New general feature for v5p9 links from New version features

v5p9 General

Author: Jean-Michel Hervouet, 23rd January 2009

Distributive schemes: a new monotonicity criterion
Principle

Full explanations on distributive schemes are given in reference [2] from page 183 on. We just recall here that all such schemes end up in a discretization of advection equation in the form:

$$S_i(\frac{f_i^{n+1}-f_i^n}{\Delta t}) = -\beta_i \Phi_T$$

Where:

- S_i is the integral of the test function of point i,
- β_i is a distribution coefficient,
- Δt the time step,
- f_i^{n+1} the value of function f at point i after advection,
- f_i^n the value of function f at point i before advection
- Φ_T actually represents $S_i(\vec{u}.\overrightarrow{grad(f)})$ where $\vec{u}$ is the advecting field.

The only thing to know here is that $\beta_i \Phi_T$ eventually takes the form:

$$\beta_i \Phi_T = \sum_j \lambda_{ij}(f_i - f_j)$$

where the coefficients λ_{ii} are zero and all coefficients λ_{ij} are positive or zero. The sum is done on all the neighbouring points of point i, i.e. all the points that belong to an element containing i. The final value of function f at point i is thus:

$$f_i^{n+1} = f_i^n\left(1 - \frac{\Delta t}{S_i}\sum_j \lambda_{ij}\right) + \frac{\Delta t}{S_i}\sum_j \lambda_{ij} f_j^n \qquad (eq.1)$$

Classically, it is said then that the monotonicity criterion consists of ensuring that all the coefficients of f_i^n and f_j^n be positive, which yields, given the fact that all λ_{ij} are positive or zero: $\Delta t \leq \frac{S_i}{\sum_j \lambda_{ij}}$

However, this condition is sufficient, but not necessary. As a matter of fact, the real monotonicity criterion reads: $min(f_j^n) < f_i^{n+1} < max(f_j^n)$

If we now rewrite the equation $eq.1$ above in the form $f_i^{n+1} = f_i^n \Delta t \sum_j \frac{1}{S_i}\lambda_{ij}(f_j^n - f_i^n)$ and denoting $\mu = \sum_j \frac{1}{S_i}\lambda_{ij}(f_j^n - f_i^n)$ we have:

$$if\ \mu > 0 : \Delta t \leq \frac{max(f_j^n) - f_i^n}{\mu}$$

Depending on the sign of μ, the new criterion yields:

- and

- $if\ \mu < 0 : \Delta t \leq \frac{min(f_j^n) - f_i^n}{\mu}$

This criterion is better than the previous one, because (here example with $\mu > 0$) $max(f_j^n) - f_i^n$ is greater than $f_j^n - f_i^n$, thus $\frac{max(f_j^n) - f_i^n}{\mu} > \frac{S_i}{\sum_j \lambda_{ij}}$

Details on the implementation

We describe here the implementation of this new criterion in the SUBROUTINE **MURD3D** of the TELEMAC-3D module. In this subroutine the initial time step Δt is left unchanged, but a reduction factor α is introduced, so that the first time step is $\alpha \Delta t$. After one iteration, the remaining time is then $(1-\alpha)\Delta t$. A current time step DTJ is initialised with DT (here Δt), and then $\mathrm{DTJ} = (1-\alpha)\mathrm{DTJ}$ is done after every iteration. The problem is thus finding the reduction factor α.

What we have called S_i so far is $\int_\Omega \Psi_i^{n+1} d\Omega$, integral of the test function at the end of the time step. In SUBROUTINE **MURD3D** we have a varying mesh, with varying test function integrals, namely:

$$\mathrm{VOLUN} = \int_\Omega \Psi_i^n d\Omega \text{ and } \mathrm{VOLU2} = \int_\Omega \Psi_i^{n+1} d\Omega$$

At the first sub-iteration the end of the time step corresponds to an integral $\alpha\mathrm{VOLU2} + (1-\alpha)\mathrm{VOLUN}$ which is put in an array called $P(x,y) = \varphi(T^{-1}(x,y))$. After every sub-iteration the final test-function integral is $\mathrm{TRA01} = \alpha\mathrm{VOLU2} + (1-\alpha\mathrm{TRA01}$.

