7
   <S ID="8">
12
               8
                    9 
   <S ID="9">
               9
                    10 
13
   <!-- Daughter artery 2 -->
14
              5
   <S ID="11">
                     11 
15
   <S ID="12">
                11
                     12 
16
   <S ID="13">
                12
                     13 
17
   <S ID="14">
                13
                     14 
18
19
   <S ID="15">
               14
                     15
```

20 </ELEMENT>

The composites, which represent groups of elements and boundary regions are defined under the section (COMPOSITE) by

```
1 <COMPOSITE>
                              <!-- Parent artery -->
2 <C ID="0"> S[0-4] </C>
   <C ID="1"> V[0] </C>
                                <!-- Inlet to domain -->
3
4
  <C ID="3"> S[5-9] </C> <!-- Daughter artery 1 -->
5
   <C ID="4"> V[10] </C>
                                <!-- Outlet of daughter artery 1 -->
6
7
   <C ID="6"> S[11-15] </C> <!-- Daughter artery 2 -->
<C ID="8"> V[15] </C>
8
   <C ID="8"> V[15] </C>
                                <!-- Outlet of daughter artery 2 -->
9
10 </COMPOSITE>
```

Each of the segments can be then represented under the section (DOMAIN) by

```
1 <DOMAIN>

2 <D ID="0"> C[0] </D> <!-- Parent artery -->

3 <D ID="1"> C[3] </D> <!-- Daughter artery 1 -->

4 <D ID="2"> C[6] </D> <!-- Daughter artery 2 -->

5 </DOMAIN>
```

We will use the different domains later to define variable material properties and crosssectional areas.

13.3.2 Time Integration Scheme

- (Method) the time-stepping method.
- (Variant) the variant to the method.
- Order the order of the method.
- **FreeParameters** any free parameters required.

13.3.3 Session Info

The PulseWaveSolver is specified through the EquationType option in the session file. This can be set as follows:

- **Projection**: Only a discontinuous projection can be specified using the following option:
 - Discontinuous for a discontinous Galerkin (DG) projection.
- UpwindTypePulse:
 - UpwindPulse

- OutputExtraFields:
 - True returns the wave speed and both characteristics
- PressureArea:
 - Beta for Eq. (13.2)
 - Empirical for Eq. (13.3)

13.3.4 Parameters

The following parameters can be specified in the (PARAMETERS) section of the session file.

- TimeStep is the time-step size;
- FinTime is the final physical time at which the simulation will stop;
- NumSteps is the equivalent of FinTime but instead of specifying the physical final time the number of time-steps is defined;
- IO_CheckSteps sets the number of steps between successive checkpoint files;
- **IO_InfoSteps** sets the number of steps between successive info stats are printed to screen;
- (rho) density of the fluid. Default value = 1.0;
- **(nue)** Poisson's ratio. Default value = 0.5;
- pext external pressure. Default value = 0;
- **pout** outflow pressure to the venous system for the terminal boundary conditions. Default value = 0;
- (h0) wall thickness. Default value = 1.0;

13.3.5 Boundary conditions

In this section we can specify the boundary conditions for our problem. First we need to define the variables under the section (VARIABLES).

1 <VARIABLES> 2 <V ID="0"> A </V> 3 <V ID="1"> u </V> 4 </VARIABLES>

The composites that we want to apply out boundary conditions then need to be defined in the **BOUNDARYREGIONS**, for example if we had three composites (C[1], C[4] and C[8]) that correspond to three vertices of the computational mesh we would define: 1 <BOUNDARYREGIONS> 2 <B ID="0"> C[1] 3 <B ID="1"> C[4] 4 <B ID="2"> C[8] 5 </BOUNDARYREGIONS>

Finally we can specify the boundary conditions on the regions specified under **BOUNDARYREGIONS**.

The Pulse Wave Solver comes with a number of boundary conditions that are unique to this solver. Boundary conditions must be provided for both the area and velocity at the inlets and outlets of the domain. Examples of the different boundary conditions will be provided in the following.

13.3.5.0.1 Inlet boundary condition: The inlet condition may be specified algebraically in four different ways: as an area variation (A-inflow); a velocity profile (U-inflow); a volume flux (Q-inflow); or by prescribing the forward characteristic (TimeDependent). When prescribing a volume flux, it must be specified in the input file via the area, as illustrated below. Note that u = 1.0.

13.3.5.0.2 Terminal boundary conditions: At the outlets of the domain there are four possible boundary conditions: reflection (Terminal), terminal resistance R-terminal, Two element windkessel (CR) CR-terminal, and three element windkessel (RCR) RCR-terminal. An example of the outflow boundary condition of the RCR terminal is given below

Where **RT** is the total peripheral resistance used in the the **R-terminal**, **CR-terminal** and **RCR-terminal** models

13.3.6 Functions

The following functions can be specified inside the (CONDITIONS) section of the session file:

- MaterialProperties: specifies β for each domain.
- A_0 : specifies A_0 for each domain as used in the tube law.

- Viscoelasticity: specifies Γ for each domain. Defaults to zero for every artery if not included.
- StrainStiffening: specifies α for each domain for Eq. (13.3). Defaults to 0.5 for every artery if not included.
- AdvectionVelocity: specifies the advection velocity v.
- InitialConditions: specifies the initial condition for unsteady problems.
- Forcing: specifies the forcing function f

As an example to specify the material properties for each domain in the previous bifurcation example we would enter:

```
1 <FUNCTION NAME="MaterialProperties">
2 <E VAR="beta" DOMAIN="0" VALUE="97" />
3 <E VAR="beta" DOMAIN="1" VALUE="87" />
4 <E VAR="beta" DOMAIN="2" VALUE="233" />
5 </FUNCTION>
```

The values of beta are used in the pressure-area relationship (Eq. (13.2)).

13.4 Examples

13.4.1 Human Vascular Network

The PulseWaveSolver is also capable of handling more complex networks, such as a complete human arterial tree proposed by Westerhof et al. [48]. In this example, we will use the refined data from [43] and set up the network shown in the figure in the right. We will explain how bifurcations are set correctly and how each arterial segment gets its correct physiological data.

First, we will set up the mesh where each arterial segment is represented by one element and two vertices respectively. Then, we will subdivide the mesh into different subdomains by using the *<COMPOSITE>* section. Here, each arterial segment is described by the contained elements and its first and last vertex.

The mesh connectivity is specified during the creation of elements by indicating the starting vertex and ending vertex of each individual artery segment. Shared vertices are used to describe bifurcations, junctions and mergers between different artery segments in the network.

The composites are then used to specify the two adjoining segments of an artery, where the first segment merely allows for description of the connectivity.

