Nektar++ Tutorial

February 7, 2011

The aim of this tutorial is to introduce the user to the spectral/*hp* element framework *Nektar++* and its use. This guide assumes the user has successfully compiled the libraries, the solvers and the utilities, as explained on the website[1]. A series of regression tests are included to check that the software is producing the expected results. Please ensure these all pass before continuing.

In this first section we will create a simple 2D mesh using *Gmsh* and convert it into a suitable input format for the *Nektar++* libraries to process. Subsequently, we will use this mesh to solve a number of common PDEs already supported by the solvers provided with *Nektar++* , namely:

1. the Helmholtz equation,
2. the Unsteady Advection-Diffusion equation,
3. the Incompressible Navier-Stokes equations.

The toy problems discussed initially should aid the user in understanding how to specify the computational domain (mesh) on which to solve a set of one or more partial differential equations as well as any necessary parameters, boundary conditions and initial conditions.

The last step of the tutorial is an example of a more substantial fluid dynamics problem, the flow past a cylinder. The point of presenting this typical problem is to introduce some more advanced features including the definition of curved elements and the use of restart files.

1 Geometry and Mesh

We start creating a very simple geometry. A square which we will mesh with 16 quadrilateral elements. The square is of size $[-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$.

In the tutorial folder `NekTutorial/Tutorial/SquareMesh/Geometry/` there are the following files

- `Square.geo` - this is the file containing the geometry specification defined in terms of *Gmsh* commands.
- `Square.msh` - *Gmsh* generated mesh data listing mesh vertices and elements. This is the mesh file understood by the *Nektar++* pre-processing utilities.

[1] www.nektar.info

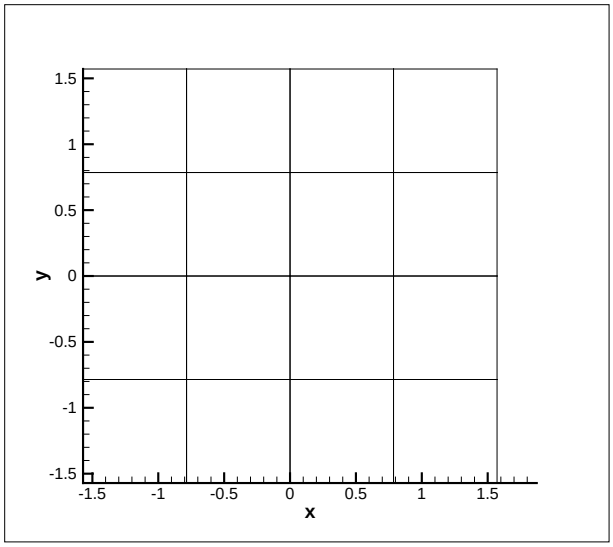

Figure 1: 16 quadrilaterals mesh

- `Square.xml` - *Nektar++* session file generated from `Square.msh` using a *Nektar++* utility. This contains only the mesh data at this stage.

If you have *Gmsh* installed, you can generate the `.msh` file yourself from the terminal using the command:

```
gmsh -2 Square.geo
```

or via the graphical user interface. However, we will not discuss the use of *Gmsh* in this tutorial. `Square.xml` has been generated using the `MeshConvert` pre-processing tool in the `utilities/builds/PreProcessing/` directory. To generate the *.xml* file from the `.msh` file, run the command

```
./MeshConvert Square.msh Square.xml
```

Figure 1 shows the resulting mesh on which we are going to solve a number of PDEs. You can examine the file

```
NekTutorial/Tutorial/SquareMesh/Geometry/Square.xml
```

to see how the various mesh entities have been defined in the *Nektar++* format.

2 Helmholtz equation

We begin by solving the 2D Helmholtz equation on the square domain:

$$\nabla^2 u + \lambda u = f \quad u(x, y) \in \Omega \tag{1}$$

with smooth forcing function

$$f(x, y) = -(\lambda + 2\pi^2) \sin(\pi x) \sin(\pi y). \tag{2}$$

This has the analytic solution

$$u_{ex}(x, y) = \sin(\pi x) \sin(\pi y). \tag{3}$$

The following files can be found in `NekTutorial/Tutorial/SquareMesh/Helmholtz/`:

- `Test_HEL.xml` - the *Nektar++* session file containing the geometry above and the necessary parameters to solve the Helmholtz problem;

The `GEOMETRY` section defines the mesh from the previous section. The expansion type and order is specified in an `EXPANSIONS` section. An expansion basis is applied to a geometry composite specified in the `GEOMETRY` section. A default entry is included by `MeshConvert`. In this case, `C[0]` refers to the set of all elements. The `TYPE` specifies the choice of polynomial functions to use in the expansion. Alternatively, for example, one might choose `FOURIER` or `CHEBYSHEV`. `MODIFIED` refers to a basis of Legendre polynomials modified to enable boundary/interior decomposition.

```
<EXPANSIONS>
    <E COMPOSITE="C[0]" NUMMODES="8" TYPE="MODIFIED"/>
</EXPANSIONS>
```

