Developing an advection scheme for Telemac-2D based on the 2nd order characteristics method

Jacek A. Jankowski, Andreas Malcherek

Bundesanstalt für Wasserbau

Karlsruhe, Hamburg

Telemac Users Club, November 2001

Contents

- Introduction: the method of characteristics
- Algorithm: 2nd order interpolation
- Software: modules
- The Hunte River
- Molenkamp test
- Hydraulic jump
- Conclusions, further work

The aim of the work

...is to deliver an advection scheme for Telemac-2D, which is:

1. unconditionally stable concerning the Courant number $C = u\Delta t / \Delta x$
2. computationally less intensive than other comparable schemes
3. numerically less diffusive than schemes presently available in Telemac-2D

Operator splitting

Telemac algorithm is based on the *operator splitting:*

- differential operators splitted due to their mathematical properties,
- treated in separate, consecutive algorithm steps
- ...by applying the most appropriate numerical methods

Side effect: a large variety of numerical options to choose from...

Advection step

The first partial solution to be added up:

$$\frac{f^{adv} - f^{n}}{\Delta t} + \mathbf{u} \cdot \nabla f = 0$$

- describes translation in a given velocity field
- 5-6 methods to treat advection in Telemac library
- intuitively the simplest is the method of characteristics:
  the time variability of a variable f observed **along a streamline**

The method of characteristics

- Advection equation says: the variable f does not change along a streamline (characteristic curve):

$$\frac{\partial f}{\partial t} + \mathbf{u} \cdot \nabla f = \frac{df}{dt} = 0$$

- The idea: *in order to find out the nodal value, follow the characteristic curve backward in time*

- Streamline equation:

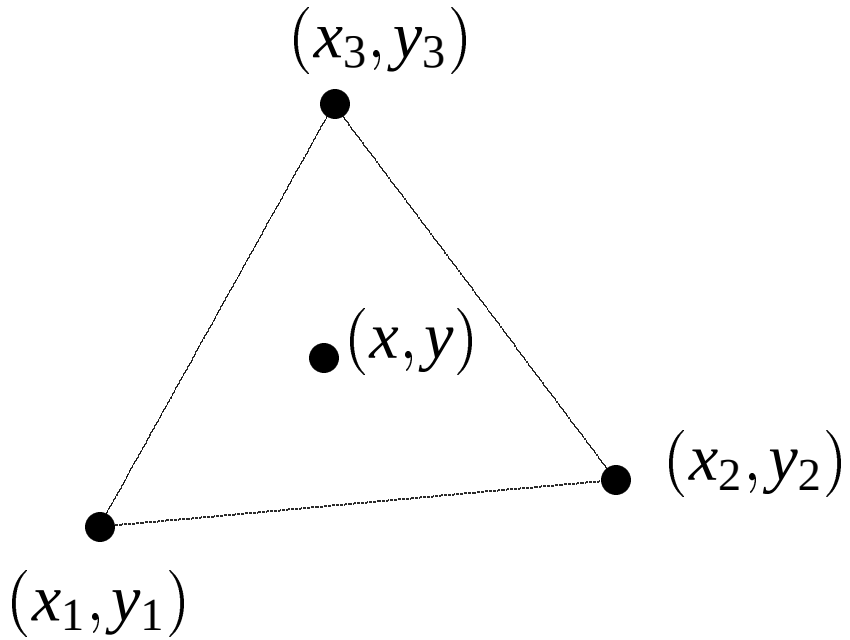
$$\frac{dx_j}{dt} = u_j \qquad \Rightarrow \qquad x_j^b = x_i^{n+1} - u_j \Delta t$$

The method of characteristics - algorithm

Algorithm steps, for each node i:

- find the characteristics base point x_i^b
- interpolate the value f_i^b from the nodal values surrounding the base point $f_{j(i)}^n$ (at the time level n)
- for time step $n+1$ set simply $f_i^{n+1} = f_i^b$
- deal with exceptional cases at the domain boundaries

First order (linear) interpolation



$$f(x,y) = \sum_{i=1}^{3} f_i \varphi_i(x,y)$$

$$\varphi_i(x_j,y_j) = \begin{cases} 0 & \text{for} \quad j \neq i \\ 1 & \text{for} \quad j = i \end{cases}$$

$$\varphi_i(x,y) = a_{10}^i x + a_{01}^i y + a_{00}^i$$

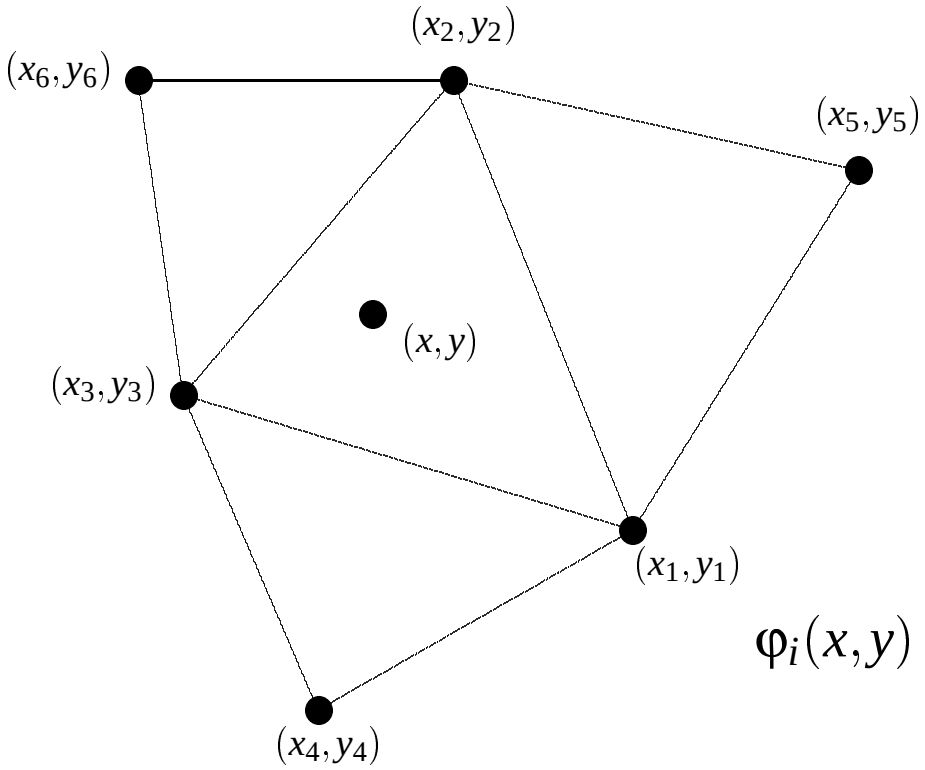
Note: Linear interpolation is applied in Telemac-2D

Why higher interpolation order?

