Programme Area: Marine

Project: PerAWAT

Title: Tidal Array Scale Numerical Modelling: Level Set Technique Implementation within Code Saturne, Validation of the Combined Implementation (Flow Solver)

Abstract:

This report summarises the work undertaken to modify Code Saturne to incorporate a free surface model. It can be read as a standalone document without reference to WG3 WP2 D1 or WG3 WP2 D2 and summarises the theory behind, and work undertaken to develop the free surface model (outlined in D1 and D2) as well as the necessary modi cations to the Code Saturne kernel and a series of validation cases.

Context:

The Performance Assessment of Wave and Tidal Array Systems (PerAWaT) project, launched in October 2009 with £8m of ETI investment. The project delivered validated, commercial software tools capable of significantly reducing the levels of uncertainty associated with predicting the energy yield of major wave and tidal stream energy arrays. It also produced information that will help reduce commercial risk of future large scale wave and tidal array developments.

PerAWaT (MA 1003) Report WG3 WP2 D3

Tidal array scale numerical modelling: Level Set Technique Implementation within Code_Saturne, validation of the combined implementation (flow solver)

Participant lead on the deliverable: DM Ingram
Other participant: DA Olivieri

Institute for Energy Systems
School of Engineering
King's Buildings, Edinburgh EH9 3JL

THE UNIVERSITY
of EDINBURGH

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The PerAWaT project

The PerAWaT project is a research and development project funded by the Energy Technologies Institute (ETI) as part of its Marine programme. The ETI's Marine Energy Programme addresses key industry technology challenges by supporting the sea-trials of near-commercial marine energy systems, and through the development and demonstration of key technologies, systems and tools that support the acceleration of the industry. As part of this effort PerAWaT is producing tools capable of accurately estimating the energy yield of major wave and tidal stream energy converters operating in arrays.

This £8M project, coordinated by GL Garrad Hassan, also involves EDF Energy, E-ON, The University of Edinburgh, University of Oxford, Queen's University Belfast, and The University of Manchester as project partners.

During the project the team are developing numerical models of devices, interactions between devices in arrays and interactions between arrays at the coastal scale. These models are being validated, by the project team, using extensive scale model tank testing and full scale data from in-service devices where appropriate.

PerAWaT will provide an accurate assessment of the likely cost of energy production from large scale wave and tidal arrays reducing the uncertainty and risk faced by marine array developers, utilities and investors. It also aims to help facilitate the large scale deployment of marine energy arrays.

1.1.1 The tidal turbine array sub-project

The PerAWaT project is split into two sub-projects that deal with wave and tidal energy conversion respectively. The tidal sub-project (comprising work groups WG3 and WG4) is led by EDF R&D, while the wave sub-project (WG1 and WG2) is led by GL

Garrad Hassan. The two tidal work groups are further divided into a number of work packages.

WG3 which deals with numerical modelling of tidal turbines is described in more detail in the following section. WG4, in contrast, deals with experimental activities and comprises the following work packages.

- WP 1 – Tidal device scale physical experiments single horizontal axis device performance and wake analysis.
- WP 2 – Tidal array scale physical experiments up to 15 devices operating within an array.
- WP 3 – Tidal device scale physical experiments single open centre horizontal axis device performance and wake analysis.
- WP 4 – Tidal coastal basin physical experiment.
- WP 5 – Tidal array scale physical experiments at 1/10th scale.

WP1 to WP3 have been used provide insight and understanding of the formation of wakes behind single turbines and the interaction of wakes from farms of devices, they have also provided validation data for use in the numerical modelling WG. In the present WP, for example, data from WG4 WP1 and WG4 WP2 has been used to validate the numerical simulations of small arrays of turbines performed using blade element momentum theory (BEMT) actuator disk models [Ingram and Olivieri, 2012; Olivieri and Ingram, 2012].

1.1.2 WG3 – Numerical modelling of tidal turbines

Numerical modelling of tidal turbines may be performed at a number of different scales: Device, Farm and Basin (Figure 1.1.2). As the scale increases retaining the mesh size needed to resolve the boundary layers on each blade becomes impractical, so models must be introduced into the solver to represent the turbine. Within the WG3 the PerAWaT project is modelling the performance of typical turbines across all these scales. Each modelling scale is the focus of a different work package, with inter comparisons being made. The work packages are:

- WP1 – Blade resolved, 3D Reynolds averaged Navier-Stokes (RANS) modelling of a single turbine rotor in *Fluent*.
- WP2 – Blade modelled, turbine resolved 3D RANS modelling of small arrays of turbines, using *Code_Saturne*.
- WP3 – Turbine and Farm modelled simulations using the shallow water (depth averaged) form of the Navier-Stokes equations using *TELEMAC*.
- WP4 – Development of the GL Garrad Hassan *TidalFarmer* tool.

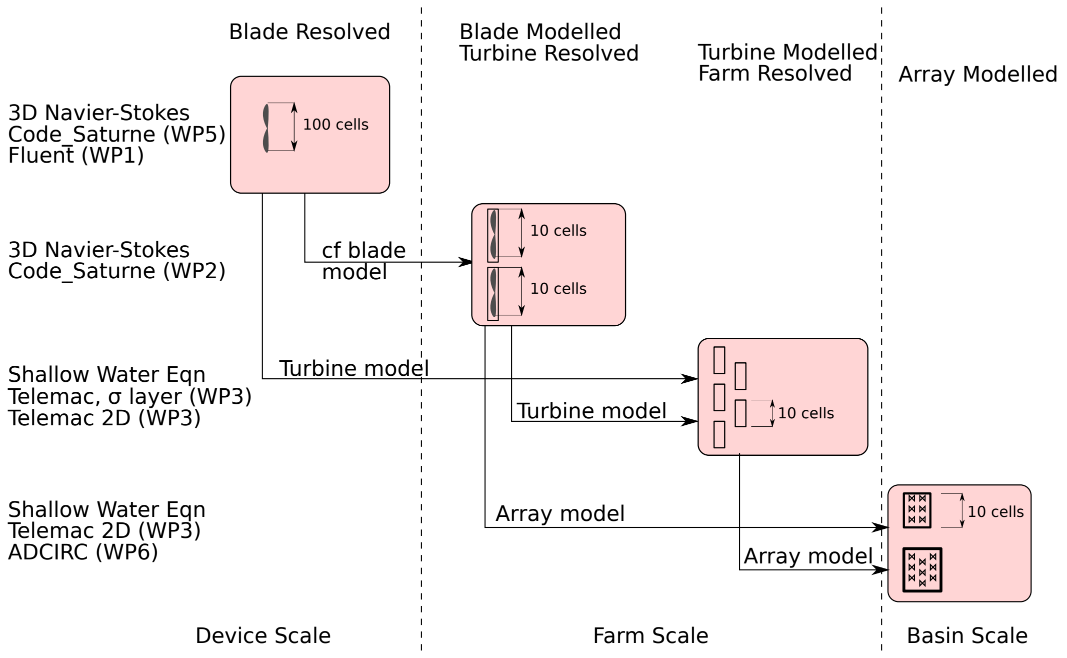


Figure 1.1: Interaction of the various scales of turbine model within WG3 of the PerAWaT project

- WP5 – Blade resolved, 3D Reynolds averaged Navier-Stokes (RANS) modelling of single turbine rotors in *Code_Saturne*.
- WP6 – Array modelled simulations using the shallow water (depth averaged) form of the Navier-Stokes equations using *ADCIRC*.

Whilst the input to WG3 WP2 comes primarily from the experimental work conducted in WG4 WP1 and WG4 WP2, results from the full-scale blade resolved RANS simulations (from WG3 WP5) have also been used. This experimental data has been used for verification and validation of the models developed in this deliverable.

1.1.3 WG3 WP2 – Array scale numerical modelling: Interactions within an array

WG3 WP2 of the PerAWaTproject seeks to make use of the existing, parallel, high performance, open source, computational fluid dynamics (CFD) code *Code_Saturne* (developed by EDF), extending the model to provide a mechanism for modelling the performance of a small array of marine current turbines at the meso-scale. The work package sits alongside WG3 WP5 “Device scale numerical modelling: Detailed CFD of other concepts”, which is also being executed by The University of Edinburgh using *Code_Saturne*.

The specific objectives within WG3WP2 are:

- Modelling the interaction of three dimensional unsteady flow and turbine wakes within an array;
- Verification of available numerical models,
- Implementation of an appropriate free surface model within the existing open source CDF code *Code_Saturne*, and
- Subsequent extension of the model to provide a mechanism for modelling the performance of a small array of marine current turbines at meso-scale.

Progress has required three major developments with the existing flow solver:

- The implementation of a free surface model which satisfies the zero-tangential shear boundary condition (Deliverables D1, D2 and D3);
- The implementation of an unsteady upstream boundary condition to model the inlet to a tidal channel (D4), and
- The development of a parameterised actuator disk model of a horizontal-axis marine current turbine (D5a, D5b, and D6).

1.2 Executive Summary

This report summarises the work undertaken to modify *Code_Saturne* to incorporate a free surface model. It can be read as a standalone document without reference to WG3 WP2 D1 or WG3 WP2 D2 and summarises the theory behind, and work undertaken to develop the free surface model (outlined in D1 and D2) as well as the necessary modifications to the *Code_Saturne* kernel and a series of validation cases.

Code_Saturne, has been under development by EDF since 1997 [Archambeau et al., 2004]. The solver uses a collocated Finite Volume Method (FVM) approach that allows three-dimensional meshes built from tetrahedral, hexahedral, prismatic, pyramidal, or polyhedral cells and also admits any type of grid structure (unstructured, block structured, or hybrid). This flexibility allows *Code_Saturne* to model highly complex geometries. It is capable of simulating both incompressible and compressible flows, with or without heat transfer and turbulence.

From the outset, *Code_Saturne* was designed as a parallel code and has been shown to operate efficiently on massively parallel supercomputers [Shang et al., 2011]. *Code_Saturne* works as follows: the pre-processor reads the mesh file and partitions the mesh using either the Metis [Karypis and Kumar, 1998] or Scotch [Chevalier and Pellegrini, 2008] libraries to produce the input files for the solver. The solver is then executed using the options specified in the input files and a log file containing diagnostic information is produced. Once the simulation is complete, the output is post-processed and converted into readable files for use by different visualisation packages (such as ParaView[1] [Squillacote, 2008] and VisIT[2] [LNLL, 2005]). Parallel code coupling capabilities are provided by EDFs FVM library. Since 2007, *Code_Saturne* has been open-source and is available to any user (www.code_saturne.org). *Code_Saturne* provides a simple graphical user interface for specifying problems and boundary conditions, but external mesh generation software must be used to generate the grid files. In the present work the ANSYS ICEM CFD[3] mesh generation software has been used.

One significant advantage of *Code_Saturne* is its industrial pedigree: the code was originally designed for industrial applications and research activities in several fields related to energy production. The production versions of the solver (currently release 3.0.0) is certified for use in the design of nuclear power stations. The development versions (currently release 3.1.0) is updated approximately every six months and includes updates to the solver and new modules.

The free-surface model used is based on the *Level Set* formulation [Sussman et al., 1994; Sethian, 1999] together with a mass-preserving redistancing operator [Ausas et al., 2011]. Level set methods have been used, as an alternative to the widely used Volume of Fluid (VoF) method [Hirt and Nichols, 1981], because they provide superior performance when modelling the advection of free surface waves [Maguire and Ingram, 2010] and allow a straight forward implementation of the free surface boundary conditions [Watanabe et al., 2008]. This provides a fully three-dimensional Navier-Stokes equation based model which is applicable to problems involving more complex wave kinematics than the depth averaged approximations used in models such as TELEMAC [Moulinec et al., 2011].

Chapter 2 describes the level-set approach and contrasts it with both the VoF and free-surface-capturing methods. This chapter also describes the implementation of the underlying advection scheme for the level-set function in *Code_Saturne* and the implementation in *Code_Saturne* of the Ausas et al. [2011] mass-preserving redistancing operator.

Because *Code_Saturne* is a an incompressible Navier-Stokes solver the implementation has required changes to be made to the internal (kernel) routines of the solver to allow the fluid density to vary across the computational domain. These changes have had to be made to stabilise the Rhie and Chow [1983] interpolation scheme and to ensure diagonal dominance in the matrix solvers of the pressure correction scheme. The work to identify and remediate theses issues was conducted by RENUDA under a subcontract from the University of Edinburgh and is described in Chapter 3. it is important to note that without the open-source nature of *Code_Saturne* it would have been impossible to make the necessary changes to the kernel routines to implement the level-set method.

Chapter 4 presents results for the following validation cases:

[1]http://www.paraview.org/
[2]https://wci.lnll.gov/codes/visit
[3]http://www.ansys.com/Products/Other+Products/ANSYS+ICEM+CFD

1. Low amplitude sloshing,
2. Subcritical flow over a submerged bump,
3. Transcritical flow over a submerged bump (without a hydraulic jump),
4. Transcritical flow over a submerged bump (with a hydraulic jump), and
5. Flow over a submerged hydrofoil.

Case 1 is compared with a semi-analytical solution. Cases 2–4 have been compared with an analytical solution for the depth averaged equations, which provides a good approximation of the real 3D flows. Finally, case 5, is compared with experimental results obtained by Duncan [1983] and to numerical simulations performed using CF/X by Gretton et al. [2010].

The validation cases show that *Code_Saturne* performs reasonably and that the code is stable when applied to these challenging problems. The performance on Cases 1 and 5 is similar to that from other published CFD simulations, though there is some mass loss in case 1 which is attributed to *Code_Saturne* pressure solver not having inflow and outflow boundaries to use to control mass in the domain. Cases 2 and 3 show good predictions of the location of the depression which forms over the bump, though could benefit from a more detailed mesh refinement study. The results for case 4 are much more mixed and a rather confused by wave breaking at the hydraulic jump, this case does demonstrate that the solver copes with complex wave breaking which is know to destabilise many flow solvers.

In concluding it is clear that whilst a level-set method has been successfully implemented within *Code_Saturne* further work is needed to determine the level of mesh refinement needed (in the location of the free surface) for high quality simulations and to explore the impact on simulations of implementing more accurate free-surface boundary conditions on the water surface. Further benefit could be obtained from implementing a seaward boundary which admits both regular and irregular waves to the domain (c.f. Ingram et al. [2009]) and to combine this with the turbine models developed in Ingram and Olivieri [2012] and Olivieri and Ingram [2012] to allow simulations of wave interactions with turbines to be performed.

1.2.1 Acceptance criteria

The acceptance criteria for this report (WG3 WP2 D3) listed in the technology contract are as follows:

- Validation methodology, including input requirements and interpretation of results,

  The validation cases, their input requirements and the interpretation of their results are discussed in Chapter 4. *Code_Saturne* has been applied to five test cases (low amplitude sloshing, subcritical flow over a bump, transcritical flow over a bump with and without a hydraulic jump, and flow over a submerged hydrofoil) for which either analytical or experimental results are available. These test cases deal both with basic free surface flow cases and with cases in which submerged structures cause changes in the free surface elevation. Such changes in elevation are often small, but could also be created by support structures and flow acceleration devices installed as part of a tidal energy project.

- Validation 'site' characteristics including boundary conditions,
- Results of validation exercise – how accurately did the additional modules/*Code_Saturne* predict/replicate the validation site conditions.

  Chapter 4 describes the upstream and downstream boundary conditions needed for free surface flows in open channels and shows that the upstream flow conditions are accurately reproduced by the flow solver. Boundary conditions are described for the level set function and the mass flow based on an analogy with the hyperbolic shallow water equations. These boundary conditions can be extended to admit waves at the upstream boundary (see Ingram et al. [2005]).

- Current limitations and applications, and areas for improvement within the modules, and

  Chapters 2 and 3 describe the theoretical background to and implementation of the level-set solver within *Code_Saturne*. The limitations of the approach are described, further areas for improvement are described at the end of Chapter 4.

- Operation of modules within *Code_Saturne*.

  The operation of the modules within *Code_Saturne* are described in Chapter 2 with the changes made to the kernel routines of *Code_Saturne* needed to allow a variable density soliton being described in Chapter 3. The source code is provided on the accompanying FTP site and can also be downloaded from the PerAWaT project archive.

Chapter 2

Methods for including Free Surfaces in Navier-Stokes Simulations

2.1 Introduction

Interface tracking methods have been implemented in CFD since the late 1950's [Harlow, 1957]. In their infancy, computer power and memory was limited such that only a single liquid phase could be simulated in 2D (the gas phase, air, was neglected in the simulation). Today, due to the increase in computer power and the volume of research, it is possible to simulate free surface flow with turbulence and surface tension effects in 3D Saruwatari et al. [2009] for short periods of time.

Approaches based on depth averaging the Navier-Stokes equations (to produce either the shallow water or Bousinesq equations [Peregrine, 1972]) have been widely used at geographic scale, but are not appropriate for detailed wave kinematics. Realistic simulations of ocean waves require methods which are able accurately to simulate a range of processes including the propagation, shoaling, breaking and possible overturning of waves prior to their possible impact on exposed structures. Even three dimensional depth averaged formulations cannot reproduce all these phenomena so the only recourse is to solve the three dimensional Navier-Stokes equations and include some model of the free surface.

It is a further requirement that the simulation continues after wave breaking, or impact, modelling the formation of splashing jets and their recombination with the main body of water. Such simulations are extremely challenging requiring both accurate resolution of the free-surface (with length scales of tenths of a meter) and computationally efficient resolution of waves with length scales of tens of meters. Many attempts to compute both free surface and interface flows have been undertaken, [see the review by Scardovelli and

Zaleski, 1999]. Three approaches to free-surface Navier-Stokes solvers, which have been applied to ocean waves are:

- The Volume of fluid (VoF) method,
- The free surface capturing method, and
- The level set method.

The first two methods will be briefly described in the following sections, while the remainder of this chapter will describe the Level Set approach in detail.

2.1.1 Volume of Fluid Method

The Volume of fluid (VoF) method has been commonly used for the free-surface computations in a fixed Cartesian grid system [Hirt and Nichols, 1981], however, a high-order scheme needs to be introduced for reconstructing highly curved surface. The VoF remains one of the most popular schemes used for free surface flows (The University of Oxford have used it within WG3 WP1) and it has an established track record in a variety of applications [Youngs, 1982; Hirt and Nichols, 1981; Lafaurie et al., 1994; Ubbink and Issa, 1999; Troch et al., 2003; Li et al., 2004]. The VoF method is intrinsically mass conservative.

In the VoF scheme, the location of the free surface is computed by tracking the evolution of the volume fractions (denoted by α) in all computational cells. At each time step the location of the free surface needs to be determined from the distribution of α. The most popular approach to interface reconstruction, PLIC (Piecewise Linear Interface Calculation), is based on the idea, that the interface can be represented as a line in two dimensions, or a plane in three. Using VoF with PLIC is the current standard approach and is used number of computer codes, such as *ANSYS Fluent* and *STAR-CCM*. Maguire and Ingram [2010], however, found that the reconstruction algorithms used in commercial implementations of VoF schemes lead to damping and dissipation when waves are propagated for more than 2 or 3 wave lengths (see Figure 2.1). This dissipation is caused by the free-surface reconstruction algorithm which is designed to have a smoothing effect on the free surface to prevent the formation of non-physical oscillations. The net effect is similar to that observed in artificial viscosity schemes for shock wave propagation where first order schemes cause wave fronts can loose amplitude and definition whilst prohibiting the formation of Gibbs oscillations.

Considering ANSYS CFX, Maguire [2011] states

> *"One persistent observation noticed in all meshes, regardless of time step, was that of wave height decay down the flume. The wave height attenuation is more severe the coarser the mesh, or larger the time step, however, it is present in all meshes and time steps to varying degrees.*
>
> *Recognising that the use of linear theory and its application to second order*

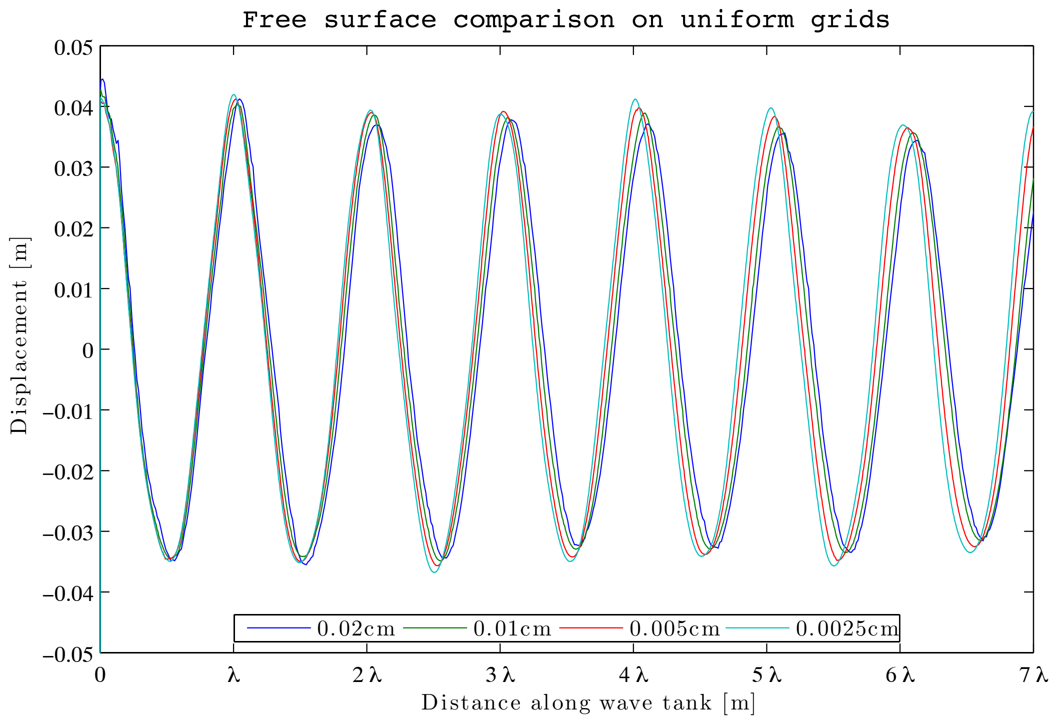


Figure 2.1: Amplitude loss during the propagation of a small amplitude wave on a regular Cartesian grid using a VoF scheme computed by Maguire [2011]. Four different computational meshes are used with $\Delta x = 0.02,\ 0.01,\ 0.005,$ and 0.0025cm.

> *waves will result in smaller wave heights and acknowledging that wave heights in physical flumes also decay, the wave height attenuation noticed here is still excessive and while not as severe as reported by Bhinder et al. [2009] (who reported CFX to be totally unusable), wave height decay is still persistent."*

Maguire's conclusions are based on an exhaustive mesh refinement study in which the finest grid computations required more than 48 hours to run, making high quality 3D simulations of propagating waves impractical. He reports similar findings are reported for the specialist commercial VoF code FLOW3D, but with much reduced computation times.

2.1.2 Free surface capturing method

The free surface capturing method [Kelecy and Pletcher, 1997] explicitly models an immiscible two-fluid system. The model is formulated as a set of partial differential equations which govern the motion of an inviscid, incompressible, variable density fluid. These equations consist of a mass conservation (density) equation (which is mathematically equivalent to the volume fraction transport equation), momentum equation and an incompressibility constraint that are solved simultaneously using the finite volume method. The formulation is based on the artificial compressibility method [Chorin, 1967; Beddhu et al., 1994] in which the pressure, density and velocity fields are directly coupled to produce a hyperbolic system of equations. Those equations are then solved using Godunov-type schemes [Qian et al., 2003, 2006].

Whilst free surface capturing methods have been successfully applied to a range of demanding problems [Gao et al., 2007; Ingram et al., 2009] they suffer from highly restrictive time steps and are extremely computationally demanding.

2.2 Level-set method

The level set method is also an Eulerian approach for tracking interfaces and shapes. As with the VoF and Free Surface Capturing methods, it has the advantage that numerical computations involving curves and surfaces can be performed on a fixed Cartesian grid without having to parameterise these shapes. A further advantage of the level set method is that it makes it very easy to follow shapes that change topology, for example during wave breaking, or when fluid droplets breakup and coalesce. The level set method was developed by Osher and Sethian [1998] and is described in detail for a number of applications in the book by Sethian [1999].

The principals of the level-set method together with the Hamiltonian equation based reinitialisation method will be outlined below, Section 2.3 presents the geometric reinitialisation approach which has been implemented in *Code_Saturne* under WG3 WP2. Geometric reinitialisation has been used to address two problems associated with the original level-set method:

1. Hamiltonian equation based method are global and must be applied to every grid point in the solution domain, making reinitialisation computationally expensive, and
2. Hamiltonian equation based methods do not guarantee mass conservation.

For simplicity of exposition we shall consider the level-set method applied to a two dimensional space, it is important to note that the method applies to spaces of arbitrary dimension and extends naturally to three-dimensional problems. Given an auxiliary function $\phi(x, y)$, we can define the level set function Γ as the zero level set of ϕ, i.e.

$$\Gamma = \{(x, y) \mid \phi(x, y) = 0\}\,. \tag{2.1}$$

This allows Γ to be manipulated implicitly by modifying ϕ. By convention we assume that ϕ is positive inside the region delimited by Γ and negative outside [Osher and Fedkiw, 2003]. Figure 2.2 shows how the level-set, $\Gamma(x, y)$, changes with modification to the auxiliary function, $\phi(x, y)$. Since $\boldsymbol{\nabla} \cdot \phi$ is perpendicular Γ, the surface unit

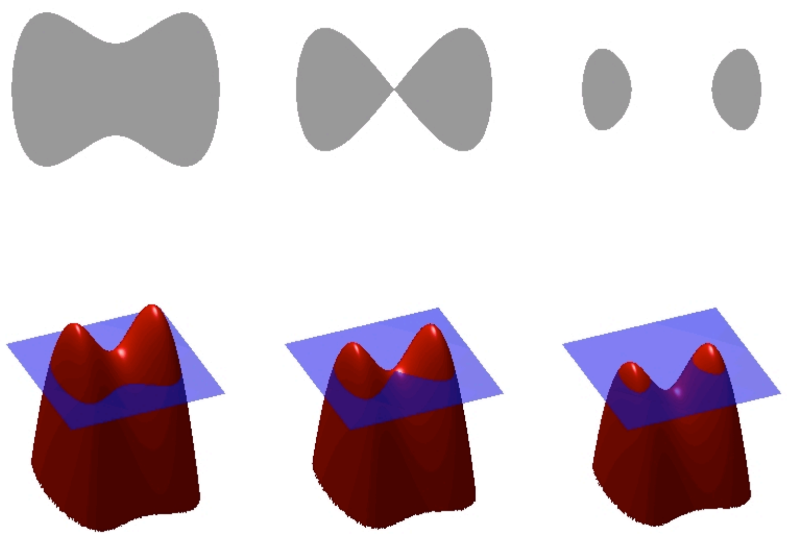


Figure 2.2: An Illustration of the level set method showing the auxiliary function, $\phi(x, y)$, in red, with the zero level in blue and the level set(s), $\Gamma(x, y)$, above in grey. From an original plot by Oleg Alexandrov

normal vector $\boldsymbol{n}$ and curvature κ can simply be computed [Watanabe et al., 2008] as follows:

$$\boldsymbol{n} = \frac{\boldsymbol{\nabla}\phi}{|\boldsymbol{\nabla}\phi|} \tag{2.2}$$

and

$$\kappa = \boldsymbol{\nabla} \cdot \boldsymbol{n}. \tag{2.3}$$

If the level set, Γ, moves in the normal direction with speed, v, then it can be shown that the level set function, ϕ, satisfies

$$\frac{\partial \phi}{\partial t} = v|\boldsymbol{\nabla}\phi|. \tag{2.4}$$

This is a partial differential equation of the Hamilton-Jacobi type and it's solution forms the basis of many level set method implementations. In the case of fluid dynamic simulations a more complex advection equation must be solved for ϕ as it is advocated with the fluid velocity, $\boldsymbol{u}$. In this case,

$$\frac{\partial \phi}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla}\phi = 0, \tag{2.5}$$

must be solved. This equation is similar to the advection equation used in VoF methods and can easily be solved my most general purpose fluid dynamics solvers as an additional scalar advection equation. Solving (2.5) numerically can lead to dissipation and dispersion errors in the solution. While dissipative errors damp out high frequency waves in the solution, dispersive errors leads to changes in the shape of ϕ (in particular smearing of sharp fronts). Figure 2.3 shows a the effect of the dispersive and dissipative errors present in the CIP0 scheme [Edwards, 2011], similar effects may be observed with any numerical advection scheme. To ensure the location of the interface remains well de-

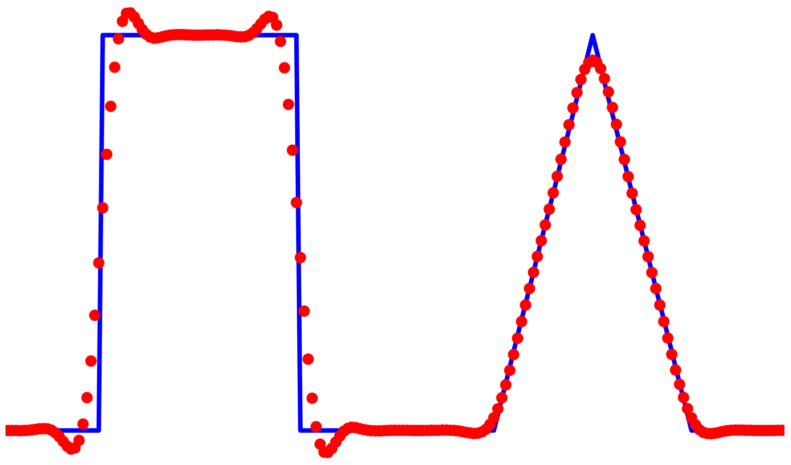


Figure 2.3: Solution to the periodic linear advection problem with initial conditions including top-hat and bi-linear functions after 1000 time steps, computed using the CIP0 scheme by Edwards [2011]

.

fined the function, $\phi(x, y, z)$, is specified as a signed distance function whos magnitude represents its distance from an iso-surface, in 3D, or iso-contour, in 2D. The sign of the function is determined by the side of the level set, Γ, the point (x, y, z) lies. Advection can causes ϕ to lose it's signed distance property, with the gradient in the neighbourhood of Γ shifts away from ± 1. In areas of extremely high or low gradient the accuracy to which the level set function can track the interface position is reduced. Edwards [2011] reports that in simple bubble advection tests a mass loss of up to 41% can be observed.

The solution to the loss of the signed distance property was first suggested by Chopp [1993]. A level set method was used to study minimal surfaces and it was found that a loss of the distance property of the level set function occurred when boundaries were applied to a level set curvature driven problem. The solution proposed was to re-distance or reinitialise the level set function at regular time intervals during the simulation. The simulation was paused to calculate the level set, Γ, and then calculating the distance to it from each grid point. The reinitialisation idea was extended further in an implementation for incompressible two phase flow by Sussman et al. [1994] and later Chang et al. [1996]. The method by Sussman et al. [1994] introduced a more sophisticated approach which did not require the zero level set to be found explicitly. Sussman et al. [1994] resistances the level set function, ϕ, by iteratively applying

$$\phi_t = S(\phi_0)\left(1 - |\boldsymbol{\nabla}\phi|\right), \tag{2.6}$$

with the initial condition of

$$\phi(\boldsymbol{x}, 0) = \phi_0(\boldsymbol{x})$$

where ϕ_0 is the initial value of the level set function and $S(\phi_0)$ is the smoothed sign function,

$$S_\epsilon(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + \epsilon^2}}.$$

To solve (2.6), it is first written as a hyperbolic equation,

$$\phi_t + \boldsymbol{w} \cdot \boldsymbol{\nabla}\phi = S(\phi), \tag{2.7}$$

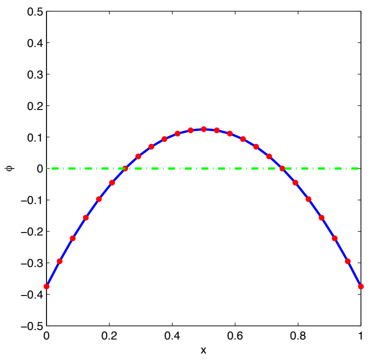
where

$$\boldsymbol{w} = S(\phi_0)\left(\frac{\nabla\phi}{|\nabla\phi|}\right).$$

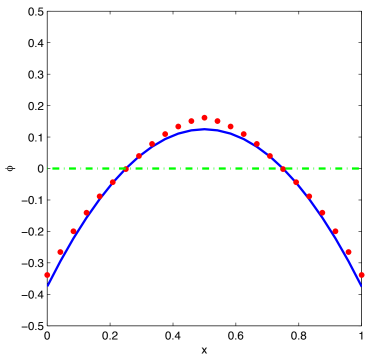
This hyperbolic equation has a characteristic, $\boldsymbol{w}$, which is an outward pointing unit normal of Γ. This equation is solved in a psudo-time until it converges, in-between the real times steps of the advection algorithm. Figure 2.4 shows the effect of up to 10 pseudo time steps of the reinitialisation algorithm from an initially quadratic level-set function. The effect of combining this with an advection scheme is shown in Figure 2.5, in this test case reinitialisation is performed once, after each real time-step, for a single iteration with a pseudo time-step, $\Delta\tau = \frac{1}{2}\Delta x$ [Edwards, 2011]. It is clear that the form of the saw tooth wave is preserved throughout the advection process and that after 16 cycles the shape of the initial wave form is exactly preserved. The capability of level set methods to resolve complex flow phenomena is illustrated by the results shown in Figure 2.6. Saruwatari et al. [2009] used a level set method to examine the formation of finger jets during the latter stages of wave breaking. She performed numerical simulations using a large eddy simulation (LES) approach within a CIP-type advection solver, the level set reinitialisation algorithm used is that described above. The solver also incorporates a source term treatment for the surface-tension forces [Watanabe et al., 2008] which makes use of the level-set function ϕ to compute the curvature of the free surface using equation (2.3).

2.3 Geometric Reinitialisation

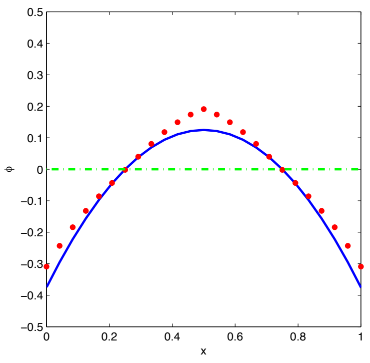
There are two principal disadvantages to the level set method outlined in the previous section, firstly the reinitialisation is global and requires the solution of an additional



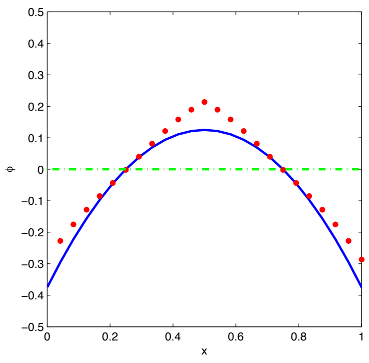
(a) 0 Steps



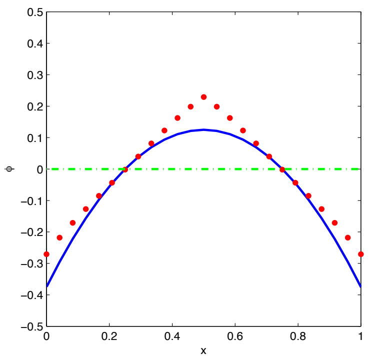
(b) 2 Steps



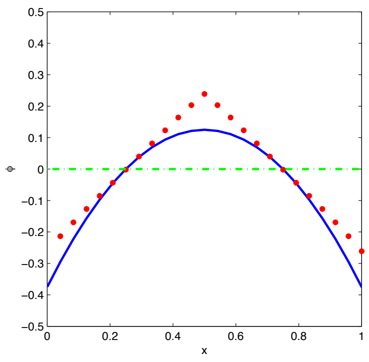
(c) 4 Steps



(d) 6 Steps

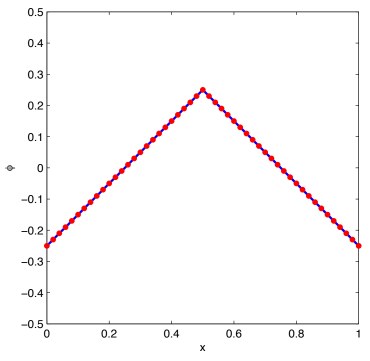


(e) 8 Steps

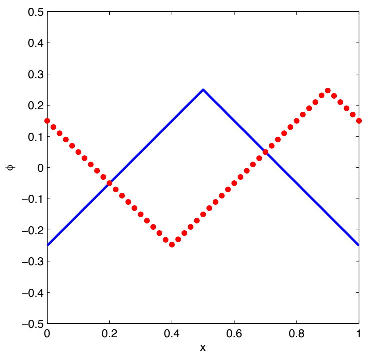


(f) 10 Steps

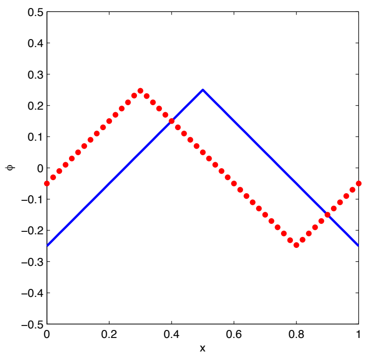
Figure 2.4: Reinitialisation of $\phi_0(x) = -2(x - \frac{1}{4})(x - \frac{3}{4})$, $\Delta t = \frac{1}{2}\Delta x$, on a 25 point grid, using the Sussman algorithm, reproduced from Edwards [2011]



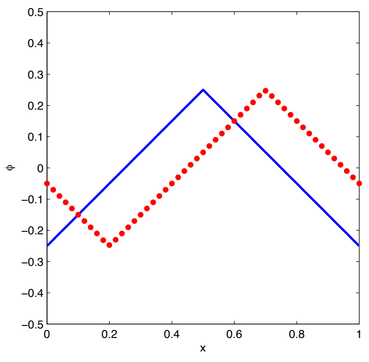
(a) 0% of simulation



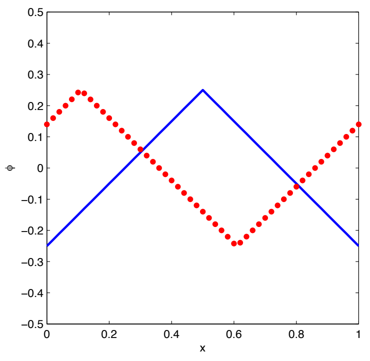
(b) 20% of simulation



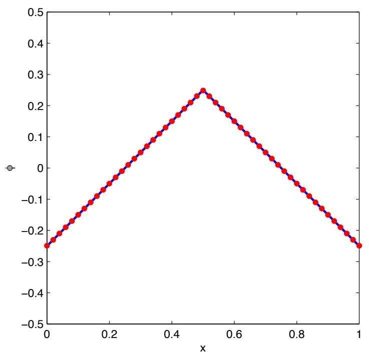
(c) 40% of simulation



(d) 60% of simulation



(e) 80% of simulation



(f) 100% of simulation

Figure 2.5: Advection of a periodic saw-tooth wave using the CIP0 scheme with Sussman reinitialisation. The function is advected for 16 cycles with one iteration of the reinitialisation algorithm at each time step, reproduced from Edwards [2011]

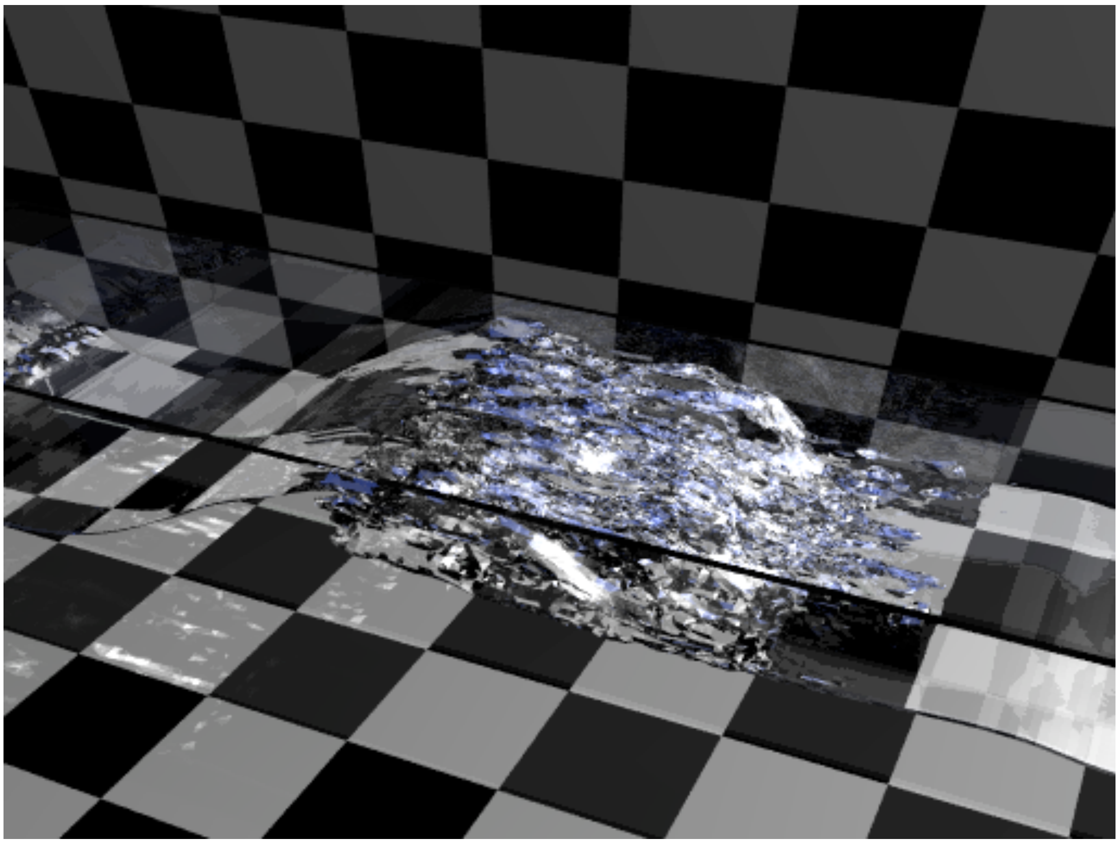


Figure 2.6: Ray traced image showing the formation of finger jets during the splashing phase of wave breaking, computed using the level-set method within a Cartesian grid CIP scheme by Saruwatari et al. [2009].

PDE on all mesh cells and secondly the method is not guaranteed to conserve mass for incompressible flow problems.

In large scale three dimensional computations the application of the reinitialisation algorithm to the whole computational domain represents a significant computational overhead. This can be addressed by using a banded approach where the PDE is only solved in the neighbourhood of the free surface [Peng et al., 1999], resulting in a significant increase in efficiency. This efficiency is possible because as one moves away from the interface (in regions where $|\phi| >> 1$) there is no advantage to accurately calculating the level set function and so this can be left unresolved. Although this improves the efficiency of the algorithm it fails to address the mass conservation issues. Errors in mass conservation can lead to unphysical motions of the interface and can severely affect the stability of the results. To counter this improvements have been suggested to PDE based reinitialisation algorithms [Edwards, 2011], and level set methods have been hybridised with volume of fluid solvers [Sussman and Puckett, 2000] or Lagrangian particle advection schemes [Losasso et al., 2008]. An interesting alternative, proposed by Mut et al. [2006] and Ausas et al. [2011], uses a geometric scheme to re-distance ϕ which guarantees mass conservation.

Because the geometric algorithm is easily implemented on unstructured grids, can be extended to three spatial dimensions, and is implicitly mass conserving, it has been implemented within *Code_Saturne*. The remainder of this section will discuss the algorithm.

2.3.1 Conservation of "mass" in level set schemes.

Mut et al. [2006] note that the level set advection equation (2.5) is in the form of a conservation law for ϕ. Assuming that the level set, Γ, represents the interface between two heterogeneous fluids (A and B) in a arbitrary region Ω, the region occupied by fluid A is

$$\Omega_A = \{\boldsymbol{x} \in \Omega,\, \phi(\boldsymbol{x}) > 0\}\,. \tag{2.8}$$

Provided the velocity field satisfies $\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0$ in Ω_A, then the volume of fluid A (denoted by $|\Omega_A|$) will be conserved since,

$$\frac{d|\Omega_A|}{dt} = \int_s \boldsymbol{u} \cdot \boldsymbol{n}\, d\Gamma = \oint_{\Omega_A} \boldsymbol{\nabla} \cdot \boldsymbol{u}\, d\Omega = 0 \tag{2.9}$$

Conservative advection schemes, however, applied to (2.5) do not result in mass conservation of A because they have been designed to conserve the mass of ϕ as if it were some function of fluid density. Such schemes conserve $\int_\omega \phi\, d\omega$ over the control volumes, ω, that form subdomains of Ω. This conservation property is useless in level set schemes since conservation of ϕ does not imply that Γ will move at the correct speed. Errors in the propagation speed of Γ lead to the mass of fluid A being created or destroyed by volume changes in Ω_A. If mass is to be conserved then the net volume change in $\Delta|\Omega_A| = 0$

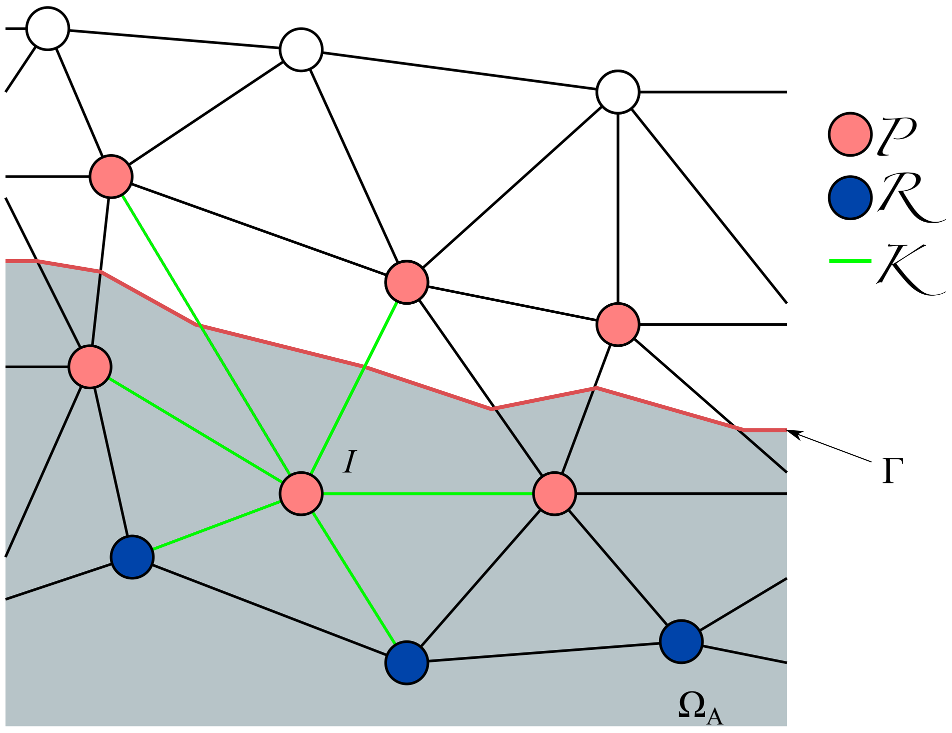


Figure 2.7: Definition sketch for the Mut et al. [2006] redistancing algorithm: The node $\mathcal{I}$ located inside the fluid region Ω_A, has a set, $\mathcal{K}$ of simpicies. The interface between the fluid regions is defined by the level set, Γ. The set of nodes adjacent to Γ in either fluid is denoted by $\mathcal{P}$ and the nodes which are inside Ω_A but not adjacent to the interface are denoted by $\mathcal{R}$.

between each time step. By developing a reinitialisation that algorithm approximates ϕ by $\widetilde{\phi}$ it is possible to ensure that this condition is satisfied.

We begin by defining $\mathcal{P}$ as the set of nodal points that are adjacent to Γ, in the sense that they are vertices of simplices inside which ϕ changes sign. $\mathcal{P}_A$ is the subset of points for which $\phi > 0$, i.e. $\mathcal{P}_A = \mathcal{P} \cap \Omega_A$. $\mathcal{R}_A$ is defined as the node points which lie inside Ω_A but are not included in $\mathcal{P}_A$. The definitions of these terms are shown in Figure 2.7.

The steps of the algorithm are as follows:

1. **Initialisation** — For a node $\mathcal{I} \in \mathcal{P}$, we define $\mathcal{C}_{\mathcal{I}}$ as the set of nodes connected to $\mathcal{I}$, i.e. $\mathcal{C}_{\mathcal{I}} \subset (\mathcal{P}_A \cup \mathcal{R}_A)$. The initial guess for $\widetilde{\phi}^0$ is simply based on the distance along the edges, and the value of the distance function at each neighbouring node:

$$\widetilde{\phi}^0 (\boldsymbol{X}_{\mathcal{I}}) = \min_{J \in \mathcal{C}_{\mathcal{I}}} \left[\phi (\boldsymbol{X}_J) + |\boldsymbol{X}_{\mathcal{I}} - \boldsymbol{X}_J| \right].$$

2. **Simplex-wise correction** — The objective is to compute $\widetilde{\phi}$ so that it approximates the signed distance, d, while at the same time preserving the volume,

$$V_K(\phi) = \int_{\mathcal{K}(\phi)} H(\phi(x))\,\mathrm{d}x,$$

where $H(.)$ is the heaviside function. In general $V_K(\widetilde{\phi}) \neq V_K(\phi)$, though the difference will be quite small. The necessary correction, Δ_K can be computed by solving

$$R_K(\Delta_K) = V_K\left(\widetilde{\phi^0} + \Delta_K\right) - V_K(\phi) = 0, \tag{2.10}$$

Δ_K can be found using a simple secant iteration,

$$\Delta_K^{n+1} = \Delta_K^n - R_K^n \frac{\Delta_K^n - \Delta_K^{n-1}}{R_K^n - R_K^{n-1}}.$$

3. **Node-wise correction** — Once the simplex-wise corrections have been computed they need to be applied to the nodal values of $\widetilde{\phi^0}$ to preserve the volume. This is done by averaging the corrections over all the simplicies that share a node. Let $\mathcal{I}$ be a node in $\mathcal{P}$ which has a set of $\mathcal{K}$ simplicies. We compute the correction

$$\psi\left(\boldsymbol{X}_{\mathcal{I}}\right) = \frac{1}{N_{\mathcal{I}}} \sum_{k\in\mathcal{K}} \Delta_k, \tag{2.11}$$

where $N_{\mathcal{I}}$ is the number of simplicies shared by node $\mathcal{I}$. Once this has been computed $\widetilde{\phi^0}$ can be updated using

$$\widetilde{\phi} = \widetilde{\phi^0} + C\psi,$$

where C is a weight which satisfies

$$\int_{\mathcal{K}} H\left[\widetilde{\phi^0}(x) + C\psi(x)\right]\,\mathrm{d}x = \int_{\mathcal{K}} H\left[\phi(x)\right]\,\mathrm{d}x. \tag{2.12}$$

As in the previous step (2.12) can be solved with a few iterations of the secant method.

Ausas et al. [2011] provides a more detailed version of the above algorithm which includes more information on the implementation. It is that paper which forms the basis of the geometric mass-preserving reinitialisation algorithm implemented in *Code_Saturne*.

2.3.2 Implementation in *Code_Saturne*

Implementation of the Mut et al. [2006] and Ausas et al. [2011] algorithm in *Code_Saturne* requires a number of changes to be made to the code.

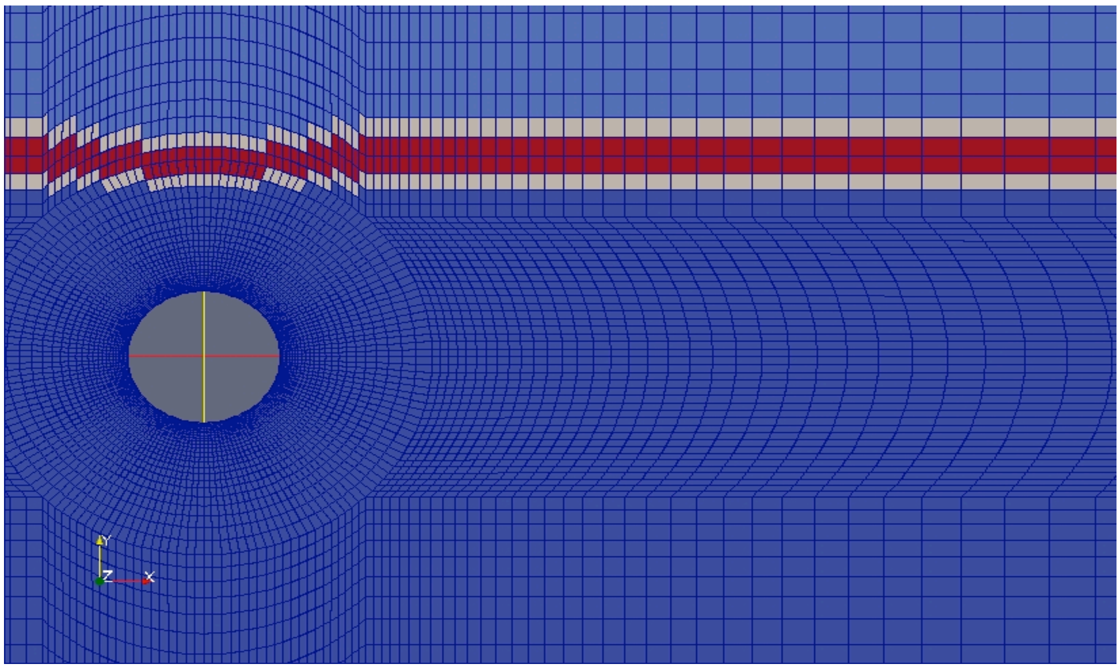


Figure 2.8: Band structure computed by the mesh connectivity algorithm: The dark red cells form the set of primary cells (i.e. cells adjacent to Γ) while the light red cells are the immediately adjacent cells, all other cells are coloured blue.

Firstly some connectivity information must be created, as *Code_Saturne* throws this information away during the pre-processor stage when the grid is read into the flow solver. Internally *Code_Saturne* represents the grid through a linked list of edges and control volumes [EDF R&D, 2013b,a], this means there is no explicit connectivity information for grid cells. It has proved necessary to write a Fortran-95 module which constructs the required connectivity information from the internal *Code_Saturne* data structures, the module is described in Olivieri and Ingram [2010]. The results of applying the algorithm to a simple test problem involving flow over a cylinder is shown in Figure 2.8.

Tables 2.1 and 2.2 show the six steps of the Ausas et al. [2011] reinitialisation algorithm. In the nomenclature used by Ausas et al. [2011] the location of the free surface, Γ, is represented by a peicewise linear approximation S_h. Ausas et al. [2011] seeks to define a correction factor, ψ, such that $\phi = \phi^* + \psi$, where $\phi^* = \widetilde{\phi}$. This form of the Mut et al. [2006] algorithm has been implemented in *Code_Saturne* and the modifications to the user modules `usiniv` and `usproj` will be described in the following sections.

In addition to coding the reinitialisation and connectivity algorithms in *Code_Saturne*, changes have had to be made to the kernel routines in the solver to allow the density to vary without compromising the pressure correction algorithm. Chapter 3 describes this work which was conducted by RENUDA UK Ltd under a subcontract from the

Table 2.1: Steps 1–4 of the Ausas et al. [2011] reinitialisation algorithm for the first neighbours of $\mathcal{L}_h$: computation of the exact distance followed by the mass correction.

Step 1: Compute the exact distance to $\mathscr{S}_h$ (Brute force)

```
• Set $\tilde{d}(\vec{X}_n)=+\infty$ for $n=1,2,\ldots,N^{\mathscr{P}}_{\text{nod}}$
do $(K=1,N^{\mathscr{K}}_{\text{el}})$
  • Find $\mathscr{S}_K$, the reconstruction of $\mathscr{S}_h$ in $K$ using $\phi_h$
  do $(I=1,N^{\mathscr{P}}_{\text{nod}})$
    • Compute $d_I$ s.t. $d_I=\min_{\vec{x}\in\mathscr{S}_K}|\vec{X}_I-\vec{x}|$
    • Set $\tilde{d}(\vec{X}_I)=d_I$ if $(\tilde{d}(\vec{X}_I)>d_I)$
  end do
end do
• Set $\phi^*_h(\vec{X}_n)=\tilde{d}(\vec{X}_n)$ for $n=1,2,\ldots,N^{\mathscr{P}}_{\text{nod}}$
```

Step 2: Find η_h, a piecewise constant function

```
do $(K=1,N^{\mathscr{K}}_{\text{el}})$
  • Set $\delta V_K=\Delta V_K(\phi_h,\phi^*_h)$
  do while $(|\delta V_K|>10^{-15})$
    • Find $\mathscr{S}_K$, the reconstruction of $\mathscr{S}_h$ in $K$ using $\phi^*_h+\eta_K$
    • Set $\eta_K=-\delta V_K/\text{size}(\mathscr{S}_K)$
    • Set $\delta V_K=\Delta V_K(\phi_h,\phi^*_h+\eta_K)$
  end do while
end do
```

Step 3: Find ξ_h, the orthogonal projection of η_h

```
do $(I=1,N^{\mathscr{P}}_{\text{nod}})$
  • Set $\xi_h(\vec{X}_I)=0$
  do $(K=1,N_I)$
    • Set $\xi_h(\vec{X}_I)\leftarrow\xi_h(\vec{X}_I)+\eta_K/N_I$
  end do
end do
```

Step 4: Find $\psi_h=\phi^*_h+C\xi_h$

```
• Initialize $\delta V^{(i)},C^{(i)}$ for $i=1,2$
• Set $i=3$
do while $(|\delta V^{(i)}|>10^{-15})$
  • Set $m^{(i)}=(C^{(i-1)}-C^{(i-2)})/(\delta V^{(i-1)}-\delta V^{(i-2)})$
  • Set $C^{(i)}=C^{(i-2)}-m^{(i)}\delta V^{(i-2)}$
  • Set $\delta V^{(i)}=\Delta V(\phi_h,\phi^*_h+C^{(i)}\xi_h)$
  • Set $i\leftarrow i+1$
end do while
• Set $C=C^{(i)}$
```

N_I is the number of simplices in $\mathscr{K}$ that contain node I.

Table 2.2: Steps 5–6 of the Ausas et al. [2011] reinitialisation algorithm for the rest of the mesh.

Step 5: Edge distance approximation

```
• Set $changes = 1$
do while ($changes == 1$)
  • Set $changes = 0$
  do ($i$el$= 1, N_{\mathrm{el}}$)
    do ($I = 1, N_{\mathrm{npe}}$)
      if (glob($I$) $\notin \mathscr{P}$) then
        • Find $e_I$ s.t. $e_I = \min_{J \in C_I}[\tilde{\phi}_h(\vec{X}_J) + |\vec{X}_J - \vec{X}_I|]$
        if ($\tilde{\phi}_h(\vec{X}_I) > e_I$) then
          • Set $\tilde{\phi}_h(\vec{X}_I) = e_I$
          • Set $changes = 1$
        end if
      end if
    end do
  end do
end do while
```

Step 6: Shadow distance correction

```
• Set $changes = 1$
do while ($changes == 1$)
  • Set $changes = 0$
    do ($i$el$= 1, N_{\mathrm{el}}$)
      do ($J = 1, N_{\mathrm{npe}}$)
        if (glob($J$) $\notin \mathscr{P}$) then
          • Find $F_J$, the opposite face of node $J$ in $i$el
          • Find $\eta_J$ s.t. $\eta_J = \min_{\vec{x} \in F_J}[\tilde{\phi}_h(\vec{x}) + |\vec{X}_J - \vec{x}|]$
          if ($\tilde{\phi}_h(\vec{X}_J) > \eta_J$) then
            • Set $\tilde{\phi}_h(\vec{X}_J) = \eta_J$
            • Set $changes = 1$
          end if
        end if
      end do
    end do
end do while
```

Nomenclature: $\mathscr{P}$, set of nodal points adjacent to $\mathscr{S}_h$; N_{el}, total number of elements (simplices); N_{npe}, number of nodes per single element (three for a triangle, four for a tetrahedron); glob(I), the global index of local incidence I; C_I, set of nodes connected to I, I not included.

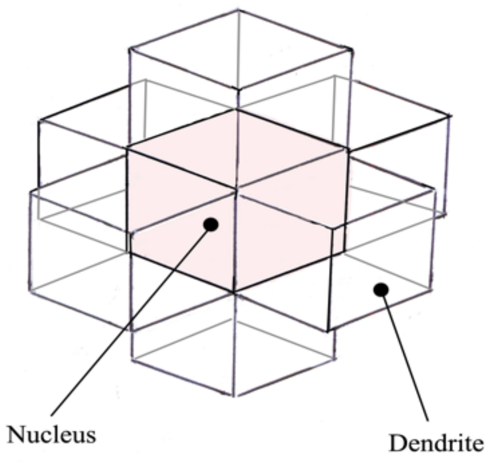


Figure 2.9: The six dendritic cells surrounding a chosen primary nucleus cell about a small region of the free surface.

University of Edinburgh.

2.4 Modifications to `usiniv` and `usproj`

The initial version of the reinitialisation functions described by Olivieri and Ingram [2010] were two dimensional. The following section describes a fully three dimensional implementation of the Ausas et al. [2011] reinitialisation algorithm which works with unstructured hexahedral meshes. As previously [Olivieri and Ingram, 2010] the reinitialisation algorithm is only applied in the immediate neighbourhood of the free surface (Figure 2.8), but here the first step is to construct a temporary unstructured tetrahedral working mesh on which to apply the Ausas et al. [2011] algorithm.

To implement steps 1–4 of the algorithm (Table 2.1) the hexahedral cells which contain the level set Γ but whose centroid value of $\phi < 0$ are identified. The dendrite cells immediately surrounding these nucleus cells are then identified (Figure 2.9). The dendrite nodes are used to calculate the weights needed for updating ϕ^* within the algorithm but are not updated themselves. Once a sweep of all the nuclei has been completed the process is repeated to identify cells which contain the level set Γ, but whose centroid value of $\phi > 0$.

The connectivity array `nbcell(iel, ifac)` is used to access the neighbouring dendrite cell connected through face `ifac` to the nucleus cell `iel`. For an internal cell there will be six faces, but for boundary cells there will be less. To prevent the reinitialisation algorithm being unnecessarily complicated by the logic needed to deal with this

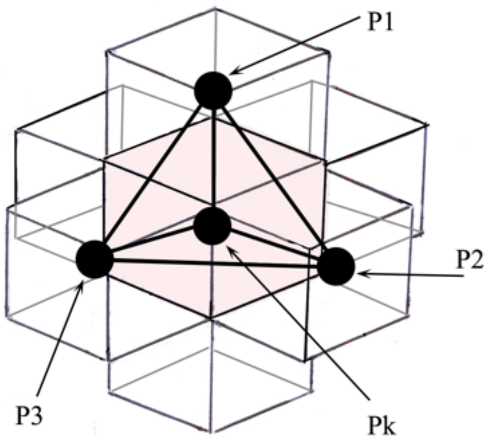


Figure 2.10: Formation of a tetrahedral simplex from the dendritic centroids and a central nucleus centroid.

case, the algorithm relies on the halo of ghost cells which surround the computational domain.