If we examine `Test_HEL.xml`, we can see where we can define the conditions which define the particular problem to solve. These are all enclosed in a `CONDITIONS` section. This section contains a number of things:

1. Solver information such as the equation type and the projection type (`Continuous` or `Discontinuous` Galerkin), along with problem parameters. Whilst solver properties are specified as quoted attributes and have the form

   ```
   <I PROPERTY="[STRING]" VALUE="[STRING]" />
   ```

   the parameters are specified as name-value pairs:

   ```
   <P> [KEY] = [VALUE] </P>
   ```

   Parameters may be freely used either from within the solver (e.g. `Lambda`), or within other expressions, such as function defintions or other parameters defined subsequently.

   Todo: Define two `SOLVERINFO` properties to set the `EQTYPE` to `Helmholtz` and the `Projection` property to `Continuous`. Then add two `PARAMETERS`, wavenumber $k = \pi$ and `Lambda` $= 1.0$.

   Note: π is a known constant called `PI`.

2. The declaration of the variable(s).

```
<VARIABLES>
    <V ID="0"> u </V>
</VARIABLES>
```

3. The specification of boundary regions in terms of composites defined in the geometry and the conditions applied on those boundaries. Boundary regions have the form

```
<B ID="[INDEX]"> [COMPOSITE-ID] </B>
```

Todo: Define a boundary region with index 0 to be the composite corresponding to the boundary edges of the domain.

The boundary conditions enforced on a region take the following format for one or more variable names specified in the `VARIABLES` section. The `REF` attribute for a boundary condition region should correspond to the `ID` of the desired `BOUNDARYREGION`.

```
<REGION REF="[B-REGION-INDEX]">
    <[TYPE] VAR="[VARIABLE]" VALUE="[EXPRESSION]">
    ...
</REGION>
```

Todo: Specify appropriate Dirichlet boundary conditions for the Helmholtz problem for the variable u on the boundary.

4. The definition of the forcing term, f, and the exact solution. A forcing term has the form

```
<F VAR="[VARIABLE]" VALUE="[EXPRESSION]">
```

Todo: Define the expression for f in the Helmholtz equation above. Remember that the expression may contain names of `PARAMETERS`, constants and basic mathematical functions.

Todo: Define the section for the `EXACTSOLUTION` along with an expression for u_{ex}. The format is identical to that for specifying the forcing function.

This completes the specification of the Helmholtz problem on the square mesh. It can then be solved using the `ADRSolver` [2]. The executable is located in the folder `solver/builds/dist/bin/`[3]

```
./ADRSolver Test_HEL.xml
```

To view the output in *Gmsh* use the post-processing tools in the `utilities` directory, as we have done for the pre-processing. You can produce outputs for *Gmsh* , *TecPlot* and *Paraview* using the corresponding converter from the terminal. For example, to convert the `Test_HEL.fld` to *Gmsh* format, use

```
./FldToGmsh Test_HEL.xml Test_HEL.fld
```

[2] ADRSolver has been design to solve a range of problems in the form of an Advection-Diffusion-Reaction equation. Switching the EQTYPE flag to another value and providing the appropriate parameters it can solve, for example, Poisson, Laplace, Steady/Unsteady Diffusion, Steady/Unsteady Advection, equations, etc.

[3] If you compiled the library in Debug mode, the executables will have the suffix `-g`.

3 Unsteady Advection-Diffusion equation

Using the same mesh, we will now solve an unsteady diffusion problem. As for the Helmholtz problem, the files are located in the directory

`NekTutorial/Tutorial/SquareMesh/UnsAdvDiffusion/`.

The equation we are solving is

$$\frac{\partial u}{\partial t} + V_X \frac{\partial u}{\partial x} + V_Y \frac{\partial u}{\partial y} = \epsilon \nabla^2 u \tag{4}$$

Setting $\epsilon = 1$ and $V_X = V_Y = 0$ the exact solution is trivial. We can use it to set Dirichlet boundary condition on the edges.

$$u_{ex} = e^{-2\pi^2 t} \sin(\pi x) \cos(\pi y) \tag{5}$$

In this case the executable is still the `ADRSolver` but we need to provide some more information in the input files, including the initial conditions and the time-integration parameters.

- Besides changing the `EQTYPE` to `UnsteadyAdvectionDiffusion`, we add a number of additional properties to specify the approach used in advancing the diffusion and advection components of the problem. Furthermore, we specify the time-stepping scheme to use.

  Todo: Add a solver information section and set the equation type to be `UnsteadyAdvectionDiffusion`. Use a continuous Galerkin projection.

