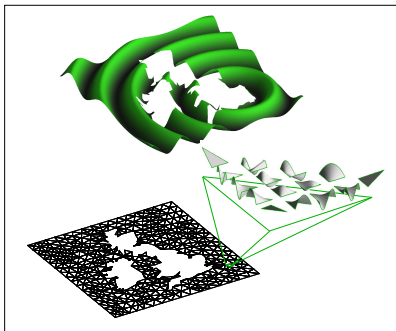


Nektar++: A Practical Introduction



Chris Cantwell

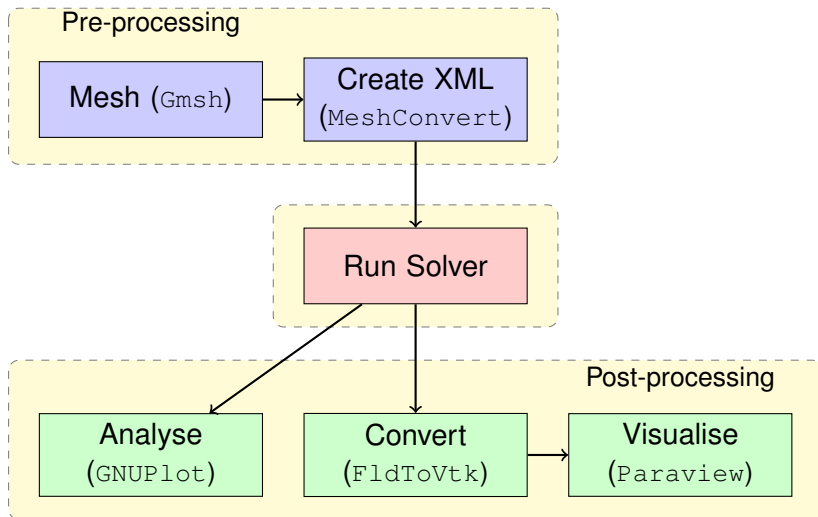
23rd August 2011

Nektar++: A Practical Introduction

- ▶ Introduce the *Nektar++* solvers and their usage.
- ▶ Outline
 - ▶ Overview of work-flow
 - ▶ XML session files
 - ▶ Defining a problem
 - ▶ Mesh generation
 - ▶ Mesh XML definition
 - ▶ Specifying conditions
 - ▶ Choosing a solver and setting parameters
 - ▶ A simple demo: Helmholtz2D
 - ▶ Visualising output
 - ▶ Running the Incompressible Navier-Stokes solver
 - ▶ Simple unsteady simulation
 - ▶ Direct or adjoint stability analysis
 - ▶ Transient growth analysis
 - ▶ Using the Nektar++ C++ framework
- ▶ Hands-on tutorial.

Useful prerequisites: basic use of a Linux terminal

Overview of work-flow



XML Documents

- ▶ **eXtensible Markup Language** - similar concept to HTML.
- ▶ A hierarchical document
 - ▶ tags (XML 'elements'), attributes, content, comments.

```
<CONTACT ID="0">  
  <FORENAME>John</FORENAME>  
  <SURNAME>Smith</SURNAME>  
</CONTACT>
```

- ▶ Open tag with <[NAME]>
- ▶ Close tag with matching </[NAME]>
- ▶ Each tag has a name (no uniqueness requirement)
- ▶ Each tag may contain zero or more attributes (ID="0")
- ▶ Each tag may contain element text

```
<FORENAME>John</FORENAME>
```

- ▶ ... or other elements
(<CONTACT><FORENAME>...</FORENAME></CONTACT>)
- ▶ Comments: (<!-- This is a comment -->)

Nektar++ XML session file

```
<NEKTAR>
  <GEOMETRY DIM="1" SPACE="2">
    <VERTEX>
      ...
    </VERTEX>
    <ELEMENT>
      ...
    </ELEMENT>
    <COMPOSITE>
      ...
    </COMPOSITE>
    <DOMAIN> C[0] </DOMAIN>
  </GEOMETRY>
  <EXPANSIONS>
    <E COMPOSITE="C[0]" NUMMODES="8"
      FIELDS="u,v,p" TYPE="MODIFIED"/>
  </EXPANSIONS>
  <CONDITIONS>
    ...
  </CONDITONS>
</NEKTAR>
```

Mesh Generation

- ▶ For complex meshes define geometry in GMSH (`example.geo`)

