Description of the depth-averaged two-phase flow code in hpGEM

Sander Rhebergen

23-06-2009

1 Introduction

In this note we give a description of the implementation of the discontinuous Galerkin finite element discretization of the depth-averaged two-phase flow equations as formulated in Rhebergen et al. [9]. The equations are of hyperbolic type (depending, however, on the parameters required - if parameters are chosen such that the equations are not hyperbolic, this code does not work). Furthermore, the PDE's have nonconservative products and we deal with these terms as described in [8] based on the theory of Dal Maso, LeFloch and Murat [1]. To deal with over- and undershoots, a WENO slope limiter is applied [6, 9] in conjunction with a discontinuity detector [4, 9].

2 The equations

Let the orientation of the Cartisian coordinate system be as is shown in Figure 1 in which θ is the angle of the x_1-x_2 plane with the horizontal. Depth-averaged variables are the particle volume fraction α, the fluid velocity vector $u = (u_1, u_2)$ and the solids velocity vector $v = (v_1, v_2)$ which are constant in the x_3 direction. The flow depth is given by h and the bottom topography by b. The constants $\varepsilon = H/L$ and $\rho = \rho^f/\rho^s$ represent the height to length ratio of the flow and the ratio between the fluid density ρ^f and the solids density ρ^s, respectively. The gravity vector is given by $\vec{g} = [g_1, g_2, -g_3]^T$ in which g_3 is the vertical component of the gravity (see Figure 1). The above quantities are all scaled and dimensionless. To obtain the variables in dimensional form, denoted by $(\cdot)^*$, we have used the following scalings: $[x^*, y^*] = L[x, y]$, $t^* = \sqrt{L/g^*}t$, $[u_1^*, u_2^*] = \sqrt{g^*L}[u_1, u_2]$, $[v_1^*, v_2^*] = \sqrt{g^*L}[v_1, v_2]$, $v_T^* = \sqrt{g^*L}v_T$, $[g_1^*, g_2^*, g_3^*]^T = g^*g[\sin(\theta), 0, \cos(\theta)]^T$ with g^* the acceleration of gravity. The depth-averaged two-phase flow model is given by [5, 7, 9]:

Continuity equations:

$$\partial_t(h(1-\alpha)) + \partial_x(h(1-\alpha)u_1) + \partial_y(h(1-\alpha)u_2) = 0 \tag{1a}$$

$$\partial_t(h\alpha) + \partial_x(h\alpha v_1) + \partial_y(h\alpha v_2) = 0 \tag{1b}$$

Momentum equations:

$$\partial_t(h(1-\alpha)u_1) + \partial_x(h(1-\alpha)u_1u_1) + \partial_y(h(1-\alpha)u_1u_2) + \varepsilon(1-\alpha)g_3h\partial_x h = S_x^f \tag{2a}$$

$$\partial_t(h(1-\alpha)u_2) + \partial_x(h(1-\alpha)u_2u_1) + \partial_y(h(1-\alpha)u_2u_2) + \varepsilon(1-\alpha)g_3h\partial_y h = S_y^f \tag{2b}$$

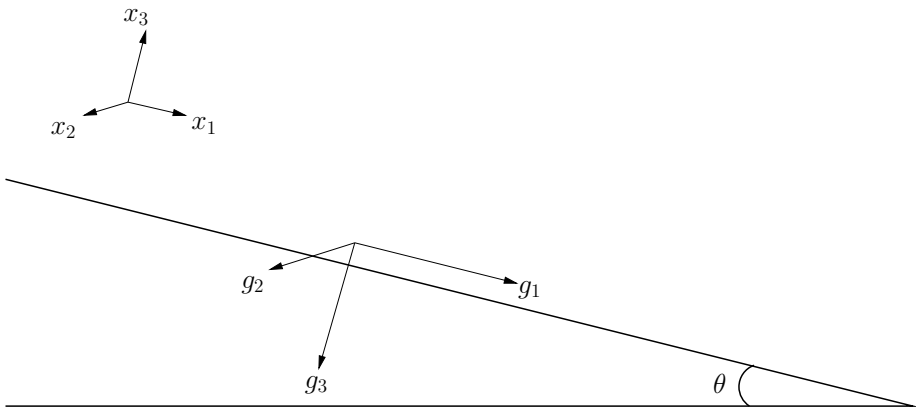

Figure 1: Orientation of the coordinate system and the gravity vector.

