Primer to hms

I. Özgen, Chair of Water Resources Management and Modeling of Hydrosystems

ilhan.oezgen@wahyd.tu-berlin.de

The Hydroinformatics Modeling System (hms) is a Java-based modeling framework. The primary aim of hms is easy extendibility with regard to physical processes. hms is has a Layer-based architecture, which means that physical processes as well as data are represented as Layer Objects. For more detail, the reader is referred to the doctoral dissertation of Busse (Busse, 2015).

A first example: Dam-break flow on dry bed

The dam-break on dry bed is the „Hello, World!“ example of shallow water flow modeling. If this case fails we don't need to model anything further.

The domain is 10 m long and the dam-break is described as a discontinuity in initial water depths, such that

$$h(x)=\begin{cases}1m,\ x<5m,\\ 0,\ x\geq 5m.\end{cases} \qquad (1)$$

The first thing to do is to create a class which represents the model. We will call it *DamBreak*. The class should implement the interface *Startable*, which tells hms that this is a simulation.

```
public class DamBreak implements Startable {

   @Override
   public void start(String... args) {


   }

}
```

The interface *Startable* requires us to write the *start*-method. We will neglect this method for a while and come back to it later. The first thing our model needs is a domain. In hms, we decided that *Domain*-Objects will be private classes inside the *Startable*-Object.

Thus, we write inside the class we just created a second class called *SimDomain* which will extend

HDomain.

```
  private class SimDomain extends HDomain {


   @Override
   public double[] getInitialValue(int layer, int variable,
         Element element, TimeStamp time) {

      return null;
   }


   @Override
   protected void addBoundarySegments() {


   }

  }
```

This gives us two additional methods to implement. The method *getInitialValue* is used to set the initial value in a cell and the method *addBoundarySegments* is used to set the boundary conditions. The initial conditions are set as following:

```
      @Override
      public double[] getInitialValue(int layer,
            int variable, Element element, TimeStamp time) {

         final double[] z = new double[1];
         final double[] e = new double[1];
         final double[] h = new double[1];
         final double[] v = new double[2];

         final Point point = element.getSpecificPoint();

         if (point.getX() <= 5.0) {
            h[0] = 1.0;
         } else {
            h[0] = 0.0;
         }

         e[0] = h[0] + z[0];

         if (layer == SurfaceFlow.LAYER_INDEX) {

            switch (variable) {

            case SurfaceFlow.INDEX_BOTTOM_ELEVATION:
               return z;
            case SurfaceFlow.INDEX_WATER_DEPTH:
               return h;
            case SurfaceFlow.INDEX_WATER_ELEVATION:
               return e;
            case SurfaceFlow.INDEX_FLOW_VELOCITY:
               return v;
```

```
        }
    }

    throw new RuntimeException("There is not such variable " + variable
          + " in layer " + layer + "!");

}
```

In hms, all variables are stored as vectors. If the variable is a scalar quantity, such as the water depth, then the vector has the dimension 1. The method returns the value for *variable* for the *Layer* with index *layer*. Here, we have set the initial water depth to 1 m in the whole domain. The bed elevation is at 0 m +NN and the velocity is 0 m/s in both x- and y-direction. We will now set the boundary conditions:

```
@Override
protected void addBoundarySegments() {

   HBoundarySegment segment;

   segment = new HBoundarySegment(SurfaceFlow.LAYER_INDEX,
         new HSurfaceFlowNoFlowValues());
   segment.addVertex(new TPoint(0.0, 0.0));
   segment.addVertex(new TPoint(10.0, 0.0));

   addBoundarySegment(segment);

   segment = new HBoundarySegment(SurfaceFlow.LAYER_INDEX,
         new HSurfaceFlowFreeOutflowValues());
   segment.addVertex(new TPoint(10.0, 0.0));
   segment.addVertex(new TPoint(10.0, 1.0));

   addBoundarySegment(segment);

   segment = new HBoundarySegment(SurfaceFlow.LAYER_INDEX,
         new HSurfaceFlowNoFlowValues());
   segment.addVertex(new TPoint(10.0, 1.0));
   segment.addVertex(new TPoint(0.0, 1.0));

   addBoundarySegment(segment);

   segment = new HBoundarySegment(SurfaceFlow.LAYER_INDEX,
         new HSurfaceFlowNoFlowValues());

   segment.addVertex(new TPoint(0.0, 1.0));
   segment.addVertex(new TPoint(0.0, 0.0));

   addBoundarySegment(segment);


}
```

Here, we create the Object *HBoundarySegment*, that holds information about the boundary condition and also the boundary geometry and we drew the boundary of a channel that is 10 m long and 2 m wide. Note that the path we draw is counter-clockwise. This is a convention of hms we must not violate, otherwise the model will not compute. All boundary conditions are closed boundaries which in hms are called *HSurfaceFlowNoFlowValues*. Downstream, we imposed a free outflow boundary condition that mirrors the water depth from the inside cell (*HSurfaceFlowFreeOutflowLimitedValues*).

