Imperial College London

MSc Thesis: HPC on the Cloud

Author:

Friedrich M. Grabner

CID - 01220997

Supervisor:

Dr C. Cantwell

Thesis submitted as part of the requirements

for the award of the MSc in Advanced Computational Methods for Aeronautics,

Flow Management, and Fluid-Structure Interaction

in the

Department of Aeronautics, Imperial College London.

September 2017

Nektar++ is a powerful set of libraries which can be used to simulate a number of physical phenomena. Currently users number in the range of low hundreds. A critical obstacle to wide scale adoption, is in part the complexity of using the software. Indeed there is a steep learning curve to both setup a simulation file and specify computational resources correctly. Limiting ourselves to Nektar++'s incompressible Navier-Stokes solver, a graphical user-interface has been developed. Named TemPSS, the interface is hosted on a Java based web platform. Moreover the entire parameter space, of the incompressible flow solver, has been mapped out using XML schema. Functioning input files for the Nektar++ incompressible flow solver are then generated through an XSLT transform, which the user can then use to run their simulation. Verification of the TemPSS interface has then been performed. Comparison of input files created both manually and through TemPSS for three separate test cases. The outputs from testing proved that TemPSS robustly generates functioning input files for Nektar++.

Acknowledgements

A number of people also should receive credit for the success of this project.

Firstly, I must thank Dr Chris Cantwell for having a clear idea of what the outcome of this project should be and guiding me through the many libraries contained within Nektar++ with wit. Additionally I would like to thank Dr Jeremy Cohen for all his technical help and insight. Professor John Darlington also requires praise for his interesting remarks regarding the broader picture of TemPSS and libhpc.

Of course I would also like to thank my various colleagues and friends who have made my time at Imperial College an extremely interesting and memorable experience.

A great contribution is also due to my family who supported me throughout my entire education leading to this point!

Contents

1 Introduction **1**

2 Project Aims and Objectives **3**

3 Methods and Theory **4**

3.1 Nekkloud, libhpc and TemPSS 4

3.1.1 Design of TemPSS 4

3.1.2 Input to Output 5

3.2 Template Tree Design 9

3.2.1 Problem Specification 10

3.2.2 Numerical Specification 11

3.2.3 Additional Parameters 11

3.2.4 Advanced Parameters 11

3.2.5 Optimsation 12

3.3 Testing of the Interface 12

3.3.1 Recommendations and Actions 12

4 Constraints **13**

4.0.2 Direct Constraints 13

4.0.3 Indirect Constraints 14

4.1 Mapping Constraints 15

4.1.1 Dimensions, Solver Type, and Field 15

4.1.2 Geometry and Boundary Conditions 16

4.1.3 Simulation Parameters 17

4.1.4 Equation Type and Simulation Parameters 18

4.1.5 Timing Details 20

4.1.6 Driver Type 21

4.1.7 Evolution Operators 21

4.2 Constraints Graph 22

5 Testing and Results **24**

5.1 NACA Aerofoil 24

5.2 T106a Quasi-3D Turbine Blade 26

5.3 Turbulent Pipe Flow 26

6 Conclusion and Further Work **29**

6.1 Conclusion 29

6.2 Further Work and Recommendations 29

6.2.1 Development of Nekkloud/Nektar++ Eco-System 29

6.2.2 Creation of Additional Templates 30

6.2.3 Partial Profiles 30

A TemPSS Tree Listing **32**

B Interactive Constraint Visualiser **36**

C NACA65 - Drag and Lift Force Measurement Comparison **38**

D T106a - Drag and Lift Force Measurement Comparison **39**

E Test TemPSS User Remarks **40**

E.1 User 1 40

E.2 User 2 40

E.3 User 3 41

Nomenclature

DNS Direct Numerical Simulation

HPC High Performance Computing

IaaS Infrastructure as a Service

SFD Selective Frequency Damping

TemPSS Temples and Profiles for Scientific Software

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

List of Figures

1.1 Convergence of functionality attempted through Nekkloud. 1

3.1 TemPSS Interface incompressible Navier-Stokes solver template seclected. 5
3.2 Example Xml Schema Defining Part of a TemPSS Tree. 6
3.3 Snippet of Overall Tree Showing Definition of Forcing Function. 7
3.4 Example Custom Node Type Limiting Input to Positive Values. 7
3.5 Example XSLT Transform File Showing How XSLT Searching for Activated Nodes. . . 7
3.6 Example XSLT transform File Showing XSLT Searching for Values of Nodes. 8
3.7 Example XSLT transform determining overall XML structure. 8
3.8 Example Output Nektar++ Input File Generated by TemPSS. 9
3.9 Accessing Documentation Available for Nodes. 10
3.10 Visualisations of the Key Branches in the TemPSS Interface. 12

4.1 Work Flow of Decisions Necessary to Create a Simulation. 13
4.2 Example Constraint File Showing the Listing of Solver Types and Projections. 14
4.3 Example Constraint File Showing the Linking of Projection Types to Solvers. 14
4.4 Information Widget Available in TemPSS Showing a List of Constraints Enforced in the Parameter Space. 15
4.5 Comparison of the Automatically Selected Fields Used in Boundary Condition Definition. 16
4.6 Boundary Regions Identified In Uploaded Geometry File. 16
4.7 Comparison of Dummy Parameter Selections. 17
4.8 Snippet of Tree Showing Sub-trees for Specification of AdaptiveSFD Evolution Operator. 22
4.9 Excerpt of Constraint Chord Diagram Showing Directionality of Choices. 23

5.1 Domain of the NACA Aerofoil. 25
5.2 Comparison of Drag for Original and TemPSS Generated Input Files of NACA65. . . 25
5.3 Domain of the T106a Turbine Blade. 26
5.4 Comparison of Drag for Original and TemPSS Generated Input Files of T106a. 27
5.5 Domain of Turbulent Pipe. 27
5.6 Modal Energy of Turbulent Pipe. 28

A.1 Diagrammatical Overview of Incompressible Navier-Stokes TemPSS Tree. 32
A.2 Problem Specification Branch for Incompressible Navier-Stokes TemPSS Tree. 32
A.3 Numerical Specification Branch for Incompressible Navier-Stokes TemPSS Tree. 33
A.4 Additional Parameters Branch for Incompressible Navier-Stokes TemPSS Tree. 34
A.5 Advanced Parameters Branch for Incompressible Navier-Stokes TemPSS Tree. 34
A.6 Optimisation Branch for Incompressible Navier-Stokes TemPSS Tree. 35

B.1 Mapping of constraints between function for Nektar++ Incompressible Navier-Stokes Solver. 36

List of Tables

4.1 Relationship Between Equation and Solver Types. 19
4.2 Relationship Between Equation and Simulation Types. 20
4.3 Relationship Between Equation and Solver Types. 20
4.4 Relationship Between Driver and Simulation Types. 21
4.5 Relationship Between Driver and Simulation Types. 22

5.1 Boundary Conditions of the NACA Aerofoil Simulation. 24
5.2 Boundary Conditions of the T106a Turbine Blade Simulation. 26
5.3 Boundary Conditions of Pipe Simulation 27

C.1 Aeroforces NACA65 Original Input 38
C.2 Aeroforces NACA65 TemPSS Generated Input 38

D.1 Aeroforces T106a Original Input 39
D.2 Aeroforces T106a TemPSS Generated Input 39

Chapter 1

Introduction

Over the past decade rapid progress in computing power has made high-order numerical methods feasible in the simulation of a broad range of engineering problems. In addition the availability of IaaS "cloud-computing" facilities, such as the Amazon EC2[1] and Google Compute Engine[2], has allowed people outside of large institutions with private clusters to gain access to HPC.

Despite the advancements made many people are still reluctant to utilise codes that implement these new methods, preferring to run commercial software on personal machines. This is in part due to the daunting and complex interfaces of open-source solvers, but also due to the large amounts of additional expertise required to successfully run a simulation.

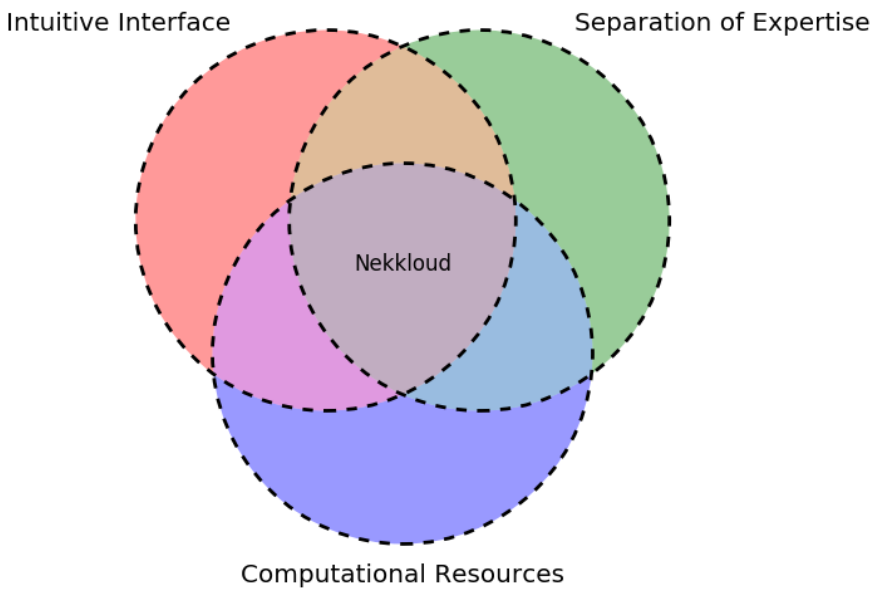


Figure 1.1: Convergence of functionality attempted through Nekkloud.

Nektar++[3] is an open-source framework of libraries that encapsulate the spectral/hp element method. These libraries can be assembled into a number of different PDE solvers that simulate a

variety of physical phenomena. Simulations are defined through the construction of XML[4] files. Whilst the documentation contains a number of tutorials and other examples, which new users can use to learn more about the code, the XML input files are best developed using a simple text-editor, such as VI or notepad. This requirement immediately alienates those more comfortable with rich text-editors and graphical user interfaces. Furthermore significant knowledge is required to correctly install Nektar++ on all supported platforms. Finally even if correct XMLs have been generated and the software is built correctly, launching a succesful simulation requires definition of computational resources. If Nektar++ is to gain acceptance from the wider community a way of overcoming these issues is needed.

Nekkloud[5] is a web-based platform, built on the libhpc framework, through which the Nektar++ solvers can be run. As shown in figure 1.1, Nekkloud streamlines the use of Nektar++ through opening access to multiple computational resources; separation of expertise into various sections, with the ability to generate many examples; and through providing an intuitive interface encompassing all possible functionality of the solvers but ensuring robust cases are generated.

The project presented within this report aims to develop the latter two of the previous points. Using the TemPSS[6], a template and profile manager and editor for the libhpc framework, a graphical interface for Nektar++'s incompressible flow solver has been developed. Through understanding of the constraints present in the code base and numerical methods the interface tries to separate the different domains of expertise required to utilise the code providing a database of example profiles from which new problems can be created.

Chapter 2

Project Aims and Objectives

Project Aims:

- Build a web-based graphical user interface, which generates robust and sane input XML files for the incompressible flow solver in Nektar++.

The incompressible flow solver was chosen as it has the largest user base and represents a foundation upon which templates for other solvers can be built.

Project Objectives:

1. Map and determine the entire parameter space of the incompressible flow solver.
2. Understand and detail the constraints between said simulation parameters.
3. Develop TemPSS transform templates that correctly instantiate all simulation parameters.
4. Represent these parameters and constraints within a logical and intuitive interface.
5. Perform beta testing of the interface with Nektar++ users and implement recommendations.
6. Test the operation of XML generation, through benchmarking against a selection of representative testcases.

Chapter 3

Methods and Theory

3.1 Nekkloud, libhpc and TemPSS

Nekkloud is the first HPC application to be developed using the libhpc framework. As a web-based platform a number of Nektar++ solvers are available to run on IaaS cloud resources or specific clusters. The libhpc services allows the user to set up compute jobs and select the platforms on which they wish to run.

Sitting beneath libhpc is TemPSS - **Tem**plates and **P**rofiles for **S**cientific **S**oftware. The TemPSS interface allows the specification of application parameter templates and profiles from which the input XML files for Nektar++ can be generated. As a Java web service, TemPSS is built using Apache Maven and subsequently deployed on an Apache Tomcat server.

3.1.1 Design of TemPSS

The two primary components of the TemPSS interface are the templates and profiles, pictured in figure 3.1.

Templates

In order to create an XML input file, using the TemPSS interface, a template must be selected - as shown in the blue box in figure 3.1. Templates define the parameter space avaiable in the tree, see appendix A. Moreover the templates available through Nekkloud are each a separate solver package that exist in Nektar++. Though, within the scope of this work, only the incompressible flow solver is considered. Upon selection of a template the relevant tree is loaded into the profile editor as shown highlighted in green box.