For a given nuclei the set of dendritic cells will each contain both positive and negative values of ϕ. Connecting the centroids of each dendritic cell to its neighbours and the nuclei forms two tetrahedrons (see Figure 2.10). Any tetrahedron whose vertices contain values of ϕ which are all of the same sign is discarded. A consequence of this construction is that three types of tetrahedron can be created, each is characterised by the location of the plane formed by the S_h cutting the tetrahedron. Figures 2.11 to 2.13 illustrate the three types.

Once a data structure containing the simplicies has been constructed the first four steps of the Ausas et al. [2011] reinitialisation algorithm can be performed. In step 1 the normal distances from P_k to free surface intersection points $S_h\overrightarrow{nb_i}$ are found and averaged to find the initial estimate of ϕ^*. In steps 2 to 4 the volume of the cut tetrahedron adjacent to P_k is used weighted by the Heaviside function,

$$H(\phi) = \begin{cases} 1 & \text{if } \phi > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Steps 5 and 6 of the algorithm, which apply to the secondary cells, are applied to the dendritic cells individually. Once again this is done in two sweeps, the first which considers cells where $\phi > 0$ and the second in which the algorithm is applied to $\phi < 0$.

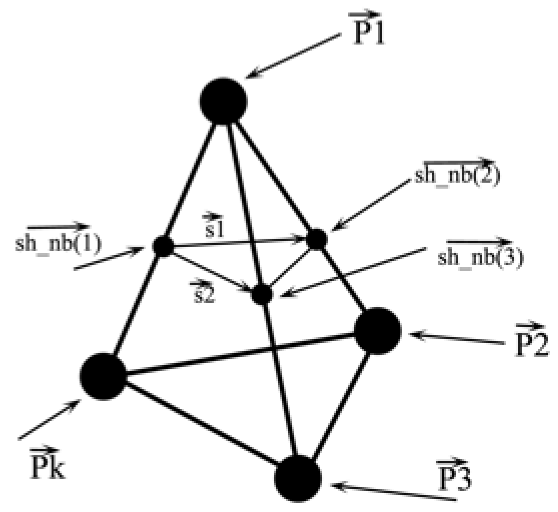


Figure 2.11: Type 1 simplex cut by the free surface at the cut points $\overset{\rightarrow}{S_h nb_1}$, $\overset{\rightarrow}{S_h nb_2}$, and $\overset{\rightarrow}{S_h nb_3}$. P_k is the centroid of the nucleus and P_1, P_2, and P_3 are the centroids of the dendrites.

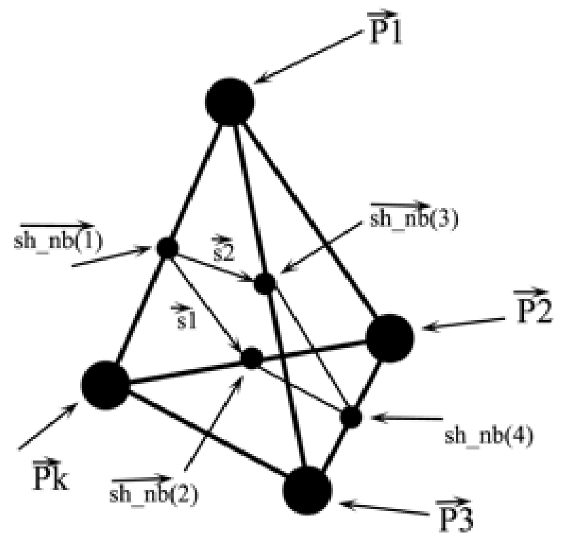


Figure 2.12: Type 2 simplex cut by the free surface at the cut points $\overset{\rightarrow}{S_h nb_1}$, $\overset{\rightarrow}{S_h nb_2}$, $\overset{\rightarrow}{S_h nb_3}$, and $\overset{\rightarrow}{S_h nb_4}$. P_k is the centroid of the nucleus and P_1, P_2, and P_3 are the centroids of the dendrites.

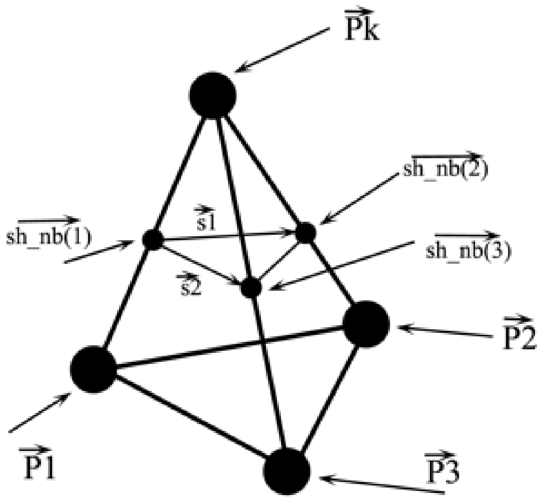


Figure 2.13: Type 3 simplex cut by the free surface at the cut points $S_h\overrightarrow{nb_1}$, $S_h\overrightarrow{nb_2}$, and $S_h\overrightarrow{nb_3}$. P_k is the centroid of the nucleus and P_1, P_2, and P_3 are the centroids of the dendrites.

2.4.1 Database Structure for Reinitialisation

The data structure required for storage of the tetrahedra surrounding the free surface is constructed in two steps. This has been implemented using an object orientated approach where the root object (nuclei, k) may have a set of κ simplicies, in each of which the level set function, ϕ, changes sign. This is illustrated in Figure 2.14. To account for the varying number of simplicies associated with each nuclei a linked list of tetrahedra is used.

This data structure is used in both `usiniv` and `usproj`. Since the Ausas et al. [2011] algorithm is performed in two sweeps, two separate lists are maintained. `ptr_NegnwbElem(i)` is used for simplicies below the free surface ($\phi < 0$) and `ptr_nwbElem(j)` for simplicies above it. Two counters `nwb_cnter_Neg` and `nwb_counter` are used to record the number of negative and positive nuclei respectively. Each of the narrow band elements (see Figure 2.14) has a number of attributes (see Table 2.3) associated with it.

This data structure is initialised in `usiniv` and used to reinitialise ϕ in `usproj`.

2.4.2 Root finding method

In the Ausas et al. [2011] reinitialisation algorithm the correction to the level set function $\psi_k = C\xi_k$, where ξ_k is a continuous function for node k, and C is an arbitrary constant.

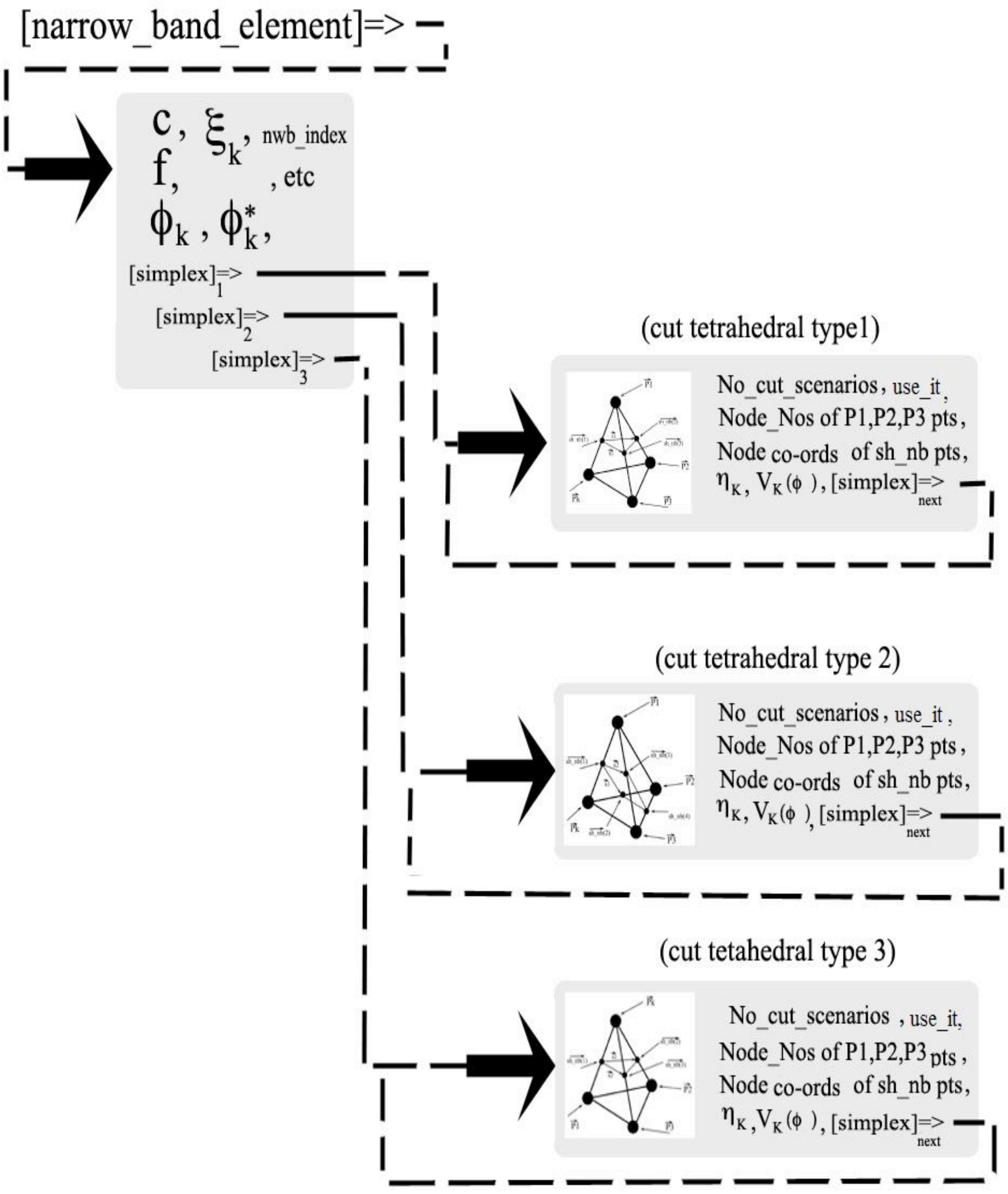


Figure 2.14: Data structure for storing the tetrahedral simplicies surrounding a nuclei node.

Table 2.3: Fortran-95 type definitions for `narrow_band_element` and `simplex` with definitions of their attributes.

`type :: simplex`	
`logical :: use_it`	Cut scenario object filter
`integer :: sim_count`	No of cut scenarios,
`integer :: sim_node(3)`	Node Nos of *P1*,*P2*,*P3* pts,
`double precision :: sh_nb(4,3)`	Node co-ords $\overrightarrow{sh\_nb}$ of free surface area,
`double precision :: eta`	η_k piecewise constant fn at k,
`double precision :: vol_phi`	$V_K(\phi)$ simplex volume above surface,
`type(simplex), pointer :: next`	*[simplex]=>* pointer to a pointer needed for linking list of cut scenarios
`end type simplex`	
`type :: narrow_band_element`	
`double precision :: cval`	C const' that globally preserves volume,
`double precision :: fval`	f fn used in Regila Falsi method,
`double precision :: ph_k`	ϕ_K level set value at node k
`double precision :: ph_star`	ϕ^*_K signed distanced fn at node k
`double precision :: xi_h`	ξ_K continous fn at node k,
`Integer :: nwb_index`	Node No of P_K the nucleus node,
`Integer :: neg`	No of primary neighbour cells with +ve ϕ,
`Integer :: pos`	No of primary neighbour cells with -ve ϕ,
`Integer :: Sec_pos`	No of secondary neighbour cells with +ve ϕ,
`Integer :: Sec_neg`	No of secondary neighbour cells with -ve ϕ,
`Integer :: Spos(8)`	Node No of one of *Sec_pos* secondary cells,
`Integer :: Ppos(8)`	Node No of one of *pos* primary cells,
`Integer :: Sneg(8)`	Node No of one of *Sec_neg* secondary cells,
`Integer :: Pneg(8)`	Node No of one of *neg* primary cells,
`logical :: nb_nochange`	(unused switch for possible debugging),
`type (simplex), pointer :: ptr_cutTetra_type1`	*[simplex]*$_1$=> pointer to type 1 cut scenarios,
`type (simplex), pointer :: ptr_cutTetra_type2`	*[simplex]*$_2$=> pointer to type 2 cut scenarios,
`type (simplex), pointer :: ptr_cutTetra_type3`	*[simplex]*$_3$=> pointer to type 3 cut scenarios,
`end type narrow_band_element`	
`!******`	
`type (narrow_band_element),allocatable :: ptr_NegnwbElm(:)`	***ptr_NegnwbElm(i)*** -ve ϕ side narrow band ptr,
`type (narrow_band_element),allocatable :: ptr_nwbElm(:)`	***ptr_nwbElm(j)*** +ve ϕ side narrow band ptr

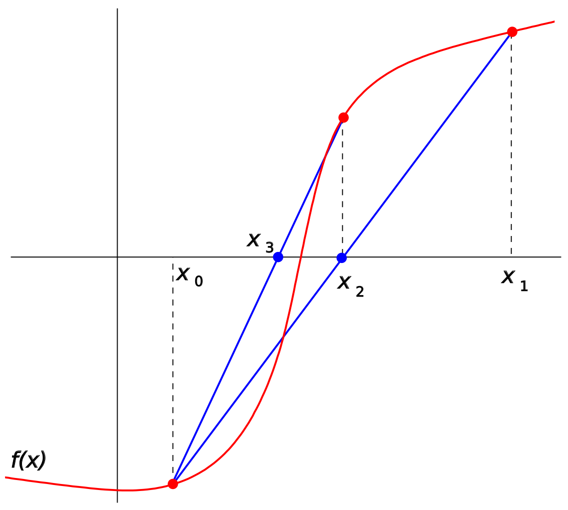


Figure 2.15: Secant method for finding the root of $f(x)$ from two starting points, x_0 and x_1.

C is determined so as to ensure conservation of volume by enforcing

$$\Delta V\left(\phi, \phi^* + C\xi\right) = 0.$$

This non-linear equation has been solved using the secant method [Gregory and Redmond, 1994]. We begin by writing the above condition as a function of the change in volume, i.e.

$$f = \Delta V\left(\phi, \phi^* + C\xi\right).$$

Given two initial gueses x_0 and x_1 for the location of the root, $f(x) = 0$, a sequence of estimates for the location of the root can be found (see Figure 2.15) by drawing the secant between $(x_n, f(x_n))$ and $(x_{n-1}, f(x_n))$,. The new value is found by computing

$$x_{n+1} = x_n - \left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_n)}\right] f(x_n).$$

In the present implementation a maximum of 20 iterations are performed with the iteration stopping when $|f(x_{n+1})| < 1.0 \times 10^{-6}$.

2.4.3 Implementation of step 6

The final step of the Ausas et al. [2011] reinitialisation algorithm modifies the level set function, ϕ, on simplex face, F_j (Figure 2.16). This involves computing the centroid, $\boldsymbol{x}$, of the cut face of the tetrahedron nearest the adjacent node $\boldsymbol{X}_J$ and determining $\phi(\boldsymbol{x})$.

We begin by finding `p_x`, the intersection point of the vector $\overrightarrow{\boldsymbol{X}_J\boldsymbol{X}_k}$ on the panel of the simplex facing $\boldsymbol{X}_J$. The intersection point `p_x` is computed using the subroutine

```
subroutine create_vec(sec, v_hat, val, p_x)
```

where `v_hat` is the outward normal vector to the panel, which lies in the direction $\overrightarrow{\boldsymbol{X}_k\boldsymbol{X}_J}$. Once this has been found a check is performed to ensure the point lies on the simplex by calling

```
logical function check_simplex(ntri, simVec, p_x)
```

where `ntri` is the number of nodes that map out the panel of the simplex considered, `simVec(i, j)` is an array which defines the nodes which lie in the plane with the surface of face F_J.

Once the location of `p_x` is determined and checked, $\phi(\boldsymbol{x})$ can be found using isotropic tri-linear interpolation [Li et al., 2005]. Essentially

$$\phi(x, y, z) = C_{00} + C_{01}x + C_{10}y + C_{11}z$$

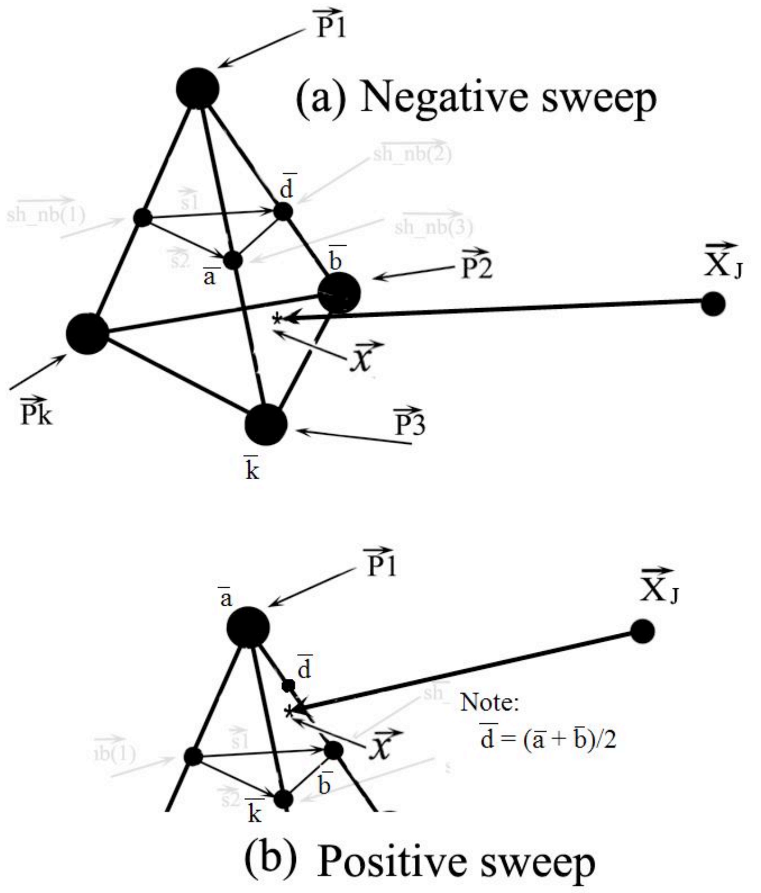


Figure 2.16: Step 6 of the Ausas et al. [2011] reinitialisation algorithm: Shadow distance correction involving four nodes $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{d}$ and $\boldsymbol{k}$ for the $\phi > 0$ and $\phi < 0$ sweeps.

where C_{00}, C_{01}, C_{10}, and C_{11} are the interpolation coefficients for a flatter end tetrahedral box, determined by solving the linear system

$$\begin{pmatrix} C_{00} \\ C_{01} \\ C_{10} \\ C_{11} \end{pmatrix} = \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{pmatrix}^{-1} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix}.$$

The coordinates of the points (x_1, y_1, z_1) to (x_4, y_4, z_4) are provided by `simVec(i, j)`. The construction and inversion of the linear system are performed by the subroutine

`convert_Vec(ntri, k, a, b, c, simVec)`

and `calc_invert_matrix(simVec, a_matrx, ntri)` .

The interpolated value of $\phi(\boldsymbol{x})$ is found using

`calc_phi_interp(4, phi, a_matrx, p_x)`

The master routine implementing step 6 of the algorithm is

`find_min_dist(current, ntri, Lr, irV, sec, iside, dI)`

where `current` is a pointer to the cut simplex under consideration, `ntri`, `Lr`, and `irV` describe the panel for which the correction is being computed, `sec` contains the position vector $\boldsymbol{x}_J$ of the adjacent dendrite node, and `iside` is a switch which is used to determine if this calculation is being performed for a positive level set sweep (`iside=1`) or a negative level sweep (`iside=-1`). Finally, `dI` is the computed value of the corrected level set function $\phi(\boldsymbol{x}_J)$, which is applied if $\phi(\boldsymbol{x}_I) > dI$.

The source code for these routines can be found in Appendices B and C.

Chapter 3

Changes made to the *Code_Saturne* kernel to include density and viscosity variation.

After implementing the level set advection and reinitialisation routines in *Code_Saturne*, describe in the previous chapter, it is necessary to define the local fluid density and local dynamic viscosity in each control volume from the level set function, ϕ. These quantities are defined using a smoothed Heaviside function,

$$H(\phi) = \max\left(1, \min\left(0, \frac{1}{2} - \frac{1}{2}\left[\frac{\phi}{\epsilon} + \frac{1}{\pi}\sin\left(\pi\frac{\phi}{\epsilon}\right)\right]\right)\right). \tag{3.1}$$

The free parameter, ϵ, should be of the order of 0.08 for good mass conservation. This formulation should ensure that numerical interface has a thickness of 2ϵ. This formulation is implemented within `usphyv.f90` (see Appendix D).

After introducing varying density and viscosity into *Code_Saturne* version 2.0, it proved impossible to obtain a stable flow solution. Applying the ghost fluid method Desjardins et al. [2008] as an internal boundary condition proved the most successful approach but still led to non-physical oscillations velocity (see Figure 3.1). Advice was sought from Renuda UK Ltd who provide support for *Code_Saturne*, and their response was as follows:

> "On the theoretical side, from the standpoint of *Code_Saturne*, I was concerned that a pressure-based method might not be amenable to the Ghost Fluid method. For the method to work, mass conservation will have to be ensured very tightly. This would require:
>
> - a sound algorithm
> - making sure that the level set yields a sharp interface consistent with the other conserved variables and where mass will not be dissipated

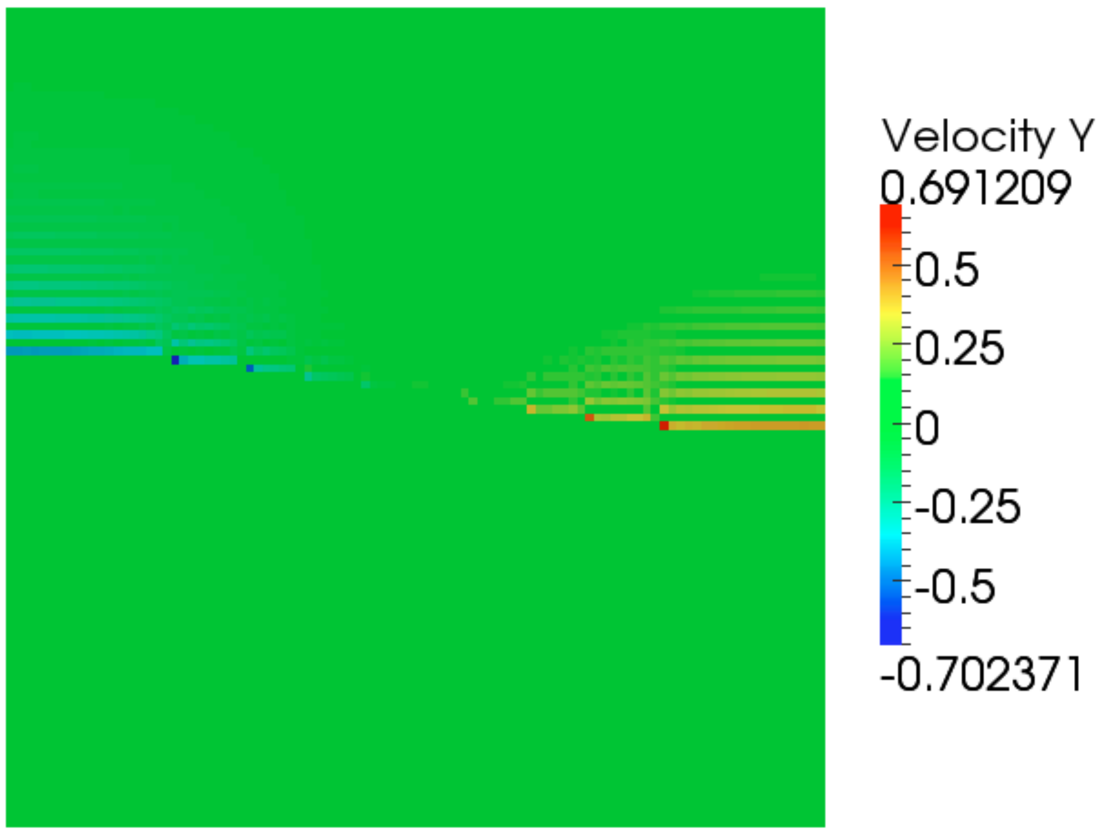


Figure 3.1: Velocity errors in the Y component of velocity in the low amplitude sloshing test case after 297 time steps.

> Regarding the present *Code_Saturne* implementation, I think that the fact that the $\frac{\partial \rho}{\partial t}$ term is not applied in the projection step (pressure correction) in the incompressible algorithm will lead to problems for the two-phase case where this term is bound to be significant."
>
> Dr. N. Tonello, RENUDA.

A further concern was raised over the pressure correction algorithm, which also assumes that because the flow is incompressible ρ is either constant or only undergoes small changes between cells, for example because of temperature changes in buoyant flow. Section 3.1 describes the issues with the Rhie and Chow [1983] interpolation scheme and Section 3.2 discusses changes made to the pressure correction algorithm. The work to identify and remediate theses issues was conducted by RENUDA under a subcontract from the University of Edinburgh. The subcontract ran from June 2012 to March 2013, under this subcontract RENUDA undertook to

- Solve the level set function (Using the Edinburgh code with comparisons to *Code_Saturne* scalar advection routines),
- modify the algorithm to enforce a divergence free velocity field, and
- modify the mass flux and Rhie-Chow formulations.

This work was conducted with advice from the EDF *Code_Saturne* development team.

3.1 Rhie and Chow Interpolation Scheme

The Rhie and Chow [1983] interpolation scheme provides a method for preventing the occurrence of checkerboard pressure oscillations in collocated finite volume schemes. These oscillations arise in the momentum equation where the discretisation of $\frac{\partial p}{\partial x}$ can cause the pressure in odd and even grid cells to become decoupled. Many CFD solvers use a staggered grid formulation, but codes like *Code_Saturne* and *CFX* make use of the Rhie and Chow [1983] scheme on a collocated grid. This is done by discretising the momentum equation

$$\frac{\partial \rho u}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} u) = -\frac{\partial p}{\partial x} + \nabla \cdot (\mu \nabla \boldsymbol{u}) + S. \tag{3.2}$$

over the control volume, P, to give,

$$a_P u_P + (\nabla_x p)_P = \left(\sum a_{nb} u_{nb}\right)_P + S_P.$$

The same discretisation is applied to the adjacent control volume, A, giving

$$a_A u_A + (\nabla_x p)_A = \left(\sum a_{nb} u_{nb}\right)_A + S_A.$$

Because mass is conserved the u velocity component on the face between P and A can also be represented as

$$a_f u_f + (\nabla_x p)_f = \left(\sum a_{nb} u_{nb}\right)_f + S_f.$$

Rhie and Chow [1983] use a weighted average of the P and A discretisation to approximate $(\sum a_{nb} u_{nb})_f + S_f$, i.e.

$$a_f u_f + (\nabla_x p)_f \approx \overline{\left(\sum a_{nb} u_{nb}\right)_f + S_f} = \overline{a_f u_f} + \overline{(\nabla_x p)_f}. \tag{3.3}$$

Assuming that $a_f \approx \overline{a_f}$, u_f can be written as

$$u_f = \overline{u_f} + \overline{d_f}\left(\overline{(\nabla_x p)_f} - \nabla_x p)_f\right), \tag{3.4}$$

where

$$\begin{aligned}
\overline{u_f} &= \alpha u_P + (1-\alpha)u_A, \\
\overline{(\nabla_x p)_f} &= \alpha \nabla_x p_P + (1-\alpha)\nabla_x p_A, \\
\nabla_x p_f &= A_f n_x \left(p_A - p_P\right), \\
a_f &= \alpha a_P + (1-\alpha)a_A, \text{and} \\
\overline{d_f} &= a_f^{-1}.
\end{aligned}$$

All that remains is to determine the weighting coefficient, α, whilst $\alpha = 0.5$ is a common choice different flow solvers use a variety of approaches to determining α.

Implementing Rhie and Chow [1983] in an incompressible solver where density varies causes issues with the weighted linear average for a_f, since the coefficient a includes the density which may not be constant between the the two control volumes A and P.

3.2 Pressure Correction

Starting with the assumption that both fluid phases are incompressible, the density field must at all times satisfy

$$\frac{\partial \rho}{\partial t} = \boldsymbol{u} \cdot \nabla p. \tag{3.5}$$

In the original version of *Code_Saturne* (as distributed by EDF) this incompressibility constraint is only imposed during the pressure correction step of the flow algorithm. In the pressure correction step

$$\nabla.(\rho \boldsymbol{u}) = \Omega,$$

with the associated correction computed as:

$$\nabla \cdot \left(\Delta t \delta p^{n+\theta}\right) = \nabla \cdot \rho \widetilde{\boldsymbol{u}} - \Omega,$$

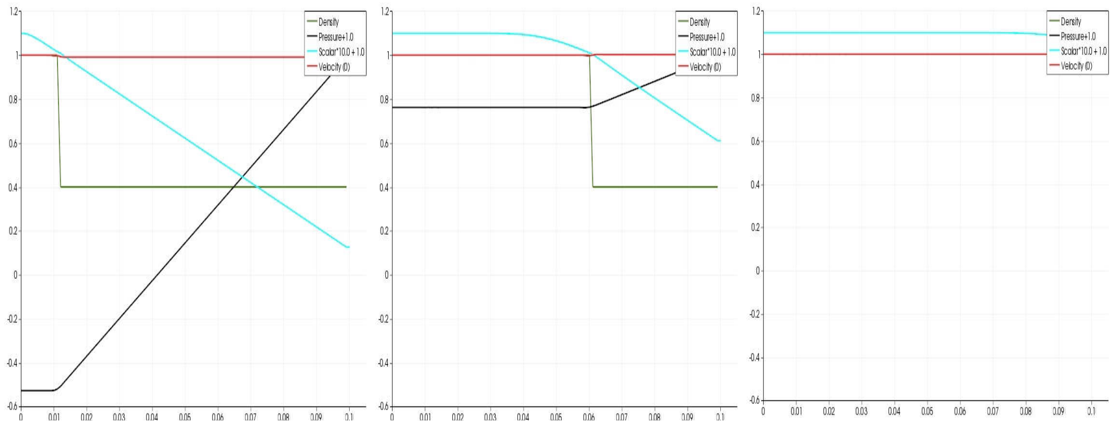
where $\widetilde{\boldsymbol{u}}$ and Ω are the predicted velocity and user-specified mass source term respectively. This formulation fails to account for the temporal variation of density and must be modified to include $\frac{\partial \rho}{\partial t}$, i.e.

$$\nabla \cdot \left(\Delta t \delta p^{n+\theta}\right) = \nabla \cdot \rho \widetilde{\boldsymbol{u}} - \Omega + \frac{\partial \rho}{\partial t}. \tag{3.6}$$

Combining (3.6) with (3.5) and using the *predicted* variables gives,

$$\nabla \cdot \left(\Delta t \delta p^{n+\theta}\right) = \rho \nabla \cdot \rho \widetilde{\boldsymbol{u}} - \Omega. \tag{3.7}$$

Modifications were made to the



$t = 0.0$s $t = 0.05$s $t = 0.1$s

Figure 3.2: Water hammer simulation for $\rho_i/\rho_0 = 2.5$

Table 3.1: Initial conditions for the water hammer simulations. rho_0 is the density in the right hand end of the domain, ρ_{ref} is the reference density specified in the *Code_Saturne* problem definition file, and ρ_i/ρ_0 is the density ratio.

Case	ρ_0	ρ_{ref}	ρ_i/ρ_0
1	0.9	1.0	1.11
2	0.9	0.9	1.11
3	0.5	1.0	2.00
4	0.5	0.5	2.00
5	0.1	1.0	10.00
6	0.1	0.1	10.00

3.2.1 Water Hammer Test Case

To test this formulation a simple one-dimensional "water hammer" test case was developed. In this case the domain is divided into two parts, with discontinuous conditions. The left hand end of the domain ($x \leq 0.01$) has a a density of $\rho_i = 1\,\mathrm{kgm}^{-3}$ while the remainder of the domain has a lower density ρ_0. Velocity is constant $u_i = u_0 = 1\,\mathrm{ms}^{-1}$. Pressure is set to a constant value p_i in the left hand end of the domain, but varies linearly in the lighter fluid (the gravity vector is set to $(1, 0, 0)^T$). The level set function ϕ is initialised as a signed distance from $x = 0.01$ and is reinitialised at every time step. As time progresses the"plug" of heavy fluid propagates along the domain (see Figure 3.2).

The water hammer case was run for a variety of density ratios (see Table 3.1). These computations showed that when *downwinding* was applied to the compute the value of density on the right hand side of (3.7) as opposed to cell centred averaging stable results could be obtained. This approach was successful up to a density ration of $\rho_i/\rho_0 = 10$, past which the computations became unstable. Further work was carried out to review the methodology and implement a new form of the pressure correction scheme. These steps are detailed in Sections 3.2.2 and 3.2.3 below.

3.2.2 Diagonal Strengthening

Trials were undertaken to strengthen the diagonal of the pressure correction matrix and evaluate the effect of the procedure on stability. Diagonal dominance was achieved within the subroutine `resolp` by adding an arbitrary constant to the diagonal terms. Such modification of the diagonal terms is allowed within *Code_Saturne* and can be applied without changing the solution [EDF R&D, 2013b].

Repeating the water hammer tests (see §3.2.1) showed that diagonal strengthening improved the solution. Adding a constant to both the matrix (in `resolp`) and the right hand side terms (in `smbrs`) allowed density ration of 1000 : 1 to be achieved. In effect the linear system is modified by applying and additive constant, α, as follows:

$$(\alpha I A)\,\boldsymbol{x} = \alpha \boldsymbol{b}.$$

Diagonal strengthening was not, however, pursued further as a generic formulation for α cannot be simply formulated. This leads to the disadvantage that a user tuneable coefficient would need to be provided and specified *a priori* for each case.

3.2.3 Pressure Correction Implementation

An alternative to diagonal strengthening is to move the density term on to the right hand side of (3.7). This enables the density term to appear inside the divergence term of the pressure correction matrix. In a finite volume formulation, where the Gauss divergence theorem is applied, this means that densities are computed on cell faces rather than at cell centres. This has the effect of distributing the density discontinuity over a cell and communicates density information from every direction. A consequence of this is that downwinding directions do not have to be explicitly specified in the scheme, as the flux formulation will select the appropriate directional weighting implicitly - in a similar manner to that employed in Godunov [1959] type schemes.

Enforcing the incompressibility continuity condition, $\nabla \cdot \boldsymbol{u} = 0$, in (3.7) gives:

$$\nabla \cdot \left(\frac{\Delta t}{\rho} \delta p^{n+\theta} \right) = \nabla \cdot \widetilde{\boldsymbol{u}} - \frac{\Omega}{\rho}. \tag{3.8}$$

Whereas diagonal strengthening (§3.2.2) only requires modification to the right hand side of the pressure correction equation, implementing (3.8) in *Code_Saturne* requires changes to the definition of the matrix components. This change requires the Δt coefficients of the cell faces to be redefined as $\Delta t/\rho$ on the internal face, on boundary faces and in the Rhie and Chow terms. For an internal face, f, separating cells i and j, the face value of the coefficients are calculated using

$$\left(\frac{\Delta t}{\rho}\right)_f = \frac{1}{2}\left(\frac{\Delta t}{\rho_i} + \frac{\Delta t}{\rho_j}\right),$$

which corresponds to the harmonic mean of the density.

The water hammer tests (see §3.2.1) showed that this implementation made it possible to obtain a very accurate and stable solution for density rations in excess of $1 : 1000$. This formulation has been applied to a number of test problems which will be described in the following chapter.

Chapter 4

Validation cases for the free-surface solver performed under PerAWaT

4.1 Introduction

The level-set formulation of *Code_Saturne* described in Chapters 2 and 3, has been subjected to five test cases

1. Low amplitude sloshing,
2. Subcritical flow over a submerged bump,
3. Transcritical flow over a submerged bump (without a hydraulic jump),
4. Transcritical flow over a submerged bump (with a hydraulic jump), and
5. Flow over a submerged hydrofoil,

reported in this chapter.

Case 1 is compared with a semi-analytical solution. Cases 2–4 have been compared with an analytical solution for the depth averaged equations, which provides a good approximation of the real 3D flows. Finally, case 5, is compared with experimental results obtained by Duncan [1983] and to numerical simulations performed using CF/X by Gretton et al. [2010].

The analytical solution for cases 2–4 has been computed using the depth averaged shallow water equations, as such shallow water codes such as TELEMAC [Moulinec et al., 2011] should provide much better agreement with these results than those obtained using a Navier-Stokes solver. The shallow water equations make assumptions of hydrostatic pressure and negligible vertical mixing which the Navier-Stokes solutions do not. In

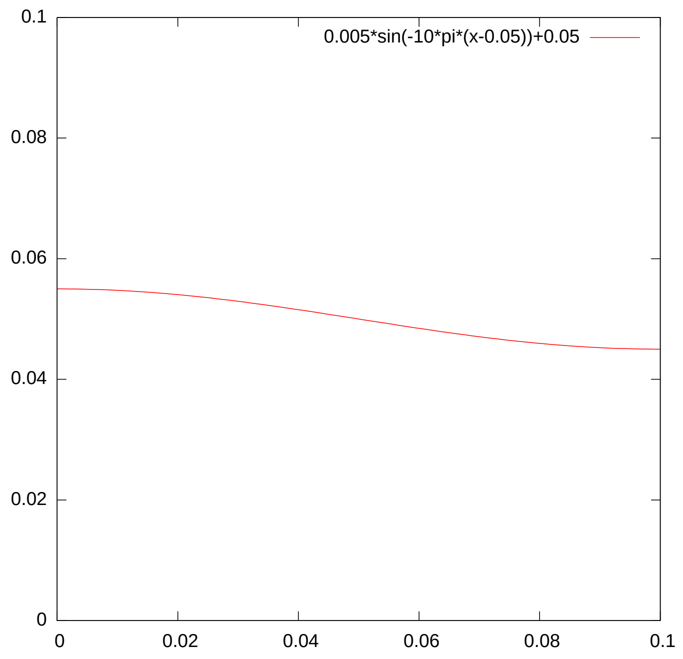


Figure 4.1: Initial location of the free surface for the low amplitude sloshing test case.

particular the Navier-Stokes results for case 4, which includes a hydraulic jump, can admit wave breaking which cannot occur in shallow water simulations. That said, these three test cases provide significant tests of the capability of the free surface solver to reproduce sub-critical, super-critical and over-expanded water flow using only the global pressure field and the local velocities.

4.2 Low amplitude sloshing

The sloshing of a liquid wave with finite amplitude under the influence of gravity is a classical test case for free surface flow problems [Tadjbakhsh and Keller, 1960] and for evaluating interface tracking methods [Qian et al., 2003]. Using linear wave theory and starting with the assumption that the waves are periodic both in time and the horizontal direction Tadjbakhsh and Keller [1960] derived an analytical solution for gravity waves on the surface of an inviscid, incompressible fluid, with finite depth.

In the test case the free-surface is initialised so that

$$\eta(x, 0) = a\cos(-kx) + d$$

where the wave amplitude, $a = 0.005$m, the wave number, $k = 10\pi$ and the water depth, $d = 0.05$m. The computational domain is a closed rectangular box 0.1m on each side, with the still water level at the mid point (Figure 4.1).

Tadjbakhsh and Keller [1960] showed that the surface profile, $\eta(x,t)$, the velocity potential $\phi(x,y,t)$, and the angular frequency, ω, of the sloshing waves depend on both the dimensionless water depth, $h = d/\lambda = dk/(2\pi)$, and the dimensionless aptitude, $\epsilon = ka$. Their analysis shows that a number of higher modes can be found in the amplitude of the oscillating surface, in particular they note that:

1. The free surface is never flat (as would be expected from a linear process) so when $t = n\pi$,

$$\epsilon\eta(x, n\pi) = \frac{1}{8}\epsilon^2\left(\omega_0^2 + 2\omega_0^{-2} - 3\omega_0^{-6}\right)\cos 2x.$$

   where $\omega_0^2 = \tanh h$, and

2. The velocity vanishes throughout the fluid when $t = (n+\frac{1}{2})\pi$, at which time part of the surface is either at its highest of lowest position. When n is even the greatest rise occurs at $x = 0$ and the free surface profile is given by

$$\begin{aligned}\epsilon\eta(x) &= \left[\epsilon + \frac{\epsilon^3}{256}\left(9\omega_0^{-8} + 6\omega_0^{-4} - 15 + 8\omega_0^4\right)\right]\cos x \\ &+ \frac{1}{8}\epsilon^2\left(\omega_0^{-2} + 3\omega_0^{-6}\right)\cos 2x \qquad (4.1)\\ &+ \frac{3}{256}\epsilon^3\left(9\omega_0^{-12} + 6\omega_0^{-8} + 30\omega_0^{-4} - 16 + \omega_0^4 + 2\omega_0^8\right)\cos 3x.\end{aligned}$$

   when n is odd the free surface profile can again be found using (4.1), substituting $-\epsilon$ for ϵ. Figure 4.2 shows the profile of the surface of the standing wave at $t = (n+\frac{1}{2})\pi$ for the present case where $h = \frac{1}{4}$, and $\epsilon = 0.05\pi$, so $\omega_0 \approx .495$.

In the computed solution we therefore expect to see several modes of oscillation, the first mode represents the linear wave theory solution, while the higher modes (2nd and 3rd order) should cause a change in the amplitude and coincide with twice and three times the natural frequency of the wave.

Qian et al. [2003] presented a comparison between the *Amazon-SC* free surface capturing code, computations by Ubbink [1997] and linear wave theory, which have been reproduced in Figure 4.3. Results from the *Code_Saturne* simulation have been presented below (Figure 4.4). The comparison shows that *Code_Saturne* is reproducing the expected physics and that both the frequency of the oscillations is correctly predicted and the higher modes are present.

The present simulations were conducted on a uniform rectangular 200×200 grid ($\Delta x = \Delta y = 0.0005$m). 50000 iterations were performed with a fixed time step of $\Delta t = 5 \times 10^{-5}$s, and the simulation took 13 hours 1 minute and 29 seconds to run on a single core of a 2.93 GHz, 6 core, Intel Xeon processor.

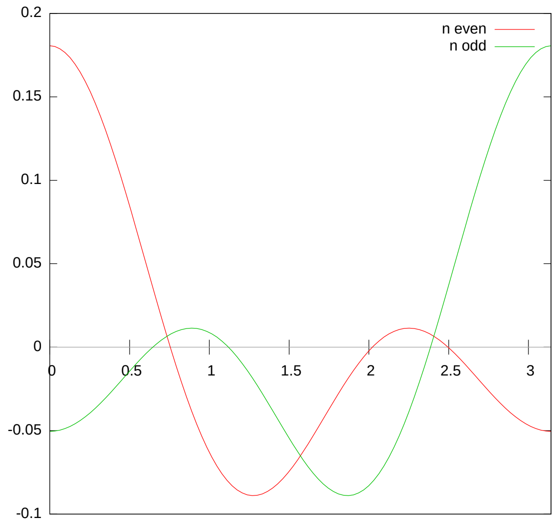


Figure 4.2: Profile of the surface of a small amplitude standing wave in finite depth water at $t = (n + \frac{1}{2})\pi$, for n even (red) and n odd (green). The curves are based on (4.1) with $\epsilon = 0.05$ and $h = 0.25$.

4.2.1 Mass conservation

Figure 4.5 shows that there is a small but consistent change in mass throughout the simulation. Over the entire 2 seconds (50000 iterations) of the simulation there is a net loss in liquid of about 1.3%. This mass loss is attributable to the fact that there are no inlet or outlet boundary conditions in the low amplitude sloshing case (which is performed in a closed box). *Code_Saturne* uses the inlet and outlet boundary conditions to ensure mass conservation within the pressure correction scheme and the absence of the ability to apply this correction leads to the observed mass loss. It is not possible to make comparisons with the total liquid mass computed by either Qian et al. [2003] or Ubbink [1997] since this quantity was not computed by either author.

4.3 Flow over a submerged bump

Vázquez-Cendón [1999] and Zhou et al. [2001] both present shallow water simulations of a set of simple test cases involving flow over a submerged bump. These tests involve flow in a 25m long flume whose bed elevation is defined by

$$Z(x) = \begin{cases} 0.2 - 0.05(x-10)^2 & \text{if } 8 < x < 12, \\ 0, & \text{otherwise.} \end{cases}$$

This classical test case was considered by the working group on dam break modelling [Goutal and Maurel, 1997]. Depended on the upstream boundary conditions the flow

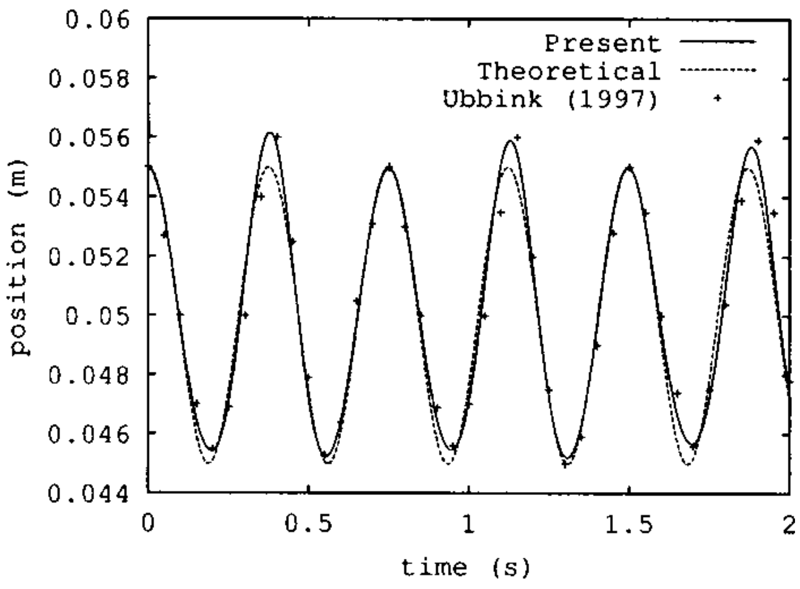
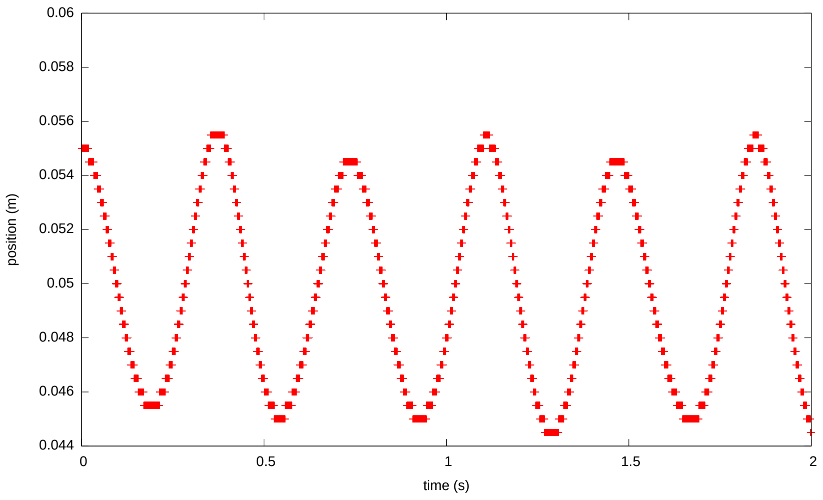


Figure 4.3: Position of the fluid interface at the left end of the domain, computed using *Amazon-SC* (labelled present) with comparisons to computations by Ubbink [1997] and linear wave theory. After Qian et al. [2003]



.

Figure 4.4: Position of the fluid interface at the left end of the domain, computed using *Code_Saturne*

.

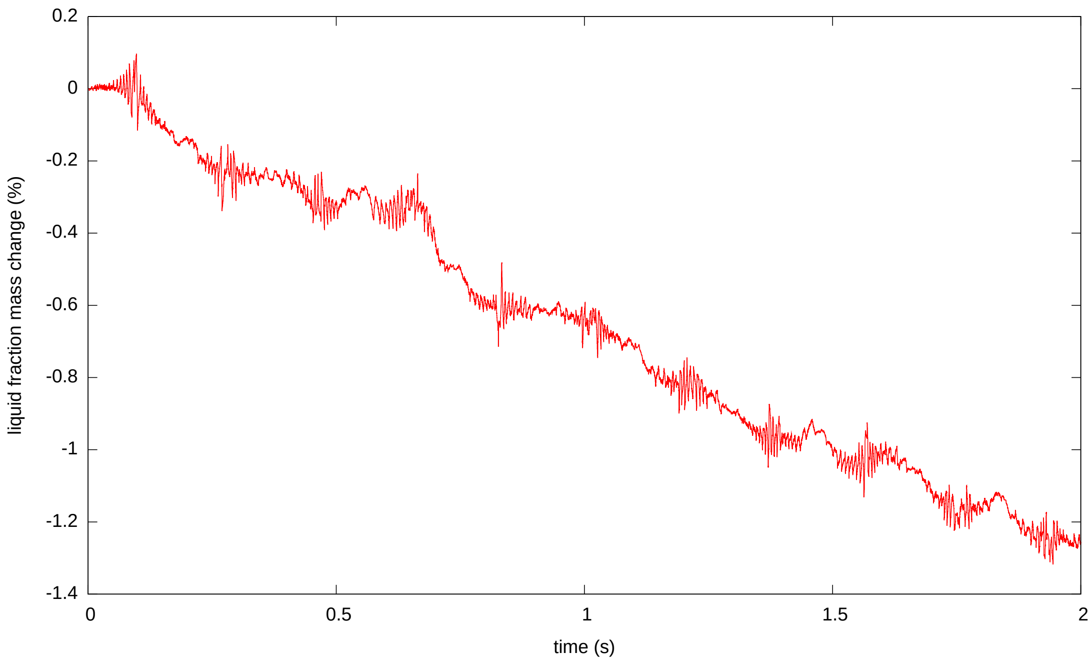


Figure 4.5: Percentage change in the liquid volume mass with time, from the low amplitude sloshing case.

.

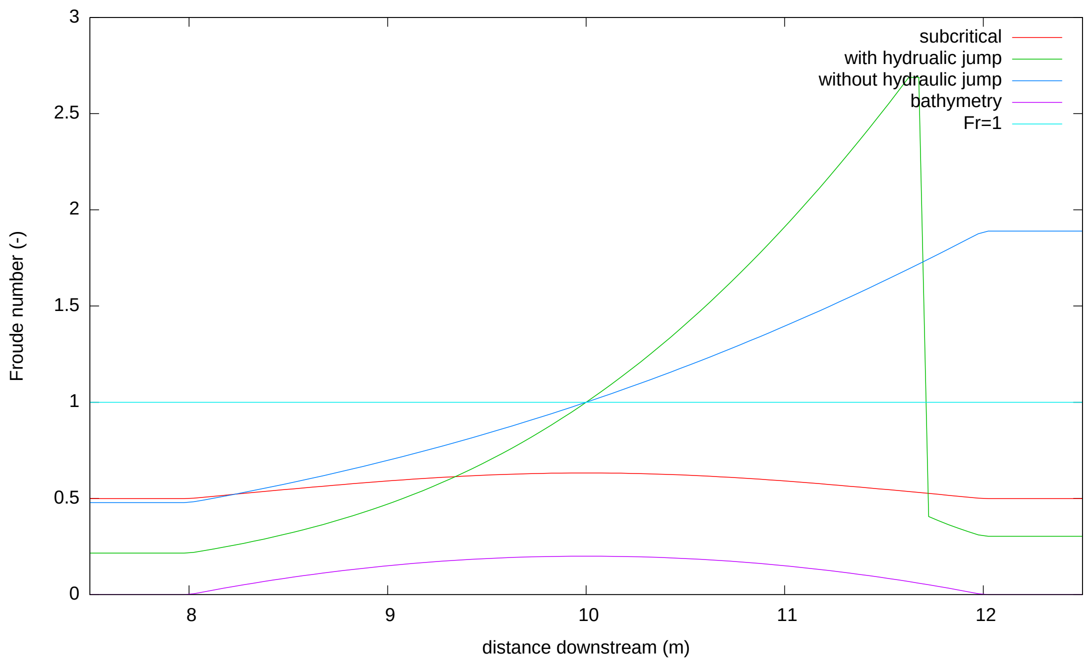


Figure 4.6: Shallow water flow over a submerged bump: Froude number over the bump for the three test cases, computed by SWASHES [Delestre et al., 2013]

may be subcritical, transcritical with or without a steady hydraulic jump or supercritical. Analytical solutions can be computed using the *SWASHES* (Shallow Water Analytic Solutions for Hydraulic and Environmental Studies) software [Delestre et al., 2013]. In the present study simulations have been performed for the subcritical, transcritical (without shock), and transcritical (with shock) cases.

The flow remans subcritical if the Froude number,

$$Fr(x) = \sqrt{\frac{\overline{u(x)}}{gd(x)}} < 1 \ \forall x > 0,$$

where $d(x)$ is the local water depth and $\overline{u(x)}$ is the depth averaged velocity in the stream wise direction (see Figure 4.6). Under these conditions the bump acts in a similar way to a submerged broad-crested weir so we expect the flow to accelerate over the front of the bump with an associated decrease in the free surface elevation (as a depression forms). Over the back of the bump, the flow decelerates and the free surface recovers (Figure 4.7).

In the transcritical cases (Figures 4.8 and 4.9) the Froude number, $Fr(x) \geq 1$ at some point on the bump (Figure 4.6). In both transcritical cases the the Froude number exceeds unity at the crest of the bump. Consequently, as the water flows over the back of the bump it continues to accelerate (with consequent decreases in depth, see Figures

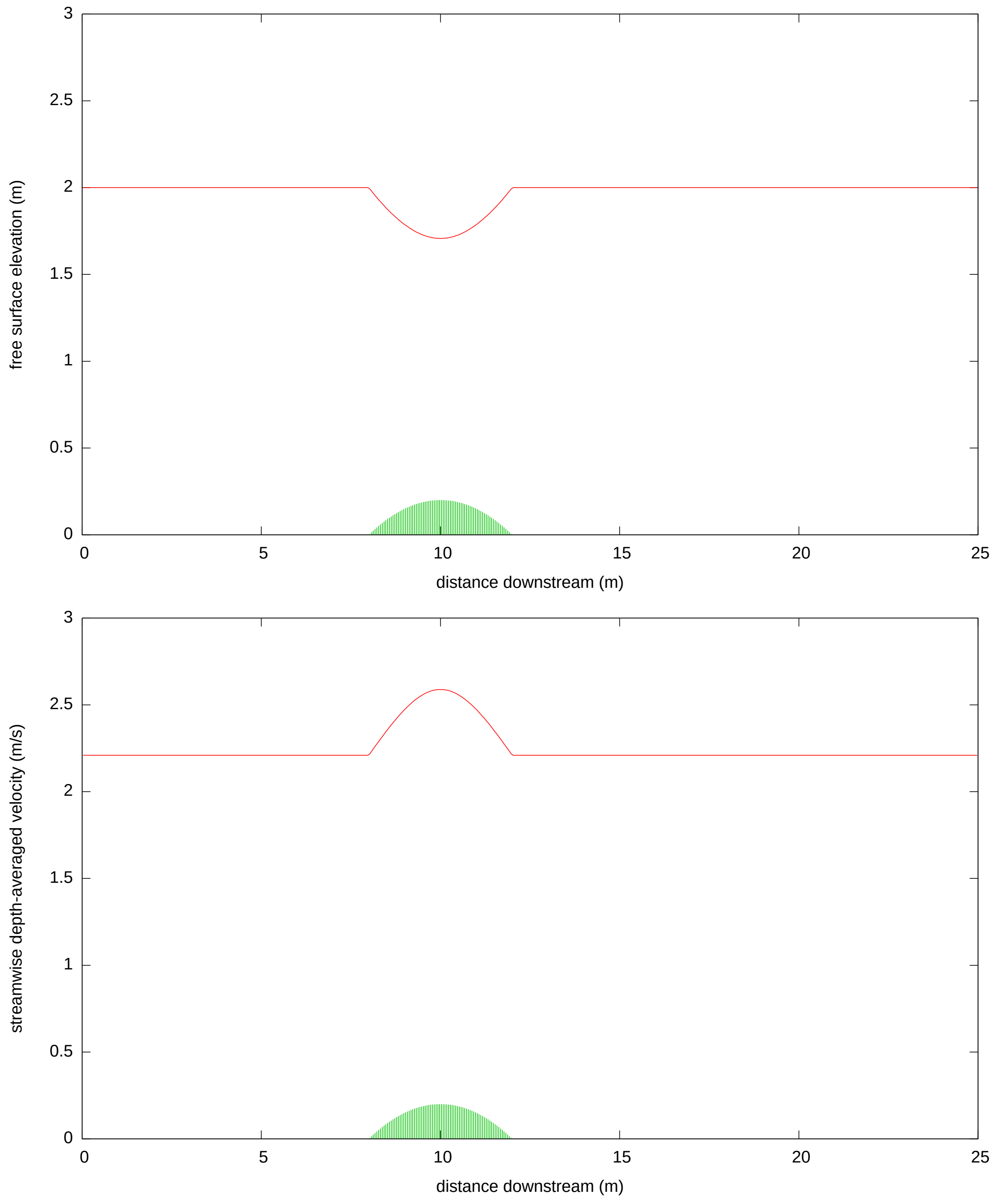


Figure 4.7: Subcritical shallow water flow over a submerged bump: Analytical solution computed using SWASHES [Delestre et al., 2013]: Free surface elevation (top) and depth averaged velocity (bottom).

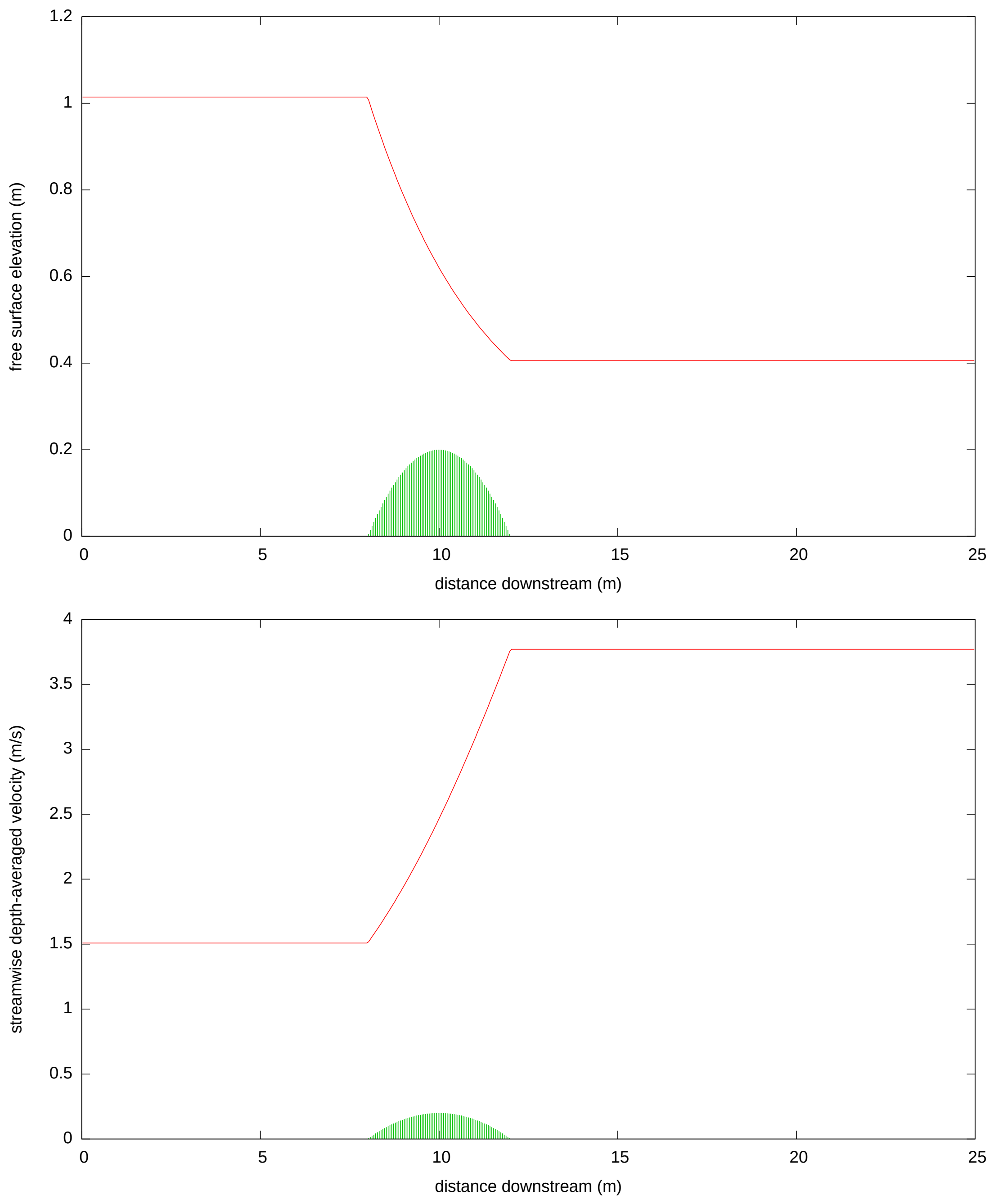


Figure 4.8: Transcritical shallow water flow over a submerged bump (without a hydraulic jump): Analytical solution computed using SWASHES [Delestre et al., 2013]: Free surface elevation (top) and depth averaged velocity (bottom).