We take hereafter the example with $\mu > 0$. Note that in SUBROUTINE **MURD3D** $\sum_j \lambda_{ij}(f_j^n - f_i^n)$ is array $\mathrm{TRA02}$, so that $\mu = \mathrm{TRA02}/\mathrm{TRA01}$. In our case $\mathrm{TRA02}$ is positive.

$$\mathrm{DTJ} \leq \frac{max(f_j^n) - f_i^n}{\mu} \text{ i.e. } \mathrm{DTJ} \leq \frac{S_i[max(f_j^n) - f_i^n]}{\sum_j \lambda_{ij}(f_j^n - f_i^n)} \text{ now becomes:}$$
$$\alpha \leq (\alpha\mathrm{VOLU2} + (1-\alpha)\mathrm{TRA01}) \frac{max(f_j^n) - f_i^n}{\mathrm{DTJ}\sum_j \lambda_{ij}(f_j^n - f_i^n)}$$

Let us denote: $\mathrm{AUX} = \frac{\mathrm{DTJ}\sum_j \lambda_{ij}(f_j^n - f_i^n)}{max(f_j^n) - f_i^n}$

The old criterion corresponds to $\mathrm{AUX} = \mathrm{DTJ}\sum_j \lambda_{ij}$, with $\sum_j \lambda_{ij} = -\mathrm{DB}$ in SUBROUTINE **MURD3D**

We have: $\alpha \leq \frac{\alpha\mathrm{VOLU2} + (1-\alpha)\mathrm{TRA01}}{\mathrm{AUX}}$ which is also: $\alpha(1 - \frac{\mathrm{VOLU2} - \mathrm{TRA01}}{\mathrm{AUX}}) \leq \frac{\mathrm{TRA01}}{\mathrm{AUX}}$, AUX being positive, we eventually find that: $\alpha \leq \frac{\mathrm{TRA01}}{\mathrm{TRA01+AUX)\text{-}VOLU2}}$

If $\mu < 0$, so is $\mathrm{TRA02}$, and we get to the same result by choosing: $\mathrm{AUX} = \frac{-\mathrm{DTJ}\sum_j \lambda_{ij}(f_j^n - f_i^n)}{f_j^n - min(f_j^n)}$

In the Fortran implementation α is computed as $\frac{\mathrm{TRA01}}{\mathrm{TRA01} - \mathrm{TRA03}}$, so we have here either:

$$\text{if } \mathrm{TRA02} > 0 : \mathrm{TRA03} = \mathrm{VOLU2} - \frac{\mathrm{DTJ.TRA02}}{max(f_j^n) - f_i^n)}$$
$$\text{if } \mathrm{TRA02} < 0 : \mathrm{TRA03} = \mathrm{VOLU2} + \frac{\mathrm{DTJ.TRA02}}{f_i^n - min(f_j^n)})$$

This result can be compared with the old criterion which gave: $\mathrm{TRA03} = \mathrm{VOLU2} - \mathrm{DTJ}\sum_j \lambda_{ij}$

It happens then that the only thing to change with the new criterion is the value of $\mathrm{TRA03}$. The possible divisions by zero contained in the presence of $max(f_j^n) - f_i^n$ or $f_i^n - min(f_j^n)$ in the denominator are easily handled because in this case $\mathrm{TRA02}$ is also 0. As a matter of fact,

$\sum_j \lambda_{ij}(f_i^n - f_j^n)$ cannot be greater than zero if $f_i^n = max(f_j^n)$.

Results

As was said in previous sections the distributive schemes are implemented with sub-iterations that ensure that the monotonicity criterion will always be obeyed. We thus have less sub-iterations, which results in a less diffusive advection scheme. However, computing the local minimum and maximum of function f is an extra cost which could spoil the computer time. Tests have shown that in fact the computer time is reduced. For example if we run 10 time steps of a Berre lagoon computation, we get at the 10th time-step the following numbers of sub-iterations, for the 3 components of velocity U, V and W, and the tracer T (salinity):

	Old criterion	**New criterion**
U	3	2
V	3	2
W	3	2
T	4	2

The computer times on HP C3700 unix machine are 35 s (old criterion) and 32 s (new criterion), a reduction in time of about 10%. A reduction of 25% of the number of iterations is generally observed.

Parallelism: discovery of a new and necessary property
Principle

The use of the new monotonicity criterion for distributive schemes (see previous section) has highlighted an old problem in parallelism. It happened that an interface point, i.e. a point belonging to at least 2 sub-domains, could have a value of a given function (namely here $\mathrm{TRA02}$) positive in one sub-domain and negative in another, thus causing a bifurcation in the algorithm and eventually a crash due to a number of sub-iterations beyond the accepted threshold of 100. Incident reports showed that in such cases a restart of the program just before the crash would suppress the problem (it actually forced the values to be the same, by reading them in a file). We thus get to the conclusion that to avoid problems in algorithms containing tests on real numbers:

An interface point must have exactly the same values in all the sub-domains it belongs to

As a matter of fact, it appeared in tests that values that should have been equal, e.g. depth or velocity, could be slightly “sub-domain-dependent”, due to truncation errors. A first analysis showed that the computation of normal vectors to boundaries could yield di¤erent values, due to the fact that coordinates of boundary points outside a sub-domain were stored and retrieved from a file without a sufficient number of digits (this is a case of forced truncation error). This problem was solved, but some differences remained, where do they come from ?