As shown by Andreas Malcherek during Telemac Users Club 2000:

- Linear interpolation: quite large numerical diffusion
- To diminish the numerical diffusion: a higher interpolation order required
- Theory: already the 2nd order yields a dramatic improvement
- Practice: applied in the FDM-code TRIM (structured mesh), 2nd order brings extraordinary improvement of results for tidal forecasts in estuaries

Second order interpolation



$$f(x,y) = \sum_{i=1}^{6} f_i \varphi_i(x,y)$$

$$\varphi_i(x_j,y_j) = \begin{cases} 0 & \text{for} \quad j \neq i \\ 1 & \text{for} \quad j = i \end{cases}$$

$$\varphi_i(x,y) = a^i_{20}x^2 + a^i_{11}xy + a^i_{02}y^2 + a^i_{10}x + a^i_{01}y + a^i_{00}$$

Analytic solution

The usual method to obtain interpolation functions in finite elements is:

- define them in a reference element: $\psi_i(\xi,\eta)$ (sexangle!)
- taking into account that $x^j = \sum_{i=1}^{N} x_i^j \psi_i(\xi,\eta)$
- transform reference interp. functions into a real element $\psi_i(\xi,\eta) \quad \rightarrow \quad \varphi_i(x,y)$

Analytic formula for an interpolation function

For complex elements mathematical software is used:

```
   FortranForm(List(1 - (y1**2*
-       ((-((-((-((-x1 + x3)*(-(x1*y1) + x2*y2)) +
-            (-x1 + x2)*(-(x1*y1) + x3*y3))*
-           (-((-x1 + x4)*(-y1 + y2)) + (-x1 + x2)*(-y1 + y4)))\
-         + (-(x2*y1) + x3*y1 + x1*y2 - x3*y2 - x1*y3 + x2*y3)*
-         (-((-x1 + x4)*(-(x1*y1) + x2*y2)) +
-               (-x1 + x2)*(-(x1*y1) + x4*y4)))*
-           ((-((-x1**2 + x2**2)*(-x1 + x5)) +
-             (-x1 + x2)*(-x1**2 + x5**2))*
-           (-(x2*y1) + x3*y1 + x1*y2 - x3*y2 - x1*y3 + x2*y3) -
-         (-((-x1**2 + x2**2)*(-x1 + x3)) +
-            (-x1 + x2)*(-x1**2 + x3**2))*
-       (-((-x1 + x5)*(-y1 + y2)) + (-x1 + x2)*(-y1 + y5))))\
-          + ((-((-x1**2 + x2**2)*(-x1 + x4)) +
-          (-x1 + x2)*(-x1**2 + x4**2))*
-         (-(x2*y1) + x3*y1 + x1*y2 - x3*y2 - x1*y3 + x2*y3) -
-         (-((-x1**2 + x2**2)*(-x1 + x3)) +
-         (-x1 + x2)*(-x1**2 + x3**2))*
-        (-((-x1 + x4)*(-y1 + y2)) + (-x1 + x2)*(-y1 + y4)))*
-          (-((-((-x1 + x3)*(-(x1*y1) + x2*y2)) +
-          (-x1 + x2)*(-(x1*y1) + x3*y3))*
-       (-((-x1 + x5)*(-y1 + y2)) + (-x1 + x2)*(-y1 + y5))) +
-           (-(x2*y1) + x3*y1 + x1*y2 - x3*y2 - x1*y3 + x2*y3)*
```

...and so on (9245 output lines, 0.5 MB file).

Lagrange condition

The Lagrange condition is applied to obtain a^i_{xy}.

For each patch node i is:

$$
\begin{aligned}
a^i_{20}x_1^2 + a^i_{11}x_1y_1 + a^i_{02}y_1^2 + a^i_{10}x_1 + a^i_{01}y_1 + a^i_{00} &= \delta_{i1} \\
a^i_{20}x_2^2 + a^i_{11}x_2y_2 + a^i_{02}y_2^2 + a^i_{10}x_2 + a^i_{01}y_2 + a^i_{00} &= \delta_{i2} \\
a^i_{20}x_3^2 + a^i_{11}x_3y_3 + a^i_{02}y_3^2 + a^i_{10}x_3 + a^i_{01}y_3 + a^i_{00} &= \delta_{i3} \\
a^i_{20}x_4^2 + a^i_{11}x_4y_4 + a^i_{02}y_4^2 + a^i_{10}x_4 + a^i_{01}y_4 + a^i_{00} &= \delta_{i4} \\
a^i_{20}x_5^2 + a^i_{11}x_5y_5 + a^i_{02}y_5^2 + a^i_{10}x_5 + a^i_{01}y_5 + a^i_{00} &= \delta_{i5} \\
a^i_{20}x_6^2 + a^i_{11}x_6y_6 + a^i_{02}y_6^2 + a^i_{10}x_6 + a^i_{01}y_6 + a^i_{00} &= \delta_{i6}
\end{aligned}
$$

Equation sets for interpolation coefficients

...which yields six $(i = 1..6)$ equations set to be solved once a run:

$$\begin{pmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \\ x_6^2 & x_6y_6 & y_6^2 & x_6 & y_6 & 1 \end{pmatrix} \begin{pmatrix} a_{20}^i \\ a_{11}^i \\ a_{02}^i \\ a_{10}^i \\ a_{01}^i \\ a_{00}^i \end{pmatrix} = \begin{pmatrix} \delta_{i1} \\ \delta_{i2} \\ \delta_{i3} \\ \delta_{i4} \\ \delta_{i5} \\ \delta_{i6} \end{pmatrix}$$

Checking if the results are right

...checking in each patch if the interpolation functions

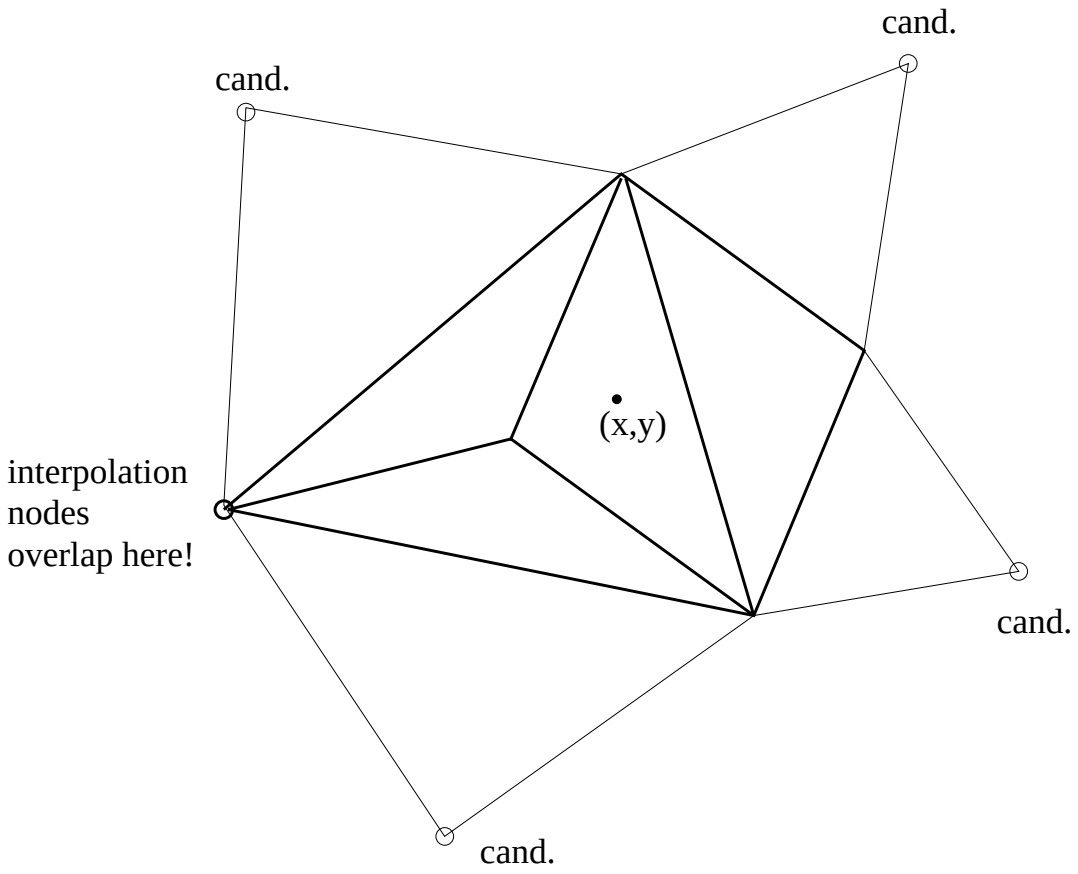
$$\varphi_i(x,y) = a^i_{20}x^2 + a^i_{11}xy + a^i_{02}y^2 + a^i_{10}x + a^i_{01}y + a^i_{00}$$