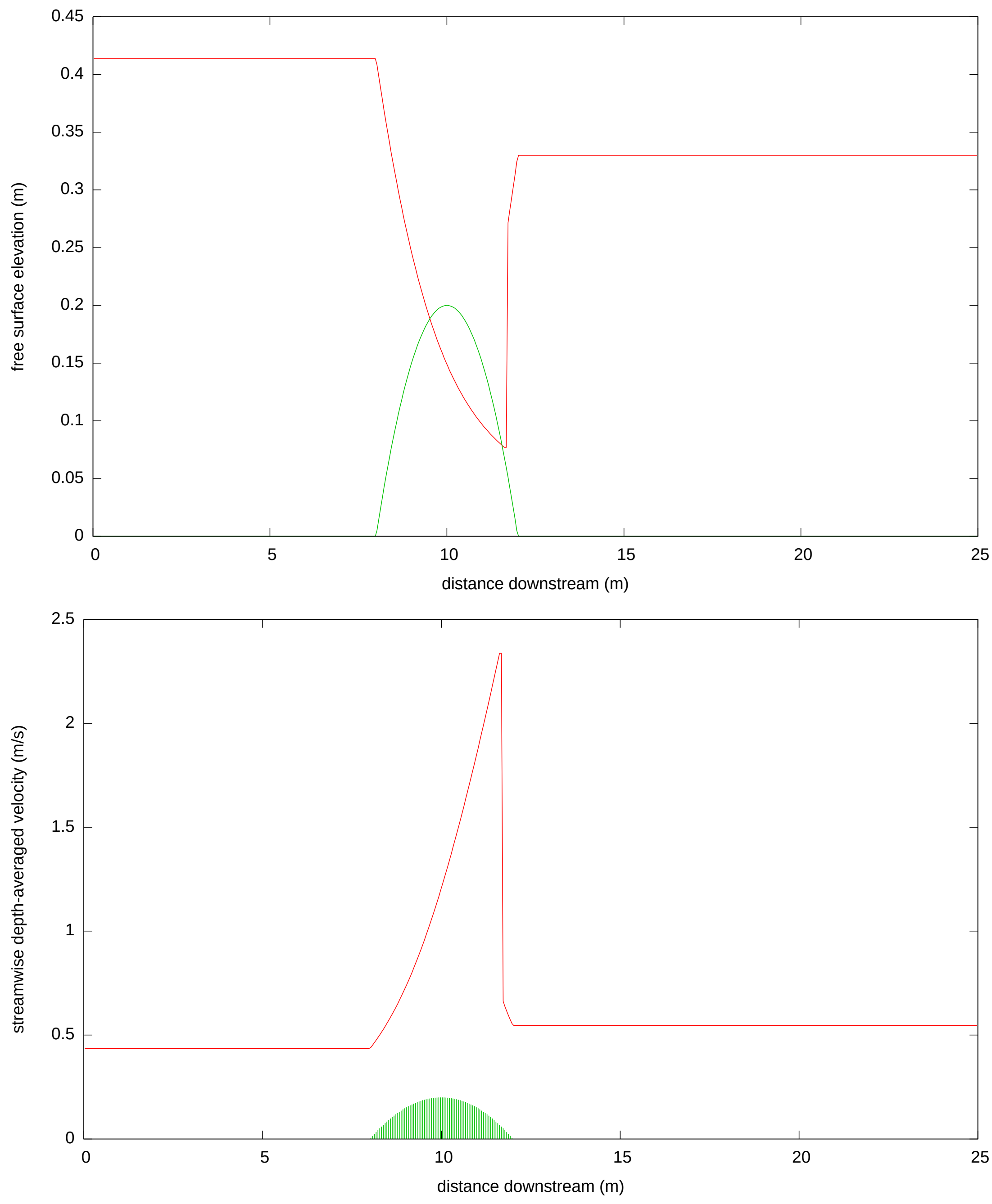


Figure 4.9: Transcritical shallow water flow over a submerged bump (with a hydraulic jump): Analytical solution computed using SWASHES [Delestre et al., 2013]: Free surface elevation (top) and depth averaged velocity (bottom).

4.8 and 4.9). In the case without a hydraulic jump the downstream discharge rate is sufficiently high that the supercritical flow is able to continue downstream to the outlet unimpeded. In the case with a jump, the discharge rate is too low for supercritical flow so the over-expanded flow "shocks down" to the subcritical condition, before flowing towards the outlet.

The transcritical case with a hydraulic jump is an extremely violent and challenging case, with complex wave breaking occurring at the hydraulic jump in experiments. It has recently been studied in detail using both 3D CFD simulations and experiments with Particle Imaging Velocimetry (PIV) [Koo et al., 2012].

4.3.1 Boundary conditions

With the addition of an additional transported quantity, ϕ, the boundary conditions at the up- and downstream boundaries must be modified to respect new physics in the model. Boundary conditions should be determined by information traveling in and out of the domain along the characteristics [Thompson, 1990]. By analogy with the shallow water equations we expect the free surface boundary conditions to be specified as follows:

Subciritcal inflow, $Fr \leq 1 \,|_{x=0}$**,** The free surface elevation *or* the mass flow rate can be specified and the other must be computed using information from inside the domain.

Supercritical inflow, $Fr > 1 \,|_{x=0}$**,** Both the free surface elevation *and* the mass flow rate should be specified.

Subciritcal outflow, $Fr \leq 1 \,|_{x=25}$**,** The free surface elevation *or* the mass flow rate can be specified and the other must be computed using information from outside the domain.

Supercritical outflow, $Fr > 1 \,|_{x=25}$**,** Neither the free surface elevation *nor* the mass flow rate should be specified.

This is in addition to the boundary conditions used for the Navier-Stokes equations, leading to the following sets of boundary conditions:

Subciritcal inflow, $Fr \leq 1 \,|_{x=0}$**,** The velocity components at the inlet are computed using the mass flow rate based on the currently computed free-surface elevation at the inlet. pressure at the inlet is also computed from inside the flow domain, following the normal approach for Navier-Stokes computations. In this case the velocity components are defined as $u = Q/\eta_1$ and $v = 0$, where η_1 is the free surface elevation adjacent to the boundary inside the flow domain.

Supercritical inflow, $Fr > 1 \,|_{x=0}$**,** Both the free surface elevation *and* the mass flow rate should be specified. Because the flow is supercritical we specify $u = Q/\eta_0$ and $v = 0$, where η_0 is the free surface elevation upstream of the boundary. The value of ϕ on the boundary is also specified so that $\eta \mid_{x=0}= \eta_\infty$. It is important to note that supercritical flow is not super-sonic so the upstream pressure must still not be specified, or the boundary conditions will over constrain the flow solution.

Subciritcal outflow, $Fr \leq 1 \mid_{x=25}$, In this case we apply the downstream weir flow condition, by fixing both η_∞ and the downstream pressure p_∞ and allowing the velocity field to develop from the solution. In specifying this boundary condition it is important to remember that p_∞ is a function of z as it needs to include the hydrostatic pressure below the water surface.

Supercritical outflow, $Fr > 1 \mid_{x=25}$, Because neither the free surface elevation *nor* the mass flow rate should be specified, a standard outlet condition can be employed. As before, however, the downstream pressure is a function of z and must include the hydrostatic pressure term below the water surface.

The boundary conditions above the water surface are simply the normal inlet and outlet conditions used in any flow solver, i.e. upstream velocity specified for an inlet and downstream pressure specified at an outlet.

4.3.2 Subcritical flow

For the subcritical test case the initial conditions simply specify a uniform initial water depth ($h = 2.0$m) and a constant initial velocity ($u = 2.21$ms^{-1}). The reference pressure is set to atmospheric pressure $p_0 = 101325$Pa, the reference density is set to $\rho = 1$kgm^{-3}, and the flow simulation is set to laminar (i.e. no turbulence model), with a molecular viscosity of, $\mu = 1.83 \times 10^{-31}$, making the solution effectively inviscid. In all three cases, the use of an approximately inviscid solution is important as it allows a direct comparison with the analytical shallow water solution. The user subroutine `cs_user_initialization` calculates the signed distance function, ϕ, based on the initial free surface location.

The upstream and downstream boundary conditions are specified according the prescriptions, described above, for subcritical inflow and subcritical outflow, with a fixed mass flow rate of $Q = 4.42$m^{3}s^{-1}, and $\eta_0 = \eta_\infty = 2.0$m. The remaining boundaries (top) and (bottom) are specified as symmetry conditions. To ensure the top boundary condition doesn't influence the flow solution it is located at $z = 10$ with the mesh being stretched towards the boundary. The gravity vector is specified as $\boldsymbol{g} = (0, -9.81, 0)^T$ so that gravity acts towards the lower boundary of the domain.

Code_Saturne is set up to perform 5000 iterations with a fixed time step of $\Delta t = 0.001$s, results are written for post-processing every 20 iterations. The flow domain was partitioned over 12 processors using the *Scotch* partitioner [Chevalier and Pellegrini, 2008]. Full details of the solver settings can be found in the `subcritical-weir/DATA/weir` datafile.

The computation took 3 hours 23 minutes to run.

An analysis of the `listing` file from the solver shows that the pressure equation converges in between 40 and 70 iterations of the pressure-correction solver (each time step) while the velocity and scalar advection equations typically converges in 2 or 3 iterations. This indicates that the solution is progressing efficiently and that convergence performance is as expected. Furthermore this computation show no appreciable change in the mass of water in the domain from time-step to time step, once the solution has converged and the depression has formed.

Figure 4.10 shows the computed velocity field and the free surface location in the neighbourhood of the bump, $5 < x < 15$. The velocity vectors show the expected increase in velocity at the flow accelerates over the bump and the expected decrease in surface elevation, centred over the crest of the bump. In the air (immediately above the free surface) the local velocity drops as the air expands into the space formed by the depression in the free surface. After the bump, the free surface recovers to the steady upstream conditions. A careful examination of the free surface shows a slight increase in surface elevation, prior to the bump, and a small secondary wave just downstream of the bump. These features are not present in the shallow water solution and can be attributed directly to the fact that the *Code_Saturne* solution is non-hydrostatic, whereas the analytical solution is based on a hydrostatic assumption.

Figure 4.11 provides a direct comparison between the analytical solution and the results from the *Code_Saturne* simulation. These results show that *Code_Saturne* accurately predicts the start and end of the depression over the bump, but that the depth of the depression is under predicted. The slight rise in surface elevation before the depression and the post depression wave can also be seen.

4.3.3 Transcritical flow without a hydraulic jump

Forthe transcitical test cases, without a hydraulic jump, the initial conditions used are in the form of a *dam break problem.* Dam break problems are in effect the water flow version of the classical Riemann problem [Fraccarollo and Toro, 1995], where two regions with discontinuous flow conditions are separated by a diaphragm which is instantaneously removed at $t = 0$ (Figure 4.12). In the transcritical case the left hand region contains water to a depth of $\eta_0 = 1.0$m with an initial velocity of $u_0 = 1.53$ms^{-1}. The right hand region, and the region above the water surface on the left both contain air also moving at $u_1 = 1.53$ms^{-1}. The reference pressure is set to atmospheric pressure $p_0 = 101325$Pa, the reference density is set to $\rho = 1$kgm^{-3}, and the flow simulation is set to laminar (i.e. no turbulence model), with a molecular viscosity of, $\mu = 1.83 \times 10^{-31}$, making the solution effectively inviscid. As before the gravity vector is specified as $\boldsymbol{g} = (0, -9.81, 0)^T$ so that gravity acts towards the lower boundary of the domain.

The upstream and downstream boundary conditions are specified according the prescriptions, described above, for subcritical inflow and supercritical outflow. At the inlet a fixed mass flow rate of $Q_0 = 1.53$m^{3}s^{-1} is specified. At the outlet only the downstream

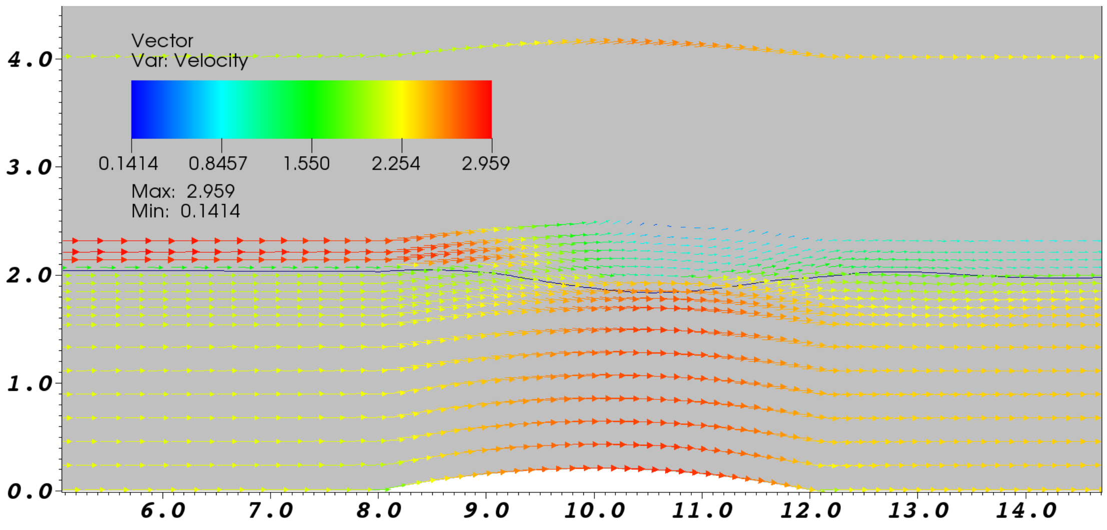


Figure 4.10: Subcritical flow over a submerged bump: computed free surface elevation (solid line) and velocity vectors

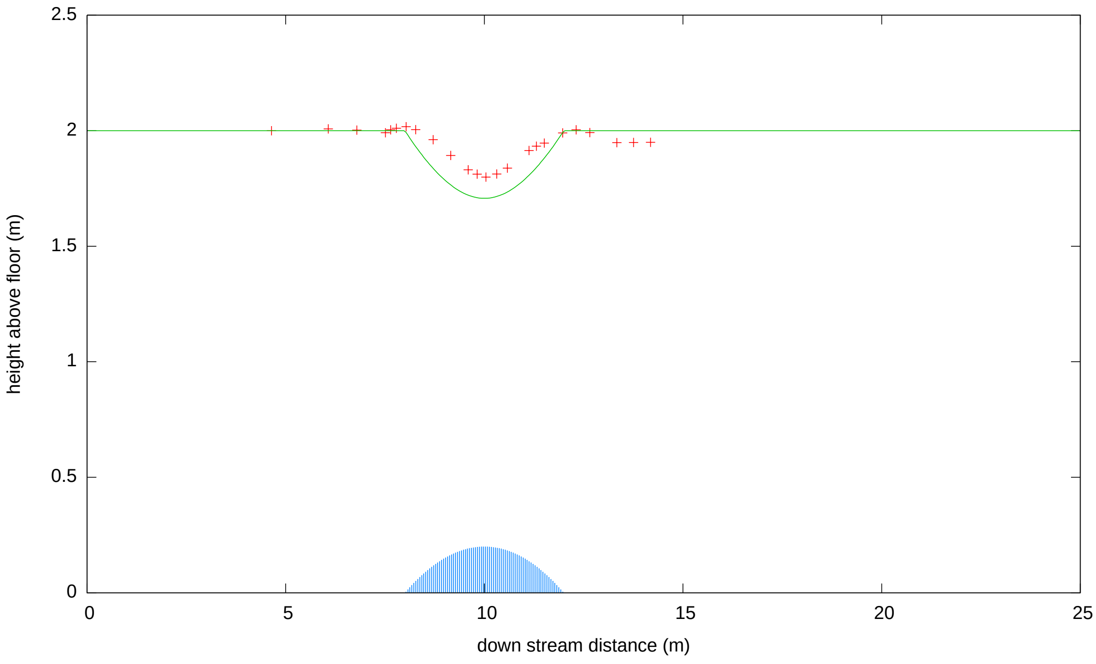


Figure 4.11: Subcritical flow over a submerged bump: Analytical free surface elevation from the shallow water solution (solid line) and computed free surface elevation using the shallow water implementation in *Code_Saturne* (crosses).

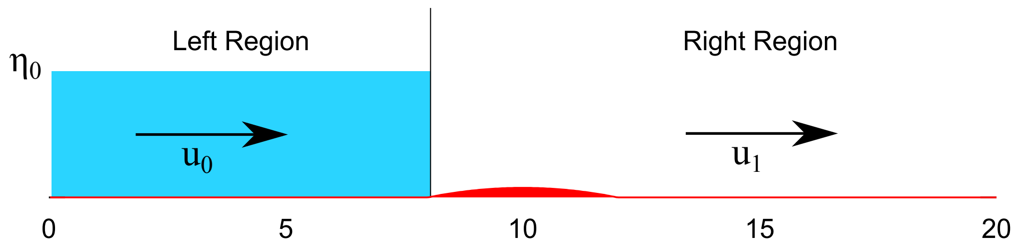


Figure 4.12: Initial conditions for a dam break problem with a dry bed on the right hand side.

pressure is used, but this does incorporate the hydrostatic component, $p_{hs} = \rho gh$,below the water surface. As in the previous case the remaining boundaries are specified as symmetry conditions with the mesh being stretched towards the top boundary.

Code_Saturne is set up to perform 60000 iterations with a fixed time step of $\Delta t = 0.001$s, results are written for post-processing every 100 iterations. The flow domain was partitioned over 12 processors using the *Scotch* partitioner [Chevalier and Pellegrini, 2008]. Full details of the solver settings can be found in the `transcritical-weir/DATA/weir` datafile.

The computation took 2 hours 45 minutes to run with no convergence issues occurring in the pressure correction algorithm.

Figure 4.13 shows the computed velocity field in the neighbourhood of the bump and the free surface location at the end of the simulation. The free surface shows that a hydraulic depression has formed over the bump, which fails to recover after passing the crest. This is characteristic of a depression in which the flow becomes supercritical. Indeed, the velocity vectors show that the water continues to accelerate on the downstream side of the bump and that a steady, uniform, velocity field exists down stream of the bump where the flow is super-critical. As expected the velocity in the air decreases down stream as the reduction in the free surface height provides an enlarged volume for the airflow.

Figure 4.14 compares the location of the free surface predicted by *Code_Saturne* with that from the analytical solution for the inviscid shallow water equations. As in the subcritical case there are small differences between the surface elevation through the depression, though in this case the start and end locations for the depression are in good agreement. It is again important to note that the shallow water solution is based on a hydrostatic approximation, while the *Code_Saturne* simulations are not and this may account for the observed differences.

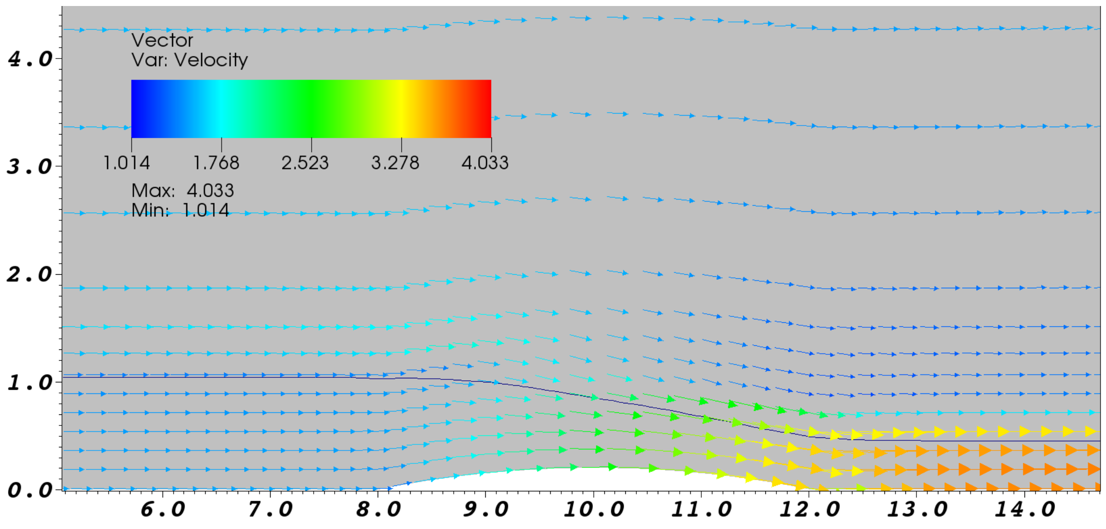


Figure 4.13: Transcritical flow over a submerged bump (without a hydraulic jump): computed free surface elevation (solid line) and velocity vectors.

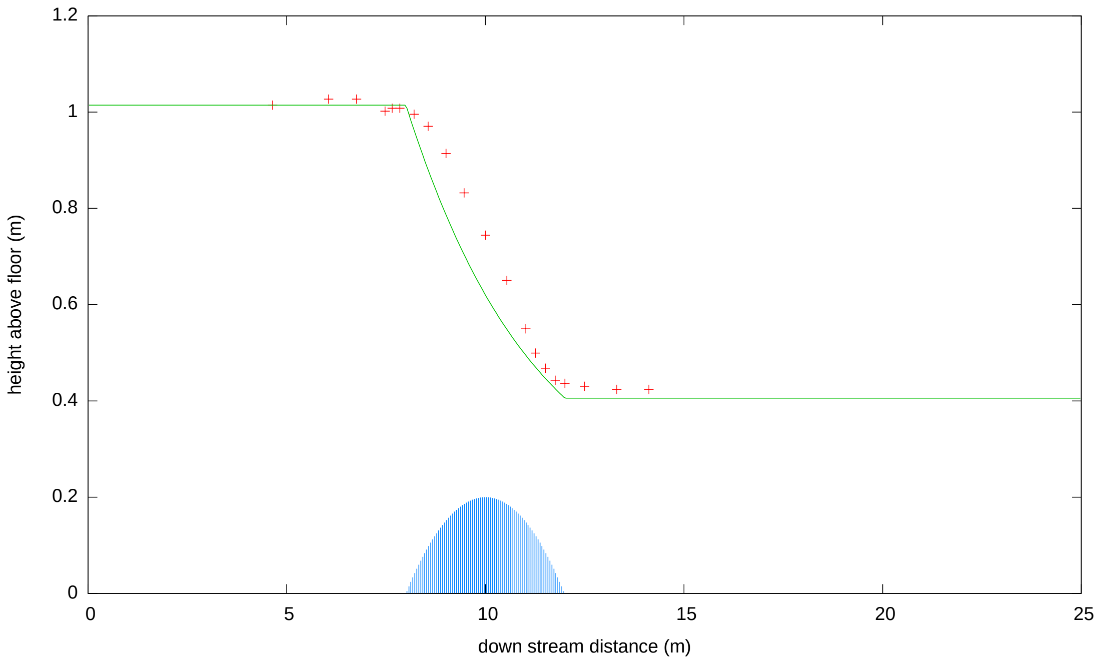


Figure 4.14: Transcritical flow over a submerged bump (without a hydraulic jump): Analytical free surface elevation from the shallow water solution (solid line) and computed free surface elevation using the shallow water implementation in *Code_Saturne* (crosses).

4.3.4 Transcritical flow with a hydraulic jump

The final of these three test cases is perhaps the most violent as it includes a hydraulic jump at the downstream end of the bump. In this case the down stream flow is not supercritical, but the upstream conditions are sufficient for supercritical flow to occur over the bump. This leads to an over expansion of the flow in the hydraulic depression, and the consequent formation of a hydraulic jump where the supercritical flow meets to subcritical flow. In the shallow water solution the hydraulic jump is simply an instantaneous change in water elevation, but in the *Code_Saturne* solution this region is associated with complex wave breaking phenomena. The processes occurring in this region have been the subject of a recent experimental study by Koo et al. [2012].

In this case the initial conditions for the *Code_Saturne* simulation do not make use of a dam break problem, but there is still an initial discontinuity in the flow velocities. Initial velocities of $u_0 = 0.433\text{ms}^{-1}$ and $u_1 = 0.545\text{ms}^{-1}$ are used for the left and right states, while the free surface is at a uniform height of, $\eta_0 = \eta_1 = 0.415\text{m}$. The upstream and downstream boundary conditions are set for subcritical in- and out-flow. At both the inlet and outlet a fixed discharge of $Q_0 = 0.18\text{m}^3\text{s}^{-1}$ is specified. Once again, the gravity vector is specified as $\boldsymbol{g} = (0, -9.81, 0)^T$ so that gravity acts towards the lower boundary of the domain.

Code_Saturne is set up to perform 100000 iterations with a fixed time step of $\Delta t = 0.001\text{s}$, results are written for post-processing every 100 iterations. The flow domain was partitioned over 12 processors using the *Scotch* partitioner [Chevalier and Pellegrini, 2008]. Full details of the solver settings can be found in the `supercritical-weir/DATA/weir` datafile.

The computation took 11 hours 16 minutes to run with no convergence issues occurring in the pressure correction algorithm. Upstream on the bump there is some evidence of poor convergence in the velocity field in the air, probably caused by a badly resolved recirculation which is trying to form above the bump. This feature has little impact on the velocity field in the water, but does slow convergence, causing an increase in run-time.

Figure 4.15 shows velocity vectors and the free surface position from the *Code_Saturne* simulation. At the instant depicted a breaking wave can be seen in the free surface, just at the end of the bump. This breaking wave is a transient feature which occurs at the latter stages of the formation of the hydraulic jump. After the wave has broken, the hydraulic jump will form and steepen until it exceeds the maximum wave steepness (circa 15%) at which time it will begin to plunge forming the breaking wave observed. To complete the formation-breaking-reformation process takes about one second and leads to small air bubbles being entrained in the downstream flow. Detailed resolution of this process would require a much finer grid in the region where the hydraulic jump occurs and probably a fully three dimensional, viscous simulation, using a sophisticated turbulence model.

Figure 4.16 shows the comparison between the analytical shallow water solution and the *Code_Saturne* predictions. As in the previous case the start of the depression is correctly predicted, however there is a small rise in surface elevation just before it. *Code_Saturne* also gives a good prediction for the minimum surface elevation, though its prediction for the end of the depression is very poor. The cluster of points observed is due to the free surface being multi-valued at that point (due to the presence of the breaking wave). The authors also note that the downstream surface elevation is under-predicted and this could be caused by poor resolution of the wave breaking process. In this case in particular their is doubt over whether the hydrostatic assumption in the shallow water equations is valid. To conduct a proper validation for this test case it is likely that a detailed series of physical experiments would have to be performed.

4.4 Flow over a submerged hydrofoil

Duncan [1983] conducted a series of experiments in which measurements were made of both the surface-height profile and the vertical distributions of total head and velocity behind a two-dimensional, fully submerged, hydrofoil moving at a constant speed and angle of attack. He reports that depending on the test conditions both breaking and non-breaking conditions can be found in the wave-train which forms behind the hydrofoil. While the main purpose of the experimental study was to examine how the drag force on the hydrofoil changed with breaking and non-breaking conditions, this series of experiments has been inspired a number of authors. In particular Kwag [2000]; Gretton et al. [2010], and Pascarelli et al. [2002] have used the experiments as validation data for numerical investigations.

In the experiments a NACA0012 hydrofoil was towed at a constant speed in a 24m long towing tank, with a rectangular cross section 0.61m $\times$ 0.61m. The NACA0012 hydrofoil was made of solid aluminium and had a chord length of 0.203m and a maximum thickness of 0.0254m. The free surface was visualised by injecting the water with 1.5ppm rhodamine-WT fluorescent dye and using a light sheet to illuminate the free surface. A 16mm cine-camera then recorded a "glowing line" on the water surface, Duncan [1983] reports that his surface disturbances are measured to within ± 0.003m. Figure 4.17 shows the experimental set-up, using a moving frame of reference centred on the hydrofoil.

4.4.1 Computation setup

The final test case used to validate the *Code_Saturne* level-set solver is case (c) from Duncan [1983], where $\alpha = 5°$, $h = 0.21$m and $u_0 = 0.8\text{ms}^{-1}$ (giving a Froude number for the hydrofoil of $F_R = 0.557$). This is one of the non-breaking cases and has been used by Gretton et al. [2010] to validate *Ansys CF/X*. Figure 4.18 shows the computational mesh in the neighbourhood of the hydrofoil. It should be noted that the an exceptionally fine

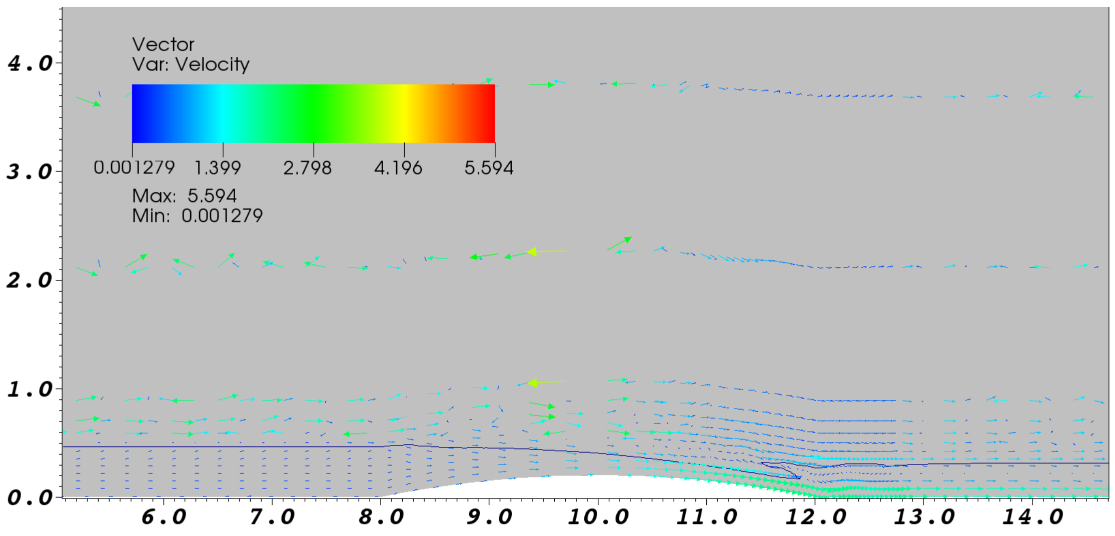


Figure 4.15: Transcritical flow over a submerged bump (with a hydraulic jump): computed free surface elevation (solid line) and velocity vectors.

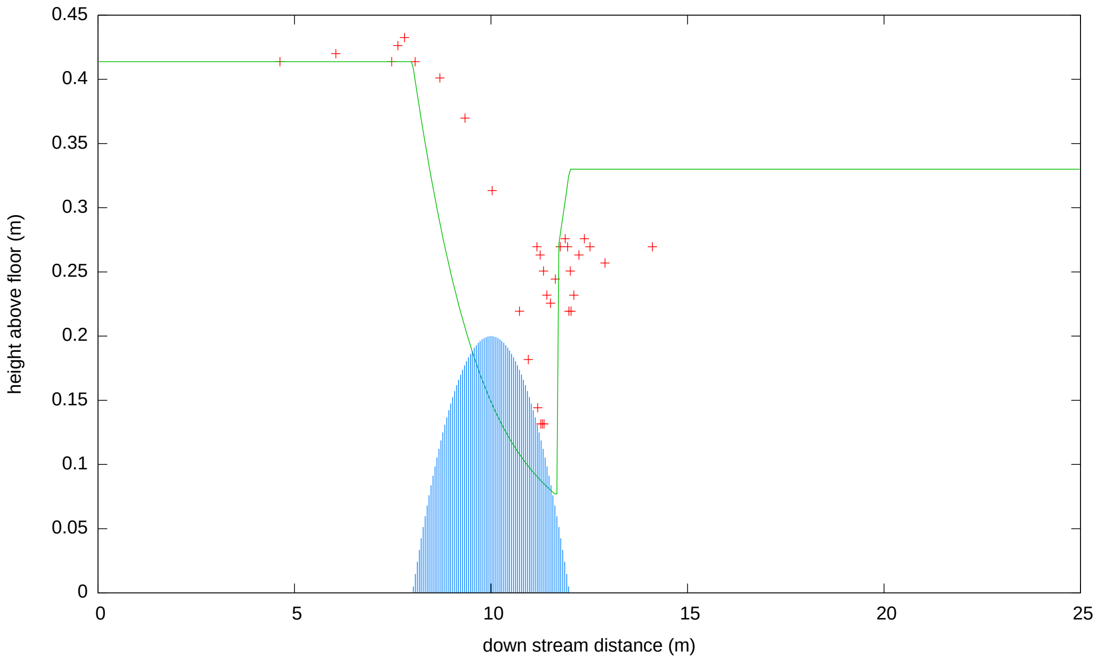


Figure 4.16: Transcritical flow over a submerged bump (with a hydraulic jump): Analytical free surface elevation from the shallow water solution (solid line) and computed free surface elevation using the shallow water implementation in *Code_Saturne* (crosses).

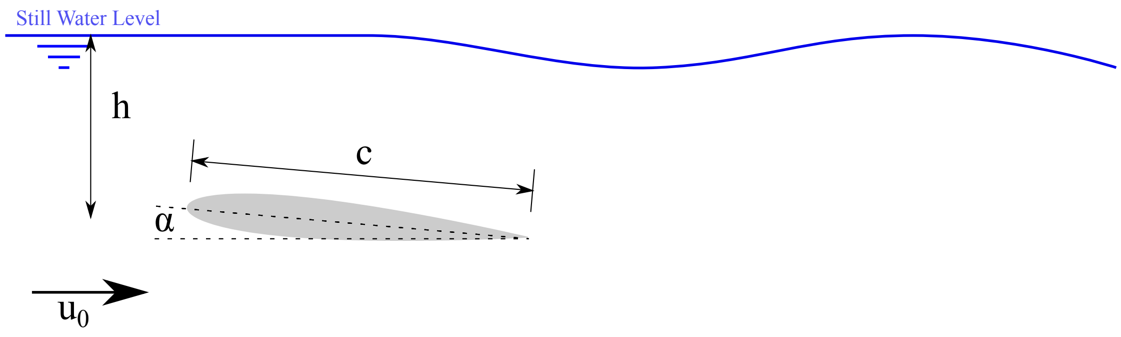


Figure 4.17: Configuration for flow over a submerged NACA0012 hydrofoil studied experimentally by Duncan [1983]. The foil (chord length, c) is submerged to a depth, h and inclined at an angle of attack, α, with a free-stream velocity of u_0.

grid is needed to resolve the boundary layer over the hydrofoil and the shear layer where the water flows over the top and bottom of the hydrofoil meet. The grid is also refined in the neighbourhood of the free surface, but to a lesser extent. The initial conditions used for the flow solver are steady, uniform, flow with a velocity of $u_0 = 0.8\text{ms}^{-1}$ and a constant uniform water depth of 0.21m above the hydrofoil. Below the hydrofoil and above the free surface the mesh is stretched towards the upper and lower symmetry boundary conditions. The reference pressure is set to atmospheric pressure (101325 Pa) and the reference density to 1kgm^{-3}. The computation is fully turbulent and uses the $k - \omega$ SST turbulence model, with a laminar viscosity of $\mu = 1.9 \times 10^{-5}$. The initial value for the turbulent kinetic energy is $k = 1.5 \times (0.01u_0)^2$ and the initial value for the dissipation is $\omega = k/1.983^5$.

The surface of the hydrofoil is modelled as a smooth no-slip wall, while the the upstream and downstream conditions are for subcritical flow with a mass flow rate of $0.7625\text{m}^3\text{s}^{-1}$. The inlet turbulence conditions are computed using a fully developed profile with a hydraulic diameter of 1. Full details of the setup can be found in `hydrofoil/DATA/hydrofoil_medium`.

The simulation was run for 13500 iterations, with a fixed time step of $\Delta t = 0.001$s, on a grid of 565184 elements. The solution is output every 500 iterations. It was run in parallel across 12 processors and the calculation took 2 days 11 hours and 4 minutes.

4.4.2 results

Figure 4.19 shows the streamlines in the water from the *Code_Saturne* simulations. The results clearly show the acceleration of the water over the upper surface of the hydrofoil and the deceleration of the flow near the leading edge of the airfoil and on the lower surface. In the zone immediately below the water surface the streamlines bend to follow the surface, but this effect quickly decays with water depth. The figure also clearly shows

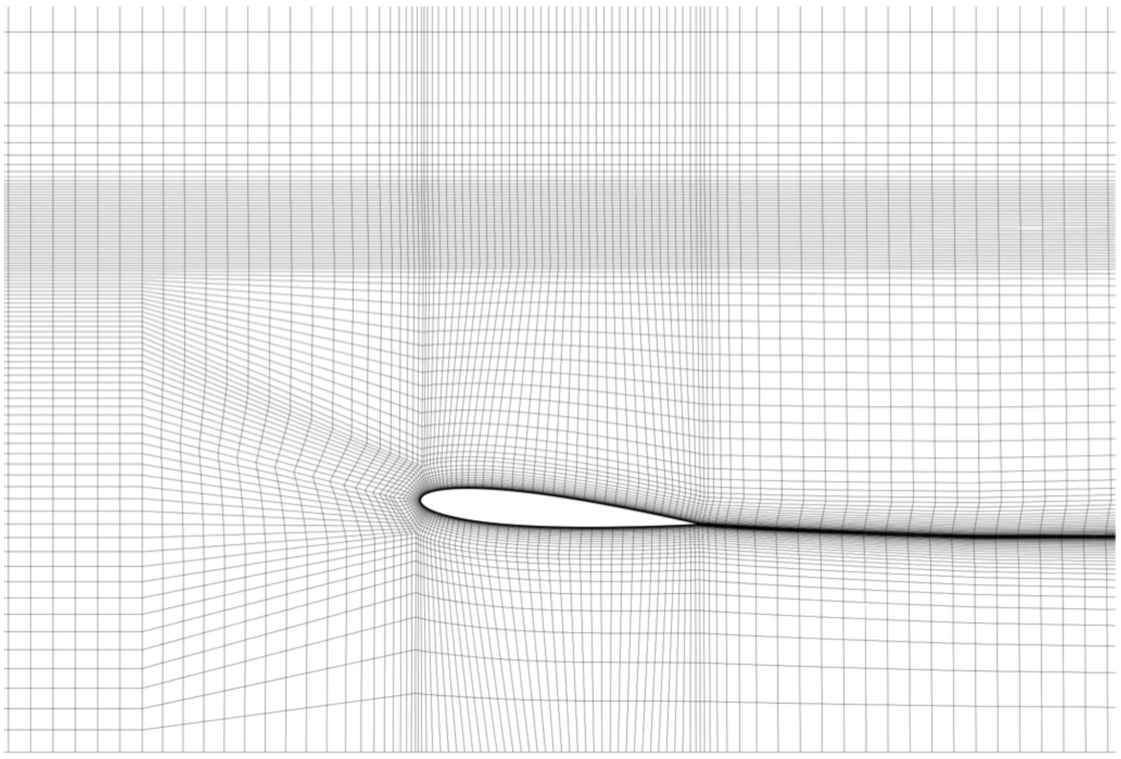


Figure 4.18: Flow over a submerged NACA0012 hydrofoil at an angle of attack, $\alpha = 5°$: Computational grid in the neighbourhood of the hydrofoil

a train of waves forming behind the hydrofoil and the formation of a depression above the foil. This depression forms in the same way as the depression for the subcritical bump case (§4.3.2), where the acceleration of the water over the hydrofoil leads to a local drop in pressure.

As the simulation proceeds the amplitude of the waves downstream from the hydrofoil increases. At the time shown ($t = 6$s) the undular wake is clearly visible for the first 1.5m downstream. Comparing the free surface with that measured by Duncan [1983] (Figure 4.20) shows that the period of the wake oscillation is accurately predicted, the prediction of the maximum deformation for the first and second waves is also close to the experimental measurements, the minimum value is under predicted by *Code_Saturne* as is the depth of the depression over the hydrofoil. Gretton et al. [2010] also found that *Ansys CF/X* under predicted the magnitude of the waves (see Figure 4.21). The results presented in Figure 4.20 show that *Code_Saturne* is performing as well as *Ansys CF/X*.

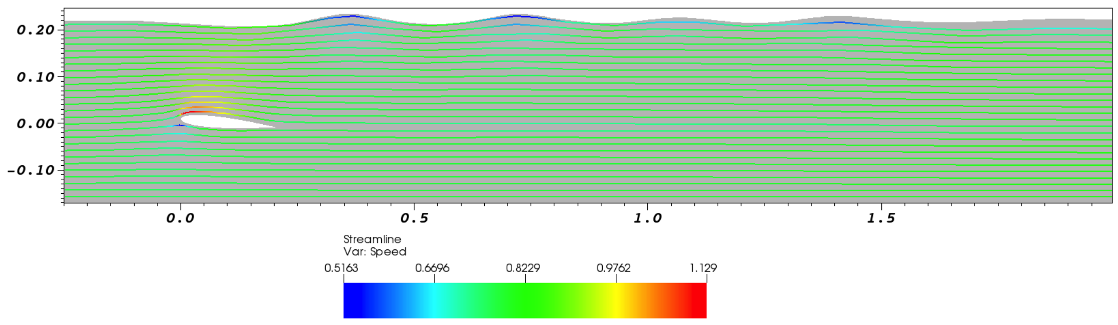


Figure 4.19: Flow at 0.8ms^{-1} over a submerged NACA0012 hydrofoil at an angle of attack, $\alpha = 5°$: Underwater streamlines (coloured by velocity) in the neighbourhood of the hydrofoil.

4.5 Conclusions

The level-set formulation of *Code_Saturne* described in Chapters 2 and 3, has been subjected to five test cases

1. Low amplitude sloshing,
2. Subcritical flow over a submerged bump,
3. Transcritical flow over a submerged bump (without a hydraulic jump),
4. Transcritical flow over a submerged bump (with a hydraulic jump), and
5. Flow over a submerged hydrofoil.

Case 1 is compared with a semi-analytical solution. Cases 2–4 have been compared with an analytical solution for the depth averaged equations, which provides a good approximation of the real 3D flows. Finally, case 5, is compared with experimental results obtained by Duncan [1983] and to numerical simulations performed using CF/X by Gretton et al. [2010].

The validation cases show that *Code_Saturne* show that the level-set formulation implemented within *Code_Saturne* is reproducing the expected physics for these test cases and also that the implementation is stable. The performance on Cases 1 and 5 is similar to that from other published CFD simulations. The slight mass loss in Case 1 is attributed to *Code_Saturne* pressure solver not having inflow and outflow boundaries to use to control mass in the domain and this could be a problem for other test cases which used a *closed box* domain. Cases 2 and 3 show good predictions of the location of the centred depression which forms over the bump. These simulations would benefit from a more detailed mesh refinement study, though this may show convergence to a non-physical solution (as found by Gretton et al. [2010] in comprehensive CF/X study of Case 5).

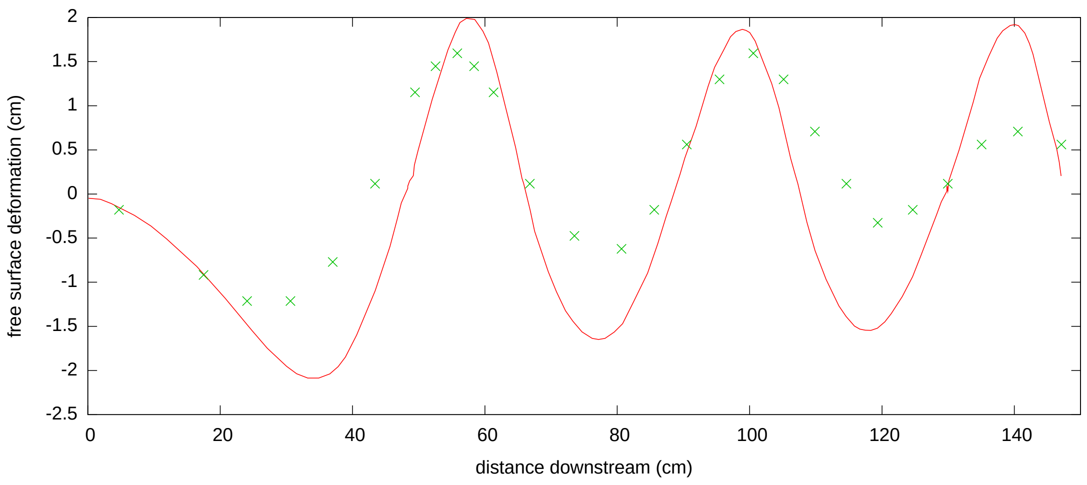


Figure 4.20: Flow at 0.8ms^{-1} over a submerged NACA0012 hydrofoil at an angle of attack, $\alpha = 5°$: Comparison between experimental measurements by Duncan [1983] (solid line) and predicted (crosses) free surface elevation.

The results for case 4 are much more mixed and a rather confused by the formation of a breaking wave at the hydraulic jump. A much more detailed, fully three dimensional, investigation of this case should be undertaken and ought to include an investigation of whether implementing a physically more accurate free-surface boundary conditions at the water surface would further stabilise the simulation. A further area for investigation in such a study is the role of turbulence models in wave breaking simulations - it would be extremely interesting to know if Reynolds averaging disrupts the wave breaking process, so large eddy simulations (LES) must be performed.

In the bump (cases 2–4) and hydrofoil (case 5) simulations *Code_Saturne* under predicts the head drop associated with flow passing over the obstruction. Tests with other commercial flow solvers (such as *Ansys CF/X*) have shown similar behaviour. It may be that further mesh refinement will improve the resolution of this feature, though a comprehensive mesh refinement study by Gretton et al. [2010] on the hydrofoil test case showed that the free surface predictions converged to their own limits rather than towards the experimental measurements.

In concluding it is clear that whilst a level-set method has been successfully implemented within *Code_Saturne* further work is needed to determine the level of mesh refinement needed (in the location of the free surface) for high quality simulations and to explore the

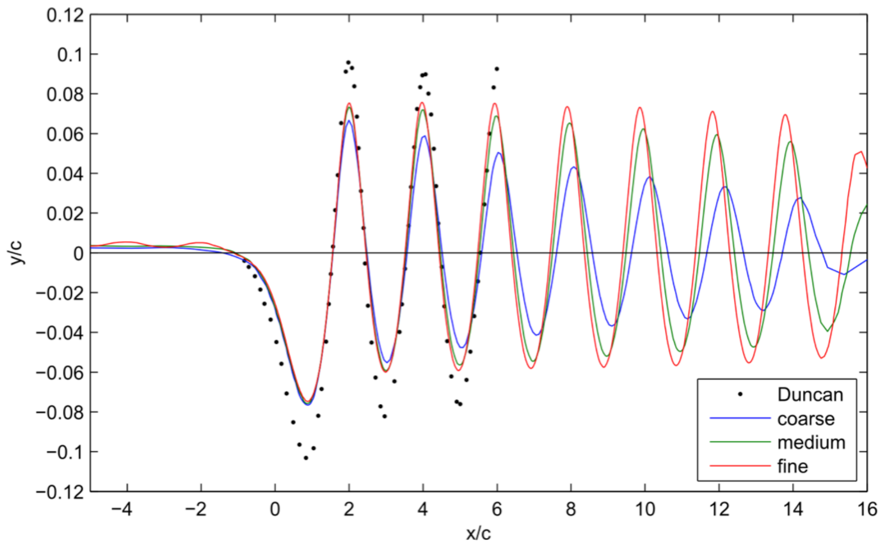


Figure 4.21: Flow at 0.8ms^{-1} over a submerged NACA0012 hydrofoil at an angle of attack, $\alpha = 5°$: Comparison between experimental measurements by Duncan [1983] and *Ansys CF/X* predictions by Gretton et al. [2010]

impact on simulations of implementing more accurate free-surface boundary conditions on the water surface. Further benefit could be obtained from implementing a seaward boundary which admits both regular and irregular waves to the domain (c.f. Ingram et al. [2009]). Such boundary conditions can be coded so that waves can be specified using either an elevation time-series, or a spectral model of the incoming waves, they have been used in both VoF and free-surface-capturing solves to create numerical wave basins and flumes which have been applied to coastal engineering and naval hydrodynamics problems.

Such boundary conditions combined with the turbine models described in Ingram and Olivieri [2012] and Olivieri and Ingram [2012] would allow simulations of wave interactions with tidal current turbines and their support structures to be performed. The computational cost of such simulations would be very large and require the use of National high performance computing systems, and even then would allow only a tens of wave interactions to be simulated. The results of these simulations would, however, provide information on how turbine wakes behave in the presence of waves and how wave fields modify the wake recovery. They would also provide useful information on how blade and turbine fatigue life is affected by surface waves.

References

Archambeau, F., Mechitoua, N., and Sakiz, M. (2004). Code_Saturne: A finite volume code for the computation of turbulent incompressible flow – Industrial applications. *International Journal on Finite Volumes*, 1(1):62 pages, electronic journal (www.ijfv.org).

Ausas, R., Dari, E., and Buscaglia, G. (2011). A geometric mass preserving redistancing scheme for level set function. *International Journal for Numerical Methods in Fluids*, 65(8):989–1010.

Beddhu, M., Taylor, L., and Whitfield, D. (1994). A time accurate calculation procedure for flows with a free surface using a modified artificial compressibility formulation. *Applied Mathematics and Computation*, 65(1):33–48.

Bhinder, M., Mingham, C., Causon, D., Rahmati, M., Aggidis, G., and Chaplin, R. (2009). A joint numerical and experimental study of a surging point absorbing wave energy converter (wraspa). In *Proceedings of ASME 28th International Conference on Ocean, Offshore and Arctic Engineering*, pages 1–7.

Chang, Y., Hou, T., Merriman, B., and Osher, S. (1996). A level set formulation of Eulerian interface capturing methods for incompressible fluid flows. *Journal of Computational Physics*, 124:449–464.

Chevalier, C. and Pellegrini, F. (2008). PT-SCOTCH: a tool for efficient parallel graph ordering. *Parallel Computing*, 34(6–8):318–331.

Chopp, D. (1993). Computing minimal surfaces via level set curvature flow. *Jounral of Computational Physics*, 106:77–91.

Chorin, A. (1967). The numerical solution of the Navier-Stokes equations for an incompressible fluid. Report NYO-1480-82, New York University.

Delestre, O., Lucas, C., Ksinant, P.-A., Darboux, F., Laguerre, C., Vo, T., James, F., and Cordier, S. (2013). SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies. *International Journal for Numerical Methods in Fluids*, 72(3):269–300.

Desjardins, O., Moureau, V., and Pitsch, H. (2008). The ghost fluid method for numerical treatment of discontinuities and interfaces. *Journal of Computational Physics*, 227:8395—8416.

Duncan, J. (1983). The breaking and non-breaking wave resistance of a two-dimensional hydrofoil. *Journal of Fluid Mechanics*, 126:507–520.

EDF R&D (2013a). *Code_Saturne 3.0.0 Pracical User's Guide.* Fluid Dynamics, Power Generation and Environment Department, Single Phase Thermal-Hydraulics Group, Electicité de France R & D, 6, quai Watier F-78401 Chatou Cedex.

EDF R&D (2013b). *Code_Saturne 3.0.0 Theory Guide.* Fluid Dynamics, Power Generation and Environment Department, Single Phase Thermal-Hydraulics Group, Electicité de France R & D, 6, quai Watier F-78401 Chatou Cedex.

Edwards, W. (2011). *Towards a Level Set Reinitialisation Method for Unstructured Grids.* PhD thesis, The University of Edinburgh, School of Engineering, King's Buildings, Edinburgh EH9 3JL.

Fraccarollo, L. and Toro, E. (1995). Experimental and numerical assessment of the shallow water model for two dimensional dam-break type problems. *Journal of Hydraulic Research*, 33(6):843–864.

Gao, F., Ingram, D., Causon, D., and Mingham, C. (2007). The development of a Cartesian cut cell method for incompressible viscous flows. *International Journal for Numerical Methods in Fluids*, 54:1033–1063.

Godunov, S. (1959). A difference scheme for numerical computation of discontinuous solution of hydrodynamic equations. *Math. Sbornik*, 47:271–306. Translated US Joint Publication Research Service, JPRS 7226 (1969).

Goutal, N. and Maurel, F., editors (1997). *Proceedings of the 2nd workshop on dam-break wave simulation.* Département Laboratoire National d'Hydraulique, Groupe Hydraulique Fluviale Electricité de France.

Gregory, J. and Redmond, D. (1994). *Introduction to Numerical Analysis.* Jones and Bartlett, London.

Gretton, G., Bryden, I., Couch, S., and Ingram, D. (2010). The CFD simulation of a lifting hydrofoil in close proximity to a free surface. In *ASME 2010 29th International Conference on Ocean, Offshore and Arctic Engineering*, volume 6 (CFD and VIV), pages 875–883. ASME.

Harlow, F. (1957). Hydrodynamic problems involving large fluid distortions. *Journal of the Association for Computing Machinery*, 4(2):137–142.

Hirt, C. and Nichols, B. (1981). Volume of Fluid (VoF) methods for dynamics of free boundaries. *Journal of Computational Physics*, 39:201–225.

Ingram, D., Causon, D., Bruce, T., Pearson, J., Gao, F., and Mingham, C. (2005). Numerical and experimental predictions of overtopping volumes for violent overtopping events. In Melby, J., editor, *Coastal Structures '03*, pages 631–642. ASCE.

Ingram, D., Gao, F., Causon, D., Mingham, C., and Troch, P. (2009). Numerical investigations of wave overtopping at coastal structures. *Coastal Engineering*, 56(2):190–202.

Ingram, D. and Olivieri, D. (2012). Tidal array scale numerical modelling: Interactions within a farm (steady flow). PerAWaT MA1003 Deliverable WG3 WP2 D5a, Energy Technologies Institute.

Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):395–392.

Kelecy, F. and Pletcher, R. (1997). The development of a free surface capturing approach for multidimensional free surface flows in closed containers. *Journal of Computational Physics*, 138:939–980.

Koo, B., Wang, Z., Yang, J., and Stern, F. (2012). Impulsive plunging wave breaking downstream of a bump in a shallow water flume— Part II: Numerical simulations. *Journal of Fluids and Structures*, 32(0):121–134.

Kwag, S.-H. (2000). Computation of water and air flow with submerged hydrofoil by interface capturing method. *KSME International Journal*, 14(7):789–795.

Lafaurie, B., Nardone, C., Scardovelli, R., Zaleski, S., and Zanetti, G. (1994). Modelling merging and fragmentation in multiphase flows with SURFER. *Journal of Computational Physics*, 113:134–147.

Li, S., Shimuta, M., and Xiao, F. (2005). A 4-th order and single-cell-based advection scheme on unstructured grids with multi-moments. *Computer Physics Communications*, 173:17–33.

Li, T., Troch, P., and De Rouck, J. (2004). Wave overtopping over a sea dike. *Journal of Computational Physics*, 198:686–726.

LNLL (2005). VisIt user's manual. Technical Report UCLR-SM-220449, Lawrence Livermore National Laboratories.

Losasso, F., Talton, J., Kwatra, N., and Fedkiw, R. (2008). Two-way coupled SPH and particle level set fluid simulation. *IEEE Transactions on Visualisation and Computer Graphics*, doi:10.1109/TVCG.2008.37 doi:10.1109/TVCG.2008.37.

Maguire, A. (2011). *Hydrodynamics, control and numerical modelling of absorbing wavemakers.* PhD thesis, The University of Edinburgh, School of Engineering, King's Buildings, Edinburgh EH9 3JL.

Maguire, A. and Ingram, D. (2010). Wavemaking in a commercial CFD code. In *Proceedings on the Third International Conference of the Application of Physical Modelling to Port and Coastal Protection.*

Moulinec, C., Denis, C., Pham, C.-T., Rougé, D., Hervouet, J.-M., Razafindrakoto, E., Barber, R., Emerson, D., and Gu, X.-J. (2011). TELEMAC: An efficient hydrodynamics suite for massively parallel architectures. *Computers and Fluids*, 15(1):30–34.

Mut, F., Buscaglia, G., and Dari, E. (2006). New mass-conserving algorithm for level set redistancing on unstructured meshes. *Journal of Applied Mechanics*, 73:1011–1016.

Olivieri, D. and Ingram, D. (2010). Tidal array scale numerical modelling: Development of the level set free surface model. PerAWaT MA1003 Deliverable WG3 WP2 D1, Energy Technologies Institute.

Olivieri, D. and Ingram, D. (2012). Tidal array scale numerical modelling: Interactions within a farm (unsteady flow). PerAWaT MA1003 Deliverable WG3 WP2 D5b, Energy Technologies Institute.

Osher, S. and Fedkiw, R. (2003). *Level Set Methods and Dynamic Implicit Surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, New York.

Osher, S. and Sethian, J. (1998). Fronts propagating with curvature-development speed: Algorithm based on Hamilton-Jacobi formulations. *Jounral of Computational Physics*, 79:12–49.

Pascarelli, A., Iaccarino, G., and Fatica, M. (2002). Towards the LES of flow past a submerged hydrofoil. In Moin, P., Mansour, N., and Bradshaw, P., editors, *Proceedings of the Summer Program 2002*, pages 169–176. Centre for Turbulence Research, Stanford University.

Peng, D., Merriman, B., Osher, S., Zhao, H., and Kang, M. (1999). A PDE-based fast local level set method. *Journal of Computational Physics*, 155:410–438.

Peregrine, D. (1972). Equations for water waves and the approximations behind them. In Meyer, R., editor, *Waves on Beaches and Resulting Sediment Transport*, pages 95–122. Accademic Press.

Qian, L., Causon, D., Ingram, D., and Mingham, C. (2003). A Cartesian cut cell two-fluid solver for hydraulic flow problems. *ASCE Journal of hydraulic Engineering*, 129(9):688–696.

Qian, L., Causon, D., Mingham, C., and Ingram, D. (2006). A free-surface capturing method for two fluid flows with moving bodies. *Proceedings of the Royal Society: A*, 462 (2065):21–42.

Rhie, C. and Chow, W. (1983). Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532.

Saruwatari, A., Watanabe, Y., and Ingram, D. (2009). Scarifying and fingering surfaces of plunging jets. *Coastal Engineering*, 56(11-12):1109–1122.

Scardovelli, R. and Zaleski, S. (1999). Direct numerical simulation of free-surface and interfacial flows. *Annual Reviews of Fluid Mechanics*, 31(567–603).

Sethian, J. (1999). *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 2nd edition edition.

Shang, Z., Moulinec, C., Emerson, D., and Gu, X. (2011). Porting and optimisation of Code_Saturne on HECToR. Technical report, Computational Science and Engineering Department, Daersbury Laboratory.

Squillacote, A. (2008). *ParaView Guide*. Kitware, 3rd edition.

Sussman, M. and Puckett, E. (2000). A coupled level set and volume-of-fluid method for computing 3D and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162:301–337.

Sussman, M., Smereka, P., and Osher, S. (1994). A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114:146–159.

Tadjbakhsh, I. and Keller, J. B. (1960). Standing surface waves of finite amplitude. *Journal of Fluid Mechanics*, 8(3):442–451.

Thompson, K. (1990). Time-dependent boundary conditions for hyperbolic systems, II. *Journal of Computational Physics*, 89:439–461.

Troch, P., Li, T., De Rouke, J., and Ingram, D. (2003). Wave interaction with a sea dike using a VOF finite-volume method. In Chung, J., Prevosto, M., Mizutani, N., Kim, G., and Grilli, S., editors, *Proceedings of the 13th International Offshore and Polar Engineering Conference*, volume 3, pages 325–332. International Society of Offshore and Polar Engineering.

Ubbink, O. (1997). *Numerical prediction of two fluid systems with sharp interface*. PhD thesis, Imperial College, London.

Ubbink, O. and Issa, R. (1999). A method for capturing sharp fluid interfaces on arbitrary meshes. *Journal of Computational Physics*, 153:26–50.

Vázquez-Cendón, M. (1999). Improved treatment of source terms in upwind schemes for the shallow water equations in channels with irregular geometry. *Journal of Computational Physics*, 148:497–526.

Watanabe, Y., Saruwatari, A., and Ingram, D. (2008). Free-surface flows under impacting droplets. *Journal of Computational Physics*, 227(4):2344–2365.

Youngs, D. (1982). Time dependent multi-material flow with large fluid distortion. In Morton, K. and Baines, M., editors, *Numerical Methods for fluid dynamics*, pages 237–285, London. Academic Press.

Zhou, J., Causon, D., Mingham, C., and Ingram, D. (2001). The surface gradient method for the treatment of source terms in the shallow water equations. *Journal of Computational Physics*, 168:1–25.

Appendix A

File structure on the FTP site

All the results files contain the last time step from the *Code_Saturne* simulations and are written out in Ensite GOLD format. They can be visualised using Paraview, ViSit, or EnSight Gold and many other visualisation tools.