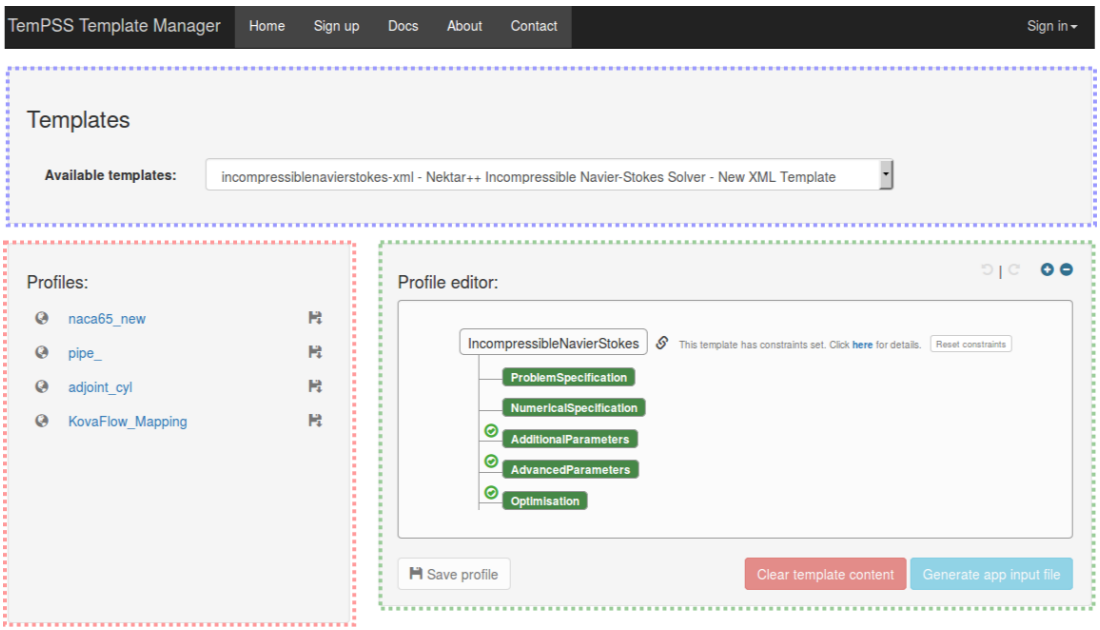


Figure 3.1: TemPSS Interface incompressible Navier-Stokes solver template seclected.

Profiles

Profiles are instantiations of templates. A profile contains the values for the entire or part of the template tree. Profiles can then be saved, as seen in the profile box highlighted in red. Once a profile is entirely filled then it is possible to transform the template and generate an output XML file.

Profiles can also be saved and shared with other users, this is part of the wider eco-system that Nekkloud is trying to foster. Users who lack expertise in one aspect of the simulation setup are able to look what other users have specified for similar problems. Figure 3.1 shows a number of saved profiles available for a user to load and tailor to their requirements.

3.1.2 Input to Output

Built using the Java web service environment, TemPSS uses a number of Java libraries and files to manage the user and profiles database. Additionally these also to define the overall function and look of the interface.

Structure of template trees, such as that for the incompressible Navier-Stokes solver, are defined by XML schema. These schema are constituted, in part, by further smaller XML style sheets. The final Nektar++ compatible XML is then generated through an XSLT transform.

XML Structure

XML is the chosen text format for both the input file for Nektar++ and schema definition for the TemPSS parameter tree. XML files are structured into multiple hierarchies of parent and child nodes. Figure 3.2 shows an example XML schema definition of a forcing function in Nektar++. The parent node is named `Force`, and it has three children: `Absorption`, `Body` and, `Noise`. The children may

also have children, such as `Absorption` has `Coeff`, `RefFlow` and, `RefFlowTime`. This can go on *ad infinitum*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Force repeatable="true" paramType="choice" optional="true">
  <Absorption>
    <Coeff type="xs:string"/>
    <RefFlow type="xs:string"/>
    <RefFlowTime type="xs:string"/>
  </Absorption>
  <Body>
    <BodyForce type="xs:string"/>
  </Body>
  <Noise>
    <Whitenoise type="positiveDouble"/>
    <UpdateFreq type="positiveDouble" optional="true"/>
    <Nsteps type="positiveDouble" optional="true"/>
  </Noise>
</Force>
```

Figure 3.2: Example Xml Schema Defining Part of a TemPSS Tree.

To define the functionality of a node keywords with additional information can be include in the node head. The options available, to both the parent and child nodes, are given by the standard XSLT library and additional custom options are defined across a number of XML style sheets.

In figure 3.2 the parent node `Force` uses keywords `repeatable`, `paramType`, and `optional`. Keywords `repeatable` and `optional` are defined in the XSL file `XsdToHtmlTransform.xsl`. These options give the ability to define between 0 to ∞ instances of `Force`. For example one may wish to have both an absorption and body force, thus the option `repeatable` allows the addition of instances of this function. In figure 3.3 we see the manifestation of the `repeatable` keyword in the form of a plus sign, clicking on this adds an extra instance to the tree. Similarly the `optional` command is seen in figure 3.3 as the slider. The slider is coloured green for force, as it has been selected, but red for the other options which are not. This switch allows options to be added in or out of the output easily.

The child node `Noise` is selected in figure 3.3. Observing that in the XML structure, figure 3.2, that defines this tree the child of `Noise` has the nodes `Whitenoise`, `UpdateFreq` and `Nsteps`. These have type defined as positiveDouble - a custom type. Figure 3.4 shows how using the XSLT standard parameters a new custom type is created which only allows positive doubles to be entered as inputs.

XSLT Transform

XSLT transforms generated an output XML, compatible with Nektar++, from the instatiated XML schema. XSLT reads the relationships between nodes as a set of paths. Named XPATH, this allows information entered at any node within the TemPSS tree top be accessed and then applied

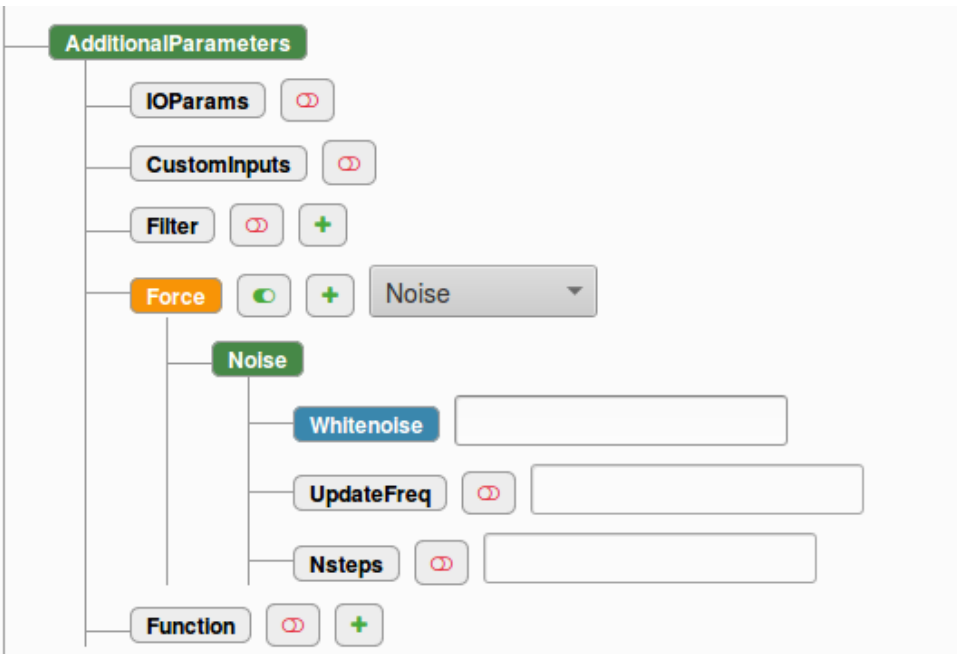


Figure 3.3: Snippet of Overall Tree Showing Definition of Forcing Function.

```
<xs:simpleType name="positiveDouble">
  <xs:restriction base="xs:double">
    <xs:minExclusive value="0"/>
  </xs:restriction>
</xs:simpleType>
```

Figure 3.4: Example Custom Node Type Limiting Input to Positive Values.

to the output XML. For example in figure 3.2 the XPATH to the value of `UpdateFreq` would be `Force/Noise/UpdateFreq`.

```
<xsl:template match="Force" mode ="AddForces">
    <FORCE>
      <xsl:if test="Absorption">
        <xsl:attribute name="TYPE">Absorption</xsl:attribute>
        <xsl:apply-templates select="Absorption" mode ="AddForces"/
           >
      </xsl:if>
      <xsl:if test="Body">
        <xsl:attribute name="TYPE">Body</xsl:attribute>
        <xsl:apply-templates select="Body" mode ="AddForces"/>
      </xsl:if>
      <xsl:if test="Noise">
        <xsl:attribute name="TYPE">Noise</xsl:attribute>
        <xsl:apply-templates select="Noise" mode ="AddForces"/>
      </xsl:if>
    </FORCE>
</xsl:template>
```

Figure 3.5: Example XSLT Transform File Showing How XSLT Searching for Activated Nodes.

In figure 3.7 the XPATH of the primary node is seen on line 2. The function `match` finds the paths specified, in this case `IncompressibleNavierStokes`, and subsequently writes the XML nodes `<NEKTAR>` to lines 3 and 8 of the output XML, figure 3.8. In order to further instantiate the output XML another template is applied on line 4 of figure 3.7. Here function `<xsl:apply-templates>` then scans for the path `AdditionalParametersForce`. If this path exists the template named `AddForces`, figure 3.5, is applied.

In the template `AddForces` the same process is repeated. If the XPATH finds `Force` in the TemPSS tree the nodes `<FORCE>` are added, which manifest in the output 3.8 on lines 3 and 7. As there are a number of options this template performs a number of `<xsl:if>` tests to check which additional templates should be utilised. Absorption has been selected in the TemPSS tree, thus the `<xsl:attribute>` places the tag `TYPE="Absorption"` in the output force node, see line 3 of figure 3.8. When the simulation is then run the Nektar++ solver will read this tag and apply an absorption force as desired.

```
<xsl:template match="Absorption" mode ="AddForces">
    <COEFF><xsl:value-of select="Coeff"/></COEFF>
    <REFFLOW><xsl:value-of select="RefFlow"/></REFFLOW>
    <REFFLOWTIME><xsl:value-of select="RefFlowTime"/></REFFLOWTIME>
</xsl:template>

<xsl:template match="Body" mode ="AddForces">
    <BODYFORCE><xsl:value-of select="Body"/></BODYFORCE>
</xsl:template>

<xsl:template match="Noise" mode ="AddForces">
    <WHITENOISE><xsl:value-of select="Whitenoise"/></WHITENOISE>
    <xsl:if test="UpdateFreq">
      <UPDATEFREQ><xsl:value-of select="UpdateFreq"/></UPDATEFREQ>
    </xsl:if>
    <xsl:if test="Nsteps">
      <NSTEPS><xsl:value-of select="Nsteps"/></NSTEPS>
    </xsl:if>
</xsl:template>
```

Figure 3.6: Example XSLT transform File Showing XSLT Searching for Values of Nodes.

```
<!-- Incompressible Navier-Stokes transform -->
<xsl:template match="/IncompressibleNavierStokes">
  <NEKTAR>
    <xsl:apply-templates select="AdditionalParameters/Force" mode =
       "AddForces"/>
  </NEKTAR>
</xsl:template>
```

Figure 3.7: Example XSLT transform determining overall XML structure.

Having applied the absorption force tag another template is called, seen in figure 3.6, to apply the conditions of the force. This template creates three new nodes in the output XML, lines 4, 5, and 6 of figure 3.8. Using the function `<xsl:value-of>` the values of the TemPSS tree nodes are then written between the output nodes of each variable.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <NEKTAR>
3   <FORCE TYPE="Absorption">
4     <COEFF>1.0</COEFF>
5     <REFFLOW>2.0</REFFLOW>
6     <REFFLOWTIME>3.0</REFFLOWTIME>
7   </FORCE>
8 </NEKTAR>
```

Figure 3.8: Example Output Nektar++ Input File Generated by TemPSS.

Using the definition of the absorption force it can be seen that the output XML can be generated in a jigsaw like fashion through a combination of templates. Though in the example a single template may have sufficed. It is essential to distill functions to their core function to encourage reuse of transform templates leading to cleaner code.

3.2 Template Tree Design

The primary motivation for Nekkloud is to improve the user experience of Nektar++. Fundamentally designing the tree is easy. However in reality this becomes increasingly complex as all functionality, of the incompressible flow solver, should be encapsulated within the TemPSS paramater tree. Moreover each branch should have as much independence from the others as possible, nevertheless a number of key intra-branch relationships persit. Branch independence should allows users to pick and choose can what they want from other profiles, and then assemble them into a complete tree germane to their specific problem.