We have completed the implementation of this class, now we will return to the *start*-method and create an instant of it in the *start*-method of the model.

```
@Override
public void start(String... args) {

   final SimDomain domain = new SimDomain();
   final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);


   final HViewer vie = new HViewer(domain.getBoundary());
   vie.add(mesh);

}
```

We have created an instance of the *NarrowingChannel* and have meshed it with a cell size of 0.1 m using the *ShapesFactory*-Object. This creates the *Mesh*. We can look at the mesh by creating an instance of the *HViewer* and add the mesh to the *HViewer* to draw it.

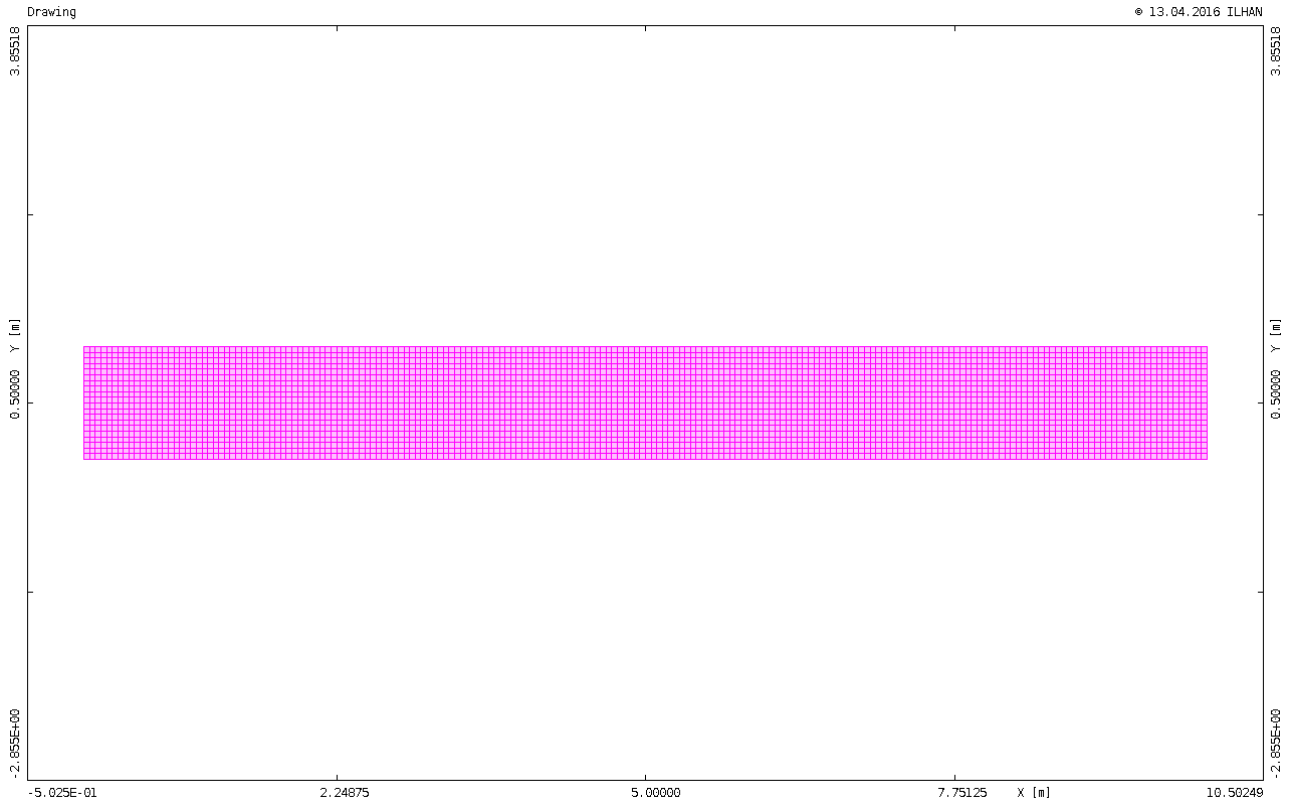
Write the main method as follows and run the model:

```
public static void main(final String[] args) {


   final DamBreak app = new DamBreak();
   app.start(args);
}
```

Running the file should give the console output:

```
HRegularGridGenerator: generate grid [20x200]

HRegularGridGenerator: finished - 4000 cells, 8220 edges
```

and open the viewer. If you click the *Shp*-button in the viewer, the mesh will be shown.