...fulfill the normal condition

$$\sum_{i=1}^{N} \varphi_i(x,y,z) = 1$$

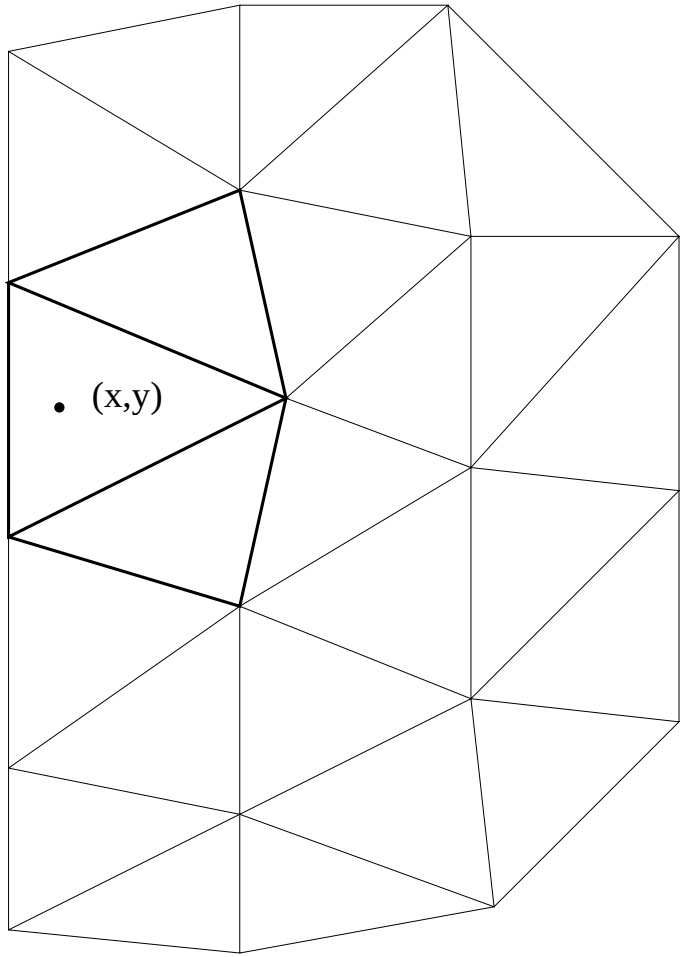
...leads to a ”discovery” of *degenerated patches*:

A degenerated patch



...the missing sixth interpolation node is replaced by the nearest one from candidates in a *broader patch*

Treatment of boundary elements



- in boundary elements linear interpolation is applied
- characteristic curves leaving/entering the domain treated as previously
- tidal flats: element status must be checked each time step

Monotony

The solution cannot be greater or smaller than the max. or min. nodal values in the given patch:

$$f_i^- = \min(f_{j(i)}^n) \leq f_i^{n+1} \leq \max(f_{j(i)}^n) = f_i^+$$

- While for linear interpolation the monotony is guaranteed, it is not the case for higher orders.
- Presently preferred: overshoots $f_i^{n+1} > f_i^+$ and undershoots $f_i^{n+1} < f_i^-$ replaced by f_i^+ or f_i^- respectively.

Code-independent module

```
module SECONDORDERINTERPOLATION
  ...
  double precision, save, pointer :: weights(:,:,:)
  integer, save, pointer :: patchnodes(:,:)
  double precision, private :: a(6,6)
  ...
contains
  subroutine INITIALIZE_SEC_ORD_INT
  ...
  subroutine SECONDORDER_INTERPOLATE
  ...
  subroutine FINALIZE_2NDORDER_INTERPOLATION
  ...
  subroutine LUSOLVER(a,b)
  ...
end module SECONDORDERINTERPOLATION
```

- Contains global methods
- Written independently from the particular code

Code-specific module

```
module CHAMET
 use bief
 ...
 private :: pcharac, pcaract, pinterp
 private :: pchar11, pgshp11, pgtsh11, pgtshp1
 private :: challoc, prunonce, sqinterp, varint

 interface mcharac
   module procedure pcharac
 end interface

 interface runonce
   module procedure prunonce
 end interface

 type(bief_obj), private, save :: pshp, pelt
 type(bief_obj), private, save :: pshpe, iklee
 type(bief_obj), private, save :: pshz, peta

contains
  subroutine CHALLOC
    use SECONDORDERINTERPOLATION
    ...
  end subroutine CHALLOC
  ...
  subroutine PCHARAC
    use BIEF
    ...
  end subroutine PCHARAC
  ...
  subroutine PCHAR11
  ...
end module CHARAC
```

- Contains methods specific for Telemac-2D
- Only two subroutines with interfaces outside

Calling in Telemac-2D

```
subroutine TELEMAC2D
  use BIEF
  use DECLARATIONS_TELEMAC
  use DECLARATIONS_TELEMAC2D
  use CHAMET
  ...

  if (chamod) call RUNONCE (...)
  ...
! in time loop
  ...
  if (chamod) then
    call MCHARAC(... , ichord)
  else
    call CHARAC(...)
  endif
  ...
end subroutine TELEMAC2D
```