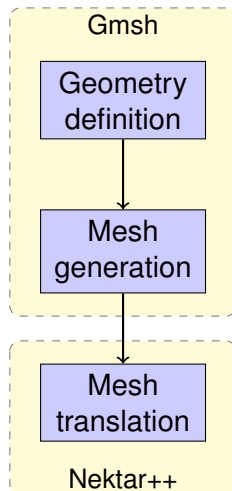
```
...  
Point(1) = {0,0,0};  
Line(1) = {16,29};  
Line Loop(54) = {1,-18,-17,16};  
Plane Surface(54) = {54};  
...
```

- ▶ Generate mesh (`example.msh`)

```
gmsht -2 example.geo
```

- ▶ Translate to XML (`example.xml`)

```
MeshConvert example.msh \  
example.xml
```



Mesh Definition

2D mesh is defined in terms of:

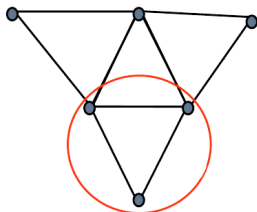
- ▶ Vertices
- ▶ Edges
- ▶ Elements

Each mesh entity has unique index

- ▶ Tag name specifies component type

Vertices have 3 coordinates (even in 2D)

```
<V ID="0"> 0.5 1.0 0.0 </V>
```



Mesh Definition

- ▶ Edges defined by two vertex IDs

```
<E ID="0"> 3 6 </E>
```

- ▶ Elements defined by set of edge IDs

```
<T ID="0"> 4 5 6 </T>
```

```
<Q ID="20"> 17 18 21 20 </Q>
```

- ▶ Composites define groups of entities
(must be of the same type)

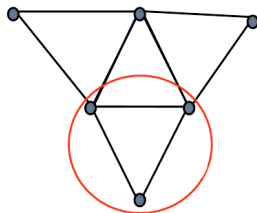
```
<C ID="0"> Q[0-4] </C>
```

```
<C ID="1"> T[5-10] </C>
```

```
<C ID="2"> E[0-8,12-16,18,19] </C>
```

- ▶ Domain defined by a set of
composites

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



Element types (tag names):

- ▶ Segment(<S>)
- ▶ Triangle (<T>)
- ▶ Quadrilateral(<Q>)
- ▶ Tetrahedron(<A>)
- ▶ Hexahedron(<H>)

Specifying Boundary Conditions

- ▶ Use composite to select boundary mesh elements

```
<C ID="2"> E[0-8,12-16,18,19] </C>
```

- ▶ Define **boundary regions** using these composites

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

- ▶ Define **boundary conditions** for each region

```
<BOUNDARYCONDITIONS>  
  <REGION REF="0">  
    <D VAR="u" VALUE="(1+y)*(1-y)" />  
    <D VAR="v" VALUE="0" />  
    <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />  
  </REGION>  
</BOUNDARYCONDITIONS>
```

- ▶ Types: Dirichlet(D), Neumann(N)
- ▶ Periodic boundaries: VALUE=[REF]

Specifying Initial Conditions

Functions define spatio-temporal data using expressions.

```
<FUNCTION NAME="InitialConditions"> <!-- Channel -->
  <E VAR="u" VALUE="(1+y)*(1-y)" />
  <E VAR="v" VALUE="0" />
  <E VAR="p" VALUE="0" />
</FUNCTION>
```

Can also be loaded from a Nektar++ field file.

```
<FUNCTION NAME="InitialConditions">
  <F FILE="channel.rst" />
</FUNCTION>
```

FUNCTIONs used for exact solutions, forcing, base flows, etc

```
<FUNCTION NAME="BaseFlow">
  <F FILE="bfs-tg.bse" />
</FUNCTION>
```

Choosing a solver and setting parameters

- **Parameters:** numeric constants of the problem.

```
<PARAMETERS>
  <P> TimeStep      = 0.002      </P>
  <P> NumSteps      = 500        </P>
  <P> IO_CheckSteps  = 1/TimeStep </P>
  <P> IO_InfoSteps   = 1         </P>
  <P> Re             = 500       </P>
  <P> Kinvis         = 1.0/Re    </P>
  <P> kdim           = 4         </P>
  <P> nvec           = 1         </P>
</PARAMETERS>
```

- **Solver properties:** specify broader features of the solver.

```
<SOLVERINFO>
  <I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes"/>
  <I PROPERTY="SOLVERTYPE"
    VALUE="VelocityCorrectionScheme"/>
  <I PROPERTY="EvolutionOperator" VALUE="Nonlinear"/>
  <I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder2"/>
  <I PROPERTY="Projection" VALUE="Continuous"/>
</SOLVERINFO>
```