Considerable time has thus been spent trying to logically and intuitively separate the relevant parts, and a full listing is available in appendix A. The code base has been separated into five key branches:

- Problem Specification
- Numerical Specification
- Additional Parameters
- Advanced Parameters
- Optimisation

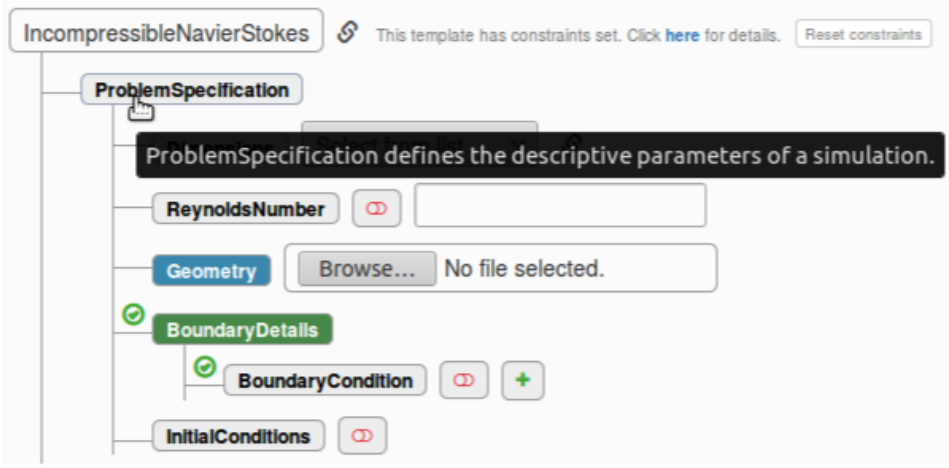


Figure 3.9: Accessing Documentation Available for Nodes.

As previously remarked when designing the interface it is integral to keep related parameters within the same branch. However there are of course a number of parameters which have relationships across branches, these should be where possible linked through a set of constraints.

Names of nodes should be selected judiciously. Firstly they should have a name which represents their function, however in the documentation and input for Nektar++ they are often abreviated. Names should remain indicative as to which abreviated function, a node relates to, so that the user guides are not obselete within the context of TemPSS. Indeed this conflict is aided through the addition of comments to nodes that can elucidate the function of nodes, as pictured in figure 3.9.

Ultimately the organisation of the branches, as listed in appendix A, could take many forms. Thus the subsequent sections try to explain the rationale behind the combination of branches that has been chosen.

3.2.1 Problem Specification

When writing the abstract describing simulations of flow problems for a journal paper the authors will immediately answer four questions:

1. What is the domain of the simulation, is it 2D or 3D?
2. At which Reynolds number is this being simulated?
3. What boundary conditions are enforced?
4. Are there any initial conditions?

The problem specification branch, see figure 3.10a, allows the immediate answering of these questions. Allowing the researcher to define fundamental simulation parameters, upload the geometry, specifiy boundary conditions, and initial conditions whether from file or expression.

3.2.2 Numerical Specification

Having now defined the simulation domain, users now would need to think how and what kind of problem they are solving. Therefore in the numerical specification branch a key node is `SimulationType`.

Three simulation types are available:

- Direct Numerical Simulation
- Stability Analysis
- Steady State Stability Analysis

These options are not real parameters specifiable within the code. Rather they are *dummy* parameters which represent types of simulation that are possible to perform in Nektar++. Selection of a dummy parameter gives only the functions that are relevant to each kind of simulation. For example a DNS simulation gives the option to specificy the time integration method, driver type, but unlike in the stability analyses, no evolution operator. Furthermore the driver types are limited to only those which are used in DNS, standard and adaptive. Thereby shaping the options that are available to users, depending on what kind of analysis they wish to perform.

Additionally in the numerical specification the user defines the solver and equation types. Though they are again constrained in these choices depending on previous decisions, as detailed in section 4.

3.2.3 Additional Parameters

Aside from the definition of the domain and selection of numerics there are a number of options which users may require. The additional parameters branch contains such options. For example the value of kinematic viscosity is used often in Nektar++, however it isn't a parameter which one would use to describe the simulation to a colleague. Therefore it requires inclusion but not in the problem specification branch, neither does it fit in the numerical specification branch.

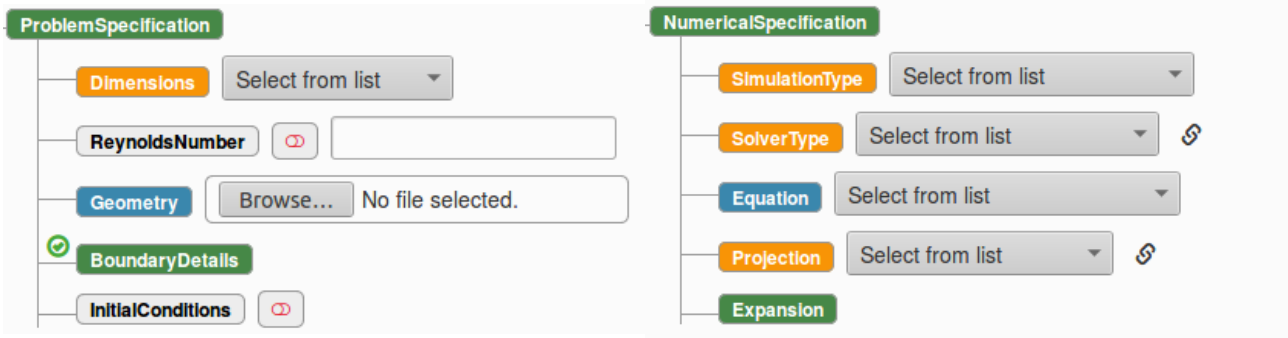
Additionally a number of analytical tools are listed here. Both filters and functions are commonly used tools to calculate the lift and drag across a body amongst others. However they are also optional and case dependent ergo they reside in this section.

3.2.4 Advanced Parameters

Advanced parameters branch contains options which improve stability or accuracy of the results. Most simulations could be run without them but certain situations would benefit from their use. One could imagine the situation where a user is possibly getting spurious oscillations within their domain, and thus are advised to switch dealiasing on. The options in this branch in this branch have some advanced requirements, which can improve simulations however a complete novice user would and should not need to use these options.

3.2.5 Optimsation

Correctly defining the optimisation parameters can lead to improved performance, however injudicious choices will hamper performance. Therefore optimisation parameters are kept in their own branch where experienced users of Nektar++ can create example settings for various cases.



(a) Problem Specification Branch.

(b) Numerical Specification Branch.

Figure 3.10: Visualisations of the Key Branches in the TemPSS Interface.

3.3 Testing of the Interface

In order to ensure that the user interface is intuitive a number of tests have been performed by users of Nektar++.

Three individuals were asked to use TemPSS to generate a test case of their choice. Below their recommendations and comments are listed with a brief explanation of the action taken. Further details of the case ran and other is given in appendix E.

3.3.1 Recommendations and Actions

Recommendation:

1. Speed up the from scratch profile instantiation time.
2. Reduce sensitivity of switches. Include additional filters.
3. Automate the boundary condition selection.

Action Taken:

1. Partial profile loading functionality is being developed.
2. Cannot do anything about button sensitivity. Additional filters, such as history points, have been added.
3. The field variables are now selected depending on dimensions and solver type.

Chapter 4

Constraints

The TemPSS interface must encapsulate all of Nektar++'s parameter space whilst remaining intuitive to operate. Withal, when placed within the wider context of Nekkloud, it is highly undesirable if users are able to launch simulations which do not run - especially if cloud resources have been allocated and paid for! TemPSS must try to prevent or limit users from choosing options that are either incorrect, incompatible or irresponsible.

In order to map constraints between functions the work-flow, of instantiating a profile, should be determined. As users navigate through this work-flow options should come in and out of scope as appropriate.

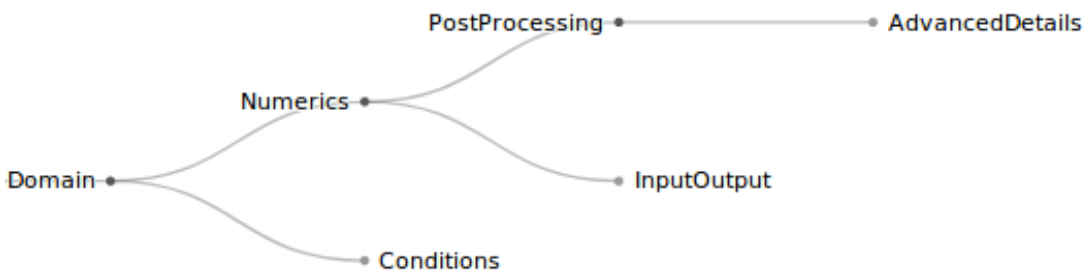


Figure 4.1: Work Flow of Decisions Necessary to Create a Simulation.

Constraints are then enforced through two different methods:

- Direct enforcement
- Indirect enforcement

4.0.2 Direct Constraints

Direct constraint enforcement is the simplest to implement. Using an XML file parameters paths are defined and options listed. In figure 4.2 possible options for solver type and projection are listed.

In figure 4.3 the solver type and projection are directly linked. Specific variables have compatible target variables listed beneath each value. For the coupled linear Navier-Stokes solver only the continuous galerkin projection is available. Whereas, for the velocity correction scheme all three projection

```
<variables>
  <variable path="NumericalSpecification.SolverType" name="SolverType">
    <domain>
      <value>CoupledLinearNS</value>
      <value>VelocityCorrectionScheme</value>
    </domain>
  </variable>

  <variable path="NumericalSpecification.Projection" name="Projection">
    <domain>
      <value>ContinuousGalerkin</value>
      <value>DiscontinuousGalerkin</value>
      <value>MixedGalerkin</value>
    </domain>
  </variable>
</variables>
```

Figure 4.2: Example Constraint File Showing the Listing of Solver Types and Projections.

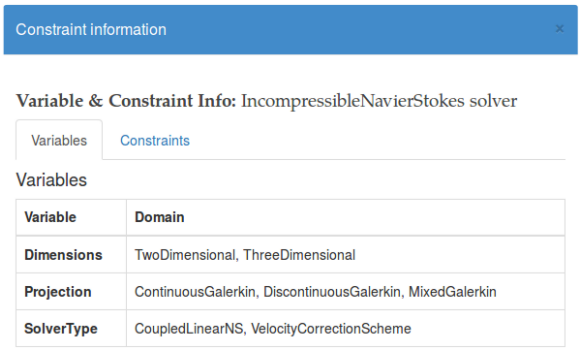
types are available. The constraint file prevents values being chosen if another option does not list an value as valid, as in figure 4.3. Additionally a small guide, as in figures 4.4a and 4.4b, details the variables and values which are constrained.

```
<constraints>
  <!-- Mapping of solver type with projection type -->
  <mapping variable="SolverType" varvalue="VelocityCorrectionScheme" targetVariable="Projection">
    <targetValue>ContinuousGalerkin</targetValue>
    <targetValue>DiscontinuousGalerkin</targetValue>
    <targetValue>MixedGalerkin</targetValue>
  </mapping>
  <mapping variable="SolverType" varvalue="CoupledLinearNS" targetVariable="Projection">
    <targetValue>ContinuousGalerkin</targetValue>
  </mapping>
</constraints>
```

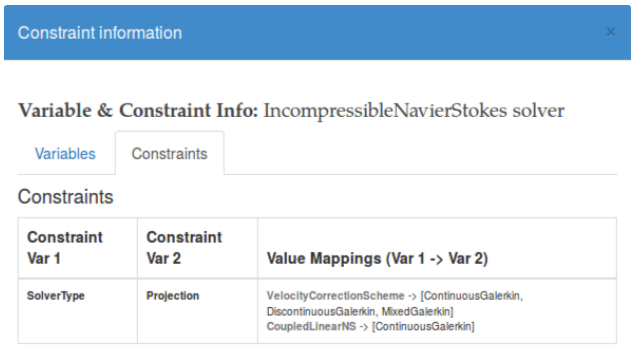
Figure 4.3: Example Constraint File Showing the Linking of Projection Types to Solvers.

4.0.3 Indirect Constraints

Indirect or implicit constraints are somewhat more subtle than the explicitly defined constraints. Through creation of various templates, for certain common functions, the tree can be altered depending upon which kind of simulation is being performed. This occurs for the dummy parameter of `simulationType`. Here the option `EvolutionOperator` does not appear if `DirectNumericalSimulation` is chosen. Whereas for both other simulation types it does.



(a) Constraint Info: Variables.



(b) Constraint Info: Constraints.

Figure 4.4: Information Widget Available in TemPSS Showing a List of Constraints Enforced in the Parameter Space.

