Nektar++ Tutorial

February 3, 2011

The aim of this tutorial is to lead the user through the *Nektar++* fetaures. The starting point is to compile the whole libraries, the solvers and the utilities as explained on the website[1] . Once we are sure that everything is compiled we can run the regression tests to control that the software is producing the expected results and is working properly.

In this document we will show how to create a 2D simple mesh using *Gmsh* and how to convert the mesh file in a proper input for *Nektar++*. Subsequently we will use this mesh to solve various problem:

1. the Helmholtz equation,
2. the Unsteady Advection-Diffusion equation,
3. and, in the end, the Incompressible Navier-Stokes equation.

This approach should help the user to understand how in *Nektar++* we can share what is the domain (mesh) where we want to solve some equations and the equation itself.
The last step of the tutorial is an example of a more complex fluid dynamics problem, the flow past a cylinder. The point of presenting this typical problem is to introduce some more advanced features like the definition of curved elements and the restart files.

[1]www.nektar.info

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 $[-\pi/2, \pi/2] \times [-\pi/2, \pi/2]$.

If you look into the folder *exercices/NekTutorial/Tutorial/SquareMesh/Geometry/* you can find the following files

- *Square.geo*
- *Square.msh*
- *Square.xml*

Square.geo is the file containing the geometry and the instructions for *Gmsh* to generate the mesh we want. *Square.msh* is the output of *Gmsh*. *Square.xml* is the input file for *Nektar++* without any definitions about the equation we want to solve over the domain.

Square.xml has been generated using the Preprocessing tools in the utilities folder. To generate a *.xml* file starting from a *.msh* file you have just to call the *MeshConvert* executable located in *utilities/builds/PreProcessing/* from the terminal as

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

Figure 1 shows the domain over which we are going to solve our equations.

2 Helmholtz equation

We first want to solve the 2D Helmholtz equation on the domain we have just created; the equation is:

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

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

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

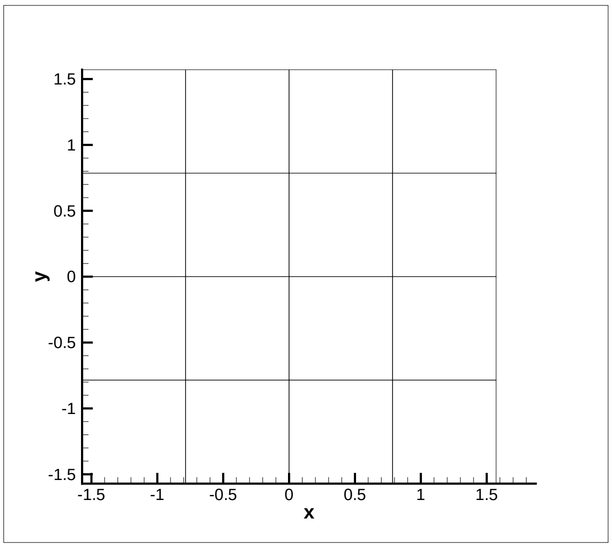

Figure 1: 16 quadrilaterals mesh

Into the folder *exercices/NekTutorial/Tutorial/SquareMesh/Helmholtz/* you can find the following files

- *Test_HEL.xml*, which is the *.xml* already prepared to use the geometry and solve the problem;
- *Test_HEL.fld*, which is the solution obtained with *Nektar++*;
- *Test_HEL_u.pos*, which is the file you can load in *Gmesh* to see the solution.

If we take a look of the *Test_HEL.xml*, we can see how we can define:

1. the expansion type and order,

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

2. the equation type, the projection type (Continous corresponds to the continuos Galerkin) and the parameters we need,

```
SOLVERINFO
 PROPERTY="EQTYPE" VALUE="Helmholtz"
 PROPERTY="Projection" VALUE="Continuous"
SOLVERINFO
PARAMETERS
 wavefreq = PI
 Lambda = 1.0
PARAMETERS
```

3. the variable/s declaration and the boundary regions linking to the defined geometry,

```
VARIABLES
 ID="0" u
VARIABLES
BOUNDARYREGIONS
 ID="0" C[1]
BOUNDARYREGIONS
```