Using Nektar++: Helmholtz2D Demo

Solve Helmholtz problem

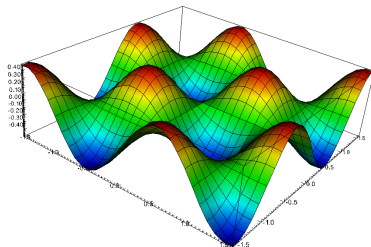
$$\nabla^2 u - \lambda u = f.$$

- Forcing function:

$$f = -(\lambda + 2\pi^2) \sin(\pi x) \sin(\pi y)$$

- Analytic solution:

$$u = \sin(\pi x) \sin(\pi y)$$



Specifying forcing and analytic solution:

```
<FUNCTION NAME="Forcing">  
  <E VAR="u" VALUE="-(Lambda + 2*PI*PI)*sin(PI*x)*sin(PI*y)" />  
</FUNCTION>  
  
<FUNCTION NAME="ExactSolution">  
  <E VAR="u" VALUE="sin(PI*x)*sin(PI*y)" />  
</FUNCTION>
```

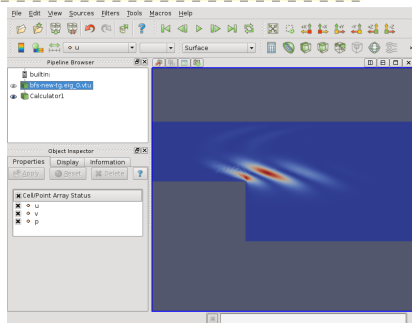
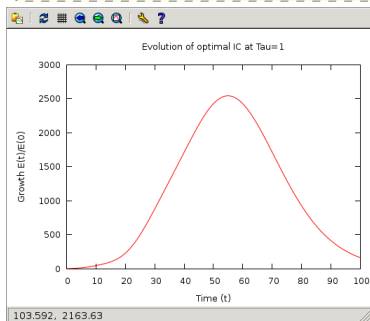
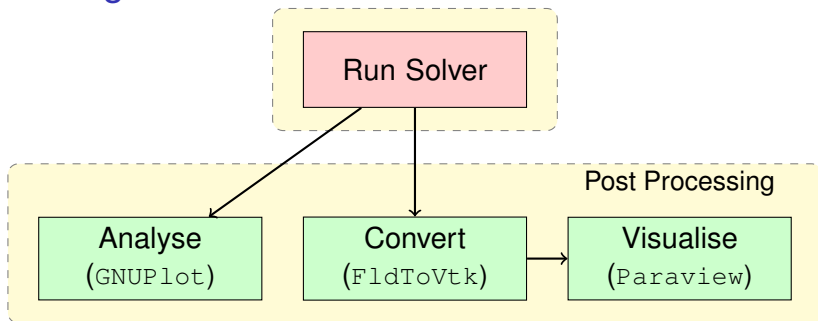
Using Nektar++: Helmholtz2D Demo

Run ADRSolver:

```
ADRSolver Test_HEL.xml
```

...and visualise output.

Visualising the results



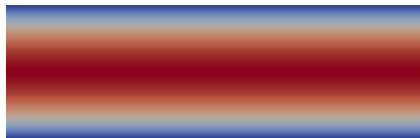
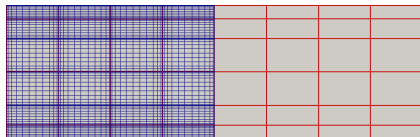
Nektar++ Incompressible Navier-Stokes Solver

Important solver properties:

```
<SOLVERINFO>
  <I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes" />
  <I PROPERTY="SolverType"
    VALUE="VelocityCorrectionScheme" />
  <I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder3" />
  <I PROPERTY="Driver" VALUE="Standard" />
  <I PROPERTY="EvolutionOperator" VALUE="Nonlinear" />
</SOLVERINFO>
```

Important parameters:

```
<PARAMETERS>
  <P> TimeStep = 0.001 </P>
  <P> NumSteps = 200000 </P>
  <P> Re = 7500 </P>
  <P> kinvis = 1/Re </P>
</PARAMETERS>
```



Run solver:

```
IncNavierStokesSolver channel.xml
```

Direct/Adjoint stability analysis - Channel

- ▶ Solver properties

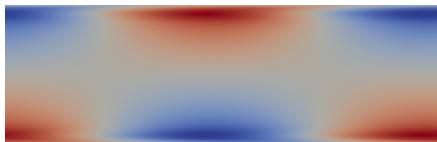
```
<I PROPERTY="Driver" VALUE="Arpack" />  
<I PROPERTY="EvolutionOperator" VALUE="Direct" />
```

- ▶ Parameters:

```
<P> FinalTime = 1 </P>  
<P> kdim = 16 </P>  
<P> nvec = 1 </P>  
<P> nits = 500 </P>  
<P> evtol = 1e-6 </P>
```

- ▶ Run solver:

```
IncNavierStokesSolver channel-direct.xml
```



Adjoint computed using EvolutionOperator set to Adjoint.