4.1 Mapping Constraints

Prior to enforcing the constraints between parameters within the solver, the entire parameter space must be determined. Within this space only a subset of parameters have relationships. For the incompressible Navier-Stokes solver the linked parameters are listed below:

- Dimensions
- Simulation Type
- Solver Type
- Equation Type
- Time Integration Details
- Boundary Conditions
- Field Details
- Evolution Operator
- Driver Type

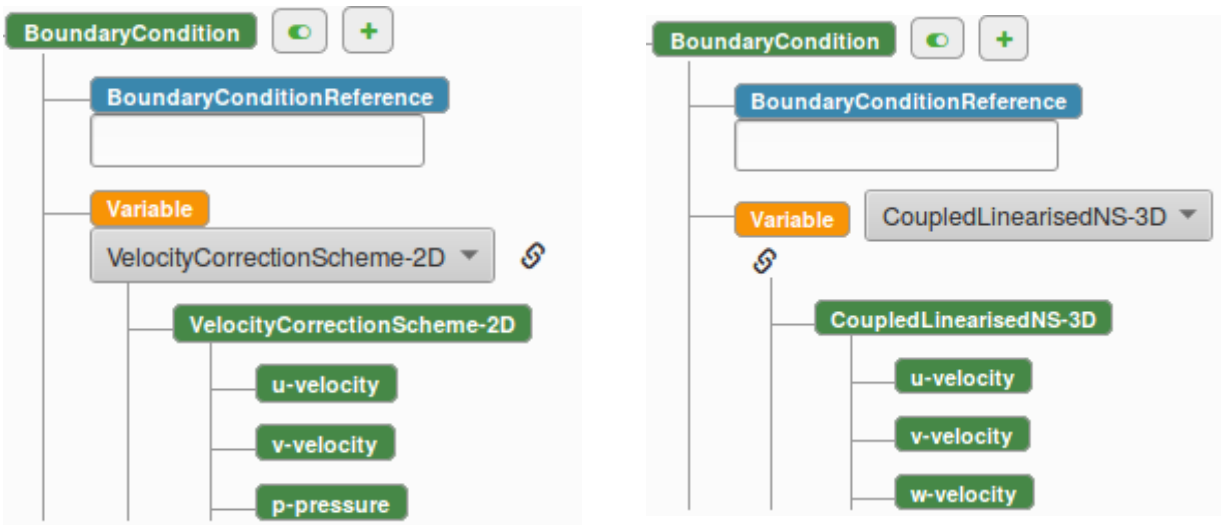
Using documentation, examples, and testing the dependencies have been ascertained and constraints implemented. Furthermore to help easily understand these relationships have been mapped out visually, see appendix B. The use of this graph is subsequently discussed in section 4.2.

4.1.1 Dimensions, Solver Type, and Field

When instantiating a simulation the user will need to understand what domain they are studying, and thus select the dimensions. Having chosen whether the domain is 2D or 3D, and if any of these are quasi-dimensions, the user has determined whether the velocity field contains u, v and w. Corollary to this, selection of solver type determines if the solution of pressure p field is required.

Knowing the dimensions, solver type and therefore the solution fields has consequences in the definition of boundary conditions and expansion. To prevent the incorrect field values being selected, the XML schema automatically selects the correct variables in the boundary conditions and for a simple expansion.

Figure 4.5 shows how the field choice restrictions manifest themselves in the template tree. In figure 4.5a the user started by selecting a two-dimensional domain and the the velocity correction scheme as solver type. Subsequently, when they tried to define the boundary conditions field variables u, v and p



(a) Velocity Correction Scheme 2D.

(b) Coupled Linear Navier-Stokes 3D.

Figure 4.5: Comparison of the Automatically Selected Fields Used in Boundary Condition Definition.

automatically appeared. Similarly, figure 4.5b shows the variables that are presented when using the coupled linear Navier-Stokes solver for a 3D domain.

Here the impact of previous decisions is shown through the restriction of choices available. Important to note is that the directionality of decisions is reversable. Insomuch that the user may start to instantiate a profile anywhere in the tree and constraints will still be enforced identically.

4.1.2 Geometry and Boundary Conditions

In the previous section the user defined the dimensions, solver type and field. Having selected these parameters the boundary conditions should be defined and applied.

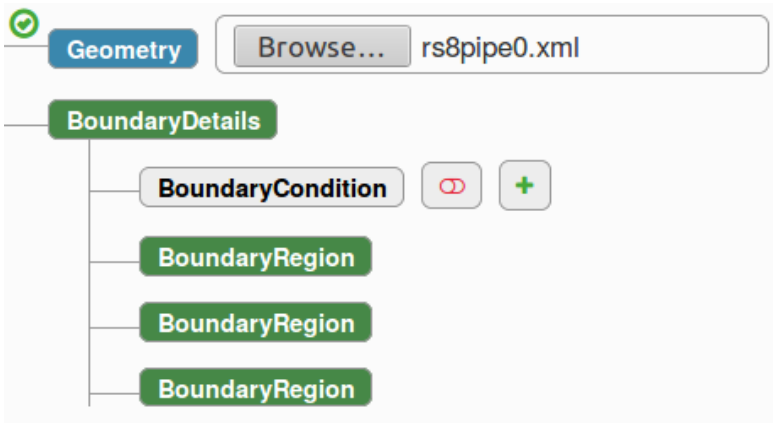


Figure 4.6: Boundary Regions Identified In Uploaded Geometry File.

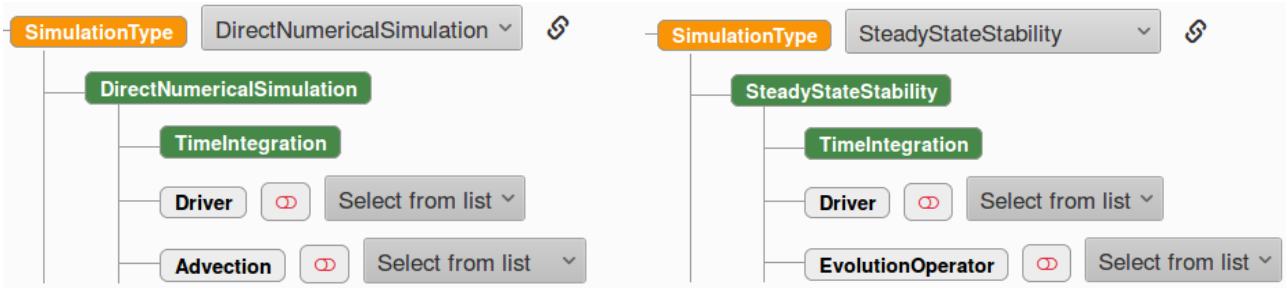
TemPSS allows the user to browse their filesystem to a Nektar++ compatible XML geometry file and upload it. Doing so both allows the geometry to be included in the final TemPSS generated output XML reducing the number of files needed to start a simulation. Additionally the geometry file is scanned for boundary composites. These composites are regions to which a bondition could be

applied ubiquitously. Figure 4.6 shows the identified boundary regions for a pipe geometry. These regions should then be linked to boundary conditions, via their boundary reference, as in figure 4.5.

4.1.3 Simulation Parameters

Reduction in time spent generating the input XMLs is a key goal of TemPSS. Through use of *dummy* parameters the number of options available to the user, at any one time, is limited. These parameters are not Nektar++ related but encapsulate three fundamental types of simulation possible to run using Nektar++'s incompressible flow solver.

- Direct Numerical Simulation
- Stability Analysis
- Steady State Stability Analysis



(a) Direct Numerical Simulation. (b) Steady State Stability.

Figure 4.7: Comparison of Dummy Parameter Selections.

Shown in figure 4.7 the dummy parameters for the DNS and steady-state stability simulation types are shown. For the DNS the advection type appears whereas for the steady state stability it is replaced by the evolution operators. In this way the number of options available at any one time can be limited simplifying the interface for users.

Direct Numerical Simulation

Direct numerical simulation is most commonly used simulation type in Nektar++'s incompressible solver. In order to run a DNS users must define time integration details. Driver and the advection types can also be specified, however the driver type is limited to the standard and adaptive configurations.

Stability Analysis

Hydrodynamic stability is a fundamental part of fluid dynamics. It allows the study of the development of turbulent states of motion from unstable flows. Nektar++ allows the study of hydrodynamic stability of flows utilising a direct numerical simulation of the linearised Navier-Stokes equations using iterative methods for computing the solution of the associated eigenproblem.

Steady State Stability Analysis

Steady state stability analysis computes the steady state Naviers-Stokes equations in order to create a base flow for the purposes of a linear stability analyses. In order to obtain a base flow from potentially unstable flows selective frequency damping is used. The evolution operator is thus limited to SFD modes.

4.1.4 Equation Type and Simulation Parameters

Nektar++ solves the incompressible Navier-Stokes equations with a forcing term, defined as follows:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0 \tag{4.1}$$

Further to this a number of additional formulations of the above equations are available:

- Steady Stokes
- Steady Oseen
- Steady Linearised Navier-Stokes
- Unsteady Stokes
- Unsteady Navier-Stokes

Steady Stokes

The incompressible Navier-Stokes equation is given in full in equation 4.1. However for the case $Re \ll 1$ the momentum component, $\frac{D\mathbf{u}}{Dt}$, can be dropped from the full equation. Such a reduction is known as Stokes flow and represents the case of flows at extremely small length scales and velocities.

$$\nabla p - \nu \nabla^2 \mathbf{u} = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0 \tag{4.2}$$

As the Stokes equations are a linearised form of the full incompressible Navier-Stokes equations there is no time dependency other than that imposed by boundary conditions. Therefore the steady stokes requires only direct methods to find a solution.

Steady Oseen

The steady Oseen equations were derived by Oseen[7] as a development to the Stokes equations. The key difference between them is the inclusion of the convective acceleration.

$$\mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \nu \nabla^2 \mathbf{u} = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0 \tag{4.3}$$

Steady Linearised Navier-Stokes

The linearised Navier-Stokes equations are formed through the perturbation of the flow field. Considering the flow field as a steady base flow with the addition of minute perturbations,

$$\mathbf{u} = \mathbf{U} + \varepsilon \mathbf{u}' \tag{4.4}$$

assuming that the $\varepsilon \ll 1$ the $\mathbf{u}' \cdot \mathbf{u}'$ terms can be neglected. This gives rise to the unsteady linearised Navier-Stokes equation:

$$\frac{\partial \mathbf{u}'}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{u}' + \mathbf{u}' \cdot \nabla \mathbf{U} = -\nabla p + \nu \nabla^2 \mathbf{u}' + \mathbf{f}, \qquad \nabla \cdot \mathbf{u}' = 0 \tag{4.5}$$

Performing the same operation to the steady Navier-Stokes gives the linearised Navier-Stokes:

$$\mathbf{u}' \cdot \nabla \mathbf{U} = -\nabla p + \nu \nabla^2 \mathbf{u}' + \mathbf{f}, \qquad \nabla \cdot \mathbf{u}' = 0 \tag{4.6}$$

Unsteady Stokes

Similarly to the Oseen equations the unsteady Stokes equations seek to include accelerative terms, however in an unsteady equation. In this case the unsteady $\frac{\partial u}{\partial t}$ is included.

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla p - \nu \nabla^2 \mathbf{u} = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0 \tag{4.7}$$

Unsteady Navier-Stokes

The final form, and most commonly used, of the equation that the incompressible Navier-Stokes solver can operate on is the full unsteady Navier-Stokes equations as in equation 4.1.

Relationship with Solver Type

Each equation has a direct relationship with the solver that is available to use. Ergo when a solver is chosen, within the TemPSS tree, the equation types become restricted. These relations are listed directly in the Nektar++ user documentation and again reproduced in table 4.3. Enforcement of these relationships is performed directly through use of the XML constraint file.

Equation	Velocity Correction Scheme	Coupled Linearised Navier-Stokes
Steady Oseen		✓
Steady Stokes	✓	
Steady Linearised Navier-Stokes		✓
Unsteady Stokes	✓	
Unsteady Navier-Stokes	✓	

Table 4.1: Relationship Between Equation and Solver Types.

Relationship with Simulation Type

Along with the choice of solver the equation type is also restricted through the simulation type. Table 4.2 shows the direct relations. These relations are derived from the test and examples cases in Nektar++, but also from the user documentation.

When performing direct numerical simulations all velocity scales are being resolved. In order to capture all the scales the equations cannot be linearised or other therefore the only suitable equation type is that of the unsteady Navier-Stokes.

For the case of stability analysis the Navier-Stokes equations should be written in the perturbed form, thus the steady linearised form is available. However using the appropriate drivers and evolution operators allows the stability analysis to be performed using both the unsteady Stokes and full Navier-Stokes equations.

Steady-State Stability analyses implement the classical and adaptive SFD in order to determine a base flow for further stability analyses. Typically these base flows are derived from fully turbulent simulations hence the unsteady Navier-Stokes is often used. However it also possible to utilise the Steady Oseen and Stokes equations to create a base flow.

Equation	DNS	Stability	Steady-State
Steady Oseen			✓
Steady Stokes			✓
Steady Linearised Navier-Stokes		✓	
Unsteady Stokes		✓	
Unsteady Navier-Stokes	✓	✓	✓

Table 4.2: Relationship Between Equation and Simulation Types.

4.1.5 Timing Details

Timing details, whilst crucial for the output of a physical solution, are dependent directly upon the choice of equation type. However equations available differs for each simulation and solver type.

In effect the equations are split into two groups; those that require time integration, and those that are solved directly. Deducing which equations fall into either category however is not simple, and has been compeleted through testing.

Equation Type	No Time Integration	Time Integration
Steady Oseen		✓
Steady Stokes	✓	
Steady Linearised Navier-Stokes		✓
Unsteady Stokes	✓	
Unsteady Navier-Stokes	✓	

Table 4.3: Relationship Between Equation and Solver Types.

Additionally solver types, coupled linear Navier-Stokes and velocity correction schemes, are explicitly

linked to equation types in the Nektar++ documentation. Therefore an indirect restriction on timing details is placed from the simulation types.