$$\partial_t(h\alpha v_1) + \partial_x(h\alpha v_1 v_1 + \varepsilon(1-\rho)\varphi_{11}\alpha\tfrac{1}{2}g_3h^2) + \partial_y(h\alpha v_1 v_2 + \varepsilon(1-\rho)\varphi_{12}\alpha\tfrac{1}{2}g_3h^2) + \varepsilon\rho\alpha g_3 h\partial_x h = S_x^s \quad (2c)$$

$$\partial_t(h\alpha v_2) + \partial_x(h\alpha v_2 v_1 + \varepsilon(1-\rho)\varphi_{21}\alpha\tfrac{1}{2}g_3h^2) + \partial_y(h\alpha v_2 v_2 + \varepsilon(1-\rho)\varphi_{22}\alpha\tfrac{1}{2}g_3h^2) + \varepsilon\rho\alpha g_3 h\partial_y h = S_y^s \quad (2d)$$

with

$$\begin{aligned} S_x^f &= -hF_1^D/\rho + h(1-\alpha)g_1 - (1-\alpha)C_D|u|u_1/\varepsilon \\ S_y^f &= -hF_2^D/\rho + h(1-\alpha)g_2 - (1-\alpha)C_D|u|u_2/\varepsilon \\ S_x^s &= (1-\rho)\varphi_{13}\alpha g_3 h + hF_1^D + g_1 h\alpha - \rho\alpha C_D|u|u_1/\varepsilon \\ S_y^s &= (1-\rho)\varphi_{23}\alpha g_3 h + hF_2^D + g_2 h\alpha - \rho\alpha C_D|u|u_2/\varepsilon \end{aligned} \quad (3)$$

Note that compared with the model by Le [5], we have added extra friction terms with the drag coefficient C_D as a leading order turbulence parameterization of the fluid phase. The latter is modelled such that it becomes the usual quadratic friction terms when $\alpha \to 1$ for a pure liquid phase [10].

The drag function F^D is a closure and has the general form $F_i^D = F_i^D(u_i, v_i, \alpha, \rho, d)$. Here we use the form given by Richardson and Zaki (see e.g. Jackson [3]):

$$F_i^D = \beta(u_i - v_i)$$

$$\beta = \frac{(1-\rho)\alpha}{v_T(1-\alpha)^n}, \quad n = \begin{cases} 3.65 & \text{for } Re_t < 0.2, \\ 4.35Re_t^{-0.03} - 1 & \text{for } 0.2 < Re_t < 1, \\ 4.45Re_t^{-0.1} - 1 & \text{for } 1 < Re_t < 500, \\ 1.39 & \text{for } 500 < Re_t, \end{cases} \quad (4)$$

in which d is the particle diameter, ρ^f the fluid density, μ_f the fluid viscosity, v_T the terminal velocity of an isolated particle falling in the fluid and $Re_t = d\rho_f v_T/\mu_f$. We remark that as

$1-\rho$ increases, the drag function F^D makes the system of PDE's increasingly stiffer. We are, however, interested in the case where ρ is approximately 0.9. In this situation the model does not have stiff source terms and no special algorithms are needed.

The functions φ were introduced by Pitman and Le [7] to relate basal and diagonal shear stresses to the normal stress in the solids phase stress tensor in the 3-dimensional two-phase model before depth-averaging. The functions φ are given by:

$$\begin{aligned}
&\varphi_{i3} = -\frac{v_i}{\|v\|}\tan(\phi_{bed}), \quad i=1,2, \qquad \varphi_{ii} = k^{\mp}, \quad i=1,2 \\
&\varphi_{12} = -\text{sign}(\partial_2 v_1)\sin(\phi_{int})k^{\mp}, \quad \varphi_{21} = -\text{sign}(\partial_1 v_2)\sin(\phi_{int})k^{\mp}, \\
&k^{\mp} = 2\frac{1 \mp \sqrt{1-\cos^2(\phi_{int})(1+\tan^2(\phi_{bed}))}}{\cos^2(\phi_{int})} - 1,
\end{aligned} \tag{5}$$