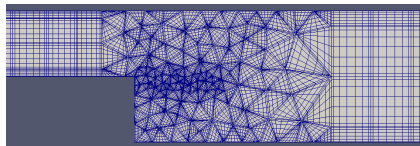
Transient growth analysis - Backward-facing Step

Solver properties

```
<I PROPERTY="Driver" VALUE="Arpack" />  
<I PROPERTY="EvolutionOperator" VALUE="TransientGrowth" />
```

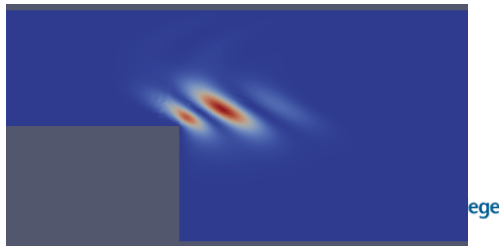
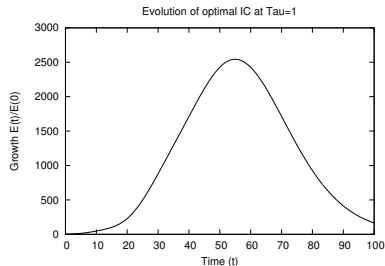
Parameters:

```
<P> FinalTime = 1 </P>  
<P> kdim = 4 </P>  
<P> nvec = 1 </P>  
<P> nits = 500 </P>  
<P> evtol = 1e-6 </P>
```



Run solver:

IncNavierStokesSolver bfs-tg.xml



Using the Nektar++ C++ framework

Several components:

- ▶ **Libraries**

Framework (data structures, algorithms) for Spectral/*hp* element methods. Problem independent.

- ▶ **Solvers**

Collection of end-user applications, implementing solvers for several PDEs.

- ▶ **Utilities**

End-user tools for manipulating input and output files for Nektar++.

- ▶ **RegressionTests**

Sequence of tests to verify correct functionality.

Using Nektar++: Basic Ingredients in 2D

Some principle components for using the Nektar++ framework:

- ▶ Session (LibUtilities)

```
LibUtilities::SessionReaderSharedPtr vSession  
    = LibUtilities::SessionReader::CreateInstance(argc, argv);
```

- ▶ Geometry and expansions definition (SpatialDomains)

```
SpatialDomains::MeshGraphSharedPtr graph2D  
    = MemoryManager<SpatialDomains::MeshGraph2D>::  
        AllocateSharedPtr(vSession);
```

- ▶ Continuous Galerkin solution field (MultiRegions)

```
MultiRegions::ExpListSharedPtr Exp;  
std::string variable = vSession->GetVariable(0);  
Exp = MemoryManager<MultiRegions::ContField2D>::  
        AllocateSharedPtr(vSession, graph2D, variable);
```

OR Discontinuous Galerkin solution field (MultiRegions)

```
Exp = MemoryManager<MultiRegions::DisContField2D>::  
        AllocateSharedPtr(vSession, graph2D, variable);
```

Final Practical Advice

Mesh Generation

- ▶ Refinement sufficient to capture flow features
- ▶ Domain size large enough not to distort base flow.
- ▶ Domain long enough for time horizon when computing transient growth.
- ▶ Avoid large changes of size between elements.
- ▶ Using elements which are too small imposes CFL restrictions.

Stability calculations

- ▶ Base flow needs to have completely converged onto a steady state.
- ▶ Arnoldi iterations typically take a long time (hours).
- ▶ Krylov dimension large for direct/adjoint (≈ 100)
- ▶ Krylov dimension smaller for transient growth (≈ 4)

Further Information

- ▶ **Nektar++ website:** `www.nektar.info`

- ▶ Example usage
- ▶ Educational Material
- ▶ Code documentation (Doxygen)

- ▶ **Code Examples:** `Nektar++/library/Demos/`

- ▶ **Book:**

G.E. Karniadakis and S.J. Sherwin, *Spectral/hp element methods for computational fluid dynamics (2nd ed.)*, Oxford University Press (2005).

- ▶ **Papers:**

- ▶ *A Generic Framework for Time-Stepping PDEs: general linear methods, object-orientated implementation and application to fluid problems*, Peter E.J. Vos, Sehun Chun, Alessandro Bolis, Claes Eskilsson, Robert M. Kirby and Spencer J. Sherwin , Int J. CFD, Submitted , 2011
- ▶ ... more on the website...

`http://www2.imperial.ac.uk/ssherw/spectralhp/pubs/`