If we think of how the operations are done, it seems that we can prove that if we start from equal values, they should remain equal, because the same operations done on the same figures should yield the same result. Let us for example look at the iterative solvers: At every iteration they do operations like: $\rho^m =< r^m, d^m > / < d^m, Ad^m >$ and $X^{m+1} = X^m - \rho^m d^m$. This is a part of the conjugate gradient algorithm, with 2 dot-products and 1 matrix-vector product Adm. If we start from the same r^m, d^m, and X^m and can show that ρ^m is the same, it is obvious that X^{m+1} is also the same (for an interface point, seen from two different sub-domains). So we have to show that

dot-product and matrix-vector product give identical values at interfaces.

: Dot-product: this property is obvious because the dot product is a single value gathering contributions from all processors and then sent back to all processors with a special communication (FUNCTION **P_DOTS**, which calls FUNCTION **P_DSUM**, which calls the MPI SUBROUTINE **MPI_AllReduce**). : Matrix-vector product: during a matrix-vector product, every interface point gathers information from its neighbouring elements. Then this information is passed to its “brothers” of other sub-domains, to be added to their contribution. This is done by the BIEF SUBROUTINE **PARCOM** with option 2. With two processors, two “brothers” will first get respectively quantities A and B. Then the first one will receive B and compute $\mathrm{A}+\mathrm{B}$. The second will receive A and compute $\mathrm{B}+\mathrm{A}$. Both points will have exactly the same result in the end. With 3 processors and 3 “brothers” with quantities A, B and C, we can have combinations like $(\mathrm{A}+\mathrm{B})+\mathrm{C}$ and $\mathrm{A}+(\mathrm{B}+\mathrm{C})$ and this will give differences due to truncation errors. This problem was first pointed out at LNHE by Charles Moulinec.

The conclusion is that iterative solvers, and more generally matrix-vector products will yield differences between brother points. Tests confirm the analysis. A SUBROUTINE **CHECK_DIGITS** was built to look for differences at interface points. In this subroutine, calls to SUBROUTINE **PARCOM** with option 3, i.e. giving the maximum between brother points, are used. The maximum is then compared with the local value. Comparisons are done between double precision numbers with tests like IF(X.NE.Y) THEN... (this is generally never done in Fortran programming to allow truncation errors, but here it is exactly what we want). In TELEMAC-3D with an option calling TELEMAC-2D, in the test-case “gouttedo” with 2 processors, there is no difference on depth and velocities. With 3 processors, there is a difference after the first time-step, at only one point, the single point that belongs to 3 sub-domains. This difference is of the order 10^{-18}.

Forcing equality of arrays at interface points

We have shown that differences between sub-domains occur only when calling SUBROUTINE **PARCOM** with option 2, for adding contributions stemming from sub-domains. These differences are only truncation errors. The idea is then to do a second call to SUBROUTINE **PARCOM**, with option 3 to choose the maximum between all values. This can be done in fact within the SUBROUTINE **PARCOM**, by calling SUBROUTINE **PARACO** twice. This is done only if option 2 is asked, and if there are more than 2 processors. Once this is achieved no more differences are noticed between two sub-domains. This has been checked on TELEMAC-2D, TELEMAC-3D and ESTEL-3D. However differences may still occur with a scalar computation, because operations like sums are still not done in the same order than in scalar.

A series of tests is now necessary to evaluate the cost of this double call to SUBROUTINE **PARACO**. If it were important, it would be necessary to build a new data structure to find more easily the maximum values between interface points that belong to more than 2 sub-domains. For example, if there is only one triple point (case with 3 subdomain) a call to FUNCTION **P_DMAX** for this very point is much faster to force a common value. Another idea would be to sort the quantities to be added with respect to their processor number.

Finite volume advection scheme: new options

For full details on this advection scheme, refer to [3]. This finite volume advection scheme is now available for suspension in SISYPHE and in TELEMAC-2D for the velocities. We detail hereafter how source terms have been dealt with, and how a variant has been designed to cope with advection

fields that do not obey the continuity equation, as is the case in one option of SISYPHE.

Source terms

We now add an explicit and an implicit source term in the tracer equation. So that the equation reads:

$$\frac{\partial(hC)}{\partial t} + div(hC\overrightarrow{u}) = h.\mathrm{SM} + C.\mathrm{SMI}(1)$$

SM is the explicit source term, SMI the implicit source term. They are currently treated in TELEMAC (SUBROUTINE **CVDFTR**) in the non-conservative equation in the form:

$$\frac{\partial C}{\partial t} + div(C\overrightarrow{u}) = \mathrm{SM} + C.\mathrm{SMI}/h$$

In suspended sediment transport, a positive SM would correspond to erosion and a negative SMI would correspond to deposition. Including also boundary terms and punctual source terms, equation $(??)$ is discretized in the form:

$$\mathrm{S}_i \frac{to}{\Delta t(h_i^{n+1}C_i^{n+1} - h_i^n C_i^n) + \sum_j C_{ij}\Phi_{ij} + b_i C_i^{boundary} - Scc_i C_i^{scc} = h^{n+1}S_i\mathrm{SM}_i + C_i^{n+1}S_i\mathrm{SMI}_i(1)}$$

where C_{ij} is equal to C_i^n if Φ_{ij} is positive, i.e. from i to j, and C_{ij} is equal to C_j^n if Φ_{ij} is negative (upwind explicit scheme). Other terms account for boundary fluxes and punctual sources terms. When all equations are summed, the terms $C_{ij}\Phi_{ij} + C_{ji}\Phi_{ji}$ will all sum to zero.