The FTP site contains the following files. Each of the *Code_Saturne* study directories contains the required data files and scripts to run the cases and a results (RESU) directory containing the output every 100 time steps for each simulation.

```
|-- MESH
| |-- 200_200_mesh31113.med
| |-- Weir_24113.med
| |-- Weir_shock_30113.med
| |-- Weir_subcritical_12213.med
| `-- hydrofoil_fine_25_2_13.med
|-- POST
|-- hydrofoil
| |-- DATA
| | |-- REFERENCE
| | | |-- cs_user_scripts.py
| | | |-- dp_C3P
| | | |-- dp_C3PSJ
| | | |-- dp_C4P
| | | |-- dp_ELE
| | | |-- dp_FCP
| | | |-- dp_FCP.xml
| | | |-- dp_FCP_new
| | | |-- dp_FUE
| | | |-- dp_FUE_new
| | | |-- dp_transfo
| | | `-- meteo
| | |-- SaturneGUI
| | `-- hydrofoil_medium
| |-- RESU
| | |-- 20130521-1152
| | | |-- Tide_level_results.dat
| | | |-- checkpoint
| | | | |-- auxiliary
| | | | `-- main
| | | |-- compile.log
| | | |-- cs_solver
| | | |-- error
| | | |-- hydrofoil_medium
| | | |-- massflow.dat
| | | |-- mesh_input
| | | |-- partition_output
| | | | `-- domain_number_12
| | | |-- performance.log
| | | |-- postprocessing
| | | | |-- ERROR.case
| | | | |-- RESULTS.case
| | | | |-- error.diag_dom_scalar2
| | | | |-- error.diag_scalar2
| | | | |-- error.geo
| | | | |-- error.residual_scalar2
| | | | |-- error.rhs_scalar2
| | | | |-- error.x_scalar2
| | | | |-- results.courantnb.00001 to 00027
| | | | |-- results.density.00001 to 00027
| | | | |-- results.distwall.00001 to 00027
| | | | |-- results.efforts.00001 to 00027
| | | | |-- results.geo
| | | | |-- results.lamvisc.00001 to 00027
| | | | |-- results.omega.00001 to 00027
| | | | |-- results.parallel_domain
| | | | |-- results.pressure.00001 to 00027
| | | | |-- results.scalar2.00001 to 00027
| | | | |-- results.turbener.00001 to 00027
| | | | |-- results.turbvisc.00001 to 00027
| | | | |-- results.velocity.00001 to 00027
| | | | |-- results.yplus.00001 to 00027
| | | |-- preprocessor.log
| | | |-- run_solver.sh
| | | |-- setup.log
| | | |-- src_saturne
| | | | |-- aa_connectivity.f90
| | | | |-- cs_user_boundary_conditions.f90
| | | | |-- cs_user_extra_operations.f90
| | | | |-- cs_user_initialization.f90
| | | | |-- cs_user_physical_properties.f90
| | | | |-- iniini.f90
| | | | `-- ulevelset.f90
| | | |-- summary
| | | `-- zerolevel_contour.dat
| | `-- check_mesh
| |     |-- check_mesh.log
| |     |-- performance.log
| |     |-- postprocessing
| |     | |-- BOUNDARY_GROUPS.case
| |     | |-- MESH_GROUPS.case
| |     | |-- QUALITY.case
| |     | |-- boundary_groups.geo
| |     | |-- mesh_groups.geo
| |     | |-- quality.bad_cell_lsq_gradient
| |     | |-- quality.bad_cell_ortho_norm
| |     | |-- quality.cell_volume
| |     | |-- quality.err_grad_lsq
| |     | |-- quality.err_grad_lsq_ext
| |     | |-- quality.err_grad_lsq_extred
| |     | |-- quality.err_grad_lsq_rc
| |     | |-- quality.err_grad_rc
| |     | |-- quality.face_warp
| |     | |-- quality.face_warp_c_max
| |     | |-- quality.face_warp_v_max
| |     | |-- quality.geo
| |     | |-- quality.grad_lsq
| |     | |-- quality.grad_lsq_ext
| |     | |-- quality.grad_lsq_extred
| |     | |-- quality.grad_lsq_rc
| |     | |-- quality.grad_rc
| |     | |-- quality.non_ortho
| |     | |-- quality.non_ortho_c_max
| |     | |-- quality.non_ortho_v_max
| |     | |-- quality.offset_c_max
| |     | |-- quality.offset_v_max
| |     | |-- quality.weighting_c_max
| |     | `-- quality.weighting_v_max
| |     `-- setup.log
| |-- SCRIPTS
| | `-- runcase
| `-- SRC
|     |-- EXAMPLES
|     | |-- cs_user_boundary_conditions-advanced.f90
|     | |-- cs_user_boundary_conditions-atmospheric.f90
|     | |-- cs_user_boundary_conditions-auto_inlet_profile.f90
|     | |-- cs_user_boundary_conditions-base.f90
|     | |-- cs_user_boundary_conditions-compressible.f90
|     | |-- cs_user_boundary_conditions-cooling_towers.f90
|     | |-- cs_user_boundary_conditions-electric_arcs.f90
|     | |-- cs_user_boundary_conditions-electric_arcs_ieljou_3_or_4.f90
|     | |-- cs_user_boundary_conditions-fuel.f90
|     | |-- cs_user_boundary_conditions-gas_3ptchem.f90
|     | |-- cs_user_boundary_conditions-gas_ebu.f90
|     | |-- cs_user_boundary_conditions-gas_libby_williams.f90
|     | |-- cs_user_boundary_conditions-pulverized_coal.f90
|     | |-- cs_user_boundary_conditions-pulverized_coal_lagrangian.f90
|     | |-- cs_user_extra_operations-energy_balance.f90
|     | |-- cs_user_extra_operations-extract_1d_profile.f90
|     | |-- cs_user_extra_operations-force_temperature.f90
|     | |-- cs_user_extra_operations-global_efforts.f90
|     | |-- cs_user_extra_operations-parallel_operations.f90
|     | |-- cs_user_extra_operations-print_statistical_moment.f90
|     | |-- cs_user_initialization-atmospheric.f90
|     | |-- cs_user_initialization-base.f90
|     | |-- cs_user_initialization-compressible.f90
|     | |-- cs_user_initialization-cooling_towers.f90
|     | |-- cs_user_initialization-electric_arcs.f90
|     | |-- cs_user_initialization-fuel.f90
|     | |-- cs_user_initialization-gas_3ptchem.f90
|     | |-- cs_user_initialization-gas_ebu.f90
|     | |-- cs_user_initialization-gas_libby_williams.f90
|     | |-- cs_user_initialization-pulverized_coal.f90
|     | |-- cs_user_initialization-time_step.f90
|     | |-- cs_user_initialization-unified_combustion_coal.f90
|     | |-- cs_user_les_inflow-base.f90
|     | |-- cs_user_parameters-output.f90
|     | `-- cs_user_postprocess-sfc.c
|     |-- REFERENCE
|     | |-- cs_user_atmospheric_model.f90
|     | |-- cs_user_boundary_conditions.f90
|     | |-- cs_user_coupling.c
|     | |-- cs_user_extra_operations.f90
|     | |-- cs_user_fluid_structure_interaction.f90
|     | |-- cs_user_initialization.f90
|     | |-- cs_user_les_inflow.f90
|     | |-- cs_user_mesh.c
|     | |-- cs_user_modules.f90
|     | |-- cs_user_parameters.f90
|     | |-- cs_user_particle_tracking.f90
|     | |-- cs_user_performance_tuning.c
|     | |-- cs_user_physical_properties.f90
|     | |-- cs_user_postprocess.c
|     | |-- cs_user_postprocess_var.f90
|     | |-- cs_user_radiative_transfer.f90
|     | |-- cs_user_solver.c
|     | |-- cs_user_source_terms.f90
|     | |-- ulevelset.f90
|     | |-- usalcl.f90
|     | |-- usctdz.f90
|     | |-- uselrc.f90
|     | |-- ushist.f90
|     | |-- uskpdc.f90
|     | |-- uslaen.f90
|     | |-- uslag1.f90
|     | |-- uslag2.f90
|     | |-- usporo.f90
|     | |-- uspt1d.f90
|     | |-- usray1.f90
|     | |-- usray2.f90
|     | |-- usthht.f90
|     | |-- ustsma.f90
|     | |-- usvort.f90
|     | `-- usvosy.f90
|     |-- aa_connectivity.f90
|     |-- cs_user_boundary_conditions.f90
|     |-- cs_user_extra_operations.f90
|     |-- cs_user_initialization.f90
|     |-- cs_user_physical_properties.f90
|     |-- iniini.f90
|     `-- ulevelset.f90
|-- sloshing
| |-- DATA
| | |-- REFERENCE
| | | |-- cs_user_scripts.py
| | | |-- dp_C3P
| | | |-- dp_C3PSJ
| | | |-- dp_C4P
| | | |-- dp_ELE
| | | |-- dp_FCP
| | | |-- dp_FCP.xml
| | | |-- dp_FCP_new
| | | |-- dp_FUE
| | | |-- dp_FUE_new
| | | |-- dp_transfo
| | | `-- meteo
| | |-- SaturneGUI
| | `-- scalar_centred_slope
| |-- RESU
| | |-- 20130509-1718
| | | |-- Tide_level_results.dat
| | | |-- checkpoint
| | | | |-- auxiliary
| | | | `-- main
| | | |-- compile.log
| | | |-- listing
| | | |-- performance.log
| | | |-- postprocessing
| | | | |-- RESULTS.case
| | | | |-- results.courantnb.00001 to 01250
| | | | |-- results.density.00001 to 01250
| | | | |-- results.efforts.00001 to 01250
| | | | |-- results.fouriernb.00001 to 01250
| | | | |-- results.geo
| | | | |-- results.lamvisc.00001 to 01250
| | | | |-- results.pressure.00001 to 01250
| | | | |-- results.scalar1.00001 to 01250
| | | | |-- results.scalar2.00001 to 01250
| | | | |-- results.total_pressure.00001 to 01250
| | | | |-- results.velocity.00001 to 01250
| | | | |-- results.yplus.00001 to 01250
| | | |-- preprocessor.log
| | | |-- scalar_centred_slope
| | | |-- setup.log
| | | |-- sloshing.svg
| | | |-- src_saturne
| | | | |-- aa_connectivity.f90
| | | | |-- cs_user_extra_operations.f90
| | | | |-- cs_user_initialization.f90
| | | | |-- cs_user_physical_properties.f90
| | | | |-- iniini.f90
| | | | `-- ulevelset.f90
| | | |-- summary
| | | |-- tide_right_air.dat
| | | |-- total_liquid.dat
| | | `-- zerolevel_left.dat
| | `-- check_mesh
| |     |-- check_mesh.log
| |     |-- performance.log
| |     |-- postprocessing
| |     | |-- BOUNDARY_GROUPS.case
| |     | |-- MESH_GROUPS.case
| |     | |-- QUALITY.case
| |     | |-- boundary_groups.geo
| |     | |-- mesh_groups.geo
| |     | |-- quality.cell_volume
| |     | |-- quality.err_grad_lsq
| |     | |-- quality.err_grad_lsq_ext
| |     | |-- quality.err_grad_lsq_extred
| |     | |-- quality.err_grad_lsq_rc
| |     | |-- quality.err_grad_rc
| |     | |-- quality.face_warp
| |     | |-- quality.face_warp_c_max
| |     | |-- quality.face_warp_v_max
| |     | |-- quality.geo
| |     | |-- quality.grad_lsq
| |     | |-- quality.grad_lsq_ext
| |     | |-- quality.grad_lsq_extred
| |     | |-- quality.grad_lsq_rc
| |     | |-- quality.grad_rc
| |     | |-- quality.non_ortho
| |     | |-- quality.non_ortho_c_max
| |     | |-- quality.non_ortho_v_max
| |     | |-- quality.offset_c_max
| |     | |-- quality.offset_v_max
| |     | |-- quality.weighting_c_max
| |     | `-- quality.weighting_v_max
| |     `-- setup.log
| |-- SCRIPTS
| | `-- runcase
| `-- SRC
|     |-- EXAMPLES
|     | |-- cs_user_boundary_conditions-advanced.f90
|     | |-- cs_user_boundary_conditions-atmospheric.f90
|     | |-- cs_user_boundary_conditions-auto_inlet_profile.f90
|     | |-- cs_user_boundary_conditions-base.f90
|     | |-- cs_user_boundary_conditions-compressible.f90
|     | |-- cs_user_boundary_conditions-cooling_towers.f90
|     | |-- cs_user_boundary_conditions-electric_arcs.f90
|     | |-- cs_user_boundary_conditions-electric_arcs_ieljou_3_or_4.f90
|     | |-- cs_user_boundary_conditions-fuel.f90
|     | |-- cs_user_boundary_conditions-gas_3ptchem.f90
|      | |-- cs_user_boundary_conditions-gas_ebu.f90
|      | |-- cs_user_boundary_conditions-gas_libby_williams.f90
|      | |-- cs_user_boundary_conditions-pulverized_coal.f90
|      | |-- cs_user_boundary_conditions-pulverized_coal_lagrangian.f90
|      | |-- cs_user_extra_operations-energy_balance.f90
|      | |-- cs_user_extra_operations-extract_1d_profile.f90
|      | |-- cs_user_extra_operations-force_temperature.f90
|      | |-- cs_user_extra_operations-global_efforts.f90
|      | |-- cs_user_extra_operations-parallel_operations.f90
|      | |-- cs_user_extra_operations-print_statistical_moment.f90
|      | |-- cs_user_initialization-atmospheric.f90
|      | |-- cs_user_initialization-base.f90
|      | |-- cs_user_initialization-compressible.f90
|      | |-- cs_user_initialization-cooling_towers.f90
|      | |-- cs_user_initialization-electric_arcs.f90
|      | |-- cs_user_initialization-fuel.f90
|      | |-- cs_user_initialization-gas_3ptchem.f90
|      | |-- cs_user_initialization-gas_ebu.f90
|      | |-- cs_user_initialization-gas_libby_williams.f90
|      | |-- cs_user_initialization-pulverized_coal.f90
|      | |-- cs_user_initialization-time_step.f90
|      | |-- cs_user_initialization-unified_combustion_coal.f90
|      | |-- cs_user_les_inflow-base.f90
|      | |-- cs_user_parameters-output.f90
|      | `-- cs_user_postprocess-sfc.c
|      |-- REFERENCE
|      | |-- cs_user_atmospheric_model.f90
|      | |-- cs_user_boundary_conditions.f90
|      | |-- cs_user_coupling.c
|      | |-- cs_user_extra_operations.f90
|      | |-- cs_user_fluid_structure_interaction.f90
|      | |-- cs_user_initialization.f90
|      | |-- cs_user_les_inflow.f90
|      | |-- cs_user_mesh.c
|      | |-- cs_user_modules.f90
|      | |-- cs_user_parameters.f90
|      | |-- cs_user_particle_tracking.f90
|      | |-- cs_user_performance_tuning.c
|      | |-- cs_user_physical_properties.f90
|      | |-- cs_user_postprocess.c
|      | |-- cs_user_postprocess_var.f90
|      | |-- cs_user_radiative_transfer.f90
|      | |-- cs_user_solver.c
|      | |-- cs_user_source_terms.f90
|      | |-- ulevelset.f90
|      | |-- usalcl.f90
|      | |-- usctdz.f90
|      | |-- uselrc.f90
|      | |-- ushist.f90
|      | |-- uskpdc.f90
|      | |-- uslaen.f90
|      | |-- uslag1.f90
|      | |-- uslag2.f90
|     | |-- usporo.f90
|     | |-- uspt1d.f90
|     | |-- usray1.f90
|     | |-- usray2.f90
|     | |-- usthht.f90
|     | |-- ustsma.f90
|     | |-- usvort.f90
|     | `-- usvosy.f90
|     |-- aa_connectivity.f90
|     |-- cs_user_extra_operations.f90
|     |-- cs_user_initialization.f90
|     |-- cs_user_physical_properties.f90
|     |-- iniini.f90
|     `-- ulevelset.f90
|-- subcritical-weir
| |-- DATA
| | |-- REFERENCE
| | | |-- cs_user_scripts.py
| | | |-- dp_C3P
| | | |-- dp_C3PSJ
| | | |-- dp_C4P
| | | |-- dp_ELE
| | | |-- dp_FCP
| | | |-- dp_FCP.xml
| | | |-- dp_FCP_new
| | | |-- dp_FUE
| | | |-- dp_FUE_new
| | | |-- dp_transfo
| | | `-- meteo
| | |-- SaturneGUI
| | `-- weir
| |-- RESU
| | |-- 20130510-1728
| | | |-- Tide_level_results.dat
| | | |-- checkpoint
| | | | |-- auxiliary
| | | | `-- main
| | | |-- compile.log
| | | |-- listing
| | | |-- massflow.dat
| | | |-- partition_output
| | | | `-- domain_number_3
| | | |-- performance.log
| | | |-- postprocessing
| | | | |-- RESULTS.case
| | | | |-- results.courantnb.00001 to 01000
| | | | |-- results.density.00001 to 01000
| | | | |-- results.efforts.00001 to 01000
| | | | |-- results.fouriernb.00001 to 01000
| | | | |-- results.geo
| | | | |-- results.lamvisc.00001 to 01000
| | | | |-- results.parallel_domain
| | | | |-- results.pressure.00001 to 01000
| | | | |-- results.scalar1.00001 to 01000
| | | | |-- results.scalar2.00001 to 01000
| | | | |-- results.total_pressure.00001 to 01000
| | | | |-- results.velocity.00001 to 01000
| | | | |-- results.yplus.00001 to 01000
| | | |-- preprocessor.log
| | | |-- setup.log
| | | |-- src_saturne
| | | | |-- aa_connectivity.f90
| | | | |-- cs_user_boundary_conditions.f90
| | | | |-- cs_user_extra_operations.f90
| | | | |-- cs_user_initialization.f90
| | | | |-- cs_user_physical_properties.f90
| | | | |-- iniini.f90
| | | | `-- ulevelset.f90
| | | |-- summary
| | | |-- weir
| | | `-- zerolevel_contour.dat
| | |-- 20130520-1104
| | | |-- Tide_level_results.dat
| | | |-- checkpoint
| | | | |-- auxiliary
| | | | `-- main
| | | |-- compile.log
| | | |-- listing
| | | |-- massflow.dat
| | | |-- partition_output
| | | | `-- domain_number_12
| | | |-- performance.log
| | | |-- postprocessing
| | | | |-- RESULTS.case
| | | | |-- results.courantnb.00001 to 02500
| | | | |-- results.density.00001 to 02500
| | | | |-- results.efforts.00001 to 02500
| | | | |-- results.fouriernb.00001
| | | | |-- results.geo
| | | | |-- results.lamvisc.00001 to 02500
| | | | |-- results.parallel_domain
| | | | |-- results.pressure.00001 to 02500
| | | | |-- results.scalar1.00001 to 02500
| | | | |-- results.scalar2.00001 to 02500
| | | | |-- results.total_pressure.00001 to 02500
| | | | |-- results.velocity.00001 to 02500
| | | | |-- results.yplus.00001 to 02500
| | | |-- preprocessor.log
| | | |-- setup.log
| | | |-- src_saturne
| | | | |-- aa_connectivity.f90
| | | | |-- cs_user_boundary_conditions.f90
| | | | |-- cs_user_extra_operations.f90
| | | | |-- cs_user_initialization.f90
| | | | |-- cs_user_physical_properties.f90
| | | | |-- iniini.f90
| | | | `-- ulevelset.f90
| | | |-- summary
| | | |-- weir
| | | `-- zerolevel_contour.dat
| | `-- check_mesh
| |     |-- check_mesh.log
| |     |-- performance.log
| |     |-- postprocessing
| |     | |-- BOUNDARY_GROUPS.case
| |     | |-- MESH_GROUPS.case
| |     | |-- QUALITY.case
| |     | |-- boundary_groups.geo
| |     | |-- mesh_groups.geo
| |     | |-- quality.cell_volume
| |     | |-- quality.err_grad_lsq
| |     | |-- quality.err_grad_lsq_ext
| |     | |-- quality.err_grad_lsq_extred
| |     | |-- quality.err_grad_lsq_rc
| |     | |-- quality.err_grad_rc
| |     | |-- quality.face_warp
| |     | |-- quality.face_warp_c_max
| |     | |-- quality.face_warp_v_max
| |     | |-- quality.geo
| |     | |-- quality.grad_lsq
| |     | |-- quality.grad_lsq_ext
| |     | |-- quality.grad_lsq_extred
| |     | |-- quality.grad_lsq_rc
| |     | |-- quality.grad_rc
| |     | |-- quality.non_ortho
| |     | |-- quality.non_ortho_c_max
| |     | |-- quality.non_ortho_v_max
| |     | |-- quality.offset_c_max
| |     | |-- quality.offset_v_max
| |     | |-- quality.weighting_c_max
| |     | `-- quality.weighting_v_max
| |     `-- setup.log
| |-- SCRIPTS
| | `-- runcase
| `-- SRC
|     |-- EXAMPLES
|     | |-- cs_user_boundary_conditions-advanced.f90
|     | |-- cs_user_boundary_conditions-atmospheric.f90
|     | |-- cs_user_boundary_conditions-auto_inlet_profile.f90
|     | |-- cs_user_boundary_conditions-base.f90
|     | |-- cs_user_boundary_conditions-compressible.f90
|     | |-- cs_user_boundary_conditions-cooling_towers.f90
|     | |-- cs_user_boundary_conditions-electric_arcs.f90
|     | |-- cs_user_boundary_conditions-electric_arcs_ieljou_3_or_4.f90
|     | |-- cs_user_boundary_conditions-fuel.f90
|     | |-- cs_user_boundary_conditions-gas_3ptchem.f90
|     | |-- cs_user_boundary_conditions-gas_ebu.f90
|     | |-- cs_user_boundary_conditions-gas_libby_williams.f90
|     | |-- cs_user_boundary_conditions-pulverized_coal.f90
|      | |-- cs_user_boundary_conditions-pulverized_coal_lagrangian.f90
|      | |-- cs_user_extra_operations-energy_balance.f90
|      | |-- cs_user_extra_operations-extract_1d_profile.f90
|      | |-- cs_user_extra_operations-force_temperature.f90
|      | |-- cs_user_extra_operations-global_efforts.f90
|      | |-- cs_user_extra_operations-parallel_operations.f90
|      | |-- cs_user_extra_operations-print_statistical_moment.f90
|      | |-- cs_user_initialization-atmospheric.f90
|      | |-- cs_user_initialization-base.f90
|      | |-- cs_user_initialization-compressible.f90
|      | |-- cs_user_initialization-cooling_towers.f90
|      | |-- cs_user_initialization-electric_arcs.f90
|      | |-- cs_user_initialization-fuel.f90
|      | |-- cs_user_initialization-gas_3ptchem.f90
|      | |-- cs_user_initialization-gas_ebu.f90
|      | |-- cs_user_initialization-gas_libby_williams.f90
|      | |-- cs_user_initialization-pulverized_coal.f90
|      | |-- cs_user_initialization-time_step.f90
|      | |-- cs_user_initialization-unified_combustion_coal.f90
|      | |-- cs_user_les_inflow-base.f90
|      | |-- cs_user_parameters-output.f90
|      | `-- cs_user_postprocess-sfc.c
|      |-- REFERENCE
|      | |-- cs_user_atmospheric_model.f90
|      | |-- cs_user_boundary_conditions.f90
|      | |-- cs_user_coupling.c
|      | |-- cs_user_extra_operations.f90
|      | |-- cs_user_fluid_structure_interaction.f90
|      | |-- cs_user_initialization.f90
|      | |-- cs_user_les_inflow.f90
|      | |-- cs_user_mesh.c
|      | |-- cs_user_modules.f90
|      | |-- cs_user_parameters.f90
|      | |-- cs_user_particle_tracking.f90
|      | |-- cs_user_performance_tuning.c
|      | |-- cs_user_physical_properties.f90
|      | |-- cs_user_postprocess.c
|      | |-- cs_user_postprocess_var.f90
|      | |-- cs_user_radiative_transfer.f90
|      | |-- cs_user_solver.c
|      | |-- cs_user_source_terms.f90
|      | |-- ulevelset.f90
|      | |-- usalcl.f90
|      | |-- usctdz.f90
|      | |-- uselrc.f90
|      | |-- ushist.f90
|      | |-- uskpdc.f90
|      | |-- uslaen.f90
|      | |-- uslag1.f90
|      | |-- uslag2.f90
|      | |-- usporo.f90
|      | |-- uspt1d.f90
|      | |-- usray1.f90
|     | |-- usray2.f90
|     | |-- usthht.f90
|     | |-- ustsma.f90
|     | |-- usvort.f90
|     | `-- usvosy.f90
|     |-- aa_connectivity.f90
|     |-- cs_user_boundary_conditions.f90
|     |-- cs_user_extra_operations.f90
|     |-- cs_user_initialization.f90
|     |-- cs_user_physical_properties.f90
|     |-- iniini.f90
|     `-- ulevelset.f90
|-- subcritical_elevation.jpeg
|-- supercritical-weir
| |-- DATA
| | |-- REFERENCE
| | | |-- cs_user_scripts.py
| | | |-- dp_C3P
| | | |-- dp_C3PSJ
| | | |-- dp_C4P
| | | |-- dp_ELE
| | | |-- dp_FCP
| | | |-- dp_FCP.xml
| | | |-- dp_FCP_new
| | | |-- dp_FUE
| | | |-- dp_FUE_new
| | | |-- dp_transfo
| | | `-- meteo
| | |-- SaturneGUI
| | `-- weir
| |-- RESU
| | |-- 20130520-1456
| | | |-- Tide_level_results.dat
| | | |-- checkpoint
| | | | |-- auxiliary
| | | | `-- main
| | | |-- compile.log
| | | |-- massflow.dat
| | | |-- partition_output
| | | | `-- domain_number_12
| | | |-- performance.log
| | | |-- postprocessing
| | | | |-- RESULTS.case
| | | | |-- results.courantnb.00001 to 01000
| | | | |-- results.density.00001 to 01000
| | | | |-- results.efforts.00001 to 01000
| | | | |-- results.fouriernb.00001 to 01000
| | | | |-- results.geo
| | | | |-- results.lamvisc.00001 to 01000
| | | | |-- results.parallel_domain
| | | | |-- results.pressure.00001 to 01000
| | | | |-- results.scalar1.00001 to 01000
| | | | |-- results.scalar2.00001 to 01000
| | | | |-- results.total_pressure.00001 to 01000
| | | | |-- results.velocity.00001 to 01000
| | | | |-- results.yplus.00001 to 01000
| | | |-- preprocessor.log
| | | |-- setup.log
| | | |-- src_saturne
| | | | |-- aa_connectivity.f90
| | | | |-- cs_user_boundary_conditions.f90
| | | | |-- cs_user_extra_operations.f90
| | | | |-- cs_user_initialization.f90
| | | | |-- cs_user_physical_properties.f90
| | | | |-- iniini.f90
| | | | `-- ulevelset.f90
| | | |-- summary
| | | |-- weir
| | | `-- zerolevel_contour.dat
| | `-- check_mesh
| |     |-- check_mesh.log
| |     |-- performance.log
| |     |-- postprocessing
| |     | |-- BOUNDARY_GROUPS.case
| |     | |-- MESH_GROUPS.case
| |     | |-- QUALITY.case
| |     | |-- boundary_groups.geo
| |     | |-- mesh_groups.geo
| |     | |-- quality.cell_volume
| |     | |-- quality.err_grad_lsq
| |     | |-- quality.err_grad_lsq_ext
| |     | |-- quality.err_grad_lsq_extred
| |     | |-- quality.err_grad_lsq_rc
| |     | |-- quality.err_grad_rc
| |     | |-- quality.face_warp
| |     | |-- quality.face_warp_c_max
| |     | |-- quality.face_warp_v_max
| |     | |-- quality.geo
| |     | |-- quality.grad_lsq
| |     | |-- quality.grad_lsq_ext
| |     | |-- quality.grad_lsq_extred
| |     | |-- quality.grad_lsq_rc
| |     | |-- quality.grad_rc
| |     | |-- quality.non_ortho
| |     | |-- quality.non_ortho_c_max
| |     | |-- quality.non_ortho_v_max
| |     | |-- quality.offset_c_max
| |     | |-- quality.offset_v_max
| |     | |-- quality.weighting_c_max
| |     | `-- quality.weighting_v_max
| |     `-- setup.log
| |-- SCRIPTS
| | `-- runcase
| `-- SRC
|     |-- EXAMPLES
|     | |-- cs_user_boundary_conditions-advanced.f90
|      | |-- cs_user_boundary_conditions-atmospheric.f90
|      | |-- cs_user_boundary_conditions-auto_inlet_profile.f90
|      | |-- cs_user_boundary_conditions-base.f90
|      | |-- cs_user_boundary_conditions-compressible.f90
|      | |-- cs_user_boundary_conditions-cooling_towers.f90
|      | |-- cs_user_boundary_conditions-electric_arcs.f90
|      | |-- cs_user_boundary_conditions-electric_arcs_ieljou_3_or_4.f90
|      | |-- cs_user_boundary_conditions-fuel.f90
|      | |-- cs_user_boundary_conditions-gas_3ptchem.f90
|      | |-- cs_user_boundary_conditions-gas_ebu.f90
|      | |-- cs_user_boundary_conditions-gas_libby_williams.f90
|      | |-- cs_user_boundary_conditions-pulverized_coal.f90
|      | |-- cs_user_boundary_conditions-pulverized_coal_lagrangian.f90
|      | |-- cs_user_extra_operations-energy_balance.f90
|      | |-- cs_user_extra_operations-extract_1d_profile.f90
|      | |-- cs_user_extra_operations-force_temperature.f90
|      | |-- cs_user_extra_operations-global_efforts.f90
|      | |-- cs_user_extra_operations-parallel_operations.f90
|      | |-- cs_user_extra_operations-print_statistical_moment.f90
|      | |-- cs_user_initialization-atmospheric.f90
|      | |-- cs_user_initialization-base.f90
|      | |-- cs_user_initialization-compressible.f90
|      | |-- cs_user_initialization-cooling_towers.f90
|      | |-- cs_user_initialization-electric_arcs.f90
|      | |-- cs_user_initialization-fuel.f90
|      | |-- cs_user_initialization-gas_3ptchem.f90
|      | |-- cs_user_initialization-gas_ebu.f90
|      | |-- cs_user_initialization-gas_libby_williams.f90
|      | |-- cs_user_initialization-pulverized_coal.f90
|      | |-- cs_user_initialization-time_step.f90
|      | |-- cs_user_initialization-unified_combustion_coal.f90
|      | |-- cs_user_les_inflow-base.f90
|      | |-- cs_user_parameters-output.f90
|      | `-- cs_user_postprocess-sfc.c
|      |-- REFERENCE
|      | |-- cs_user_atmospheric_model.f90
|      | |-- cs_user_boundary_conditions.f90
|      | |-- cs_user_coupling.c
|      | |-- cs_user_extra_operations.f90
|      | |-- cs_user_fluid_structure_interaction.f90
|      | |-- cs_user_initialization.f90
|      | |-- cs_user_les_inflow.f90
|      | |-- cs_user_mesh.c
|      | |-- cs_user_modules.f90
|      | |-- cs_user_parameters.f90
|      | |-- cs_user_particle_tracking.f90
|      | |-- cs_user_performance_tuning.c
|      | |-- cs_user_physical_properties.f90
|      | |-- cs_user_postprocess.c
|      | |-- cs_user_postprocess_var.f90
|      | |-- cs_user_radiative_transfer.f90
|      | |-- cs_user_solver.c
|      | |-- cs_user_source_terms.f90
|      | |-- ulevelset.f90
|      | |-- usalcl.f90
|      | |-- usctdz.f90
|      | |-- uselrc.f90
|      | |-- ushist.f90
|      | |-- uskpdc.f90
|      | |-- uslaen.f90
|      | |-- uslag1.f90
|      | |-- uslag2.f90
|      | |-- usporo.f90
|      | |-- uspt1d.f90
|      | |-- usray1.f90
|      | |-- usray2.f90
|      | |-- usthht.f90
|      | |-- ustsma.f90
|      | |-- usvort.f90
|      | `-- usvosy.f90
|      |-- aa_connectivity.f90
|      |-- cs_user_boundary_conditions.f90
|      |-- cs_user_extra_operations.f90
|      |-- cs_user_initialization.f90
|      |-- cs_user_physical_properties.f90
|      |-- iniini.f90
|      `-- ulevelset.f90
|-- transcritical-weir
| |-- DATA
| | |-- REFERENCE
| | | |-- cs_user_scripts.py
| | | |-- dp_C3P
| | | |-- dp_C3PSJ
| | | |-- dp_C4P
| | | |-- dp_ELE
| | | |-- dp_FCP
| | | |-- dp_FCP.xml
| | | |-- dp_FCP_new
| | | |-- dp_FUE
| | | |-- dp_FUE_new
| | | |-- dp_transfo
| | | `-- meteo
| | |-- SaturneGUI
| | `-- weir
| |-- RESU
| | |-- 20130520-1509
| | | |-- Tide_level_results.dat
| | | |-- checkpoint
| | | | |-- auxiliary
| | | | `-- main
| | | |-- compile.log
| | | |-- partition_output
| | | | `-- domain_number_12
| | | |-- performance.log
| | | |-- postprocessing
| | | | |-- RESULTS.case
| | | | |-- results.courantnb.00001 to 00600
| | | | |-- results.density.00001 to 00600
| | | | |-- results.efforts.00001 to 00600
| | | | |-- results.fouriernb.00600 to 00600
| | | | |-- results.geo
| | | | |-- results.lamvisc.00001 to 00600
| | | | |-- results.parallel_domain
| | | | |-- results.pressure.00001 to 00600
| | | | |-- results.scalar1.00001 to 00600
| | | | |-- results.scalar2.00001 to 00600
| | | | |-- results.total_pressure.00001 to 00600
| | | | |-- results.velocity.00001 to 00600
| | | | |-- results.yplus.00001 to 00600
| | | |-- preprocessor.log
| | | |-- setup.log
| | | |-- src_saturne
| | | | |-- aa_connectivity.f90
| | | | |-- cs_user_boundary_conditions.f90
| | | | |-- cs_user_extra_operations.f90
| | | | |-- cs_user_initialization.f90
| | | | |-- cs_user_physical_properties.f90
| | | | |-- iniini.f90
| | | | `-- ulevelset.f90
| | | |-- summary
| | | |-- weir
| | | `-- zerolevel_contour.dat
| | `-- check_mesh
| |     |-- check_mesh.log
| |     |-- performance.log
| |     |-- postprocessing
| |     | |-- BOUNDARY_GROUPS.case
| |     | |-- MESH_GROUPS.case
| |     | |-- QUALITY.case
| |     | |-- boundary_groups.geo
| |     | |-- mesh_groups.geo
| |     | |-- quality.cell_volume
| |     | |-- quality.err_grad_lsq
| |     | |-- quality.err_grad_lsq_ext
| |     | |-- quality.err_grad_lsq_extred
| |     | |-- quality.err_grad_lsq_rc
| |     | |-- quality.err_grad_rc
| |     | |-- quality.face_warp
| |     | |-- quality.face_warp_c_max
| |     | |-- quality.face_warp_v_max
| |     | |-- quality.geo
| |     | |-- quality.grad_lsq
| |     | |-- quality.grad_lsq_ext
| |     | |-- quality.grad_lsq_extred
| |     | |-- quality.grad_lsq_rc
| |     | |-- quality.grad_rc
| |     | |-- quality.non_ortho
| |     | |-- quality.non_ortho_c_max
| |     | |-- quality.non_ortho_v_max
| |     | |-- quality.offset_c_max
| |     | |-- quality.offset_v_max
| |     | |-- quality.weighting_c_max
| |     | `-- quality.weighting_v_max
| |     `-- setup.log
| |-- SCRIPTS
| | `-- runcase
| `-- SRC
|     |-- EXAMPLES
|     | |-- cs_user_boundary_conditions-advanced.f90
|     | |-- cs_user_boundary_conditions-atmospheric.f90
|     | |-- cs_user_boundary_conditions-auto_inlet_profile.f90
|     | |-- cs_user_boundary_conditions-base.f90
|     | |-- cs_user_boundary_conditions-compressible.f90
|     | |-- cs_user_boundary_conditions-cooling_towers.f90
|     | |-- cs_user_boundary_conditions-electric_arcs.f90
|     | |-- cs_user_boundary_conditions-electric_arcs_ieljou_3_or_4.f90
|     | |-- cs_user_boundary_conditions-fuel.f90
|     | |-- cs_user_boundary_conditions-gas_3ptchem.f90
|     | |-- cs_user_boundary_conditions-gas_ebu.f90
|     | |-- cs_user_boundary_conditions-gas_libby_williams.f90
|     | |-- cs_user_boundary_conditions-pulverized_coal.f90
|     | |-- cs_user_boundary_conditions-pulverized_coal_lagrangian.f90
|     | |-- cs_user_extra_operations-energy_balance.f90
|     | |-- cs_user_extra_operations-extract_1d_profile.f90
|     | |-- cs_user_extra_operations-force_temperature.f90
|     | |-- cs_user_extra_operations-global_efforts.f90
|     | |-- cs_user_extra_operations-parallel_operations.f90
|     | |-- cs_user_extra_operations-print_statistical_moment.f90
|     | |-- cs_user_initialization-atmospheric.f90
|     | |-- cs_user_initialization-base.f90
|     | |-- cs_user_initialization-compressible.f90
|     | |-- cs_user_initialization-cooling_towers.f90
|     | |-- cs_user_initialization-electric_arcs.f90
|     | |-- cs_user_initialization-fuel.f90
|     | |-- cs_user_initialization-gas_3ptchem.f90
|     | |-- cs_user_initialization-gas_ebu.f90
|     | |-- cs_user_initialization-gas_libby_williams.f90
|     | |-- cs_user_initialization-pulverized_coal.f90
|     | |-- cs_user_initialization-time_step.f90
|     | |-- cs_user_initialization-unified_combustion_coal.f90
|     | |-- cs_user_les_inflow-base.f90
|     | |-- cs_user_parameters-output.f90
|     | `-- cs_user_postprocess-sfc.c
|     |-- REFERENCE
|     | |-- cs_user_atmospheric_model.f90
|     | |-- cs_user_boundary_conditions.f90
|     | |-- cs_user_coupling.c
|     | |-- cs_user_extra_operations.f90
|     | |-- cs_user_fluid_structure_interaction.f90
|     | |-- cs_user_initialization.f90
|     | |-- cs_user_les_inflow.f90
|     | |-- cs_user_mesh.c
|     | |-- cs_user_modules.f90
|     | |-- cs_user_parameters.f90
|     | |-- cs_user_particle_tracking.f90
|     | |-- cs_user_performance_tuning.c
|     | |-- cs_user_physical_properties.f90
|     | |-- cs_user_postprocess.c
|     | |-- cs_user_postprocess_var.f90
|     | |-- cs_user_radiative_transfer.f90
|     | |-- cs_user_solver.c
|     | |-- cs_user_source_terms.f90
|     | |-- ulevelset.f90
|     | |-- usalcl.f90
|     | |-- usctdz.f90
|     | |-- uselrc.f90
|     | |-- ushist.f90
|     | |-- uskpdc.f90
|     | |-- uslaen.f90
|     | |-- uslag1.f90
|     | |-- uslag2.f90
|     | |-- usporo.f90
|     | |-- uspt1d.f90
|     | |-- usray1.f90
|     | |-- usray2.f90
|     | |-- usthht.f90
|     | |-- ustsma.f90
|     | |-- usvort.f90
|     | `-- usvosy.f90
|     |-- aa_connectivity.f90
|     |-- cs_user_boundary_conditions.f90
|     |-- cs_user_extra_operations.f90
|     |-- cs_user_initialization.f90
|     |-- cs_user_physical_properties.f90
|     |-- iniini.f90
|     `-- ulevelset.f90
`-- tree.txt

81 directories, 70936 files
```