in which the "$-$" in the "$\mp$" applies when $\partial_k v_k > 0$ and the "$+$" applies when $\partial_k v_k < 0$. Furthermore, $\|\cdot\|$ is the Euclidean norm, ϕ_{int} is the internal angle of friction, which measures how layers of solid particles slide over one another and ϕ_{bed} is the basal angle of friction, indicating how easily solid particles slide over the bottom [2].

3 The weak formulation

For the notation and derivation of the weak formulation, see [8, 9]. The weak formulation is given by:

Find a solution $U \in W_h$ such that for all test functions $W \in W_h$:

$$\begin{aligned}
0 = &\sum_K \int_K \left(W_i U_{i,t} - W_{i,k}F_{ik} + W_i G_{ikr} V_{r,k} - W_i S_i\right) dK + \sum_{\mathcal{S}} \int_{\mathcal{S}} (W_i^L - W_i^R)\widehat{P}_i^{nc}\, d\mathcal{S} \\
&+ \sum_{\mathcal{S}} \int_{\mathcal{S}} \{\!\{W_i\}\!\} \left(\int_0^1 G_{ikr}(\phi(\tau;U^L,U^R)) \frac{\partial \phi_r}{\partial \tau}(\tau;U^L,U^R)\, d\tau\, n_k^L \right) d\mathcal{S}.
\end{aligned} \tag{6}$$

The last term makes the weak formulation different from standard discontinuous Galerkin finite element formulations. It is needed to introduce a measure for the nonconservative product where U is discontinuous. Note that an extra function, $\phi(\tau;U_L,U_R)$, has been introduced to deal with the regularization of U across the discontinuity. In [8] the effect of the choice of $\phi(\tau;U_L,U_R)$ on the numerical solution was investigated. We concluded that the numerical diffusion has a regularizing effect across discontinuities, which significantly reduces the dependence of the solution on $\phi(\tau;U_L,U_R)$, so that it often does not matter in practice how $\phi(\tau;U_L,U_R)$ is chosen. We adopt a linear path: $\phi(\tau;U_L,U_R) = U_L + \tau(U_R - U_L)$. Furthermore, we use here the NCP numerical flux $\widehat{P}^{nc}(U^L,U^R,n^L)$ designed in [8] for systems containing nonconservative products as a generalization of the HLL flux [11]. The NCP numerical flux $\widehat{P}^{nc}(U^L,U^R,n^L)$ reads:

$$\widehat{P}_i^{nc}(U_L,U_R,n^L) = \begin{cases} F_{ik}^L n_k^L - \frac{1}{2}\int_0^1 G_{ikr}(\bar{\phi}(\tau;U_L,U_R))\frac{\partial \bar{\phi}_r}{\partial \tau}(\tau;U_L,U_R)\, d\tau n_k^L & \text{if } S_L > 0, \\ \{\!\{F_{ik}\}\!\} n_k^L + \frac{1}{2}(S_R\bar{U}_i^* + S_L\bar{U}_i^* - S_L U_i^L - S_R U_i^R) & \text{if } S_L < 0 < S_R, \\ F_{ik}^R n_k^L + \frac{1}{2}\int_0^1 G_{ikr}(\bar{\phi}(\tau;U_L,U_R))\frac{\partial \bar{\phi}_r}{\partial \tau}(\tau;U_L,U_R)\, d\tau n_k^L & \text{if } S_R < 0, \end{cases} \tag{7}$$

with $\bar{U}^*$ given by:

$$\bar{U}_i^* = \frac{S_R U_i^R - S_L U_i^L + (F_{ik}^L - F_{ik}^R) n_k^L}{S_R - S_L} - \frac{1}{S_R - S_L} \int_0^1 G_{ikr}(\phi(\tau; U_L, U_R)) \frac{\partial \phi_r}{\partial \tau}(\tau; U_L, U_R)\, d\tau\, n_k^L. \quad (8)$$