4.1.6 Driver Type

Drivers manage the simulation at a high-level. Within the incompressible Navier-Stokes solver five separate drivers exist. Specifying no driver is equivocal to selecting the standard driver. Selecting a different driver effectively changes the simulation type that is being run. Ergo if a standard driver is chosen then DNS is being performed whereas selecting the steady state indicates that a steady stability analysis is to be run. A brief synopsis is given by table 4.5.

Driver Type	DNS	Stability	Steady-State
Standard	✓	✓	
Adaptive	✓		
Arpack		✓	
Modified Arnoldi		✓	
Steady State			✓

Table 4.4: Relationship Between Driver and Simulation Types.

For DNS the available drivers are the standard and adaptive. Standard driver completes a number of tasks; prints a summary of conditions defined; sets up initial and boundary conditions; calls the solver type; and writes the outputs to a .fld. However if a user wishes to utilise adaptive polynomial ordering then they are to use the adaptive driver which is required when running quasi-dimension problems.

If a stability analysis is being run the drivers available, including standard, are the modified Arnoldi and Arpack. Modified Arnoldi driver computes a number of leading eigenvalues and eigen modes, thus making it suitable for stability analyses. Similarly the Arpack driver allows a solution of eigenmodes using a selection of other methods.

For steady state stability analysis the only option available is the steady state driver.

4.1.7 Evolution Operators

Evolution operators allow the study of the change in pertubations in stability analyses. Each operator slightly alters the Navier-Stokes equations and therefore they are exclusive to both forms of stability analysis.

For the adaptive SFD operator a number of additional parameters are required. Selecting this operator then opens a sub-tree from which these options can be defined, see figure 4.8. This again highlights how the XML schema can be used to bring variables in and out of scope.

Evolution Operator	Stability	Steady-State
Adjoint	✓	
Direct	✓	
NonLinear	✓	
Skew Symmetric	✓	
Transient Growth	✓	
Classical SFD		✓
Adaptive SFD		✓

Table 4.5: Relationship Between Driver and Simulation Types.

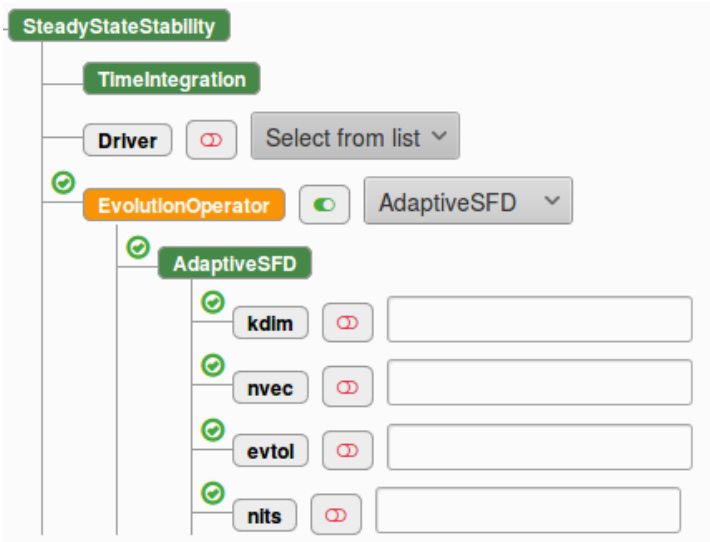


Figure 4.8: Snippet of Tree Showing Sub-trees for Specification of AdaptiveSFD Evolution Operator.

4.2 Constraints Graph

Visual tools are often the most effective at simply conveying the relationship between nodes in a network. Therefore a chord diagram was created, detailing the relationship between the parameter available within the incompressible flow solver.

However simply connecting parameters together does not imply any directionality to the decisions. Therefore an interactive visualisation has been created which show the relationship of parameter pairs, and also gives a directionality between these relations. An example image of the chord graph, and instructions in how to access it is given in appendix B.

Figure 4.9 shows a snippet of this interactive diagram. A user can select any parameter, with like parameters being grouped together. Once a parameter has been selected the links between related parameters are illuminated. Green shows that a link is dependent upon the selected parameters, speak this is a downwind selection. Whereas blue shows that the linked node is upstream of the selected parameter.

In the figure 4.9 the user has selected quasi dimensions. The blue links show that in order to select this the user should have chosen either 2D or 3D as the dimension field. The green shows that now the user can choose from either the coupled linear Navier-Stokes solver or velocity correction scheme.

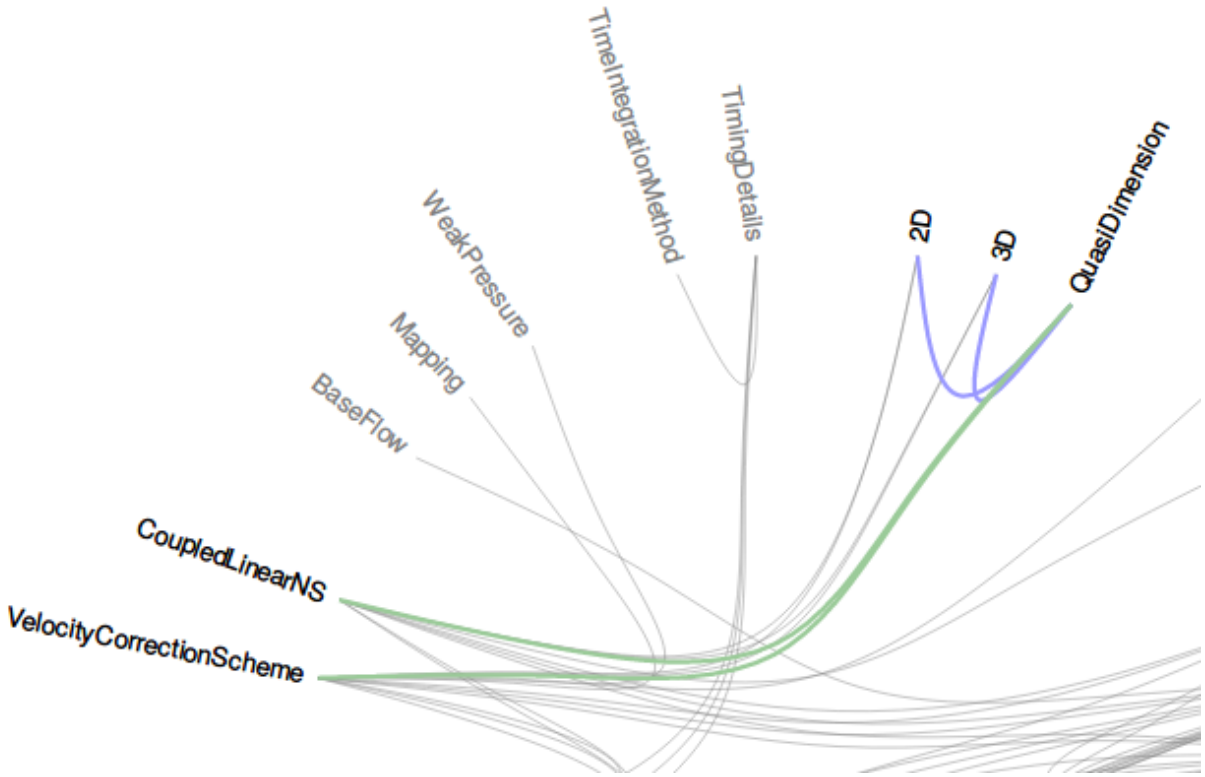


Figure 4.9: Excerpt of Constraint Chord Diagram Showing Directionality of Choices.

Moving then to the next option a user can see which other parameters become available and slowly navigate their way through the key decisions.

This tool is not only helpful in encapsulating these relationships clearly, but through definition of a simple `.json` file new relationships can be rapidly visualised.

Chapter 5

Testing and Results

TemPSS must be relied upon to accurately and consistently create functioning input files for Nektar++, which also prevent logical inconsistencies from being defined. To test that TemPSS is correctly generating XMLs, a number of test cases have been developed. These cases represent between them a wide variety of functionality of the incompressible Navier-Stokes solver.

- Flow past a NACA aerofoil
- Simulation of T106a turbine blade using quasi-dimension
- Fully-developed turbulent pipe flow

Developing the input files using TemPSS the results are then compared against the original simulations to ensure they produce identical results.

5.1 NACA Aerofoil

Simulation of the flow past a NACA aerofoil, at Reynolds number $135,000$, has been performed. This case tests a number of key functions in the incompressible Navier-Stokes solver. Moreover many of the potential Nekkloud users would wish to simulate flows past aerofoils and other aerodynamic components, thus demonstrating this capability is key to increasing user numbers.

Boundary	Type
1	Inlet
2	Periodic with 6
3	Periodic with 5
4	Outlet
5	Periodic with 3
6	Periodic with 2
7	Wall

Table 5.1: Boundary Conditions of the NACA Aerofoil Simulation.

Domain of the simulated NACA aerofoil is shown in figure 5.1, with the boundaries labelled described in table 5.1. To ensure that the simulations are being defined correctly two simulations have been run. The first the original input file defined manually, followed by the same simulation generated through TemPSS. Using the aerodynamic forces filter the lift and drag acting on the body has been calculated at a number of time intervals.

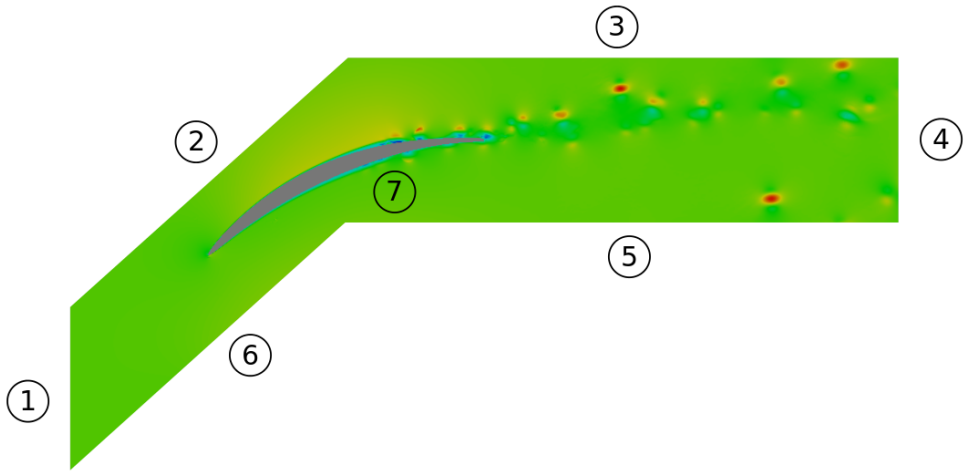


Figure 5.1: Domain of the NACA Aerofoil.

Running for a total of 5000 time intervals the aerodynamic forces were measured every 1000 iterations. Results for both the original input file and that generated through TemPSS are list in table C.2. Data from this table is subsequently visualised in figure 5.2 in which it can be clearly seen that both the drag forces for the original and TemPSS generated inputs are identical and thus for such cases TemPSS is working as expected.

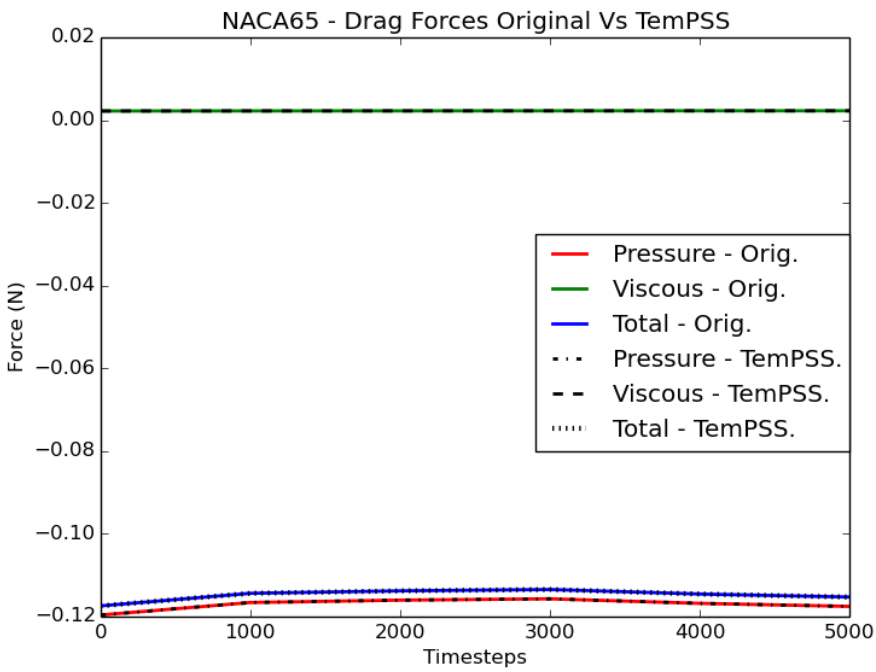


Figure 5.2: Comparison of Drag for Original and TemPSS Generated Input Files of NACA65.

5.2 T106a Quasi-3D Turbine Blade

The case T106a simulates a turbine blade, pictured in figure 5.3, at Reynolds number $50,000$ and an angle of attack of $37.7°$. Unique to this simulation is that a Fourier expansion is used to simulate in the homogeneous component.