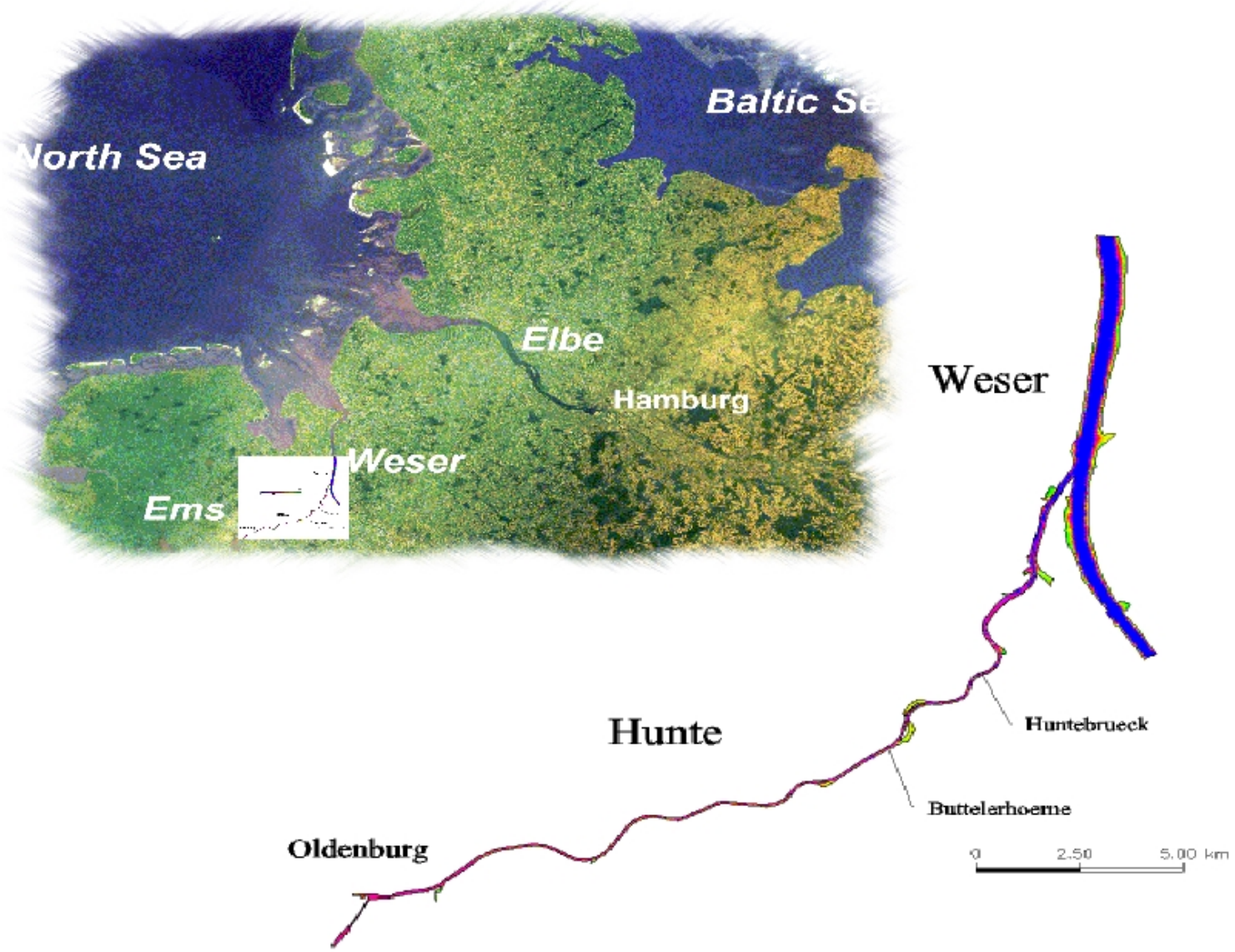
- Telemac uses the specific module only
- By defined interfaces programmers can work on their modules thoroughly independently
- Modifications only when interfaces change
- Code-independent modules reusable for other codes

Run time for the hydraulic jump test case

case	not optimised	optimised
no module, 1st order	55s	55s
module, 1st order	68s	54s
module, 2nd order	115s	61s

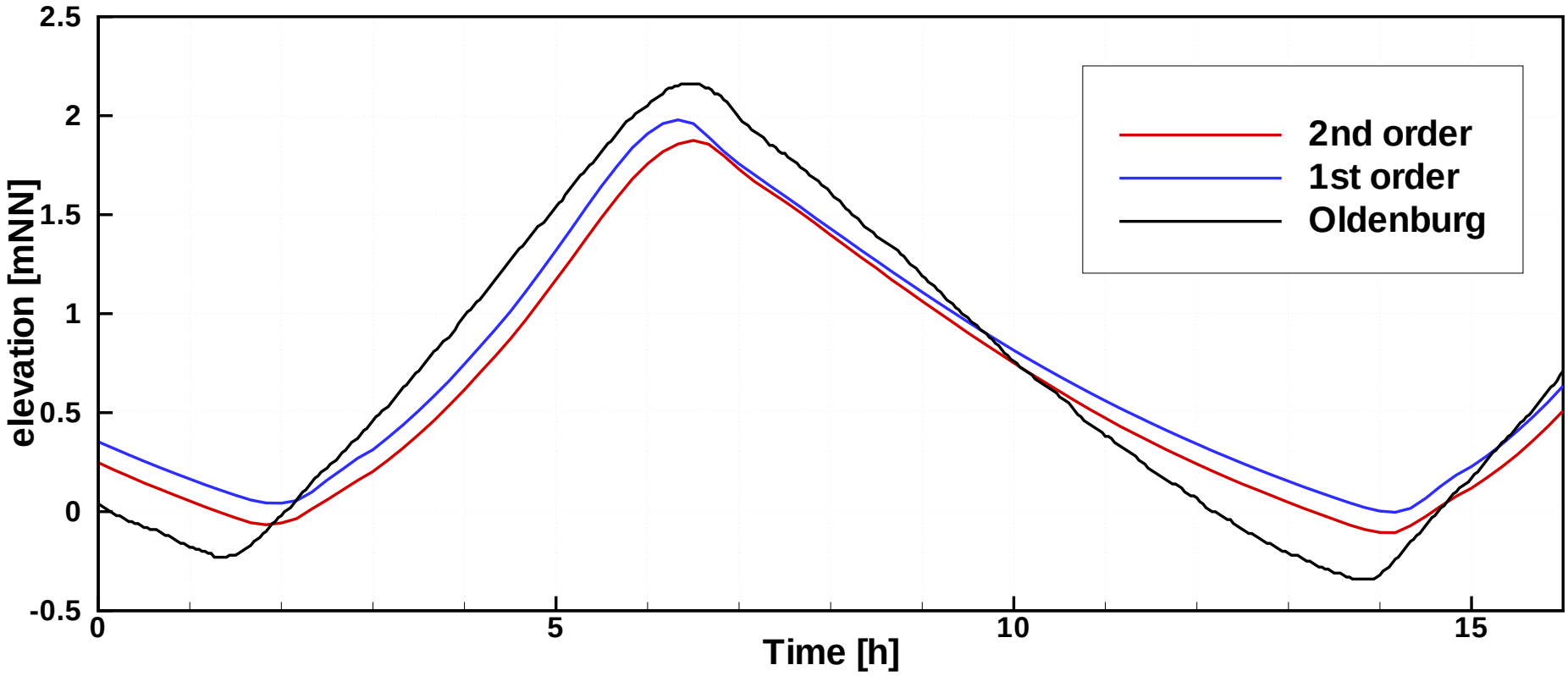
Note: **2nd order**: 1st order functions are computed as well