The wave speeds S_L and S_R in the numerical flux are usually approximated by the minimum and maximum eigenvalues of the Jacobian matrix. The eigenvalues however cannot be calculated exactly and we approximate them as follows [9]:

$$\begin{aligned} \lambda_1 &= q_v \\ \lambda_2 &= q_u \\ \lambda_{3,4} &= q_u \pm \mathcal{A} \\ \lambda_{5,6} &= q_v \pm \mathcal{B} \\ \mathcal{A} &= \sqrt{\varepsilon g_3 h (1-\alpha)} \\ \mathcal{B} &= \sqrt{\tfrac{1}{2} h \varepsilon g_3 (1-\rho)(1+\alpha)(\varphi_{11} n_1^2 + \varphi_{22} n_2^2 + \varphi_{12} n_1 n_2 + \varphi_{21} n_1 n_2)}, \end{aligned} \quad (9)$$

in which $q_v = n_1 v_1 + n_2 v_2$ and $q_u = n_1 u_1 + n_2 u_2$.

4 Time discretization

By replacing U and W in the weak formulation (6) by their polynomial expansions, a system of ordinary differential equations, $dU/dt = R(U)$, is obtained, with $R(U)$ the residual. A third order TVD Runge Kutta scheme to step forward in time is used:

$$\begin{aligned} U^{(1)} &= u^n + \Delta t R(U^n) \\ U^{(2)} &= \tfrac{1}{4}\big(3U^n + U^{(1)} + \Delta t R(U^{(1)})\big) \\ U^{(3)} &= \tfrac{1}{3}\big(U^n + 2U^{(2)} + 2\Delta t R(U^{(2)})\big), \end{aligned} \quad (10)$$

where $U^n = U(t)$ and $U^{n+1} = U(t + \Delta t)$.

5 Slope limiter and discontinuity detector

In numerical discretizations of the weak formulation (6), spurious oscillations generally appear near discontinuities. Using the Krivodonova discontinuity detector [4], we apply a slope limiter only near discontinuities to deal with these spurious oscillations. We use the slope limiter given in [6] which we describe briefly here for reasons of clarity.

The idea of the slope limiter is to replace the original polynomial P_0 by a new polynomial P that uses the data u_m of the midpoint of the original element in element K_k and midpoints u_a, u_b, u_c and u_d of its neighboring elements. Eight polynomials are constructed, four Lagrange polynomials, P_i, $i = 1, 2, 3, 4$ and four Hermite polynomials P_i, $i = 5, 6, 7, 8$. For the Hermite polynomials we also need the physical gradient of the data in the neighboring elements at the points $\vec{x}$, i.e., ∇u_a, ∇u_b, ∇u_c and ∇u_d (see Fig. 2).

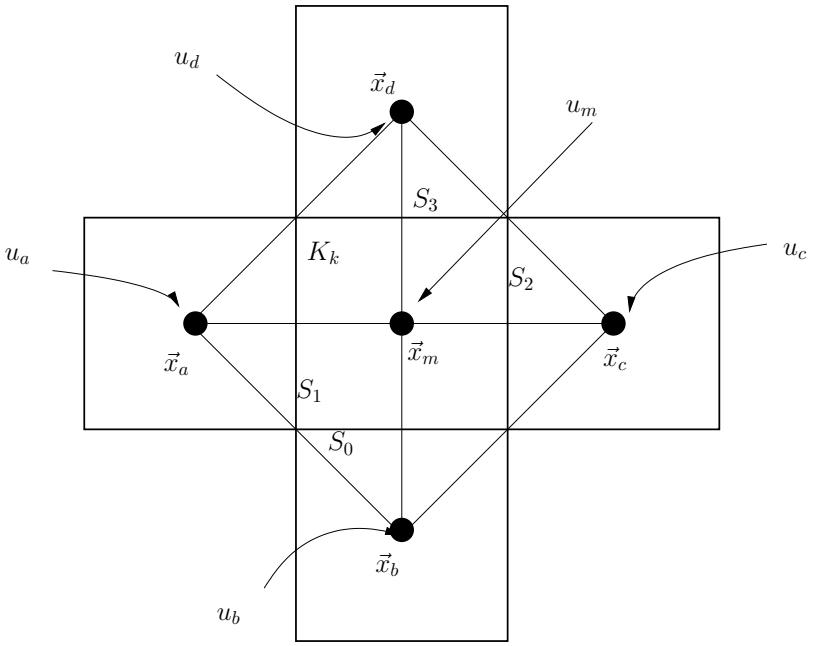

Figure 2: Slope limiter in 2D.