```
1 <GEOMETRY DIM="1" SPACE="1">

2 <VERTEX>

3 <V ID="0"> 0.000e+00 0.000e+00 </V> <!-- 1 -->
```



¥.	Artery	Length (<i>cm</i>)	Area (cm^{ω})	$\beta (kg s^{-z} cm^{-z})$	R_t
1	Ascending Aorta	4.0	5.983	97	-
2	Aortic Arch I	2.0	5.147	87	-
3	Brachiocephalic	3.4	1.219	233	-
4	R. Subclavian I	3.4	0.562	423	-
5	R. Carotid	17.7	0.432	516	-
6	R. Vertebral	14.8	0.123	2590	0.906
7	R. Subclavian II	42.2	0.510	466	-
8	R. Radial	23.5	0.106	2866	0.82
9	R. Ulnar I	6.7	0.145	2246	-
0	R. Interosseous	7.9	0.031	12894	0.956
1	R. Ulnar II	17.1	0.133	2446	0.893
2	R. Internal Carotid	17.6	0.121	2644	0.784
3	R. External Carotid	17.7	0.121	2467	0.79
4	Aortic Arch II	3.9	3.142	130	-
5	L. Carotid	20.8	0.430	519	-
6	L. Internal Carotid	17.6	0.121	2644	0.784
7	L. External Carotid	17.7	0.121	2467	0.791
8	Thoracic Aorta I	5.2	3.142	124	-
9	L. Subclavian I	3.4	0.562	416	-
0	Vertebral	14.8	0.123	2590	0.906
ň	L. Subclavian II	42.2	0.510	466	-
2	L Radial	23.5	0.106	2866	0.821
2	L. Illnar I	6.7	0.145	2246	01021
4	L. Interosseous	7.9	0.031	12894	0.956
5	L. Illnar II	17.1	0.133	2446	0.893
6	Intercostals	8.0	0.196	885	0.697
7	Thorngia Aorta II	10.4	3.017	117	0.021
12	Abdominal I	5.3	1.011	167	
0	Coline I	2.0	0.478	475	
0	Collac II	1.0	0.196	1805	-
	Hopotia	1.0	0.120	1149	0.025
9	Castria	7.1	0.102	1142	0.920
12	Gastric	6.0	0.102	1007	0.921
6	Spienic Summine Manufacia	5.3	0.238	500	0.93
4	Abdeminal II	5.9	0.430	209	0.934
6	Abdominal II	2.0	0.220	221	0.0.01
0	L. Renal	3.2	0.332	000	0.801
14	Abdominal III	1.0	1.021	2/8	0.001
8	R. Renal	3.2	0.159	1181	0.861
9	Abdominal IV	10.6	0.697	381	
0	Interior Mesenteric	5.0	0.080	1895	0.918
1	Abdominal V	1.0	0.578	399	-
2	R. Common Illac	5.9	0.328	649	-
3	L. Common Iliac	5.8	0.328	649	-
4	L. External illac	14.4	0.252	1493	
6	L. Internal Illiac	5.0	0.181	3134	0.925
6	L. Femoral	44.3	0.139	2559	-
7	L. Deep Femoral	12.6	0.126	2652	0.885
8	L. Posterior Tibial	32.1	0.110	5808	0.724
9	L. Anterior Tibial	34.3	0.060	9243	0.716
0	R. External Iliac	14.5	0.252	1493	-
1	R. Internal Iliac	5.1	0.181	3134	0.925
2	R. Femoral	44.4	0.139	2559	-
3	R. Deep Femoral	12.7	0.126	2652	0.888
4	L. Posterior Tibial	32.2	0.110	5808	0.724
5	R. Anterior Tibial	34.4	0.060	9243	0.716

4	<v id="1"> 4.000e+00 0.000e+00 0.000e+00</v>
5	
6	<v id="2"> 4.000e+00 0.000e+00 0.000e+00</v> 2
7	<v id="3"> 6.000e+00 0.000e+00 0.000e+00</v>
8	
9	<v id="4"> 4.000e+00 0.000e+00 0.000e+00</v> 3
10	<v id="5"> 7.400e+00 0.000e+00 0.000e+00</v>
11	
12	
13	
14	<v id="108"> 109.100e+00 -45.000e+00 0.000e+00</v> 55
15	<v id="109"> 143.500e+00 -45.000e+00 0.000e+00</v>
16	
17	<element></element>
18	<s id="0"> 0 1</s>
19	<s id="1"> 1 2</s>
20	<s id="2"> 1 4</s>
21	<s id="3"> 2 3</s>
22	<s id="4"> 4 5</s>
23	<s id="5"> 5 6</s>
24	<s id="6"> 5 8</s>
25	<s id="7"> 6 7</s>
26	<s id="8"> 8 9</s>
27	•
28	•

```
29
         <S ID="106"> 103 108 
30
         <S ID="107"> 108 109 
31
        <S ID="108"> 85 98 
32
   <ELEMENT>
33
   <COMPOSITE>
34
        <C ID="0"> S[0] </C> <!-- 1 -->
35
36
         <C ID="1"> V[0] </C>
         <C ID="2"> V[1] </C>
37
38
         <C ID="3"> S[1,3] </C> <!-- 2 -->
39
         <C ID="4"> V[2] </C>
40
         <C ID="5"> V[3] </C>
41
42
43
         <C ID="6"> S[2,4] </C> <!-- 3 -->
44
         <C ID="7"> V[4] </C>
45
         <C ID="8"> V[5] </C>
46
            .
47
            .
48
         <C ID="162"> S[106,107] </C> <!-- 55 -->
49
         <C ID="163"> V[108] </C>
50
         <C ID="164"> V[109] </C>
51
52
     </COMPOSITE>
53 </GEOMETRY>
```

Then the choice of polynomial order, solver information, area of the arteries and other parameters are specified.

```
1 <EXPANSIONS>
     <E COMPOSITE="C[0]" NUMMODES="5" FIELDS="A,u" TYPE="MODIFIED" />
2
3
      <E COMPOSITE="C[3]" NUMMODES="5" FIELDS="A,u" TYPE="MODIFIED" />
4
         . . .
\mathbf{5}
      <E COMPOSITE="C[162]" NUMMODES="5" FIELDS="A,u" TYPE="MODIFIED" />
6
7 </EXPANSIONS>
8
9 <CONDITIONS>
10
      <PARAMETERS>
11
12
        <P> TimeStep= 1e-4</P><P> FinTime= 1.0</P><P> NumSteps= FinTime/TimeStep</P>
13
14
15
        <P> I0_CheckSteps = NumSteps/50
                                                    </P>
16
                        = 0.126
= 0.110
= 0.060
17
            . . .
        <P> A53
                                                   </P>
18
       <P> A54
19
                                                   </P>
         <P> A55
                                                    </P>
20
     </PARAMETERS>
21
22
      <TIMEINTEGRATIONSCHEME>
23
          <METHOD> RungeKutta </METHOD>
24
          <VARIANT> SSP </VARIANT>
25
          <ORDER> 2 </ORDER>
26
```

```
27
      </TIMEINTEGRATIONSCHEME>
28
29
      <SOLVERINFO>
          <I PROPERTY="EQTYPE" VALUE="PulseWavePropagation" />
30
          <I PROPERTY="Projection" VALUE="DisContinuous" />
31
          <I PROPERTY="TimeIntegrationMethod" VALUE="RungeKutta2_ImprovedEuler" />
32
33
          <I PROPERTY="UpwindTypePulse" VALUE="UpwindPulse"/>
34
      </SOLVERINFO>
35
      <VARIABLES>
36
          <V ID="0"> A </V>
37
          <V ID="1"> u </V>
38
      </VARIABLES>
39
```

The vertices where the network terminates are specified as boundary regions based on their subsequent composite ids.



In the boundary conditions section the inflow and outflow conditions are set up. Here we use an inflow boundary condition for the area at the beginning of the ascending aorta taken from [43] and plotted on the right. Potential choices for inflow boundary conditions include Q-Inflow and Time-Dependent inflow. The outflow conditions for the terminal regions of the network could be specified by different models including eTerminal, R, CR, RCR and Time-Dependent outflow.

