TRUST

TRUST team

Nov 27, 2025

CONTENTS

Hi there and welcome to the **TRUST** documentation page !

TRUST is a High Performance Computing (HPC) thermohydraulic engine for Computational Fluid Dynamics (CFD) and is developed at the Department of System and Structure Modelisation (DM2S) of the French Atomic Energy Commission (CEA).

TRUST in an open-source software under the BSD license, available on Github. Although the software was primarily designed for solving incompressible single-phase flows, you can now use it for solving multiphase flow problems.

TRUST is also progressively ported to support GPU acceleration, using the Kokkos library. It has been selected to be a demonstrator of the CExA project.

When speaking of the **TRUST** code, we use the world platform as it serves as a base for other research and industrial CEA codes, called BALTIKS. TrioCFD is one of those BALTIKS.

If you need help to navigate between the different options for creating, running and, post-processing a test case, make sure to check out the *TRUST User Guide*. This guide lists the available options in **TRUST** and gives you some advice depending on your applications, but also helps you better understand what is under each keyword you will need to use.

Here are some useful links that you can visit too:

- **TRUST Code**
  https://github.com/cea-trust-platform/trust-code
- **TRUST Website**
  https://cea-trust-platform.github.io
- **TRUST Support**
  trust@cea.fr

Do not forget to cite TRUST.

CHAPTER

ONE

TRUST USER GUIDE

You will find here the **TRUST** user guide, giving you a brief overview on how to use **TRUST**. It will help you understand how to use effectively **TRUST** and will give you some details regarding what is hidden beneath the keywords you will use.

If you want to better understand the C++ part of the code, go check the ../dev_corner/index.

Do not forget that you can use the research bar located on the top right of your screen to quickly lookup a precise element or keyword.

Table Of Contents

1.1 TRUST General Guide

This is the general guide for **TRUST** users. You can find informations regarding the TRUST in general, the options of a *.data* files and also some advices.

1.1.1 Introduction

TRUST is a High Performance Computing (HPC) thermal-hydraulic engine for Computational Fluid Dynamics (CFD) developed at the Departement of System and Structure Modelisation (DM2S) of the French Atomic Energy Commission (CEA).

The acronym **TRUST** stands for **TR**io_**U** **S**oftware for **T**hermohydraulics. This software was originally designed for conduction, incompressible single-phase, and Low Mach Number (LMN) flows with a robust Weakly-Compressible (WC) multi-species solver. However, a huge effort has been conducted recently, and now TRUST is able to simulate real compressible multi-phase flows.

TRUST is also being progressively ported to support GPU acceleration (NVidia/AMD), using the Kokkos library.

The software is OpenSource with a BSD license, available on GitHub via this link.

You can easily create new project based on **TRUST** plateform. Theses projects are named **BALTIK** projects (**B**uild an **A**pplication **L**inked to **T**r**I**o_U **K**ernel).

A bit of history: the Modular Software Named Trio_U

TRUST was born from the cutting in two pieces of **Trio_U** software. **Trio_U** was a software brick based on the **Kernel** brick (which contains the equations, space discretizations, numerical schemes, parallelism...) and used by other CEA applications (see Figure 1).