To construct the Lagrange polynomials consider the surface through x_m, x_a and x_b. Construct the polynomial through this surface $P_1 = \hat{P}_1^a + \hat{P}_1^b x + \hat{P}_1^c y$. The coefficients $\hat{P}_1^a$, $\hat{P}_1^b$ and $\hat{P}_1^c$ are found by solving:

$$\begin{bmatrix} 1 & x_m & y_m \\ 1 & x_a & y_a \\ 1 & x_b & y_b \end{bmatrix} \begin{bmatrix} \hat{P}_1^a \\ \hat{P}_1^b \\ \hat{P}_1^c \end{bmatrix} = \begin{bmatrix} u_m \\ u_a \\ u_b \end{bmatrix}.$$

In the same way, polynomials P_2, P_3 and P_4 are constructed by considering the remaining three surfaces.

Each of the four Hermite polynomials are determined by looking at the current element and one of the neighbors, e.g., the first Hermite polynomial, P_5, is found by looking at the neighboring element sharing face S_0. In the midpoint x_b, the gradient of the solution is ∇u_b, while the solution in the midpoint of the current element is u_m. The first Hermite polynomial is given by: $P_5 = \hat{P}_5^a + \hat{P}_5^b x + \hat{P}_5^c y$ where:

$$\begin{aligned} \hat{P}_5^a &= u_m - x_b \cdot \nabla u_b, \\ \hat{P}_5^b &= \partial_x u_b \quad \text{in } x_b, \\ \hat{P}_5^c &= \partial_y u_b \quad \text{in } x_b. \end{aligned}$$

In the same way, polynomials P_6, P_7 and P_8 are constructed by considering the remaining three surfaces.

The linear approximation of the original polynomial is determined just like the Hermite polynomials. In the midpoint x_m, the solution is u_m and the gradient is ∇u_m. The linear approximation is: $P_0 = \hat{P}_0^a + \hat{P}_0^b x + \hat{P}_0^c y$ where:

$$\begin{aligned}
\hat{P}_0^a &= u_m - x_m \cdot \nabla u_m, \\
\hat{P}_0^b &= \partial_x u_m \quad \text{in } x_m, \\
\hat{P}_0^c &= \partial_y u_m \quad \text{in } x_m.
\end{aligned}$$

Now project P_j, $j = 0, ..., 8$, onto the DG space and solve for $(\hat{u}_0)_j$, $(\hat{u}_1)_j$ and $(\hat{u}_2)_j$:

$$\begin{bmatrix} \int_{K_k} \psi_0\psi_0 \, dK & \int_{K_k} \psi_0\psi_1 \, dK & \int_{K_k} \psi_0\psi_2 \, dK \\ \int_{K_k} \psi_1\psi_0 \, dK & \int_{K_k} \psi_1\psi_1 \, dK & \int_{K_k} \psi_1\psi_2 \, dK \\ \int_{K_k} \psi_2\psi_0 \, dK & \int_{K_k} \psi_2\psi_1 \, dK & \int_{K_k} \psi_2\psi_2 \, dK \end{bmatrix} \begin{bmatrix} (\hat{u}_0)_j \\ (\hat{u}_1)_j \\ (\hat{u}_2)_j \end{bmatrix} = \begin{bmatrix} \int_{K_k} \psi_0 P_j \, dK \\ \int_{K_k} \psi_1 P_j \, dK \\ \int_{K_k} \psi_2 P_j \, dK \end{bmatrix}.$$