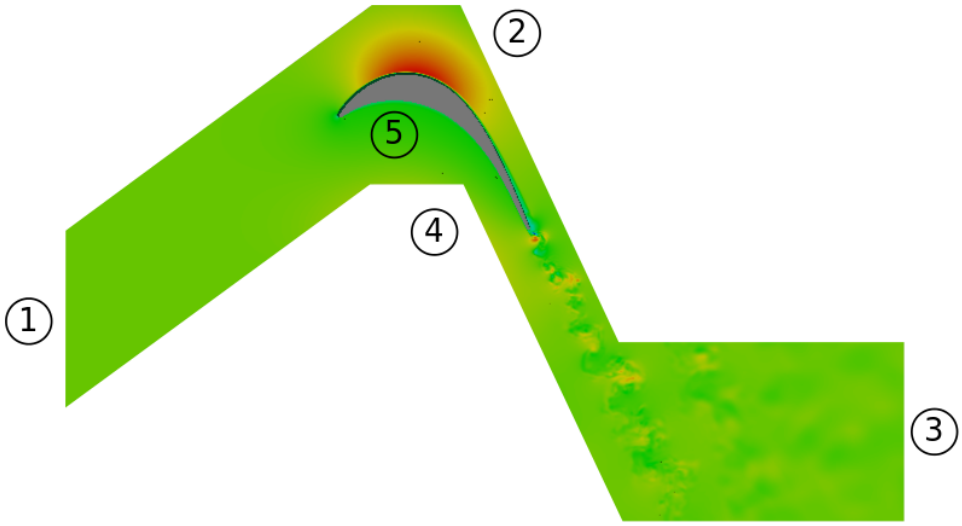


Figure 5.3: Domain of the T106a Turbine Blade.

As with the simulation of the NACA65 profile above, TemPSS has been verified through the comparison of the drag and lift forces. Running for 20 timesteps only, and writing every 10, the drag force is shown in figure 5.4. Additionally appendix D details the exact figures for both lift and drag from the original and TemPSS generated inputs. Again the drag and lift forces match identically so it can be infered that TemPSS is correctly generating the input XMLs.

Boundary	Type
1	Inlet
2	Periodic with 4
3	Outlet
4	Periodic with 2
5	Wall

Table 5.2: Boundary Conditions of the T106a Turbine Blade Simulation.

5.3 Turbulent Pipe Flow

A fully-developed turbulent pipe flow, with a periodic boundary condition, has also been simulated. Pipe flow is one of the canonical flow cases and offers insight into many aspects of fundamental fluid mechanics, such as turbulence transition. This simulation tests the ability of TemPSS to start a classical DNS simulation with a fully turbulent restart file as the initial conditions.

Checking that both input files are running an identical simulation the modal energy with time has been measured for each timestep. Shown in figure 5.6 it is seen that both the original and TemPSS

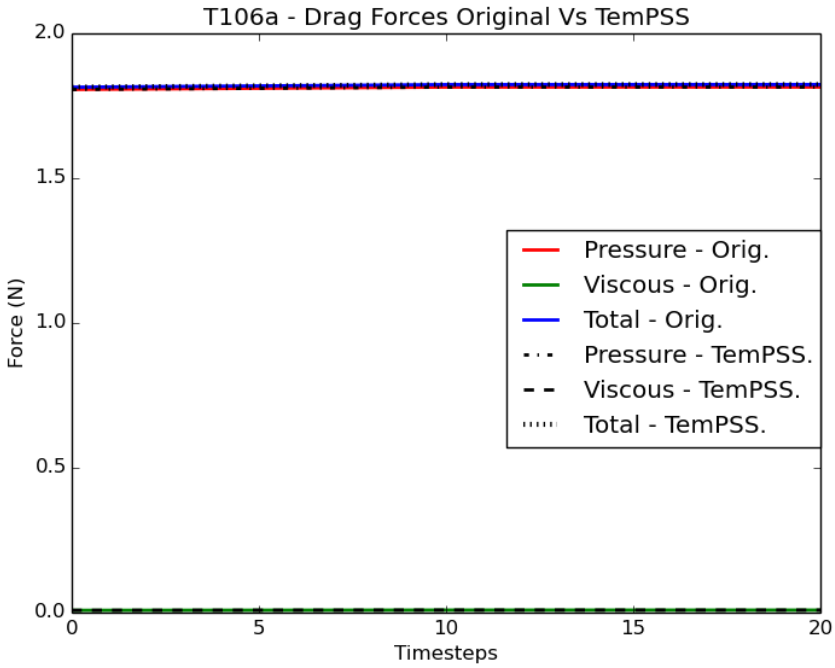


Figure 5.4: Comparison of Drag for Original and TemPSS Generated Input Files of T106a.

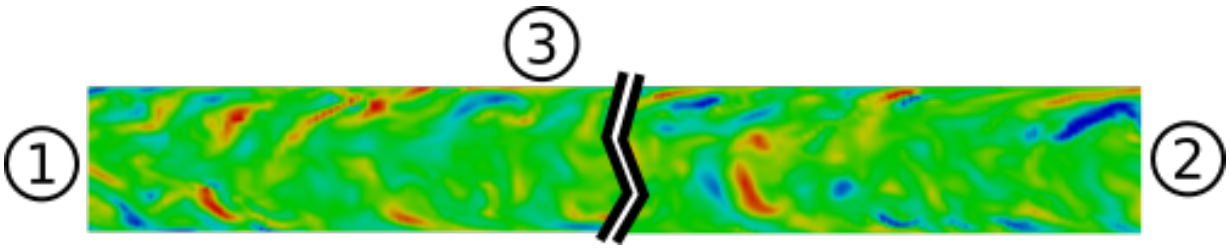


Figure 5.5: Domain of Turbulent Pipe.

input have the same modal energies for the entire 1400+ timesteps of the simulation. The same procedure was performed for the simulation enstrophy and kinetic energies, giving an identical result.

Boundary	Type
1	Periodic with 2
2	Periodic with 1
3	Wall

Table 5.3: Boundary Conditions of Pipe Simulation .

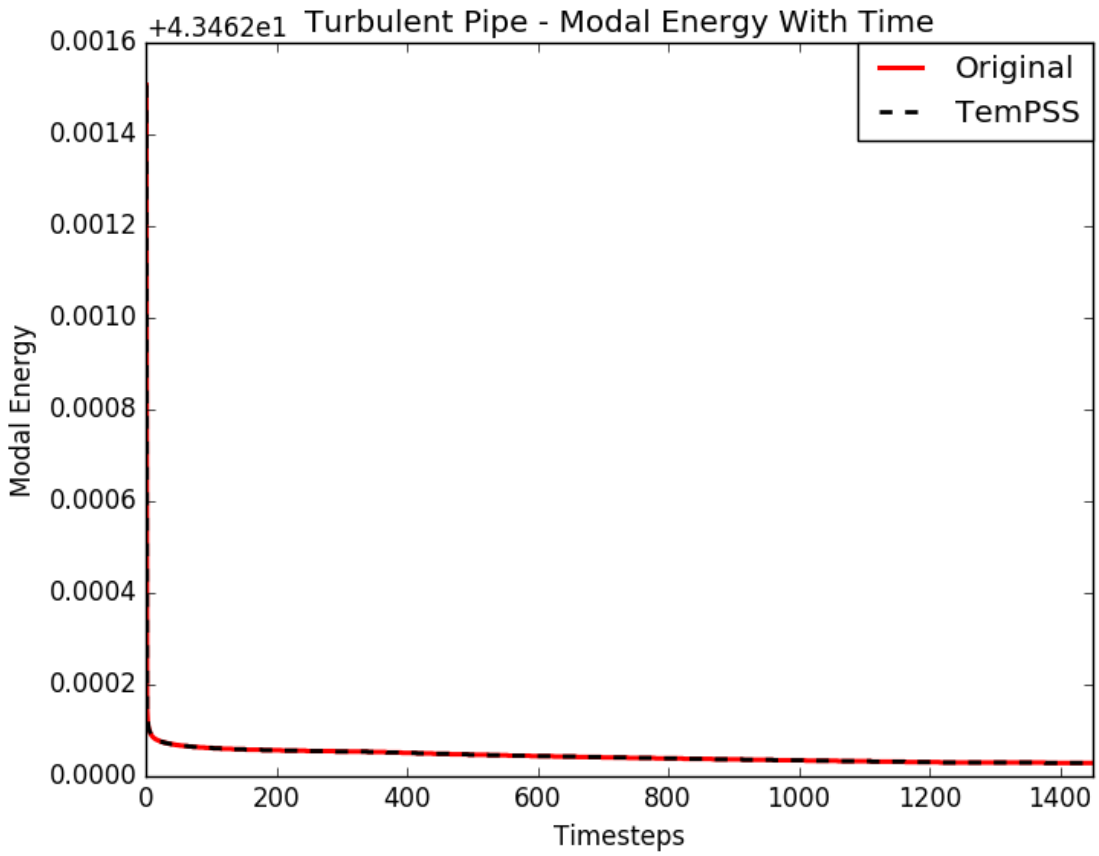


Figure 5.6: Modal Energy of Turbulent Pipe.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

This report details the development and validation of a web-based graphical user interface named TemPSS. TemPSS generates input files for the incompressible flow solver in Nektar++. In aim of this the the incompressible Navier-Stokes solver parameter space has been encapsulated, mapped, and organised for intuitive operation of the graphical user interface. Included within this is the testing of the interface by a number of Nektar++ users. Additionally the relationships of between functions have been determined, mapped and visualised interactively, see appendix B.

With respect to the original goals of the project, as defined in section 2, the project can be deemed a success. However there is scope for further work and a number of recommendations for which are now presented.

6.2 Further Work and Recommendations

The principal objective of Nekkloud is the expansion of the user base for Nektar++. To fulfil this goal Nekkloud; simplifies the user interface of the code; reduces the expertise required by individuals to produce an actionable result; and gives direct access to HPC facilities. Within this there are a number of suggestions which would aid all three of these requirements:

1. Improve and consolidate the user eco-system within Nekkloud/Nektar++
2. Continue to add templates to TemPSS
3. Add the ability to load separate sections of profiles

6.2.1 Development of Nekkloud/Nektar++ Eco-System

For users of Nektar++ outside of Imperial, or other supporting institutions, using the solvers requires following the documentation and tutorials available on the Nektar++ webpages. Whilst these re-

sources are sophisticated and cater to a wide variety cases, there are inevitably a number of situations which require greater elucidation and/or support. In order to deal with these issues the users are given access to the core Nektar++ development team through a mailing list.

The Nekkloud platform has the potential to improve the quality of the support greatly. Immediately Nekkloud gives users the ability to utilise the code without the need to install locally, as they will utilise the cloud instances of Nektar++. However the most powerful way to improve the eco-system would be the inclusion of a forum in lieu of a mailing list. This is as:

- The mailing list has poor search facilities, and search functions in a forum would allows users to see if the same problem has previously been covered.
- Queries posted to the mailing list are typically only answered by core Nektar++ developers. Using a forum allows for expertise generated within the user base to be leveraged in addition to the of the developers
- As part of the separation of expertise rationale public profiles are available in TemPSS. Users may need a description of puplic profiles and a place to discuss them

Creation of a user community would also distinguish Nektar++ from *competitor* codes. Many widely used codes do not have an online forum, however those that do have significantly higher user numbers.

6.2.2 Creation of Additional Templates

Within the scope of this project the only template developed was that of the incompressible Navier-Stokes solver. Though this is the most commonly used solver, addition of the compressible flow solver is an extremely important functionality for Nekkloud.

Though in effect the difference between the incompressible and compresible solvers is very little, as functions such as filters, optimisation, and so on are identical. There are however a number of differences in the equations used and the field which must be solver. Adding the compressible solver, and others, would need experienced use of the XML structure as to maximise the amount of code common to both solvers.

6.2.3 Partial Profiles

Part of the overall objective of this project is to separate the expertise required to run Nektar++. Though the ability to load pre-filled profiles edit and save the new versions is currently possible. It could be imagined that a user may wish to start a simulation from scratch knowing only the key parameters, i.e. those in the problem specification branch. Subsequently they could look through the available public profiles, select a branch from one and load it into their tree, perhaps then them may wish to repeat this. This functionality would greatly increase the speed at which simulations could be defined.

Bibliography

[1] Amazon Web Services Inc. Amazon elastic compute cloud (amazon ec2), cloud computing servers. *https://aws.amazon.com/ec2*, accessed 1st June 2017.

[2] Google Inc. Google compute engine, iaas services. *https://cloud.google.com/compute/*, accessed 1st July 2017.

[3] A. Comerford A. Bolis G. Rocco G. Mengaldo D. De Grazia S. Yakovlev J-E. Lombard D. Ekelschot B. Jordi H. Xu Y. Mohamied C. Eskilsson B. Nelson P. Vos C. Biotto R. M. Kirby C. D. Cantwell, D. Moxey and S. J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer physics communications*, 192:205–219, 2015.

[4] D. Solo, J. Reagle, and D Eastlake. (extensible markup language) xml-signature syntax and processing. *RFC Editor*, 2002.

[5] J. Cohen, D. Moxey, C. Cantwell, P. Burovskiy, J. Darlington, and S. J. Sherwin. Nekkloud: A software environment for high-order finite element analysis on clusters and clouds. *IEEE International Conference on Cluster Computing (CLUSTER)*, 2013.