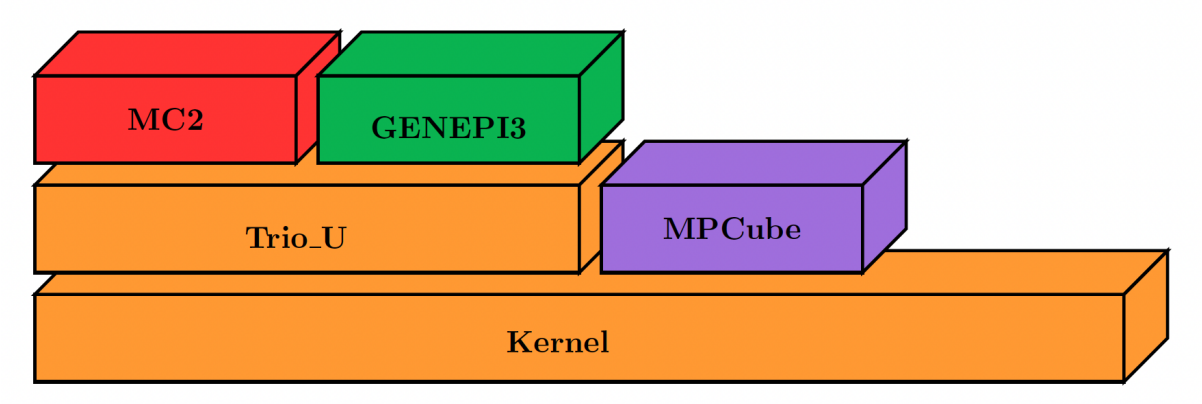


Figure 1.1.1: Figure 1: Trio_U brick software

In 2015, **Trio_U** was divided in two parts: **TRUST** and **TrioCFD**.

- **TRUST** is a new platform, its name means: **TR**io_U **S**oftware for **T**hermohydraulics.
- **TrioCFD** is an open source BALTIK project based on **TRUST**.

Here are some other selected BALTIKS based on the TRUST platform (see Figure 2).

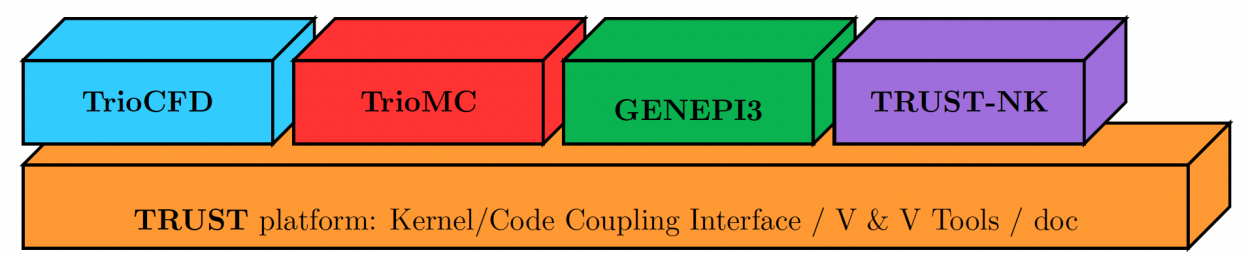


Figure 1.1.2: Figure 2: Selected BALTIKS based on the TRUST platform.

Short History

TRUST is developed at the Laboratory of High Performance Computing and Numerical Analysis (LCAN) of the Software Engineering and Simulation Service (SGLS) in the Department of System and Structure Modeling (DM2S).

The project starts in 1994 and improved versions were built ever since:

- **1994 :** Start of the project Trio_U
- **1997 :** v1.0 - Finite Difference Volume (VDF) method only
- **1998 :** v1.1 - Finite Element Volume (VEF) method only
- **2000 :** v1.2 - Parallel MPI version
- **2001 :** v1.3 - Radiation model (TrioCFD now)

- **2002 :** v1.4 - LES turbulence models (TrioCFD now)
- **2006 :** v1.5 - VDF/VEF Front Tracking method (TrioCFD now)
- **2009 :** v1.6 - Data structure revamped
- **2015 :** v1.7 - Separation TRUST & TrioCFD + switch to open source
- **2019 :** v1.8 - New polyheadral discretization (PolyMAC) + Multiphase problem + Weakly Compressible model
- **2022 :** v1.9.0 - Modern C++ code (templates, CRTP, ...) + remove MACROS + support GPU (NVidia/AMD)
- **2025 :** v1.9.6 - Unified version to handle 32-64b integers + VEF discretisation supported on GPU

1.1.2 How to use a data file

To launch a calculation with **TRUST**, you need to write a data file which is an input file for **TRUST** and will contain all the information about your simulation. Data files are written following some rules as shown below. But their language is not a programming language, users can't make loops or switch. We will now explain how to fill a data file. First you must specify some basic information like the dimension of your domain, its name, the problem type...