After the polynomial reconstruction is performed, an oscillation indicator is used to assess the smoothness of P_i. The oscillation indicator for the polynomial P_i, $i = 0, ..., 8$, is defined as $o_i = \|\nabla P_i\|$, with $\|\cdot\|$ the Euclidian norm. The coefficients of the new solution u in element K_k are constructed as the sum of all the polynomials multiplied by a weight, $\hat{u}_q = \sum_{i=0}^{8} w_i (\hat{u}_q)_i$, $q = 0, 1, 2$, in which the weights are computed as:

$$w_i = \frac{(\epsilon + o_i(P_i))^{-\gamma}}{\sum_{j=0}^{8} (\epsilon + o_i(P_j))^{-\gamma}}, \tag{11}$$

where γ is a positive number and ϵ a small number to avoid division by 0.

The discontinuity detector introduced in Krivodonova et al. [4] defines for each element K_k a measure of the discontinuity $\mathcal{I}_k$. This will indicate regions where the gradient of a variable $\mathcal{V}$ is large. For the depth-averaged two-phase flow equations, depending on the situation, we choose either $\mathcal{V} = h$ and $\mathcal{V} = \alpha$. The discontinuity detector is given by:

$$\mathcal{I}_k^n = \max(\mathcal{I}_k^n(h), \mathcal{I}_k^n(\alpha)), \quad \mathcal{I}_k^n(\mathcal{V}) = \frac{\sum_{\mathcal{S}_m \in \partial K_k} \int_{\mathcal{S}_m} |\mathcal{V}^R - \mathcal{V}^L| \, d\mathcal{S}}{h_K^{(p+1)/2} |\partial K_k| \|\mathcal{V}\|_\infty}, \tag{12}$$

where h_K is the cell measure defined as the radius of the largest circumscribed circle in the element K_k, p the polynomial order, $|\partial K_k|$ the surface area of the element and $\|\cdot\|_\infty$ the maximum norm. The solution is estimated [4] to be smooth when $\mathcal{I}_k < 1$ and non-smooth when $\mathcal{I}_k > 1$.

6 Implementation in hpGEM

In this section we describe the various files of the implementation in hpGEM.

6.1 contraction.hyb

Mesh file. This file was made using Matlab. Other programs can also be used to make mesh files.

6.2 dgPitLe.cc

class TecplotWriteFunction

Determine here (together with the names on Line 576) what you write to the output file.

OneFunction

Needed in "double compute_cell_measure" to compute the volume of an element.

toggle_time_slab

Data from previous time-step is updated to new time-level.

get_config

Read input file "in_TwoPhase_centaur".

get_mesh

Generates a mesh or reads the mesh file.

compute_cell_measure

Computes the volume of an element. Needed to determine the time step.

main

Main part of the code. Calls the prepocessing functions (L507-554), plots the initial condition (L558-582), does the time-stepping loop over the stages (L586-664), calls data updating (L666-737) and writes data to a file (L738-750).

6.3 dim.hh

A parameter used in the whole code: set the dimension of your problem.

6.4 EdgeContainer.hh, EdgeDescriptor.hh, EdgeDescriptor.icc, EdgeFactory.hh, EdgeFactory.icc, Edge.hh,MeshWithEdges.hh

Extending the depth-averaged two-phase flow space DG code to a space-time DG code is straight-forward. However, to extend the slope-limiter to space-time, we require support for edges. These files provide basic support for edges. The edges are used in the slope limiting procecure where limiting is performed only on the most recent time faces of the space-time mesh.

6.5 element_integration.cc

Addition of the element contributions of the weak formulation (calculated in element_integration.hh) to the Residual.

6.6 element_integration.hh

Calculation of all the element contributions of the weak formulation. The flux part F is calculated in "void ff" while the source terms S are calculated in "void fs". Multiplication of the F terms by the gradients of the basis-functions happens in (L313-318) while the source terms are multiplied by the basisfunctions in (L320-325).

6.7 face_integration.cc

Addition of the face contributions of the weak formulation (calculated in NumericalFlux.hh and ncNumericalFlux.hh) to the Residual.

6.8 face_integration.hh

Header file of "face_integration.cc"

6.9 GlobalDataContainer.hh

GloblaDataContainer.hh has State.hh and dim.hh. State.hh defines the flow field, numerical flux vector and residue vector per finite element. It sets expansion coefficients of each flow field and allocates memory for numerical flux and residue vector. It also stores the max wave speed in each finite element.