Good job! Now, we have the geometry. We need to add the physics and the numerics to the model before we can run it. Firstly, we create instances of DefaultSurfaceFlowSettings, that define the physical process parameters of surface flow and the DefaultNumericalSettings, that define the numerical parameters. We created these classes as default settings, based on our own experience. Feel free to use the setter and getter methods to set different parameters.

```
   @Override
   public void start(String... args) {

      final SimDomain domain = new SimDomain();
      final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);

      final DefaultSurfaceFlowSettings surfaceFlowSettings = new
DefaultSurfaceFlowSettings();
      final DefaultNumericsSettings numericsSettings = new
DefaultNumericsSettings();

      final HViewer vie = new HViewer(domain.getBoundary());
      vie.add(mesh);

   }
```

In fact, let's set some parameters together. For example, we would like to have second order accuracy in space:

```
numericsSettings.setSecondOrderInSpace(isSecondOrder);
```

We don't want to wait too long for the simulation to finish, let's use all available processors to run the simulation in parallel:

```
numericsSettings.setMaxThreadCount(0);
```

Note that 0 means all available processors. And we would like to set the Courant number to 0.3:

```
surfaceFlowSettings.setCflCriteria(0.3);
```

We would like to limit the slope using the min-mod limiter function:

```
surfaceFlowSettings.setLimiterFunction(LimiterFunction.MIN_MOD);
```

So now the *start*-method looks like this:

```
   @Override
   public void start(String... args) {

      final SimDomain domain = new SimDomain();
      final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);

      final DefaultSurfaceFlowSettings surfaceFlowSettings = new
DefaultSurfaceFlowSettings();
      final DefaultNumericsSettings numericsSettings = new
DefaultNumericsSettings();

      numericsSettings.setSecondOrderInSpace(true);
      numericsSettings.setMaxThreadCount(0);
      surfaceFlowSettings.setCflCriteria(0.3);
      surfaceFlowSettings.setLimiterFunction(LimiterFunction.MIN_MOD);

      final HViewer vie = new HViewer(domain.getBoundary());
      vie.add(mesh);

   }
```

We said before that hms is Layer-based. We will now create a layer that represents the surface flow process:

```
      final Layer layer = LayerFactory.getSurfaceFlowLayer(domain, mesh,
            surfaceFlowSettings, numericsSettings);
```

Layers are managed by the *LayerManager*-class. We therefore create a *HLayerManager* and add the

created *layer* to it:

```
      final HLayerManager manager = new HLayerManager();
      manager.addLayer(layer);
```

Now, the *start*-method looks like this:

```
   @Override
   public void start(String... args) {

      final SimDomain domain = new SimDomain();
      final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);

      final DefaultSurfaceFlowSettings surfaceFlowSettings = new
DefaultSurfaceFlowSettings();
      final DefaultNumericsSettings numericsSettings = new
DefaultNumericsSettings();

      numericsSettings.setSecondOrderInSpace(true);
      numericsSettings.setMaxThreadCount(0);
      surfaceFlowSettings.setCflCriteria(0.3);
      surfaceFlowSettings.setLimiterFunction(LimiterFunction.MIN_MOD);

      final Layer layer = LayerFactory.getSurfaceFlowLayer(domain, mesh,
            surfaceFlowSettings, numericsSettings);

      final HLayerManager manager = new HLayerManager();
      manager.addLayer(layer);

      final HViewer vie = new HViewer(domain.getBoundary());
      vie.add(mesh);

   }
```

OK, the model is almost ready. For now, we will just add these lines to the code:

```
      manager.setSystemTimeStep(1.0e-3);
      vie.add(manager);
```

Here, we told the model that the first step it takes should be 1.0e-3 s large and we added the manager to the viewer. This way, we can watch the model as it simulates the dam-break. Now, remove the line

```
      vie.add(mesh);
```