Equation $(??)$ gives the following value of C_i^{n+1}:

$$C_i^{n+1} = \frac{h_i^n C_i^n - \frac{\Delta t}{S_i}(b_i C_i^{boundary} - Scc_i C_i^{scc}) - \frac{\Delta t}{S_i}\sum_j C_{ij}\Phi_{ij}}{h_i^{n+1}} + \frac{\Delta t}{S_i}\mathrm{SM}_i + \frac{\Delta t}{S_i}\frac{C_i^{n+1}}{h_i^{n+1}}\mathrm{SMI}_i(1)$$

which can as well be written:

$$C_i^{n+1} = C_i^n - \frac{\Delta t b_i}{h_i^{n+1}S_i}(C_i^{boundary} - C_i^n) - \frac{\Delta t}{h_i^{n+1}S_i}\sum_j (C_{ij} - C_i^n)\Phi_{ij} + \frac{\Delta t Scc_i}{h_i^{n+1}S_i}(C_i^{scc} - C_i^n) + \Delta t\mathrm{SM}_i + \Delta t\frac{C_i^{n+1}}{h_i^{n+1}}\mathrm{SMI}_i(1)$$

by using the fact that the continuity equation is written:

$$h_i^{n+1} = h_i^n - \frac{\Delta t}{S_i}(-Scc_i + \sum_j \Phi_{ij} + b_i)(1)$$

With an explicit upwind scheme, it gives:

$$C_i^{n+1} = \left[1 + \frac{\Delta t}{h_i^{n+1}S_i}\left(\sum_j \min(\Phi_{ij}, 0) + \min(b_i, 0) - \max(Scc_i, 0)\right)\right] C_i^n - \frac{\Delta t}{h_i^{n+1}S_i}\left(\sum_j \min(\Phi_{ij}, 0) + \min(b_i, 0)C_i^{boundary} - \max(Scc_i, 0)C_i^{scc}\right)$$

$$+\Delta t\ SM_i + \Delta t \frac{C_i^{n+1}}{h_i^{n+1}} SMI_i$$

This new formula would require a new monotonicity criterion, however explicit and implicit source terms do not necessarily obey the monotonicity criterion, so we choose to keep the previous criterion. If we call C_i^{old} the previous C_i^{n+1} obtained without our new explicit and implicit terms, we have:

$$C_i^{n+1} = C_i^{old} + \Delta t\ SM_i + \Delta t \frac{C_i^{n+1}}{h_i^{n+1}} SMI_i$$

and we get at the final result:

$$C_i^{n+1} = \left(C_i^{old} + \Delta t\ SM_i\right) / \left(1 - \Delta t \frac{SMI_i}{h_i^{n+1}}\right)$$

Great care must be taken that SMI_i remains negative to keep the monotonicity. This is the case in Sisyphe, where the deposition has been intentionally kept fully implicit, with the very purpose of having positive concentrations.

A variant for Sisyphe

In Sisyphe there is an option “correction on convection velocity” that takes into account the fact that the suspended sediment has a maximum concentration near the bottom, and is thus not advected with the depth-averaged velocity. In this case the advecting field is the depth averaged velocity multiplied by a correction factor, and the resulting velocity field does not obey the continuity equation 3.5. This should not be a problem for finite volumes as we solve the conservative equation, but unfortunately in the derivation of the scheme we use equation 3.5 to get the new depth and this equation is no longer valid. In fact the use of this continuity equation consists of replacing the coefficient of C_i^n, which is:

$$coef = \frac{h_i^n - \frac{\Delta t}{S_i}\left(-Scc_i + \sum_j \Phi_{ij} + b_j\right)}{h_i^{n+1}}$$

by 1. The denominator is the real depth, the numerator is the depth that would be obtained with the continuity equation, if the non conservative field were to be used. They are equal when a conservative velocity field is chosen. The variant consists of choosing the coefficient $coef$ instead of 1 for C_i^n. This new coefficient changes the monotonicity criterion but we decide to keep the previous criterion. As a matter of fact, when continuity is not ensured by the velocity field, the monotonicity cannot be guaranteed. However, $coef$ should remain a positive number, being a ratio of two depths, and the concentration should remain positive. The test-case for exemplifying the effect of this new variant is called “16.conservation” in French test-cases (test.fr) of Sisyphe 5.9 package. It is in fact the Telemac-2D “gouttedo” test with mobile bed. There is bedload and suspension. With coefficient 1 for C_i^n the total loss of sediment is 0.1477758 10^2 m^3. With the variant it becomes 0.1195841 10^{-16} which is about the machine accuracy. Note: in the implementation in subroutine TVF, the “depth that would be obtained with the continuity equation” was already computed and is kept as H, h_i^{n+1} is now called $HLIN$. As a matter of fact, sub-iterations for monotonicity have to be taken into account, and the intermediate depths can be obtained by linear interpolation between h_i^n and h_i^{n+1}. This is true because the fluxes are kept constant during the time step.