The Hunte River

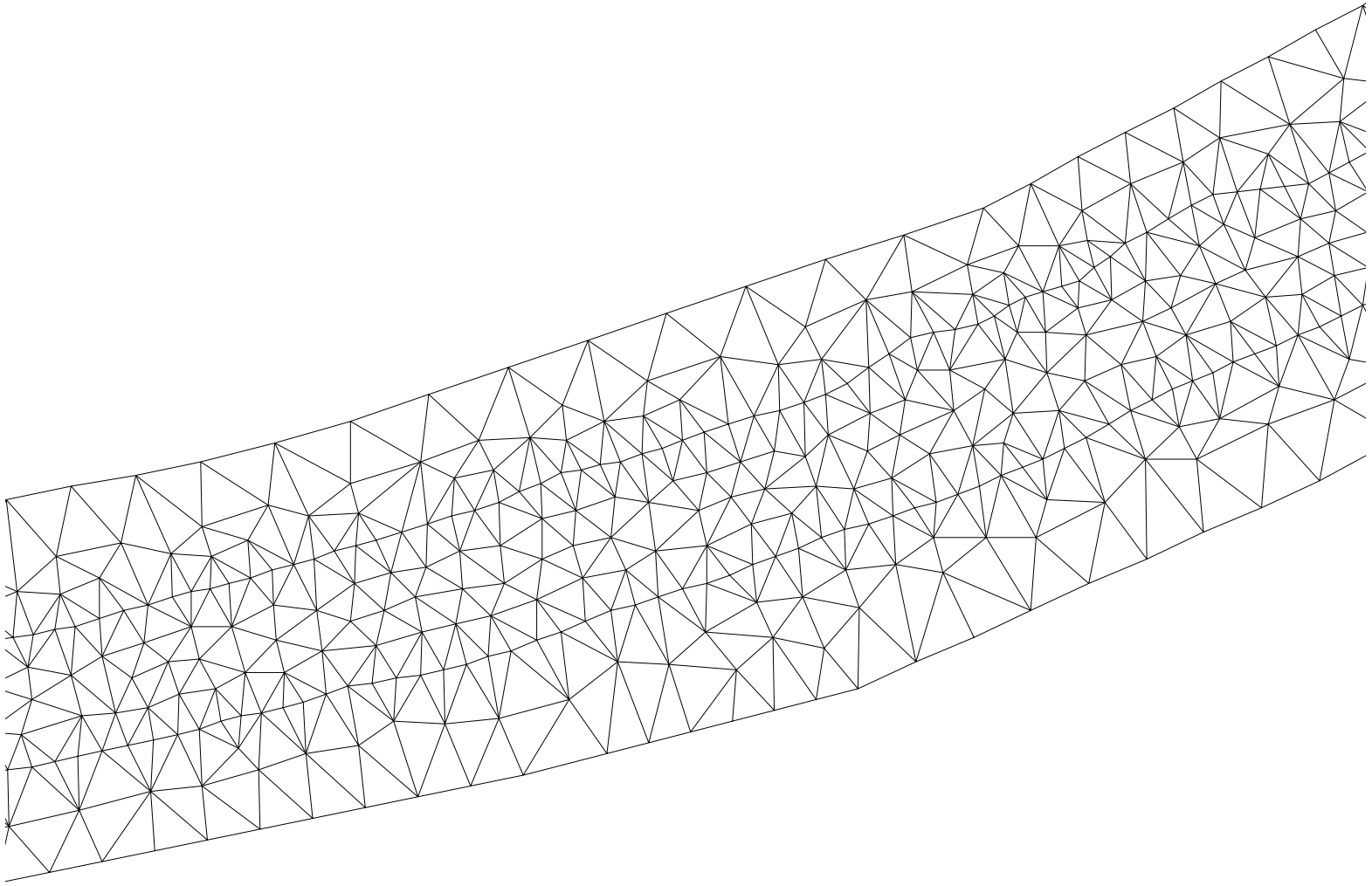


Tidal gauge in Oldenburg

Modelled tidal signal in Oldenburg harbour compared with the tide gauge



The Hunte River mesh



Molenkamp test (rotating cone)

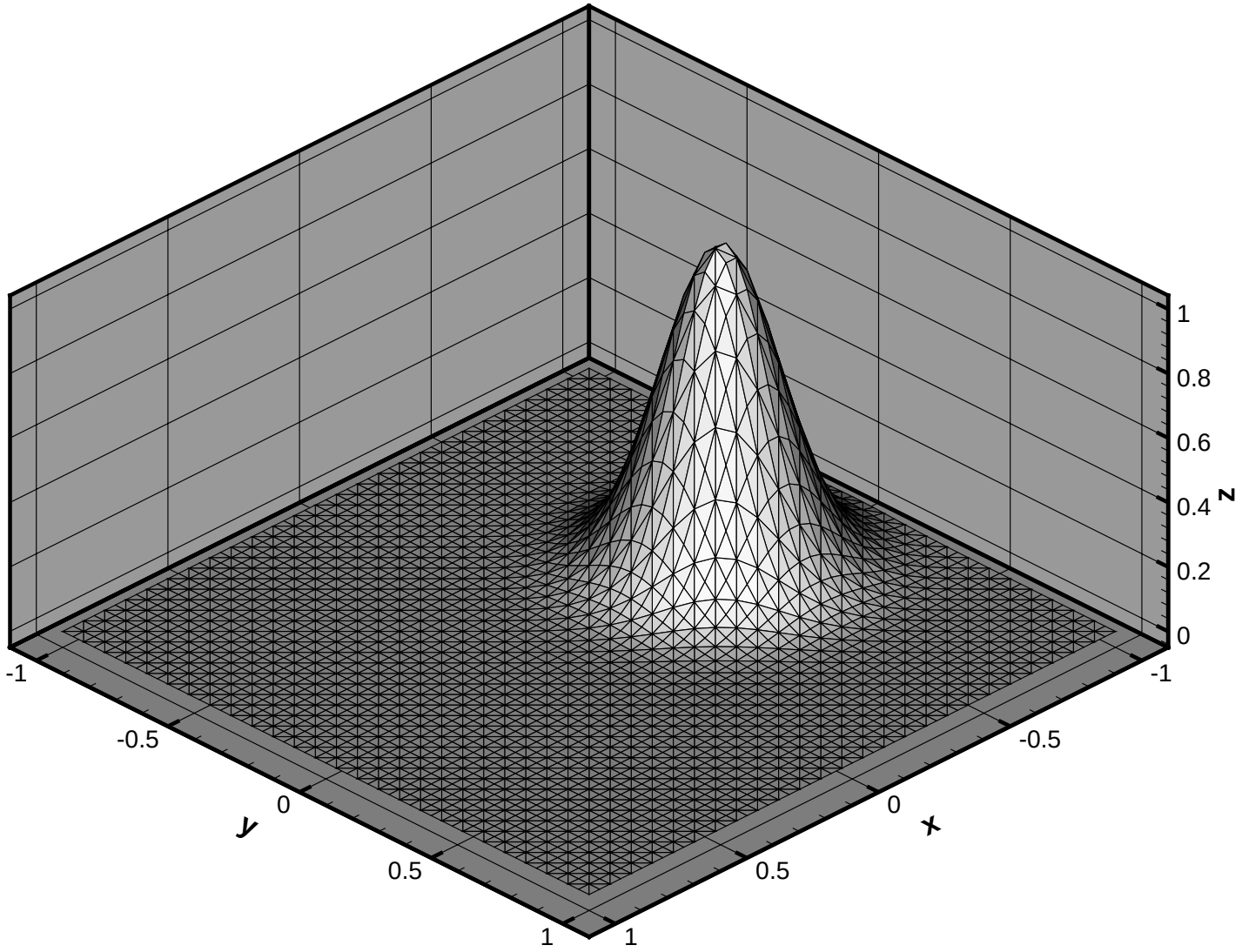
An asymmetrically placed Gaussian profile in a rotating velocity field.

In theory: no change of shape awaited during and after one rotation.