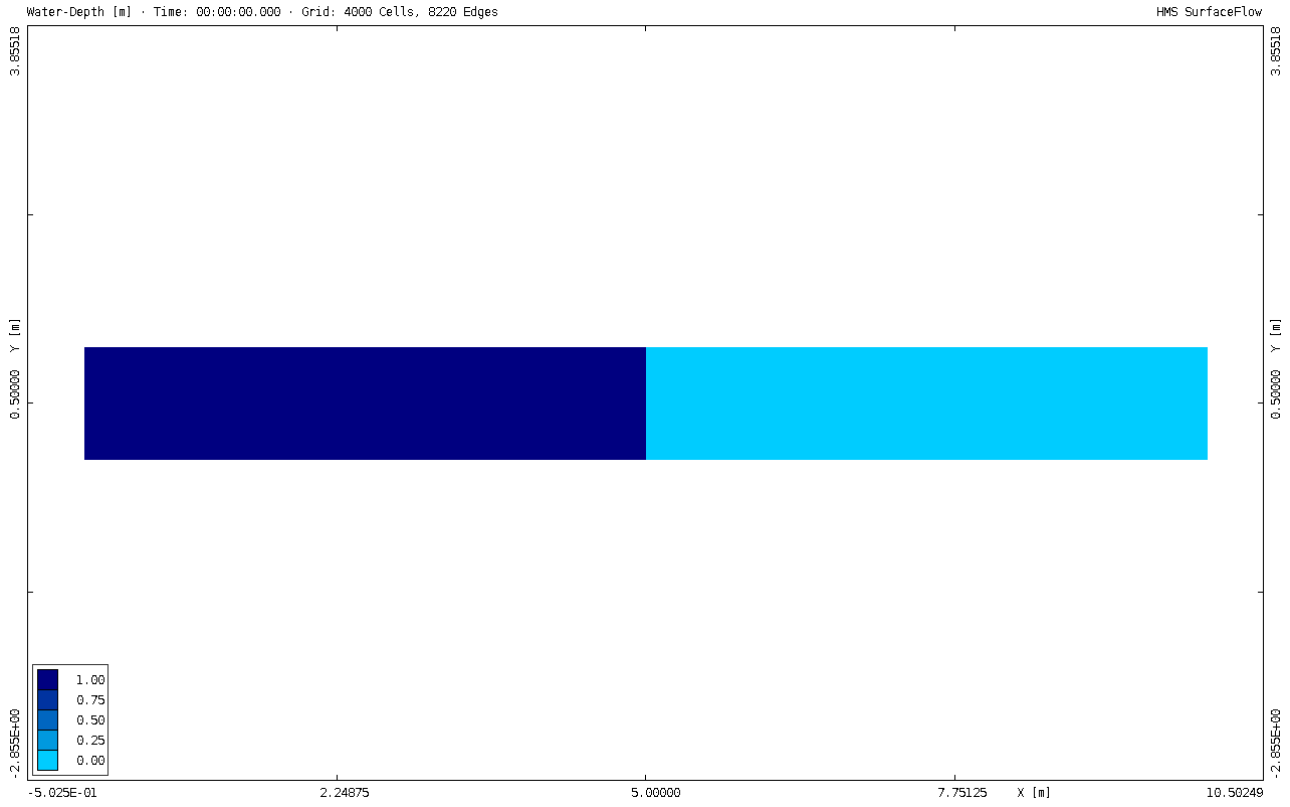
from the *start*-method and we are ready!

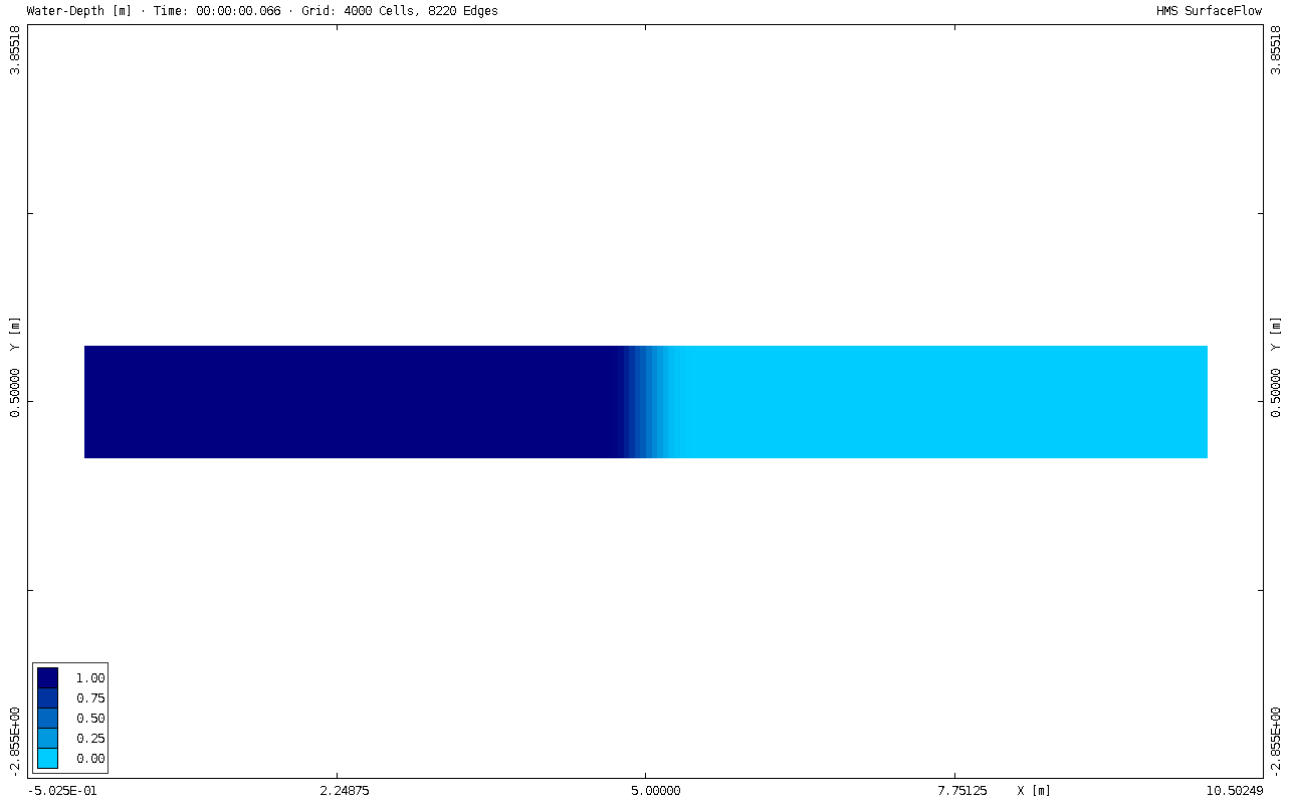
```
   @Override
   public void start(String... args) {
```