```
1 <BOUNDARYCONDITIONS>
      <REGION REF="0"> <!-- Inflow -->
\mathbf{2}
          <D VAR="A" USERDEFINEDTYPE="TimeDependent"</pre>
3
              VALUE="5.983*(1+0.597*(sin(6.28*t + 0.628) - 0.588)*
4
                   (1./(1+exp(-2*200*(sin(6.28*t + 0.628) - 0.588)))))" />
5
          <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="0.0" />
6
      </REGION>
\overline{7}
      <REGION REF="1">
8
          <D VAR="A" USERDEFINEDTYPE="TimeDependent" VALUE="A6" />
9
```

```
<D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="0.0" />
10
      </REGION>
11
12
      <REGION REF="2">
          <D VAR="A" USERDEFINEDTYPE="TimeDependent" VALUE="A8" />
13
          <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="0.0" />
14
      </REGION>
15
      <REGION REF="3">
16
          <D VAR="A" USERDEFINEDTYPE="TimeDependent" VALUE="A10" />
17
          <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="0.0" />
18
     </REGION>
19
20
      . . . .
      <REGION REF="28">
21
          <D VAR="A" USERDEFINEDTYPE="TimeDependent" VALUE="A55" />
22
23
          <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="0.0" />
24
      </REGION>
25 </BOUNDARYCONDITIONS>
```

Again, for the initial conditions we start our simulation from static equilibrium conditions $A = A_0$ and for u being initially at rest. The following lines show how we specify A_0 and β for different arterial segments.

```
1 <FUNCTION NAME="InitialConditions">
     <E VAR="A" DOMAIN="0" VALUE="5.983" />
2
      <E VAR="u" DOMAIN="0" VALUE="0.0" />
3
4 </FUNCTION>
5
    . . .
6 <FUNCTION NAME="InitialConditions">
     <E VAR="A" DOMAIN="54" VALUE="A55" />
7
     <E VAR="u" DOMAIN="54" VALUE="0.0" />
8
9 </FUNCTION>
10
11 <FUNCTION NAME="A O">
      <E VAR="A_0" DOMAIN="0" VALUE="A1" />
12
13
         . . .
      <E VAR="A_0" DOMAIN="54" VALUE="A55" />
14
15 </FUNCTION>
16
17 <FUNCTION NAME="MaterialProperties">
      <E VAR="beta" DOMAIN="0" VALUE="97" />
18
19
      <E VAR="beta" DOMAIN="54" VALUE="9243" />
20
21 </FUNCTION>
```

Our simulation is started as described before and the results show the time history for the conservative variables A and u, as well as for the characteristic variables W1 and W2 at the beginning of the ascending aorta (Artery 1). We can see that physically correct the shape of the inflow boundary condition appears in the forward traveling characteristic W1. As we do not have a terminal resistance at the outflow, one would normally expect W2 to be constant. However this is not the case, as bifurcations cause reflections if the radii of parent and daughter vessels are not well matching, leading to changes in W2. The shapes of A and u result from this facts and show the values for the physiological variables during one cardiac cycle. We may annotate that this values slightly differ from



in vivo measurements due to the missing terminal resistance, which will be added in future.

These short examples should give an insight to the functionality of our PulseWaveSolver and show that results such as luminal area and pressure within the artery can be simulated. These results can contribute to understanding the physiology of the human vascular system and they can be used for patient-specific planning of medical interventions.

13.4.2 Stented Artery

In the following we will explain the usage of the Pulse Wave solver to model the flow and pressure variation through a stented artery - a cardiovascular procedure in which a small mesh tube is inserted into an artery to restore blood flow through a constricted region. Due to the implantation of the stent this region will have different material properties compared to the the surrounding unstended tissue; hence will influence the propagation of waves through this system. The stent scenario to be modelled is a straight arterial segment with a stent situated between $x = a_1$ and $x = a_2$ as shown below.



Figure 13.2 Model of straight artery with a stent in the middle.

270 Chapter 13 Pulse Wave Solver

13.4.2.0.1 Geometry: In the following we describe the geometry setup for modelling 1D flow in a stent. This is done by defining vertices, elements and composites. The vertices of the domain are shown below, consisting of 30 elements (Ω) and 31 vertices (V[n]).



Figure 13.3 1D arterial domain consisting of 30 elements and 31 vertices.

To represent the above in the xml file, we define 31 vertices as follows:

```
1 <VERTEX>

2 <V ID="0"> 0.000e+00 0.000e+00 0.000e+00</V>

3 . .

4 . .

5 . .

6 <V ID="30">30.000e+00 0.000e+00 0.000e+00</V>

7 </VERTEX>
```

and the connectivity of these vertices to make up the 30 elements:

```
1 <ELEMENT>

2 <S ID="0"> 0 1 

3 .

4 .

5 .

6 <S ID="29"> 29 30 

7 </ELEMENT>
```

These elements are combined to three different composites (shown below): composite 0 represents all the elements; composite 1 the inflow boundary and composite 2 the outflow boundary.



Figure 13.4 Three composites (C[0], C[1] and C[2]) for the stunted artery.

The above composites are specified as follows:

```
1 <COMPOSITE>

2 <C ID="0"> S[0-29] </C>

3 <C ID="1"> V[0] </C>

4 <C ID="2"> V[30] </C>

5 </COMPOSITE>
```

Finally the domain is specified by the first composite by

```
1 <DOMAIN>
2 <D ID="0"> C[0] </D>
3 </DOMAIN>
```

13.4.2.0.2 Expansion: For the expansions we use 4th-order polynomials which define our two variables A and u on the domain.

```
1 <EXPANSIONS>
2 <E COMPOSITE="C[0]" NUMMODES="5" FIELDS="A,u" TYPE="MODIFIED" />
3 </EXPANSIONS>
```

13.4.2.0.3 Time Integration Scheme: For the Time Integration Scheme we use a basic Forward Euler method.