Coupling with Delwaq: now possible with tidal flats

The coupling with Delwaq relies on a common understanding (between finite elements and finite volumes) of the continuity equation. In the case of tidal flats, the treatment of negative values on finite elements side is rather complex and so far could not be transmitted to finite volumes. As a matter of fact, the treatment of tidal flats cannot be translated into a modification of the velocity field in the continuity equation, as was done e.g. for the “compatibility of free surface gradient” (see section “suppressing wiggles” in reference [3]). A solution came from Leo Postma at Deltares, who emitted the remark that Delwaq only needs fluxes, regardless of the velocities that created them. The problem then only consists of finding the fluxes between points that result from the treatment of tidal flats, and this is indeed possible and explained hereafter. We first recall the ideas implemented in the interface between Telemac and Delwaq. We assume that the 2D continuity equation:

$$\frac{h^{n+1}-h^n}{\Delta t}+\mathrm{div}(h[\theta_u\,\overrightarrow{u}^{n+1}+(1-\theta_u)\,\overrightarrow{u}^n])=Sce$$

has been discretized in the following matrix form:

$$\frac{M}{\Delta t}(H^{n+1}-H^n)+BM1\;U^{n+1}+BM2\;V^{n+1}=CV1$$

where M is the mass-matrix and Δt the time-step. If we use mass lumping, this will give for every degree of freedom i:

$$\frac{S_i}{\Delta t}(h_i^{n+1}-h_i^n)+(BM1\;U^{n+1}+BM2\;V^{n+1}-CV1)_i=0$$

where $S_i=\int_\Omega \Psi_i \mathrm{d}\Omega$ is as well the area of the finite volume around point i, and what we call the volume of basis i. If there is no mass lumping, this equation remains valid if h_i^{n+1} and h_i^n are replaced with $\tilde{h}_i^{n+1}$ and $\tilde{h}_i^n$, where:

$$\tilde{h}_i=\frac{(MH)_i}{S_i}$$

which is in fact what says exactly Equation 4.1. The quantity:

$$(BM1\;U^{n+1}+BM2\;V^{n+1}-CV1)_i$$

can be interpreted as the flux leaving point i, it is in fact generally (i.e. without source terms and boundary fluxes): $-\int_\Omega h\,\overrightarrow{u}.\overrightarrow{grad}(\Psi_i)\mathrm{d}\Omega$ These fluxes leaving points are then translated into fluxes between points as explained in reference [3]. If we can now write the continuity equation in the form:

$$\frac{S_i}{\Delta t}(H_i^{n+1}-h_i^n)+\text{effect of tidal flats treatment}+(BM1\;U^{n+1}+BM2\;V^{n+1}-CV1)_i=0$$

then the effect of tidal flats treatment on depth can be simply added to the other fluxes and treated in the same way. The only condition is that it must be done at the element level, as for the real fluxes. Now let us detail the treatment of tidal flats. It is composed of a modification of the free surface gradient in the momentum equation (which spoils nothing in the continuity equation) and of a smoothing of negative depths (which is the problem here). The smoothing is done in the following way:

- Negative depths are extracted in a work array $T1$: for every point i, $T1(i)=\min(H(i);0.D0)$
- Negative depths are removed from depth: $H(i)=H(i)-T1(i)$

- Negative depths are smoothed, i.e. $T1$ is replaced by $\frac{MT1}{S}$,where M is the mass matrix and S is the lumped mass matrix. This is done several times (currently a value of two is hardcoded)

- Smoothed negative depths are added to others: $H(j) = H(j) + T1(j)$

It is important to understand that after this treatment, depths may still be negative, they are just less negative. The difference is that without treatment the depth may become infinitely negative. Smoothing stabilizes the numerical scheme. The depth smoothing is mass-conservative in the sense that the integral of depth is left unchanged. The only problem is that the continuity equation is spoiled. The effect of one smoothing is the following, if we call h^{old} the depth before smoothing and h^{new} the depth after:

$$\frac{S_i}{\Delta t}(H_i^{new} - h_i^{old} = \frac{(MT1)_i - S_i T1_i}{\Delta t}$$

such contributions should be added if there is more than one smoothing. We deduce from this that the continuity equation that is actually solved is:

$$\frac{S_i}{\Delta t}(h_i^{n+1} - h_i^n) - \frac{(MT1)_i - S_i T1_i}{\Delta t} + (BM1\ U^{n+1} + BM2\ V^{n+1} - CV1)_i = 0$$