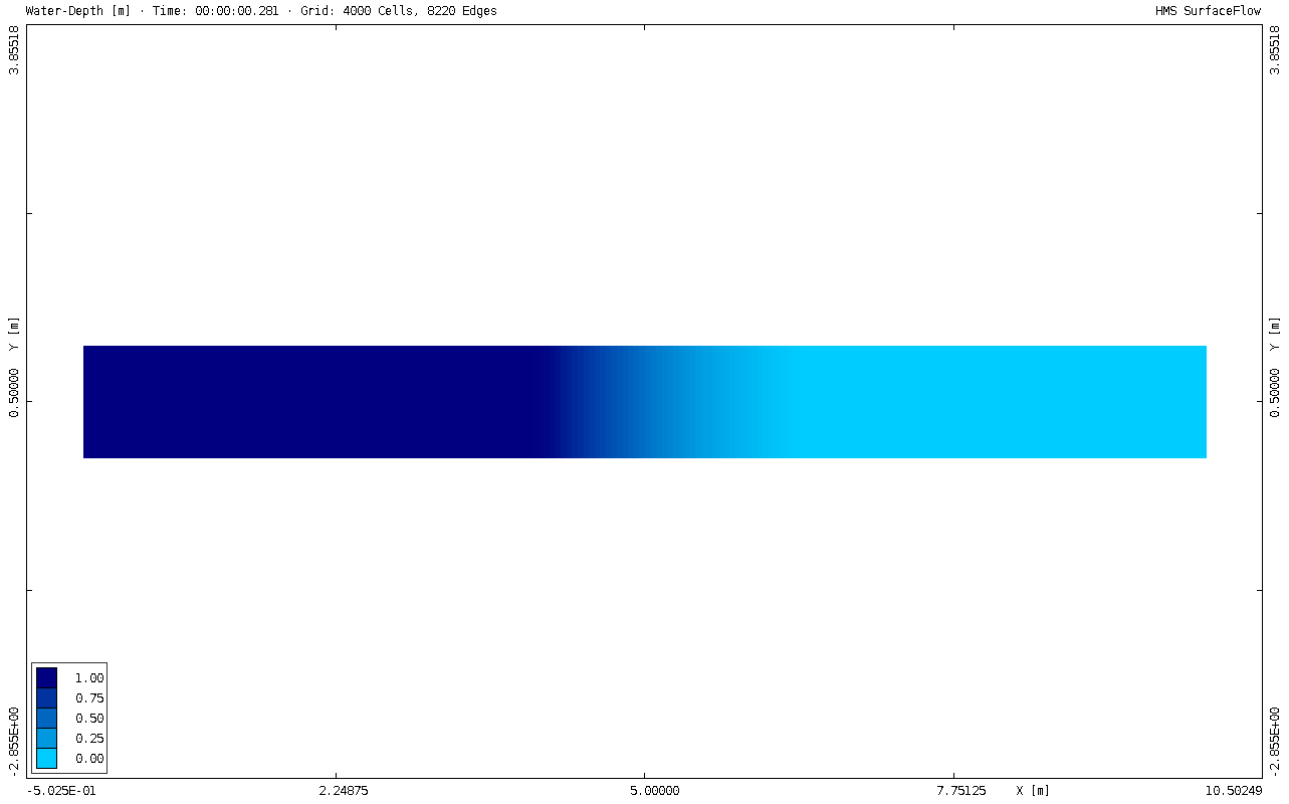
```
      final SimDomain domain = new SimDomain();
      final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);

      final DefaultSurfaceFlowSettings surfaceFlowSettings = new
DefaultSurfaceFlowSettings();
      final DefaultNumericsSettings numericsSettings = new
DefaultNumericsSettings();

      numericsSettings.setSecondOrderInSpace(isSecondOrder);
      numericsSettings.setMaxThreadCount(0);
      surfaceFlowSettings.setCflCriteria(0.3);
      surfaceFlowSettings.setLimiterFunction(LimiterFunction.MIN_MOD);

      final Layer layer = LayerFactory.getSurfaceFlowLayer(domain, mesh,
            surfaceFlowSettings, numericsSettings);

      final HLayerManager manager = new HLayerManager();
      manager.addLayer(layer);

      final HViewer vie = new HViewer(domain.getBoundary());

      manager.setSystemTimeStep(1.0e-3);
      vie.add(manager);

   }
```