Appendix B

usinv.f90

```
!-------------------------------------------------------------------------------
!                      Code_Saturne version 2.0.0-rc1
!                      --------------------------

!     This file is part of the Code_Saturne Kernel, element of the
!     Code_Saturne CFD tool.

!     Copyright (C) 1998-2008 EDF S.A., France

!     contact: saturne-support@edf.fr

!     The Code_Saturne Kernel is free software; you can redistribute it
!     and/or modify it under the terms of the GNU General Public License
!     as published by the Free Software Foundation; either version 2 of
!     the License, or (at your option) any later version.

!     The Code_Saturne Kernel is distributed in the hope that it will be
!     useful, but WITHOUT ANY WARRANTY; without even the implied warranty
!     of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!     GNU General Public License for more details.

!     You should have received a copy of the GNU General Public License
!     along with the Code_Saturne Kernel; if not, write to the
!     Free Software Foundation, Inc.,
!     51 Franklin St, Fifth Floor,
!     Boston, MA  02110-1301  USA

!-------------------------------------------------------------------------------
module connectivity
  integer(8)          :: nbcell(1000000,6)
  integer(8)          :: nwb_counter
  integer(8)          :: nwb_cnter_Neg
  double precision  :: bc_pos(6)
  double precision  :: scale_length
  double precision, parameter  :: tide_surf = 0.0d0
  double precision  :: tide_surf_outlet
  !******
  !Generic object structure for narrow band spinal node
  !consisting of first and secondary cells
  !******
  type :: simplex
     logical                        :: use_it
     integer                        :: sim_count
     integer                        :: sim_node(3)
     double precision               :: sh_nb(4,3)
     double precision               :: eta
     double precision               :: vol_phi
     type(simplex), pointer         :: next
  end type simplex
  type :: narrow_band_element
     double precision  :: cval
     double precision  :: fval
     double precision  :: ph_k
     double precision  :: ph_star
     double precision  :: xi_h
     integer           :: nwb_index
     integer           :: neg
     integer           :: pos
     integer           :: Sec_pos
     integer           :: Sec_neg
     integer           :: Spos(8)
     integer           :: Ppos(8)
     integer           :: Sneg(8)
     integer           :: Pneg(8)
     logical           :: nb_nochange
     type (simplex), pointer       :: ptr_cutTetra_type1
     type (simplex), pointer       :: ptr_cutTetra_type2
     type (simplex), pointer       :: ptr_cutTetra_type3
  end type narrow_band_element
  !******
  type (narrow_band_element), allocatable :: ptr_NegnwbElm(:)
  type (narrow_band_element), allocatable :: ptr_nwbElm(:)

contains
  subroutine check_zero(sval,text,iel)
   implicit none
   ! check if sval is zero
   double precision :: sval
   integer          :: iel
   character (*) :: text

   if (dabs(sval).lt.tiny(1.0)) then
             !if (irangp.le.0) then
          print*,text,iel
             !endif
 stop
   endif
  end subroutine check_zero
  subroutine error_message(text)
    character (*) :: text
    print*,text
    stop
  end subroutine error_message
end module connectivity



subroutine usiniv &
                                !================

      ( idbia0 , idbra0 ,                                              &
      ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
      nnod   , lndfac , lndfbr , ncelbr ,                            &
      nvar   , nscal  , nphas  ,                                     &
      nideve , nrdeve , nituse , nrtuse ,                            &
      ifacel , ifabor , ifmfbr , ifmcel , iprfml , maxelt , lstelt , &
      ipnfac , nodfac , ipnfbr , nodfbr ,                            &
      idevel , ituser , ia     ,                                     &
      xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
      dt     , rtp    , propce , propfa , propfb , coefa  , coefb  , &
      rdevel , rtuser , ra     )

   use connectivity
   !========================================
   ! Purpose:
   ! -------

   !    User subroutine.

   !    Initialize variables

   ! This subroutine is called at beginning of the computation
   ! (restart or not) before the loop time step

   ! This subroutine enables to initialize or modify (for restart)
   !     unkown variables and time step values

   ! rom and viscl values are equal to ro0 and viscl0 or initialize
   ! by reading the restart file
   ! viscls and cp variables (when there are defined) have no value
   ! excepted if they are read from a restart file

   ! Physical quantities are defined in the following arrays:
   !  propce (physical quantities defined at cell center),
   !  propfa (physical quantities defined at interior face center),
   !  propfa (physical quantities defined at border face center).
   !
   ! Examples:
   !  propce(iel, ipproc(irom  (iphas))) means rom  (iel, iphas)
   !  propce(iel, ipproc(iviscl(iphas))) means viscl(iel, iphas)
   !  propce(iel, ipproc(icp   (iphas))) means cp   (iel, iphas)
   !  propce(iel, ipproc(ivisls(iscal))) means visls(iel, iscal)
   !  propfa(ifac, ipprof(ifluma(ivar))) means flumas(ifac, ivar)
   !  propfb(ifac, ipprob(irom (iphas))) means romb  (ifac, iphas)
   !  propfb(ifac, ipprob(ifluma(ivar))) means flumab(ifac, ivar)

   ! Modification of the behaviour law of physical quantities (rom, viscl,
   ! viscls, cp) is not done here. It is the purpose of the user subroutine
   ! usphyv

   ! Cells identification
   ! ===================

   ! Cells may be identified using the 'getcel' subroutine.
   ! The syntax of this subroutine is described in the 'usclim' subroutine,
   ! but a more thorough description can be found in the user guide.


   !-------------------------------------------------------------------------------
   ! Arguments
   !__________________.____._____.________________________________________________.
   ! name             !type!mode ! role                                           !
   !__________________!____!_____!________________________________________________!
   ! idbia0           ! i  ! <-- ! number of first free position in ia            !
   ! idbra0           ! i  ! <-- ! number of first free position in ra            !
   ! ndim             ! i  ! <-- ! spatial dimension                              !
   ! ncelet           ! i  ! <-- ! number of extended (real + ghost) cells        !
   ! ncel             ! i  ! <-- ! number of cells                                !
   ! nfac             ! i  ! <-- ! number of interior faces                       !
   ! nfabor           ! i  ! <-- ! number of boundary faces                       !
   ! nfml             ! i  ! <-- ! number of families (group classes)             !
   ! nprfml           ! i  ! <-- ! number of properties per family (group class)  !
   ! nnod             ! i  ! <-- ! number of vertices                             !
   ! lndfac           ! i  ! <-- ! size of nodfac indexed array                   !
   ! lndfbr           ! i  ! <-- ! size of nodfbr indexed array                   !
   ! ncelbr           ! i  ! <-- ! number of cells with faces on boundary         !
   ! nvar             ! i  ! <-- ! total number of variables                      !
   ! nscal            ! i  ! <-- ! total number of scalars                        !
   ! nphas            ! i  ! <-- ! number of phases                               !
   ! nideve, nrdeve   ! i  ! <-- ! sizes of idevel and rdevel arrays              !
   ! nituse, nrtuse   ! i  ! <-- ! sizes of ituser and rtuser arrays              !
   ! ifacel(2, nfac)  ! ia ! <-- ! interior faces -> cells connectivity           !
   ! ifabor(nfabor)   ! ia ! <-- ! boundary faces -> cells connectivity           !
   ! ifmfbr(nfabor)   ! ia ! <-- ! boundary face family numbers                   !
   ! ifmcel(ncelet)   ! ia ! <-- ! cell family numbers                            !
   ! iprfml           ! ia ! <-- ! property numbers per family                    !
   !  (nfml, nprfml)  !    !     !                                                !
   ! maxelt           ! i  ! <-- ! max number of cells and faces (int/boundary)   !
   ! lstelt(maxelt)   ! ia ! --- ! work array                                     !
   ! ipnfac(nfac+1)   ! ia ! <-- ! interior faces -> vertices index (optional)    !
   ! nodfac(lndfac)   ! ia ! <-- ! interior faces -> vertices list (optional)     !
   ! ipnfbr(nfabor+1) ! ia ! <-- ! boundary faces -> vertices index (optional)    !
   ! nodfbr(lndfbr)   ! ia ! <-- ! boundary faces -> vertices list (optional)     !
   ! idevel(nideve)   ! ia ! <-> ! integer work array for temporary development   !
   ! ituser(nituse)   ! ia ! <-> ! user-reserved integer work array             !
   ! ia(*)            ! ia ! --- ! main integer work array                      !
   ! xyzcen           ! ra ! <-- ! cell centers                                 !
   !  (ndim, ncelet)  !    !     !                                              !
   ! surfac           ! ra ! <-- ! interior faces surface vectors               !
   !  (ndim, nfac)    !    !     !                                              !
   ! surfbo           ! ra ! <-- ! boundary faces surface vectors               !
   !  (ndim, nfabor)  !    !     !                                              !
   ! cdgfac           ! ra ! <-- ! interior faces centers of gravity            !
   !  (ndim, nfac)    !    !     !                                              !
   ! cdgfbo           ! ra ! <-- ! boundary faces centers of gravity            !
   !  (ndim, nfabor)  !    !     !                                              !
   ! xyznod           ! ra ! <-- ! vertex coordinates (optional)                !
   !  (ndim, nnod)    !    !     !                                              !
   ! volume(ncelet)   ! ra ! <-- ! cell volumes                                 !
   ! dt(ncelet)       ! ra ! <-- ! time step (per cell)                         !
   ! rtp(ncelet, *)   ! ra ! <-- ! computed variables at cell centers at current  !
   !                  !    !     ! time steps                                   !
   ! propce(ncelet, *)! ra ! <-- ! physical properties at cell centers          !
   ! propfa(nfac, *)  ! ra ! <-- ! physical properties at interior face centers   !
   ! propfb(nfabor, *)! ra ! <-- ! physical properties at boundary face centers   !
   ! coefa, coefb     ! ra ! <-- ! boundary conditions                          !
   !  (nfabor, *)     !    !     !                                              !
   ! rdevel(nrdeve)   ! ra ! <-> ! real work array for temporary development    !
   ! rtuser(nrtuse)   ! ra ! <-> ! user-reserved real work array                !
   ! ra(*)            ! ra ! --- ! main real work array                         !
   !__________________!____!_____!______________________________________________!

   !     Type: i (integer), r (real), s (string), a (array), l (logical),
   !           and composite types (ex: ra real array)
   !     mode: <-- input, --> output, <-> modifies data, --- work array
   !=======================================

   implicit none

   !=======================================
   ! Common blocks
   !=======================================

   include "paramx.h"
   include "pointe.h"
   include "numvar.h"
   include "optcal.h"
   include "cstphy.h"
   include "entsor.h"
   include "parall.h"
   include "period.h"

   !=======================================

   ! Arguments

   integer          ilelt, nlelt
   integer          idbia0 , idbra0
   integer          ndim   , ncelet , ncel   , nfac   , nfabor
   integer          nfml   , nprfml
   integer          nnod   , lndfac , lndfbr , ncelbr
   integer          nvar   , nscal  , nphas
   integer          nideve , nrdeve , nituse , nrtuse

   integer          ifacel(2,nfac) , ifabor(nfabor)
   integer          ifmfbr(nfabor) , ifmcel(ncelet)
   integer          iprfml(nfml,nprfml), maxelt, lstelt(maxelt)
   integer          ipnfac(nfac+1), nodfac(lndfac)
   integer          ipnfbr(nfabor+1), nodfbr(lndfbr)
   integer          idevel(nideve), ituser(nituse), ia(*)

   double precision xyzcen(ndim,ncelet)
   double precision surfac(ndim,nfac), surfbo(ndim,nfabor)
   double precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)
   double precision xyznod(ndim,nnod), volume(ncelet)
   double precision dt(ncelet), rtp(ncelet,*), propce(ncelet,*)
   double precision propfa(nfac,*), propfb(nfabor,*)
   double precision coefa(nfabor,*), coefb(nfabor,*)
   double precision rdevel(nrdeve), rtuser(nrtuse), ra(*)

   ! Local variables


   logical :: switch1,switch2
   integer          idebia, idebra
   integer          iel, iutile, iel1,iel2, i, j, ival,ival2,impout(6),ii
   integer          ifac
   integer(8)       con(ncel,6), ihuge
   double precision a(3), b(3), c(3)

   !####local variables for Level Set modelling
   double precision  phi_nbck,phi_nbck2, phi_cenck, max_val,min_val,xpos1,xpos2,sec(3)
   integer(8)        icen
   integer           ifac2, n_of_f, jmax,jmin,isnbb,Number_Of_Faces,Number_Of_Faces2
   logical           ifind, iswitch1,iswitch2
   integer           nd_ix, nd_i, nd_k, nx, ni, nk

   !================[Redistancing code variables]=================

   integer          ival_nb,ival_nbNeg, ielt, nlelt2, k, xi,icount, N_i,MAXIT
   double precision sign_val, dist_ptR(3),dif_pr(3),dif_xr(3),t_val,dif(3),tdiv,k_hat(3),scale_diff,scal
   double precision v_hat(3), n_div, n_hat(3), dist1, dist2, dist3, dI, dImin, x(3), po(3), r(3)
   double precision S_h,s_val, Sk, phi,phi_starr,x_n(3),po_n(3), delta_k, eta_k,eta_sum,xi_sum,fMin,t_sm
   double precision fl,fh,f,c_val,c_l,c_h,c_1,c_2,dc,swap,del,c_acc, s_k,C_const,ix_val
   double precision dist_prR(3),dval,phi3,rdist
   double precision del_xdist
   double precision value, psi_v
   double precision, dimension(9,3) :: r_stencil
   integer          inod(4), ir,irV(4),ival_array(3)
   logical          Lr(4),error_iteration,ipos_checkdo
```

```
integer          I_it,ilim,icen2,icen3,n_faces

double precision  phi_val(9),rcen(3),Grad_phi(3),phi_max,phi_min,psi_value,del_A
integer           i_vertex
!================[Free surface modelling code variables]=================

integer          ionbb, i_group,iorder,n_group, n_stencil,id, itype, n_col,nbox,ntri
integer          ieln, ErrorFlag
integer          ichange,icheck,iprim_neg,isec_pos,isec_neg,iprim_pos
integer          ineg,inod2,iph_XI,inext2,ipos,prim_pos,ir0,ir1,ir2,isim_count,isimplxMax
integer          itype_max,isimpx,isampleNeg,Nneg,Npos,Nmax, iside
double precision phi_1, small_diff,t_s2_left(3),t_s2_right(3),p_j(3),theta_a,theta_b,theta_c
double precision n_hat1(3),n_hat2(3),n_hat3(3), a_right, a_left,t_o2_left(3), a_tswf_f2, theta_f2
double precision theta_s2, theta_o2,theta_1,theta_2,theta_3, a_tswf_s2, &
                           & a_tswf_o2,lc_tswf_o2,lc_tswf_s2
double precision tswf_s2_1(3),tswf_s2_2(3),tswf_o2_1(3),tswf_o2_2(3),t_o2_right(3)
double precision tpj0(3),tpj1(3),tpj2(3),lb_value,lc_value, parallel_chk,tdiv2,tdiv1,tdiv0,lc_tswf_f2
double precision t_21(3),t_23(3),t_32(3),tswf_f2_1(3),tswf_f2_2(3),t_f2_right(3),t_f2_left(3), xval
double precision diff1(3), diff2(3), diff3(3),px0(3), propU, propV, propW, Uvel(3), propP
double precision prop_u(3),prop_v(3),prop_w(3),prop_val(3),xyz(3),vol1,vol2,kv(3),s1(3),s2(3)
double precision rlook1(3),rlook2(3), xvalue, yvalue, density,Area_k,deltaV_k,f2,v(3),sum_scale
double precision, parameter :: pi = 3.141592D0
real                             :: chk_x(100)
real, dimension(:,:), allocatable :: data_array
real, dimension(:,:), allocatable :: chk_pts
real, allocatable                :: px(:)
real, dimension(:,:), allocatable :: a_matrx
logical          iseen
double precision, parameter         :: U_init = 0.0    ! m/s
double precision  h_s, xrtp,xrtp2,tide_start
double precision, parameter         :: small_value = 1.0d-6
double precision, parameter         ::  wave_amplitude = 0.005
double precision, parameter         :: k_wave  = 10.0 * pi
double precision, parameter         ::  water_density = 10.0
type(simplex), pointer              :: current,previous
!=======================================
! 1.  Initialization of local variables
!=======================================
idebia = idbia0
idebra = idbra0
c_acc  = 1.0d-6
fMin  = 1.0d-6
ihuge  = 2.0e10
ir     = -ihuge
jmax   = -ihuge
icount = 0
MAXIT  = 20
do i = 1, ncel
    do j = 1, 6
        con(i,j) = -ihuge
        nbcell(i,j) = -ihuge
    enddo
enddo
```

```
if (isuite.eq.0) then

    !*****************Find the scale_length to avoid inacuracies with the method
    scale_length = 1.0
    print*,'scale_length = ',scale_length
    print*,'tide_surf = ',tide_surf
    !=======================================
    ! (A) Unknown variables initialization:
    !      ONLY done if there is no restart computation
    !=======================================

    ! --- Example:  isca(1) is the variable number in RTP related to the first
    !               user-defined scalar variable
    !               rtp(iel,isca(1)) is the value of this variable in cell number
    !               iel.


    !=======================================
    ! (B) Building connectivity between particular indexed cell in the flow field and
    !     its local neighbour cells. For a hexahedral cell this involves 6 neighbours
    !=======================================
    do ifac = 1, nfac
        iel1 = ifacel(1, ifac)
        iel2 = ifacel(2,ifac)
        switch1 = .true.
        switch2 = .true.
        Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
        do i = 1, Number_Of_Faces
            if ((switch1).and.(con(iel1,i).eq.(-ihuge) )) then
                con(iel1,i) = ifac
                switch1 = .false.
            endif
            if ((switch2).and.(con(iel2,i).eq.(-ihuge) )) then
                con(iel2,i) = ifac
                switch2 = .false.
            endif
        enddo
    enddo

    !====================
    !(C) Create 'nbcell(iel,Number_Of_Faces)' which provides connectivity between a cell centre of ce
    !and 1 to 'Number_Of_Faces' neighbour cells over 1 to ncel mesh cells in the flow field for any g
    !polyhedral mesh (note in this case this is hexahedral)
    !====================&
    do iel = 1, ncel
        do ifac = 1,6
            if (con(iel,ifac).gt.(-ihuge)) then
                iel2 = ifacel(2,con(iel,ifac))
                iel1 = ifacel(1,con(iel,ifac))
                if (iel1.ne.iel) then
                    nbcell(iel,ifac) = iel1
                endif
                if (iel2.ne.iel) then
                    nbcell(iel,ifac) = iel2
                endif
            endif
        enddo
    enddo
    !=======================================
    ! (D) Initial set up of flow field values for velocity and scalars
    !=======================================

    do iel = 1, ncel
        tide_start = wave_amplitude * dsin(-k_wave * xyzcen(1,iel) ) + tide_surf
        if (xyzcen(2,iel).ge.( tide_start)) then
            rtp(iel,isca(1)) = 120
        else
            rtp(iel,isca(1)) = -120
        endif
        rtp(iel,iu(1)) = 0.0d0
        rtp(iel,iv(1)) = 0.0d0
        rtp(iel,iw(1)) = 0.0d0
        rtp(iel,isca(2)) = scale_length * (xyzcen(2,iel)) - scale_length * (tide_start)
    enddo

    !=======================================
    ! (E) Initial set up of Level set first and second neighbour cells surrounding isocontour S_h
    !    --------------------------------------------------
    !=======================================
    !===========Setup the first neighbour cells surrounding S_h
    !-------------positive side sweep--------------
    allocate(ptr_nwbElm(50000))
    nwb_counter = 0
    do iel = 1,ncel
        phi_cenck       = rtp(iel,isca(2))                                 ! positive kth node
        Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
        ! ++++++++looking for field region where iso-surface S_h exists
        iprim_neg = 0
        iprim_pos = 0
        do ifac = 1,Number_of_Faces
            phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
            if ((phi_nbck.lt.0.0d0).and.(phi_cenck.gt.0.0d0)) then
                !-------check negative side---------
                if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                    iprim_neg = iprim_neg + 1
                endif
            endif
        enddo
        if (iprim_neg > 0) then
            do ifac2 = 1, Number_of_Faces
                phi_nbck2  = rtp(nbcell(iel,ifac2),isca(2))
                if ( phi_nbck2.gt.0.0d0) then
                    if (secondary_cell_select(nbcell(iel,ifac2)) <= 0) then
                        iprim_pos = iprim_pos + 1
                    endif
                endif
            enddo
        endif
        !-----------------------------------------------------------------------------------------------
        !     Create a new set of temporary simplex tetrahedra surrounding a piece of the isosurface
        !-----------------------------------------------------------------------------------------------
        if ((iprim_neg.gt.0).and.(iprim_neg.lt.4).and.(phi_cenck.gt.0)) then
            nwb_counter   = nwb_counter + 1
            if (nwb_counter.gt.ihuge) then
                deallocate(ptr_nwbElm)
                stop 0
            endif
            ptr_nwbElm(nwb_counter)%nwb_index = iel
            !**************************************************
            iprim_neg = 0
            iprim_pos = 0
            isec_pos  = 0
            Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
            do ifac = 1, Number_of_Faces
                phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
                if ((phi_nbck.lt.0.0d0).and.(phi_cenck.gt.0.0d0)) then
                    rtp(iel,isca(1)) = 40.0                                    ! prim scalar1 +ve
                    if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                        iprim_neg = iprim_neg + 1
                        ptr_nwbElm(nwb_counter)%Pneg(iprim_neg) = nbcell(iel,ifac)   ! negative prima
                    endif
                else if ( (phi_nbck.gt.0.0d0).and.(phi_cenck.gt.0.0d0) ) then
                    if (secondary_cell_select(nbcell(iel,ifac)) == 1) then
                        isec_pos = isec_pos + 1
                        ptr_nwbElm(nwb_counter)%Spos(isec_pos) = nbcell(iel,ifac)   ! positive second
                        rtp(nbcell(iel,ifac),isca(1)) = 80.0             ! sec scalar1 +ve
                    else
                        iprim_pos = iprim_pos + 1
                        ptr_nwbElm(nwb_counter)%Ppos(iprim_pos) = nbcell(iel,ifac)   ! positive prima
                        rtp(nbcell(iel,ifac),isca(1)) = 40.0        !  prim scalar1 +ve
                    endif
                endif
            enddo
            !***********************NOTE NO NEGATIVE SECONDARY*********************
            ptr_nwbElm(nwb_counter)%neg      =  iprim_neg  ! total number of negative primary nodes
            ptr_nwbElm(nwb_counter)%pos      =  iprim_pos  ! total number of positive primary nodes
            ptr_nwbElm(nwb_counter)%Sec_pos   =  isec_pos   ! total number of positive secondary node
        endif
        !-----------------------------------------------------------------------------------------------

    enddo

    !-------------negative side sweep--------------
    allocate(ptr_NegnwbElm(50000))
    nwb_cnter_Neg = 0
    do iel = 1,ncel
        phi_cenck      = rtp(iel,isca(2))                                 ! negative kth node
        Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
        ! ++++++++looking for field region where iso-surface S_h exists
```

```
iprim_neg = 0
iprim_pos = 0
do ifac = 1,Number_of_Faces
    phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
    if ((phi_nbck.gt.0.0d0).and.(phi_cenck.lt.0.0d0)) then         !!!NOTE NOW phi_cenck negative
        !-------check positive side---------
        if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
            iprim_pos = iprim_pos + 1
        endif
    endif
enddo
if (iprim_pos > 0) then
    do ifac2 = 1, Number_of_Faces
        phi_nbck2  = rtp(nbcell(iel,ifac2),isca(2))
        if ( phi_nbck2.lt.0.0d0) then
            if (secondary_cell_select(nbcell(iel,ifac2)) <= 0) then
                iprim_neg = iprim_neg + 1
            endif
        endif
    enddo
endif
!------------------------------------------------------------------------------------------
!     Create a new set of temporary simplex tetrahedra surrounding a piece of the isosurface
!------------------------------------------------------------------------------------------
if ((iprim_pos.gt.0).and.(iprim_pos.lt.4).and.(phi_cenck.lt.0.0d0)) then
    nwb_cnter_Neg   = nwb_cnter_Neg + 1
    if (nwb_cnter_Neg.gt.ihuge) then
        deallocate(ptr_NegnwbElm)
        stop 0
    endif
    ptr_NegnwbElm(nwb_cnter_Neg)%nwb_index = iel
    !***************************************************
    iprim_neg = 0
    iprim_pos = 0
    isec_neg  = 0
    Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
    do ifac = 1, Number_of_Faces
        phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
        if ((phi_nbck.gt.0.0d0).and.(phi_cenck.lt.0.0d0)) then          !!!NOTE NOW phi_cenck negati
            rtp(iel,isca(1)) = -40.0                                      ! prim scalar1 -ve
            if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                iprim_pos = iprim_pos + 1
                ptr_NegnwbElm(nwb_cnter_Neg)%Ppos(iprim_pos) = nbcell(iel,ifac)   ! positive primary
            endif
        else if ( (phi_nbck.lt.0.0d0).and.(phi_cenck.lt.0.0d0) ) then
            if (secondary_cell_select(nbcell(iel,ifac)) == 1) then
                isec_neg = isec_neg + 1
                ptr_NegnwbElm(nwb_cnter_Neg)%Sneg(isec_neg) = nbcell(iel,ifac)   ! negative secondary
                rtp(nbcell(iel,ifac),isca(1)) = -80.0             ! sec scalar1 -ve
            else
                iprim_neg = iprim_neg + 1
                ptr_NegnwbElm(nwb_cnter_Neg)%Pneg(iprim_neg) = nbcell(iel,ifac)   ! negative primary
                rtp(nbcell(iel,ifac),isca(1)) = -40.0        !  prim scalar1 -ve
                endif
            endif
        enddo
        !******************************** NOTE NO POSITIVE SECONDARY**********************
        ptr_NegnwbElm(nwb_cnter_Neg)%neg       =  iprim_neg  ! total number of negative primary n
        ptr_NegnwbElm(nwb_cnter_Neg)%pos       =  iprim_pos  ! total number of positive primary n
        ptr_NegnwbElm(nwb_cnter_Neg)%Sec_neg   =  isec_neg   ! total number of negative secondary
    endif
    !--------------------------------------------------------------------------------------------

enddo !end of iel loop
!-----------end of negative side sweep
!======================================
!     Initial redistancing (START)
!======================================


!*****************************************************************
!           Step 1:: re-Compute the exact distance to S_h
!*****************************************************************
!-------------------------------------------------
!---------positive side of free surface-----------
!-------------------------------------------------
iside = 0
!    iseen = .true.
do iel = 1, nwb_counter
    ival_nb  = ptr_nwbElm(iel)%nwb_index
    Nneg = ptr_nwbElm(iel)%neg
    Npos = ptr_nwbElm(iel)%pos
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    call create_simplices( iside, ptr_nwbElm(iel), Nneg, Npos,itype_max)
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
            Nmax = 3
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
            Nmax = 4
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
            Nmax = 3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        do isimpx = 1, isimplxMax

            do inod2 = 1, Nmax
                if (itype == 1) then
                    if ( inod2 == 1) then
                        ir   =  ival_nb               ! Kth node
                    else
                        ir  =  current%sim_node(inod2)
                    endif
                    ieln =  current%sim_node(1)
                else if (itype == 2) then
                    select case (inod2)
                    case (1,2)
                        ieln  =  current%sim_node(inod2)
                        ir   =  ival_nb
                    case (3,4)
                        ival = inod2 - 2
                        ieln  =  current%sim_node(ival)
                        ir   =  current%sim_node(3)
                    case default
                        call error_message('Error2 in step1 with isimplxMax in USINV')
                    end select
                else if (itype == 3) then
                    ieln  =  current%sim_node(inod2)
                    ir    =  ival_nb
                else
                    call error_message('Error3 in step1 with isimplxMax in USINV')
                endif
                s_val   = rtp(ir,isca(2)) -  rtp(ieln,isca(2))
                !******** check if s_val is zero******
                call check_zero(s_val,'Error4 in USINV with Sk reconstruction at iel = ',iel)
                S_h  = - rtp(ieln,isca(2))/s_val
                if (S_h > 0.98) then
                    current%use_it = .false.                                       !new
                endif
                a(1) = scale_length * (xyzcen(1,ir) )
                a(2) = scale_length * (xyzcen(2,ir) )
                a(3) = scale_length * (xyzcen(3,ir) )
                b(1) = scale_length * (xyzcen(1,ieln) )
                b(2) = scale_length * (xyzcen(2,ieln) )
                b(3) = scale_length * (xyzcen(3,ieln) )
                call position_vec(a, b, S_h, c)
                current%sh_nb(inod2,1) = c(1)
                current%sh_nb(inod2,2) = c(2)
                current%sh_nb(inod2,3) = c(3)
            enddo ! inod2 loop
            current => current%next
        enddo ! loop isimpx
    enddo ! loop itype
enddo ! iel loop


!^^^^^^^^^^^check for zero volume simplices == (start) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
iside = 0
do iel = 1, nwb_counter
    ival_nb    =   ptr_nwbElm(iel)%nwb_index
    phi        =   1.0
    Npos = ptr_nwbElm(iel)%pos
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        ! &&& scan for isimplxMax per simplex cut type considered &&&
        do isimpx = 1, isimplxMax
            if (current%use_it) then                               !new
                call Vol_k(.true. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                if (vol2 < tiny(1.0)) then
                    !print*,iel,'positive vol_k is = ',vol2,isimplxMax
                    current%use_it = .false.
                endif
            endif
            current => current%next
        enddo ! isimpx loop
    enddo    ! itype loop
enddo  ! iel loop

!^^^^^^^^^^^check for zero volume simplices == (end) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


!==================Compute dI such that dI = min x belongs S_k|XI-x|===(start)
do iel = 1, nwb_counter
    ival_nb   = ptr_nwbElm(iel)%nwb_index
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    dImin = huge(1.0)     !***set d(Xn) = +infinity for n = 1,2,...Nnod_P
    do itype = 1, itype_max
```

```
select case (itype)
case (1)
    isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
    current => ptr_nwbElm(iel)%ptr_cutTetra_type1
case (2)
    isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
    current => ptr_nwbElm(iel)%ptr_cutTetra_type2
case (3)
    isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
    current => ptr_nwbElm(iel)%ptr_cutTetra_type3
case default
    call error_message('Error1 in step1 with isimplxMax in USINV')
end select
! && scan for isimplxMax simplices per simplex cut type considered &&
do isimpx = 1, isimplxMax
    ir0   =  1        ! Kth node
    ir1   =  2
    ir2   =  3
    a(1) = scale_length * (xyzcen(1,ival_nb) )        ! k node
    a(2) = scale_length * (xyzcen(2,ival_nb) )
    a(3) = scale_length * (xyzcen(3,ival_nb) )
    kv(1) = current%sh_nb(ir0,1)      ! edge of free surface near kth node
    kv(2) = current%sh_nb(ir0,2)
    kv(3) = current%sh_nb(ir0,3)
    dif(1) = kv(1) - a(1)
    dif(2) = kv(2) - a(2)
    dif(3) = kv(3) - a(3)
    s1(1) = current%sh_nb(ir1,1) - kv(1)
    s1(2) = current%sh_nb(ir1,2) - kv(2)
    s1(3) = current%sh_nb(ir1,3) - kv(3)
    s2(1) = current%sh_nb(ir2,1) - kv(1)
    s2(2) = current%sh_nb(ir2,2) - kv(2)
    s2(3) = current%sh_nb(ir2,3) - kv(3)
    call cross_product(s1,s2,v)
    tdiv     = dabs(dot_product(v,v))
    !if (current%use_it) then                          !new
    if (tdiv < small_value ) then
        !dist1 = rtp(ival_nb,isca(2))
        !if (dabs(dImin) > dabs(dist1)) then
        !  dImin = dabs(dist1)
        !endif
        current => current%next
        cycle
    endif
    !endif                                             !new
    tdiv1 =  dsqrt(tdiv)
    v_hat(1) = v(1)/tdiv1
    v_hat(2) = v(2)/tdiv1
    v_hat(3) = v(3)/tdiv1
    dist1 = dabs(dot_product(dif, v_hat))
    if ((dabs(dImin) > dabs(dist1))) then
        dImin = dabs(dist1)
    endif
```

```
            current => current%next
        enddo ! loop isimplx
    enddo ! loop itype
    ptr_nwbElm(iel)%ph_star = dabs(dImin)  !^^^^^^set phi*(Xn) = d(Xn) for n = 1,2,...Nnod_P ^^^^
enddo ! iel loop


!-------------------------------------------------
!---------negative side of free surface-----------
!-------------------------------------------------
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg  = ptr_NegnwbElm(iel)%nwb_index
    Nneg = ptr_NegnwbElm(iel)%neg
    Npos = ptr_NegnwbElm(iel)%pos
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    call create_simplices( iside, ptr_NegnwbElm(iel), Npos, Nneg,itype_max)
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
            Nmax = 3
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
            Nmax = 4
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
            Nmax = 3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select

        do isimpx = 1, isimplxMax

            do inod2 = 1, Nmax
                if (itype == 1) then
                    if ( inod2 == 1) then
                        ir   =  ival_nbNeg                ! Kth node
                    else
                        ir   =  current%sim_node(inod2)
                    endif
                    ieln =  current%sim_node(1)
                else if (itype == 2) then
                    select case (inod2)
                    case (1,2)
                        ieln  =  current%sim_node(inod2)
                        ir   =  ival_nbNeg
                    case (3,4)
                        ival = inod2 - 2
                        ieln  =  current%sim_node(ival)
                        ir   =  current%sim_node(3)
                    case default
                        call error_message('Error2 in step1 with isimplxMax in USINV')
                    end select
                else if (itype == 3) then
                    ieln  =  current%sim_node(inod2)
                    ir    =  ival_nbNeg
                else
                    call error_message('Error3 in step1 with isimplxMax in USINV')
                endif
                s_val   = rtp(ir,isca(2)) -  rtp(ieln,isca(2))
                !******** check if s_val is zero******
                call check_zero(s_val,'Error4 in USINV with Sk reconstruction at iel = ',iel)
                S_h  = - rtp(ieln,isca(2))/s_val
                if (S_h > 0.98) current%use_it = .false.                                 !new
                a(1) = scale_length * (xyzcen(1,ir) )
                a(2) = scale_length * (xyzcen(2,ir) )
                a(3) = scale_length * (xyzcen(3,ir) )
                b(1) = scale_length * (xyzcen(1,ieln) )
                b(2) = scale_length * (xyzcen(2,ieln) )
                b(3) = scale_length * (xyzcen(3,ieln) )
                call position_vec(a, b, S_h, c)
                current%sh_nb(inod2,1) = c(1)
                current%sh_nb(inod2,2) = c(2)
                current%sh_nb(inod2,3) = c(3)
            enddo ! inod2 loop
            current => current%next
        enddo ! loop isimpx
    enddo ! loop itype
enddo ! iel loop

!^^^^^^^^^^^check for zero volume simplices == (start) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    phi           =   1.0
    Npos = ptr_NegnwbElm(iel)%pos
    Nneg = ptr_NegnwbElm(iel)%neg
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
```

```
                isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select
            ! &&& scan for isimplxMax simplices per simplex cut type considered &&&
            do isimpx = 1, isimplxMax
                if (current%use_it) then                               !new
                    call Vol_k(.true. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                    if (vol2 < tiny(1.0)) then
                        !print*,iel,'Negative vol_k is = ',vol2,isimplxMax
                        current%use_it = .false.
                    endif
                endif
                current => current%next
            enddo ! isimpx loop
        enddo    ! itype loop
    enddo  ! iel loop
    !^^^^^^^^^^^check for zero volume simplices == (end)

    !===================Compute dI such that dI = min x belongs S_k|XI-x|===(start)
    do iel = 1, nwb_cnter_Neg
        ival_nbNeg   = ptr_NegnwbElm(iel)%nwb_index
        Npos = ptr_NegnwbElm(iel)%pos
        if (Npos > 3) then
            itype_max = 3
        else
            itype_max = Npos
        endif
        dImin = huge(1.0)     !***set d(Xn) = +infinity for n = 1,2,...Nnod_P
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
            case (2)
                isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select
            ! &&& scan for isimplxMax per simplex cut type considered &&&
            do isimpx = 1, isimplxMax
                ir0   =  1        ! Kth node
                ir1   =  2
                ir2   =  3
                a(1) = scale_length * (xyzcen(1,ival_nbNeg) )        ! k node
                a(2) = scale_length * (xyzcen(2,ival_nbNeg) )
                a(3) = scale_length * (xyzcen(3,ival_nbNeg) )
                kv(1) = current%sh_nb(ir0,1)      ! edge of free surface near kth node
                kv(2) = current%sh_nb(ir0,2)
                kv(3) = current%sh_nb(ir0,3)
                dif(1) = kv(1) - a(1)
                dif(2) = kv(2) - a(2)
                dif(3) = kv(3) - a(3)
                s1(1) = current%sh_nb(ir1,1) - kv(1)
                s1(2) = current%sh_nb(ir1,2) - kv(2)
                s1(3) = current%sh_nb(ir1,3) - kv(3)
                s2(1) = current%sh_nb(ir2,1) - kv(1)
                s2(2) = current%sh_nb(ir2,2) - kv(2)
                s2(3) = current%sh_nb(ir2,3) - kv(3)
                call cross_product(s1,s2,v)
                tdiv     = dabs(dot_product(v,v))
                tdiv1 =  dsqrt(tdiv)
                !if (current%use_it) then
                if (tdiv < small_value ) then
                    !dist1 = - rtp(ival_nbNeg,isca(2))
                    !if ( dabs(dImin) > dabs(dist1)) then
                    !   dImin = dabs(dist1)
                    !endif
                    current => current%next
                    cycle
                endif
                !endif
                v_hat(1) = v(1)/tdiv1
                v_hat(2) = v(2)/tdiv1
                v_hat(3) = v(3)/tdiv1
                dist1 = dabs(dot_product(dif, v_hat))
                if ((dabs(dImin) > dabs(dist1))) then
                    dImin = dabs(dist1)
                endif
                current => current%next
            enddo ! loop isimplx
        enddo ! loop itype
        ptr_NegnwbElm(iel)%ph_star = - dImin  !^^^^^^set phi*(Xn) = d(Xn) for n = 1,2,...Nnod_P ^^^^^^
    enddo ! iel loop



    !===================Compute dI such that dI = min x belongs S_k|XI-x|====(end)
    !*****************************************************************
    !           Step 2:: Find eta_h, a piecewise constant function
    !*****************************************************************
    !===================Find eta_h, a piecewise constant function (simplex wise mass correction) == (start)
    !---------positive side of free surface-----------
    iside = 0
    do iel = 1, nwb_counter
        ival_nb    =   ptr_nwbElm(iel)%nwb_index
        phi        =   rtp(ival_nb,isca(2))
        phi_starr  =   ptr_nwbElm(iel)%ph_star
```

```
        !print*,iel,phi,phi_starr
        eta_k      =   phi - phi_starr
        Npos = ptr_nwbElm(iel)%pos
        Nneg = ptr_nwbElm(iel)%neg
        if (Nneg > 3) then
            itype_max = 3
        else
            itype_max = Nneg
        endif
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type1
            case (2)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select
            ! &&& scan for isimplxMax simplices per simplex cut type considered &&&
            do isimpx = 1, isimplxMax
                call Vol_k(.false. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                current%vol_phi = vol2
                call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,eta_k,Area_K,vol1)
                deltaV_k = vol2 - vol1
                icount   = 0
                do
                    if ((dabs(deltaV_k) < 1.0d-8).or.(icount > 20)) exit
                    icount = icount + 1
                    if ( Area_K < tiny(1.0) ) then
                        exit
                    endif
                    eta_k =  -3.0 * deltaV_k/Area_K     !Changed
                    call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,eta_k,Area_K,vol1)
                    deltaV_k = vol2 - vol1
                enddo
                if (current%use_it) then                              !new
                    current%eta = eta_k
                else
                    current%eta = 0.0
                endif                                                 !new
                current => current%next
            enddo ! isimpx loop
        enddo    ! itype loop
    enddo  ! iel loop


    !---------negative side of free surface-----------
    iside = 1
```

```
do iel = 1, nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    phi        =   rtp(ival_nbNeg,isca(2))
    phi_starr  =   ptr_NegnwbElm(iel)%ph_star
    eta_k      =   phi - phi_starr
    Npos = ptr_NegnwbElm(iel)%pos
    Nneg = ptr_NegnwbElm(iel)%neg
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        ! &&& scan for isimplxMax per simplex cut type considered &&&
        do isimpx = 1, isimplxMax
            call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi,0.0d0,Area_K,vol2)
            current%vol_phi = vol2
            call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi_starr,eta_k,Area_K,vol1)
            deltaV_k = vol2 - vol1
            icount   = 0
            do
                if ((dabs(deltaV_k) < 1.0d-8).or.(icount > 20)) exit
                icount = icount + 1
                if ( Area_K < tiny(1.0) ) then
                    exit
                endif
                eta_k =  -3.0 * deltaV_k/Area_K     !Changed
                call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi_starr,eta_k,Area_K,vol1)
                deltaV_k = vol2 - vol1
            enddo
            if (current%use_it) then                             !new
                current%eta = eta_k
            else
                current%eta = 0.0
            endif                                                !new
            current => current%next
        enddo ! isimpx loop
    enddo    ! itype loop
enddo  ! iel loop
```

```
!===================Find eta_h, a piecewise constant function (simplex wise mass correction) ==
!*****************************************************************
!          Step 3:: Find Xi_h, the ortogonal projection of eta_h
!*****************************************************************
!===================Find Xi_h (node wise mass correction) == (start)
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    xi_sum = 0.0
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    isim_count = 0
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        ! &&& scan for isimplxMax per simplex cut type considered &&&
        do isimpx = 1, isimplxMax
            if (current%use_it) then                             !new
                isim_count = isim_count  + 1
                xi_sum = xi_sum + current%eta
            endif                                               !new
            current => current%next
        enddo ! isimpx loop
    enddo    ! itype loop
    if (isim_count == 0) then
        isim_count = 1
        !print*,iel,' pos xi_sum = ', xi_sum
    endif
    ptr_nwbElm(iel)%xi_h = xi_sum/real(isim_count)
enddo

!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    xi_sum = 0.0
    Npos = ptr_NegnwbElm(iel)%pos
    if (Npos > 3) then
```

```
        itype_max = 3
    else
        itype_max = Npos
    endif
    isim_count = 0
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        ! &&& scan for isimplxMax per simplex cut type considered &&&
        do isimpx = 1, isimplxMax
            if (current%use_it) then                             !new
                isim_count = isim_count  + 1
                xi_sum = xi_sum + current%eta
            endif                                               !new
            current => current%next
        enddo ! isimpx loop
    enddo    ! itype loop
    if (isim_count == 0) then
        isim_count = 1
        !print*,iel,' neg xi_sum = ', xi_sum
    endif
    ptr_NegnwbElm(iel)%xi_h = xi_sum/real(isim_count)
enddo

!===================Find Xi_h (node wise mass correction) == (end)
!*****************************************************************
!           Step 4(i):: Find psi_h = C xi_h
!*****************************************************************
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    error_iteration = .true.
    c_1     =  0.0
    c_2     =  scale_length !1.0
    fl      =  deltaV(iside,ptr_nwbElm(iel),c_1)
    fh      =  deltaV(iside,ptr_nwbElm(iel),c_2)
    if ( (fh * fl) > 0.0) then
        c_2     =  - scale_length !1.0
        fh      =  deltaV(iside,ptr_nwbElm(iel),c_2)
    endif
    if ( ( (fl * fh) > 0.0).and.(dabs(fl) > fMin).and.(dabs(fh) > fMin)) then
        print*,'Error at iel = ',iel, 'root must be bracketed between arguments'
        print*,iel,fl,fh,fMin
        stop
    else if ((dabs(fl) < fMin).and.(dabs(fh) < fMin)) then
        error_iteration = .false.
        ptr_nwbElm(iel)%cval = 0.0
        continue
    endif
    if ( fl  > 0.0 ) then
        c_l = c_1
        c_h = c_2
    else
        c_l = c_2
        c_h = c_1
        swap = fl
        fl = fh
        fh = swap
    endif
    dc = c_h - c_l
    do j = 1, MAXIT
        if (dabs(fl - fh) < tiny(1.0)) then
            error_iteration = .false.
            exit
        endif
        c_val = c_l + dc * fl/(fl - fh)
        f = deltaV(iside,ptr_nwbElm(iel),c_val)
        if (f < 0.0) then
            del = c_l - c_val
            c_l = c_val
            fl = f
        else
            del = c_h - c_val
            c_h = c_val
            fh  = f
        endif
        dc  = c_h - c_l
        ptr_nwbElm(iel)%fval = f
        ptr_nwbElm(iel)%cval = c_val
        if ( (dabs(del) < c_acc).or.(dabs(f) < tiny(1.0)) ) then
            error_iteration = .false.
            exit
        endif
    enddo ! j loop
    if (error_iteration) then
        print*,'Maximum number of iteration exceeded at iel = ', iel
        stop
    endif
enddo  ! iel loop

!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    error_iteration = .true.
    c_1     =  0.0
```

```
c_2     =   scale_length !1.0
fl      =  deltaV(iside,ptr_NegnwbElm(iel),c_1)
fh      =  deltaV(iside,ptr_NegnwbElm(iel),c_2)
if ( (fh * fl) > 0.0) then
    c_2     =  - scale_length !1.0
    fh      =  deltaV(iside,ptr_NegnwbElm(iel),c_2)
endif
if ( ( (fl * fh) > 0.0).and.(dabs(fl) > fMin).and.(dabs(fh) > fMin)) then
    print*,'Error at iel = ',iel, 'root must be bracketed between arguments'
    print*,iel,fl,fh,fMin
    stop
else if ((dabs(fl) < fMin).and.(dabs(fh) < fMin)) then
    error_iteration = .false.
    ptr_NegnwbElm(iel)%cval = 0.0
    continue
endif
if ( fl  > 0.0 ) then
    c_l = c_1
    c_h = c_2
else
    c_l = c_2
    c_h = c_1
    swap = fl
    fl = fh
    fh = swap
endif
dc = c_h - c_l
do j = 1, MAXIT
    if (dabs(fl - fh) < tiny(1.0)) then
        error_iteration = .false.
        exit
    endif
    c_val = c_l + dc * fl/(fl - fh)
    f = deltaV(iside,ptr_NegnwbElm(iel),c_val)
    if (f < 0.0) then
        del = c_l - c_val
        c_l = c_val
        fl = f
    else
        del = c_h - c_val
        c_h = c_val
        fh  = f
    endif
    dc  = c_h - c_l
    ptr_NegnwbElm(iel)%fval = f
    ptr_NegnwbElm(iel)%cval = c_val
    if ( (dabs(del) < c_acc).or.(dabs(f) < tiny(1.0)) ) then
        error_iteration = .false.
        exit
    endif
enddo ! j loop
if (error_iteration) then
    print*,'Maximum number of iteration exceeded at iel = ', iel
```

```
        stop
    endif
enddo  ! iel loop

!===================Find C  == (start)
!**************************************************************
!           Step 4(ii):: Redistance the Level set first cells surrounding isocontour S_h
!**************************************************************
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    ival_nb    =   ptr_nwbElm(iel)%nwb_index
    ix_val            = ptr_nwbElm(iel)%xi_h
    C_const           = ptr_nwbElm(iel)%cval
    phi_starr         = ptr_nwbElm(iel)%ph_star
    !print*,iel,rtp(ival_nb,isca(2)),phi_starr,C_const,ix_val
    rtp(ival_nb,isca(2)) = phi_starr + ( C_const * ix_val )  !NOTE MOST IMPORTANT PART which upda
enddo  ! iel loop

!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    ix_val            = ptr_NegnwbElm(iel)%xi_h
    C_const           = ptr_NegnwbElm(iel)%cval
    phi_starr         = ptr_NegnwbElm(iel)%ph_star
    !print*,iel,rtp(ival_nbNeg,isca(2)),phi_starr,C_const,ix_val
    rtp(ival_nbNeg,isca(2)) = phi_starr + ( C_const * ix_val )  !NOTE MOST IMPORTANT PART which u
enddo  ! iel loop

!**************************************************************
!           Step 5:: Edge distance approximation
!**************************************************************
! ----------update available secondary nodes on positive side of isosurface--------
iside = 0
ichange = 1
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_counter
        Npos = ptr_nwbElm(iel)%pos
        isnbb = ptr_nwbElm(iel)%Sec_pos
        do i = 1, isnbb
            dImin    = huge(1)
            iph_XI = ptr_nwbElm(iel)%Spos(i)
            a(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
            a(2)   = scale_length * (xyzcen(2,iph_XI))
            a(3)   = scale_length * (xyzcen(3,iph_XI))
            ! |XJ - XI| edge distance approx considered of Npos of them
            do ipos = 1,Npos
                icen = ptr_nwbElm(iel)%Ppos(ipos)
                b(1)   = scale_length * (xyzcen(1,icen))  - a(1)
                b(2)   = scale_length * (xyzcen(2,icen))  - a(2)
                b(3)   = scale_length * (xyzcen(3,icen))  - a(3)
                dI     = rtp(icen,isca(2)) + dsqrt( dabs(dot_product(b,b)) )
                if ( dabs(dImin) > dabs(dI)) then
                    dImin = dI
                endif
            enddo !=======(Find minimum phi_h(X_I) ....end)
            if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                rtp(iph_XI,isca(2)) = dImin  !Edge distance approximation of phi at XI secondary node
                ichange = 1
            endif
        enddo
    enddo
enddo

! -----------update available secondary nodes on negative side of isosurface--------
iside = 1
ichange = 1
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_cnter_Neg
        Nneg = ptr_NegnwbElm(iel)%neg
        isnbb = ptr_NegnwbElm(iel)%Sec_neg
        do i = 1, isnbb
            dImin    = huge(1.0)
            iph_XI = ptr_NegnwbElm(iel)%Sneg(i)
            a(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
            a(2)   = scale_length * (xyzcen(2,iph_XI))
            a(3)   = scale_length * (xyzcen(3,iph_XI))
            ! |XJ - XI| edge distance approx considered of Npos of them
            do ineg = 1,Nneg
                icen = ptr_NegnwbElm(iel)%Pneg(ineg)
                b(1)   = scale_length * (xyzcen(1,icen))  - a(1)
                b(2)   = scale_length * (xyzcen(2,icen))  - a(2)
                b(3)   = scale_length * (xyzcen(3,icen))  - a(3)
                dI     = rtp(icen,isca(2)) - dsqrt( dabs(dot_product(b,b)) )
                if ( dabs(dImin) > dabs(dI)) then
                    dImin = dI
                endif
            enddo !=======(Find minimum phi_h(X_I) ....end)
            if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                rtp(iph_XI,isca(2)) = -dImin  !Edge distance approximation of phi at XI secondary node
                ichange = 1
            endif
        enddo
    enddo
enddo

!-------------------------------------------------------------------------------------
!*****************************************************************
!           Step 6:: Shadow distance correction
!*****************************************************************
! -----------update available secondary nodes on positive side of isosurface--------
```

```
iside = 1
ichange = 0
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_cnter_Neg
        Npos = ptr_NegnwbElm(iel)%pos
        ival_nbNeg    = ptr_NegnwbElm(iel)%nwb_index
        if (Npos > 3) then
            itype_max = 3
        else
            itype_max = Npos
        endif
        ! &&&&&&&&& scan for isimplxMax simplices per simplex cut type considered
        do itype = 1, itype_max
            isnbb = ptr_NegnwbElm(iel)%Sec_neg
            do i = 1, isnbb
                dImin    = huge(1)
                iph_XI = ptr_NegnwbElm(iel)%Sneg(i)
                sec(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
                sec(2)   = scale_length * (xyzcen(2,iph_XI))
                sec(3)   = scale_length * (xyzcen(3,iph_XI))
                ! scan positive faces of simplex
                select case (itype)
                case (1)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
                case (2)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
                case (3)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    if (itype == 1) then
                        ! --------face 1
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .true. ; irV(2)  = current%sim_node(3)
                        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! ---------face 2
                        ntri = 4
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 1
                        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
                        Lr(4) = .false. ; irV(4)  = 2
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! --------face 3
                        ntri = 4
                        Lr(1) = .true. ; irV(1)  = current%sim_node(3)
                        Lr(2) = .false. ; irV(2)  = 3
                        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
                        Lr(4) = .false. ; irV(4)  = 2
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! --------face 4
                        ntri = 4
                        Lr(1) = .true.  ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 1
                        Lr(3) = .true. ; irV(3)  = current%sim_node(3)
                        Lr(4) = .false. ; irV(4)  = 3
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                    else if (itype == 2) then
                        ! ---------face 1
                        ntri = 4
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .true. ; irV(2)  = current%sim_node(3)
                        Lr(3) = .false. ; irV(3)  = 2
                        Lr(4) = .false. ; irV(4)  = 4
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! ---------face 2
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 2
                        Lr(3) = .false. ; irV(3)  = 1
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! ---------face 3
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = current%sim_node(3)
                        Lr(2) = .false. ; irV(2)  = 3
                        Lr(3) = .false. ; irV(3)  = 4
                        call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
```

```
                            ! ---------face 4
                            ntri = 4
                            Lr(1) = .true.  ; irV(1)  = ival_nbNeg
                            Lr(2) = .false. ; irV(2)  = 1
                            Lr(3) = .true. ; irV(3)  = current%sim_node(3)
                            Lr(4) = .false. ; irV(4)  = 3
                            call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                            if ( dabs(dImin) > dabs(dI)) then
                                dImin = dI
                            endif
                        else if (itype == 3) then
                            ! --------face 1
                            ntri = 3
                            Lr(1) = .false. ; irV(1)  = 1
                            Lr(2) = .true. ; irV(2)   = ival_nbNeg
                            Lr(3) = .false. ; irV(3)  = 2
                            call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                            if ( dabs(dImin) > dabs(dI)) then
                                dImin = dI
                            endif
                            ! ---------face 2
                            ntri = 3
                            Lr(1) = .true. ; irV(1)  = ival_nbNeg
                            Lr(2) = .false. ; irV(2)  = 1
                            Lr(3) = .false. ; irV(3)  = 3
                            call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                            if ( dabs(dImin) > dabs(dI)) then
                                dImin = dI
                            endif
                            ! --------face 3
                            ntri = 3
                            Lr(1) = .true. ; irV(1)  = ival_nbNeg
                            Lr(2) = .false. ; irV(2)  = 3
                            Lr(3) = .false. ; irV(3)  = 2
                            call find_min_dist(current,ntri,Lr,irV,sec,iside,dI)
                            if ( dabs(dImin) > dabs(dI)) then
                                dImin = dI
                            endif
                        endif
                        current => current%next
                    enddo !---end of isimplx loop
                    if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                        rtp(iph_XI,isca(2)) = -dImin  !shaddow distance approximation of phi at XI se
                        ichange = 1
                    endif
                enddo !---end of isnbb secondary nodes loop
            enddo !---end of itype loop
        enddo !---end of nwb_cnter_Neg loop
enddo !---ichange  loop !=======(Find minimum phi_h(X_I) ....end)
!-------------------------------------------------------------------------------
!=======================================
!     Initial redistancing (END)
!=======================================
   endif  ! end of isuite if block
   print*,'End of usinv'
   return
   !*********************************************************************************
contains

   !*********************************************************************************
   !
   subroutine find_min_dist(cur,ntri,Lr,ir,sec,iside,dImin)

      use connectivity
      implicit none

      ! code saturne variables
      !integer          ndim   , ncelet
      !double precision rtp(ncelet,*)

      type(simplex), pointer :: cur
      double precision sec(3),dImin
      integer          ntri, iside,ir(4)
      logical          Lr(4)

      ! local variables
      double precision, parameter         ::  small_value = 1.0d-6
      double precision, parameter         ::  th_d = 0.3333333333333
      double precision  k(3),a(3),s1(3),b(3),c(3),s2(3),dif(3),phi_val(4),d(3)
      double precision  v_hat(3),val,tdiv1,tdiv,a_matrix(4,4),v(3),phi_interp
      real  p_x(3),simVec(4,3)

      !------------positive side of free surface---------------------
      if (Lr(1) ) then
          k(1)   = scale_length * (xyzcen(1,ir(1) ) )  ! kth node
          k(2)   = scale_length * (xyzcen(2,ir(1) ) )
          k(3)   = scale_length * (xyzcen(3,ir(1) ) )
      else
          k(1)   = cur%sh_nb(ir(1),1)
          k(2)   = cur%sh_nb(ir(1),2)
          k(3)   = cur%sh_nb(ir(1),3)
      endif
      if (Lr(2) ) then
          s1(1) = scale_length * (xyzcen(1,ir(2)) ) - k(1)
          s1(2) = scale_length * (xyzcen(2,ir(2)) ) - k(2)
          s1(3) = scale_length * (xyzcen(3,ir(2)) ) - k(3)
      else
          s1(1) = cur%sh_nb(ir(2),1) - k(1)
          s1(2) = cur%sh_nb(ir(2),2) - k(2)
          s1(3) = cur%sh_nb(ir(2),3) - k(3)
      endif
      if (Lr(3) ) then
          s2(1) = scale_length * (xyzcen(1,ir(3) ) ) - k(1)
          s2(2) = scale_length * (xyzcen(2,ir(3) ) ) - k(2)
          s2(3) = scale_length * (xyzcen(3,ir(3) ) ) - k(3)
```

```
else
    s2(1) = cur%sh_nb(ir(3),1) - k(1)
    s2(2) = cur%sh_nb(ir(3),2) - k(2)
    s2(3) = cur%sh_nb(ir(3),3) - k(3)
endif
if ( ntri == 4) then
    if (Lr(4) ) then
        d(1)   = scale_length * (xyzcen(1,ir(4) ) )
        d(2)   = scale_length * (xyzcen(2,ir(4) ) )
        d(3)   = scale_length * (xyzcen(3,ir(4) ) )
    else
        d(1)   = cur%sh_nb(ir(4),1)
        d(2)   = cur%sh_nb(ir(4),2)
        d(3)   = cur%sh_nb(ir(4),3)
    endif
endif
call cross_product(s1,s2,v)
a     =  s1 + k
b     =  s2 + k
if ( ntri == 3) then
    c     = (a + b + k) * th_d - sec
else if ( ntri == 4) then
    c     = (a + b + d + k) * 0.25 - sec
endif
val   = dot_product(c,v)
if (val < 0.0d0) call cross_product(s2,s1,v)
tdiv     = dabs(dot_product(v,v))
if (tdiv < small_value ) return
tdiv1    =  dsqrt(tdiv)
v_hat(1) = v(1)/tdiv1
v_hat(2) = v(2)/tdiv1
v_hat(3) = v(3)/tdiv1
dif      = k - sec
val = dot_product(dif,v_hat)
! establish point of intersection p_x on face of simplex
call create_vec(sec, v_hat, val, p_x)
if ( ntri == 3) then
    d = 0.5 * (a + b)
endif
call convert_Vec(4,k,a,d,b,simVec)
if (check_simplex(ntri,simVec,p_x) ) then
    call calc_invert_matrix(simVec,a_matrx,4)
    phi_val(1) = rtp(ir(1), isca(2) )
    phi_val(2) = rtp(ir(2), isca(2) )
    phi_val(4) = rtp(ir(3), isca(2) )
    if ( ntri == 3) then
        phi_val(3) = 0.5 * (phi_val(2) + phi_val(4))
    else if ( ntri == 4) then
        phi_val(3) = rtp(ir(4), isca(2) )
    endif
    phi_interp = calc_phi_interp(4,phi_val,a_matrx,p_x)
    b          =  p_x - sec
    if (iside == 1 ) then
```

```
            dImin = phi_interp + sqrt( dabs(dot_product(b,b)) )
        else
            dImin = phi_interp - sqrt( dabs(dot_product(b,b)) )
        endif
    endif
    return
end subroutine find_min_dist

!*************************************************************************************
!  calculate interpolated level set value from neighbour level set values
!  using 3D geometrically isotropic trilinear interpolation
!*************************************************************************************
double precision function calc_phi_interp(ntri,prop_neighbs,a_matrx,p_x)
    implicit none
    real, dimension(:,:)                            :: a_matrx
    real, dimension(:)                              :: p_x
    double precision, dimension(:)                  :: prop_neighbs
    double precision,dimension(:), allocatable      :: c
    double precision                                :: prop_sum
    integer                                         :: i, j, ntri

    allocate(c(ntri))
    do i = 1, ntri
        c(i) = 0.0
        do j = 1, ntri
            c(i) = c(i) + a_matrx(i,j) * prop_neighbs(i)
        enddo
    enddo
    prop_sum = 0.0
    do i = 1, ntri
        if (i == 1) then
            prop_sum = prop_sum + c(i)
        else
            prop_sum = prop_sum + c(i) * p_x(i-1)
        endif
    enddo
    calc_phi_interp = prop_sum
    deallocate(c)
    return
end function calc_phi_interp

!******************************************************************************************
!  create_simplices
!
!  This subroutine creates the simplices associated with node k and its neighbour
!              cells forming part of the free surface region near node k
!******************************************************************************************
subroutine create_simplices(iside,ptr, Nneg, Npos,itype_max)
    use connectivity
    implicit none

    type (narrow_band_element)        :: ptr
    integer                           :: Nneg, Npos, itype_max,iside
```

```
! local variables
type(simplex), pointer           :: current,previous
integer itype,isimplex, inext, isamplePos,p(3), icount

do itype = 1, itype_max
    if (itype == 1) then
        nullify(ptr%ptr_cutTetra_type1)
        allocate(ptr%ptr_cutTetra_type1)
        current => ptr%ptr_cutTetra_type1
        current%use_it = .true.
        isimplex = 0
        do isampleNeg = 1, Nneg
            if (iside == 0) then       ! positive side
                p(1)     = ptr%Pneg(isampleNeg)
            else if (iside == 1) then  ! negative side
                p(1)     = ptr%Ppos(isampleNeg)
            endif
            inext = 1
            do isamplePos = 1, Npos
                if (iside == 0) then      ! positive side
                    p(2) = ptr%Ppos(isamplePos)
                else if (iside == 1) then ! negative side
                    p(2) = ptr%Pneg(isamplePos)
                endif
                inext = inext + 1
                if (inext == Npos) inext = 1
                if (iside == 0) then      ! positive side
                    p(3) = ptr%Ppos(inext)
                else if (iside == 1) then ! negative side
                    p(3) = ptr%Pneg(inext)
                endif
                isimplex = isimplex + 1
                current%sim_node(1) = p(1)
                current%sim_node(2) = p(2)
                current%sim_node(3) = p(3)
                allocate(current%next)
                nullify( current%next%next )
                current => current%next
                current%use_it = .true.
            enddo
        enddo
        ptr%ptr_cutTetra_type1%sim_count = isimplex
    else if (itype == 2) then
        nullify(ptr%ptr_cutTetra_type2)
        allocate(ptr%ptr_cutTetra_type2)
        current => ptr%ptr_cutTetra_type2
        current%use_it = .true.
        isimplex = 0
        inext = 1
        do isampleNeg = 1, Nneg
            if (iside == 0) then       ! positive side
                p(1)     = ptr%Pneg(isampleNeg)
            else if (iside == 1) then  ! negative side
                p(1)     = ptr%Ppos(isampleNeg)
            endif
            inext    = inext + 1
            if (inext == Nneg) inext = 1
            if (iside == 0) then       ! positive side
                p(2)     =   ptr%Pneg(inext)
            else if (iside == 1) then  ! negative side
                p(2)     =   ptr%Ppos(inext)
            endif
            do isamplePos = 1, Npos
                if (iside == 0) then      ! positive side
                    p(3) = ptr%Ppos(isamplePos)
                else if (iside == 1) then ! negative side
                    p(3) = ptr%Pneg(isamplePos)
                endif
                isimplex = isimplex + 1
                current%sim_node(1) = p(1)
                current%sim_node(2) = p(2)
                current%sim_node(3) = p(3)
                allocate(current%next)
                nullify( current%next%next )
                current => current%next
                current%use_it = .true.
            enddo
        enddo
        ptr%ptr_cutTetra_type2%sim_count = isimplex
    else if (itype == 3) then
        nullify(ptr%ptr_cutTetra_type3)
        allocate(ptr%ptr_cutTetra_type3)
        current => ptr%ptr_cutTetra_type3
        current%use_it = .true.
        isimplex = 0
        inext    = 1 ; inext2 = 2
        do isampleNeg = 1, Nneg
            if (iside == 0) then       ! positive side
                p(1)     = ptr%Pneg(isampleNeg)
            else if (iside == 1) then  ! negative side
                p(1)     = ptr%Ppos(isampleNeg)
            endif
            inext = inext + 1
            if (inext == Nneg) inext = 1
            if (iside == 0) then       ! positive side
                p(2)     =   ptr%Pneg(inext)
            else if (iside == 1) then  ! negative side
                p(2)     =   ptr%Ppos(inext)
            endif
            inext2 = inext2 + 1
            if (inext2 == Nneg) inext2 = 1
            if (iside == 0) then       ! positive side
                p(3)     =   ptr%Pneg(inext2)
            else if (iside == 1) then  ! negative side
                p(3)     =   ptr%Ppos(inext2)
```

```
                endif
                isimplex = isimplex + 1
                current%sim_node(1) = p(1)
                current%sim_node(2) = p(2)
                current%sim_node(3) = p(3)
                allocate(current%next)
                nullify( current%next%next )
                current => current%next
                current%use_it = .true.
            enddo
            ptr%ptr_cutTetra_type3%sim_count = isimplex
        endif
    enddo
    return

end subroutine create_simplices


!*******************************************************************************************************
!  secondary_cell_select
!
!  This subroutine finds what cells are secondary cells in the 3D environment either side of the isos
!*******************************************************************************************************
integer function  secondary_cell_select(ineighb)
   use connectivity
   implicit none
   ! code saturne variables
   ! integer          ndim   , ncelet
   ! double precision rtp(ncelet,*)

   integer(8), intent(in )                :: ineighb


   ! local variables
   integer             ifac, isgn, Number_Of_Faces
   double precision    phi_cenck, phi_nbck
   secondary_cell_select = 1
   phi_cenck       = rtp(ineighb,isca(2))
   if ( dabs(phi_cenck) < tiny(1.0)) then
       secondary_cell_select = 0
       return
   else if ( phi_cenck < 0.0d0) then
       isgn = -1
   else if ( phi_cenck > 0.0d0) then
       isgn = 1
   endif
   Number_Of_Faces=count(nbcell(ineighb,:)>-ihuge)
   do ifac = 1, Number_Of_Faces
       phi_nbck  = rtp(nbcell(ineighb,ifac),isca(2))
       if ( isgn > 0) then
           if ( phi_nbck < 0.0d0) then
               secondary_cell_select = -1
               return
           endif
       else if ( isgn < 0 ) then
           if ( phi_nbck > 0.0d0) then
               secondary_cell_select = -1
               return
           endif
       endif
   enddo  ! ifac loop
end function secondary_cell_select
!**************************************************************************************
!***********************************************************************************************************
!  Vol_k
!
!  This subroutine calculates the difference in volumes defined by phi and phi*,
!  piecewise constant simplexwise mass correction function
!  and 3D reconstruction of the isosurface locally over simplex k
!***********************************************************************************************************
subroutine Vol_k(icheck,iside,current,ival_nb,itype,phi_value,val_adj,Area_K,Volume_k)
   use connectivity
   implicit none


   type(simplex), pointer, intent(in)            :: current

   logical, intent(in)                           :: icheck
   integer, intent(in)                           :: iside
   integer, intent(in)                           :: itype
   integer, intent(in)                           :: ival_nb
   double precision, intent(in )                 :: phi_value,val_adj
   double precision, intent(out)                 :: Area_K
   double precision, intent(out)                 :: Volume_k

   ! local variables
   double precision, parameter                   ::  vol_const = 0.16666667
   double precision  a(3),b(3),c(3),c1(3),c4(3),k(3),dif(3),v(3),s(3)
   double precision  sh_nb_temp(4,3),s_val, S_K, S_h, phi,vol_A, vol_B
   double precision  Area_K2,Area_K1,s1(3),s2(3),p3(3),pmid(3),vol_01
   double precision  vol_021,vol_022,vol_03,c2(3),c3(3),Area_1
   integer           ieln, inod2,Nmax,is1, ir, it_value1, it_value2

   !--------------------------------------------------------------------------
   ! Adjust reconstructed isosurface Sk by 'valadj' amount normal to surface
   !--------------------------------------------------------------------------
   select case (itype)
   case (1)
       Nmax = 3
   case (2)
       Nmax = 4
   case (3)
       Nmax = 3
   case default
       call error_message('Error1 in Vol_k in USINV')
   end select
```

```
!----------------------
if (icheck) then
    do inod2 = 1, Nmax
        sh_nb_temp(inod2,1) = current%sh_nb(inod2,1)
        sh_nb_temp(inod2,2) = current%sh_nb(inod2,2)
        sh_nb_temp(inod2,3) = current%sh_nb(inod2,3)
    enddo
else
    do inod2 = 1, Nmax
        if (itype == 1) then
            if ( inod2 == 1) then
                ir   =  ival_nb                ! Kth node
                phi  =     phi_value + val_adj
            else
                ir  =  current%sim_node(inod2)
                phi =  rtp(ir,isca(2))  + val_adj        ! new
            endif
            ieln =  current%sim_node(1)
        else if (itype == 2) then
            select case (inod2)
            case (1,2)
                ieln  =  current%sim_node(inod2)
                ir   =  ival_nb
                phi = phi_value + val_adj               ! new
            case (3,4)
                ival = inod2 - 2
                ieln  =  current%sim_node(ival)
                ir   =  current%sim_node(3)
                phi = rtp(ir,isca(2)) + val_adj         ! new
            case default
                call error_message('Error2 in Vol_k in USINV')
            end select
        else if (itype == 3) then
            ieln  =  current%sim_node(inod2)
            ir    =  ival_nb
            phi = phi_value + val_adj                 ! new
        else
            call error_message('Error3 in Vol_k in USINV')
        endif
        s_val    = phi  -  rtp(ieln,isca(2))
        call check_zero(s_val,'Error1 in USINV with Sk reconstruction at iel = ',iel)
        S_h  = - rtp(ieln,isca(2))/s_val
        a(1) = scale_length * (xyzcen(1,ir) )
        a(2) = scale_length * (xyzcen(2,ir) )
        a(3) = scale_length * (xyzcen(3,ir) )
        b(1) = scale_length * (xyzcen(1,ieln) )
        b(2) = scale_length * (xyzcen(2,ieln) )
        b(3) = scale_length * (xyzcen(3,ieln) )
        call position_vec(a, b, S_h, c)
        sh_nb_temp(inod2,1) = c(1)
        sh_nb_temp(inod2,2) = c(2)
        sh_nb_temp(inod2,3) = c(3)
    enddo
```

```
endif
!---------------------------------------------------------------------------
! calculate new simplex volume based on adjustment to reconstructed isosurface Sk
!---------------------------------------------------------------------------
! volume of small negative simplex
c1(1) = sh_nb_temp(1,1)  ! edge of free surface near kth node
c1(2) = sh_nb_temp(1,2)
c1(3) = sh_nb_temp(1,3)
if ( itype == 2 ) then
    !find volume of first sub-tetrahedron
    ir   =  current%sim_node(3)
    do id = 1,ndim
        a(id)  = scale_length * (xyzcen(id,ival_nb) )
        c2(id) = sh_nb_temp(2,id)
        c3(id) = sh_nb_temp(3,id)
        c4(id) = sh_nb_temp(4,id)
        p3(id) = scale_length * (xyzcen(id,ir) )
    enddo
    !---- vol01 volume----------------
    pmid = 0.5 * (p3 + a)
    s1 = pmid - a
    s2 = c1   - a
    call cross_product(s1,s2,v)
    dif   = c2 - a
    vol_01 = dabs(dot_product(dif, v))
    !vol_021 volume
    s1  = c1 - c2
    s2  = c4 - c2
    dif = pmid - c2
    call cross_product(s1,s2,v)
    vol_021 = dabs(dot_product(dif, v))
    !vol_022 volume
    s1  = c1 - c3
    s2  = c4 - c3
    dif = pmid - c3
    call cross_product(s1,s2,v)
    vol_022 = dabs(dot_product(dif, v))
    !vol_03 volume
    s1  = p3 - c4
    s2  = pmid - c2
    dif = pmid - c4
    call cross_product(s1,s2,v)
    vol_03 = dabs(dot_product(dif, v))
    if ( iside == 0) then
        Volume_k = vol_const*(vol_01 + vol_021 + vol_022 + vol_03)
    else
        vol_A = vol_01 + vol_021 + vol_022 + vol_03
    endif
    ! find Area_k
    s1 = c3 - c1
    s2 = c2 - c1
    call cross_product(s1,s2,v)
    S_k     = dabs(dot_product(v,v))
```

```
        if (S_k < tiny(1.0)) then
            Area_1 = 0.0
        else
            Area_1 = 0.5 * dsqrt(S_k)
        endif
        s1 = c3 - c4
        s2 = c2 - c4
        call cross_product(s1,s2,v)
        S_k     = dabs(dot_product(v,v))
        if (S_k < tiny(1.0)) then
            Area_K = Area_1
        else
            Area_K = (0.5 * dsqrt(S_k)) + Area_1
        endif
        if (iside == 1) then
            ieln = current%sim_node(1)
            do id = 1,ndim
                a(id)  = scale_length * (xyzcen(id,ival_nb) )
                b(id)  = scale_length * (xyzcen(id,ieln) )
            enddo
            dif = b - a
            is1 = current%sim_node(2)
            s1(1)   = scale_length * (xyzcen(1,is1) ) - a(1)
            s1(2)   = scale_length * (xyzcen(2,is1) ) - a(2)
            s1(3)   = scale_length * (xyzcen(3,is1) ) - a(3)
            is1 = current%sim_node(3)
            s2(1)   = scale_length * (xyzcen(1,is1) ) - a(1)
            s2(2)   = scale_length * (xyzcen(2,is1) ) - a(2)
            s2(3)   = scale_length * (xyzcen(3,is1) ) - a(3)
            call cross_product(s1,s2,v)
            vol_B = dabs(dot_product(dif,v))
            volume_K = vol_const * (vol_B - vol_A)
        endif
        return
    else
        if (itype == 1) then
            ieln =  current%sim_node(1)
        else if (itype == 3) then
            ieln = ival_nb   ! positive peak at node k
        else
            call error_message('Error4 in Vol_k in USINV')
        endif
        b(1) = scale_length * (xyzcen(1,ieln) )
        b(2) = scale_length * (xyzcen(2,ieln) )
        b(3) = scale_length * (xyzcen(3,ieln) )
        dif(1) = b(1) - c1(1)   ! height of small simplex
        dif(2) = b(2) - c1(2)
        dif(3) = b(3) - c1(3)
        s1(1) = sh_nb_temp(2,1) - c1(1)
        s1(2) = sh_nb_temp(2,2) - c1(2)
        s1(3) = sh_nb_temp(2,3) - c1(3)
        s2(1) = sh_nb_temp(3,1) - c1(1)
        s2(2) = sh_nb_temp(3,2) - c1(2)
```

```
      s2(3) = sh_nb_temp(3,3) - c1(3)
      call cross_product(s1,s2,v)
      S_k     = dabs(dot_product(v,v))
      if (S_k < tiny(1.0)) then
          Area_K       = 0.0
      else
          Area_K       = 0.5 * dsqrt(S_k)
      endif
      vol_A =  dabs(dot_product(dif, v))
      if (iside == 0) then
          it_value1 = 3
          it_value2 = 1
      else if (iside == 1) then
          it_value1 = 1
          it_value2 = 3
      endif
      if (itype == it_value1) then
          Volume_k = vol_const * vol_A
          return
      else if (itype == it_value2) then
          !------ larger simplex next
          if (iside == 0) then
              ieln =  ival_nb
          else if (iside == 1) then
              ieln =  current%sim_node(1)
          endif
          a(1) = scale_length * (xyzcen(1,ieln) )
          a(2) = scale_length * (xyzcen(2,ieln) )
          a(3) = scale_length * (xyzcen(3,ieln) )
          dif(1) = b(1) - a(1)   ! height of the larger simplex
          dif(2) = b(2) - a(2)
          dif(3) = b(3) - a(3)
          is1 =  current%sim_node(2)
          s1(1) = scale_length * (xyzcen(1,is1) ) - a(1)
          s1(2) = scale_length * (xyzcen(2,is1) ) - a(2)
          s1(3) = scale_length * (xyzcen(3,is1) ) - a(3)
          is1 =  current%sim_node(3)
          s2(1) = scale_length * (xyzcen(1,is1) ) - a(1)
          s2(2) = scale_length * (xyzcen(2,is1) ) - a(2)
          s2(3) = scale_length * (xyzcen(3,is1) ) - a(3)
          call cross_product(s1,s2,v)
          vol_B =  dabs(dot_product(dif, v))
          !-------resultant simplex volume below free surface
          Volume_k = vol_const * (vol_B - vol_A)
          return
      else
          call error_message('Error5 in Vol_k in USINV')
      endif
   endif
end subroutine Vol_k
!*****************************************************************************************
!  deltaV
!
```

```
!  This subroutine calculates the difference in volumes defined by phi and phi*, node wise mass corre
!  which is used in the false position algorithm to find the constant C which globally preserves volu
!*****************************************************************************************************
double precision function deltaV(iside,ptr, C)
   use connectivity
   implicit none


   type (narrow_band_element), intent(in)       :: ptr
   double precision, intent(in )                :: C
   integer,intent(in)                           :: iside

   ! local variables
   integer             Nneg, Npos,itype_max,itype,isimpx,isimplxMax
   integer             ival_nb
   double precision    s_k, phi, phi_starr, Cxi_h, f_sum, Area_K
   double precision    vol2, vol1

   f_sum = 0.0
   Cxi_h = ptr%xi_h * C
   phi_starr  =   ptr%ph_star
   Npos = ptr%pos
   Nneg = ptr%neg
   ival_nb    =   ptr%nwb_index
   if (iside == 0) then
       if (Nneg > 3) then
           itype_max = 3
       else
           itype_max = Nneg
       endif
   else if (iside == 1) then
       if (Npos > 3) then
           itype_max = 3
       else
           itype_max = Npos
       endif
   endif
   do itype = 1, itype_max
       select case (itype)
       case (1)
           isimplxMax = ptr%ptr_cutTetra_type1%sim_count
           current => ptr%ptr_cutTetra_type1
       case (2)
           isimplxMax = ptr%ptr_cutTetra_type2%sim_count
           current => ptr%ptr_cutTetra_type2
       case (3)
           isimplxMax = ptr%ptr_cutTetra_type3%sim_count
           current => ptr%ptr_cutTetra_type3
       case default
           call error_message('Error1 in step1 with isimplxMax in USINV')
       end select
       ! &&& scan for isimplxMax per simplex cut type considered &&&
       if (isimplxMax > 0) then
```

```
         do isimpx = 1, isimplxMax
             if (current%use_it) then                              !new
                 vol2 = current%vol_phi
                 call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,Cxi_h,Area_K,vol1)
                 f_sum = f_sum + (vol2 - vol1)
             endif                                                 !new
             current => current%next
         enddo
     endif
   enddo
   deltaV = f_sum
   return
end function deltaV
!========================================
!-----------------------------------------------------------------------------------------------------
!  Returns inverted matrix for step 6 needed with 3D geometrically isotropic
!  trilinear interpolation involved with the intersection on a face of a tetrahedral simplex
!-----------------------------------------------------------------------------------------------------
subroutine calc_invert_matrix(x,a_matrx,ntri)
   implicit none
   real, dimension(:,:), allocatable :: Matrix, invMatrix
   real, dimension(:,:)              :: a_matrx, x
   integer                           :: ntri

   allocate(Matrix(ntri,ntri))
   allocate(invMatrix(ntri,ntri))
   do i = 1, ntri
       do j = 1, ntri
           if (j == 1) then
               Matrix(i,j) = 1.0
           else
               Matrix(i,j) = x(i,j-1)
           endif
       enddo
   enddo
   call FindInv(Matrix, invMatrix, ntri, ErrorFlag)
   do i = 1, ntri
       do j = 1, ntri
           a_matrx(i,j) = invMatrix(i,j)
       enddo
   enddo
   deallocate(Matrix)
   deallocate(invMatrix)
   return
end subroutine calc_invert_matrix
!--------------------------------------------------------------------------------
!  check to ensure that the intersection p_x is within the particular face
!  of the tetrahedral simplex
!--------------------------------------------------------------------------------
logical function check_simplex(n_tri,simVec,p_x)
   implicit none
   real, dimension(:,:)             :: simVec
   real, dimension(:)               :: p_x
   ! local variables
   double precision  n_vec(3),r_mid(3),nface_hat(3),sim_dif(3), t_vec(3),val
   double precision  v_smll,n_val,dval
   integer id,n_tri

   do i = 1, n_tri
       do id = 1, ndim
           if (i < n_tri) then
               t_vec(id) = simVec(i+1,id) - simVec(i,id)
               sim_dif(id) = (simVec(i+1,id) + simVec(i,id) ) * 0.5 - p_x(id)
           else
               t_vec(id) = simVec(1,id) - simVec(i,id)
               sim_dif(id) = (simVec(1,id) + simVec(i,id) ) * 0.5 - p_x(id)
           endif
       enddo
       call cross_product(nface_hat, t_vec, n_vec)
       n_val = dsqrt(dabs(dot_product(n_vec,n_vec)))
       if ( n_val < tiny(1.0))  then
           check_simplex = .true.
           return
       endif
       n_vec(1) = n_vec(1)/n_val;n_vec(2) = n_vec(2)/n_val;n_vec(3) = n_vec(3)/n_val;
       val = dot_product(sim_dif, n_vec)
       dval = dsqrt(dabs(dot_product(sim_dif,sim_dif)))
       v_smll = dval * 0.01
       if (dabs(val) < v_smll) then
           check_simplex = .true.
       else if (val >= 0.0) then
           check_simplex = .true.
       else
           check_simplex = .false.
           return
       endif
   enddo
   return
end function check_simplex
!----------------------------------------------------------------------------------

!==========================================
subroutine convert_Vec(ntri,k,a,b,c,simVec)

   implicit none

   integer                         , intent (IN)    :: ntri
   double precision, dimension(:), intent (IN)    :: k
   double precision, dimension(:), intent (IN)    :: a
   double precision, dimension(:), intent (IN)    :: b
   double precision, dimension(:), intent (IN)    :: c
   real          , dimension(:,:), intent (OUT)   :: simVec

   do i = 1, ntri
       if (i == 1) then
```

```
            do j = 1, ndim
                simVec(i,j) = k(j)
            enddo
        else if (i == 2) then
            do j = 1, ndim
                simVec(i,j) = a(j)
            enddo
        else if (i == 3) then
            do j = 1, ndim
                simVec(i,j) = b(j)
            enddo
        else if (i == 4) then
            do j = 1, ndim
                simVec(i,j) = c(j)
            enddo
        endif
    enddo
    return

end subroutine convert_Vec
!========================================
subroutine create_vec (A, B, TSCAL, C)

   implicit none

   double precision, dimension(3), intent (IN)    :: A                               ! multiplicand 3-vector
   double precision, dimension(3), intent (IN)    :: B                               ! multiplier 3-vector
   double precision,               intent (IN)    :: TSCAL                           ! scalar
   real,             dimension(3), intent (OUT)   :: C                               ! result: 3-vector position

   C(1) = A(1) + TSCAL * B(1)
   C(2) = A(2) + TSCAL * B(2)
   C(3) = A(3) + TSCAL * B(3)
   return

end subroutine create_vec
!=========================================
!=========================================
subroutine POSITION_VEC (A, B, TSCAL, C)

   implicit none

   double precision, dimension(3), intent (IN)    :: A                               ! multiplicand 3-vector
   double precision, dimension(3), intent (IN)    :: B                               ! multiplier 3-vector
   double precision,               intent (IN)    :: TSCAL                           ! scalar
   double precision, dimension(3), intent (OUT)   :: C                               ! result: 3-vector position
   !local variables
   double precision, dimension(3)                 :: D

   D(1) = B(1) - A(1)
   D(2) = B(2) - A(2)
   D(3) = B(3) - A(3)
   C(1) = A(1) + TSCAL * D(1)
   C(2) = A(2) + TSCAL * D(2)
   C(3) = A(3) + TSCAL * D(3)
   return

end subroutine POSITION_VEC
!========================================

function dot_product (V1, V2) result (PROD)

   implicit none

   double precision, dimension(3), intent(IN) :: V1, V2
   double precision :: PROD


   PROD = V1(1)*V2(1) + V1(2)*V2(2) + V1(3)*V2(3)
   return

end function DOT_PRODUCT
!**********************************************************************************************
!**********************************************************************************************
!  CROSS_PRODUCT
!
!  Returns the right-handed vector cross product of two 3d-vectors:  C = A x B.
!
!  Code pasted from: http://www.davidgsimpson.com/software/crossprd_f90.txt
!  Checked veracity with Wikipedia
!**********************************************************************************************

subroutine CROSS_PRODUCT (A, B, C)                                         ! cross product (right-

   implicit none                                                              ! no default typing

   double precision, dimension(3), intent (IN)    :: A                        ! multiplicand 3d-ve
   double precision, dimension(3), intent (IN)    :: B                        ! multiplier 3d-vect
   double precision, dimension(3), intent (OUT)   :: C                        ! result: 3d-vector


   C(1) = A(2)*B(3) - A(3)*B(2)                                               ! compute cross prod
   C(2) = A(3)*B(1) - A(1)*B(3)
   C(3) = A(1)*B(2) - A(2)*B(1)

   return

end subroutine CROSS_PRODUCT

!----------------------------------------------------------------------------------
!Subroutine to find the inverse of a square matrix
!Author : Louisda16th a.k.a Ashwith J. Rego
!Reference : Algorithm has been well explained in:
!http://math.uww.edu/~mcfarlat/inverse.htm
!http://www.tutor.ms.unimelb.edu.au/matrix/matrix_inverse.html
!------------------------------------------------------------------------------------
```

```
subroutine FINDInv(matrix, inverse, n, errorflag)
   implicit none
   !Declarations
   integer, intent(IN) :: n
   integer, intent(OUT) :: errorflag  !Return error status. -1 for error, 0 for normal
   real, intent(IN), dimension(n,n) :: matrix  !Input matrix
   real, intent(OUT), dimension(n,n) :: inverse !Inverted matrix

   logical :: FLAG = .true.
   integer :: i, j, k, l
   real :: m
   real, dimension(n,2*n) :: augmatrix !augmented matrix

   !Augment input matrix with an identity matrix
   do i = 1, n
       do j = 1, 2*n
           if (j <= n ) then
               augmatrix(i,j) = matrix(i,j)
           else if ((i+n) == j) then
               augmatrix(i,j) = 1
           else
               augmatrix(i,j) = 0
           endif
       end do
   end do

   !Reduce augmented matrix to upper traingular form
   do k =1, n-1
       if (augmatrix(k,k) == 0) then
           FLAG = .false.
           do i = k+1, n
               if (augmatrix(i,k) /= 0) then
                   do j = 1,2*n
                       augmatrix(k,j) = augmatrix(k,j)+augmatrix(i,j)
                   end do
                   FLAG = .true.
                   exit
               endif
               if (FLAG .eqv. .false.) then
                   print*, "Matrix is non - invertible"
                   inverse = 0
                   errorflag = -1
                   return
               endif
           end do
       endif
       do j = k+1, n
           m = augmatrix(j,k)/augmatrix(k,k)
           do i = k, 2*n
               augmatrix(j,i) = augmatrix(j,i) - m*augmatrix(k,i)
           end do
       end do
   end do
```

```
      !Test for invertibility
      do i = 1, n
          if (augmatrix(i,i) == 0) then
              print*, "Matrix is non - invertible"
              inverse = 0
              errorflag = -1
              return
          endif
      end do

      !Make diagonal elements as 1
      do i = 1 , n
          m = augmatrix(i,i)
          do j = i , (2 * n)
              augmatrix(i,j) = (augmatrix(i,j) / m)
          end do
      end do

      !Reduced right side half of augmented matrix to identity matrix
      do k = n-1, 1, -1
          do i =1, k
              m = augmatrix(i,k+1)
              do j = k, (2*n)
                  augmatrix(i,j) = augmatrix(i,j) -augmatrix(k+1,j) * m
              end do
          end do
      end do

      !store answer
      do i =1, n
          do j = 1, n
              inverse(i,j) = augmatrix(i,j+n)
          end do
      end do
      errorflag = 0
   end subroutine FINDinv
end subroutine usiniv
```