At element level, this new contribution can be easily computed, by using bits of code from the Telemac subroutines computing $\int_\Omega F\Phi_i \mathrm{d}\Omega$ and $\int_\Omega \Phi_i \mathrm{d}\Omega$. It gives for $\frac{(MT)_i - S_i T1_i}{\Delta t}$ ($T1$ would be used as the function **F**, SURFAC is the area of the element):

```
DO IELEM = 1 , NELEM
C
C Values of function F at the 3 points of the triangle
F1 = F(IKLE1(IELEM))
F2 = F(IKLE2(IELEM))
F3 = F(IKLE3(IELEM))
C
COEF = SURFAC(IELEM)/12.D0
C
W1(IELEM) = COEF * ( F2+F3-2*F1 )
W2(IELEM) = COEF * ( F1+F3-2*F2 )
W3(IELEM) = COEF * ( F1+F2-2*F3 )
C
ENDDO
```

This solution applied to the Malpasset test case show a full validation of the approach, the control of mass error done by the interface between Telemac and Delwaq (subroutine **TEL4DEL**) gives at every time step: maximum mass error without correction: order of 100. maximum mass error with correction: order of 10^{10}.

TELEMAC-2D: quadratic elements

This implementation of quadratic elements in TELEMAC-2D has been successfully performed by Algiane Froehly (MATMECA, Bordeaux). It has then been adapted for parallelism. We give hereafter some basic explanations. A full account can be found in her report (reference [1])

A quadratic interpolation of the velocity eld is a well-known solution to stabil- ity problems raised by the Ladyzhenskaya-Babus ka-Brezzi condition in Navier Stokes equations (also called discrete inf-sup condition). The pressure (or the depth in Shallow Water equations) remains linear. For quadratic interpolation, we add 3 degrees of freedom, numbered 4, 5 and 6 on Figure 5.1. We recall here that the linear bases carried by points 1, 2 and 3 are :

$$\begin{aligned}\lambda_1(\alpha,\beta) &= (1-\alpha-\beta)\\ \lambda_2(\alpha,\beta) &= \alpha\\ \lambda_3(\alpha,\beta) &= \beta\end{aligned}$$

where α and β are the coordinates in the reference triangle given in Figure 5.2: The coordinates of the 6 degrees of freedom are:

$$\begin{aligned}\text{point } 1 &: (0,0)\\ \text{point } 2 &: (1,0)\\ \text{point } 3 &: (0,1)\\ \text{point } 4 &: (\tfrac{1}{2},0)\\ \text{point } 5 &: (\tfrac{1}{2},\tfrac{1}{2})\\ \text{point } 6 &: (0,\tfrac{1}{2})\end{aligned}$$

The quadratic interpolation polynoms $P_i(x,y)$, with i = 1...6, are such that $P_i(x_j,y_j)=\delta_{ij}$ where T is the isoparametric transformation that gives the real triangle as a function of the reference triangle and φi are the basis functions in the reference triangle. In practice T^{-1} is never built and the computation of integrals is done in the reference triangle.

The 6 quadratic basis $\varphi_i(\alpha,\beta)$ are chosen to ensure the following property:

Figure 5.1: The six points of a quadratic triangle

$$\sum_{i=1}^{6}\varphi_i(\alpha,\beta)=1,\forall(\alpha,\beta)\in triangle$$

Moreover every basis must be equal to 1 on its own point and zero on the five others. This is veri ed if we take:

$$\begin{aligned}&\text{For } i=1,2,3,\varphi_i(\alpha,\beta)=(2\times\lambda_i(\alpha,\beta)-1)\times\lambda_i(\alpha,\beta)\\ &\quad\text{and for } i=4,5,6,\varphi_i(\alpha,\beta)=4\times\lambda_k(\alpha,\beta)\times\lambda_l(\alpha,\beta)\end{aligned}$$

where k and l are the indices of points of the segment where is point i. More precisely:

$$\varphi_1(\alpha,\beta) = (2 \times \lambda_1(\alpha,\beta) - 1) \times \lambda_1(\alpha,\beta)$$
$$\varphi_2(\alpha,\beta) = (2 \times \lambda_2(\alpha,\beta) - 1) \times \lambda_2(\alpha,\beta)$$
$$\varphi_3(\alpha,\beta) = (2 \times \lambda_3(\alpha,\beta) - 1) \times \lambda_3(\alpha,\beta)$$
$$\varphi_4(\alpha,\beta) = 4 \times \lambda_1(\alpha,\beta) \times \lambda_2(\alpha,\beta)$$
$$\varphi_5(\alpha,\beta) = 4 \times \lambda_2(\alpha,\beta) \times \lambda_3(\alpha,\beta)$$
$$\varphi_6(\alpha,\beta) = 4 \times \lambda_3(\alpha,\beta) \times \lambda_1(\alpha,\beta)$$