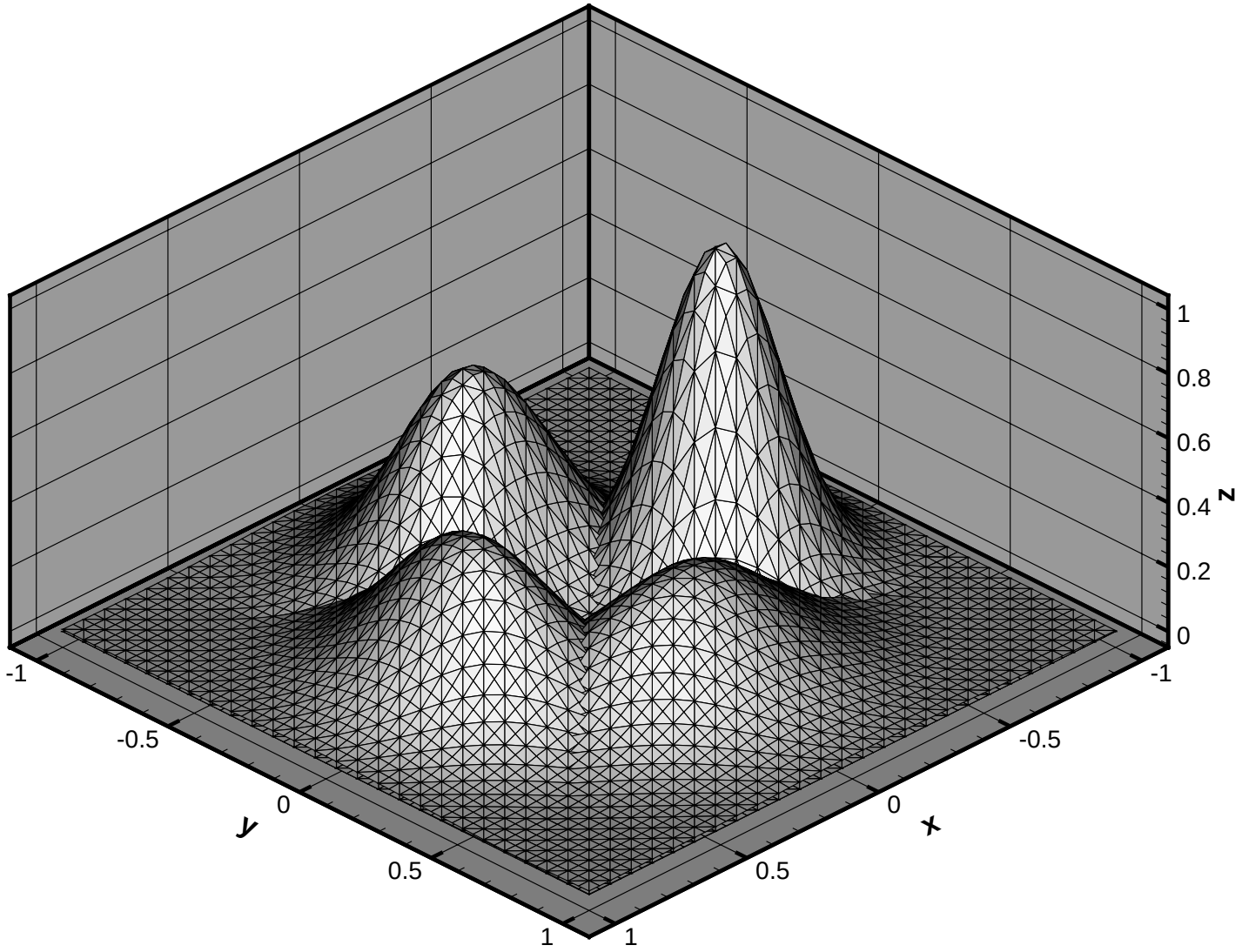
Difficulties:

- Courant number varies along the profile
- advection direction changes relative to the mesh
- the profile slope is relatively large
- BCs and the point with $(u, v) = (0, 0)$ interfere with the solution

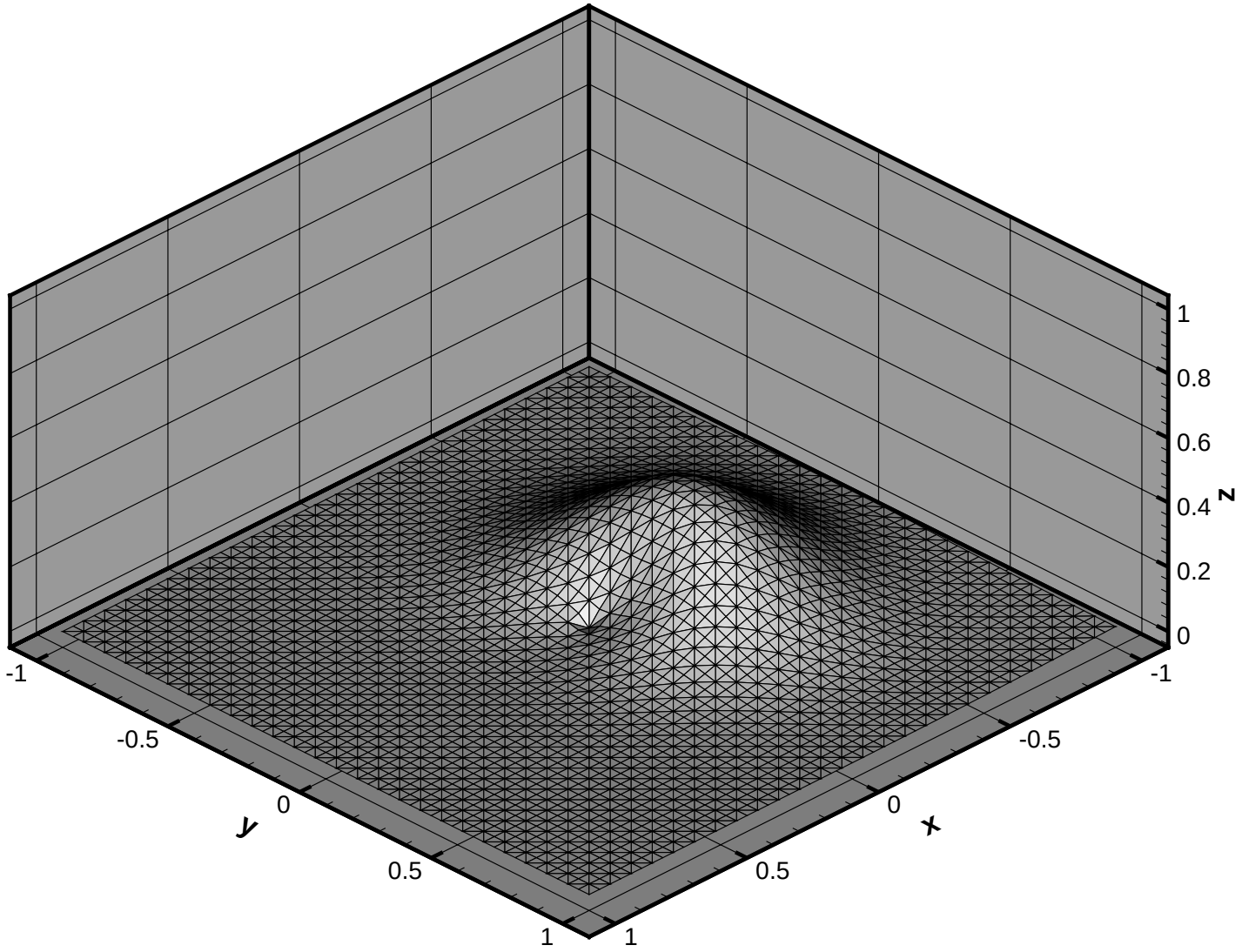
Rotating cone - start profile



Rotating cone - travelling...