```
1 <TIMEINTEGRATIONSCHEME>

2 <METHOD> ForwardEuler </METHOD>

3 <ORDER> 1 </ORDER>

4 </TIMEINTEGRATIONSCHEME>
```

13.4.2.0.4 Solver Information: The Discontinuous Galerkin Method is used as projection scheme and the time-integration is performed by a simple Forward Euler scheme. A full list of possible time integration scheme is given in the parameter section of the Pulse Wave Solver

```
1 <SOLVERINFO>
2 <I PROPERTY="EQTYPE" VALUE="PulseWavePropagation" />
3 <I PROPERTY="Projection" VALUE="DisContinuous" />
4 <I PROPERTY="TimeIntegrationMethod" VALUE="ForwardEuler" />
5 <I PROPERTY="UpwindTypePulse" VALUE="UpwindPulse"/>
6 </SOLVERINFO>
```

13.4.2.0.5 Parameters: Parameters used for the simulation are taken from [43]

1	<param< th=""><th>ETERS></th><th></th><th></th><th></th></param<>	ETERS>			
2	<p></p>	TimeStep	=	2e-6	
3	<p></p>	FinTime	=	0.25	
4	<p></p>	NumSteps	=	FinTime/TimeStep	
5	<p></p>	IO_CheckSteps	=	NumSteps/50	
6	<p></p>	IO_InfoSteps	=	100	
7	<p></p>	Т	=	0.33	
8	<p></p>	hO	=	1.0	
9	<p></p>	rho	=	1.0	
10	<p></p>	nue	=	0.5	
11	<p></p>	pext	=	0.0	
12	<p></p>	a1	=	10.0	
13	<p></p>	a2	=	20.0	
14	<p></p>	kappa	=	100.0	
15	<p></p>	YO	=	1.9099e+5	
16	<p></p>	k	=	2	
17	<p></p>	k1	=	200	
18	<th>AMETERS></th> <th></th> <th></th> <th></th>	AMETERS>			

272 Chapter 13 Pulse Wave Solver

13.4.2.0.6 Boundary conditions: At the inflow we apply a pressure boundary condition as shown in the figure below. This condition models the pressure variation during one heartbeat. A simple absorbing outflow boundary condition is applied the right end of the tube.



Figure 13.5 Pressure profile applied at the inlet of the artery

These are defined in the xml file as follows,

```
1 <BOUNDARYREGIONS>
     <B ID="0"> C[1] </B>
\mathbf{2}
     <B ID="1"> C[2] </B>
3
4 </BOUNDARYREGIONS>
5
6 <BOUNDARYCONDITIONS>
\overline{7}
     <REGION REF="0">
         <D VAR="A" USERDEFINEDTYPE="TimeDependent" VALUE=</pre>
8
        "(2000*sin(2*PI*t/T)*1./(1+exp(-2*k1*(T/2-t))-pext)/451352+1)^2" />
9
         <D VAR="u" USERDEFINEDTYPE="TimeDependent" VALUE="1.0" />
10
     </REGION>
11
     <REGION REF="1">
12
         <D VAR="A" VALUE="1.0" />
13
         <D VAR="u" VALUE="1.0" />
14
     </REGION>
15
16 </BOUNDARYCONDITIONS>
```

The simulation starts from the static equilibrium of the vessel with normalised area and velocity.

13.4.2.0.7 Functions: The stent is introduced by applying a variable material properties function (β - see Eq. (13.2)) along the vessel in the x direction, shown graphically below and defined in the xml file by



Figure 13.6 material property variation along the artery. The stiff region in the middle represents the stent.

13.4.2.1 Simulation

The simulation is started by running

PulseWaveSolver Test_1.xml

It will take about 60 seconds on a 2.4GHz Intel Core 2 Duo processor and therefore is computationally realisable at every clinical site.

13.4.2.2 Results

As a result we get a 3-dimensional interpretation of the aortic cross-sectional area varying in axial direction both for the stented and non-stented vessel. In case of the stent, the rigid metal mesh will restrict the deformation of the area in that specific part of the artery compared to the normal vessel (Fig. 13.7).

Also, if we look at the pressure at three points within the artery (P, M, D) we will recognize that there are major differences between the stented and normal vessel. While in the normal vessel (left) the pressure wave applied at the inflow is propagated without any losses, this does not hold for the stented artery (right). Here, the stiffening at the stent causes reflections and thus there are losses for total pressure at the medial (M) and distal (D) point.



Figure 13.7



13.5 Further Information

The PulseWaveSolver has been developed with contributions by various students and researchers at the Department of Aeronautics, Imperial College London. Further information on the solver and its underlying mathematical framework can be found in [40, 39].

13.6 Future Development

The PulseWaveSolver is a useful tool for computational modelling of one-dimensional blood flow in the human body. However, there are several ideas for future development which include:

- 1. Inclusion of a pre-processor and post-processor.
- 2. Profiling the code to improve performance.
- 3. Cleaning up the input file to make the input format more user-friendly.
- 4. Modelling of valves and alternative pressure-area laws for models of venous flow.
- 5. Incorporating a model of the heart.

13.7 References

[1] Reavette R, Sherwin SJ, Tang MX, Weinberg PW. Comparison of arterial wave intensity analysis by pressure-velocity and diameter-velocity methods in a virtual population of adult aubjects. *Journal of Engineering in Medicine*. 2020.

[2] Alastruey J. Numerical modelling of pulse wave propagation in the cardiovascular system: development, validation and clinical applications. *PhD thesis, Imperial College London.* 2006.

CHAPTER **14**

Shallow Water Solver

14.1 Synopsis

The ShallowWaterSolver is a solver for depth-integrated wave equations of shallow water type. Presently the following equations are supported:

Value	Description
LinearSWE	Linearized SWE solver in primitive variables
	(constant still water depth)
NonlinearSWE	Nonlinear SWE solver in conservative variables
	(constant still water depth)

14.1.1 The Shallow Water Equations

The shallow water equations (SWE) is a two-dimensional system of nonlinear partial differential equations of hyperbolic type that are fundamental in hydraulic, coastal and environmental engineering. In deriving the SWE the vertical velocity is considered negligible and the horizontal velocities are assumed uniform with depth. The SWE are hence valid when the water depth can be considered small compared to the characteristic length scale of the problem, as typical for flows in rivers and shallow coastal areas. Despite the limiting restrictions the SWE can be used to describe many important phenomena, for example storm surges, tsunamis and river flooding.

The two-dimensional SWE is stated in conservation form as

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(U) = \mathbf{S}(U)$$

where $\mathbf{F}(U) = [\mathbf{E}(U), \mathbf{G}(U)]$ is the flux vector and the vector of conserved variables read $\mathbf{U} = [H, Hu, Hv]^{\mathrm{T}}$. Here $H(\mathbf{x}, t) = \zeta(\mathbf{x}, t) + d(\mathbf{x})$ is the total water depth, $\zeta(\mathbf{x}, t)$ is the free surface elevation and $d(\mathbf{x})$ is the still water depth. The depth-averaged velocity is
denoted by $\mathbf{u}(\mathbf{x},t) = [u,v]^{\mathrm{T}}$, where u and v are the velocities in the x- and y-directions, respectively. The content of the flux vector is

$$\mathbf{E}(U) = \begin{bmatrix} Hu \\ Hu^2 + gH^2/2 \\ Huv \end{bmatrix}, \qquad \mathbf{G}(U) = \begin{bmatrix} Hv \\ Hvu \\ Hv^2 + gH^2/2 \end{bmatrix},$$

in which g is the acceleration due to gravity. The source term $\mathbf{S}(U)$ accounts for, e.g., forcing due to bed friction, bed slope, Coriolis force and higher-order dispersive effects (Boussinesq terms). In the distributed version of the ShallowWaterSolver only the Coriolis force is included.

14.2 Usage

ShallowWaterSolver session.xml

14.3 Session file configuration

14.3.1 Time Integration Scheme

- (Method) the time-stepping method.
- **Variant** the variant to the method.
- Order the order of the method.
- **FreeParameters** any free parameters required.

14.3.2 Solver Info

- Eqtype : Specifies the equation to solve. This should be set to NonlinearSWE.
- UpwindType
- Projection

14.3.3 Parameters

• Gravity

14.3.4 Functions

- Coriolis: Specifies the Coriolis force (variable name: 'f')
- WaterDepth: Specifies the water depth (variable name: 'd')

278 Chapter 14 Shallow Water Solver

14.4 Examples

14.4.1 Rossby modon case

This example, provided in RossbyModon_Nonlinear_DG.xml is of a discontinuous Galerkin simulation of the westward propagation of an equatorial Rossby modon.

14.4.1.1 Input Options

For what concern the ShallowWaterSolver the **<TIMEINTEGRATIONSCHEME>** and **<SOLVERINFO>** section allows us to specify the solver, the type of projection (continuous or discontinuous), the explicit time integration scheme to use and (in the case the discontinuous Galerkin method is used) the choice of numerical flux. A typical example would be:

```
1 <TIMEINTEGRATIONSCHEME>
     <METHOD> RungeKutta </METHOD>
2
     <ORDER> 4 </ORDER>
3
4 </TIMEINTEGRATIONSCHEME>
1 <SOLVERINFO>
   <I PROPERTY="EqType"
                                                                      />
                                        VALUE="NonlinearSWE"
2
  <I PROPERTY="Projection"
                                        VALUE="DisContinuous"
                                                                      />
3
   <I PROPERTY="UpwindType"
                                        VALUE="HLLC"
4
                                                                      />
5 </SOLVERINFO>
```

In the **<PARAMETERS>** section we, in addition to the normal setting of time step etc., also define the acceleration of gravity by setting the parameter "Gravity":

1	<parameters></parameters>		
2	<p> TimeStep</p>	= 0.04	
3	<p> NumSteps</p>	= 1000	
4	I0_CheckSteps	= 100	
5	I0_InfoSteps	= 100	
6	<p> Gravity</p>	= 1.0	
7			

We specify f which is the Coriolis parameter and d denoting the still water depth as analytic functions:

```
1 <FUNCTION NAME="Coriolis">
2  <E VAR="f" VALUE="0+1*y" />
3 </FUNCTION>
4
5 <FUNCTION NAME="WaterDepth">
6  <E VAR="d" VALUE="1" />
7 </FUNCTION>
```

Initial values and boundary conditions are given in terms of primitive variables (please note that also the output files are given in terms of primitive variables). For the discontinuous Galerkin we typically enforce any slip wall boundaries weakly using symmetry technique. This is given by the <code>USERDEFINEDTYPE="Wall"</code> choice in the <code><BOUNDARYCONDITIONS></code> section:

```
1 <BOUNDARYCONDITIONS>
2 <REGION REF="0">
3 <D VAR="eta" USERDEFINEDTYPE="Wall" VALUE="0" />
4 <D VAR="u" USERDEFINEDTYPE="Wall" VALUE="0" />
5 <D VAR="v" USERDEFINEDTYPE="Wall" VALUE="0" />
6 </REGION>
7 </BOUNDARYCONDITIONS>
```

14.4.1.2 Running the code

After the input file has been copied to the build directory of the ShallowWaterSolver the code can be executed by:

./ShallowWaterSolver Rossby_Nonlinear_DG.xml

14.4.1.3 Post-proceesing

After the final time step the solver will write an output file RossbyModon_Nonlinear_DG.fld. We can convert it to tecplot format by using the FieldConvert utility. Thus we execute the following command:

```
FieldConvert RossbyModon_Nonlinear_DG.xml RossbyModon_Nonlinear_DG.fld \
    RossbyModon_Nonlinear_DG.dat
```

This will generate a file called RossbyModon_Nonlinear_DG.dat that can be loaded directly into tecplot:



Part IV Reference

CHAPTER **15**

Optimisation

One of the most frequently asked questions when performing almost any scientific computation is: how do I make my simulation faster? Or, equivalently, why is my simulation running so slowly?

The spectral element method is no exception to this rule. The purpose of this section is to highlight some of the easiest parameters that can be tuned to attain optimum performance for a given simulation.

Details are kept as untechnical as possible, but some background information on the underlying numerical methods is necessary in order to understand the various options available and the implications that they can have on your simulation.

In the current version of the library we now attempt to turn on some of these optimisations automatically and so you will likely observe a <code>session.opt</code> file appear in your directory which can be viewed to see what settings are being selected.

15.1 Collections and MatrixFree operations

The Collections and associated MatrixFree libraries adds optimisations to perform certain elemental operations collectively by applying an operator using either matrixmatrix or unrolled matrix free operations, rather than a sequence of matrix-vector multiplications. Certain operators benefit more than other from this treatment, so the following implementations are available:

- StdMat: Perform operations using collated matrix-matrix type elemental operation.
- SumFac: Perform operation using collated matrix-matrix type sum factorisation (i.e. direction by direction) /operations.
- IterPerExp: Loop through elements, performing matrix-vector operation utilising StdRegions building blocks.

- MatrixFree: call matrix free implementations that can utilise vectorisation by performing SIMD (single instruction multiple data) operations over multiple elements concurrently.
- NoCollections: Use the original LocalRegions implementation to perform the operation which involves looping over the elements which may subsequently call the StdRegions implementations.

All configuration relating to Collections is given in the **COLLECTIONS** Xml element within the **(NEKTAR)** XML element.

15.1.1 Automatic tuning and the -writeoptfile command line option

By default we now try to select the optimal choice of implementation when you first run a solver. If you run the solver in verbose mode you will observe an output of the form:

```
1 Collection Implementation for Tetrahedron ( 4 4 4 ) for ngeoms = 428
\mathbf{2}
              Op.
                   :
                          opt. Impl.
                                            (IterLocExp, IterStdExp, StdMat,
                   MatrixFree)
      SumFac.
                                            (0.000344303, 0.000336822, 0.000340444,
           BwdTrans:
3
                          MatFree
      0.000185503, 6.80494e-05)
           Helmholtz:
                                            (0.00227906, 0.00481378,
                          MatFree
4
            , 0.000374155)
           IPWrtBase:
                          MatFree
                                            (0.000364424, 0.000318054, 0.000291705,
5
      0.000138584, 8.37257e-05)
6
           IPWrtDBase:
                          MatFree
                                            (0.00378674, 0.00308545, 0.00100464,
      0.000653242, 0.000283372)
                                            (0.000881537, 0.000774604, 0.00407994,
7
           PhysDeriv :
                          MatFree
      0.000540257, 0.000185529)
8 Collection Implemenation for Prism ( 4 4 4 ) for ngeoms = 136
                   :
                                            (IterLocExp, IterStdExp, StdMat,
9
              Op.
                         opt. Impl.
      SumFac,
                   MatrixFree)
          BwdTrans:
                                            (0.000131559, 0.000130099, 0.000237854,
                          MatFree
10
      8.40501e-05, 2.78436e-05)
                                            (0.000988519, 0.00133484,
           Helmholtz:
                          MatFree
11
             , 0.000166906)
           IPWrtBase:
                          MatFree
                                            (0.000113946, 0.000105544, 0.00022007,
12
      5.74802e-05, 3.18842e-05)
           IPWrtDBase:
                          MatFree
                                            (0.00148209, 0.000717362, 0.000885148,
13
      0.000257414, 0.00011241)
14
           PhysDeriv :
                          MatFree
                                            (0.000295485, 0.000247841, 0.00186362,
      0.000219107, 7.38712e-05)
```

This shows the selected collection operation, in this case MatrixFree, for the different operators implmentations and the various approaches. Note that IterLocExp is equivalent to NoCollection and IterStdExp is directly related to the IterPerExp option.

This choice of optimisation is then written into a file called **Session.opt** where **Session** is name of the user defined xml file. We note that the optimal choice is currently based on the volumetric elements of the mesh (i.e. Tris and Quads in 2D and Tets, Pyramids, Prisms and Hexs in 3D) and not on the boundary conditions. In the case of a parallel

run the root process will write the file based on the optimisation on this processor. In the case one type of element is not on the root processor the output form the highest rank process with this element shape will be outputted. Once this file is present it will be read directly rather than re-running the auto-tuning.

15.1.2 Manually selecting the COLLECTIONS section

The COLLECTIONS section can be set manually within the COLLECTIONS tag as shown in the following example. Note this section can be added in either the input Session.xml file or the Session.opt file that is auto-generated.

Different implementations may be chosen for different element shapes and expansion orders. Specifying * for ORDER sets the default implementation for any expansion orders not explicitly defined.

```
1 <COLLECTIONS>
     <OPERATOR TYPE="BwdTrans">
2
         <ELEMENT TYPE="T" ORDER="*"
                                        TMPTYPE="IterPerExp" />
3
          <ELEMENT TYPE="T" ORDER="1-5" IMPTYPE="StdMat" />
4
     </OPERATOR>
5
6
     <OPERATOR TYPE="IProductWRTBase">
          <ELEMENT TYPE="Q" ORDER="*" IMPTYPE="SumFac" />
7
     </OPERATOR>
8
9 </COLLECTIONS>
```

15.1.2.1 Default implementation

The default implementation for all operators may be chosen through setting the DEFAULT attribute of the COLLECTIONS XML element to one of StdMat, SumFac, IterPerExp, NoCollection or Matrixfree. The StdMat sets up a standard matrix for the element in the collection as the underlying operator. The following uses the collated matrix-matrix type elemental operation for all operators and expansion orders:

```
1 <COLLECTIONS DEFAULT="StdMat" />
```

The NoCollection option iterates over each expansion in the local region calling the local operator which is implemented in a sum factorization method within the element. The IterPerExp holds a standard expansion and then also holds an expanded copy of the geometric factors within the collection operator. SumFac is a sum factorization implementation which undertakes each direction of the method over multiple elements in the collection. Finally MatrixFree implements a vectorisation suitable version of the sum factorization which has minimal memory movement but requires some initial data re-orientation when vectorising over multiple elements.

15.1.2.2 Auto-tuning

The choice of implementation for each operator, for the given mesh and expansion orders, can be selected selected automatically through an attribute in the **COLLECITON** section. To enable this, add the following to the *Nektar++* session file:

284 Chapter 15 Optimisation

```
1 <COLLECTIONS DEFAULT="auto" />
```

This will collate elements from the given mesh and given expansion orders, run and time each implementation strategy in turn, and select the fastest performing case. Note that the selections will be mesh- and order- specific. The selections made via auto-tuning are output if the **-verbose** command-line switch is given.

15.1.3 Collection size

The maximum number of elements within a single collection can be enforced using the $(\tt MAXSIZE)$ attribute.

CHAPTER **16**

Command-line Options

--verbose

Displays extra info.

--version

Displays software version, and source control information if applicable.

--help

Displays help information about the available command-line options for the executable.

--parameter [key]=[value]

Override a parameter (or define a new one) specified in the XML file.

--solverinfo [key]=[value]

Override a solverinfo (or define a new one) specified in the XML file.

--io-format [format]

Determines the output format for writing Nektar++ field files that are used to store, for example, checkpoint and solution field files. The default for format is Xml, which is an XML-based format, which is written as one file per process. If Nektar++ is compiled with HDF5 support, then an alternative option is Hdf5, which will write one file for all processes and can be more efficient for very large-scale parallel jobs.

--npx [int]

When using a fully-Fourier expansion, specifies the number of processes to use in the x-coordinate direction.

--npy [int]

When using a fully-Fourier expansion or 3D expansion with two Fourier directions, specifies the number of processes to use in the y-coordinate direction. **286** Chapter 16 Command-line Options

--npz [int]

When using Fourier expansions, specifies the number of processes to use in the z-coordinate direction.

--part-info

Prints detailed information about the generated partitioning, such as number of elements, number of local degrees of freedom and the number of boundary degrees of freedom.

--part-only [int]

Partition the mesh only into the specified number of partitions, write to file and exit. This can be used to pre-partition a very large mesh on a single high-memory node, prior to being executed on a multi-node cluster.

--use-metis

Forces the use of METIS for mesh partitioning. Requires the **NEKTAR_USE_METIS** option to be set.

--use-scotch

Forces the use of Scotch for mesh partitioning. If Nektar++ is compiled with METIS support, the default is to use METIS.

--use-hdf5-node-comm

Partition the Hdf5-format mesh in parallel to avoid one single thread runs out of memory in serial partitioning.

--set-start-time [float]

Set the starting time of the simulation. This overwrites the time in the file-type initial condition.

--set-start-chknumber [int]

Set the starting number of the checkpoint file. This overwrites the checkpoint number in the file-type initial condition.

CHAPTER **17**

Frequently Asked Questions

17.1 Compilation and Testing

Q. I compile Nektar++ successfully but, when I run ctest, all the tests fail. What might be wrong?

On Linux or Mac, if you compile the ThirdParty version of Boost, rather than using version supplied with your operating system (or MacPorts on a Mac), the libraries will be installed in the ThirdParty/dist/lib subdirectory of your Nektar++ directory. When Nektar++ executables are run, the Boost libraries will not be found as this path is not searched by default. To allow the Boost libraries to be found set the following environmental variable, substituting **\$NEKTAR_HOME** with the absolute path of your Nektar++ directory:

• On Linux (sh, bash, etc)

export LD_LIBRARY_PATH=\${NEKTAR_HOME}/ThirdParty/dist/lib

or (csh, etc)

setenv LD_LIBRARY_PATH \${NEKTAR_HOME}/ThirdParty/dist/lib

• On Mac

export DYLD_LIBRARY_PATH=\${NEKTAR_HOME}/ThirdParty/dist/lib

Q. How to I compile Nektar++ to run in parallel?

Parallel execution of all Nektar++ solvers is available using MPI. To compile using MPI, enable the NEKTAR_USE_MPI option in the CMake configuration. On recent versions of

MPI, the solvers can still be run in serial when compiled with MPI. More information on Nektar++ compilation options is available in Section 1.3.5.

Q. When compiling Nektar++, I receive the following error:

```
CMake Error: The following variables are used in this
oject, but they are set to NOTFOUND.
Please set them or make sure they are set and tested
rrectly in the CMake files:
NATIVE_BLAS (ADVANCED)
linked by target "LibUtilities" in directory
/path/to/nektar++/library/LibUtilities
NATIVE_LAPACK (ADVANCED)
linked by target "LibUtilities" in directory
/path/to/nektar++/library/LibUtilities
```

This is caused by one of two problems:

- The BLAS and LAPACK libraries and development files are not installed. On Linux systems, both the LAPACK library package (usually called liblapack3 or lapack) and the development package (usually called liblapack-dev or lapack-devel) must be installed. Often the latter is missing.
- An alternative BLAS/LAPACK library should be used. HPC systems frequently use the Intel compilers (icc, icpc) and the Intel Math Kernel Library (MKL). This software should be made available (if using the modules environment) and the option <code>NEKTAR_USE_MKL</code> should be enabled.
- Q. When I compile Nektar++ I receive an error

```
error: #error "SEEK_SET is #defined but must not be for the C++ binding of MPI. Include mpi.h before stdio.h"
```

This can be fixed by including the flags

-DMPICH_IGNORE_CXX_SEEK -DMPICH_SKIP_MPICXX

in the CMAKE_CXX_FLAGS option within the ccmake configuration.

Q. After installing Nektar++ on my local HPC cluster, when I run the 'ctest' command, all the parallel tests fail. Why is this?

The parallel tests are those which include the word **parallel** or **par**. On many HPC systems, the MPI binaries used to execute jobs are not available on the login nodes, to prevent inadvertent parallel runs outside of the queuing system. Consequently, these

tests will not execute. To fully test the code, you can submit a job to the queuing system using a minimum of two cores, to run the **ctest** command.

Q. When running any Nektar++ executable on Windows, I receive an error that zlib.dll cannot be found. How do I fix this?

Windows searches for DLL files in directories specified in the PATH environmental variable. You should add the location of the ThirdParty files to your path. To fix this (example for Windows XP):

- As an administrator, open "System Properties" in control panel, select the "Advanced" tab, and select "Environment Variables".
- Edit the system variable 'path' and append

C:\path\to\nektar++\ThirdParty\dist\bin

to the end, replacing path\to\nektar++ appropriately.

Q. When compiling Nektar++ Thirdparty libraries I get an error "CMake Error: Problem extracting tar"

Nektar++ tries to download the appropriate ThirdParty libraries. However if the download protocols are restricted on your computer this may fail leading to the error "'CMake Error: Problem extracting tar". These libraries are available from

http://www.nektar.info/thirdparty/

and can be downloaded directly into the **\$NEKTAR_HOME/ThirdParty** directory

17.2 Usage

Q. How do I run a solver in parallel?

In a desktop environment, simply prefix the solver executable with the mpirun helper. For example, to run the Incompressible Navier-Stokes solver on a 4-core desktop computer, you would run

mpirun -np 4 IncNavierStokesSolver Cyl.xml

In a cluster environment, using PBS for example, the mpiexec command should be used.

Q. How can I generate a mesh for use with Nektar++?

Nektar++ supports a number of mesh input formats. These are converted to the Nektar++ native XML format (see Section 3) using the NekMesh utility (see Section 4. Supported formats include:

- Gmsh (.msh)
- Polygon (.ply)
- Nektar (.rea)
- Semtex (.sem)

Bibliography

- [1] M Ainsworth and S Sherwin. Domain decomposition preconditioners for p and hp finite element approximation of stokes equations. *COMPUTER METHODS IN APPLIED MECHANICS AND ENGINEERING*, 175:243–266, 1999.
- [2] R. R. Aliev and A. V. Panfilov. A simple two-variable model of cardiac excitation. Chaos, Solitons & Fractals, 7:293–301, 1996.
- [3] Ivo Babuška and Manil Suri. The p and h-p versions of the finite element method, basic principles and properties. *SIAM review*, 36(4):578–632, 1994.
- [4] Y. Bao, R. Palacios, M. Graham, and S.J. Sherwin. Generalized "thick" strip modelling for vortex-induced vibration of long flexible cylinders. J. Comp. Phys, 321:1079–1097, 2016.
- [5] P-E Bernard, J-F Remacle, Richard Comblen, Vincent Legat, and Koen Hillewaert. High-order discontinuous galerkin schemes on general 2d manifolds applied to the shallow water equations. *Journal of Computational Physics*, 228(17):6514–6535, 2009.
- [6] CD Cantwell, D Moxey, A Comerford, A Bolis, G Rocco, G Mengaldo, D De Grazia, S Yakovlev, J-E Lombard, D Ekelschot, et al. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015.
- [7] D. De Grazia, G. Mengaldo, D. Moxey, P. E. Vincent, and S. J. Sherwin. Connections between the discontinuous galerkin method and high-order flux reconstruction schemes. *International Journal for Numerical Methods in Fluids*, 75(12):860–877, 2014.
- [8] S. Dong. A convective-like energy-stable open boundary condition for simulation of incompressible flows. *Journal of Computational Physics*, 302:300–328, 2015.
- [9] S. Dong, G. E. Karniadakis, and C. Chryssostomidis. A robust and accurate outflow boundary condition for incompressible flow simulations on severely-truncated unbounded domains. *Journal of Computational Physics*, 261:95–136, 2014.

292 Bibliography

- [10] F Ducros, V Ferrand, Franck Nicoud, C Weber, D Darracq, C Gacherieu, and Thierry Poinsot. Large-eddy simulation of the shock/turbulence interaction. *Journal* of Computational Physics, 152(2):517–549, 1999.
- [11] Niederer "et al.". Verification of cardiac tissue electrophysiology simulators using an n-version benchmark. *Philos Transact A Math Phys Eng Sci*, 369:4331–51, 2011.
- [12] Roland Ewert and Wolfgang Schröder. Acoustic perturbation equations based on flow decomposition via source filtering. *Journal of Computational Physics*, 188(2):365–398, 7 2003.
- [13] Paul F. Fischer. Projection techniques for iterative solution of ax = b with successive right-hand sides. Computer Methods in Applied Mechanics and Engineering, 163(1):193 – 204, 1998.
- [14] Abel Gargallo-Peiró, Xevi Roca, Jaime Peraire, and Josep Sarrate. Distortion and quality measures for validating and generating high-order tetrahedral meshes. *Engineering with Computers*, 31(3):423–437, 2015.