Appendix C

usproj.f90

```
!-------------------------------------------------------------------------------
!                      Code_Saturne version 2.0.0-rc1
!                      --------------------------

!     This file is part of the Code_Saturne Kernel, element of the
!     Code_Saturne CFD tool.

!     Copyright (C) 1998-2009 EDF S.A., France

!     contact: saturne-support@edf.fr

!     The Code_Saturne Kernel is free software; you can redistribute it
!     and/or modify it under the terms of the GNU General Public License
!     as published by the Free Software Foundation; either version 2 of
!     the License, or (at your option) any later version.

!     The Code_Saturne Kernel is distributed in the hope that it will be
!     useful, but WITHOUT ANY WARRANTY; without even the implied warranty
!     of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!     GNU General Public License for more details.

!     You should have received a copy of the GNU General Public License
!     along with the Code_Saturne Kernel; if not, write to the
!     Free Software Foundation, Inc.,
!     51 Franklin St, Fifth Floor,
!     Boston, MA  02110-1301  USA

!-------------------------------------------------------------------------------

subroutine usproj &
      !================

      ( idbia0 , idbra0 ,                                              &
      ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
      nnod   , lndfac , lndfbr , ncelbr ,                            &
```

```
   nvar   , nscal  , nphas  ,                                    &
   nbpmax , nvp    , nvep   , nivep  , ntersl , nvlsta , nvisbr , &
   nideve , nrdeve , nituse , nrtuse ,                          &
   ifacel , ifabor , ifmfbr , ifmcel , iprfml , maxelt , lstelt , &
   ipnfac , nodfac , ipnfbr , nodfbr , itepa  ,                 &
   idevel , ituser , ia     ,                                    &
   xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
   dt     , rtpa   , rtp    , propce , propfa , propfb ,         &
   coefa  , coefb  ,                                             &
   ettp   , ettpa  , tepa   , statis , stativ , tslagr , parbor , &
   rdevel , rtuser , ra     )

!=======================================
! Purpose:
! -------

!    User subroutine.

!    Called at end of each time step, very general purpose
!    (i.e. anything that does not have another dedicated user subroutine)


! Several examples are given here:

!  - compute a thermal balance
!    (if needed, see note  below on adapting this to any scalar)

!  - compute global efforts on a subset of faces

!  - arbitrarily modify a calculation variable

!  - extract a 1 d profile

!  - print a moment

!  - examples on using parallel utility functions

! These examples are valid when using periodicity (iperio .gt. 0)
! and in parallel (irangp .ge. 0).

! The thermal balance compution also illustates a few other features,
! including the required precautions in parallel or with periodicity):
! - gradient calculation
! - computation of a value depending on cells adjacent to a face
!   (see synchronization of Dt and Cp)
! - computation of a global sum in parallel (parsom)


! Cells, boundary faces and interior faces identification
! =======================================================

! Cells, boundary faces and interior faces may be identified using
! the subroutines 'getcel', 'getfbr' and 'getfac' (respectively).
!  getfbr(string, nelts, eltlst):
!  - string is a user-supplied character string containing selection criteria;
!  - nelts is set by the subroutine. It is an integer value corresponding to
!    the number of boundary faces verifying the selection criteria;
!  - lstelt is set by the subroutine. It is an integer array of size nelts
!    containing the list of boundary faces verifying the selection criteria.

!  string may contain:
!  - references to colors (ex.: 1, 8, 26, ...)
!  - references to groups (ex.: inlet, group1, ...)
!  - geometric criteria (ex. x < 0.1, y >= 0.25, ...)
!  These criteria may be combined using logical operators ('and', 'or') and
!  parentheses.
!  Example: '1 and (group2 or group3) and y < 1' will select boundary faces
!  of color 1, belonging to groups 'group2' or 'group3' and with face center
!  coordinate y less than 1.

! Similarly, interior faces and cells can be identified using the 'getfac'
! and 'getcel' subroutines (respectively). Their syntax are identical to
! 'getfbr' syntax.

! For a more thorough description of the criteria syntax, it can be referred
! to the user guide.


!-------------------------------------------------------------------------------
! Arguments
!__________________.____._____.________________________________________________.
! name             !type!mode ! role                                           !
!__________________!____!_____!________________________________________________!
! idbia0           ! i  ! <-- ! number of first free position in ia            !
! idbra0           ! i  ! <-- ! number of first free position in ra            !
! ndim             ! i  ! <-- ! spatial dimension                              !
! ncelet           ! i  ! <-- ! number of extended (real + ghost) cells        !
! ncel             ! i  ! <-- ! number of cells                                !
! nfac             ! i  ! <-- ! number of interior faces                       !
! nfabor           ! i  ! <-- ! number of boundary faces                       !
! nfml             ! i  ! <-- ! number of families (group classes)             !
! nprfml           ! i  ! <-- ! number of properties per family (group class)  !
! nnod             ! i  ! <-- ! number of vertices                             !
! lndfac           ! i  ! <-- ! size of nodfac indexed array                   !
! lndfbr           ! i  ! <-- ! size of nodfbr indexed array                   !
! ncelbr           ! i  ! <-- ! number of cells with faces on boundary         !
! nvar             ! i  ! <-- ! total number of variables                      !
! nscal            ! i  ! <-- ! total number of scalars                        !
! nphas            ! i  ! <-- ! number of phases                               !
! nbpmax           ! i  ! <-- ! max. number of particles allowed               !
! nvp              ! i  ! <-- ! number of particle-defined variables           !
! nvep             ! i  ! <-- ! number of real particle properties             !
! nivep            ! i  ! <-- ! number of integer particle properties          !
! ntersl           ! i  ! <-- ! number of return coupling source terms         !
! nvlsta           ! i  ! <-- ! number of Lagrangian statistical variables     !
! nvisbr           ! i  ! <-- ! number of boundary statistics                 !
! nideve, nrdeve   ! i  ! <-- ! sizes of idevel and rdevel arrays             !
! nituse, nrtuse   ! i  ! <-- ! sizes of ituser and rtuser arrays             !
! ifacel(2, nfac)  ! ia ! <-- ! interior faces -> cells connectivity          !
! ifabor(nfabor)   ! ia ! <-- ! boundary faces -> cells connectivity          !
! ifmfbr(nfabor)   ! ia ! <-- ! boundary face family numbers                  !
! ifmcel(ncelet)   ! ia ! <-- ! cell family numbers                           !
! iprfml           ! ia ! <-- ! property numbers per family                   !
!  (nfml, nprfml)  !    !     !                                               !
! maxelt           ! i  ! <-- ! max number of cells and faces (int/boundary)  !
! lstelt(maxelt)   ! ia ! --- ! work array                                    !
! ipnfac(nfac+1)   ! ia ! <-- ! interior faces -> vertices index (optional)   !
! nodfac(lndfac)   ! ia ! <-- ! interior faces -> vertices list (optional)    !
! ipnfbr(nfabor+1) ! ia ! <-- ! boundary faces -> vertices index (optional)   !
! nodfbr(lndfbr)   ! ia ! <-- ! boundary faces -> vertices list (optional)    !
! itepa            ! ia ! <-- ! integer particle attributes                   !
!  (nbpmax, nivep) !    !     !   (containing cell, ...)                      !
! idevel(nideve)   ! ia ! <-- ! integer work array for temporary development  !
! ituser(nituse)   ! ia ! <-- ! user-reserved integer work array              !
! ia(*)            ! ia ! --- ! main integer work array                       !
! xyzcen           ! ra ! <-- ! cell centers                                  !
!  (ndim, ncelet)  !    !     !                                               !
! surfac           ! ra ! <-- ! interior faces surface vectors                !
!  (ndim, nfac)    !    !     !                                               !
! surfbo           ! ra ! <-- ! boundary faces surface vectors                !
!  (ndim, nfabor)  !    !     !                                               !
! cdgfac           ! ra ! <-- ! interior faces centers of gravity             !
!  (ndim, nfac)    !    !     !                                               !
! cdgfbo           ! ra ! <-- ! boundary faces centers of gravity             !
!  (ndim, nfabor)  !    !     !                                               !
! xyznod           ! ra ! <-- ! vertex coordinates (optional)                 !
!  (ndim, nnod)    !    !     !                                               !
! volume(ncelet)   ! ra ! <-- ! cell volumes                                  !
! dt(ncelet)       ! ra ! <-- ! time step (per cell)                          !
! rtp, rtpa        ! ra ! <-- ! calculated variables at cell centers          !
!  (ncelet, *)     !    !     !  (at current and previous time steps)         !
! propce(ncelet, *)! ra ! <-- ! physical properties at cell centers           !
! propfa(nfac, *)  ! ra ! <-- ! physical properties at interior face centers  !
! propfb(nfabor, *)! ra ! <-- ! physical properties at boundary face centers  !
! coefa, coefb     ! ra ! <-- ! boundary conditions                           !
!  (nfabor, *)     !    !     !                                               !
! ettp, ettpa      ! ra ! <-- ! particle-defined variables                    !
!  (nbpmax, nvp)   !    !     !  (at current and previous time steps)         !
! tepa             ! ra ! <-- ! real particle properties                      !
!  (nbpmax, nvep)  !    !     !  (statistical weight, ...                     !
! statis           ! ra ! <-- ! statistic means                               !
!  (ncelet, nvlsta)!    !     !                                               !
! stativ(ncelet,   ! ra ! <-- ! accumulator for variance of volume statisitics !
!       nvlsta -1)!    !     !                                               !
! tslagr           ! ra ! <-- ! Lagrangian return coupling term               !
!  (ncelet, ntersl)!    !     !  on carrier phase                             !
! parbor           ! ra ! <-- ! particle interaction properties               !
!  (nfabor, nvisbr)!    !     !  on boundary faces                            !
! rdevel(nrdeve)   ! ra ! <-> ! real work array for temporary development      !
! rtuser(nrtuse)   ! ra ! <-- ! user-reserved real work array                  !
! ra(*)            ! ra ! --- ! main real work array                           !
!__________________!____!_____!________________________________________________!

!     Type: i (integer), r (real), s (string), a (array), l (logical),
!           and composite types (ex: ra real array)
!     mode: <-- input, --> output, <-> modifies data, --- work array
!========================================

use connectivity

implicit none

!========================================
! Common blocks
!========================================

include "dimfbr.h"
include "paramx.h"
include "pointe.h"
include "numvar.h"
include "optcal.h"
include "cstphy.h"
include "cstnum.h"
include "entsor.h"
include "lagpar.h"
include "lagran.h"
include "parall.h"
include "period.h"
include "ppppar.h"
include "ppthch.h"
include "ppincl.h"

!========================================

! Arguments

integer          idbia0 , idbra0
integer          ndim   , ncelet , ncel   , nfac   , nfabor
integer          nfml   , nprfml
integer          nnod   , lndfac , lndfbr , ncelbr
integer          nvar   , nscal  , nphas
integer          nbpmax , nvp    , nvep   , nivep
integer          ntersl , nvlsta , nvisbr
integer          nideve , nrdeve , nituse , nrtuse

integer          ifacel(2,nfac) , ifabor(nfabor)
integer          ifmfbr(nfabor) , ifmcel(ncelet)
integer          iprfml(nfml,nprfml)
integer          maxelt, lstelt(maxelt)
integer          ipnfac(nfac+1), nodfac(lndfac)
integer          ipnfbr(nfabor+1), nodfbr(lndfbr)
integer          itepa(nbpmax,nivep)
integer          idevel(nideve), ituser(nituse)
integer          ia(*)

double precision xyzcen(ndim,ncelet)
double precision surfac(ndim,nfac), surfbo(ndim,nfabor)
double precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)
double precision xyznod(ndim,nnod), volume(ncelet)
double precision dt(ncelet), rtp(ncelet,*), rtpa(ncelet,*)
double precision propce(ncelet,*)
double precision propfa(nfac,*), propfb(ndimfb,*)
double precision coefa(ndimfb,*), coefb(ndimfb,*)
double precision ettp(nbpmax,nvp) , ettpa(nbpmax,nvp)
double precision tepa(nbpmax,nvep)
double precision statis(ncelet,nvlsta), stativ(ncelet,nvlsta-1)
double precision tslagr(ncelet,ntersl)
double precision parbor(nfabor,nvisbr)
double precision rdevel(nrdeve), rtuser(nrtuse), ra(*)

! Local variables


logical :: switch1,switch2
integer          idebia, idebra
integer          iel, iutile, iel1,iel2, i, j, ival,ival2,impout(6),ii
integer          ifac
integer(8)       con(ncel,6), ihuge
double precision a(3), b(3), c(3)

!####local variables for Level Set modelling
double precision  phi_nbck,phi_nbck2, phi_cenck, max_val,min_val,xpos1,xpos2,sec(3)
integer(8)        icen
integer           ifac2, n_of_f, jmax,jmin,isnbb,Number_Of_Faces,Number_Of_Faces2
logical           ifind, iswitch1,iswitch2
integer           nd_ix, nd_i, nd_k, nx, ni, nk

!================[Redistancing code variables]=================

integer          ival_nb,ival_nbNeg, ielt, nlelt2, k, xi,icount, N_i,MAXIT
double precision sign_val, dist_ptR(3),dif_pr(3),dif_xr(3),t_val,dif(3),tdiv,k_hat(3),scale_diff,scal
double precision v_hat(3), n_div, n_hat(3), dist1, dist2, dist3, dI, dImin, x(3), po(3), small_v, r(3
double precision S_h,s_val, Sk, phi,phi_starr,x_n(3),po_n(3), delta_k, eta_k,eta_sum,xi_sum,fMin,t_sm
double precision fl,fh,f,c_val,c_l,c_h,c_1,c_2,dc,swap,del,c_acc, s_k,C_const,ix_val
double precision dist_prR(3),dval,phi3,rdist
double precision value, psi_v
integer          ir,irV(4)
logical          Lr(4),error_iteration,ipos_checkdo
integer          I_it,ilim,icen2,icen3,n_faces

double precision  phi_val(9),rcen(3),Grad_phi(3),phi_max,phi_min,psi_value,del_A
integer           i_vertex
!================[Free surface modelling code variables]=================
integer          ionbb, i_group,iorder,n_group, n_stencil,id, itype, n_col,nbox,ntri
integer          ieln, ErrorFlag
integer          ichange,icheck,iprim_neg,isec_pos,isec_neg,iprim_pos
integer          ineg,inod2,iph_XI,inext2,ipos,prim_pos,ir0,ir1,ir2,isim_count,isimplxMax
integer          itype_max,isimpx,isampleNeg,Nneg,Npos,Nmax, iside
double precision phi_1, small_diff,t_s2_left(3),t_s2_right(3),p_j(3),theta_a,theta_b,theta_c
double precision n_hat1(3),n_hat2(3),n_hat3(3), a_right, a_left,t_o2_left(3), a_tswf_f2, theta_f2
double precision theta_s2, theta_o2,theta_1,theta_2,theta_3, a_tswf_s2, a_tswf_o2,lc_tswf_o2,lc_tswf_s2
double precision tswf_s2_1(3),tswf_s2_2(3),tswf_o2_1(3),tswf_o2_2(3),t_o2_right(3)
double precision tpj0(3),tpj1(3),tpj2(3),lb_value,lc_value, parallel_chk,tdiv2,tdiv1,tdiv0,lc_tswf_f2
double precision t_21(3),t_23(3),t_32(3),tswf_f2_1(3),tswf_f2_2(3),t_f2_right(3),t_f2_left(3), xval
double precision diff1(3), diff2(3), diff3(3),px0(3), propU, propV, propW, Uvel(3), propP
double precision prop_u(3),prop_v(3),prop_w(3),prop_val(3),xyz(3),vol1,vol2,kv(3),s1(3),s2(3)
double precision rlook1(3),rlook2(3), xvalue, yvalue, density,Area_k,deltaV_k,f2,v(3),sum_scale

real                                :: chk_x(100)
real, dimension(:,:), allocatable :: data_array
real, dimension(:,:), allocatable :: chk_pts
real, allocatable                 :: px(:)
real, dimension(:,:), allocatable :: a_matrx
logical          projection, ileft,ifond2,ilog,chk,ifond,ichk,idebug1,idebug2
double precision  h_s, xrtp,xrtp2,tide_start
double precision, parameter         :: small_value = 1.0d-6
type(simplex), pointer              :: current,previous

!========================================

print*,'start usproj'

!========================================
! 1.  Initialization
!========================================

! Memory management
idebia = idbia0
idebra = idbra0
ihuge  = 2.0e10
fMin  = 1.0d-6
c_acc  = 1.0d-6
MAXIT  = 20

if (isuite.eq.0) then !******isuite if block
    !===============================
    do ii = 1, 1

        impout(ii) = impusr(ii)

    enddo
    open(impout(1),file='Tide_level_results.dat')
    ! print*,'nwb_counter = ',nwb_counter

    !=================================
    !==================================narrow band filter scheme====================
    !==================================scalar one TEST==============================

    !-----------------------------------------------------------------------------
    !         NEW RE-DISTANCING ROUTINE (START)
    !-----------------------------------------------------------------------------
    !=======================================
    ! (E) Initial set up of Level set first and second neighbour cells surrounding isocontour S_h
    !    -------------------------------------------------
    !=======================================
    !===========Setup the first neighbour cells surrounding S_h
    if ( (allocated(ptr_nwbElm)).and.(allocated(ptr_NegnwbElm)) ) then
        !-----------------------------------------------------------------------------------------
        !        clear dynamic simplex memory at the end of run
        !-----------------------------------------------------------------------------------------
        !-------------positive side memory purge--------------
        do iel = 1, nwb_counter
            do itype = 1, itype_max
                select case (itype)
                case (1)
                    isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                    current => ptr_nwbElm(iel)%ptr_cutTetra_type1
                case (2)
                    isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
                    current => ptr_nwbElm(iel)%ptr_cutTetra_type2
                case (3)
                    isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                    current => ptr_nwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    do while ( associated( current ) )
                        previous => current
                        current => current%next
                        deallocate( previous )
                    end do
                enddo   ! loop of isimplx
            enddo      ! loop of itype
        enddo ! loop of iel
        !-------------negative side memory purge--------------
        do iel = 1, nwb_cnter_Neg
            do itype = 1, itype_max
                select case (itype)
                case (1)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
                case (2)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
                case (3)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    do while ( associated( current ) )
                        previous => current
                        current => current%next
                        deallocate( previous )
                    end do
                enddo   ! loop of isimplx
            enddo      ! loop of itype
        enddo ! loop of iel
        !----------------------------------------------------------------------------------------------
        !           clear dynamic simplex memory at the end of run (completed)
        !----------------------------------------------------------------------------------------------
        !===========Setup the first neighbour cells surrounding S_h
        !-------------positive side sweep--------------
        nwb_counter = 0
        do iel = 1,ncel
            phi_cenck       = rtp(iel,isca(2))                                 ! positive kth node
            Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
            ! ++++++++looking for field region where iso-surface S_h exists
            iprim_neg = 0
            iprim_pos = 0
            do ifac = 1,Number_of_Faces
                phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
                if ((phi_nbck.lt.0.0d0).and.(phi_cenck.gt.0.0d0)) then
                    !-------check negative side---------
                    if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                        iprim_neg = iprim_neg + 1
                    endif
                endif
            enddo
            if (iprim_neg > 0) then
                do ifac2 = 1, Number_of_Faces
                    phi_nbck2  = rtp(nbcell(iel,ifac2),isca(2))
                    if ( phi_nbck2.gt.0.0d0) then
                        if (secondary_cell_select(nbcell(iel,ifac2)) <= 0) then
                            iprim_pos = iprim_pos + 1
                        endif
                    endif
                enddo
            endif
            !-----------------------------------------------------------------------------------------------
            !     Create a new set of temporary simplex tetrahedra surrounding a piece of the isosurface
            !-----------------------------------------------------------------------------------------------
            if ((iprim_neg.gt.0).and.(iprim_neg.lt.4).and.(phi_cenck.gt.0)) then
                nwb_counter   = nwb_counter + 1
                if (nwb_counter.gt.ihuge) then
                    deallocate(ptr_nwbElm)
                    stop 0
                endif
                ptr_nwbElm(nwb_counter)%nwb_index = iel
                !**********************
```

```
            iprim_neg = 0
            iprim_pos = 0
            isec_pos  = 0
            Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
            do ifac = 1, Number_of_Faces
                phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
                if ((phi_nbck.lt.0.0d0).and.(phi_cenck.gt.0.0d0)) then
                    rtp(iel,isca(1)) = 40.0                                ! prim scalar1
                    if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                        iprim_neg = iprim_neg + 1
                        ptr_nwbElm(nwb_counter)%Pneg(iprim_neg) = nbcell(iel,ifac)   ! negative p
                    endif
                else if ( (phi_nbck.gt.0.0d0).and.(phi_cenck.gt.0.0d0) ) then
                    if (secondary_cell_select(nbcell(iel,ifac)) == 1) then
                        isec_pos = isec_pos + 1
                        ptr_nwbElm(nwb_counter)%Spos(isec_pos) = nbcell(iel,ifac)   ! positive se
                        rtp(nbcell(iel,ifac),isca(1)) = 80.0            ! sec scalar1 +ve
                    else
                        iprim_pos = iprim_pos + 1
                        ptr_nwbElm(nwb_counter)%Ppos(iprim_pos) = nbcell(iel,ifac)   ! positive p
                        rtp(nbcell(iel,ifac),isca(1)) = 40.0        !  prim scalar1 +ve
                    endif
                endif
            enddo
            !************************NOTE NO NEGATIVE SECONDARY*********************
            ptr_nwbElm(nwb_counter)%neg      =  iprim_neg  ! total number of negative primary no
            ptr_nwbElm(nwb_counter)%pos      =  iprim_pos  ! total number of positive primary no
            ptr_nwbElm(nwb_counter)%Sec_pos  =  isec_pos   ! total number of positive secondary
        endif
        !------------------------------------------------------------------------------------

    enddo

    !-------------negative side sweep--------------
    nwb_cnter_Neg = 0
    do iel = 1,ncel
        phi_cenck      = rtp(iel,isca(2))                              ! negative kth node
        Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
        ! ++++++++looking for field region where iso-surface S_h exists
        iprim_neg = 0
        iprim_pos = 0
        do ifac = 1,Number_of_Faces
            phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
            if ((phi_nbck.gt.0.0d0).and.(phi_cenck.lt.0.0d0)) then         !!!NOTE NOW phi_cenck
                !-------check positive side---------
                if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                    iprim_pos = iprim_pos + 1
                endif
            endif
        enddo
        if (iprim_pos > 0) then
            do ifac2 = 1, Number_of_Faces
                phi_nbck2  = rtp(nbcell(iel,ifac2),isca(2))
                if ( phi_nbck2.lt.0.0d0) then
                    if (secondary_cell_select(nbcell(iel,ifac2)) <= 0) then
                        iprim_neg = iprim_neg + 1
                    endif
                endif
            enddo
        endif
        !-----------------------------------------------------------------------------------------------
        !     Create a new set of temporary simplex tetrahedra surrounding a piece of the isosurface
        !-----------------------------------------------------------------------------------------------
        if ((iprim_pos.gt.0).and.(iprim_pos.lt.4).and.(phi_cenck.lt.0.0d0)) then
            nwb_cnter_Neg   = nwb_cnter_Neg + 1
            if (nwb_cnter_Neg.gt.ihuge) then
                deallocate(ptr_NegnwbElm)
                stop 0
            endif
            ptr_NegnwbElm(nwb_cnter_Neg)%nwb_index = iel
            !**********************
            iprim_neg = 0
            iprim_pos = 0
            isec_neg  = 0
            Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
            do ifac = 1, Number_of_Faces
                phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
                if ((phi_nbck.gt.0.0d0).and.(phi_cenck.lt.0.0d0)) then         !!!NOTE NOW phi_cenck neg
                    rtp(iel,isca(1)) = -40.0                                       ! prim scalar1 -ve
                    if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                        iprim_pos = iprim_pos + 1
                        ptr_NegnwbElm(nwb_cnter_Neg)%Ppos(iprim_pos) = nbcell(iel,ifac)   ! positive prim
                    endif
                else if ( (phi_nbck.lt.0.0d0).and.(phi_cenck.lt.0.0d0) ) then
                    if (secondary_cell_select(nbcell(iel,ifac)) == 1) then
                        isec_neg = isec_neg + 1
                        ptr_NegnwbElm(nwb_cnter_Neg)%Sneg(isec_neg) = nbcell(iel,ifac)   ! negative secon
                        rtp(nbcell(iel,ifac),isca(1)) = -80.0              ! sec scalar1 -ve
                    else
                        iprim_neg = iprim_neg + 1
                        ptr_NegnwbElm(nwb_cnter_Neg)%Pneg(iprim_neg) = nbcell(iel,ifac)   ! negative prim
                        rtp(nbcell(iel,ifac),isca(1)) = -40.0         !  prim scalar1 -ve
                    endif
                endif
            enddo
            !*** NOTE NO POSITIVE SECONDARY ***
            ptr_NegnwbElm(nwb_cnter_Neg)%neg       =  iprim_neg  ! total number of negative primary nodes
            ptr_NegnwbElm(nwb_cnter_Neg)%pos       =  iprim_pos  ! total number of positive primary nodes
            ptr_NegnwbElm(nwb_cnter_Neg)%Sec_neg   =  isec_neg   ! total number of negative secondary noo
        endif
        !-----------------------------------------------------------------------------------------------

    enddo !end of iel loop
    !-----------end of negative side sweep

    !======================================
    !     Initial redistancing (START)
    !====================================
    !**********************************
    !           Step 1:: re-Compute the exact distance to S_h
    !**********************************
    !-------------------------------------------------
    !---------positive side of free surface-----------
    !-------------------------------------------------
    iside = 0
    do iel = 1, nwb_counter
        ival_nb  = ptr_nwbElm(iel)%nwb_index
        Nneg = ptr_nwbElm(iel)%neg
        Npos = ptr_nwbElm(iel)%pos
        if (Nneg > 3) then
            itype_max = 3
        else
            itype_max = Nneg
        endif
        call create_simplices( iside, ptr_nwbElm(iel), Nneg, Npos,itype_max)
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type1
                Nmax = 3
            case (2)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type2
                Nmax = 4
            case (3)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type3
                Nmax = 3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select

            do isimpx = 1, isimplxMax

                do inod2 = 1, Nmax
                    if (itype == 1) then
                        if ( inod2 == 1) then
                            ir   =  ival_nb                ! Kth node
                        else
                            ir  =  current%sim_node(inod2)
                        endif
                        ieln =  current%sim_node(1)
                    else if (itype == 2) then
                        select case (inod2)
                        case (1,2)
                            ieln  =  current%sim_node(inod2)
                            ir   =  ival_nb
                        case (3,4)
                            ival = inod2 - 2
                            ieln  =  current%sim_node(ival)
                            ir    =  current%sim_node(3)
                        case default
                            call error_message('Error2 in step1 with isimplxMax in USINV')
                        end select
                    else if (itype == 3) then
                        ieln  =  current%sim_node(inod2)
                        ir    =  ival_nb
                    else
                        call error_message('Error3 in step1 with isimplxMax in USINV')
                    endif
                    s_val   = rtp(ir,isca(2)) -  rtp(ieln,isca(2))
                    !******** check if s_val is zero******
                    call check_zero(s_val,'Error4 in USINV with Sk reconstruction at iel = ',iel)
                    S_h  = - rtp(ieln,isca(2))/s_val
                    if (S_h > 0.98) current%use_it = .false.                                    !new
                    a(1) = scale_length * (xyzcen(1,ir) )
                    a(2) = scale_length * (xyzcen(2,ir) )
                    a(3) = scale_length * (xyzcen(3,ir) )
                    b(1) = scale_length * (xyzcen(1,ieln) )
                    b(2) = scale_length * (xyzcen(2,ieln) )
                    b(3) = scale_length * (xyzcen(3,ieln) )
                    call position_vec(a, b, S_h, c)
                    current%sh_nb(inod2,1) = c(1)
                    current%sh_nb(inod2,2) = c(2)
                    current%sh_nb(inod2,3) = c(3)
                enddo ! inod2 loop
                current => current%next
            enddo ! loop isimpx
        enddo ! loop itype
    enddo ! iel loop


    ! ***check for zero volume simplices == (start)  ***
    iside = 0
    do iel = 1, nwb_counter
        ival_nb    =   ptr_nwbElm(iel)%nwb_index
        phi        =   1.0
        Npos = ptr_nwbElm(iel)%pos
        Nneg = ptr_nwbElm(iel)%neg
        if (Nneg > 3) then
            itype_max = 3
        else
            itype_max = Nneg
        endif
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type1
            case (2)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
```

```
                current => ptr_nwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select
            !  *** scan for isimplxMax per simplex cut type considered  ***
            do isimpx = 1, isimplxMax
                if (current%use_it) then                              !new
                    call Vol_k(.true. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                    if (vol2 < tiny(1.0)) then
                        !print*,iel,'positive vol_k is = ',vol2,isimplxMax
                        current%use_it = .false.
                    endif
                endif
                current => current%next
            enddo ! isimpx loop
            ! *** *** *** *** *** *** *** ***
        enddo    ! itype loop
    enddo  ! iel loop

    ! ***check for zero volume simplices == (end)  ***


    ! Compute dI such that dI = min x belongs S_k|XI-x|===(start)
    do iel = 1, nwb_counter
        ival_nb   = ptr_nwbElm(iel)%nwb_index
        Nneg = ptr_nwbElm(iel)%neg
        if (Nneg > 3) then
            itype_max = 3
        else
            itype_max = Nneg
        endif
        dImin = huge(1.0)     !***set d(Xn) = +infinity for n = 1,2,...Nnod_P
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type1
            case (2)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select
            !  *** scan for isimplxMax per simplex cut type considered  ***
            do isimpx = 1, isimplxMax
                ir0   =  1        ! Kth node
                ir1   =  2
                ir2   =  3
                a(1) = scale_length * (xyzcen(1,ival_nb) )        ! k node
                a(2) = scale_length * (xyzcen(2,ival_nb) )
                a(3) = scale_length * (xyzcen(3,ival_nb) )
                kv(1) = current%sh_nb(ir0,1)      ! edge of free surface near kth node
                kv(2) = current%sh_nb(ir0,2)
                kv(3) = current%sh_nb(ir0,3)
                dif(1) = kv(1) - a(1)
                dif(2) = kv(2) - a(2)
                dif(3) = kv(3) - a(3)
                s1(1) = current%sh_nb(ir1,1) - kv(1)
                s1(2) = current%sh_nb(ir1,2) - kv(2)
                s1(3) = current%sh_nb(ir1,3) - kv(3)
                s2(1) = current%sh_nb(ir2,1) - kv(1)
                s2(2) = current%sh_nb(ir2,2) - kv(2)
                s2(3) = current%sh_nb(ir2,3) - kv(3)
                call cross_product(s1,s2,v)
                tdiv     = dabs(dot_product(v,v))
                tdiv1 =  dsqrt(tdiv)
                !if (current%use_it) then
                if (tdiv < small_value ) then
                    !dist1 = rtp(ival_nb,isca(2))
                    !if ( dabs(dImin) > dabs(dist1)) then
                    !   dImin = dabs(dist1)
                    !endif
                    current => current%next
                    cycle
                endif
                !endif
                v_hat(1) = v(1)/tdiv1
                v_hat(2) = v(2)/tdiv1
                v_hat(3) = v(3)/tdiv1
                dist1 = dabs(dot_product(dif, v_hat))
                if (dabs(dImin) > dabs(dist1)) then
                    dImin = dabs(dist1)
                endif
                current => current%next
            enddo ! loop isimplx
            ! *** *** *** *** *** *** *** ***&
        enddo ! loop itype
        ptr_nwbElm(iel)%ph_star = dImin  !^^^^^^set phi*(Xn) = d(Xn) for n = 1,2,...Nnod_P ^^^^^^
    enddo ! iel loop

    !--------------------------------------------------
    !---------negative side of free surface-----------
    !--------------------------------------------------
    iside = 1
    do iel = 1, nwb_cnter_Neg
        ival_nbNeg  = ptr_NegnwbElm(iel)%nwb_index
        Nneg = ptr_NegnwbElm(iel)%neg
        Npos = ptr_NegnwbElm(iel)%pos
        if (Npos > 3) then
            itype_max = 3
```

```
else
    itype_max = Npos
endif
call create_simplices( iside, ptr_NegnwbElm(iel), Npos, Nneg,itype_max)
do itype = 1, itype_max
    select case (itype)
    case (1)
        isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
        current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        Nmax = 3
    case (2)
        isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
        current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        Nmax = 4
    case (3)
        isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
        current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        Nmax = 3
    case default
        call error_message('Error1 in step1 with isimplxMax in USINV')
    end select

    do isimpx = 1, isimplxMax

        do inod2 = 1, Nmax
            if (itype == 1) then
                if ( inod2 == 1) then
                    ir   =  ival_nbNeg                ! Kth node
                else
                    ir  =  current%sim_node(inod2)
                endif
                ieln =  current%sim_node(1)
            else if (itype == 2) then
                select case (inod2)
                case (1,2)
                    ieln  =  current%sim_node(inod2)
                    ir    =  ival_nbNeg
                case (3,4)
                    ival = inod2 - 2
                    ieln  =  current%sim_node(ival)
                    ir    =  current%sim_node(3)
                case default
                    call error_message('Error2 in step1 with isimplxMax in USINV')
                end select
            else if (itype == 3) then
                ieln  =  current%sim_node(inod2)
                ir    =  ival_nbNeg
            else
                call error_message('Error3 in step1 with isimplxMax in USINV')
            endif
            s_val   = rtp(ir,isca(2)) -  rtp(ieln,isca(2))
            !******** check if s_val is zero******
            call check_zero(s_val,'Error4 in USINV with Sk reconstruction at iel = ',iel)
```

```
                S_h  = - rtp(ieln,isca(2))/s_val
                if (S_h > 0.98) current%use_it = .false.                                       !new
                a(1) = scale_length * (xyzcen(1,ir) )
                a(2) = scale_length * (xyzcen(2,ir) )
                a(3) = scale_length * (xyzcen(3,ir) )
                b(1) = scale_length * (xyzcen(1,ieln) )
                b(2) = scale_length * (xyzcen(2,ieln) )
                b(3) = scale_length * (xyzcen(3,ieln) )
                call position_vec(a, b, S_h, c)
                current%sh_nb(inod2,1) = c(1)
                current%sh_nb(inod2,2) = c(2)
                current%sh_nb(inod2,3) = c(3)
            enddo ! inod2 loop
            current => current%next
        enddo ! loop isimpx
    enddo ! loop itype
enddo ! iel loop


! ***check for zero volume simplices == (start)  ***
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    phi           =   1.0
    Npos = ptr_NegnwbElm(iel)%pos
    Nneg = ptr_NegnwbElm(iel)%neg
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                              !new
                call Vol_k(.true. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                if (vol2 < tiny(1.0)) then
                    !print*,iel,'Negative vol_k is = ',vol2,isimplxMax
                    current%use_it = .false.
                endif
            endif
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop
! ***check for zero volume simplices == (end)  ***^^^^^


! Compute dI such that dI = min x belongs S_k|XI-x|===(start)
do iel = 1, nwb_cnter_Neg
    ival_nbNeg   = ptr_NegnwbElm(iel)%nwb_index
    Npos = ptr_NegnwbElm(iel)%pos
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    dImin = huge(1.0)     !***set d(Xn) = +infinity for n = 1,2,...Nnod_P
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            ir0   =  1        ! Kth node
            ir1   =  2
            ir2   =  3
            a(1) = scale_length * (xyzcen(1,ival_nbNeg) )        ! k node
            a(2) = scale_length * (xyzcen(2,ival_nbNeg) )
            a(3) = scale_length * (xyzcen(3,ival_nbNeg) )
            kv(1) = current%sh_nb(ir0,1)      ! edge of free surface near kth node
            kv(2) = current%sh_nb(ir0,2)
            kv(3) = current%sh_nb(ir0,3)
            dif(1) = kv(1) - a(1)
            dif(2) = kv(2) - a(2)
            dif(3) = kv(3) - a(3)
            s1(1) = current%sh_nb(ir1,1) - kv(1)
            s1(2) = current%sh_nb(ir1,2) - kv(2)
            s1(3) = current%sh_nb(ir1,3) - kv(3)
            s2(1) = current%sh_nb(ir2,1) - kv(1)
            s2(2) = current%sh_nb(ir2,2) - kv(2)
            s2(3) = current%sh_nb(ir2,3) - kv(3)
            call cross_product(s1,s2,v)
            tdiv     = dabs(dot_product(v,v))
            tdiv1 =  dsqrt(tdiv)
            !if (current%use_it) then
            if (tdiv < small_value ) then
                !dist1 = - rtp(ival_nbNeg,isca(2))
                !if ((dabs(dImin) > dabs(dist1))) then
                !   dImin = dabs(dist1)
                !endif
                current => current%next
                cycle
            endif
            !endif
            v_hat(1) = v(1)/tdiv1
            v_hat(2) = v(2)/tdiv1
            v_hat(3) = v(3)/tdiv1
            dist1 = dabs(dot_product(dif, v_hat))
            if (dabs(dImin) > dabs(dist1)) then
                dImin = dabs(dist1)
            endif
            current => current%next
        enddo ! loop isimplx
        ! *** *** *** *** *** *** *** ***&
    enddo ! loop itype
    ptr_NegnwbElm(iel)%ph_star = - dImin  !^^^^^^set phi*(Xn) = d(Xn) for n = 1,2,...Nnod_P ^^^^^^
enddo ! iel loop

! Compute dI such that dI = min x belongs S_k|XI-x|====(end)
!***********************************
!           Step 2:: Find eta_h, a piecewise constant function
!***********************************
! Find eta_h, a piecewise constant function (simplex wise mass correction) == (start)
!---------positive side of free surface-----------
iside = 0
do iel = 1, nwb_counter
    ival_nb    =   ptr_nwbElm(iel)%nwb_index
    phi        =   rtp(ival_nb,isca(2))
    phi_starr  =   ptr_nwbElm(iel)%ph_star
    eta_k      =   phi - phi_starr
    ! if ( ntcabs == 17) then
    !   print*,phi,phi_starr
    ! endif
    Npos = ptr_nwbElm(iel)%pos
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax simplices per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            call Vol_k(.false. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
            current%vol_phi = vol2
            call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,eta_k,Area_K,vol1)
            deltaV_k = vol2 - vol1
            icount   = 0
            do
                if ((dabs(deltaV_k) < 1.0d-8).or.(icount > 20)) exit
                icount = icount + 1
                if ( Area_K < tiny(1.0) ) then
                    exit
                endif
                eta_k =  -3.0 * deltaV_k/Area_K    ! Changed
                call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,eta_k,Area_K,vol1)
                deltaV_k = vol2 - vol1
            enddo
            if (current%use_it) then                              !new
                current%eta = eta_k
            else
                current%eta = 0.0
            endif                                                 !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop

!---------negative side of free surface-----------
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    phi         =   rtp(ival_nbNeg,isca(2))
    phi_starr  =   ptr_NegnwbElm(iel)%ph_star
    eta_k      =   phi - phi_starr
    Npos = ptr_NegnwbElm(iel)%pos
    Nneg = ptr_NegnwbElm(iel)%neg
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    do itype = 1, itype_max
```

```
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi,0.0d0,Area_K,vol2)
            current%vol_phi = vol2
            call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi_starr,eta_k,Area_K,vol1)
            deltaV_k = vol2 - vol1
            icount   = 0
            do
                if ((dabs(deltaV_k) < 1.0d-8).or.(icount > 20)) exit
                icount = icount + 1
                if ( Area_K < tiny(1.0) ) then
                    exit
                endif
                eta_k =  -3.0 * deltaV_k/Area_K    ! Changed
                call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi_starr,eta_k,Area_K,vol1)
                deltaV_k = vol2 - vol1
            enddo
            if (current%use_it) then                              !new
                current%eta = eta_k
            else
                current%eta = 0.0
            endif                                                !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop

! Find eta_h, a piecewise constant function (simplex wise mass correction) == (end)
!**********************************
!           Step 3:: Find Xi_h, the ortogonal projection of eta_h
!**********************************
! Find Xi_h (node wise mass correction) == (start)
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    xi_sum = 0.0
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    isim_count = 0
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                              !new
                isim_count = isim_count  + 1
                xi_sum = xi_sum + current%eta
            endif                                                !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
    if (isim_count == 0) then
        isim_count = 1
        !print*,iel,' pos xi_sum = ', xi_sum
    endif
    ptr_nwbElm(iel)%xi_h = xi_sum/real(isim_count)
enddo

!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    xi_sum = 0.0
    Npos = ptr_NegnwbElm(iel)%pos
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    isim_count = 0
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                              !new
                isim_count = isim_count  + 1
                xi_sum = xi_sum + current%eta
            endif                                                 !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
    if (isim_count == 0) then
        isim_count = 1
        ! print*,iel,' neg xi_sum = ', xi_sum
    endif
    ptr_NegnwbElm(iel)%xi_h = xi_sum/real(isim_count)
enddo
! Find Xi_h (node wise mass correction) == (end)
!***********************************
!           Step 4(i):: Find psi_h = C xi_h
!***********************************
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    error_iteration = .true.
    c_1     =  0.0
    c_2     =  scale_length !1.0
    fl      =  deltaV(iside,ptr_nwbElm(iel),c_1)
    fh      =  deltaV(iside,ptr_nwbElm(iel),c_2)
    if ( (fh * fl) > 0.0) then
        c_2     =  -scale_length !1.0
        fh      =  deltaV(iside,ptr_nwbElm(iel),c_2)
    endif
    if ( ( (fl * fh) > 0.0).and.(dabs(fl) > fMin).and.(dabs(fh) > fMin)) then
        print*,'Error at iel = ',iel, 'root must be bracketed between arguments'
        stop
    else if ((dabs(fl) < fMin).and.(dabs(fh) < fMin)) then
        error_iteration = .false.
        ptr_nwbElm(iel)%cval = 0.0
        continue
    endif
    if ( fl  > 0.0 ) then
        c_l = c_1
        c_h = c_2
    else
        c_l = c_2
        c_h = c_1
```

```
            swap = fl
            fl = fh
            fh = swap
        endif
        dc = c_h - c_l
        do j = 1, MAXIT
            if (dabs(fl - fh) < tiny(1.0)) then
                error_iteration = .false.
                exit
            endif
            c_val = c_l + dc * fl/(fl - fh)
            f = deltaV(iside,ptr_nwbElm(iel),c_val)
            if (f < 0.0) then
                del = c_l - c_val
                c_l = c_val
                fl = f
            else
                del = c_h - c_val
                c_h = c_val
                fh  = f
            endif
            dc  = c_h - c_l
            ptr_nwbElm(iel)%fval = f
            ptr_nwbElm(iel)%cval = c_val
            if ( (dabs(del) < c_acc).or.(dabs(f) < tiny(1.0)) ) then
                error_iteration = .false.
                exit
            endif
        enddo ! j loop
        if (error_iteration) then
            print*,'Maximum number of iteration exceeded at iel = ', iel
            stop
        endif
    enddo  ! iel loop
    !---------negative side of free surface-----------
    iside = 1
    do iel = 1,nwb_cnter_Neg
        error_iteration = .true.
        c_1     =  0.0
        c_2     =  scale_length !1.0
        fl      =  deltaV(iside,ptr_NegnwbElm(iel),c_1)
        fh      =  deltaV(iside,ptr_NegnwbElm(iel),c_2)
        if ( (fh * fl) > 0.0) then
            c_2     =  -scale_length !1.0
            fh      =  deltaV(iside,ptr_NegnwbElm(iel),c_2)
        endif
        if ( ( (fl * fh) > 0.0).and.(dabs(fl) > fMin).and.(dabs(fh) > fMin)) then
            print*,'Error at iel = ',iel, 'root must be bracketed between arguments'
            stop
        else if ((dabs(fl) < fMin).and.(dabs(fh) < fMin)) then
            error_iteration = .false.
            ptr_NegnwbElm(iel)%cval = 0.0
            continue
        endif
        if ( fl  > 0.0 ) then
            c_l = c_1
            c_h = c_2
        else
            c_l = c_2
            c_h = c_1
            swap = fl
            fl = fh
            fh = swap
        endif
        dc = c_h - c_l
        do j = 1, MAXIT
            if (dabs(fl - fh) < tiny(1.0)) then
                error_iteration = .false.
                exit
            endif
            c_val = c_l + dc * fl/(fl - fh)
            f = deltaV(iside,ptr_NegnwbElm(iel),c_val)
            if (f < 0.0) then
                del = c_l - c_val
                c_l = c_val
                fl = f
            else
                del = c_h - c_val
                c_h = c_val
                fh  = f
            endif
            dc  = c_h - c_l
            ptr_NegnwbElm(iel)%fval = f
            ptr_NegnwbElm(iel)%cval = c_val
            if ( (dabs(del) < c_acc).or.(dabs(f) < tiny(1.0)) ) then
                error_iteration = .false.
                exit
            endif
        enddo ! j loop
        if (error_iteration) then
            print*,'Maximum number of iteration exceeded at iel = ', iel
            stop
        endif
    enddo  ! iel loop
    ! Find C  == (start)
    !***********************************
    !           Step 4(ii):: Redistance the Level set first cells surrounding isocontour S_h
    !***********************************
    !---------positive side of free surface-----------
    iside = 0
    do iel = 1,nwb_counter
        ival_nb    =   ptr_nwbElm(iel)%nwb_index
        ix_val            = ptr_nwbElm(iel)%xi_h
        C_const           = ptr_nwbElm(iel)%cval
        phi_starr         = ptr_nwbElm(iel)%ph_star
        !if ( ntcabs == 19) then
        ! print*,iel,rtp(ival_nb,isca(2)),phi_starr,C_const,ix_val
        !endif
        rtp(ival_nb,isca(2)) = phi_starr + ( C_const * ix_val )  !NOTE MOST IMPORTANT PART which
    enddo  ! iel loop
    !---------negative side of free surface-----------
    iside = 1
    do iel = 1,nwb_cnter_Neg
        ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
        ix_val            = ptr_NegnwbElm(iel)%xi_h
        C_const           = ptr_NegnwbElm(iel)%cval
        phi_starr         = ptr_NegnwbElm(iel)%ph_star
        !print*,iel,rtp(ival_nbNeg,isca(2)),phi_starr,C_const,ix_val
        rtp(ival_nbNeg,isca(2)) = phi_starr + ( C_const * ix_val )  !NOTE MOST IMPORTANT PART whi
    enddo  ! iel loop
    !***********************************
    !           Step 5:: Edge distance approximation
    !***********************************
    ! -----------update available secondary nodes on positive side of isosurface--------
    iside = 0
    ichange = 1
    do
        if (ichange == 0) exit
        ichange = 0
        do iel = 1, nwb_counter
            Npos = ptr_nwbElm(iel)%pos
            isnbb = ptr_nwbElm(iel)%Sec_pos
            do i = 1, isnbb
                dImin    = huge(1)
                iph_XI = ptr_nwbElm(iel)%Spos(i)
                a(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
                a(2)   = scale_length * (xyzcen(2,iph_XI))
                a(3)   = scale_length * (xyzcen(3,iph_XI))
                ! |XJ - XI| edge distance approx considered of Npos of them
                do ipos = 1,Npos
                    icen = ptr_nwbElm(iel)%Ppos(ipos)
                    b(1)   = scale_length * (xyzcen(1,icen))  - a(1)
                    b(2)   = scale_length * (xyzcen(2,icen))  - a(2)
                    b(3)   = scale_length * (xyzcen(3,icen))  - a(3)
                    dI     = rtp(icen,isca(2)) + dsqrt( dabs(dot_product(b,b)) )
                    if ( dabs(dImin) > dabs(dI)) then
                        dImin = dI
                    endif
                enddo !=======(Find minimum phi_h(X_I) ....end)
                if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                    rtp(iph_XI,isca(2)) = dImin  !Edge distance approximation of phi at XI second
                    ichange = 1
                endif
            enddo
        enddo
    enddo

    ! -----------update available secondary nodes on negative side of isosurface--------
    iside = 1
```

```
ichange = 1
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_cnter_Neg
        Nneg = ptr_NegnwbElm(iel)%neg
        isnbb = ptr_NegnwbElm(iel)%Sec_neg
        do i = 1, isnbb
            dImin    = huge(1.0)
            iph_XI = ptr_NegnwbElm(iel)%Sneg(i)
            a(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
            a(2)   = scale_length * (xyzcen(2,iph_XI))
            a(3)   = scale_length * (xyzcen(3,iph_XI))
            ! |XJ - XI| edge distance approx considered of Npos of them
            do ineg = 1,Nneg
                icen = ptr_NegnwbElm(iel)%Pneg(ineg)
                b(1)   = scale_length * (xyzcen(1,icen))  - a(1)
                b(2)   = scale_length * (xyzcen(2,icen))  - a(2)
                b(3)   = scale_length * (xyzcen(3,icen))  - a(3)
                dI     = rtp(icen,isca(2)) - dsqrt( dabs(dot_product(b,b)) )
                if ( dabs(dImin) > dabs(dI)) then
                    dImin = dI
                endif
            enddo !=======(Find minimum phi_h(X_I) ....end)
            if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                rtp(iph_XI,isca(2)) = -dImin  !Edge distance approximation of phi at XI secondary noo
                ichange = 1
            endif
        enddo
    enddo
enddo
!--------------------------------------------------------------------------------------
!***********************************
!           Step 6:: Shadow distance correction
!***********************************
! -----------update available secondary nodes on positive side of isosurface--------
iside = 1
ichange = 0
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_cnter_Neg
        Npos = ptr_NegnwbElm(iel)%pos
        ival_nbNeg    = ptr_NegnwbElm(iel)%nwb_index
        if (Npos > 3) then
            itype_max = 3
        else
            itype_max = Npos
        endif
        !  *** scan for isimplxMax simplices per simplex cut type considered
        do itype = 1, itype_max
            isnbb = ptr_NegnwbElm(iel)%Sec_neg
            do i = 1, isnbb
```

```
dImin    = huge(1)
iph_XI = ptr_NegnwbElm(iel)%Sneg(i)
sec(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
sec(2)   = scale_length * (xyzcen(2,iph_XI))
sec(3)   = scale_length * (xyzcen(3,iph_XI))
! scan positive faces of simplex
select case (itype)
case (1)
    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
case (2)
    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
case (3)
    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
case default
    call error_message('Error1 in step1 with isimplxMax in USINV')
end select
do isimpx = 1, isimplxMax
    if (itype == 1) then
        ! --------face 1
        ntri = 3
        Lr(1) = .true. ; irV(1)  = ival_nbNeg
        Lr(2) = .true. ; irV(2)  = current%sim_node(3)
        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
        ! ---------face 2
        ntri = 4
        Lr(1) = .true. ; irV(1)  = ival_nbNeg
        Lr(2) = .false. ; irV(2)  = 1
        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
        Lr(4) = .false. ; irV(4)  = 2
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
        ! --------face 3
        ntri = 4
        Lr(1) = .true. ; irV(1)  = current%sim_node(3)
        Lr(2) = .false. ; irV(2)  = 3
        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
        Lr(4) = .false. ; irV(4)  = 2
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
        ! --------face 4
        ntri = 4
        Lr(1) = .true.  ; irV(1)  = ival_nbNeg
        Lr(2) = .false. ; irV(2)  = 1
        Lr(3) = .true. ; irV(3)  = current%sim_node(3)
        Lr(4) = .false. ; irV(4)  = 3
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
    else if (itype == 2) then
        ! ---------face 1
        ntri = 4
        Lr(1) = .true. ; irV(1)  = ival_nbNeg
        Lr(2) = .true. ; irV(2)  = current%sim_node(3)
        Lr(3) = .false. ; irV(3)  = 2
        Lr(4) = .false. ; irV(4)  = 4
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
        ! ---------face 2
        ntri = 3
        Lr(1) = .true. ; irV(1)  = ival_nbNeg
        Lr(2) = .false. ; irV(2)  = 2
        Lr(3) = .false. ; irV(3)  = 1
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
        ! ---------face 3
        ntri = 3
        Lr(1) = .true. ; irV(1)  = current%sim_node(3)
        Lr(2) = .false. ; irV(2)  = 3
        Lr(3) = .false. ; irV(3)  = 4
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
        ! ---------face 4
        ntri = 4
        Lr(1) = .true.  ; irV(1)  = ival_nbNeg
        Lr(2) = .false. ; irV(2)  = 1
        Lr(3) = .true. ; irV(3)  = current%sim_node(3)
        Lr(4) = .false. ; irV(4)  = 3
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
        if ( dabs(dImin) > dabs(dI)) then
            dImin = dI
        endif
    else if (itype == 3) then
        ! --------face 1
        ntri = 3
        Lr(1) = .false. ; irV(1)  = 1
        Lr(2) = .true. ; irV(2)   = ival_nbNeg
        Lr(3) = .false. ; irV(3)  = 2
        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
```

```
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! ---------face 2
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 1
                        Lr(3) = .false. ; irV(3)  = 3
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! --------face 3
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 3
                        Lr(3) = .false. ; irV(3)  = 2
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                    endif
                    current => current%next
                enddo !---end of isimplx loop
                if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                    rtp(iph_XI,isca(2)) = -dImin  !shaddow distance approximation of phi at X
                    ichange = 1
                endif
            enddo !---end of isnbb secondary nodes loop
        enddo !---end of itype loop
    enddo !---end of nwb_cnter_Neg loop
enddo !---ichange  loop !=======(Find minimum phi_h(X_I) ....end)
!--------------------------------------------------------------------------------------

!=======================================
!      redistancing (END)
!=======================================

if ( ntcabs.ge.ntmabs) then
    !------- clear dynamic memory at the end of run----------------------------------
    !---------positive side of free surface-----------
    do iel = 1, nwb_counter
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type1
            case (2)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    do while ( associated( current ) )
                        previous => current
                        current => current%next
                        deallocate( previous )
                    end do
                enddo   ! loop of isimplx
            enddo      ! loop of itype
        enddo ! loop of iel
        deallocate(ptr_nwbElm)
        !---------negative side of free surface-----------
        do iel = 1, nwb_cnter_Neg
            do itype = 1, itype_max
                select case (itype)
                case (1)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
                case (2)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
                case (3)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    do while ( associated( current ) )
                        previous => current
                        current => current%next
                        deallocate( previous )
                    end do
                enddo   ! loop of isimplx
            enddo      ! loop of itype
        enddo ! loop of iel
        deallocate(ptr_NegnwbElm)

        !------- clear dynamic memory at the end of run (completed)----------------------------
        print*,'final time step = ',ntcabs
    else
        !if (irangp.le.0) then
        if (mod(ntcabs,1) == 0) then
            print*,'time step = ',ntcabs
        endif
        ! endif
    endif
endif  !end of if "(allocated(ptr_nwbElm))..."  block
!-------------------------------------------------------------------------------------------
!         NEW RE-DISTANCING ROUTINE (end)
!-------------------------------------------------------------------------------------------
```

```
       ! --------------------------------------------------
       ! Close files at final time step
       ! --------------------------------------------------

       if (ntcabs.eq.ntmabs) then

           !if (irangp.le.0) then

           do ii = 1, 1

               close(impout(ii))

           enddo

           !endif

       endif
       ! --------------------------------------------------
       ! Close files at final time step
       ! --------------------------------------------------

   endif !***********isuite if block
   return

contains

   !***********************
   !
   subroutine find_min_dist(cur,ntri,Lr,ir,sec,iside,dImin)

      use connectivity
      implicit none

      ! code saturne variables
      !integer          ndim   , ncelet
      !double precision rtp(ncelet,*)

      type(simplex), pointer :: cur
      double precision sec(3),dImin
      integer          ntri, iside,ir(4)
      logical          Lr(4)

      ! local variables
      double precision, parameter         ::  small_value = 1.0d-6
      double precision, parameter         ::  th_d = 0.3333333333333
      double precision  k(3),a(3),s1(3),b(3),c(3),s2(3),dif(3),phi_val(4),d(3)
      double precision  v_hat(3),val,tdiv1,tdiv,a_matrix(4,4),v(3),phi_interp
      real  p_x(3),simVec(4,3)

      !-----------positive side of free surface--------------------
      if (Lr(1) ) then
          k(1)   = scale_length * (xyzcen(1,ir(1) ) )  ! kth node
          k(2)   = scale_length * (xyzcen(2,ir(1) ) )
```

```
    k(3)   = scale_length * (xyzcen(3,ir(1) ) )
else
    k(1)   = cur%sh_nb(ir(1),1)
    k(2)   = cur%sh_nb(ir(1),2)
    k(3)   = cur%sh_nb(ir(1),3)
endif
if (Lr(2) ) then
    s1(1) = scale_length * (xyzcen(1,ir(2)) ) - k(1)
    s1(2) = scale_length * (xyzcen(2,ir(2)) ) - k(2)
    s1(3) = scale_length * (xyzcen(3,ir(2)) ) - k(3)
else
    s1(1) = cur%sh_nb(ir(2),1) - k(1)
    s1(2) = cur%sh_nb(ir(2),2) - k(2)
    s1(3) = cur%sh_nb(ir(2),3) - k(3)
endif
if (Lr(3) ) then
    s2(1) = scale_length * (xyzcen(1,ir(3) ) ) - k(1)
    s2(2) = scale_length * (xyzcen(2,ir(3) ) ) - k(2)
    s2(3) = scale_length * (xyzcen(3,ir(3) ) ) - k(3)
else
    s2(1) = cur%sh_nb(ir(3),1) - k(1)
    s2(2) = cur%sh_nb(ir(3),2) - k(2)
    s2(3) = cur%sh_nb(ir(3),3) - k(3)
endif
if ( ntri == 4) then
    if (Lr(4) ) then
        d(1)   = scale_length * (xyzcen(1,ir(4) ) )
        d(2)   = scale_length * (xyzcen(2,ir(4) ) )
        d(3)   = scale_length * (xyzcen(3,ir(4) ) )
    else
        d(1)   = cur%sh_nb(ir(4),1)
        d(2)   = cur%sh_nb(ir(4),2)
        d(3)   = cur%sh_nb(ir(4),3)
    endif
endif
call cross_product(s1,s2,v)
a     =  s1 + k
b     =  s2 + k
if ( ntri == 3) then
    c     = (a + b + k) * th_d - sec
else if ( ntri == 4) then
    c     = (a + b + d + k) * 0.25 - sec
endif
val   = dot_product(c,v)
if (val < 0.0d0) call cross_product(s2,s1,v)
tdiv     = dabs(dot_product(v,v))
if (tdiv < small_value ) return
tdiv1    =  dsqrt(tdiv)
v_hat(1) = v(1)/tdiv1
v_hat(2) = v(2)/tdiv1
v_hat(3) = v(3)/tdiv1
dif      = k - sec
val = dot_product(dif,v_hat)
```

```
   ! establish point of intersection p_x on face of simplex
   call create_vec(sec, v_hat, val, p_x)
   if ( ntri == 3) then
       d = 0.5 * (a + b)
   endif
   call convert_Vec(4,k,a,d,b,simVec)
   if (check_simplex(ntri,simVec,p_x) ) then
       call calc_invert_matrix(simVec,a_matrx,4)
       phi_val(1) = rtp(ir(1), isca(2) )
       phi_val(2) = rtp(ir(2), isca(2) )
       phi_val(4) = rtp(ir(3), isca(2) )
       if ( ntri == 3) then
           phi_val(3) = 0.5 * (phi_val(2) + phi_val(4))
       else if ( ntri == 4) then
           phi_val(3) = rtp(ir(4), isca(2) )
       endif
       phi_interp = calc_phi_interp(4,phi_val,a_matrx,p_x)
       b          =  p_x - sec
       if (iside == 1 ) then
           dImin = phi_interp + sqrt( dabs(dot_product(b,b)) )
       else
           dImin = phi_interp - sqrt( dabs(dot_product(b,b)) )
       endif
   endif
   return
end subroutine find_min_dist

!************************
!  calculate interpolated level set value from neighbour level set values
!  using 3D geometrically isotropic trilinear interpolation
!************************
double precision function calc_phi_interp(ntri,prop_neighbs,a_matrx,p_x)
   implicit none
   real, dimension(:,:)                            :: a_matrx
   real, dimension(:)                              :: p_x
   double precision, dimension(:)                  :: prop_neighbs
   double precision,dimension(:), allocatable      :: c
   double precision                                :: prop_sum
   integer                                         :: i, j, ntri

   allocate(c(ntri))
   do i = 1, ntri
       c(i) = 0.0
       do j = 1, ntri
           c(i) = c(i) + a_matrx(i,j) * prop_neighbs(i)
       enddo
   enddo
   prop_sum = 0.0
   do i = 1, ntri
       if (i == 1) then
           prop_sum = prop_sum + c(i)
       else
           prop_sum = prop_sum + c(i) * p_x(i-1)
```

```
      endif
   enddo
   calc_phi_interp = prop_sum
   deallocate(c)
   return
end function calc_phi_interp

!*******************************
!  create_simplices
!
!  This subroutine creates the simplices associated with node k and its neighbour
!              cells forming part of the free surface region near node k
!*******************************
subroutine create_simplices(iside,ptr, Nneg, Npos,itype_max)
   use connectivity
   implicit none

   type (narrow_band_element)        :: ptr
   integer                           :: Nneg, Npos, itype_max,iside

   ! local variables
   type(simplex), pointer           :: current,previous
   integer itype,isimplex, inext, isamplePos,p(3), icount

   do itype = 1, itype_max
       if (itype == 1) then
           nullify(ptr%ptr_cutTetra_type1)
           allocate(ptr%ptr_cutTetra_type1)
           current => ptr%ptr_cutTetra_type1
           current%use_it = .true.
           isimplex = 0
           do isampleNeg = 1, Nneg
               if (iside == 0) then       ! positive side
                   p(1)     = ptr%Pneg(isampleNeg)
               else if (iside == 1) then  ! negative side
                   p(1)     = ptr%Ppos(isampleNeg)
               endif
               inext = 1
               do isamplePos = 1, Npos
                   if (iside == 0) then      ! positive side
                       p(2) = ptr%Ppos(isamplePos)
                   else if (iside == 1) then ! negative side
                       p(2) = ptr%Pneg(isamplePos)
                   endif
                   inext = inext + 1
                   if (inext == Npos) inext = 1
                   if (iside == 0) then      ! positive side
                       p(3) = ptr%Ppos(inext)
                   else if (iside == 1) then ! negative side
                       p(3) = ptr%Pneg(inext)
                   endif
                   isimplex = isimplex + 1
                   current%sim_node(1) = p(1)
```

```
                current%sim_node(2) = p(2)
                current%sim_node(3) = p(3)
                allocate(current%next)
                nullify( current%next%next )
                current => current%next
                current%use_it = .true.
            enddo
        enddo
        ptr%ptr_cutTetra_type1%sim_count = isimplex
    else if (itype == 2) then
        nullify(ptr%ptr_cutTetra_type2)
        allocate(ptr%ptr_cutTetra_type2)
        current => ptr%ptr_cutTetra_type2
        current%use_it = .true.
        isimplex = 0
        inext = 1
        do isampleNeg = 1, Nneg
            if (iside == 0) then       ! positive side
                p(1)     = ptr%Pneg(isampleNeg)
            else if (iside == 1) then  ! negative side
                p(1)     = ptr%Ppos(isampleNeg)
            endif
            inext    = inext + 1
            if (inext == Nneg) inext = 1
            if (iside == 0) then       ! positive side
                p(2)     =   ptr%Pneg(inext)
            else if (iside == 1) then  ! negative side
                p(2)     =   ptr%Ppos(inext)
            endif
            do isamplePos = 1, Npos
                if (iside == 0) then      ! positive side
                    p(3) = ptr%Ppos(isamplePos)
                else if (iside == 1) then ! negative side
                    p(3) = ptr%Pneg(isamplePos)
                endif
                isimplex = isimplex + 1
                current%sim_node(1) = p(1)
                current%sim_node(2) = p(2)
                current%sim_node(3) = p(3)
                allocate(current%next)
                nullify( current%next%next )
                current => current%next
                current%use_it = .true.
            enddo
        enddo
        ptr%ptr_cutTetra_type2%sim_count = isimplex
    else if (itype == 3) then
        nullify(ptr%ptr_cutTetra_type3)
        allocate(ptr%ptr_cutTetra_type3)
        current => ptr%ptr_cutTetra_type3
        current%use_it = .true.
        isimplex = 0
        inext    = 1 ; inext2 = 2
```

```
          do isampleNeg = 1, Nneg
              if (iside == 0) then       ! positive side
                  p(1)     = ptr%Pneg(isampleNeg)
              else if (iside == 1) then  ! negative side
                  p(1)     = ptr%Ppos(isampleNeg)
              endif
              inext = inext + 1
              if (inext == Nneg) inext = 1
              if (iside == 0) then       ! positive side
                  p(2)     =   ptr%Pneg(inext)
              else if (iside == 1) then  ! negative side
                  p(2)     =   ptr%Ppos(inext)
              endif
              inext2 = inext2 + 1
              if (inext2 == Nneg) inext2 = 1
              if (iside == 0) then       ! positive side
                  p(3)     =   ptr%Pneg(inext2)
              else if (iside == 1) then  ! negative side
                  p(3)     =   ptr%Ppos(inext2)
              endif
              isimplex = isimplex + 1
              current%sim_node(1) = p(1)
              current%sim_node(2) = p(2)
              current%sim_node(3) = p(3)
              allocate(current%next)
              nullify( current%next%next )
              current => current%next
              current%use_it = .true.
          enddo
          ptr%ptr_cutTetra_type3%sim_count = isimplex
      endif
   enddo
   return

end subroutine create_simplices


!****************
!  secondary_cell_select
!
!  This subroutine finds what cells are secondary cells in the 3D environment either side of the isosurface
!****************
integer function  secondary_cell_select(ineighb)
   use connectivity
   implicit none
   ! code saturne variables
   ! integer          ndim   , ncelet
   ! double precision rtp(ncelet,*)

   integer(8), intent(in )                :: ineighb


   ! local variables
   integer             ifac, isgn, Number_Of_Faces
   double precision    phi_cenck, phi_nbck
   secondary_cell_select = 1
   phi_cenck       = rtp(ineighb,isca(2))
   if ( dabs(phi_cenck) < tiny(1.0)) then
       secondary_cell_select = 0
       return
   else if ( phi_cenck < 0.0d0) then
       isgn = -1
   else if ( phi_cenck > 0.0d0) then
       isgn = 1
   endif
   Number_Of_Faces=count(nbcell(ineighb,:)>-ihuge)
   do ifac = 1, Number_Of_Faces
       phi_nbck  = rtp(nbcell(ineighb,ifac),isca(2))
       if ( isgn > 0) then
           if ( phi_nbck < 0.0d0) then
               secondary_cell_select = -1
               return
           endif
       else if ( isgn < 0 ) then
           if ( phi_nbck > 0.0d0) then
               secondary_cell_select = -1
               return
           endif
       endif
   enddo  ! ifac loop
end function secondary_cell_select
!*************************
!****************************************
!  Vol_k
!
!  This subroutine calculates the difference in volumes defined by phi and phi*,
!  piecewise constant simplexwise mass correction function
!  and 3D reconstruction of the isosurface locally over simplex k
!****************************************
subroutine Vol_k(icheck,iside,current,ival_nb,itype,phi_value,val_adj,Area_K,Volume_k)
   use connectivity
   implicit none


   type(simplex), pointer, intent(in)            :: current

   logical, intent(in)                           :: icheck
   integer, intent(in)                           :: iside
   integer, intent(in)                           :: itype
   integer, intent(in)                           :: ival_nb
   double precision, intent(in )                 :: phi_value,val_adj
   double precision, intent(out)                 :: Area_K
   double precision, intent(out)                 :: Volume_k

   ! local variables
   double precision, parameter                   ::  vol_const = 0.16666667
   double precision  a(3),b(3),c(3),c1(3),c4(3),k(3),dif(3),v(3),s(3)
   double precision  sh_nb_temp(4,3),s_val, S_K, S_h, phi,vol_A, vol_B
   double precision  Area_K2,Area_K1,s1(3),s2(3),p3(3),pmid(3),vol_01
   double precision  vol_021,vol_022,vol_03,c2(3),c3(3),Area_1
   integer           ieln, inod2,Nmax,is1, ir, it_value1, it_value2

   !-----------------------------------------------------------------------
   ! Adjust reconstructed isosurface Sk by 'valadj' amount normal to surface
   !-----------------------------------------------------------------------
   select case (itype)
   case (1)
       Nmax = 3
   case (2)
       Nmax = 4
   case (3)
       Nmax = 3
   case default
       call error_message('Error1 in Vol_k in USINV')
   end select
   !----------------------
   if (icheck) then
       do inod2 = 1, Nmax
           sh_nb_temp(inod2,1) = current%sh_nb(inod2,1)
           sh_nb_temp(inod2,2) = current%sh_nb(inod2,2)
           sh_nb_temp(inod2,3) = current%sh_nb(inod2,3)
       enddo
   else
       do inod2 = 1, Nmax
           if (itype == 1) then
               if ( inod2 == 1) then
                   ir   =  ival_nb                ! Kth node
                   phi  =     phi_value + val_adj
               else
                   ir  =  current%sim_node(inod2)
                   phi =  rtp(ir,isca(2))  + val_adj        ! new
               endif
               ieln =  current%sim_node(1)
           else if (itype == 2) then
               select case (inod2)
               case (1,2)
                   ieln  =  current%sim_node(inod2)
                   ir   =  ival_nb
                   phi = phi_value + val_adj               ! new
               case (3,4)
                   ival = inod2 - 2
                   ieln  =  current%sim_node(ival)
                   ir   =  current%sim_node(3)
                   phi = rtp(ir,isca(2)) + val_adj         ! new
               case default
                   call error_message('Error2 in Vol_k in USINV')
               end select
           else if (itype == 3) then
               ieln  =  current%sim_node(inod2)
```

```
                ir    =  ival_nb
                phi = phi_value + val_adj                 ! new
            else
                call error_message('Error3 in Vol_k in USINV')
            endif
            s_val    = phi  -  rtp(ieln,isca(2))
            call check_zero(s_val,'Error1 in USINV with Sk reconstruction at iel = ',iel)
            S_h  = - rtp(ieln,isca(2))/s_val
            a(1) = scale_length * (xyzcen(1,ir) )
            a(2) = scale_length * (xyzcen(2,ir) )
            a(3) = scale_length * (xyzcen(3,ir) )
            b(1) = scale_length * (xyzcen(1,ieln) )
            b(2) = scale_length * (xyzcen(2,ieln) )
            b(3) = scale_length * (xyzcen(3,ieln) )
            call position_vec(a, b, S_h, c)
            sh_nb_temp(inod2,1) = c(1)
            sh_nb_temp(inod2,2) = c(2)
            sh_nb_temp(inod2,3) = c(3)
        enddo
    endif
    !-----------------------------------------------------------------------
    ! calculate new simplex volume based on adjustment to reconstructed isosurface Sk
    !-----------------------------------------------------------------------
    ! volume of small negative simplex
    c1(1) = sh_nb_temp(1,1)  ! edge of free surface near kth node
    c1(2) = sh_nb_temp(1,2)
    c1(3) = sh_nb_temp(1,3)
    if ( itype == 2 ) then
        !find volume of first sub-tetrahedron
        ir   =  current%sim_node(3)
        do id = 1,ndim
            a(id)  = scale_length * (xyzcen(id,ival_nb) )
            c2(id) = sh_nb_temp(2,id)
            c3(id) = sh_nb_temp(3,id)
            c4(id) = sh_nb_temp(4,id)
            p3(id) = scale_length * (xyzcen(id,ir) )
        enddo
        !---- vol01 volume----------------
        pmid = 0.5 * (p3 + a)
        s1 = pmid - a
        s2 = c1   - a
        call cross_product(s1,s2,v)
        dif   = c2 - a
        vol_01 = dabs(dot_product(dif, v))
        !vol_021 volume
        s1  = c1 - c2
        s2  = c4 - c2
        dif = pmid - c2
        call cross_product(s1,s2,v)
        vol_021 = dabs(dot_product(dif, v))
        !vol_022 volume
        s1  = c1 - c3
        s2  = c4 - c3
```

```
dif = pmid - c3
call cross_product(s1,s2,v)
vol_022 = dabs(dot_product(dif, v))
!vol_03 volume
s1  = p3 - c4
s2  = pmid - c2
dif = pmid - c4
call cross_product(s1,s2,v)
vol_03 = dabs(dot_product(dif, v))
if ( iside == 0) then
    Volume_k = vol_const*(vol_01 + vol_021 + vol_022 + vol_03)
else
    vol_A = vol_01 + vol_021 + vol_022 + vol_03
endif
! find Area_k
s1 = c3 - c1
s2 = c2 - c1
call cross_product(s1,s2,v)
S_k     = dabs(dot_product(v,v))
if (S_k < tiny(1.0)) then
    Area_1 = 0.0
else
    Area_1 = 0.5 * dsqrt(S_k)
endif
s1 = c3 - c4
s2 = c2 - c4
call cross_product(s1,s2,v)
S_k     = dabs(dot_product(v,v))
if (S_k < tiny(1.0)) then
    Area_K = Area_1
else
    Area_K = (0.5 * dsqrt(S_k)) + Area_1
endif
if (iside == 1) then
    ieln = current%sim_node(1)
    do id = 1,ndim
        a(id)  = scale_length * (xyzcen(id,ival_nb) )
        b(id)  = scale_length * (xyzcen(id,ieln) )
    enddo
    dif = b - a
    is1 = current%sim_node(2)
    s1(1)   = scale_length * (xyzcen(1,is1) ) - a(1)
    s1(2)   = scale_length * (xyzcen(2,is1) ) - a(2)
    s1(3)   = scale_length * (xyzcen(3,is1) ) - a(3)
    is1 = current%sim_node(3)
    s2(1)   = scale_length * (xyzcen(1,is1) ) - a(1)
    s2(2)   = scale_length * (xyzcen(2,is1) ) - a(2)
    s2(3)   = scale_length * (xyzcen(3,is1) ) - a(3)
    call cross_product(s1,s2,v)
    vol_B = dabs(dot_product(dif,v))
    volume_K = vol_const * (vol_B - vol_A)
endif
return
```

```
else
    if (itype == 1) then
        ieln =  current%sim_node(1)
    else if (itype == 3) then
        ieln = ival_nb   ! positive peak at node k
    else
        call error_message('Error4 in Vol_k in USINV')
    endif
    b(1) = scale_length * (xyzcen(1,ieln) )
    b(2) = scale_length * (xyzcen(2,ieln) )
    b(3) = scale_length * (xyzcen(3,ieln) )
    dif(1) = b(1) - c1(1)   ! height of small simplex
    dif(2) = b(2) - c1(2)
    dif(3) = b(3) - c1(3)
    s1(1) = sh_nb_temp(2,1) - c1(1)
    s1(2) = sh_nb_temp(2,2) - c1(2)
    s1(3) = sh_nb_temp(2,3) - c1(3)
    s2(1) = sh_nb_temp(3,1) - c1(1)
    s2(2) = sh_nb_temp(3,2) - c1(2)
    s2(3) = sh_nb_temp(3,3) - c1(3)
    call cross_product(s1,s2,v)
    S_k     = dabs(dot_product(v,v))
    if (S_k < tiny(1.0)) then
        Area_K       = 0.0
    else
        Area_K       = 0.5 * dsqrt(S_k)
    endif
    vol_A =  dabs(dot_product(dif, v))
    if (iside == 0) then
        it_value1 = 3
        it_value2 = 1
    else if (iside == 1) then
        it_value1 = 1
        it_value2 = 3
    endif
    if (itype == it_value1) then
        Volume_k = vol_const * vol_A
        return
    else if (itype == it_value2) then
        !------ larger simplex next
        if (iside == 0) then
            ieln =  ival_nb
        else if (iside == 1) then
            ieln =  current%sim_node(1)
        endif
        a(1) = scale_length * (xyzcen(1,ieln) )
        a(2) = scale_length * (xyzcen(2,ieln) )
        a(3) = scale_length * (xyzcen(3,ieln) )
        dif(1) = b(1) - a(1)   ! height of the larger simplex
        dif(2) = b(2) - a(2)
        dif(3) = b(3) - a(3)
        is1 =  current%sim_node(2)
        s1(1) = scale_length * (xyzcen(1,is1) ) - a(1)
```

```
            s1(2) = scale_length * (xyzcen(2,is1) ) - a(2)
            s1(3) = scale_length * (xyzcen(3,is1) ) - a(3)
            is1 =  current%sim_node(3)
            s2(1) = scale_length * (xyzcen(1,is1) ) - a(1)
            s2(2) = scale_length * (xyzcen(2,is1) ) - a(2)
            s2(3) = scale_length * (xyzcen(3,is1) ) - a(3)
            call cross_product(s1,s2,v)
            vol_B =  dabs(dot_product(dif, v))
            !-------resultant simplex volume below free surface
            Volume_k = vol_const * (vol_B - vol_A)
            return
        else
            call error_message('Error5 in Vol_k in USINV')
        endif
    endif
end subroutine Vol_k
!*****************************************
!  deltaV
!
!  This subroutine calculates the difference in volumes defined by phi and phi*, node wise mass correction fu
!  which is used in the false position algorithm to find the constant C which globally preserves volume delta
!*****************************************
double precision function deltaV(iside,ptr, C)
   use connectivity
   implicit none


   type (narrow_band_element), intent(in)       :: ptr
   double precision, intent(in )                :: C
   integer,intent(in)                           :: iside

   ! local variables
   integer             Nneg, Npos,itype_max,itype,isimpx,isimplxMax
   integer             ival_nb
   double precision    s_k, phi, phi_starr, Cxi_h, f_sum, Area_K
   double precision    vol2, vol1

   f_sum = 0.0
   Cxi_h = ptr%xi_h * C
   phi_starr  =   ptr%ph_star
   Npos = ptr%pos
   Nneg = ptr%neg
   ival_nb    =   ptr%nwb_index
   if (iside == 0) then
       if (Nneg > 3) then
           itype_max = 3
       else
           itype_max = Nneg
       endif
   else if (iside == 1) then
       if (Npos > 3) then
           itype_max = 3
       else
```

```
           itype_max = Npos
       endif
   endif
   do itype = 1, itype_max
       select case (itype)
       case (1)
           isimplxMax = ptr%ptr_cutTetra_type1%sim_count
           current => ptr%ptr_cutTetra_type1
       case (2)
           isimplxMax = ptr%ptr_cutTetra_type2%sim_count
           current => ptr%ptr_cutTetra_type2
       case (3)
           isimplxMax = ptr%ptr_cutTetra_type3%sim_count
           current => ptr%ptr_cutTetra_type3
       case default
           call error_message('Error1 in step1 with isimplxMax in USINV')
       end select
       !  *** scan for isimplxMax per simplex cut type considered  ***
       if (isimplxMax > 0) then
           do isimpx = 1, isimplxMax
               if (current%use_it) then                              !new
                   vol2 = current%vol_phi
                   call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,Cxi_h,Area_K,vol1)
                   f_sum = f_sum + (vol2 - vol1)
               endif                                                 !new
               current => current%next
           enddo
       endif
   enddo
   deltaV = f_sum
   return
end function deltaV
!=======================================
!----------------------------------------------------------------------------------------------------
!  Returns inverted matrix for step 6 needed with 3D geometrically isotropic
!  trilinear interpolation involved with the intersection on a face of a tetrahedral simplex
!----------------------------------------------------------------------------------------------------
subroutine calc_invert_matrix(x,a_matrx,ntri)
   implicit none
   real, dimension(:,:), allocatable :: Matrix, invMatrix
   real, dimension(:,:)              :: a_matrx, x
   integer                           :: ntri

   allocate(Matrix(ntri,ntri))
   allocate(invMatrix(ntri,ntri))
   do i = 1, ntri
       do j = 1, ntri
           if (j == 1) then
               Matrix(i,j) = 1.0
           else
               Matrix(i,j) = x(i,j-1)
           endif
       enddo
   enddo
   call FindInv(Matrix, invMatrix, ntri, ErrorFlag)
   do i = 1, ntri
       do j = 1, ntri
           a_matrx(i,j) = invMatrix(i,j)
       enddo
   enddo
   deallocate(Matrix)
   deallocate(invMatrix)
   return
end subroutine calc_invert_matrix
!--------------------------------------------------------------------------------
!  check to ensure that the intersection p_x is within the particular face
!  of the tetrahedral simplex
!--------------------------------------------------------------------------------
logical function check_simplex(n_tri,simVec,p_x)
   implicit none
   real, dimension(:,:)             :: simVec
   real, dimension(:)               :: p_x

   ! local variables
   double precision  n_vec(3),r_mid(3),nface_hat(3),sim_dif(3), t_vec(3),val
   double precision  v_smll,n_val,dval
   integer id,n_tri

   do i = 1, n_tri
       do id = 1, ndim
           if (i < n_tri) then
               t_vec(id) = simVec(i+1,id) - simVec(i,id)
               sim_dif(id) = (simVec(i+1,id) + simVec(i,id) ) * 0.5 - p_x(id)
           else
               t_vec(id) = simVec(1,id) - simVec(i,id)
               sim_dif(id) = (simVec(1,id) + simVec(i,id) ) * 0.5 - p_x(id)
           endif
       enddo
       call cross_product(nface_hat, t_vec, n_vec)
       n_val = dsqrt(dabs(dot_product(n_vec,n_vec)))
       if ( n_val < tiny(1.0))  then
           check_simplex = .true.
           return
       endif
       n_vec(1) = n_vec(1)/n_val;n_vec(2) = n_vec(2)/n_val;n_vec(3) = n_vec(3)/n_val;
       val = dot_product(sim_dif, n_vec)
       dval = dsqrt(dabs(dot_product(sim_dif,sim_dif)))
       v_smll = dval * 0.01
       if (dabs(val) < v_smll) then
           check_simplex = .true.
       else if (val >= 0.0) then
           check_simplex = .true.
       else
           check_simplex = .false.
           return
       endif
   enddo
   return
end function check_simplex
!----------------------------------------------------------------------------------

!==========================================
subroutine convert_Vec(ntri,k,a,b,c,simVec)

   implicit none

   integer                       , intent (IN)    :: ntri
   double precision, dimension(:), intent (IN)    :: k
   double precision, dimension(:), intent (IN)    :: a
   double precision, dimension(:), intent (IN)    :: b
   double precision, dimension(:), intent (IN)    :: c
   real          , dimension(:,:), intent (OUT)   :: simVec

   do i = 1, ntri
       if (i == 1) then
           do j = 1, ndim
               simVec(i,j) = k(j)
           enddo
       else if (i == 2) then
           do j = 1, ndim
               simVec(i,j) = a(j)
           enddo
       else if (i == 3) then
           do j = 1, ndim
               simVec(i,j) = b(j)
           enddo
       else if (i == 4) then
           do j = 1, ndim
               simVec(i,j) = c(j)
           enddo
       endif
   enddo
   return

end subroutine convert_Vec
!==========================================
subroutine create_vec (A, B, TSCAL, C)

   implicit none

   double precision, dimension(3), intent (IN)    :: A                             ! multiplicand 3-vec
   double precision, dimension(3), intent (IN)    :: B                             ! multiplier 3-vecto
   double precision,               intent (IN)    :: TSCAL                         ! scalar
   real,             dimension(3), intent (OUT)   :: C                             ! result: 3-vector p

   C(1) = A(1) + TSCAL * B(1)
   C(2) = A(2) + TSCAL * B(2)
   C(3) = A(3) + TSCAL * B(3)
   return
```

```
end subroutine create_vec

!=========================================
subroutine POSITION_VEC (A, B, TSCAL, C)

   implicit none

   double precision, dimension(3), intent (IN)    :: A                           ! multiplicand 3-vector
   double precision, dimension(3), intent (IN)    :: B                           ! multiplier 3-vector
   double precision,               intent (IN)    :: TSCAL                       ! scalar
   double precision, dimension(3), intent (OUT)   :: C                           ! result: 3-vector position
   !local variables
   double precision, dimension(3)                 :: D

   D(1) = B(1) - A(1)
   D(2) = B(2) - A(2)
   D(3) = B(3) - A(3)
   C(1) = A(1) + TSCAL * D(1)
   C(2) = A(2) + TSCAL * D(2)
   C(3) = A(3) + TSCAL * D(3)
   return

end subroutine POSITION_VEC
!===========================================

function dot_product (V1, V2) result (PROD)

   implicit none

   double precision, dimension(3), intent(IN) :: V1, V2
   double precision :: PROD


   PROD = V1(1)*V2(1) + V1(2)*V2(2) + V1(3)*V2(3)
   return

end function DOT_PRODUCT
!***********************
!***********************
!  CROSS_PRODUCT
!
!  Returns the right-handed vector cross product of two 3d-vectors:  C = A x B.
!
!  Code pasted from: http://www.davidgsimpson.com/software/crossprd_f90.txt
!  Checked veracity with Wikipedia
!***********************

subroutine CROSS_PRODUCT (A, B, C)                                            ! cross product (right-handed)

   implicit none                                                               ! no default typing

   double precision, dimension(3), intent (IN)    :: A                           ! multiplicand 3d-vector
   double precision, dimension(3), intent (IN)    :: B                             ! multiplier 3d-vect
   double precision, dimension(3), intent (OUT)   :: C                             ! result: 3d-vector


   C(1) = A(2)*B(3) - A(3)*B(2)                                                    ! compute cross prod
   C(2) = A(3)*B(1) - A(1)*B(3)
   C(3) = A(1)*B(2) - A(2)*B(1)

   return

end subroutine CROSS_PRODUCT

!-----------------------------------------------------------------------------------
!Subroutine to find the inverse of a square matrix
!Author : Louisda16th a.k.a Ashwith J. Rego
!Reference : Algorithm has been well explained in:
!http://math.uww.edu/~mcfarlat/inverse.htm
!http://www.tutor.ms.unimelb.edu.au/matrix/matrix_inverse.html
!------------------------------------------------------------------------------------
subroutine FINDInv(matrix, inverse, n, errorflag)
   implicit none
   !Declarations
   integer, intent(IN) :: n
   integer, intent(OUT) :: errorflag  !Return error status. -1 for error, 0 for normal
   real, intent(IN), dimension(n,n) :: matrix  !Input matrix
   real, intent(OUT), dimension(n,n) :: inverse !Inverted matrix

   logical :: FLAG = .true.
   integer :: i, j, k, l
   real :: m
   real, dimension(n,2*n) :: augmatrix !augmented matrix

   !Augment input matrix with an identity matrix
   do i = 1, n
       do j = 1, 2*n
           if (j <= n ) then
               augmatrix(i,j) = matrix(i,j)
           else if ((i+n) == j) then
               augmatrix(i,j) = 1
           else
               augmatrix(i,j) = 0
           endif
       end do
   end do

   !Reduce augmented matrix to upper traingular form
   do k =1, n-1
       if (augmatrix(k,k) == 0) then
           FLAG = .false.
           do i = k+1, n
               if (augmatrix(i,k) /= 0) then
                   do j = 1,2*n
                       augmatrix(k,j) = augmatrix(k,j)+augmatrix(i,j)
```

```
                end do
                FLAG = .true.
                exit
            endif
            if (FLAG .eqv. .false.) then
                print*, "Matrix is non - invertible"
                inverse = 0
                errorflag = -1
                return
            endif
        end do
    endif
    do j = k+1, n
        m = augmatrix(j,k)/augmatrix(k,k)
        do i = k, 2*n
            augmatrix(j,i) = augmatrix(j,i) - m*augmatrix(k,i)
        end do
    end do
end do

!Test for invertibility
do i = 1, n
    if (augmatrix(i,i) == 0) then
        print*, "Matrix is non - invertible"
        inverse = 0
        errorflag = -1
        return
    endif
end do

!Make diagonal elements as 1
do i = 1 , n
    m = augmatrix(i,i)
    do j = i , (2 * n)
        augmatrix(i,j) = (augmatrix(i,j) / m)
    end do
end do

!Reduced right side half of augmented matrix to identity matrix
do k = n-1, 1, -1
    do i =1, k
        m = augmatrix(i,k+1)
        do j = k, (2*n)
            augmatrix(i,j) = augmatrix(i,j) -augmatrix(k+1,j) * m
        end do
    end do
end do