Rotating cone - after one turn



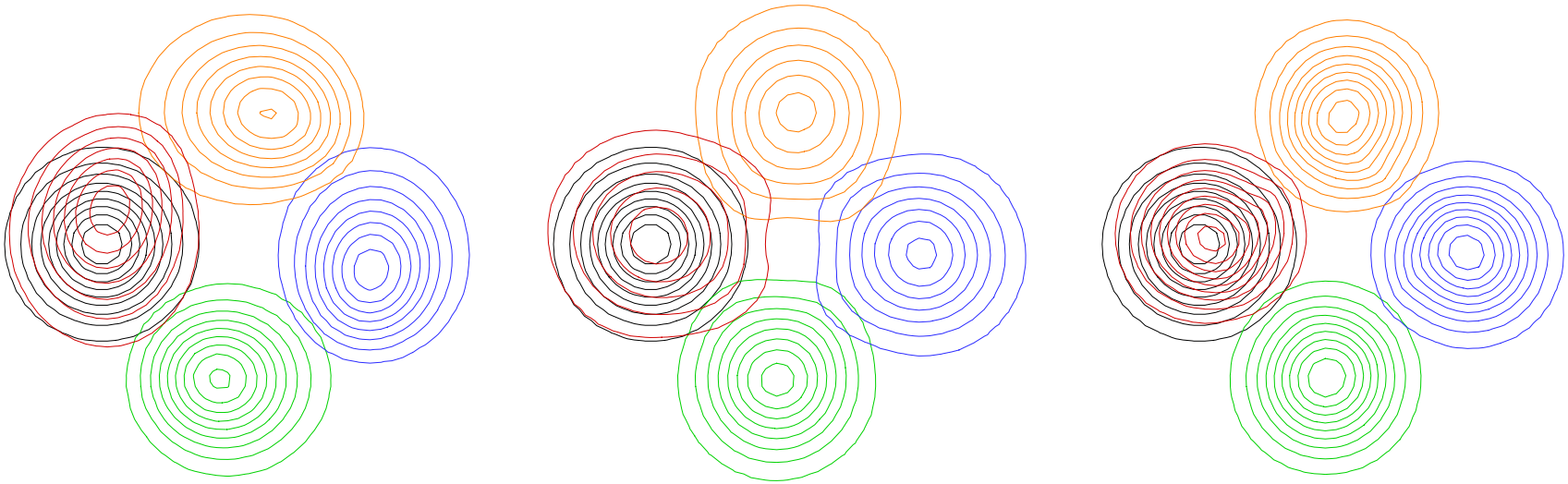
Peak value variability after one turn

Courant Number at peak	SUPG modified semi-impl.	1st ord (linear)	2nd ord no check	2nd ord patch min/max	2nd ord elem min/max	2nd ord patch linear	2nd ord elem linear
0.1	0.901	0.307	0.893	0.846	0.834	0.729	0.682
0.2	0.883	0.320	0.892	0.853	0.841	0.738	0.717
0.5	0.831	0.365	0.896	0.869	0.862	0.763	0.761
0.8	0.788	0.473	0.907	0.883	0.875	0.811	0.804
1.0	**0.762**	**0.570**	**0.951**	**0.931**	**0.927**	**0.876**	**0.867**
1.3	0.726	0.642	0.980	0.961	0.951	0.906	0.899
1.6	0.699*	0.654	0.973	0.962	0.950	0.900	0.882
2.0	0.670*	0.708-	0.979-	0.965	0.961-	0.909	0.905
5.0	*	0.862-	0.984-	0.979-	0.976-	0.952-	0.943-

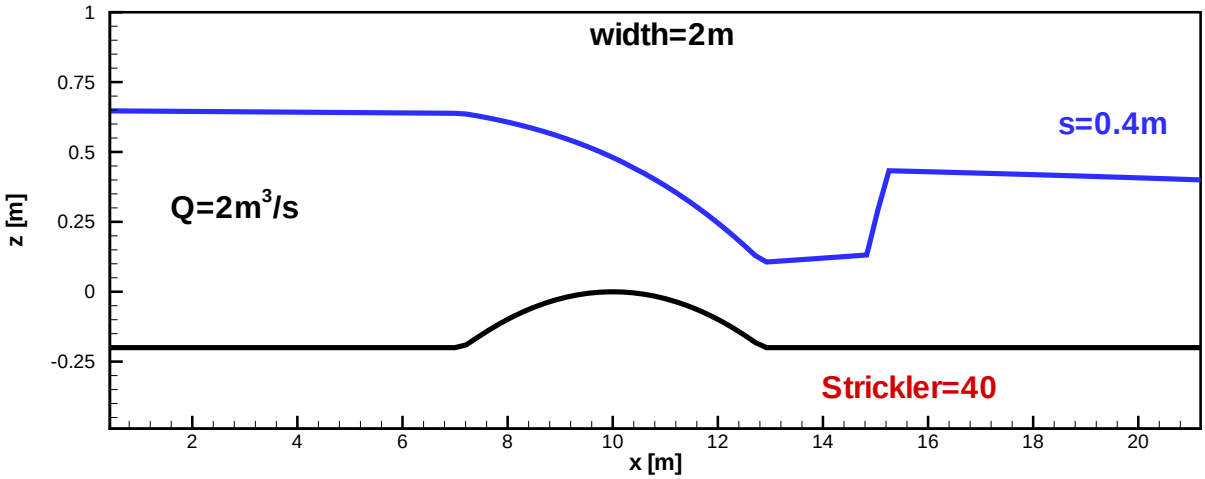
Peak value at start is **0.993**, $\varepsilon = 10^{-6}$ for checking over/undershoots.

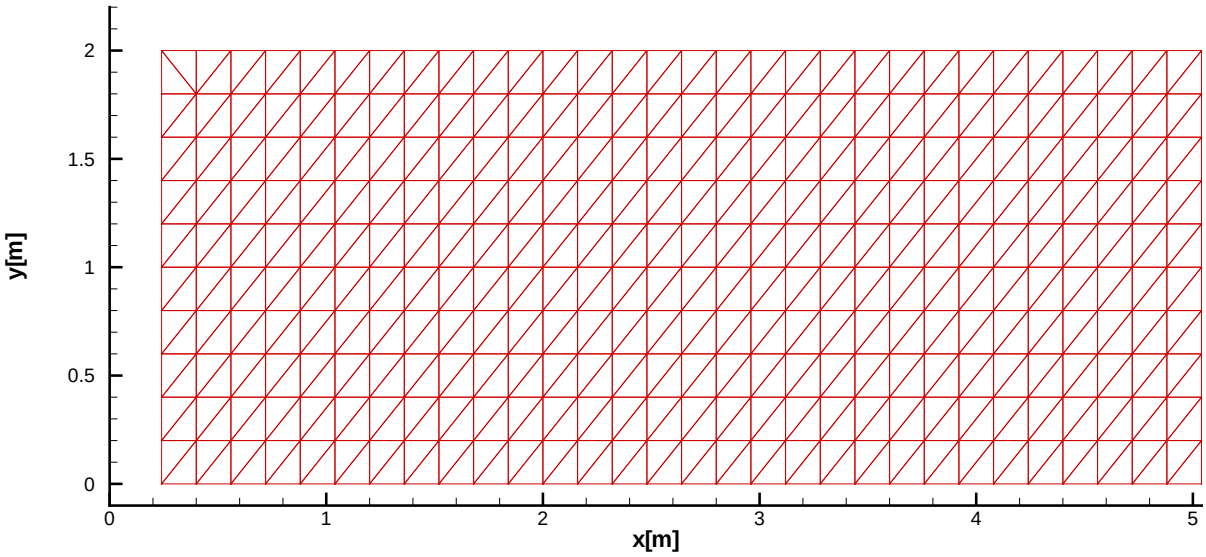
$*$ – profile much distorted, - – peak arriving ”too late”.