[6] P. Austing, C. Cantwell, J. Cohen, D Moxey, and J Nowell. Tempss: A web service for managing libhpc application parameter templates and profiles. *https://github.com/london-escience/tempss*, Accessed July 2017.

[7] C. W. Oseen. Über die stoke'sche formel und über eine verwandte aufgabe in der hydrodynamik. *Arkiv för matematik, astronomi och fysik*, 6, 1911.

Appendix A

TemPSS Tree Listing

These diagrams can be found at `https://github.com/maxrobot/chord_diag`. Instructions are listed in `README.md`.

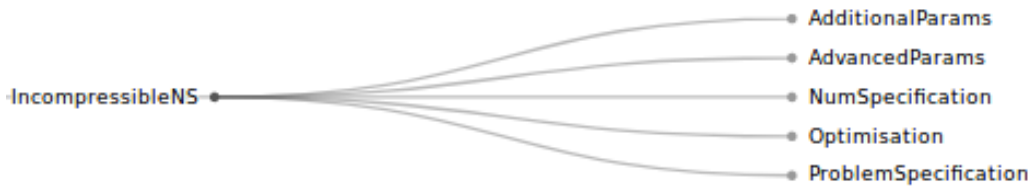


Figure A.1: Diagrammatical Overview of Incompressible Navier-Stokes TemPSS Tree.

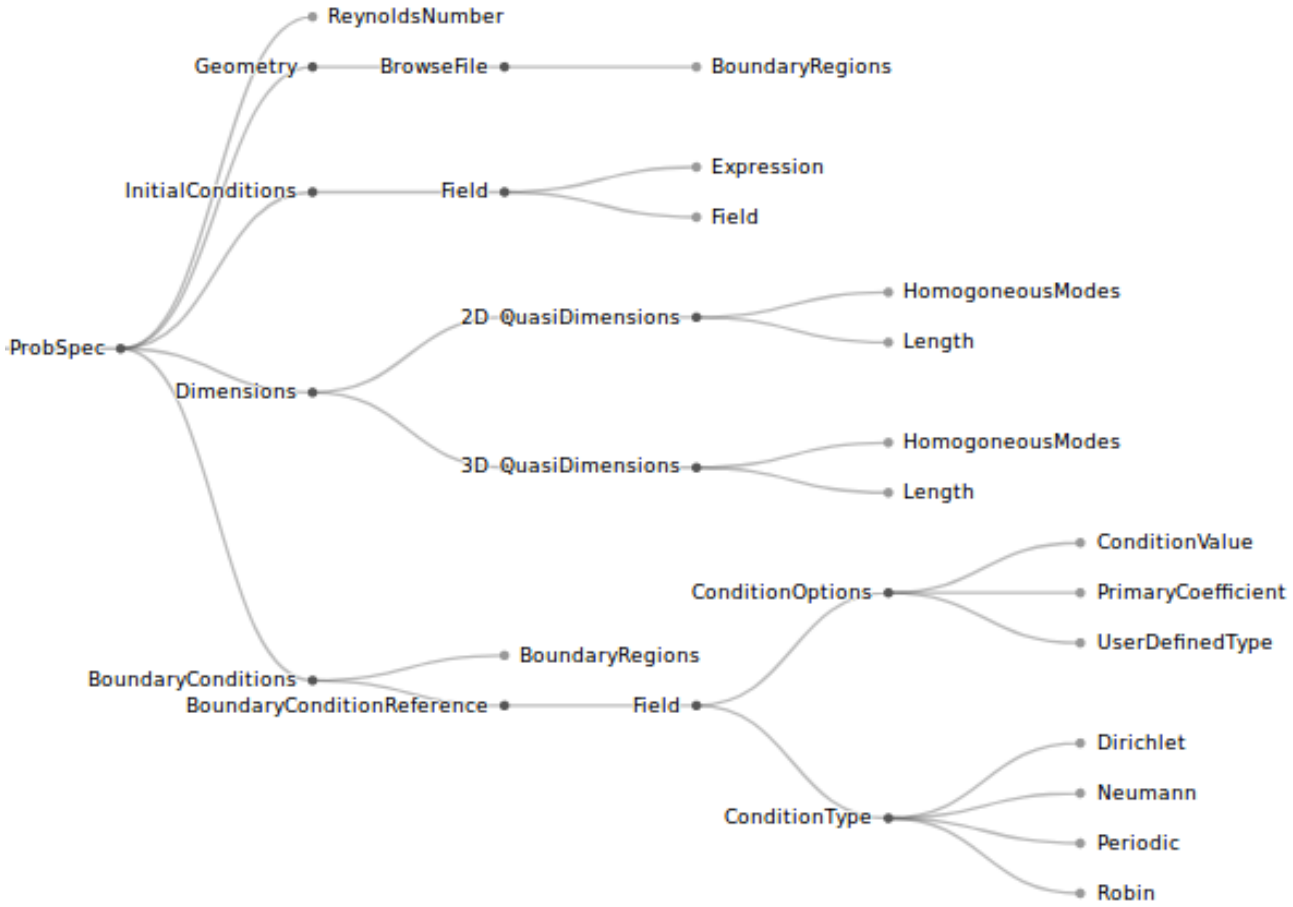


Figure A.2: Problem Specification Branch for Incompressible Navier-Stokes TemPSS Tree.

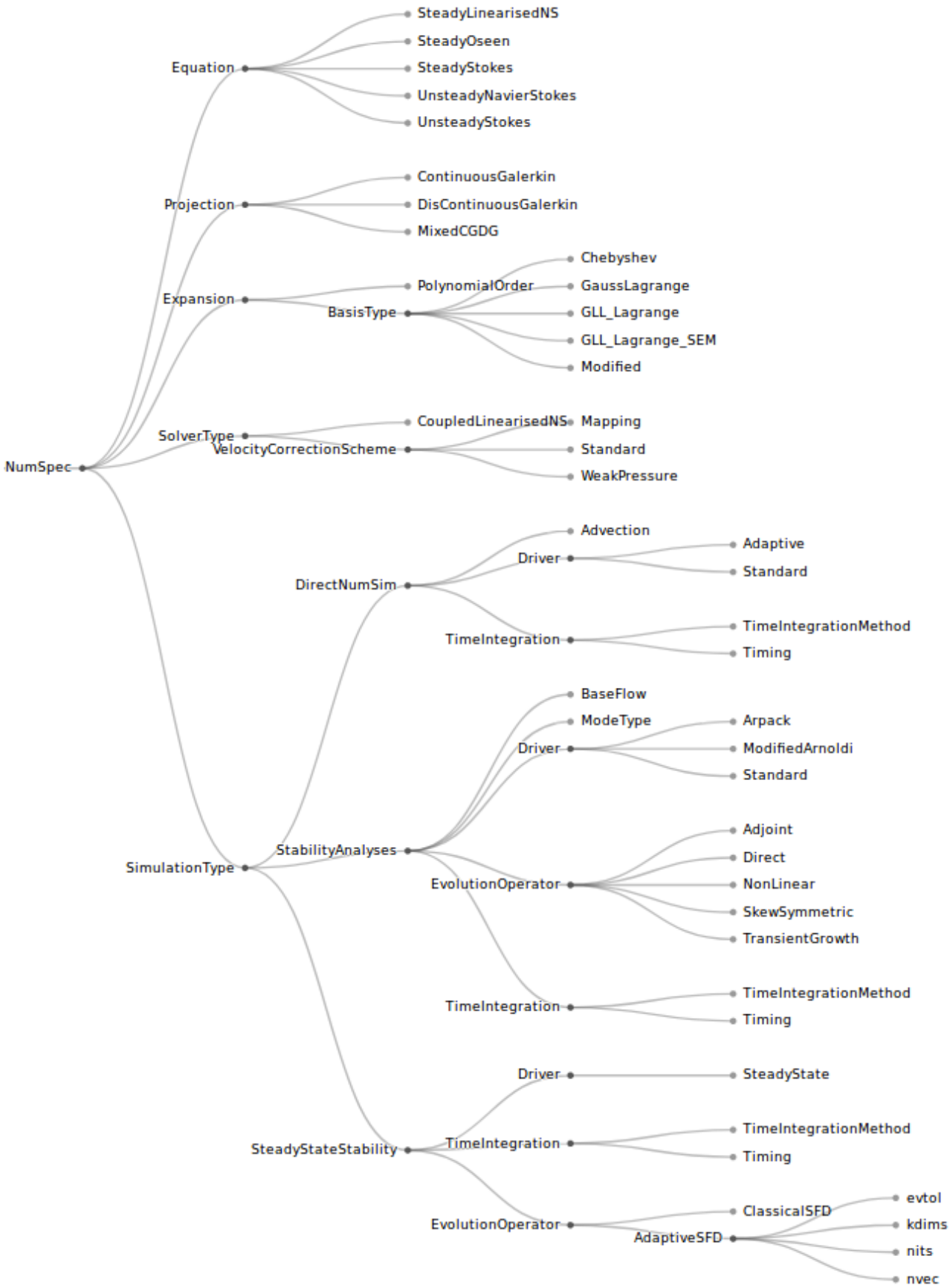


Figure A.3: Numerical Specification Branch for Incompressible Navier-Stokes TemPSS Tree.

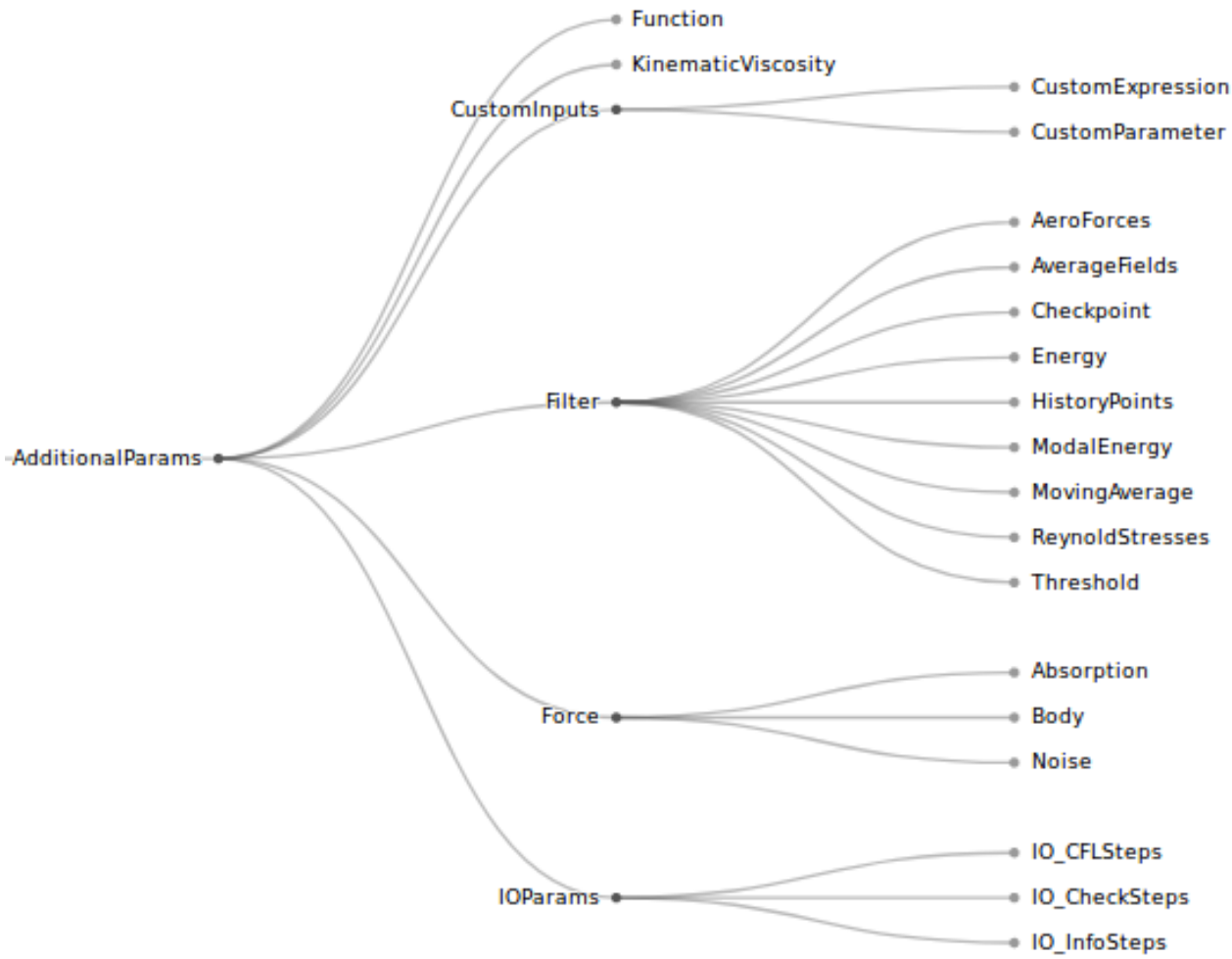


Figure A.4: Additional Parameters Branch for Incompressible Navier-Stokes TemPSS Tree.