!store answer
do i =1, n
    do j = 1, n
        inverse(i,j) = augmatrix(i,j+n)
    end do
```

```
      end do
      errorflag = 0
   end subroutine FINDinv
end subroutine usproj

!-------------------------------------------------------------------------------

!                      Code_Saturne version 2.0.0-rc1
!                      --------------------------

!     This file is part of the Code_Saturne Kernel, element of the
!     Code_Saturne CFD tool.

!     Copyright (C) 1998-2009 EDF S.A., France

!     contact: saturne-support@edf.fr

!     The Code_Saturne Kernel is free software; you can redistribute it
!     and/or modify it under the terms of the GNU General Public License
!     as published by the Free Software Foundation; either version 2 of
!     the License, or (at your option) any later version.

!     The Code_Saturne Kernel is distributed in the hope that it will be
!     useful, but WITHOUT ANY WARRANTY; without even the implied warranty
!     of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!     GNU General Public License for more details.

!     You should have received a copy of the GNU General Public License
!     along with the Code_Saturne Kernel; if not, write to the
!     Free Software Foundation, Inc.,
!     51 Franklin St, Fifth Floor,
!     Boston, MA  02110-1301  USA

!-------------------------------------------------------------------------------

subroutine usproj &
      !================

      ( idbia0 , idbra0 ,                                              &
      ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
      nnod   , lndfac , lndfbr , ncelbr ,                            &
      nvar   , nscal  , nphas  ,                                     &
      nbpmax , nvp    , nvep   , nivep  , ntersl , nvlsta , nvisbr , &
      nideve , nrdeve , nituse , nrtuse ,                            &
      ifacel , ifabor , ifmfbr , ifmcel , iprfml , maxelt , lstelt , &
      ipnfac , nodfac , ipnfbr , nodfbr , itepa  ,                   &
      idevel , ituser , ia     ,                                     &
      xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
      dt     , rtpa   , rtp    , propce , propfa , propfb ,          &
      coefa  , coefb  ,                                              &
      ettp   , ettpa  , tepa   , statis , stativ , tslagr , parbor , &
      rdevel , rtuser , ra     )
```

```
!=======================================
! Purpose:
! -------

!    User subroutine.

!    Called at end of each time step, very general purpose
!    (i.e. anything that does not have another dedicated user subroutine)


! Several examples are given here:

!  - compute a thermal balance
!    (if needed, see note  below on adapting this to any scalar)

!  - compute global efforts on a subset of faces

!  - arbitrarily modify a calculation variable

!  - extract a 1 d profile

!  - print a moment

!  - examples on using parallel utility functions

! These examples are valid when using periodicity (iperio .gt. 0)
! and in parallel (irangp .ge. 0).

! The thermal balance compution also illustates a few other features,
! including the required precautions in parallel or with periodicity):
! - gradient calculation
! - computation of a value depending on cells adjacent to a face
!   (see synchronization of Dt and Cp)
! - computation of a global sum in parallel (parsom)


! Cells, boundary faces and interior faces identification
! =======================================================

! Cells, boundary faces and interior faces may be identified using
! the subroutines 'getcel', 'getfbr' and 'getfac' (respectively).

!  getfbr(string, nelts, eltlst):
!  - string is a user-supplied character string containing selection criteria;
!  - nelts is set by the subroutine. It is an integer value corresponding to
!    the number of boundary faces verifying the selection criteria;
!  - lstelt is set by the subroutine. It is an integer array of size nelts
!    containing the list of boundary faces verifying the selection criteria.

!  string may contain:
!  - references to colors (ex.: 1, 8, 26, ...)
!  - references to groups (ex.: inlet, group1, ...)
!  - geometric criteria (ex. x < 0.1, y >= 0.25, ...)
!  These criteria may be combined using logical operators ('and', 'or') and
!  parentheses.
!  Example: '1 and (group2 or group3) and y < 1' will select boundary faces
!  of color 1, belonging to groups 'group2' or 'group3' and with face center
!  coordinate y less than 1.

! Similarly, interior faces and cells can be identified using the 'getfac'
! and 'getcel' subroutines (respectively). Their syntax are identical to
! 'getfbr' syntax.

! For a more thorough description of the criteria syntax, it can be referred
! to the user guide.


!-------------------------------------------------------------------------------
! Arguments
!__________________.____._____.________________________________________________.
! name             !type!mode ! role                                           !
!__________________!____!_____!________________________________________________!
! idbia0           ! i  ! <-- ! number of first free position in ia            !
! idbra0           ! i  ! <-- ! number of first free position in ra            !
! ndim             ! i  ! <-- ! spatial dimension                              !
! ncelet           ! i  ! <-- ! number of extended (real + ghost) cells        !
! ncel             ! i  ! <-- ! number of cells                                !
! nfac             ! i  ! <-- ! number of interior faces                       !
! nfabor           ! i  ! <-- ! number of boundary faces                       !
! nfml             ! i  ! <-- ! number of families (group classes)             !
! nprfml           ! i  ! <-- ! number of properties per family (group class)  !
! nnod             ! i  ! <-- ! number of vertices                             !
! lndfac           ! i  ! <-- ! size of nodfac indexed array                   !
! lndfbr           ! i  ! <-- ! size of nodfbr indexed array                   !
! ncelbr           ! i  ! <-- ! number of cells with faces on boundary         !
! nvar             ! i  ! <-- ! total number of variables                      !
! nscal            ! i  ! <-- ! total number of scalars                        !
! nphas            ! i  ! <-- ! number of phases                               !
! nbpmax           ! i  ! <-- ! max. number of particles allowed               !
! nvp              ! i  ! <-- ! number of particle-defined variables           !
! nvep             ! i  ! <-- ! number of real particle properties             !
! nivep            ! i  ! <-- ! number of integer particle properties          !
! ntersl           ! i  ! <-- ! number of return coupling source terms         !
! nvlsta           ! i  ! <-- ! number of Lagrangian statistical variables     !
! nvisbr           ! i  ! <-- ! number of boundary statistics                  !
! nideve, nrdeve   ! i  ! <-- ! sizes of idevel and rdevel arrays              !
! nituse, nrtuse   ! i  ! <-- ! sizes of ituser and rtuser arrays              !
! ifacel(2, nfac)  ! ia ! <-- ! interior faces -> cells connectivity           !
! ifabor(nfabor)   ! ia ! <-- ! boundary faces -> cells connectivity           !
! ifmfbr(nfabor)   ! ia ! <-- ! boundary face family numbers                   !
! ifmcel(ncelet)   ! ia ! <-- ! cell family numbers                            !
! iprfml           ! ia ! <-- ! property numbers per family                    !
!  (nfml, nprfml)  !    !     !                                                !
! maxelt           ! i  ! <-- ! max number of cells and faces (int/boundary)   !
! lstelt(maxelt)   ! ia ! --- ! work array                                     !
! ipnfac(nfac+1)   ! ia ! <-- ! interior faces -> vertices index (optional)    !
! nodfac(lndfac)   ! ia ! <-- ! interior faces -> vertices list (optional)     !
! ipnfbr(nfabor+1) ! ia ! <-- ! boundary faces -> vertices index (optional)    !
! nodfbr(lndfbr)   ! ia ! <-- ! boundary faces -> vertices list (optional)     !
! itepa            ! ia ! <-- ! integer particle attributes                    !
!  (nbpmax, nivep) !    !     !   (containing cell, ...)                       !
! idevel(nideve)   ! ia ! <-- ! integer work array for temporary development   !
! ituser(nituse)   ! ia ! <-- ! user-reserved integer work array               !
! ia(*)            ! ia ! --- ! main integer work array                        !
! xyzcen           ! ra ! <-- ! cell centers                                   !
!  (ndim, ncelet)  !    !     !                                                !
! surfac           ! ra ! <-- ! interior faces surface vectors                 !
!  (ndim, nfac)    !    !     !                                                !
! surfbo           ! ra ! <-- ! boundary faces surface vectors                 !
!  (ndim, nfabor)  !    !     !                                                !
! cdgfac           ! ra ! <-- ! interior faces centers of gravity              !
!  (ndim, nfac)    !    !     !                                                !
! cdgfbo           ! ra ! <-- ! boundary faces centers of gravity              !
!  (ndim, nfabor)  !    !     !                                                !
! xyznod           ! ra ! <-- ! vertex coordinates (optional)                  !
!  (ndim, nnod)    !    !     !                                                !
! volume(ncelet)   ! ra ! <-- ! cell volumes                                   !
! dt(ncelet)       ! ra ! <-- ! time step (per cell)                           !
! rtp, rtpa        ! ra ! <-- ! calculated variables at cell centers           !
!  (ncelet, *)     !    !     !  (at current and previous time steps)          !
! propce(ncelet, *)! ra ! <-- ! physical properties at cell centers            !
! propfa(nfac, *)  ! ra ! <-- ! physical properties at interior face centers   !
! propfb(nfabor, *)! ra ! <-- ! physical properties at boundary face centers   !
! coefa, coefb     ! ra ! <-- ! boundary conditions                            !
!  (nfabor, *)     !    !     !                                                !
! ettp, ettpa      ! ra ! <-- ! particle-defined variables                     !
!  (nbpmax, nvp)   !    !     !  (at current and previous time steps)          !
! tepa             ! ra ! <-- ! real particle properties                       !
!  (nbpmax, nvep)  !    !     !  (statistical weight, ...                      !
! statis           ! ra ! <-- ! statistic means                                !
!  (ncelet, nvlsta)!    !     !                                                !
! stativ(ncelet,   ! ra ! <-- ! accumulator for variance of volume statisitics !
!        nvlsta -1)!    !     !                                                !
! tslagr           ! ra ! <-- ! Lagrangian return coupling term                !
!  (ncelet, ntersl)!    !     !  on carrier phase                              !
! parbor           ! ra ! <-- ! particle interaction properties                !
!  (nfabor, nvisbr)!    !     !  on boundary faces                             !
! rdevel(nrdeve)   ! ra ! <-> ! real work array for temporary development      !
! rtuser(nrtuse)   ! ra ! <-- ! user-reserved real work array                  !
! ra(*)            ! ra ! --- ! main real work array                           !
!__________________!____!_____!________________________________________________!

!     Type: i (integer), r (real), s (string), a (array), l (logical),
!           and composite types (ex: ra real array)
!     mode: <-- input, --> output, <-> modifies data, --- work array
!=======================================

use connectivity
```

```
implicit none

!========================================
! Common blocks
!========================================

include "dimfbr.h"
include "paramx.h"
include "pointe.h"
include "numvar.h"
include "optcal.h"
include "cstphy.h"
include "cstnum.h"
include "entsor.h"
include "lagpar.h"
include "lagran.h"
include "parall.h"
include "period.h"
include "ppppar.h"
include "ppthch.h"
include "ppincl.h"

!========================================

! Arguments

integer          idbia0 , idbra0
integer          ndim   , ncelet , ncel   , nfac   , nfabor
integer          nfml   , nprfml
integer          nnod   , lndfac , lndfbr , ncelbr
integer          nvar   , nscal  , nphas
integer          nbpmax , nvp    , nvep   , nivep
integer          ntersl , nvlsta , nvisbr
integer          nideve , nrdeve , nituse , nrtuse

integer          ifacel(2,nfac) , ifabor(nfabor)
integer          ifmfbr(nfabor) , ifmcel(ncelet)
integer          iprfml(nfml,nprfml)
integer          maxelt, lstelt(maxelt)
integer          ipnfac(nfac+1), nodfac(lndfac)
integer          ipnfbr(nfabor+1), nodfbr(lndfbr)
integer          itepa(nbpmax,nivep)
integer          idevel(nideve), ituser(nituse)
integer          ia(*)

double precision xyzcen(ndim,ncelet)
double precision surfac(ndim,nfac), surfbo(ndim,nfabor)
double precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)
double precision xyznod(ndim,nnod), volume(ncelet)
double precision dt(ncelet), rtp(ncelet,*), rtpa(ncelet,*)
double precision propce(ncelet,*)
double precision propfa(nfac,*), propfb(ndimfb,*)
double precision coefa(ndimfb,*), coefb(ndimfb,*)
double precision ettp(nbpmax,nvp) , ettpa(nbpmax,nvp)
double precision tepa(nbpmax,nvep)
double precision statis(ncelet,nvlsta), stativ(ncelet,nvlsta-1)
double precision tslagr(ncelet,ntersl)
double precision parbor(nfabor,nvisbr)
double precision rdevel(nrdeve), rtuser(nrtuse), ra(*)

! Local variables


logical :: switch1,switch2
integer          idebia, idebra
integer          iel, iutile, iel1,iel2, i, j, ival,ival2,impout(6),ii
integer          ifac
integer(8)       con(ncel,6), ihuge
double precision a(3), b(3), c(3)

!####local variables for Level Set modelling
double precision  phi_nbck,phi_nbck2, phi_cenck, max_val,min_val,xpos1,xpos2,sec(3)
integer(8)        icen
integer           ifac2, n_of_f, jmax,jmin,isnbb,Number_Of_Faces,Number_Of_Faces2
logical           ifind, iswitch1,iswitch2
integer           nd_ix, nd_i, nd_k, nx, ni, nk

!================[Redistancing code variables]=================

integer          ival_nb,ival_nbNeg, ielt, nlelt2, k, xi,icount, N_i,MAXIT
double precision sign_val, dist_ptR(3),dif_pr(3),dif_xr(3),t_val,dif(3),tdiv,k_hat(3),scale_diff,scale_lengt
double precision v_hat(3), n_div, n_hat(3), dist1, dist2, dist3, dI, dImin, x(3), po(3), small_v, r(3)
double precision S_h,s_val, Sk, phi,phi_starr,x_n(3),po_n(3), delta_k, eta_k,eta_sum,xi_sum,fMin,t_small
double precision fl,fh,f,c_val,c_l,c_h,c_1,c_2,dc,swap,del,c_acc, s_k,C_const,ix_val
double precision dist_prR(3),dval,phi3,rdist
double precision value, psi_v
integer          ir,irV(4)
logical          Lr(4),error_iteration,ipos_checkdo
integer          I_it,ilim,icen2,icen3,n_faces

double precision  phi_val(9),rcen(3),Grad_phi(3),phi_max,phi_min,psi_value,del_A
integer           i_vertex
!================[Free surface modelling code variables]=================

integer          ionbb, i_group,iorder,n_group, n_stencil,id, itype, n_col,nbox,ntri
integer          ieln, ErrorFlag
integer          ichange,icheck,iprim_neg,isec_pos,isec_neg,iprim_pos
integer          ineg,inod2,iph_XI,inext2,ipos,prim_pos,ir0,ir1,ir2,isim_count,isimplxMax
integer          itype_max,isimpx,isampleNeg,Nneg,Npos,Nmax, iside
double precision phi_1, small_diff,t_s2_left(3),t_s2_right(3),p_j(3),theta_a,theta_b,theta_c
double precision n_hat1(3),n_hat2(3),n_hat3(3), a_right, a_left,t_o2_left(3), a_tswf_f2, theta_f2
double precision theta_s2, theta_o2,theta_1,theta_2,theta_3, a_tswf_s2, a_tswf_o2,lc_tswf_o2,lc_tswf_s2
double precision tswf_s2_1(3),tswf_s2_2(3),tswf_o2_1(3),tswf_o2_2(3),t_o2_right(3)
double precision tpj0(3),tpj1(3),tpj2(3),lb_value,lc_value, parallel_chk,tdiv2,tdiv1,tdiv0,lc_tswf_f2
double precision t_21(3),t_23(3),t_32(3),tswf_f2_1(3),tswf_f2_2(3),t_f2_right(3),t_f2_left(3), xval
double precision diff1(3), diff2(3), diff3(3),px0(3), propU, propV, propW, Uvel(3), propP
double precision prop_u(3),prop_v(3),prop_w(3),prop_val(3),xyz(3),vol1,vol2,kv(3),s1(3),s2(3)
double precision rlook1(3),rlook2(3), xvalue, yvalue, density,Area_k,deltaV_k,f2,v(3),sum_scale

real                              :: chk_x(100)
real, dimension(:,:), allocatable :: data_array
real, dimension(:,:), allocatable :: chk_pts
real, allocatable                 :: px(:)
real, dimension(:,:), allocatable :: a_matrx
logical          projection, ileft,ifond2,ilog,chk,ifond,ichk,idebug1,idebug2
double precision  h_s, xrtp,xrtp2,tide_start
double precision, parameter         :: small_value = 1.0d-6
type(simplex), pointer              :: current,previous

!========================================

print*,'start usproj'

!========================================
! 1.  Initialization
!========================================

! Memory management
idebia = idbia0
idebra = idbra0
ihuge  = 2.0e10
fMin  = 1.0d-6
c_acc  = 1.0d-6
MAXIT  = 20

if (isuite.eq.0) then !******isuite if block
    !================================
    do ii = 1, 1

        impout(ii) = impusr(ii)

    enddo
    open(impout(1),file='Tide_level_results.dat')
    ! print*,'nwb_counter = ',nwb_counter

    !=================================
    !=================================narrow band filter scheme========================
    !================================scalar one TEST=========================================
    
    !----------------------------------------------------------------------------------
    !         NEW RE-DISTANCING ROUTINE (START)
    !----------------------------------------------------------------------------------
    !====================================
    ! (E) Initial set up of Level set first and second neighbour cells surrounding isocontour S_h
    !    ----------------------------------------------
    !====================================
    !===========Setup the first neighbour cells surrounding S_h
    if ( (allocated(ptr_nwbElm)).and.(allocated(ptr_NegnwbElm)) ) then
        !---------------------------------------------------------------------------------
```

```
!        clear dynamic simplex memory at the end of run
!------------------------------------------------------------------------------------------
!-------------positive side memory purge--------------
do iel = 1, nwb_counter
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        do isimpx = 1, isimplxMax
            do while ( associated( current ) )
                previous => current
                current => current%next
                deallocate( previous )
            end do
        enddo   ! loop of isimplx
    enddo      ! loop of itype
enddo ! loop of iel
!-------------negative side memory purge--------------
do iel = 1, nwb_cnter_Neg
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        do isimpx = 1, isimplxMax
            do while ( associated( current ) )
                previous => current
                current => current%next
                deallocate( previous )
            end do
        enddo   ! loop of isimplx
    enddo      ! loop of itype
enddo ! loop of iel
!----------------------------------------------------------------------------------------------
```

```
!           clear dynamic simplex memory at the end of run (completed)
!-----------------------------------------------------------------------------------------------------------------
!===========Setup the first neighbour cells surrounding S_h
!-------------positive side sweep--------------
nwb_counter = 0
do iel = 1,ncel
    phi_cenck      = rtp(iel,isca(2))                                ! positive kth node
    Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
    ! ++++++++looking for field region where iso-surface S_h exists
    iprim_neg = 0
    iprim_pos = 0
    do ifac = 1,Number_of_Faces
        phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
        if ((phi_nbck.lt.0.0d0).and.(phi_cenck.gt.0.0d0)) then
            !-------check negative side---------
            if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                iprim_neg = iprim_neg + 1
            endif
        endif
    enddo
    if (iprim_neg > 0) then
        do ifac2 = 1, Number_of_Faces
            phi_nbck2  = rtp(nbcell(iel,ifac2),isca(2))
            if ( phi_nbck2.gt.0.0d0) then
                if (secondary_cell_select(nbcell(iel,ifac2)) <= 0) then
                    iprim_pos = iprim_pos + 1
                endif
            endif
        enddo
    endif
    !-------------------------------------------------------------------------------------------------------------
    !     Create a new set of temporary simplex tetrahedra surrounding a piece of the isosurf
    !-------------------------------------------------------------------------------------------------------------
    if ((iprim_neg.gt.0).and.(iprim_neg.lt.4).and.(phi_cenck.gt.0)) then
        nwb_counter   = nwb_counter + 1
        if (nwb_counter.gt.ihuge) then
            deallocate(ptr_nwbElm)
            stop 0
        endif
        ptr_nwbElm(nwb_counter)%nwb_index = iel
        !*********************
        iprim_neg = 0
        iprim_pos = 0
        isec_pos  = 0
        Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
        do ifac = 1, Number_of_Faces
            phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
            if ((phi_nbck.lt.0.0d0).and.(phi_cenck.gt.0.0d0)) then
                rtp(iel,isca(1)) = 40.0                              ! prim scalar1
                if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                    iprim_neg = iprim_neg + 1
                    ptr_nwbElm(nwb_counter)%Pneg(iprim_neg) = nbcell(iel,ifac)   ! negative p
                endif
            else if ( (phi_nbck.gt.0.0d0).and.(phi_cenck.gt.0.0d0) ) then
                if (secondary_cell_select(nbcell(iel,ifac)) == 1) then
                    isec_pos = isec_pos + 1
                    ptr_nwbElm(nwb_counter)%Spos(isec_pos) = nbcell(iel,ifac)   ! positive secondary
                    rtp(nbcell(iel,ifac),isca(1)) = 80.0              ! sec scalar1 +ve
                else
                    iprim_pos = iprim_pos + 1
                    ptr_nwbElm(nwb_counter)%Ppos(iprim_pos) = nbcell(iel,ifac)   ! positive primary
                    rtp(nbcell(iel,ifac),isca(1)) = 40.0         !  prim scalar1 +ve
                endif
            endif
        enddo
        !***********************NOTE NO NEGATIVE SECONDARY*********************
        ptr_nwbElm(nwb_counter)%neg       =  iprim_neg  ! total number of negative primary nodes
        ptr_nwbElm(nwb_counter)%pos       =  iprim_pos  ! total number of positive primary nodes wit
        ptr_nwbElm(nwb_counter)%Sec_pos   =  isec_pos   ! total number of positive secondary nodes
    endif
    !-----------------------------------------------------------------------------------------

enddo

!-------------negative side sweep--------------
nwb_cnter_Neg = 0
do iel = 1,ncel
    phi_cenck       = rtp(iel,isca(2))                               ! negative kth node
    Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
    ! ++++++++looking for field region where iso-surface S_h exists
    iprim_neg = 0
    iprim_pos = 0
    do ifac = 1,Number_of_Faces
        phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
        if ((phi_nbck.gt.0.0d0).and.(phi_cenck.lt.0.0d0)) then         !!!NOTE NOW phi_cenck negativ
            !-------check positive side---------
            if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                iprim_pos = iprim_pos + 1
            endif
        endif
    enddo
    if (iprim_pos > 0) then
        do ifac2 = 1, Number_of_Faces
            phi_nbck2  = rtp(nbcell(iel,ifac2),isca(2))
            if ( phi_nbck2.lt.0.0d0) then
                if (secondary_cell_select(nbcell(iel,ifac2)) <= 0) then
                    iprim_neg = iprim_neg + 1
                endif
            endif
        enddo
    endif
    !-----------------------------------------------------------------------------------------
    !     Create a new set of temporary simplex tetrahedra surrounding a piece of the isosurface
    !-----------------------------------------------------------------------------------------
    if ((iprim_pos.gt.0).and.(iprim_pos.lt.4).and.(phi_cenck.lt.0.0d0)) then
        nwb_cnter_Neg   = nwb_cnter_Neg + 1
```

```
            if (nwb_cnter_Neg.gt.ihuge) then
                deallocate(ptr_NegnwbElm)
                stop 0
            endif
            ptr_NegnwbElm(nwb_cnter_Neg)%nwb_index = iel
            !**********************
            iprim_neg = 0
            iprim_pos = 0
            isec_neg  = 0
            Number_Of_Faces=count(nbcell(iel,:)>-ihuge)
            do ifac = 1, Number_of_Faces
                phi_nbck  = rtp(nbcell(iel,ifac),isca(2))
                if ((phi_nbck.gt.0.0d0).and.(phi_cenck.lt.0.0d0)) then          !!!NOTE NOW phi_c
                    rtp(iel,isca(1)) = -40.0                                      ! prim scalar1
                    if (secondary_cell_select(nbcell(iel,ifac)) <= 0) then
                        iprim_pos = iprim_pos + 1
                        ptr_NegnwbElm(nwb_cnter_Neg)%Ppos(iprim_pos) = nbcell(iel,ifac)   ! posit
                    endif
                else if ( (phi_nbck.lt.0.0d0).and.(phi_cenck.lt.0.0d0) ) then
                    if (secondary_cell_select(nbcell(iel,ifac)) == 1) then
                        isec_neg = isec_neg + 1
                        ptr_NegnwbElm(nwb_cnter_Neg)%Sneg(isec_neg) = nbcell(iel,ifac)   ! negati
                        rtp(nbcell(iel,ifac),isca(1)) = -80.0              ! sec scalar1 -ve
                    else
                        iprim_neg = iprim_neg + 1
                        ptr_NegnwbElm(nwb_cnter_Neg)%Pneg(iprim_neg) = nbcell(iel,ifac)   ! negat
                        rtp(nbcell(iel,ifac),isca(1)) = -40.0         !  prim scalar1 -ve
                    endif
                endif
            enddo
            !*** NOTE NO POSITIVE SECONDARY ***
            ptr_NegnwbElm(nwb_cnter_Neg)%neg       =  iprim_neg  ! total number of negative prima
            ptr_NegnwbElm(nwb_cnter_Neg)%pos       =  iprim_pos  ! total number of positive prima
            ptr_NegnwbElm(nwb_cnter_Neg)%Sec_neg   =  isec_neg   ! total number of negative secon
        endif
        !-------------------------------------------------------------------------------------

    enddo !end of iel loop
    !-----------end of negative side sweep

    !=======================================
    !     Initial redistancing (START)
    !=======================================
    !**********************************
    !           Step 1:: re-Compute the exact distance to S_h
    !**********************************
    !-------------------------------------------------
    !---------positive side of free surface-----------
    !-------------------------------------------------
    iside = 0
    do iel = 1, nwb_counter
        ival_nb  = ptr_nwbElm(iel)%nwb_index
        Nneg = ptr_nwbElm(iel)%neg
```

```
Npos = ptr_nwbElm(iel)%pos
if (Nneg > 3) then
    itype_max = 3
else
    itype_max = Nneg
endif
call create_simplices( iside, ptr_nwbElm(iel), Nneg, Npos,itype_max)
do itype = 1, itype_max
    select case (itype)
    case (1)
        isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
        current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        Nmax = 3
    case (2)
        isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
        current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        Nmax = 4
    case (3)
        isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
        current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        Nmax = 3
    case default
        call error_message('Error1 in step1 with isimplxMax in USINV')
    end select

    do isimpx = 1, isimplxMax

        do inod2 = 1, Nmax
            if (itype == 1) then
                if ( inod2 == 1) then
                    ir   =  ival_nb                ! Kth node
                else
                    ir  =  current%sim_node(inod2)
                endif
                ieln =  current%sim_node(1)
            else if (itype == 2) then
                select case (inod2)
                case (1,2)
                    ieln  =  current%sim_node(inod2)
                    ir    =  ival_nb
                case (3,4)
                    ival = inod2 - 2
                    ieln  =  current%sim_node(ival)
                    ir    =  current%sim_node(3)
                case default
                    call error_message('Error2 in step1 with isimplxMax in USINV')
                end select
            else if (itype == 3) then
                ieln  =  current%sim_node(inod2)
                ir    =  ival_nb
            else
                call error_message('Error3 in step1 with isimplxMax in USINV')
            endif
```

```
                s_val   = rtp(ir,isca(2)) -  rtp(ieln,isca(2))
                !******** check if s_val is zero******
                call check_zero(s_val,'Error4 in USINV with Sk reconstruction at iel = ',iel)
                S_h  = - rtp(ieln,isca(2))/s_val
                if (S_h > 0.98) current%use_it = .false.
                a(1) = scale_length * (xyzcen(1,ir) )
                a(2) = scale_length * (xyzcen(2,ir) )
                a(3) = scale_length * (xyzcen(3,ir) )
                b(1) = scale_length * (xyzcen(1,ieln) )
                b(2) = scale_length * (xyzcen(2,ieln) )
                b(3) = scale_length * (xyzcen(3,ieln) )
                call position_vec(a, b, S_h, c)
                current%sh_nb(inod2,1) = c(1)
                current%sh_nb(inod2,2) = c(2)
                current%sh_nb(inod2,3) = c(3)
            enddo ! inod2 loop
            current => current%next
        enddo ! loop isimpx
    enddo ! loop itype
enddo ! iel loop


! ***check for zero volume simplices == (start)  ***
iside = 0
do iel = 1, nwb_counter
    ival_nb    =   ptr_nwbElm(iel)%nwb_index
    phi        =   1.0
    Npos = ptr_nwbElm(iel)%pos
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                               !new
                call Vol_k(.true. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                if (vol2 < tiny(1.0)) then
```

```
                    !print*,iel,'positive vol_k is = ',vol2,isimplxMax
                    current%use_it = .false.
                endif
            endif
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop

! ***check for zero volume simplices == (end)  ***


!====================Compute dI such that dI = min x belongs S_k|XI-x|===(start)
do iel = 1, nwb_counter
    ival_nb   = ptr_nwbElm(iel)%nwb_index
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    dImin = huge(1.0)     !***set d(Xn) = +infinity for n = 1,2,...Nnod_P
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            ir0   =  1        ! Kth node
            ir1   =  2
            ir2   =  3
            a(1) = scale_length * (xyzcen(1,ival_nb) )        ! k node
            a(2) = scale_length * (xyzcen(2,ival_nb) )
            a(3) = scale_length * (xyzcen(3,ival_nb) )
            kv(1) = current%sh_nb(ir0,1)      ! edge of free surface near kth node
            kv(2) = current%sh_nb(ir0,2)
            kv(3) = current%sh_nb(ir0,3)
            dif(1) = kv(1) - a(1)
            dif(2) = kv(2) - a(2)
            dif(3) = kv(3) - a(3)
            s1(1) = current%sh_nb(ir1,1) - kv(1)
            s1(2) = current%sh_nb(ir1,2) - kv(2)
            s1(3) = current%sh_nb(ir1,3) - kv(3)
            s2(1) = current%sh_nb(ir2,1) - kv(1)
            s2(2) = current%sh_nb(ir2,2) - kv(2)
            s2(3) = current%sh_nb(ir2,3) - kv(3)
            call cross_product(s1,s2,v)
            tdiv     = dabs(dot_product(v,v))
            tdiv1 =  dsqrt(tdiv)
            !if (current%use_it) then
            if (tdiv < small_value ) then
                !dist1 = rtp(ival_nb,isca(2))
                !if ( dabs(dImin) > dabs(dist1)) then
                !   dImin = dabs(dist1)
                !endif
                current => current%next
                cycle
            endif
            !endif
            v_hat(1) = v(1)/tdiv1
            v_hat(2) = v(2)/tdiv1
            v_hat(3) = v(3)/tdiv1
            dist1 = dabs(dot_product(dif, v_hat))
            if (dabs(dImin) > dabs(dist1)) then
                dImin = dabs(dist1)
            endif
            current => current%next
        enddo ! loop isimplx
        ! *** *** *** *** *** *** *** ***&
    enddo ! loop itype
    ptr_nwbElm(iel)%ph_star = dImin  !^^^^^^set phi*(Xn) = d(Xn) for n = 1,2,...Nnod_P ^^^^^^
enddo ! iel loop

!-------------------------------------------------
!---------negative side of free surface-----------
!-------------------------------------------------
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg  = ptr_NegnwbElm(iel)%nwb_index
    Nneg = ptr_NegnwbElm(iel)%neg
    Npos = ptr_NegnwbElm(iel)%pos
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    call create_simplices( iside, ptr_NegnwbElm(iel), Npos, Nneg,itype_max)
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
            Nmax = 3
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
            Nmax = 4
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
            Nmax = 3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select

        do isimpx = 1, isimplxMax

            do inod2 = 1, Nmax
                if (itype == 1) then
                    if ( inod2 == 1) then
                        ir   =  ival_nbNeg                ! Kth node
                    else
                        ir  =  current%sim_node(inod2)
                    endif
                    ieln =  current%sim_node(1)
                else if (itype == 2) then
                    select case (inod2)
                    case (1,2)
                        ieln  =  current%sim_node(inod2)
                        ir   =  ival_nbNeg
                    case (3,4)
                        ival = inod2 - 2
                        ieln  =  current%sim_node(ival)
                        ir   =  current%sim_node(3)
                    case default
                        call error_message('Error2 in step1 with isimplxMax in USINV')
                    end select
                else if (itype == 3) then
                    ieln  =  current%sim_node(inod2)
                    ir    =  ival_nbNeg
                else
                    call error_message('Error3 in step1 with isimplxMax in USINV')
                endif
                s_val   = rtp(ir,isca(2)) -  rtp(ieln,isca(2))
                !******** check if s_val is zero******
                call check_zero(s_val,'Error4 in USINV with Sk reconstruction at iel = ',iel)
                S_h  = - rtp(ieln,isca(2))/s_val
                if (S_h > 0.98) current%use_it = .false.                                !new
                a(1) = scale_length * (xyzcen(1,ir) )
                a(2) = scale_length * (xyzcen(2,ir) )
                a(3) = scale_length * (xyzcen(3,ir) )
                b(1) = scale_length * (xyzcen(1,ieln) )
                b(2) = scale_length * (xyzcen(2,ieln) )
                b(3) = scale_length * (xyzcen(3,ieln) )
                call position_vec(a, b, S_h, c)
                current%sh_nb(inod2,1) = c(1)
                current%sh_nb(inod2,2) = c(2)
                current%sh_nb(inod2,3) = c(3)
```

```
            enddo ! inod2 loop
            current => current%next
        enddo ! loop isimpx
    enddo ! loop itype
enddo ! iel loop


! ***check for zero volume simplices == (start)  ***
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    phi           =   1.0
    Npos = ptr_NegnwbElm(iel)%pos
    Nneg = ptr_NegnwbElm(iel)%neg
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                              !new
                call Vol_k(.true. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
                if (vol2 < tiny(1.0)) then
                    !print*,iel,'Negative vol_k is = ',vol2,isimplxMax
                    current%use_it = .false.
                endif
            endif
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop
! ***check for zero volume simplices == (end)  ***^^^^^


!===================Compute dI such that dI = min x belongs S_k|XI-x|===(start)
do iel = 1, nwb_cnter_Neg
    ival_nbNeg   = ptr_NegnwbElm(iel)%nwb_index
```

```
Npos = ptr_NegnwbElm(iel)%pos
if (Npos > 3) then
    itype_max = 3
else
    itype_max = Npos
endif
dImin = huge(1.0)     !***set d(Xn) = +infinity for n = 1,2,...Nnod_P
do itype = 1, itype_max
    select case (itype)
    case (1)
        isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
        current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
    case (2)
        isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
        current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
    case (3)
        isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
        current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
    case default
        call error_message('Error1 in step1 with isimplxMax in USINV')
    end select
    !  *** scan for isimplxMax per simplex cut type considered  ***
    do isimpx = 1, isimplxMax
        ir0   =  1        ! Kth node
        ir1   =  2
        ir2   =  3
        a(1) = scale_length * (xyzcen(1,ival_nbNeg) )        ! k node
        a(2) = scale_length * (xyzcen(2,ival_nbNeg) )
        a(3) = scale_length * (xyzcen(3,ival_nbNeg) )
        kv(1) = current%sh_nb(ir0,1)      ! edge of free surface near kth node
        kv(2) = current%sh_nb(ir0,2)
        kv(3) = current%sh_nb(ir0,3)
        dif(1) = kv(1) - a(1)
        dif(2) = kv(2) - a(2)
        dif(3) = kv(3) - a(3)
        s1(1) = current%sh_nb(ir1,1) - kv(1)
        s1(2) = current%sh_nb(ir1,2) - kv(2)
        s1(3) = current%sh_nb(ir1,3) - kv(3)
        s2(1) = current%sh_nb(ir2,1) - kv(1)
        s2(2) = current%sh_nb(ir2,2) - kv(2)
        s2(3) = current%sh_nb(ir2,3) - kv(3)
        call cross_product(s1,s2,v)
        tdiv     = dabs(dot_product(v,v))
        tdiv1 =  dsqrt(tdiv)
        !if (current%use_it) then
        if (tdiv < small_value ) then
            !dist1 = - rtp(ival_nbNeg,isca(2))
            !if ((dabs(dImin) > dabs(dist1))) then
            !   dImin = dabs(dist1)
            !endif
            current => current%next
            cycle
        endif
```

```
            !endif
            v_hat(1) = v(1)/tdiv1
            v_hat(2) = v(2)/tdiv1
            v_hat(3) = v(3)/tdiv1
            dist1 = dabs(dot_product(dif, v_hat))
            if (dabs(dImin) > dabs(dist1)) then
                dImin = dabs(dist1)
            endif
            current => current%next
        enddo ! loop isimplx
        ! *** *** *** *** *** *** *** ***&
    enddo ! loop itype
    ptr_NegnwbElm(iel)%ph_star = - dImin  !^^^^^^set phi*(Xn) = d(Xn) for n = 1,2,...Nnod_P ^
enddo ! iel loop

!===================Compute dI such that dI = min x belongs S_k|XI-x|====(end)
!***********************************
!           Step 2:: Find eta_h, a piecewise constant function
!***********************************
! Find eta_h, a piecewise constant function (simplex wise mass correction) == (start)
!---------positive side of free surface-----------
iside = 0
do iel = 1, nwb_counter
    ival_nb    =   ptr_nwbElm(iel)%nwb_index
    phi        =   rtp(ival_nb,isca(2))
    phi_starr  =   ptr_nwbElm(iel)%ph_star
    eta_k      =   phi - phi_starr
    ! if ( ntcabs == 17) then
    !   print*,phi,phi_starr
    ! endif
    Npos = ptr_nwbElm(iel)%pos
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax simplices per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
```

```
            call Vol_k(.false. ,iside,current,ival_nb,itype,phi,0.0d0,Area_K,vol2)
            current%vol_phi = vol2
            call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,eta_k,Area_K,vol1)
            deltaV_k = vol2 - vol1
            icount   = 0
            do
                if ((dabs(deltaV_k) < 1.0d-8).or.(icount > 20)) exit
                icount = icount + 1
                if ( Area_K < tiny(1.0) ) then
                    exit
                endif
                eta_k =  -3.0 * deltaV_k/Area_K    ! Changed
                call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,eta_k,Area_K,vol1)
                deltaV_k = vol2 - vol1
            enddo
            if (current%use_it) then                              !new
                current%eta = eta_k
            else
                current%eta = 0.0
            endif                                                 !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop

!---------negative side of free surface-----------
iside = 1
do iel = 1, nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    phi        =   rtp(ival_nbNeg,isca(2))
    phi_starr  =   ptr_NegnwbElm(iel)%ph_star
    eta_k      =   phi - phi_starr
    Npos = ptr_NegnwbElm(iel)%pos
    Nneg = ptr_NegnwbElm(iel)%neg
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi,0.0d0,Area_K,vol2)
            current%vol_phi = vol2
            call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi_starr,eta_k,Area_K,vol1)
            deltaV_k = vol2 - vol1
            icount   = 0
            do
                if ((dabs(deltaV_k) < 1.0d-8).or.(icount > 20)) exit
                icount = icount + 1
                if ( Area_K < tiny(1.0) ) then
                    exit
                endif
                eta_k =  -3.0 * deltaV_k/Area_K    ! Changed
                call Vol_k(.false. ,iside,current,ival_nbNeg,itype,phi_starr,eta_k,Area_K,vol
                deltaV_k = vol2 - vol1
            enddo
            if (current%use_it) then                              !new
                current%eta = eta_k
            else
                current%eta = 0.0
            endif                                                !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
enddo  ! iel loop

! Find eta_h, a piecewise constant function (simplex wise mass correction) == (end)
!***********************************
!           Step 3:: Find Xi_h, the ortogonal projection of eta_h
!***********************************
! Find Xi_h (node wise mass correction) == (start)
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    xi_sum = 0.0
    Nneg = ptr_nwbElm(iel)%neg
    if (Nneg > 3) then
        itype_max = 3
    else
        itype_max = Nneg
    endif
    isim_count = 0
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_nwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                              !new
                isim_count = isim_count  + 1
                xi_sum = xi_sum + current%eta
            endif                                                 !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
    if (isim_count == 0) then
        isim_count = 1
        !print*,iel,' pos xi_sum = ', xi_sum
    endif
    ptr_nwbElm(iel)%xi_h = xi_sum/real(isim_count)
enddo

!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    xi_sum = 0.0
    Npos = ptr_NegnwbElm(iel)%pos
    if (Npos > 3) then
        itype_max = 3
    else
        itype_max = Npos
    endif
    isim_count = 0
    do itype = 1, itype_max
        select case (itype)
        case (1)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
        case (2)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
        case (3)
            isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
            current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        do isimpx = 1, isimplxMax
            if (current%use_it) then                              !new
                isim_count = isim_count  + 1
                xi_sum = xi_sum + current%eta
            endif                                               !new
            current => current%next
        enddo ! isimpx loop
        ! *** *** *** *** *** *** *** ***
    enddo    ! itype loop
    if (isim_count == 0) then
        isim_count = 1
        ! print*,iel,' neg xi_sum = ', xi_sum
    endif
    ptr_NegnwbElm(iel)%xi_h = xi_sum/real(isim_count)
enddo
! Find Xi_h (node wise mass correction) == (end)
!************************************
!           Step 4(i):: Find psi_h = C xi_h
!************************************
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    error_iteration = .true.
    c_1     =  0.0
    c_2     =  scale_length !1.0
    fl      =  deltaV(iside,ptr_nwbElm(iel),c_1)
    fh      =  deltaV(iside,ptr_nwbElm(iel),c_2)
    if ( (fh * fl) > 0.0) then
        c_2     =  -scale_length !1.0
        fh      =  deltaV(iside,ptr_nwbElm(iel),c_2)
    endif
    if ( ( (fl * fh) > 0.0).and.(dabs(fl) > fMin).and.(dabs(fh) > fMin)) then
        print*,'Error at iel = ',iel, 'root must be bracketed between arguments'
        stop
    else if ((dabs(fl) < fMin).and.(dabs(fh) < fMin)) then
        error_iteration = .false.
        ptr_nwbElm(iel)%cval = 0.0
        continue
    endif
    if ( fl  > 0.0 ) then
        c_l = c_1
        c_h = c_2
    else
        c_l = c_2
        c_h = c_1
        swap = fl
        fl = fh
        fh = swap
    endif
    dc = c_h - c_l
    do j = 1, MAXIT
        if (dabs(fl - fh) < tiny(1.0)) then
            error_iteration = .false.
            exit
        endif
        c_val = c_l + dc * fl/(fl - fh)
        f = deltaV(iside,ptr_nwbElm(iel),c_val)
        if (f < 0.0) then
            del = c_l - c_val
            c_l = c_val
            fl = f
        else
            del = c_h - c_val
            c_h = c_val
            fh  = f
        endif
        dc  = c_h - c_l
        ptr_nwbElm(iel)%fval = f
        ptr_nwbElm(iel)%cval = c_val
        if ( (dabs(del) < c_acc).or.(dabs(f) < tiny(1.0)) ) then
            error_iteration = .false.
            exit
        endif
    enddo ! j loop
    if (error_iteration) then
        print*,'Maximum number of iteration exceeded at iel = ', iel
        stop
    endif
enddo  ! iel loop
!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    error_iteration = .true.
    c_1     =  0.0
    c_2     =  scale_length !1.0
    fl      =  deltaV(iside,ptr_NegnwbElm(iel),c_1)
    fh      =  deltaV(iside,ptr_NegnwbElm(iel),c_2)
    if ( (fh * fl) > 0.0) then
        c_2     =  -scale_length !1.0
        fh      =  deltaV(iside,ptr_NegnwbElm(iel),c_2)
    endif
    if ( ( (fl * fh) > 0.0).and.(dabs(fl) > fMin).and.(dabs(fh) > fMin)) then
        print*,'Error at iel = ',iel, 'root must be bracketed between arguments'
        stop
    else if ((dabs(fl) < fMin).and.(dabs(fh) < fMin)) then
        error_iteration = .false.
        ptr_NegnwbElm(iel)%cval = 0.0
        continue
    endif
    if ( fl  > 0.0 ) then
        c_l = c_1
        c_h = c_2
    else
        c_l = c_2
        c_h = c_1
        swap = fl
        fl = fh
        fh = swap
    endif
    dc = c_h - c_l
    do j = 1, MAXIT
        if (dabs(fl - fh) < tiny(1.0)) then
            error_iteration = .false.
            exit
        endif
        c_val = c_l + dc * fl/(fl - fh)
        f = deltaV(iside,ptr_NegnwbElm(iel),c_val)
        if (f < 0.0) then
            del = c_l - c_val
            c_l = c_val
            fl = f
        else
            del = c_h - c_val
            c_h = c_val
            fh  = f
        endif
        dc  = c_h - c_l
        ptr_NegnwbElm(iel)%fval = f
        ptr_NegnwbElm(iel)%cval = c_val
        if ( (dabs(del) < c_acc).or.(dabs(f) < tiny(1.0)) ) then
            error_iteration = .false.
            exit
        endif
    enddo ! j loop
    if (error_iteration) then
        print*,'Maximum number of iteration exceeded at iel = ', iel
        stop
    endif
enddo  ! iel loop
! Find C  == (start)
!***********************************
!           Step 4(ii):: Redistance the Level set first cells surrounding isocontour S_h
!***********************************
!---------positive side of free surface-----------
iside = 0
do iel = 1,nwb_counter
    ival_nb    =   ptr_nwbElm(iel)%nwb_index
    ix_val           = ptr_nwbElm(iel)%xi_h
    C_const          = ptr_nwbElm(iel)%cval
    phi_starr        = ptr_nwbElm(iel)%ph_star
    !if ( ntcabs == 19) then
    ! print*,iel,rtp(ival_nb,isca(2)),phi_starr,C_const,ix_val
    !endif
    rtp(ival_nb,isca(2)) = phi_starr + ( C_const * ix_val )  !NOTE MOST IMPORTANT PART which
enddo  ! iel loop
!---------negative side of free surface-----------
iside = 1
do iel = 1,nwb_cnter_Neg
    ival_nbNeg    =   ptr_NegnwbElm(iel)%nwb_index
    ix_val           = ptr_NegnwbElm(iel)%xi_h
    C_const          = ptr_NegnwbElm(iel)%cval
    phi_starr        = ptr_NegnwbElm(iel)%ph_star
    !print*,iel,rtp(ival_nbNeg,isca(2)),phi_starr,C_const,ix_val
    rtp(ival_nbNeg,isca(2)) = phi_starr + ( C_const * ix_val )  !NOTE MOST IMPORTANT PART which upda
enddo  ! iel loop
!**********************************
!          Step 5:: Edge distance approximation
!**********************************
! -----------update available secondary nodes on positive side of isosurface--------
iside = 0
ichange = 1
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_counter
        Npos = ptr_nwbElm(iel)%pos
        isnbb = ptr_nwbElm(iel)%Sec_pos
        do i = 1, isnbb
            dImin    = huge(1)
            iph_XI = ptr_nwbElm(iel)%Spos(i)
            a(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
            a(2)   = scale_length * (xyzcen(2,iph_XI))
            a(3)   = scale_length * (xyzcen(3,iph_XI))
            ! |XJ - XI| edge distance approx considered of Npos of them
            do ipos = 1,Npos
                icen = ptr_nwbElm(iel)%Ppos(ipos)
                b(1)   = scale_length * (xyzcen(1,icen))  - a(1)
                b(2)   = scale_length * (xyzcen(2,icen))  - a(2)
                b(3)   = scale_length * (xyzcen(3,icen))  - a(3)
                dI     = rtp(icen,isca(2)) + dsqrt( dabs(dot_product(b,b)) )
                if ( dabs(dImin) > dabs(dI)) then
                    dImin = dI
                endif
            enddo !=======(Find minimum phi_h(X_I) ....end)
            if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                rtp(iph_XI,isca(2)) = dImin  !Edge distance approximation of phi at XI secondary nod
                ichange = 1
            endif
        enddo
    enddo
enddo

! -----------update available secondary nodes on negative side of isosurface--------
iside = 1
ichange = 1
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_cnter_Neg
        Nneg = ptr_NegnwbElm(iel)%neg
        isnbb = ptr_NegnwbElm(iel)%Sec_neg
        do i = 1, isnbb
            dImin    = huge(1.0)
            iph_XI = ptr_NegnwbElm(iel)%Sneg(i)
            a(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
            a(2)   = scale_length * (xyzcen(2,iph_XI))
            a(3)   = scale_length * (xyzcen(3,iph_XI))
            ! |XJ - XI| edge distance approx considered of Npos of them
            do ineg = 1,Nneg
                icen = ptr_NegnwbElm(iel)%Pneg(ineg)
                b(1)   = scale_length * (xyzcen(1,icen))  - a(1)
                b(2)   = scale_length * (xyzcen(2,icen))  - a(2)
                b(3)   = scale_length * (xyzcen(3,icen))  - a(3)
                dI     = rtp(icen,isca(2)) - dsqrt( dabs(dot_product(b,b)) )
                if ( dabs(dImin) > dabs(dI)) then
                    dImin = dI
                endif
            enddo !=======(Find minimum phi_h(X_I) ....end)
            if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                rtp(iph_XI,isca(2)) = -dImin  !Edge distance approximation of phi at XI secon
                ichange = 1
            endif
        enddo
    enddo
enddo
!-------------------------------------------------------------------------------------
!***********************************
!           Step 6:: Shadow distance correction
!***********************************
! -----------update available secondary nodes on positive side of isosurface--------
iside = 1
ichange = 0
do
    if (ichange == 0) exit
    ichange = 0
    do iel = 1, nwb_cnter_Neg
        Npos = ptr_NegnwbElm(iel)%pos
        ival_nbNeg    = ptr_NegnwbElm(iel)%nwb_index
        if (Npos > 3) then
            itype_max = 3
        else
            itype_max = Npos
        endif
        !  *** scan for isimplxMax simplices per simplex cut type considered
        do itype = 1, itype_max
            isnbb = ptr_NegnwbElm(iel)%Sec_neg
            do i = 1, isnbb
                dImin    = huge(1)
                iph_XI = ptr_NegnwbElm(iel)%Sneg(i)
                sec(1)   = scale_length * (xyzcen(1,iph_XI))  ! XI secondary node
                sec(2)   = scale_length * (xyzcen(2,iph_XI))
                sec(3)   = scale_length * (xyzcen(3,iph_XI))
                ! scan positive faces of simplex
                select case (itype)
                case (1)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
                case (2)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
                case (3)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    if (itype == 1) then
                        ! --------face 1
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .true. ; irV(2)  = current%sim_node(3)
                        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! ---------face 2
                        ntri = 4
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 1
                        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
                        Lr(4) = .false. ; irV(4)  = 2
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! --------face 3
                        ntri = 4
                        Lr(1) = .true. ; irV(1)  = current%sim_node(3)
                        Lr(2) = .false. ; irV(2)  = 3
                        Lr(3) = .true. ; irV(3)  = current%sim_node(2)
                        Lr(4) = .false. ; irV(4)  = 2
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                        ! --------face 4
                        ntri = 4
                        Lr(1) = .true.  ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 1
                        Lr(3) = .true. ; irV(3)  = current%sim_node(3)
                        Lr(4) = .false. ; irV(4)  = 3
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                    else if (itype == 2) then
                        ! ---------face 1
                        ntri = 4
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .true. ; irV(2)  = current%sim_node(3)
```

```
            Lr(3) = .false. ; irV(3)  = 2
            Lr(4) = .false. ; irV(4)  = 4
            call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
            if ( dabs(dImin) > dabs(dI)) then
                dImin = dI
            endif
            ! ---------face 2
            ntri = 3
            Lr(1) = .true. ; irV(1)  = ival_nbNeg
            Lr(2) = .false. ; irV(2)  = 2
            Lr(3) = .false. ; irV(3)  = 1
            call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
            if ( dabs(dImin) > dabs(dI)) then
                dImin = dI
            endif
            ! ---------face 3
            ntri = 3
            Lr(1) = .true. ; irV(1)  = current%sim_node(3)
            Lr(2) = .false. ; irV(2)  = 3
            Lr(3) = .false. ; irV(3)  = 4
            call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
            if ( dabs(dImin) > dabs(dI)) then
                dImin = dI
            endif
            ! ---------face 4
            ntri = 4
            Lr(1) = .true.  ; irV(1)  = ival_nbNeg
            Lr(2) = .false. ; irV(2)  = 1
            Lr(3) = .true. ; irV(3)  = current%sim_node(3)
            Lr(4) = .false. ; irV(4)  = 3
            call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
            if ( dabs(dImin) > dabs(dI)) then
                dImin = dI
            endif
        else if (itype == 3) then
            ! --------face 1
            ntri = 3
            Lr(1) = .false. ; irV(1)  = 1
            Lr(2) = .true. ; irV(2)   = ival_nbNeg
            Lr(3) = .false. ; irV(3)  = 2
            call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
            if ( dabs(dImin) > dabs(dI)) then
                dImin = dI
            endif
            ! ---------face 2
            ntri = 3
            Lr(1) = .true. ; irV(1)  = ival_nbNeg
            Lr(2) = .false. ; irV(2)  = 1
            Lr(3) = .false. ; irV(3)  = 3
            call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
            if ( dabs(dImin) > dabs(dI)) then
                dImin = dI
            endif
```

```
                        ! --------face 3
                        ntri = 3
                        Lr(1) = .true. ; irV(1)  = ival_nbNeg
                        Lr(2) = .false. ; irV(2)  = 3
                        Lr(3) = .false. ; irV(3)  = 2
                        call find_min_dist(current,ntri,Lr,irV,sec,-1,dI)
                        if ( dabs(dImin) > dabs(dI)) then
                            dImin = dI
                        endif
                    endif
                    current => current%next
                enddo !---end of isimplx loop
                if (dabs(rtp(iph_XI,isca(2))) > dabs(dImin)) then
                    rtp(iph_XI,isca(2)) = -dImin  !shaddow distance approximation of phi at XI second
                    ichange = 1
                endif
            enddo !---end of isnbb secondary nodes loop
        enddo !---end of itype loop
    enddo !---end of nwb_cnter_Neg loop
enddo !---ichange  loop !=======(Find minimum phi_h(X_I) ....end)
!-------------------------------------------------------------------------------------

!=======================================
!      redistancing (END)
!======================================

if ( ntcabs.ge.ntmabs) then
    !------- clear dynamic memory at the end of run----------------------------------
    !---------positive side of free surface-----------
    do iel = 1, nwb_counter
        do itype = 1, itype_max
            select case (itype)
            case (1)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type1%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type1
            case (2)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type2%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type2
            case (3)
                isimplxMax = ptr_nwbElm(iel)%ptr_cutTetra_type3%sim_count
                current => ptr_nwbElm(iel)%ptr_cutTetra_type3
            case default
                call error_message('Error1 in step1 with isimplxMax in USINV')
            end select
            do isimpx = 1, isimplxMax
                do while ( associated( current ) )
                    previous => current
                    current => current%next
                    deallocate( previous )
                end do
            enddo   ! loop of isimplx
        enddo      ! loop of itype
    enddo ! loop of iel
```

```
        deallocate(ptr_nwbElm)
        !---------negative side of free surface-----------
        do iel = 1, nwb_cnter_Neg
            do itype = 1, itype_max
                select case (itype)
                case (1)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type1%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type1
                case (2)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type2%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type2
                case (3)
                    isimplxMax = ptr_NegnwbElm(iel)%ptr_cutTetra_type3%sim_count
                    current => ptr_NegnwbElm(iel)%ptr_cutTetra_type3
                case default
                    call error_message('Error1 in step1 with isimplxMax in USINV')
                end select
                do isimpx = 1, isimplxMax
                    do while ( associated( current ) )
                        previous => current
                        current => current%next
                        deallocate( previous )
                    end do
                enddo   ! loop of isimplx
            enddo      ! loop of itype
        enddo ! loop of iel
        deallocate(ptr_NegnwbElm)

        !------- clear dynamic memory at the end of run (completed)---------------------------
        print*,'final time step = ',ntcabs
    else
        !if (irangp.le.0) then
        if (mod(ntcabs,1) == 0) then
            print*,'time step = ',ntcabs
        endif
        ! endif
    endif
endif  !end of if "(allocated(ptr_nwbElm))..."  block
!-------------------------------------------------------------------------------------------
!         NEW RE-DISTANCING ROUTINE (end)
!-------------------------------------------------------------------------------------------
! --------------------------------------------------
! Close files at final time step
! --------------------------------------------------

if (ntcabs.eq.ntmabs) then

    !if (irangp.le.0) then

    do ii = 1, 1

        close(impout(ii))
```

```
          enddo

          !endif

      endif
      ! ------------------------------------------------
      ! Close files at final time step
      ! ------------------------------------------------

  endif !***********isuite if block
  return

contains

  !***********************
  !
  subroutine find_min_dist(cur,ntri,Lr,ir,sec,iside,dImin)

     use connectivity
     implicit none

     ! code saturne variables
     !integer          ndim   , ncelet
     !double precision rtp(ncelet,*)

     type(simplex), pointer :: cur
     double precision sec(3),dImin
     integer          ntri, iside,ir(4)
     logical          Lr(4)

     ! local variables
     double precision, parameter         ::  small_value = 1.0d-6
     double precision, parameter         ::  th_d = 0.3333333333333
     double precision  k(3),a(3),s1(3),b(3),c(3),s2(3),dif(3),phi_val(4),d(3)
     double precision  v_hat(3),val,tdiv1,tdiv,a_matrix(4,4),v(3),phi_interp
     real  p_x(3),simVec(4,3)

     !------------positive side of free surface---------------------
     if (Lr(1) ) then
         k(1)   = scale_length * (xyzcen(1,ir(1) ) )  ! kth node
         k(2)   = scale_length * (xyzcen(2,ir(1) ) )
         k(3)   = scale_length * (xyzcen(3,ir(1) ) )
     else
         k(1)   = cur%sh_nb(ir(1),1)
         k(2)   = cur%sh_nb(ir(1),2)
         k(3)   = cur%sh_nb(ir(1),3)
     endif
     if (Lr(2) ) then
         s1(1) = scale_length * (xyzcen(1,ir(2)) ) - k(1)
         s1(2) = scale_length * (xyzcen(2,ir(2)) ) - k(2)
         s1(3) = scale_length * (xyzcen(3,ir(2)) ) - k(3)
     else
         s1(1) = cur%sh_nb(ir(2),1) - k(1)
```

```
    s1(2) = cur%sh_nb(ir(2),2) - k(2)
    s1(3) = cur%sh_nb(ir(2),3) - k(3)
endif
if (Lr(3) ) then
    s2(1) = scale_length * (xyzcen(1,ir(3) ) ) - k(1)
    s2(2) = scale_length * (xyzcen(2,ir(3) ) ) - k(2)
    s2(3) = scale_length * (xyzcen(3,ir(3) ) ) - k(3)
else
    s2(1) = cur%sh_nb(ir(3),1) - k(1)
    s2(2) = cur%sh_nb(ir(3),2) - k(2)
    s2(3) = cur%sh_nb(ir(3),3) - k(3)
endif
if ( ntri == 4) then
    if (Lr(4) ) then
        d(1)   = scale_length * (xyzcen(1,ir(4) ) )
        d(2)   = scale_length * (xyzcen(2,ir(4) ) )
        d(3)   = scale_length * (xyzcen(3,ir(4) ) )
    else
        d(1)   = cur%sh_nb(ir(4),1)
        d(2)   = cur%sh_nb(ir(4),2)
        d(3)   = cur%sh_nb(ir(4),3)
    endif
endif
call cross_product(s1,s2,v)
a     =  s1 + k
b     =  s2 + k
if ( ntri == 3) then
    c     = (a + b + k) * th_d - sec
else if ( ntri == 4) then
    c     = (a + b + d + k) * 0.25 - sec
endif
val   = dot_product(c,v)
if (val < 0.0d0) call cross_product(s2,s1,v)
tdiv     = dabs(dot_product(v,v))
if (tdiv < small_value ) return
tdiv1    =  dsqrt(tdiv)
v_hat(1) = v(1)/tdiv1
v_hat(2) = v(2)/tdiv1
v_hat(3) = v(3)/tdiv1
dif      = k - sec
val = dot_product(dif,v_hat)
! establish point of intersection p_x on face of simplex
call create_vec(sec, v_hat, val, p_x)
if ( ntri == 3) then
    d = 0.5 * (a + b)
endif
call convert_Vec(4,k,a,d,b,simVec)
if (check_simplex(ntri,simVec,p_x) ) then
    call calc_invert_matrix(simVec,a_matrx,4)
    phi_val(1) = rtp(ir(1), isca(2) )
    phi_val(2) = rtp(ir(2), isca(2) )
    phi_val(4) = rtp(ir(3), isca(2) )
    if ( ntri == 3) then
```

```
            phi_val(3) = 0.5 * (phi_val(2) + phi_val(4))
        else if ( ntri == 4) then
            phi_val(3) = rtp(ir(4), isca(2) )
        endif
        phi_interp = calc_phi_interp(4,phi_val,a_matrx,p_x)
        b          =  p_x - sec
        if (iside == 1 ) then
            dImin = phi_interp + sqrt( dabs(dot_product(b,b)) )
        else
            dImin = phi_interp - sqrt( dabs(dot_product(b,b)) )
        endif
    endif
    return
end subroutine find_min_dist

!************************
!  calculate interpolated level set value from neighbour level set values
!  using 3D geometrically isotropic trilinear interpolation
!************************
double precision function calc_phi_interp(ntri,prop_neighbs,a_matrx,p_x)
   implicit none
   real, dimension(:,:)                             :: a_matrx
   real, dimension(:)                               :: p_x
   double precision, dimension(:)                   :: prop_neighbs
   double precision,dimension(:), allocatable       :: c
   double precision                                 :: prop_sum
   integer                                          :: i, j, ntri

   allocate(c(ntri))
   do i = 1, ntri
       c(i) = 0.0
       do j = 1, ntri
           c(i) = c(i) + a_matrx(i,j) * prop_neighbs(i)
       enddo
   enddo
   prop_sum = 0.0
   do i = 1, ntri
       if (i == 1) then
           prop_sum = prop_sum + c(i)
       else
           prop_sum = prop_sum + c(i) * p_x(i-1)
       endif
   enddo
   calc_phi_interp = prop_sum
   deallocate(c)
   return
end function calc_phi_interp

!******************************
!  create_simplices
!
!  This subroutine creates the simplices associated with node k and its neighbour
!              cells forming part of the free surface region near node k
```

```
!*******************************
subroutine create_simplices(iside,ptr, Nneg, Npos,itype_max)
   use connectivity
   implicit none

   type (narrow_band_element)        :: ptr
   integer                           :: Nneg, Npos, itype_max,iside

   ! local variables
   type(simplex), pointer           :: current,previous
   integer itype,isimplex, inext, isamplePos,p(3), icount

   do itype = 1, itype_max
       if (itype == 1) then
           nullify(ptr%ptr_cutTetra_type1)
           allocate(ptr%ptr_cutTetra_type1)
           current => ptr%ptr_cutTetra_type1
           current%use_it = .true.
           isimplex = 0
           do isampleNeg = 1, Nneg
               if (iside == 0) then       ! positive side
                   p(1)     = ptr%Pneg(isampleNeg)
               else if (iside == 1) then  ! negative side
                   p(1)     = ptr%Ppos(isampleNeg)
               endif
               inext = 1
               do isamplePos = 1, Npos
                   if (iside == 0) then      ! positive side
                       p(2) = ptr%Ppos(isamplePos)
                   else if (iside == 1) then ! negative side
                       p(2) = ptr%Pneg(isamplePos)
                   endif
                   inext = inext + 1
                   if (inext == Npos) inext = 1
                   if (iside == 0) then      ! positive side
                       p(3) = ptr%Ppos(inext)
                   else if (iside == 1) then ! negative side
                       p(3) = ptr%Pneg(inext)
                   endif
                   isimplex = isimplex + 1
                   current%sim_node(1) = p(1)
                   current%sim_node(2) = p(2)
                   current%sim_node(3) = p(3)
                   allocate(current%next)
                   nullify( current%next%next )
                   current => current%next
                   current%use_it = .true.
               enddo
           enddo
           ptr%ptr_cutTetra_type1%sim_count = isimplex
       else if (itype == 2) then
           nullify(ptr%ptr_cutTetra_type2)
           allocate(ptr%ptr_cutTetra_type2)
```

```
    current => ptr%ptr_cutTetra_type2
    current%use_it = .true.
    isimplex = 0
    inext = 1
    do isampleNeg = 1, Nneg
        if (iside == 0) then       ! positive side
            p(1)     = ptr%Pneg(isampleNeg)
        else if (iside == 1) then  ! negative side
            p(1)     = ptr%Ppos(isampleNeg)
        endif
        inext    = inext + 1
        if (inext == Nneg) inext = 1
        if (iside == 0) then       ! positive side
            p(2)     =   ptr%Pneg(inext)
        else if (iside == 1) then  ! negative side
            p(2)     =   ptr%Ppos(inext)
        endif
        do isamplePos = 1, Npos
            if (iside == 0) then      ! positive side
                p(3) = ptr%Ppos(isamplePos)
            else if (iside == 1) then ! negative side
                p(3) = ptr%Pneg(isamplePos)
            endif
            isimplex = isimplex + 1
            current%sim_node(1) = p(1)
            current%sim_node(2) = p(2)
            current%sim_node(3) = p(3)
            allocate(current%next)
            nullify( current%next%next )
            current => current%next
            current%use_it = .true.
        enddo
    enddo
    ptr%ptr_cutTetra_type2%sim_count = isimplex
else if (itype == 3) then
    nullify(ptr%ptr_cutTetra_type3)
    allocate(ptr%ptr_cutTetra_type3)
    current => ptr%ptr_cutTetra_type3
    current%use_it = .true.
    isimplex = 0
    inext    = 1 ; inext2 = 2
    do isampleNeg = 1, Nneg
        if (iside == 0) then       ! positive side
            p(1)     = ptr%Pneg(isampleNeg)
        else if (iside == 1) then  ! negative side
            p(1)     = ptr%Ppos(isampleNeg)
        endif
        inext = inext + 1
        if (inext == Nneg) inext = 1
        if (iside == 0) then       ! positive side
            p(2)     =   ptr%Pneg(inext)
        else if (iside == 1) then  ! negative side
            p(2)     =   ptr%Ppos(inext)
```

```
               endif
               inext2 = inext2 + 1
               if (inext2 == Nneg) inext2 = 1
               if (iside == 0) then       ! positive side
                   p(3)     =   ptr%Pneg(inext2)
               else if (iside == 1) then  ! negative side
                   p(3)     =   ptr%Ppos(inext2)
               endif
               isimplex = isimplex + 1
               current%sim_node(1) = p(1)
               current%sim_node(2) = p(2)
               current%sim_node(3) = p(3)
               allocate(current%next)
               nullify( current%next%next )
               current => current%next
               current%use_it = .true.
           enddo
           ptr%ptr_cutTetra_type3%sim_count = isimplex
       endif
   enddo
   return

end subroutine create_simplices


!****************
!  secondary_cell_select
!
!  This subroutine finds what cells are secondary cells in the 3D environment either side of the isos
!****************
integer function  secondary_cell_select(ineighb)
   use connectivity
   implicit none
   ! code saturne variables
   ! integer          ndim   , ncelet
   ! double precision rtp(ncelet,*)

   integer(8), intent(in )                :: ineighb


   ! local variables
   integer            ifac, isgn, Number_Of_Faces
   double precision    phi_cenck, phi_nbck
   secondary_cell_select = 1
   phi_cenck       = rtp(ineighb,isca(2))
   if ( dabs(phi_cenck) < tiny(1.0)) then
       secondary_cell_select = 0
       return
   else if ( phi_cenck < 0.0d0) then
       isgn = -1
   else if ( phi_cenck > 0.0d0) then
       isgn = 1
   endif
   Number_Of_Faces=count(nbcell(ineighb,:)>-ihuge)
   do ifac = 1, Number_Of_Faces
       phi_nbck  = rtp(nbcell(ineighb,ifac),isca(2))
       if ( isgn > 0) then
           if ( phi_nbck < 0.0d0) then
               secondary_cell_select = -1
               return
           endif
       else if ( isgn < 0 ) then
           if ( phi_nbck > 0.0d0) then
               secondary_cell_select = -1
               return
           endif
       endif
   enddo  ! ifac loop
end function secondary_cell_select
!*************************
!****************************************
!  Vol_k
!
!  This subroutine calculates the difference in volumes defined by phi and phi*,
!  piecewise constant simplexwise mass correction function
!  and 3D reconstruction of the isosurface locally over simplex k
!****************************************
subroutine Vol_k(icheck,iside,current,ival_nb,itype,phi_value,val_adj,Area_K,Volume_k)
   use connectivity
   implicit none


   type(simplex), pointer, intent(in)            :: current

   logical, intent(in)                           :: icheck
   integer, intent(in)                           :: iside
   integer, intent(in)                           :: itype
   integer, intent(in)                           :: ival_nb
   double precision, intent(in )                 :: phi_value,val_adj
   double precision, intent(out)                 :: Area_K
   double precision, intent(out)                 :: Volume_k

   ! local variables
   double precision, parameter                   ::  vol_const = 0.16666667
   double precision  a(3),b(3),c(3),c1(3),c4(3),k(3),dif(3),v(3),s(3)
   double precision  sh_nb_temp(4,3),s_val, S_K, S_h, phi,vol_A, vol_B
   double precision  Area_K2,Area_K1,s1(3),s2(3),p3(3),pmid(3),vol_01
   double precision  vol_021,vol_022,vol_03,c2(3),c3(3),Area_1
   integer           ieln, inod2,Nmax,is1, ir, it_value1, it_value2

   !-------------------------------------------------------------------------
   ! Adjust reconstructed isosurface Sk by 'valadj' amount normal to surface
   !-------------------------------------------------------------------------
   select case (itype)
   case (1)
       Nmax = 3
```

```
case (2)
    Nmax = 4
case (3)
    Nmax = 3
case default
    call error_message('Error1 in Vol_k in USINV')
end select
!----------------------
if (icheck) then
    do inod2 = 1, Nmax
        sh_nb_temp(inod2,1) = current%sh_nb(inod2,1)
        sh_nb_temp(inod2,2) = current%sh_nb(inod2,2)
        sh_nb_temp(inod2,3) = current%sh_nb(inod2,3)
    enddo
else
    do inod2 = 1, Nmax
        if (itype == 1) then
            if ( inod2 == 1) then
                ir   =  ival_nb                ! Kth node
                phi  =     phi_value + val_adj
            else
                ir  =  current%sim_node(inod2)
                phi =  rtp(ir,isca(2))  + val_adj        ! new
            endif
            ieln =  current%sim_node(1)
        else if (itype == 2) then
            select case (inod2)
            case (1,2)
                ieln  =  current%sim_node(inod2)
                ir   =  ival_nb
                phi = phi_value + val_adj               ! new
            case (3,4)
                ival = inod2 - 2
                ieln  =  current%sim_node(ival)
                ir   =  current%sim_node(3)
                phi = rtp(ir,isca(2)) + val_adj         ! new
            case default
                call error_message('Error2 in Vol_k in USINV')
            end select
        else if (itype == 3) then
            ieln  =  current%sim_node(inod2)
            ir    =  ival_nb
            phi = phi_value + val_adj                 ! new
        else
            call error_message('Error3 in Vol_k in USINV')
        endif
        s_val    = phi  -  rtp(ieln,isca(2))
        call check_zero(s_val,'Error1 in USINV with Sk reconstruction at iel = ',iel)
        S_h  = - rtp(ieln,isca(2))/s_val
        a(1) = scale_length * (xyzcen(1,ir) )
        a(2) = scale_length * (xyzcen(2,ir) )
        a(3) = scale_length * (xyzcen(3,ir) )
        b(1) = scale_length * (xyzcen(1,ieln) )
        b(2) = scale_length * (xyzcen(2,ieln) )
        b(3) = scale_length * (xyzcen(3,ieln) )
        call position_vec(a, b, S_h, c)
        sh_nb_temp(inod2,1) = c(1)
        sh_nb_temp(inod2,2) = c(2)
        sh_nb_temp(inod2,3) = c(3)
    enddo
endif
!---------------------------------------------------------------------------
! calculate new simplex volume based on adjustment to reconstructed isosurface Sk
!---------------------------------------------------------------------------
! volume of small negative simplex
c1(1) = sh_nb_temp(1,1)  ! edge of free surface near kth node
c1(2) = sh_nb_temp(1,2)
c1(3) = sh_nb_temp(1,3)
if ( itype == 2 ) then
    !find volume of first sub-tetrahedron
    ir   =  current%sim_node(3)
    do id = 1,ndim
        a(id)  = scale_length * (xyzcen(id,ival_nb) )
        c2(id) = sh_nb_temp(2,id)
        c3(id) = sh_nb_temp(3,id)
        c4(id) = sh_nb_temp(4,id)
        p3(id) = scale_length * (xyzcen(id,ir) )
    enddo
    !---- vol01 volume----------------
    pmid = 0.5 * (p3 + a)
    s1 = pmid - a
    s2 = c1   - a
    call cross_product(s1,s2,v)
    dif   = c2 - a
    vol_01 = dabs(dot_product(dif, v))
    !vol_021 volume
    s1  = c1 - c2
    s2  = c4 - c2
    dif = pmid - c2
    call cross_product(s1,s2,v)
    vol_021 = dabs(dot_product(dif, v))
    !vol_022 volume
    s1  = c1 - c3
    s2  = c4 - c3
    dif = pmid - c3
    call cross_product(s1,s2,v)
    vol_022 = dabs(dot_product(dif, v))
    !vol_03 volume
    s1  = p3 - c4
    s2  = pmid - c2
    dif = pmid - c4
    call cross_product(s1,s2,v)
    vol_03 = dabs(dot_product(dif, v))
    if ( iside == 0) then
        Volume_k = vol_const*(vol_01 + vol_021 + vol_022 + vol_03)
    else
```

```
            vol_A = vol_01 + vol_021 + vol_022 + vol_03
        endif
        ! find Area_k
        s1 = c3 - c1
        s2 = c2 - c1
        call cross_product(s1,s2,v)
        S_k     = dabs(dot_product(v,v))
        if (S_k < tiny(1.0)) then
            Area_1 = 0.0
        else
            Area_1 = 0.5 * dsqrt(S_k)
        endif
        s1 = c3 - c4
        s2 = c2 - c4
        call cross_product(s1,s2,v)
        S_k     = dabs(dot_product(v,v))
        if (S_k < tiny(1.0)) then
            Area_K = Area_1
        else
            Area_K = (0.5 * dsqrt(S_k)) + Area_1
        endif
        if (iside == 1) then
            ieln = current%sim_node(1)
            do id = 1,ndim
                a(id)  = scale_length * (xyzcen(id,ival_nb) )
                b(id)  = scale_length * (xyzcen(id,ieln) )
            enddo
            dif = b - a
            is1 = current%sim_node(2)
            s1(1)   = scale_length * (xyzcen(1,is1) ) - a(1)
            s1(2)   = scale_length * (xyzcen(2,is1) ) - a(2)
            s1(3)   = scale_length * (xyzcen(3,is1) ) - a(3)
            is1 = current%sim_node(3)
            s2(1)   = scale_length * (xyzcen(1,is1) ) - a(1)
            s2(2)   = scale_length * (xyzcen(2,is1) ) - a(2)
            s2(3)   = scale_length * (xyzcen(3,is1) ) - a(3)
            call cross_product(s1,s2,v)
            vol_B = dabs(dot_product(dif,v))
            volume_K = vol_const * (vol_B - vol_A)
        endif
        return
    else
        if (itype == 1) then
            ieln =  current%sim_node(1)
        else if (itype == 3) then
            ieln = ival_nb   ! positive peak at node k
        else
            call error_message('Error4 in Vol_k in USINV')
        endif
        b(1) = scale_length * (xyzcen(1,ieln) )
        b(2) = scale_length * (xyzcen(2,ieln) )
        b(3) = scale_length * (xyzcen(3,ieln) )
        dif(1) = b(1) - c1(1)   ! height of small simplex
```

```
dif(2) = b(2) - c1(2)
dif(3) = b(3) - c1(3)
s1(1) = sh_nb_temp(2,1) - c1(1)
s1(2) = sh_nb_temp(2,2) - c1(2)
s1(3) = sh_nb_temp(2,3) - c1(3)
s2(1) = sh_nb_temp(3,1) - c1(1)
s2(2) = sh_nb_temp(3,2) - c1(2)
s2(3) = sh_nb_temp(3,3) - c1(3)
call cross_product(s1,s2,v)
S_k     = dabs(dot_product(v,v))
if (S_k < tiny(1.0)) then
    Area_K       = 0.0
else
    Area_K       = 0.5 * dsqrt(S_k)
endif
vol_A =  dabs(dot_product(dif, v))
if (iside == 0) then
    it_value1 = 3
    it_value2 = 1
else if (iside == 1) then
    it_value1 = 1
    it_value2 = 3
endif
if (itype == it_value1) then
    Volume_k = vol_const * vol_A
    return
else if (itype == it_value2) then
    !------ larger simplex next
    if (iside == 0) then
        ieln =  ival_nb
    else if (iside == 1) then
        ieln =  current%sim_node(1)
    endif
    a(1) = scale_length * (xyzcen(1,ieln) )
    a(2) = scale_length * (xyzcen(2,ieln) )
    a(3) = scale_length * (xyzcen(3,ieln) )
    dif(1) = b(1) - a(1)   ! height of the larger simplex
    dif(2) = b(2) - a(2)
    dif(3) = b(3) - a(3)
    is1 =  current%sim_node(2)
    s1(1) = scale_length * (xyzcen(1,is1) ) - a(1)
    s1(2) = scale_length * (xyzcen(2,is1) ) - a(2)
    s1(3) = scale_length * (xyzcen(3,is1) ) - a(3)
    is1 =  current%sim_node(3)
    s2(1) = scale_length * (xyzcen(1,is1) ) - a(1)
    s2(2) = scale_length * (xyzcen(2,is1) ) - a(2)
    s2(3) = scale_length * (xyzcen(3,is1) ) - a(3)
    call cross_product(s1,s2,v)
    vol_B =  dabs(dot_product(dif, v))
    !-------resultant simplex volume below free surface
    Volume_k = vol_const * (vol_B - vol_A)
    return
else
```

```
            call error_message('Error5 in Vol_k in USINV')
        endif
    endif
 end subroutine Vol_k
!****************************************
!  deltaV
!
!  This subroutine calculates the difference in volumes defined by phi and phi*, node wise mass corre
!  which is used in the false position algorithm to find the constant C which globally preserves volu
!****************************************
double precision function deltaV(iside,ptr, C)
   use connectivity
   implicit none


   type (narrow_band_element), intent(in)       :: ptr
   double precision, intent(in )                :: C
   integer,intent(in)                           :: iside

   ! local variables
   integer            Nneg, Npos,itype_max,itype,isimpx,isimplxMax
   integer            ival_nb
   double precision    s_k, phi, phi_starr, Cxi_h, f_sum, Area_K
   double precision    vol2, vol1

   f_sum = 0.0
   Cxi_h = ptr%xi_h * C
   phi_starr  =   ptr%ph_star
   Npos = ptr%pos
   Nneg = ptr%neg
   ival_nb    =   ptr%nwb_index
   if (iside == 0) then
       if (Nneg > 3) then
           itype_max = 3
       else
           itype_max = Nneg
       endif
   else if (iside == 1) then
       if (Npos > 3) then
           itype_max = 3
       else
           itype_max = Npos
       endif
   endif
   do itype = 1, itype_max
       select case (itype)
       case (1)
           isimplxMax = ptr%ptr_cutTetra_type1%sim_count
           current => ptr%ptr_cutTetra_type1
       case (2)
           isimplxMax = ptr%ptr_cutTetra_type2%sim_count
           current => ptr%ptr_cutTetra_type2
       case (3)
```

```
            isimplxMax = ptr%ptr_cutTetra_type3%sim_count
            current => ptr%ptr_cutTetra_type3
        case default
            call error_message('Error1 in step1 with isimplxMax in USINV')
        end select
        !  *** scan for isimplxMax per simplex cut type considered  ***
        if (isimplxMax > 0) then
            do isimpx = 1, isimplxMax
                if (current%use_it) then                                !new
                    vol2 = current%vol_phi
                    call Vol_k(.false. ,iside,current,ival_nb,itype,phi_starr,Cxi_h,Area_K,vol1)
                    f_sum = f_sum + (vol2 - vol1)
                endif                                                   !new
                current => current%next
            enddo
        endif
    enddo
    deltaV = f_sum
    return
end function deltaV
!========================================
!---------------------------------------------------------------------------------------------------------
!  Returns inverted matrix for step 6 needed with 3D geometrically isotropic
!  trilinear interpolation involved with the intersection on a face of a tetrahedral simplex
!--------------------------------------------------------------------------------------------------------
subroutine calc_invert_matrix(x,a_matrx,ntri)
    implicit none
    real, dimension(:,:), allocatable :: Matrix, invMatrix
    real, dimension(:,:)              :: a_matrx, x
    integer                           :: ntri

    allocate(Matrix(ntri,ntri))
    allocate(invMatrix(ntri,ntri))
    do i = 1, ntri
        do j = 1, ntri
            if (j == 1) then
                Matrix(i,j) = 1.0
            else
                Matrix(i,j) = x(i,j-1)
            endif
        enddo
    enddo
    call FindInv(Matrix, invMatrix, ntri, ErrorFlag)
    do i = 1, ntri
        do j = 1, ntri
            a_matrx(i,j) = invMatrix(i,j)
        enddo
    enddo
    deallocate(Matrix)
    deallocate(invMatrix)
    return
end subroutine calc_invert_matrix
!--------------------------------------------------------------------------------
```

```
!  check to ensure that the intersection p_x is within the particular face
!  of the tetrahedral simplex
!-------------------------------------------------------------------------------
logical function check_simplex(n_tri,simVec,p_x)
   implicit none
   real, dimension(:,:)             :: simVec
   real, dimension(:)               :: p_x

   ! local variables
   double precision  n_vec(3),r_mid(3),nface_hat(3),sim_dif(3), t_vec(3),val
   double precision  v_smll,n_val,dval
   integer id,n_tri

   do i = 1, n_tri
       do id = 1, ndim
           if (i < n_tri) then
               t_vec(id) = simVec(i+1,id) - simVec(i,id)
               sim_dif(id) = (simVec(i+1,id) + simVec(i,id) ) * 0.5 - p_x(id)
           else
               t_vec(id) = simVec(1,id) - simVec(i,id)
               sim_dif(id) = (simVec(1,id) + simVec(i,id) ) * 0.5 - p_x(id)
           endif
       enddo
       call cross_product(nface_hat, t_vec, n_vec)
       n_val = dsqrt(dabs(dot_product(n_vec,n_vec)))
       if ( n_val < tiny(1.0))  then
           check_simplex = .true.
           return
       endif
       n_vec(1) = n_vec(1)/n_val;n_vec(2) = n_vec(2)/n_val;n_vec(3) = n_vec(3)/n_val;
       val = dot_product(sim_dif, n_vec)
       dval = dsqrt(dabs(dot_product(sim_dif,sim_dif)))
       v_smll = dval * 0.01
       if (dabs(val) < v_smll) then
           check_simplex = .true.
       else if (val >= 0.0) then
           check_simplex = .true.
       else
           check_simplex = .false.
           return
       endif
   enddo
   return
end function check_simplex
!----------------------------------------------------------------------------------

!==========================================
subroutine convert_Vec(ntri,k,a,b,c,simVec)

   implicit none

   integer                        , intent (IN)    :: ntri
   double precision, dimension(:), intent (IN)    :: k
   double precision, dimension(:), intent (IN)    :: a
   double precision, dimension(:), intent (IN)    :: b
   double precision, dimension(:), intent (IN)    :: c
   real          , dimension(:,:), intent (OUT)   :: simVec

   do i = 1, ntri
       if (i == 1) then
           do j = 1, ndim
               simVec(i,j) = k(j)
           enddo
       else if (i == 2) then
           do j = 1, ndim
               simVec(i,j) = a(j)
           enddo
       else if (i == 3) then
           do j = 1, ndim
               simVec(i,j) = b(j)
           enddo
       else if (i == 4) then
           do j = 1, ndim
               simVec(i,j) = c(j)
           enddo
       endif
   enddo
   return

end subroutine convert_Vec
!==========================================
subroutine create_vec (A, B, TSCAL, C)

   implicit none

   double precision, dimension(3), intent (IN)    :: A                             ! multiplicand 3-vector
   double precision, dimension(3), intent (IN)    :: B                             ! multiplier 3-vector
   double precision,               intent (IN)    :: TSCAL                         ! scalar
   real,             dimension(3), intent (OUT)   :: C                             ! result: 3-vector position

   C(1) = A(1) + TSCAL * B(1)
   C(2) = A(2) + TSCAL * B(2)
   C(3) = A(3) + TSCAL * B(3)
   return

end subroutine create_vec
!============================================
!==========================================
subroutine POSITION_VEC (A, B, TSCAL, C)

   implicit none

   double precision, dimension(3), intent (IN)    :: A                             ! multiplicand 3-vector
   double precision, dimension(3), intent (IN)    :: B                             ! multiplier 3-vector
   double precision,               intent (IN)    :: TSCAL                         ! scalar
   double precision, dimension(3), intent (OUT)   :: C                             ! result: 3-vector position
   !local variables
   double precision, dimension(3)                 :: D

   D(1) = B(1) - A(1)
   D(2) = B(2) - A(2)
   D(3) = B(3) - A(3)
   C(1) = A(1) + TSCAL * D(1)
   C(2) = A(2) + TSCAL * D(2)
   C(3) = A(3) + TSCAL * D(3)
   return

end subroutine POSITION_VEC
!============================================

function dot_product (V1, V2) result (PROD)

   implicit none

   double precision, dimension(3), intent(IN) :: V1, V2
   double precision :: PROD


   PROD = V1(1)*V2(1) + V1(2)*V2(2) + V1(3)*V2(3)
   return

end function DOT_PRODUCT
!***********************
!***********************
!  CROSS_PRODUCT
!
!  Returns the right-handed vector cross product of two 3d-vectors:  C = A x B.
!
!  Code pasted from: http://www.davidgsimpson.com/software/crossprd_f90.txt
!  Checked veracity with Wikipedia
!***********************

subroutine CROSS_PRODUCT (A, B, C)                                         ! cross product (right-

   implicit none                                                              ! no default typing

   double precision, dimension(3), intent (IN)    :: A                        ! multiplicand 3d-ve
   double precision, dimension(3), intent (IN)    :: B                        ! multiplier 3d-vect
   double precision, dimension(3), intent (OUT)   :: C                        ! result: 3d-vector


   C(1) = A(2)*B(3) - A(3)*B(2)                                               ! compute cross prod
   C(2) = A(3)*B(1) - A(1)*B(3)
   C(3) = A(1)*B(2) - A(2)*B(1)

   return

end subroutine CROSS_PRODUCT
```

```
!-----------------------------------------------------------------------------------
!Subroutine to find the inverse of a square matrix
!Author : Louisda16th a.k.a Ashwith J. Rego
!Reference : Algorithm has been well explained in:
!http://math.uww.edu/~mcfarlat/inverse.htm
!http://www.tutor.ms.unimelb.edu.au/matrix/matrix_inverse.html
!-------------------------------------------------------------------------------------
subroutine FINDInv(matrix, inverse, n, errorflag)
   implicit none
   !Declarations
   integer, intent(IN) :: n
   integer, intent(OUT) :: errorflag  !Return error status. -1 for error, 0 for normal
   real, intent(IN), dimension(n,n) :: matrix  !Input matrix
   real, intent(OUT), dimension(n,n) :: inverse !Inverted matrix

   logical :: FLAG = .true.
   integer :: i, j, k, l
   real :: m
   real, dimension(n,2*n) :: augmatrix !augmented matrix

   !Augment input matrix with an identity matrix
   do i = 1, n
       do j = 1, 2*n
           if (j <= n ) then
               augmatrix(i,j) = matrix(i,j)
           else if ((i+n) == j) then
               augmatrix(i,j) = 1
           else
               augmatrix(i,j) = 0
           endif
       end do
   end do

   !Reduce augmented matrix to upper traingular form
   do k =1, n-1
       if (augmatrix(k,k) == 0) then
           FLAG = .false.
           do i = k+1, n
               if (augmatrix(i,k) /= 0) then
                   do j = 1,2*n
                       augmatrix(k,j) = augmatrix(k,j)+augmatrix(i,j)
                   end do
                   FLAG = .true.
                   exit
               endif
               if (FLAG .eqv. .false.) then
                   print*, "Matrix is non - invertible"
                   inverse = 0
                   errorflag = -1
                   return
               endif
           end do
       endif
```

```
          do j = k+1, n
              m = augmatrix(j,k)/augmatrix(k,k)
              do i = k, 2*n
                  augmatrix(j,i) = augmatrix(j,i) - m*augmatrix(k,i)
              end do
          end do
      end do

      !Test for invertibility
      do i = 1, n
          if (augmatrix(i,i) == 0) then
              print*, "Matrix is non - invertible"
              inverse = 0
              errorflag = -1
              return
          endif
      end do

      !Make diagonal elements as 1
      do i = 1 , n
          m = augmatrix(i,i)
          do j = i , (2 * n)
              augmatrix(i,j) = (augmatrix(i,j) / m)
          end do
      end do

      !Reduced right side half of augmented matrix to identity matrix
      do k = n-1, 1, -1
          do i =1, k
              m = augmatrix(i,k+1)
              do j = k, (2*n)
                  augmatrix(i,j) = augmatrix(i,j) -augmatrix(k+1,j) * m
              end do
          end do
      end do

      !store answer
      do i =1, n
          do j = 1, n
              inverse(i,j) = augmatrix(i,j+n)
          end do
      end do
      errorflag = 0
   end subroutine FINDinv
end subroutine usproj
```