4. the definition of the boundary condition value (in this case we have used the exact solution as a Dirichlet boundary condition), the definition of the forcing term and the exact solution,

```
BOUNDARYCONDITIONS
 REGION REF="0"
 D VAR="u" VALUE="sin(PI*x)*sin(PI*y)"
 REGION
BOUNDARYCONDITIONS
FORCING
 F VAR="u" VALUE="-(Lambda + 2*PI*PI)*sin(PI*x)*sin(PI*y)"
FORCING
EXACTSOLUTION
 F VAR="u" VALUE="sin(PI*x)*sin(PI*y)"
EXACTSOLUTION
```

The Helmholtz equation is then solved with the executable *ADRSolver*[2]. The executable is located in the folder *solver/builds/dist/bin/*[3]

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

To produce the output for *Gmesh* you need to use the *PostProcessing* tools in the folder *utilities*, as we have done for the *PreProcessing*. You can produce outputs for *Gmesh*, *TecPlot* and *Paraview* using the related converter from the terminal. For example for *Gmsh* is

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

This executable will produce *Test_HEL_u.pos* that can be loaded in *Gmsh*.

3 Unsteady Advection-Diffusion equation

Using the same mesh, we are going to solve now an Unsteady-Advection-Diffusion problem. As before for the Helmholtz problem, the files are located in *exercices/NekTutorial/Tutorial/SquareMesh/UnsAdvDiffusion/*. Here you can find the same kind of files which has been presented for the Helmholtz problem.

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 and we can use it to set Dirichlet boundary condition on the edges.

[2]ADRSolver has been design to solve all the problems derived from a typical Advection-Diffusion-Reaction equation. Switching the EQTYPE flag to another value and providing the proper information you can solve for example Poisson, Laplace, Steady/Unsteady Diffusion, Steady/Unsteady Advection, etc.

[3]Depending on the compilation mode you can find: ADRSolver, if you have compiled the solvers in Release mode, or ADRSolver-g, if you have compiled the solvers in Debug mode.

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

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

```
SOLVERINFO
 PROPERTY=”EQTYPE” VALUE=”UnsteadyAdvectionDiffusion”
 PROPERTY=”Projection” VALUE=”Continuous”
 PROPERTY=”DiffusionAdvancement” VALUE=”Implicit”
 PROPERTY=”AdvectionAdvancement” VALUE=”Explicit”
 PROPERTY=”TimeIntegrationMethod” VALUE=”IMEXOrder2”
SOLVERINFO
```

```
PARAMETERS
 TimeStep = 0.00001
 NumSteps = 2000
 IO_CheckSteps = 2000
 IO_InfoSteps = 2000
 wavefreq = PI
 epsilon = 1.0
PARAMETERS
```

```
USERDEFINEDEQNS
 F LHS=”Vx” VALUE=”0.0”
 F LHS=”Vy” VALUE=”0.0”
USERDEFINEDEQNS
```

```
INITIALCONDITIONS
 F VAR=”u” VALUE=”sin(wavefreq*x)*cos(wavefreq*y)”
INITIALCONDITIONS
```

4 Incompressible Navier-Stokes equation

Again, using the same mesh, we can solve the Incompressible Navier-Stokes equation. In this case the executable in not the *ADRSolver* anymore, but

the $IncNavierStokesSolver$, which is located in the same folder of the previous one.[4] All the files, as for the previous examples, are located in *exercices/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 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 variable, the 2 velocity components and the pressure.

```
SOLVERINFO
 I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes"
 I PROPERTY="AdvectionForm" VALUE="Convective"
 I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder1"
SOLVERINFO
```

```
PARAMETERS
 TimeStep = 0.00001
 NumSteps = 1000
 IO_CheckSteps = 1000
 IO_InfoSteps = 1000
 Kinvis = 0.0001
PARAMETERS
```

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

```
VARIABLES
 V ID="0" u
 V ID="1" v
 V ID="2" p
VARIABLES
```

5 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 *exercices/NekTutorial/Tutorial/VortexShedding/*. In the $.xml$ file we can see how curved elements are defined in *Nektar++* and how we can use a previous solution to initialise the flow. As a matter of fact a $.rst$ file is actually a $.fld$ file obtained with *Nektar++* and used to set the flow. In this case we want to start from a converged solution to speed up the simulation.

6 Extra examples

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