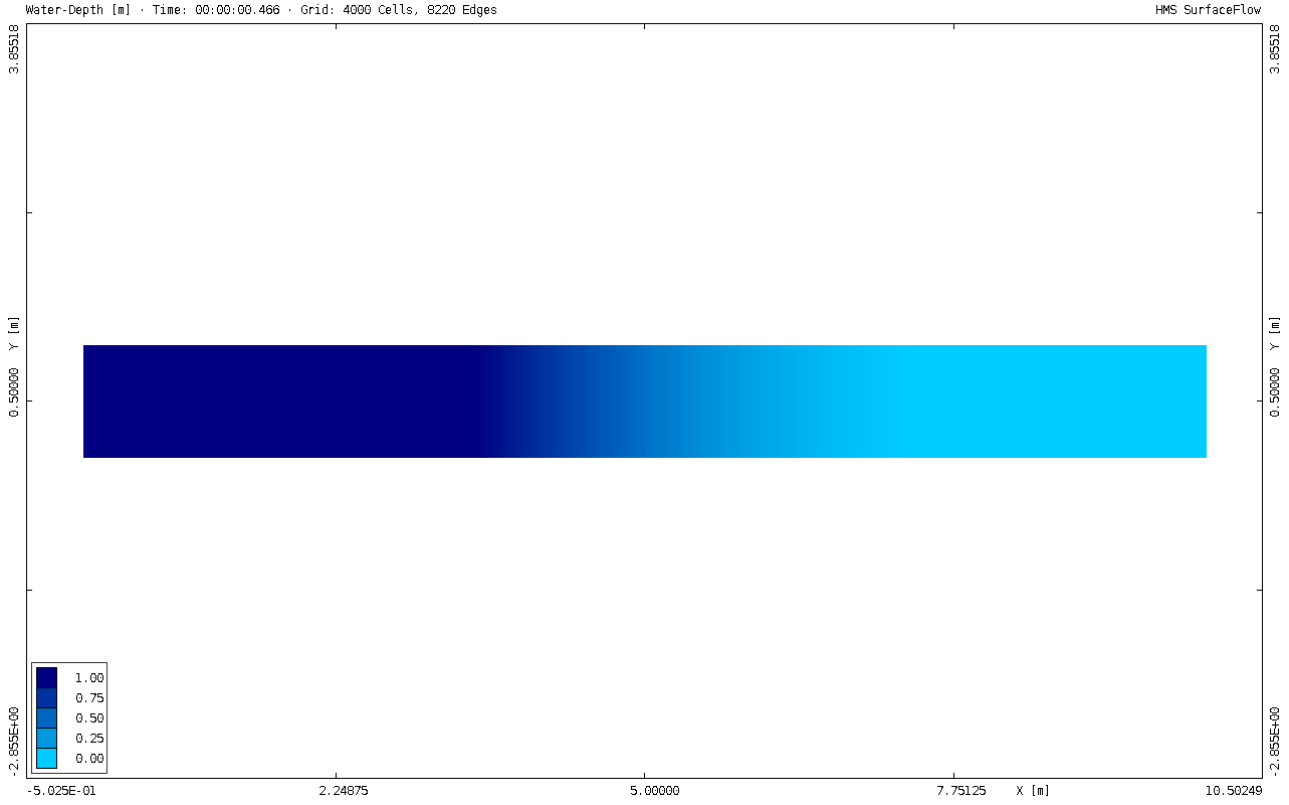
Click on the *Run*-button in the viewer to start the simulation. You should see something like this:

(Click on the *Var*-button and select the variable you would like to watch and click the *Rng*-button to rescale your legend.)


Water-Depth [m] · Time: 00:00:00.000 · Grid: 4000 Cells, 8220 Edges
HMS SurfaceFlow
3.85518
0.50000 Y [m]
-2.855E+00
1.00
0.75
0.50
0.25
0.00
-5.025E-01
2.24875
5.00000
7.75125
X [m]
10.50249


Water-Depth [m] · Time: 00:00:00.066 · Grid: 4000 Cells, 8220 Edges
HMS SurfaceFlow
3.85518
0.50000 Y [m]
-2.855E+00
1.00
0.75
0.50
0.25
0.00
-5.025E-01
2.24875
5.00000
7.75125
X [m]
10.50249


Water-Depth [m] · Time: 00:00:00.281 · Grid: 4000 Cells, 8220 Edges
HMS SurfaceFlow
3.85518
0.50000
Y [m]
-2.855E+00
1.00
0.75
0.50
0.25
0.00
-5.025E-01
2.24875
5.00000
7.75125
X [m]
10.50249


Water-Depth [m] · Time: 00:00:00.466 · Grid: 4000 Cells, 8220 Edges
HMS SurfaceFlow
3.85518
0.50000
Y [m]
-2.855E+00
1.00
0.75
0.50
0.25
0.00
-5.025E-01
2.24875
5.00000
7.75125
X [m]
10.50249

You can now watch the shock wave advance to the right while a rarefaction-wave fan spreads to the left.