Courant number = 1: SUPG, linear, 2nd order



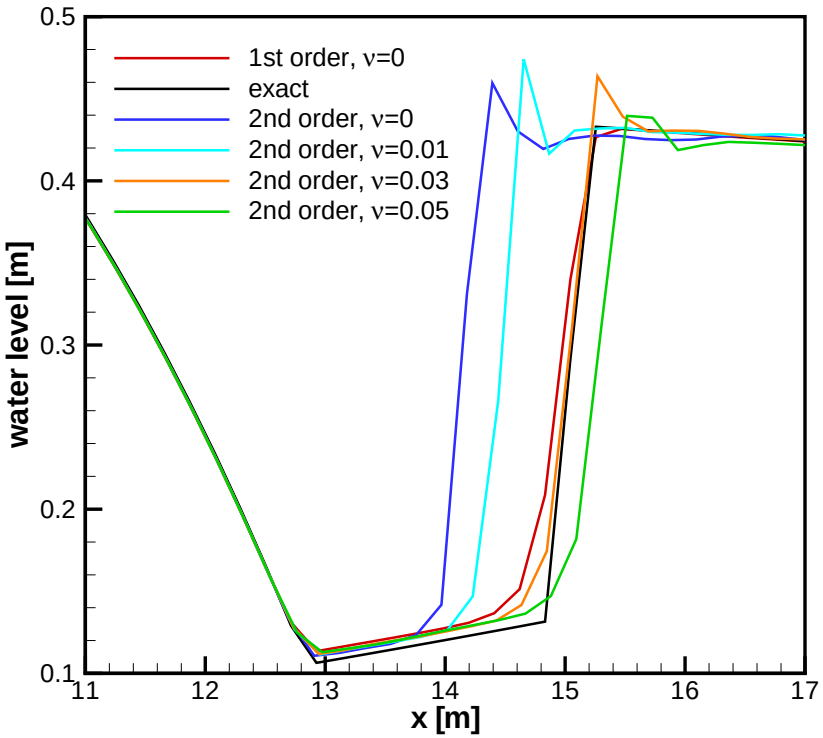
Hydraulic jump

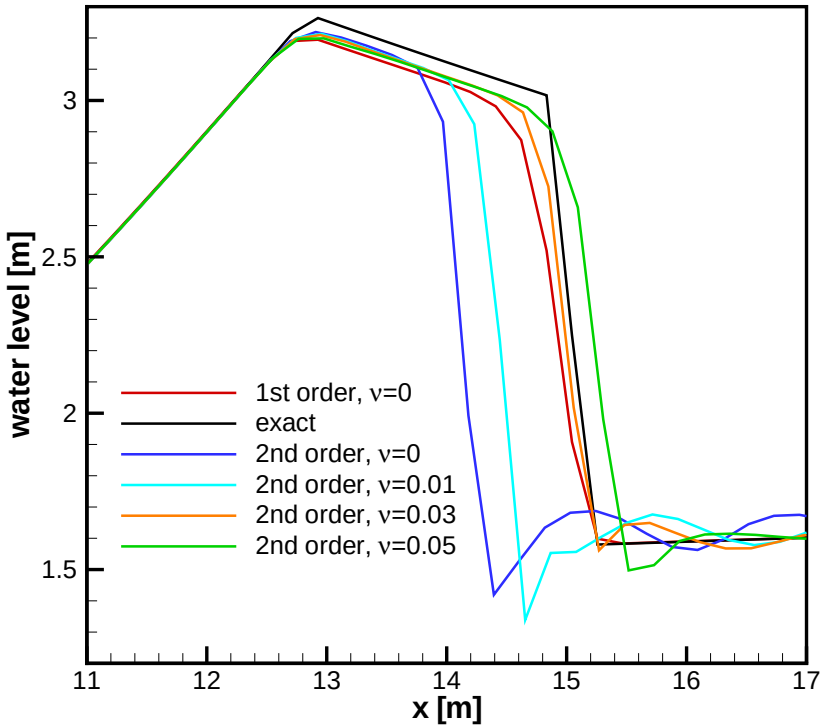




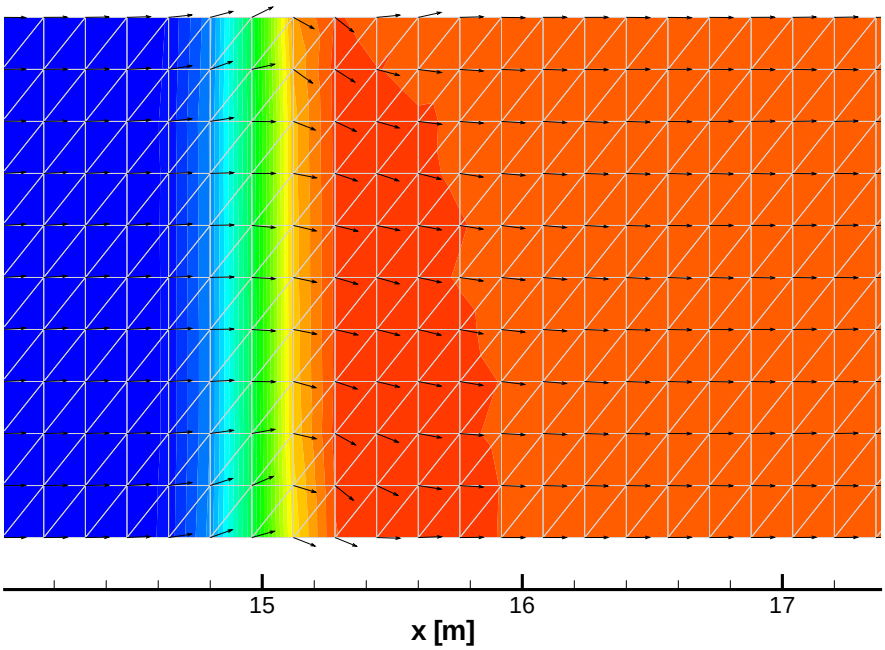
Courant number ≈ 0.1 ... 0.3

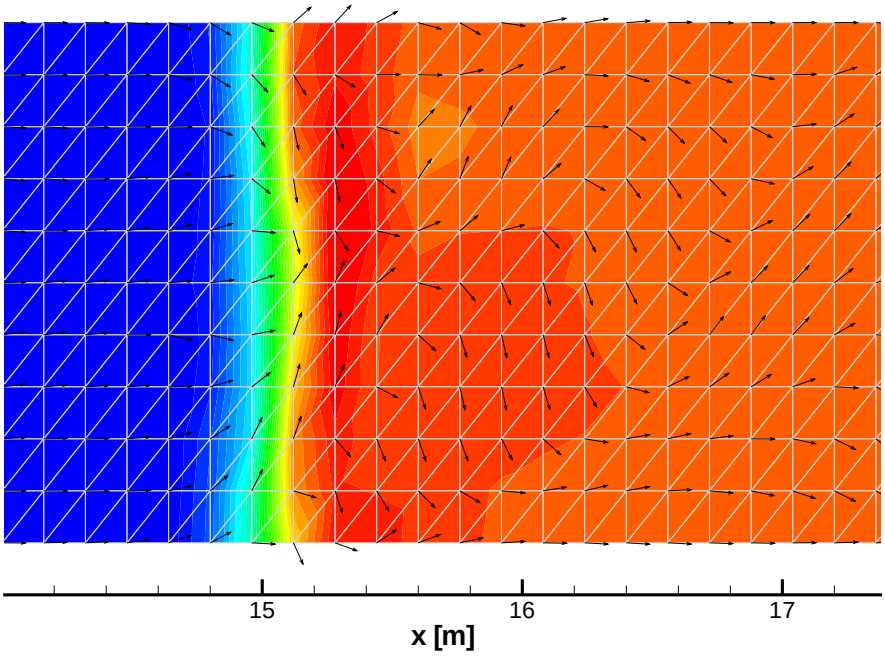
Hydraulic jump: elevation, velocity





Hydraulic jump: boundary problems





Conclusions

Good news:

- One of the best advection schemes is available for Telemac-2D!

Bad news:

- Present implementation seems to yield improvements for ”patch-friendly” (fine, regular...) meshes...
- Further work and testing required...

Further work?

Further testing and developments required:

- consistent interpolation in interior and boundary elements?
- under/overshooting treatment?
- theoretical investigations?
- optimization for high-performance machines?