Appendix D

usphyv.f90

```
!-------------------------------------------------------------------------------

!                      Code_Saturne version 2.0.0-rc1
!                      --------------------------

!     This file is part of the Code_Saturne Kernel, element of the
!     Code_Saturne CFD tool.

!     Copyright (C) 1998-2009 EDF S.A., France

!     contact: saturne-support@edf.fr

!     The Code_Saturne Kernel is free software; you can redistribute it
!     and/or modify it under the terms of the GNU General Public License
!     as published by the Free Software Foundation; either version 2 of
!     the License, or (at your option) any later version.

!     The Code_Saturne Kernel is distributed in the hope that it will be
!     useful, but WITHOUT ANY WARRANTY; without even the implied warranty
!     of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!     GNU General Public License for more details.

!     You should have received a copy of the GNU General Public License
!     along with the Code_Saturne Kernel; if not, write to the
!     Free Software Foundation, Inc.,
!     51 Franklin St, Fifth Floor,
!     Boston, MA  02110-1301  USA

!-------------------------------------------------------------------------------

Subroutine usphyv &
      !================

      ( idbia0 , idbra0 ,                                              &
      ndim   , ncelet , ncel   , nfac   , nfabor , nfml   , nprfml , &
```

```
   nnod   , lndfac , lndfbr , ncelbr ,                            &
   nvar   , nscal  , nphas  ,                                     &
   nideve , nrdeve , nituse , nrtuse , nphmx  ,                   &
   ifacel , ifabor , ifmfbr , ifmcel , iprfml ,                   &
   ipnfac , nodfac , ipnfbr , nodfbr , ibrom  ,                   &
   idevel , ituser , ia     ,                                     &
   xyzcen , surfac , surfbo , cdgfac , cdgfbo , xyznod , volume , &
   dt     , rtp    , rtpa   ,                                     &
   propce , propfa , propfb ,                                     &
   coefa  , coefb  ,                                              &
   w1     , w2     , w3     , w4     ,                            &
   w5     , w6     , w7     , w8     ,                            &
   rdevel , rtuser , ra     )

!===================
! Purpose:
! -------

!    User subroutine.

!    Definition of physical variable laws.

! Warning:
! -------

! It is forbidden to modify turbulent viscosity "visct" here
!       =========
!    (a specific subroutine is dedicated to that: usvist)


! icp(iphas) = 1 must have been specified
!                ========================
!    in usini1 if we wish to define a varible specific heat
!    cp for phase iphas (otherwise: memory overwrite).


! ivisls(iphas) = 1 must have been specified
!                   ========================
!    in usini1 if we wish to define a variable viscosity
!    viscls for phase iphas (otherwise: memory overwrite).


! Notes:
! -----

! This routine is called at the beginning of each time step

!    Thus, AT THE FIRST TIME STEP (non-restart case), the only
!    values initialized before this call are those defined
!      - in usini1 :
!             . density    (initialized at ro0(iphas))
!             . viscosity  (initialized at viscl0(iphas))
!      - in usiniv :
```

```
!             . calculation variables (initialized at 0 by defaut
!             or to the value given in the GUI or in usiniv)

! We may define here variation laws for cell properties, for:
!     - density                                    rom    kg/m3
!         (possibly also at boundary faces         romb   kg/m3)
!     - molecular viscosity                        viscl  kg/(m s)
!     - specific heat                              cp     J/(kg degrees)
!     - "diffusivities" associated with scalars   viscls kg/(m s)


! The types of boundary faces at the previous time step are available
!   (except at the first time step, where arrays itypfb and itrifb have
!   not been initialized yet)


! It is recommended to keep only the minimum necessary in this file
!   (i.e. remove all unused example code)


! Cells identification
! ====================

! Cells may be identified using the 'getcel' subroutine.
! The syntax of this subroutine is described in the 'usclim' subroutine,
! but a more thorough description can be found in the user guide.


! Arguments
!__________________.____._____.________________________________________________.
! name             !type!mode ! role                                           !
!__________________!____!_____!________________________________________________!
! idbia0           ! i  ! <-- ! number of first free position in ia            !
! idbra0           ! i  ! <-- ! number of first free position in ra            !
! ndim             ! i  ! <-- ! spatial dimension                              !
! ncelet           ! i  ! <-- ! number of extended (real + ghost) cells        !
! ncel             ! i  ! <-- ! number of cells                                !
! nfac             ! i  ! <-- ! number of interior faces                       !
! nfabor           ! i  ! <-- ! number of boundary faces                       !
! nfml             ! i  ! <-- ! number of families (group classes)             !
! nprfml           ! i  ! <-- ! number of properties per family (group class)  !
! nnod             ! i  ! <-- ! number of vertices                             !
! lndfac           ! i  ! <-- ! size of nodfac indexed array                   !
! lndfbr           ! i  ! <-- ! size of nodfbr indexed array                   !
! ncelbr           ! i  ! <-- ! number of cells with faces on boundary         !
! nvar             ! i  ! <-- ! total number of variables                      !
! nscal            ! i  ! <-- ! total number of scalars                        !
! nphas            ! i  ! <-- ! number of phases                               !
! nideve, nrdeve   ! i  ! <-- ! sizes of idevel and rdevel arrays              !
! nituse, nrtuse   ! i  ! <-- ! sizes of ituser and rtuser arrays              !
! nphmx            ! e  ! <-- ! nphsmx                                         !
! ifacel(2, nfac)  ! ia ! <-- ! interior faces -> cells connectivity           !
! ifabor(nfabor)   ! ia ! <-- ! boundary faces -> cells connectivity           !
! ifmfbr(nfabor)   ! ia ! <-- ! boundary face family numbers                  !
! ifmcel(ncelet)   ! ia ! <-- ! cell family numbers                           !
! iprfml           ! ia ! <-- ! property numbers per family                   !
!  (nfml, nprfml)  !    !     !                                               !
! ipnfac(nfac+1)   ! ia ! <-- ! interior faces -> vertices index (optional)   !
! nodfac(lndfac)   ! ia ! <-- ! interior faces -> vertices list (optional)    !
! ipnfbr(nfabor+1) ! ia ! <-- ! boundary faces -> vertices index (optional)   !
! nodfbr(lndfbr)   ! ia ! <-- ! boundary faces -> vertices list (optional)    !
! ibrom            ! te ! <-- ! indicateur de remplissage de romb             !
!   (nphmx   )     !    !     !                                               !
! idevel(nideve)   ! ia ! <-> ! integer work array for temporary development  !
! ituser(nituse)   ! ia ! <-> ! user-reserved integer work array              !
! ia(*)            ! ia ! --- ! main integer work array                       !
! xyzcen           ! ra ! <-- ! cell centers                                  !
!  (ndim, ncelet)  !    !     !                                               !
! surfac           ! ra ! <-- ! interior faces surface vectors                !
!  (ndim, nfac)    !    !     !                                               !
! surfbo           ! ra ! <-- ! boundary faces surface vectors                !
!  (ndim, nfabor)  !    !     !                                               !
! cdgfac           ! ra ! <-- ! interior faces centers of gravity             !
!  (ndim, nfac)    !    !     !                                               !
! cdgfbo           ! ra ! <-- ! boundary faces centers of gravity             !
!  (ndim, nfabor)  !    !     !                                               !
! xyznod           ! ra ! <-- ! vertex coordinates (optional)                 !
!  (ndim, nnod)    !    !     !                                               !
! volume(ncelet)   ! ra ! <-- ! cell volumes                                  !
! dt(ncelet)       ! ra ! <-- ! time step (per cell)                          !
! rtp, rtpa        ! ra ! <-- ! calculated variables at cell centers          !
!  (ncelet, *)     !    !     !  (at current and previous time steps)         !
! propce(ncelet, *)! ra ! <-- ! physical properties at cell centers           !
! propfa(nfac, *)  ! ra ! <-- ! physical properties at interior face centers  !
! propfb(nfabor, *)! ra ! <-- ! physical properties at boundary face centers  !
! coefa, coefb     ! ra ! <-- ! boundary conditions                           !
!  (nfabor, *)     !    !     !                                               !
! w1...8(ncelet    ! tr ! --- ! tableau de travail                            !
! rdevel(nrdeve)   ! ra ! <-> ! real work array for temporary development     !
! rtuser(nrtuse)   ! ra ! <-> ! user-reserved real work array                 !
! ra(*)            ! ra ! --- ! main real work array                          !
!__________________!____!_____!_______________________________________________!

!     Type: i (integer), r (real), s (string), a (array), l (logical),
!           and composite types (ex: ra real array)
!     mode: <-- input, --> output, <-> modifies data, --- work array
!=========================
Use connectivity
Implicit None

!=========================
! Common blocks
!=========================

Include "dimfbr.h" !
Include "paramx.h"
Include "pointe.h"
Include "numvar.h"
Include "optcal.h"
Include "cstphy.h"

Include "cstnum.h" !


Include "entsor.h"

Include "lagpar.h" !
Include "lagran.h" !


Include "parall.h"
Include "period.h"

Include "ppppar.h" !
Include "ppthch.h" !
Include "ppincl.h" !

!====================

! Arguments

Integer          idbia0 , idbra0
Integer          ndim   , ncelet , ncel   , nfac   , nfabor
Integer          nfml   , nprfml
Integer          nnod   , lndfac , lndfbr , ncelbr
Integer          nvar   , nscal  , nphas
Integer          nideve , nrdeve , nituse , nrtuse , nphmx

Integer          ifacel(2,nfac) , ifabor(nfabor)
Integer          ifmfbr(nfabor) , ifmcel(ncelet)
Integer          iprfml(nfml,nprfml)
Integer          ipnfac(nfac+1), nodfac(lndfac)
Integer          ipnfbr(nfabor+1), nodfbr(lndfbr), ibrom(nphmx)
Integer          idevel(nideve), ituser(nituse), ia(*)

Double Precision xyzcen(ndim,ncelet)
Double Precision surfac(ndim,nfac), surfbo(ndim,nfabor)
Double Precision cdgfac(ndim,nfac), cdgfbo(ndim,nfabor)
Double Precision xyznod(ndim,nnod), volume(ncelet)
Double Precision dt(ncelet), rtp(ncelet,*), rtpa(ncelet,*)
Double Precision propce(ncelet,*)
Double Precision propfa(nfac,*), propfb(nfabor,*)
Double Precision coefa(nfabor,*), coefb(nfabor,*)
Double Precision w1(ncelet),w2(ncelet),w3(ncelet),w4(ncelet)
Double Precision w5(ncelet),w6(ncelet),w7(ncelet),w8(ncelet)
Double Precision rdevel(nrdeve), rtuser(nrtuse), ra(*)
```

```
! Local variables

Integer          idebia, idebra,ifac,Number_Of_Faces
Integer(8)       ihuge
Integer          ivart, iclvar, iel, iphas
Integer          ipcrom, ipcptot, ipcvis, ipccp
Integer          ipcvsl, ith, iscal, ii, impout(6)
Integer          iutile
Double Precision vara, varb, varc, varam, varbm, varcm, vardm
Double Precision                  varal, varbl, varcl, vardl
Double Precision                  varac, varbc
! water to air density smoothing
Double Precision  h_s, xrtp, xrtp2, density
Double Precision, Parameter ::   rho_water = 997.d0, rho_air = 1.0d0
Double Precision, Parameter ::  epsilon = 0.002
Double Precision, Parameter ::  dynVis_air = 1.983d-5, dynVis_water = 0.001
Double Precision,  Parameter :: div_pi = 1.0/pi

!========================
! 0. Initializations
!========================

! --- Memory initialization

idebia = idbia0
idebra = idbra0


!==============================
!    do ii = 1, 1

!      impout(ii) = impusr(ii)

!   enddo
!   open(impout(1),file='check_rho_Prof.dat')

!=========================

! Variable density, as a function of the Level Set scalar
! ================

! We use the same density law for all phases.
!    Values of this property are assigned to cell centers.
!    (and optionally to boundary faces).

!=======================

iphas = 1 ! Single phase problem

! Density and total pressure  at cell centers
! -----------------------
!   Law:              rho        =   function(Level Set scalar)
!   Code:    propce(iel, ipcrom) =   f(xrtp)
```

```
   Do iel = 1, ncel
       xrtp = rtp(iel,isca(2))/scale_length
       xrtp2 = - xrtp
       If (dabs(xrtp) < epsilon) Then
           h_s =  0.5 - 0.5 * ((xrtp/epsilon) +  (dsin(pi * (xrtp/epsilon)) /pi ) )
       Else If (xrtp2 > epsilon) Then
           h_s = 1.0
       Else If (xrtp > epsilon) Then
           h_s = 0.0
       Endif
       ! The position of the density of phase iphas in propce
       ! (physical properties at element centers) given by ipcrom.
       ipcrom = ipproc(irom(iphas))
       propce(iel,ipcrom) = rho_air + (rho_water - rho_air) * h_s
       density = propce(iel,ipcrom)
       ipcvis = ipproc(iviscl(iphas))
       propce(iel,ipcvis) = dynVis_air + (dynVis_water - dynVis_air) * h_s
       !   if (ntcabs == 1) then
       !    write(impout(1),"(3g17.9)") xyzcen(1,iel), xyzcen(2,iel), density
       !  endif
   Enddo

   !============================
   ! --------------------------------------------------
   ! Close files at final time step
   ! --------------------------------------------------

   !  if (ntcabs.eq.ntmabs) then

   !if (irangp.le.0) then

   !  do ii = 1, 1

   !     close(impout(ii))

   !   enddo

   !endif

   !  endif
   ! --------------------------------------------------
   ! Close files at final time step
   ! --------------------------------------------------


   !endif   ! --- Test on 'iutile'
   !print*,'end of usphyv'
   Return
End Subroutine usphyv
```