You may notice that the simulation now runs forever. If we want to set a time horizon to the simulation we need to use *Listener*-Objects. *Listener*-Objects listen to the simulation and when they are triggered through an event, e.g. the time horizon is overshot, they perform an action. One such *Listener* is the WStopSimulationMonitor. Add these lines to the start-method:

```
final WStopSimulationMonitor stop = new WstopSimulationMonitor(0.5);
manager.addListener(stop);
```

to stop the simulation after 0.5 s.

```
   @Override
   public void start(String... args) {

      final SimDomain domain = new SimDomain();
      final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);

      final DefaultSurfaceFlowSettings surfaceFlowSettings = new
DefaultSurfaceFlowSettings();
      final DefaultNumericsSettings numericsSettings = new
DefaultNumericsSettings();

      numericsSettings.setSecondOrderInSpace(isSecondOrder);
      numericsSettings.setMaxThreadCount(0);
      surfaceFlowSettings.setCflCriteria(0.3);
      surfaceFlowSettings.setLimiterFunction(LimiterFunction.MIN_MOD);

      final Layer layer = LayerFactory.getSurfaceFlowLayer(domain, mesh,
            surfaceFlowSettings, numericsSettings);

      final HLayerManager manager = new HLayerManager();
      manager.addLayer(layer);

      final WStopSimulationMonitor stop = new WstopSimulationMonitor(0.5);
      manager.addListener(stop);

      final HViewer vie = new HViewer(domain.getBoundary());

      manager.setSystemTimeStep(1.0e-3);
      vie.add(manager);

   }
```

Now the simulation stops at 0.5 s. We most likely would like to write out results to compare with reference solution. For this particular case, an analytical solution for the water depth exists:

$$h(x,t)=\frac{1}{9g}\left(2\sqrt{g h_0}-\frac{x}{t}\right)^2 \quad (2)$$

g is of course gravity and h_0 is the initial water depth at the left side. How can we compare the model results with this analytical solution? We will add another *Listener*, this time the *WPlotOverLineMonitor*, that writes out a solution over a drawn section in CSV format:

```
final WPlotOverLineMonitor plot = new WPlotOverLineMonitor(
               new WConstantInterval(0.5), SurfaceFlow.LAYER_INDEX, mesh,
               new TPolyline(new TPoint(0.0, 0.55),
                     new TPoint(10.0, 0.55)));
manager.addListener(plot);
```

The easiest way to write out the file would be to extend the WStopSimulationMonitor:

```
         final WStopSimulationMonitor stop = new WStopSimulationMonitor(
               runtime) {

            @Override
            public void postprocess(final Event event) {
               WToolbox.writeGnuPlotFile(new File(
                     HMS.HMS_WORKSPACE + "/RESULT.csv"),
                     plot.getData(SurfaceFlow.META_DATA,
                           SurfaceFlow.INDEX_WATER_DEPTH),
                     new String[] { "x", "water depth" },
                     " ");
            }

         };
```

Thus, the start-method should now look like this:

```
   @Override
   public void start(String... args) {

      final SimDomain domain = new SimDomain();
      final Mesh mesh = ShapesFactory.getRegularGrid(domain, 0.05);

      final DefaultSurfaceFlowSettings surfaceFlowSettings = new
DefaultSurfaceFlowSettings();
      final DefaultNumericsSettings numericsSettings = new
DefaultNumericsSettings();

      numericsSettings.setSecondOrderInSpace(isSecondOrder);
      numericsSettings.setMaxThreadCount(0);
      surfaceFlowSettings.setCflCriteria(0.3);
      surfaceFlowSettings.setLimiterFunction(LimiterFunction.MIN_MOD);

      final Layer layer = LayerFactory.getSurfaceFlowLayer(domain, mesh,
            surfaceFlowSettings, numericsSettings);

      final HLayerManager manager = new HLayerManager();
      manager.addLayer(layer);
```

```
final WPlotOverLineMonitor plot = new WPlotOverLineMonitor(
               new WConstantInterval(0.5), SurfaceFlow.LAYER_INDEX, mesh,
               new TPolyline(new TPoint(0.0, 0.55),
                     new TPoint(10.0, 0.55)));
manager.addListener(plot);


final WStopSimulationMonitor stop = new WStopSimulationMonitor(
      0.5) {

      @Override
      public void postprocess(final Event event) {
      WToolbox.writeGnuPlotFile(new File(
      HMS.HMS_WORKSPACE + "/RESULT.csv"),
      plot.getData(SurfaceFlow.META_DATA,
      SurfaceFlow.INDEX_WATER_DEPTH),
      new String[] { "x", "water depth" },
      " ");
      }
   };

      manager.addListener(stop);

      final HViewer vie = new HViewer(domain.getBoundary());

      manager.setSystemTimeStep(1.0e-3);
      vie.add(manager);

   }
```

We used some fancy tools already implemented. This will create a GnuPlot file in the

HMS_WORKSPACE directory, which is defined as:

```
public static final String HMS_WORKSPACE = (Files.USER_HOME + "/HMS/Workspace");
```