6.10 GlobalParam.hh

Declaration of all global parameters needed in the implementation.

6.11 initial_conditions.hh

Setup of your initial conditions.

6.12 in_TwoPhase_centaur

Input file where you can change your parameters depending on what you want to calculate.

6.13 Makefile

The make file. Comment L15 and uncomment L16 if you want to compile in debug mode. Uncomment L20 and L21 if you want to make a profile of your code.

6.14 MakeMeans.hh

It may happen, even though the code has a slope limiter, that the particle volume fraction α or the flow depth h become less that 0. If this is the case, then in the element where this occurs, we set the slopes to 0 so that only the mean is used in this element to determine the solution.

6.15 MatrixType.hh

Defines an ublas matrix extended with a multiplication operator. Required to define the FluxVectorWithWaveSpeed type with a matrix. Used in face integration.

6.16 ncNumericalFlux.hh

In "void calcNcGFlux" we calculate the term between the large brackets of

$$\sum_{\mathcal{S}} \int_{\mathcal{S}} \{\!\{W_i\}\!\} \Big(\int_0^1 G_{ikr}(\phi(\tau; U^L, U^R)) \frac{\partial \phi_r}{\partial \tau}(\tau; U^L, U^R)\, d\tau\, n_k^L \Big)\, d\mathcal{S}. \tag{13}$$

6.17 NumericalFlux.hh

Calculation of the NCP-flux.

6.18 Poly2D.hh

Describes what basis-functions we use.

6.19 PrimitiveMatrix.hh

For the time integration we use a third-order TVD Runge-Kutta scheme (10) to solve the system of ordinary differential equations, $dU/dt = R(U)$. If U is the vector of conservative variables and V is the vector of primitive variables, then we found it more efficient to insteady solve the following system of ordinary differential equations:

$$\frac{dU}{dV}\frac{dV}{dt} = R(U), \tag{14}$$

in which dU/dV is the primitive matrix determined in "PrimitiveMatrix.hh".

6.20 projection.hh

Projection of the continuous initial condition onto the discontinuous DG space.

6.21 runge_kutta.cc

void reset_residue

At the beginning of every Runge-Kutta stage the residual is reset to 0.

void do_runge_kutta

Depending on which stage we're computing, we update to the next stage according to (10).

6.22 runge_kutta.hh

The header file to "runge_kutta.hh".

6.23 ShearStressRelations.cc

The shearstress relations (5) are computed here.

6.24 ShearStressRelations.hh

Header file to "ShearStressRelations.cc".

6.25 SlopeLimiterEulerKrivSpace.hh

Calculation of the discontinuity detector and slope limiter (see Section 5).

6.26 State.hh

Declaring our unknowns.

6.27 SWtypedefs.hh

Type definitions of some important template objects and types. typedef customizes the names of templates which are intended to use several times.

6.28 vsz

A parameter used in the whole code. "vsz" is the number of PDE's you're solving.

7 How to run the code

1. change to directory hpgemdev/samples/../TwoPhaseFlow
2. in the Makefile, make sure that the path LIB_HPGEM_DIR is set correctly:
   LIB_HPGEM_DIR = ../../
3. type "make" in the shell
4. after compiling, run your code by typing "./dgPitLe" in the shell
5. input parameters can be changed in the "in_TwoPhase_centaur" file
6. contractiontec.dat contains the output data to be read with Tecplot