- [15] Georg Geiser, Holger Nawroth, Arash Hosseinzadeh, Feichi Zhang, Henning Bockhorn, Peter Habisreuther, Johannes Janicka, Christian O. Paschereit, and Wolfgang Schröder. Thermoacoustics of a turbulent premixed flame. In 20th AIAA/CEAS Aeroacoustics Conference. American Institute of Aeronautics and Astronautics.
- [16] David Gottlieb, Steven A Orszag, and CAMBRIDGE HYDRODYNAMICS INC MA. Numerical analysis of spectral methods. SIAM, 1977.
- [17] J.L. Guermond and J. Shen. Velocity-correction projection methods for incompressible flows. SIAM J. Numer. Anal., 41:112–134, 2003.
- [18] Jan S Hesthaven and Tim Warburton. Nodal high-order methods on unstructured grids: I. time-domain solution of maxwell's equations. *Journal of Computational Physics*, 181(1):186–221, 2002.
- [19] B. E. Jordi, C. J. Cotter, and S. J. Sherwin. Encapsulated formulation of the selective frequency damping method. *Phys. Fluids*, 2014.
- [20] G. E. Karniadakis, M. Israeli, and S. A. Orszag. High-order splitting methods for the incompressible Navier–Stokes equations. 97(2):414–443, 1991.
- [21] G. E. Karniadakis and S. J. Sherwin. Spectral/hp Element Methods for Computational Fluid Dynamics. Oxford Science Publications, 2005.
- [22] Robert M Kirby and Spencer J Sherwin. Stabilisation of spectral/hp element methods through spectral vanishing viscosity: Application to fluid mechanics modelling. *Computer methods in applied mechanics and engineering*, 195(23):3128–3144, 2006.
- [23] Jonas Koko. Vectorized matlab codes for linear two-dimensional elasticity. Scientific Programming, 15(3):157–172, 2007.

- [24] Kilian Lackhove. *Hybrid Noise Simulation for Enclosed Configurations*. Doctoral thesis, Technische Universität Darmstadt, 2018.
- [25] C. H. Luo and Y. Rudy. A model of the ventricular cardiac action potential. depolarization repolarization and their interaction. *Circulation research*, 68:1501– 1526, 1991.
- [26] Xian Luo, Martin R. Maxey, and George Em Karniadakis. Smoothed profile method for particulate flows: Error analysis and simulations. *Journal of Computational Physics*, 228(5):1750–1769, 2009.
- [27] R. J. Ramirez M. Courtemanche and S. Nattel. Ionic mechanisms underlying human atrial action potential properties: insights from a mathematical model. *American Journal of Physiology-Heart and Circulatory Physiology*, 275:H301–H321, 1998.
- [28] Y. Maday, A. T. Patera, and E.M. Ronquist. An operator-integration-factor splitting method for time-dependent problems: Application to incompressible fluid flow. J. Sci. Comp., 4:263–292, 1990.
- [29] Yvon Maday, Sidi M Ould Kaber, and Eitan Tadmor. Legendre pseudospectral viscosity method for nonlinear conservation laws. SIAM Journal on Numerical Analysis, 30(2):321–342, 1993.
- [30] Gianmarco Mengaldo, Daniele De Grazia, Freddie Witherden, Antony Farrington, Peter Vincent, Spencer Sherwin, and Joaquim Peiro. A Guide to the Implementation of Boundary Conditions in Compact High-Order Methods for Compressible Aerodynamics. American Institute of Aeronautics and Astronautics, 2014/08/10 2014.
- [31] RC Moura, SJ Sherwin, and Joaquim Peiró. Eigensolution analysis of spectral/hp continuous galerkin approximations to advection–diffusion problems: Insights into spectral vanishing viscosity. *Journal of Computational Physics*, 307:401–422, 2016.
- [32] Rodrigo C Moura, Andrea Cassinelli, André FC da Silva, Erik Burman, and Spencer J Sherwin. Gradient jump penalty stabilisation of spectral/hp element discretisation for under-resolved turbulence simulations. *Computer Methods in Applied Mechanics* and Engineering, 388:114200, 2022.
- [33] D. Moxey, M. Hazan, J. Peiró, and S. J. Sherwin. An isoparametric approach to high-order curvilinear boundary-layer meshing. *Comp. Meth. Appl. Mech. Eng.*, 2014.
- [34] D. Moxey, M. Hazan, J. Peiró, and S. J. Sherwin. On the generation of curvilinear meshes through subdivision of isoparametric elements. to appear in proceedings of Tetrahedron IV, 2014.
- [35] Yasuya Nakayama and Ryoichi Yamamoto. Simulation method to resolve hydrodynamic interactions in colloidal dispersions. *Phys. Rev. E*, 71:036707, Mar 2005.

294 Bibliography

- [36] David J Newman and George Em Karniadakis. A direct numerical simulation study of flow past a freely vibrating cable. *Journal of Fluid Mechanics*, 344:95–136, 1997.
- [37] Anthony T Patera. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *Journal of computational Physics*, 54(3):468–488, 1984.
- [38] P.-O. Persson and J. Peraire. Sub-cell shock capturing for Discontinuous Galerkin methods. In 44th AIAA Aerospace Sciences Meeting and Exhibit, page 112, 2006.
- [39] N Pignier. One-dimensional modelling of blood flow in the cardiovascular system, 2012.
- [40] CJ Roth. Pulse wave propagation in the human vascular system, 2012.
- [41] S Sherwin. A substepping navier-stokes splitting scheme for spectral/hp element discretisations. pages 43–52. Elsevier Science, 2003.
- [42] SJ Sherwin and M Ainsworth. Unsteady navier-stokes solvers using hybrid spectral/hp element methods. APPLIED NUMERICAL MATHEMATICS, 33:357–363, 2000.
- [43] SJ Sherwin, L Formaggia, J Peiró, and V Franke. Computational modelling of 1d blood flow with variable mechanical properties and its application to the simulation of wave propagation in the human arterial system. Int. J. Numer. Meth. Fluids, 43:673-700, 2003.
- [44] SJ Sherwin and G Em Karniadakis. Tetrahedral< i> hp</i> finite elements: Algorithms and flow simulations. Journal of Computational Physics, 124(1):14–45, 1996.
- [45] K. H. W. J. ten Tusscher and A. V. Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology-Heart and Circulatory Physiology*, 291:H1088–H1100, 2006.
- [46] M Turner, J Peiró, and D Moxey. A Variational Framework for High-Order Mesh Generation. In 25th International Meshing Roundtable, volume 163, pages 340–352, 2016.
- [47] Zhicheng Wang, Michael S Triantafyllou, Yiannis Constantinides, and George Em Karniadakis. A spectral-element/Fourier smoothed profile method for large-eddy simulations of complex VIV problems. *Computers & Fluids*, 172:84–96, 2018.
- [48] N Westerhof. Anatomic studies of the human systemic arterial tree. J. Biomech., 2:121–143, 1969.
- [49] D Xiu, SJ Sherwin, S Dong, and GE Karniadakis. Strong and auxiliary forms of the semi-lagrangian method for incompressible flows. J. Sci. Comp., 25:323–346, 2005.
- [50] Olgierd Cecil Zienkiewicz and Robert Leroy Taylor. Basic formulation and linear problems. McGraw-Hill, 1989.