Navigate there and open the file with a text editor. Here is what you will see:

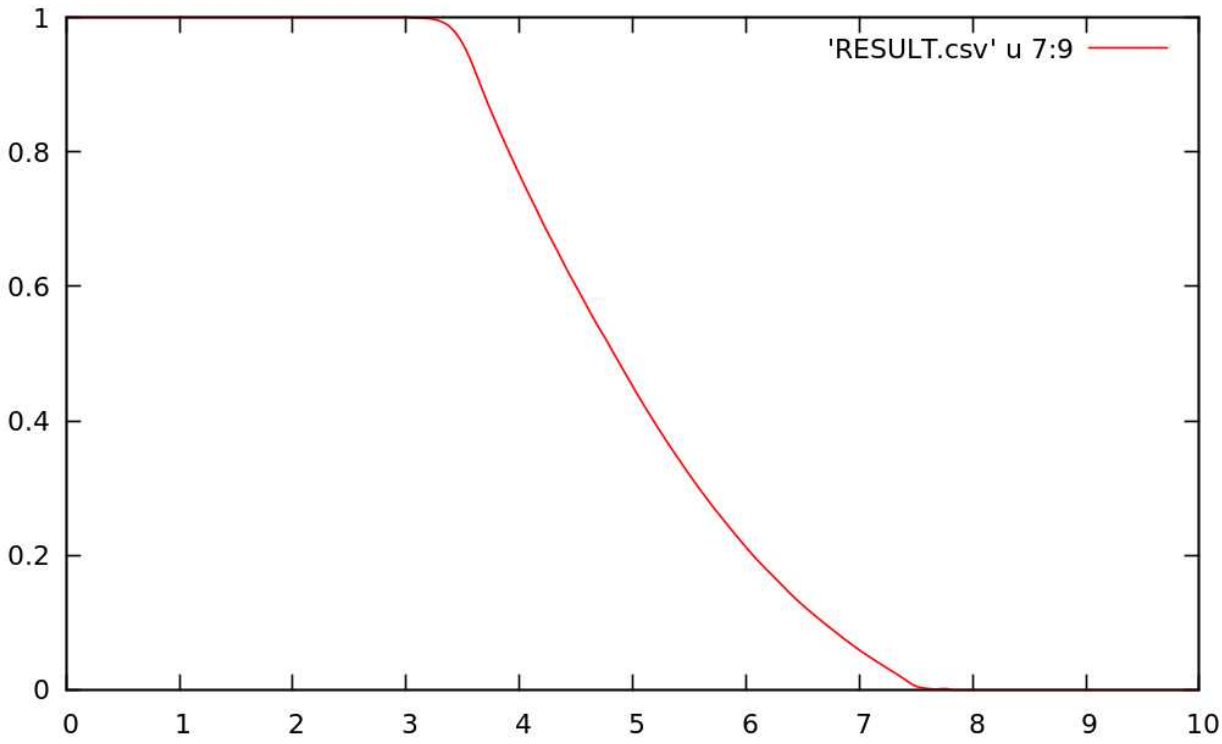
```
#

#x #water depth

0.0 0.05 0.0 0.55 0.05 0.55 0.025 0.525 1.0

0.05000000000000001 0.1 0.05000000000000001 0.55 0.1 0.55 0.07500000000000001 0.525 1.0

0.1 0.15 0.1 0.55 0.15 0.55 0.125 0.525 1.0

0.15000000000000002 0.2 0.15000000000000002 0.55 0.2 0.55 0.17500000000000002 0.525 1.0

0.2 0.25 0.2 0.55 0.25 0.55 0.225 0.525 1.0

0.25 0.30000000000000004 0.25 0.55 0.30000000000000004 0.55 0.275 0.525 1.0

0.30 0.35 0.30000000000000004 0.55 0.3500000000000001 0.55 0.32500000000000007 0.525 1.0

0.35 0.40 0.35000000000000003 0.55 0.4000000000000001 0.55 0.37500000000000006 0.525 1.0
```

This looks strange, the thing to know is that in hms, the columns mean the following:

```
/**

 * Returns the data for the given cross section. Each line contains the
 * following data:<br>
 * l1 local coordinate 1 <br>
 * l2 local coordinate 2 <br>
 * x1 global x coordinate for point 1 <br>
 * y1 global y coordinate for point 1 <br>
 * x2 global x coordinate for point 2 <br>
 * y2 global y coordinate for point 2 <br>
 * x global x coordinate of the cell <br>
 * y global y coordinate of the cell <br>
 * phi the cell data, length depends on dimension of variable
 *
 * @return data for given cross section
 */
public double[][] getData(final CalcLayerMetaData meta, final int variable)
```

Start gnuplot (open terminal and type 'gnuplot') and write

```
> plot 'RESULT.csv' u 7:9 with lines
```



The plot of the analytical solution in gnuplot is trivial and left for the reader as an exercise.