Without changing anything to the code, after compiling and running the code, the first 10 iterations on your screen should equal:

```
2.68 - Time: 0.0244584. Rh: 0.81254. Ra: 0.81254. Rvx: 0.81254. Rvy: 0.81254. Rux: 0.81254. Ruy: 0.81254.
timestep: 1
4.53 - Time: 0.0486998. Rh: 0.698582. Ra: 0.698582. Rvx: 0.698582. Rvy: 0.698582. Rux: 0.698582. Ruy: 0.698582.
timestep: 2
6.48 - Time: 0.0725483. Rh: 0.781962. Ra: 0.781962. Rvx: 0.781962. Rvy: 0.781962. Rux: 0.781962. Ruy: 0.781962.
timestep: 3
8.62 - Time: 0.0961327. Rh: 0.71563. Ra: 0.71563. Rvx: 0.71563. Rvy: 0.71563. Rux: 0.71563. Ruy: 0.71563.
timestep: 4
10.64 - Time: 0.119568. Rh: 0.616439. Ra: 0.616439. Rvx: 0.616439. Rvy: 0.616439. Rux: 0.616439. Ruy: 0.616439.
timestep: 5
12.66 - Time: 0.142749. Rh: 0.478024. Ra: 0.478024. Rvx: 0.478024. Rvy: 0.478024. Rux: 0.478024. Ruy: 0.478024.
timestep: 6
14.86 - Time: 0.165712. Rh: 0.418277. Ra: 0.418277. Rvx: 0.418277. Rvy: 0.418277. Rux: 0.418277. Ruy: 0.418277.
timestep: 7
16.93 - Time: 0.188464. Rh: 0.466356. Ra: 0.466356. Rvx: 0.466356. Rvy: 0.466356. Rux: 0.466356. Ruy: 0.466356.
timestep: 8
19.02 - Time: 0.211161. Rh: 0.489615. Ra: 0.489615. Rvx: 0.489615. Rvy: 0.489615. Rux: 0.489615. Ruy: 0.489615.
timestep: 9
21.08 - Time: 0.233679. Rh: 0.478148. Ra: 0.478148. Rvx: 0.478148. Rvy: 0.478148. Rux: 0.478148. Ruy: 0.478148.
timestep: 10
23.13 - Time: 0.25609. Rh: 0.433219. Ra: 0.433219. Rvx: 0.433219. Rvy: 0.433219. Rux: 0.433219. Ruy: 0.433219.
```

Loading the final data output file (contractiontec.dat) with tecplot, movies can be made of the different flow variables. A movie of the free-surface is also available on the hpGEM website (http://wwwhome.math.utwente.nl/%7Ehpgemdev/).

Acknowledgements

Thanks to Vijaya Ambati and Henk Sollie for their contributions to Section 6 and Onno Bokhove for his comments on the text.

References

[1] G. Dal Maso, P.G. LeFloch and F. Murat, Definition and weak stability of nonconservative products, J. Math. Pures Appl. 74 (1995) 483.

[2] K. Hutter and K.R. Rajagopal, On flows of granular materials, Continuum Mech. Thermodyn. 6 (1994) 81.

[3] R. Jackson, The dynamics of fluidized particles, Cambridge University Press, 2000.

[4] L. Krivodonova, J. Xin, J.F. Remacle, N. Chevaugeon and J.E. Flaherty, Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws, Appl. Numer. Math. 48 (2004) 323.

[5] L.H. Le, New models for geophysical flows, Ph.D. dissertation, State University of New York at Buffalo, 2006.

[6] H. Luo, J.D. Baum and R. Löhner, A Hermite WENO-based limiter for discontinuous Galerkin method on unstructured grids, J. Comput. Phys. 225 (2007) 686.

[7] E.B. Pitman and L. Le, A two-fluid model for avalanche and debris flows, Phil. Trans. R. Soc. A 363 (2005) 1573.

[8] S. Rhebergen, O. Bokhove and J.J.W. van der Vegt, Discontinuous Galerkin finite element methods for hyperbolic nonconservative partial differential equations, J. Comput. Phys. 227 (2008) 1887.

[9] S. Rhebergen, O. Bokhove and J.J.W. van der Vegt, Discontinuous Galerkin finite element method for shallow two-phase flows, Comput. Methods Appl. Mech. Engrg. , 198 (2009) 819.

[10] P.A. Tassi, S. Rhebergen, C.A. Vionnet and O. Bokhove, A discontinuous Galerkin finite element model for river bed evolution under shallow flows, Comput. Methods Appl. Mech. Engrg. , 197 (2008) 2930.

[11] E.F. Toro, Riemann solvers and numerical methods for fluid dynamics, Springer Verlag, 1997.