Figure 5.2: reference triangle

Remark: on boundaries a point number 3 is added in the middle and the interpolation polynoms are:

$$\begin{aligned}\varphi_1(\xi) &= 2 \times (1-\xi) \times (\tfrac{1}{2} - \xi)\\ \varphi_2(\xi) &= (2 \times \xi - 1) \times \xi\\ \varphi_3(\xi) &= 4 \times \xi \times (1-\xi)\end{aligned}$$

As they have 6 points, these new quadratic elements have a similarity with prisms, e.g. element matrices have the same number of extra-diagonal terms. They are thus stored in the same way, i.e.:

$$\begin{pmatrix} . & 1 & 2 & 3 & 4 & 5 \\ 16 & . & 6 & 7 & 8 & 9 \\ 17 & 21 & . & 10 & 11 & 12 \\ 18 & 22 & 25 & . & 13 & 14 \\ 19 & 23 & 26 & 28 & . & 15 \\ 20 & 24 & 27 & 29 & 30 & . \end{pmatrix}$$

Rectangular matrices are stored in the following way, for 3 x 6 matrices:

$$\begin{pmatrix} . & 1 & 2 & 3 & 4 & 5 \\ 6 & . & 7 & 8 & 9 & 10 \\ 11 & 12 & . & 13 & 14 & 15 \end{pmatrix}$$

and for 6 x 3 matrices:

$$\begin{pmatrix} . & 1 & 2 \\ 3 & . & 4 \\ 5 & 6 & . \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{pmatrix}$$

Most subroutines written for prisms may as well be re-used for quadratic triangles. All the subroutines computing matrices or vectors have been built with Maple V.

The list of new or modi□ed Maple-built subroutines in the BIEF library and their function is given hereafter.

□ϕ represents the test functions and ϕ the basis.

MT01CC: mass-matrix of size 6 x 6
FORMUL = 'MATMAS' : MT01CC$_{ij}$ = $\int_\Omega \phi_i \Psi_j d\Omega$

MT02CC : diffusion matrix of size 6 x 6
FORMUL = 'MATDIF' : MT02CC$_{ij}$ = $\int_\Omega \nu \times \vec{\nabla}\phi_i.\vec{\nabla}\Psi_j \mathrm{d}\Omega$
Viscosity ν is left linear, with same cases (isotropic or not).

MT03CC : used for advection with SUPG technique, of size 6 x 6
FORMUL = 'MASUPG' : MT03CC$_{ij}$ = $\int_\Omega \vec{F}\phi_i.\vec{U}\vec{\nabla}\Phi_j \mathrm{d}\Omega$
Function F is piece-wise constant and U is linear or quadratic.

MT04AA : SUPG matrix, of size 3 x 3
FORMUL = 'MAUGUG' : MT04AA$_{ij}$ = $XMUL \int_\Omega ((\vec{U}).\vec{\nabla}\Psi_i)((\vec{U}).\vec{\nabla}\Psi_j)\mathrm{d}\Omega$
The case of quadratic velocity has been added.

MT04CC : SUPG matrix of size 6 x 6
FORMUL = 'MAUGUG' : MT04CC$_{ij}$ = $XMUL \int_\Omega ((\vec{U}).\vec{\nabla}\Psi_i)((\vec{U}).\vec{\nabla}\Psi_j)\mathrm{d}\Omega$
U and V may be linear or quadratic.

MT05AA : linear advection matrix, of size 3 x 3, with variants for distributive schemes
FORMUL = 'MATVGR' : MT05AA$_{ij}$ = $XMUL \int_\Omega \Psi_i \times \vec{U}.\vec{\nabla}\{Psi_j \mathrm{d}\Omega$
The velocity may now be quadratic, but the N scheme only uses linear points.

MT05CC : quadratic advection matrix, of size 6 x 6, with variants for distributive schemes
FORMUL = 'MATVGR' : MT05CC$_{ij}$ = $XMUL \int_\Omega \Psi_i\vec{U}.\vec{\nabla}\{Psi_j \mathrm{d}\Omega$
The velocity may be piece-wise constant, linear or quadratic, but the N scheme only uses linear points.

MT06OC : size 3 x 3 (boundary matrix).
FORMUL = 'FMATMA' : MT06OC$_{ij}$ = $XMUL \int_{segment} F\Psi_i\Psi_j \mathrm{d}l$
F may be piece-wise constant, linear or quadratic. The connectivity table giving the access to the boundary points has been extended.

MT06CC : size 6 x 6
FORMUL = 'FMATMA' : MT06CC$_{ij}$ = $XMUL \int_{segment} F\Psi_i\Psi_j \mathrm{d}l$
F may be piece-wise constant, linear or quadratic.