  Todo: Add additional `SOLVERINFO` properties to specify

  - `DiffusionAdvancement` as `Implicit`,
  - `AdvectionAdvancement` as `Explicit`,
  - the `TimeIntegrationMethod` to be `IMEXOrder2`.

- Two parameters must be specified for time integration: `TimeStep` and `NumSteps`. The `IO_CheckSteps` and `IO_InfoSteps` parameters control the frequency of checkpoint files and status information.

  Todo: Use a timestep of 0.00001 and integrate for 2000 steps. Checkpoint the solution every 2000 steps and print information every 200 steps.

- The boundary conditions are now time-dependent. This is specified by an additional attribute `USERDEFINEDTYPE=''TimeDependent''` (double-quotes) after the `VAR` attribute.

  Todo: Specify appropriate boundary regions and conditions for the problem.

- An additional section specifies the advection velocity. In our case, we have set it to zero as we will solve only the diffusion problem.

- Finally, the initial conditions are specified in exactly the same way as the forcing function and exact solution from the Helmholtz problem using a section named `INITIALCONDITIONS`

  Todo: Specify appropriate initial conditions and exact solutions to the problem.

Todo: Run the `ADRSolver` with your session file and ensure the error in the solution is appropriately small.

4 Incompressible Navier-Stokes equations

Using the same mesh, we can solve the Incompressible Navier-Stokes equations. In this case the executable in not the `ADRSolver`, but the `IncNavierStokesSolver`[4], which is still located in the same directory as `ADRSOlver`. All the files, as for the previous examples, are located in `NekTutorial/Tutorial/SquareMesh/IncNavStokes/`.

Considering an incompressible, isothermal flow with constant density and viscosity, the governing equations are the Navier-Stokes (NS) coupled to a velocity divergence-free constraint. In terms of primitive variables (V, p), the variational formulation is written as

$$\frac{\partial V}{\partial t} + V \cdot \nabla V = -\nabla p + \nu \nabla^2 V \tag{6}$$

$$\nabla \cdot V = 0 \tag{7}$$

where $p(x, t)$ is the kinematic pressure field and ν is the kinematic viscosity. The first flow we are going to solve is the Taylor decaying vortex described by the following equations

$$u = -\cos(x)\sin(y)e^{-2t/Re} \tag{8}$$

$$v = \sin(x)\cos(y)e^{-2t/Re} \tag{9}$$

$$p = -\frac{1}{4}\Big(\cos(2x) + \cos(2y)\Big)e^{-4t/Re} \tag{10}$$

In this case we define 3 variables: the 2 velocity components and the pressure. We also include an additional parameter to specify the kinematic viscosity.

Todo: Examine the template file and familiarise yourself with the additional aspects for using the Navier-Stokes solver. Try running it with the `IncNavierStokesSolver`.

4.1 Flow past a cylinder

Here a simulation of the two-dimensional flow past a circular cylinder in a free stream. This is an illustrative example of the use of *Nektar++* framework to solve more complex fluid dynamics problems The solution, which highlights the vortex shedding, has been obtained using the 2^{nd} order stiffly stable splitting scheme with $\Delta t = 0.001s$ and 5^{th} order spectral/hp expansion on a mesh of 1500 quadrilaterals. The cylinder has a diameter $D = 0.4$ and the domain is defined by a rectangle $[-4\,, 16] \times [-5\,, 5]$. Boundary conditions for the velocity filed are of Dirichlet type at the inflow, where a constant velocity in x-direction is imposed ($u = 1$ and $v = 0$) and of Neumann type (homogeneous) at the outflow and on the upper and lower domain limits. The pressure boundary conditions are of Neumann type at the inflow and on the upper and lower domain limits ($\partial p/\partial n = 0$). The pressure value at the outflow has been set to zero (Dirichlet boundary condition).

In this case the solver is still the `IncNavierStokesSolver` and all the files are stored in

`NekTutorial/Tutorial/VortexShedding/`.

In the *.xml* file we can see how curved elements are defined in *Nektar++* . Each curve references an edge and specifies the number of points and the type of point distribution used to prescribe the curve, along with the coordinates.

[4] As for the ADRSolver we can have the Release and the Debug version.

Rather than providing an expression for each variable in the initial condition, we can instead read all the variables from an existing file. Such an initial condition takes the form:

```
<R FILE="[FILENAME]"/>
```

Todo: Update the `VxShed.xml` file to read the initial condition from the file `VxShed.rst`

As a matter of fact a *.rst* file is actually a *.fld* file obtained with *Nektar++* from a previous simulation on the same mesh. In this case we choose to start from a converged solution to speed up the simulation.

5 Extra examples

In the folder `NekTutorial/Tutorial/RegTests/`, you can find plenty of examples which are directly taken from the regression tests.