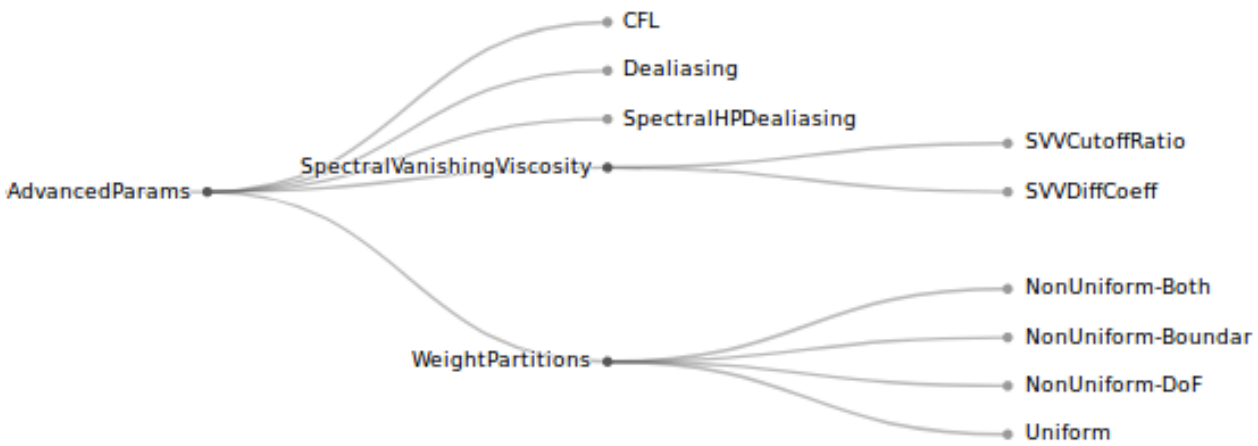


Figure A.5: Advanced Parameters Branch for Incompressible Navier-Stokes TemPSS Tree.

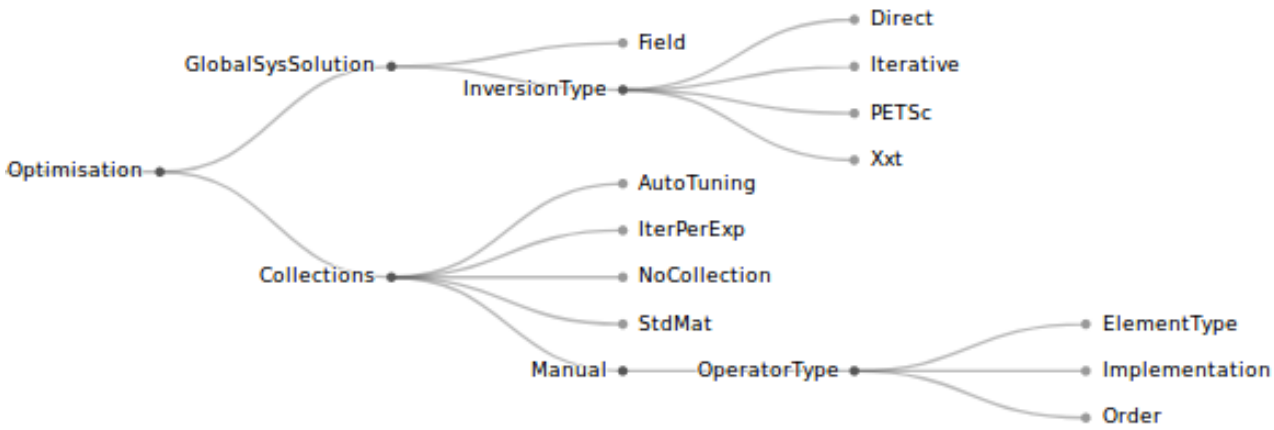


Figure A.6: Optimisation Branch for Incompressible Navier-Stokes TemPSS Tree.

Appendix B

Interactive Constraint Visualiser

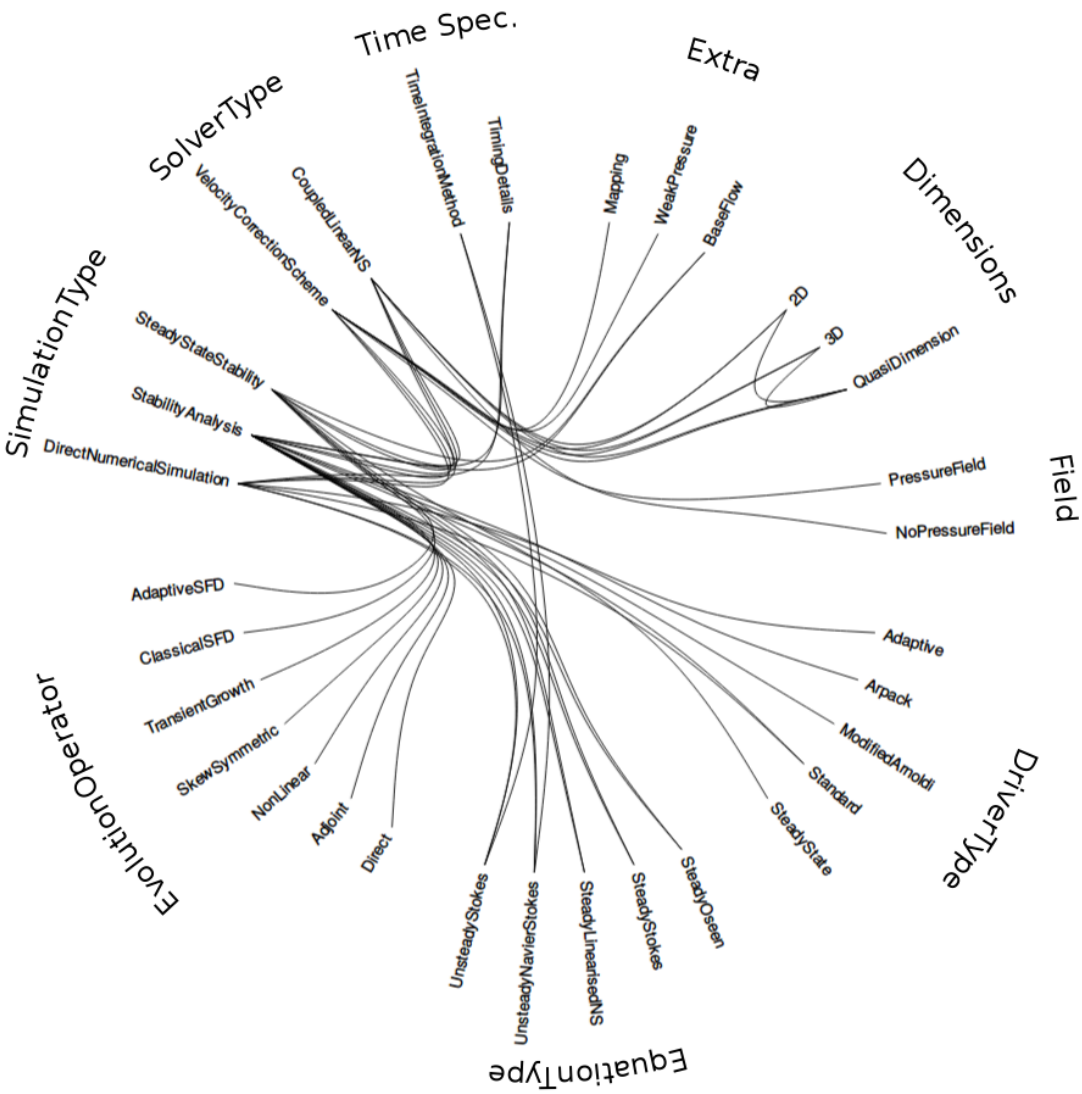


Figure B.1: Mapping of constraints between function for Nektar++ Incompressible Navier-Stokes Solver.

Figure B.1 shows a still image of an interactive the map of relationships between functions available

in Nektar++. Using an interactive map allowed the easy visualisation, and editing, of relationships between functions with directionality of dependence. Hovering over a function, with a mouse cursor, a green line is illuminated to the functions which are dependent on the function being chosen. Similarly a blue line shows backwards which functions are required to use a certain functionality.

This diagram can be found at `https://github.com/maxrobot/chord_diag`. Instructions are listed in `README.md`.

Appendix C

NACA65 - Drag and Lift Force Measurement Comparison

Timestep	Pressure x-axis	Viscous x-axis	Total x-axis	Pressure y-axis	Viscous y-axis	Total y-axis
0	-0.11962121	0.002211952	-0.11740926	0.23502283	0.0014489827	0.23647181
1000	-0.11659737	0.0022292138	-0.11436815	0.23281936	0.0014352338	0.2342546
2000	-0.11602215	0.002242833	-0.11377932	0.23268846	0.0014328509	0.23412131
3000	-0.11570363	0.0022527072	-0.11345093	0.23299908	0.0014310583	0.23443014
4000	-0.11679946	0.0022576795	-0.11454179	0.23428351	0.0014288948	0.2357124
5000	-0.11753313	0.0022588983	-0.11527424	0.23468889	0.0014266195	0.23611551

Table C.1: Aeroforces NACA65 Original Input

Timestep	Pressure x-axis	Viscous x-axis	Total x-axis	Pressure y-axis	Viscous y-axis	Total y-axis
0	-0.11962121	0.002211952	-0.11740926	0.23502283	0.0014489827	0.23647181
1000	-0.11659737	0.0022292138	-0.11436815	0.23281936	0.0014352338	0.2342546
2000	-0.11602215	0.002242833	-0.11377932	0.23268846	0.0014328509	0.23412131
3000	-0.11570363	0.0022527072	-0.11345093	0.23299908	0.0014310583	0.23443014
4000	-0.11679946	0.0022576795	-0.11454179	0.23428351	0.0014288948	0.2357124
5000	-0.11753313	0.0022588983	-0.11527424	0.23468889	0.0014266195	0.23611551

Table C.2: Aeroforces NACA65 TemPSS Generated Input

Appendix D

T106a - Drag and Lift Force Measurement Comparison

Timestep	Pressure x-axis	Viscous x-axis	Total x-axis	Pressure y-axis	Viscous y-axis	Total y-axis
0	1.8063751	0.0078590878	1.8142342	0.69212451	-0.0080855233	0.68403899
10	1.8144775	0.0086553326	1.8231328	0.69820648	-0.0094729673	0.68873351
20	1.8146635	0.0085670641	1.8232306	0.69832116	-0.0093212748	0.68899988

Table D.1: Aeroforces T106a Original Input

Timestep	Pressure x-axis	Viscous x-axis	Total x-axis	Pressure y-axis	Viscous y-axis	Total y-axis
0	1.8063751	0.0078590878	1.8142342	0.69212451	-0.0080855233	0.68403899
10	1.8144775	0.0086553326	1.8231328	0.69820648	-0.0094729673	0.68873351
20	1.8146635	0.0085670641	1.8232306	0.69832116	-0.0093212748	0.68899988

Table D.2: Aeroforces T106a TemPSS Generated Input

Appendix E

Test TemPSS User Remarks

E.1 User 1

Question 1: What problem are you tring to solver?
I am always using the incompressible NS solver for problems of turbomachinery interest. I focus on both compressor and turbine blades, and for now I always use the 2.5D formulation with the Fourier expansion in the homogeneous direction. I always have some filters turned on, such as average fields, Reynolds stresses, history points and aerodynamic forces.
Question 2: Were you able to utilise TemPSS unaided?
I think with your explanation going through it the first time I was able to use it unaided, and it would actually be of help in case I had to create a case from scratch. In fact, I have never had to do so, always having a previous case from which to keep the structure of the session file.
Question 3: Do you have any recommendations?
The only recommendations would be aimed at speeding up the creation process the first time, but I do think it will be hard to figure out a way to do so - I'd say it's a pretty good tool already!

E.2 User 2

Question 1: What problem are you tring to solver?
I am studying the vortex interactions in the wake of a low incidence delta wing. Some PIV experiment was made a few years ago in Cambridge, and I am using the provided data to define my inlet boundary condition (velocity components in the plane straight after the wing). this boundary condition also defines my initial condition for convergence speed purpose. My domain of study is a box of size $1m * 1m * 1.18m$, meshed with 220+k tetrahedrons. My git branch is feature IncNS Sensor. The essential features used (apart from the common numerical definitions) are SpectralhpDealiasing, SpectralVanishingViscosity set to 'DGKernel' (feature not officially released) as well as GlobalSysSoln refinements. I am also using the History Points feature to study the velocity components in the vortex

cores. I mostly run fifth and seventh order simulations (Nummodes = 6 and 8).

Question 2: Were you able to utilise TemPSS unaided?

I was able to use the software unaided.

Question 3: Do you have any recommendations?

Ensure that you do not have to click too specifically on the buttons. Also include the filter history points please.

E.3 User 3

Question 1: What problem are you tring to solver?

My cases are that of stability past a cylinder using eigenvalue analysis.

Question 2: Were you able to utilise TemPSS unaided?

Using the interface to define the numerical methods was fine but I had some problem defining the boundary fields.

Question 3: Do you have any recommendations?

It would be useful if there was a way to make the boundary condition definition more rapid, because it is very slow to fill them out one by one. But actually it is better than using the terminal for the first time.