MT07CC : partly lumped mass-matrix, size 6 x 6.
FORMUL = 'MSLUMP' : MT07CC$_{ij}$ = $XMUL \int_\Omega (1-F)\Psi_i + F\Psi_i\Psi_j \mathrm{d}\Omega$
F is piece-wise constant.

MT08AC : size 3 x 6
FORMUL = 'MATFGR X' : MT08AC$_{ij}$ = $-XMUL \int_\Omega \Psi_j F\partial_x\Psi_i \mathrm{d}\Omega$
or 'MATFGR Y' : MT08AC$_{ij}$ = $-XMUL \int_\Omega \Psi_j F\partial_x\Psi_i \mathrm{d}\Omega$
Here Ψ_i is linear and Ψ_j is quadratic. F is linear or quadratic.

MT11AC : size 3 x 6.
FORMUL = 'MATGRF X' : MT11AC$_{ij}$ = $-XMUL \int_\Omega \Psi_j F\partial_x(F\times\Psi_i)\mathrm{d}\Omega$
or 'MATGRF Y' : MT11AC$_{ij}$ = $-XMUL \int_\Omega \Psi_j F\partial_y(F\times\Psi_i)\mathrm{d}\Omega$
Here Ψ_i is linear and Ψ_j is quadratic. F is linear or quadratic.

MT12AC : matrix used with the SUPG technique, size 3 x 6
FORMUL = 'MATUGH X' : MT12CC$_{ij}$ = $-XMUL \int_\Omega \Psi_j F\partial_x(F)(\vec{U}).(\vec{\nabla})\Psi_i \mathrm{d}\Omega$
or 'MATUGH Y' : MT12CC$_{ij}$ = $-XMUL \int_\Omega \Psi_j F\partial_y(F)(\vec{U}).(\vec{\nabla})\Psi_i \mathrm{d}\Omega$
Here Ψ_i is linear and Ψ_j is quadratic. The velocity fi□eld is piece-wise constant or quadratic.

MT13CA : gradient matrix, size 6 x 3
FORMUL = 'MATGRA X' : MT13CC$_{ij}$ = $XMUL \int_\Omega \Psi_i \partial_x \Psi_j d\Omega$
or 'MATUGH Y' : MT13CC$_{ij}$ = $XMUL \int_\Omega \Psi_i \partial_y \Psi_j d\Omega$
Here Ψ_i is quadratic and Ψ_j is linear.

MT13CC : gradient matrix, size 6 x 6.
FORMUL = 'MATGRA X' : MT13CC$_{ij}$ = $XMUL \int_\Omega \Psi_i \partial_x \Psi_j d\Omega$
or 'MATUGH Y' : MT13CC$_{ij}$ = $XMUL \int_\Omega \Psi_i \partial_y \Psi_j d\Omega$
Here Ψ_i is quadratic and Ψ_j is quadratic.

VC00CC : “volume” of basis functions, 6-component vector on every element.
FORMUL = 'MASBAS' : VC00CC$_i$ = $XMUL \int_\Omega \Psi_i d\Omega$

VC10OO : ux through boundaries, 2-component vector.
FORMUL = 'FLUBDF' : VC10OO$_i$ = $XMUL \int_\Omega \Psi_i F\vec{U}.\vec{n} d\Omega$
Here Ψ_i is quadratic. F is linear or quadratic, the velocity is linear or quadratic, given for all the mesh or only on the boundary.

VC13CC : gradient vector, 6-component.
FORMUL = 'GRADFX' : VC13CC$_i$ = $XMUL \int_\Omega \Psi_i \partial_x F d\Omega$
'GRADFY' : VC13CC$_i$ = $XMUL \int_\Omega \Psi_i \partial_y F d\Omega$
Ψ_i is quadratic, F linear or quadratic, and may be discontinuous between elements.

VC14AA : for turbulence in K - ϵ model, 3-component vector.
FORMUL = 'PRODFX' : VC14AA$_i$ = $XMUL \int_\Omega \Psi_i F(2(\partial_x U^2 + \partial_y V^2) + (\partial_x U + \partial_y V)^2 d\Omega$
'GRADFY' : VC13CC$_i$ = $XMUL \int_\Omega \Psi_i \partial_y F d\Omega$
Here Ψ_i is linear, the velocity may be linear or quadratic.

Bibliography
[1] FROEHLY A.: Rapport de Projet de Fin d'Etudes. 2008
[2] HERVOUET J.-M.: Hydrodynamics of free surface flows, modelling with the finite element method. Wiley & sons. 2007
[3] HERVOUET J.-M., PHAM C.-T.: Telemac version 5.7, release notes. Telemac-2D and Telemac-3D. 2007
[4] HERVOUET J.-M., RAZAFINDRAKOTO E., VILLARET C.: Telemac version 5.8, release notes. Telemac-2D, Telemac-3D and Sisyphe. 2008