Basic Rules

There is no line concept in **TRUST** `.data` files.

Data files uses `blocks`. They may be defined using the braces:

```
{
   a block
}
```

Objects Notion

Objects are created in the data set as follows:

```
[ export ] Type identificateur
```

- **export**: if this keyword is included, *identificateur* (identifier) will have a global range, if not, its range will be applied to the block only (the associated object will be destroyed on exiting the block).
- **Type**: must be a type of object recognised by **TRUST**, which correspond to the C++ classes. The list of recognised types is given in the file `hierarchie.dump`.
- **identificateur**: the identifier of the created object type *Type* corresponds to an instancy of the C++ class *Type*. **TRUST** exits an error if the identifier has already been used.

There are several object types. Physical objects, for example:

- A **Fluide_incompressible** (incompressible_Fluid) object. This type of object is defined by its physical characteristics (its dynamic viscosity μ (keyword **mu**), its density ρ (keyword **rho**), etc...).
- A **Domaine**.

More abstract object types also exist:

- A **VDF**, **VEFPreP1B**, **PolyMAC_P0P1NC** or **PolyMAC_P0** according to the discretization type.
- A **Scheme_euler_explicit** to indicate the time scheme type.
- A **Solveur_pression** to denote the pressure system solver type.

- A **Uniform_field** to define, for example, the gravity field.

Interpretor Notion

Interprete (interpretor) type objects are then used to handle the created objects with the following syntax:

```
Type_interprete argument
```

- **Type_interprete**: any type derived from the **Interprete** (Interpretor) type recognised by **TRUST**.
- **argument**: an argument may comprise one or several object identifiers and/or one or several data blocks.

Interpretors allow some operations to be carried out on objects.

Currently available general interpretors include **Read**, **Read_file**, **Ecrire** (Write), **Ecrire_fichier** (Write_file), **Associate**.

Example

A data set to write Ok on screen:

```
Nom a_name        # Creation of an object type. Name identifier a_name #
Read a_name Ok    # Allocates the string "Ok" to a_name #
Ecrire a_name     # Write a_name on screen #
```

Important Remarks

- To insert *comments* in the data set, use # .. # (or /* ... */), the character # must always be enclosed by blanks.
- Comma separates items in a list (a comma must be enclosed with spaces or a new line).
- Interpretor keywords are not case sensitive.
- **On the contrary, object names (identifiers) are case sensitive.**
- In the following description, items (keywords or values) enclosed by [and] are *optional*.
- **TRUST** keywords can be highlighted with your file editor via the command line:

  ```
  trust -config gedit|vim|emacs
  ```

- You can search for a keyword meaning in the *TRUST Keyword Reference Manual*
- You can search for example of a keyword usage in the pre-existing trust cases thanks to:

```
source $MY_TRUST_FOLDER/env_TRUST.sh
trust -search "keyword_name"
```

Running a Data File

To use **TRUST**, your shell must be "bash". So ensure you are in the right shell:

```
echo $0
/bin/bash
```

To run your data file, you must initialize the **TRUST** environment using the following command:

```
source $my_path_to_TRUST_installation/env_TRUST.sh
TRUST vX.Y.Z support : trust@cea.fr
Loading personal configuration /$path_to_my_home_directory/.perso_TRUST.env
```

Then, you can launch sequentially your test case by using:

```
trust my_test_case.data
```

For more informations regarding parallel test case, go to the *Parallel Simulations* section.

Data file structure

In the sequel, let's dive into the structure of a `.data` file. You will find a full template of a `.data` file at the end of this page.

Dimension

First, you need to choose the dimension of your problem, it can be either a two or three dimension problem:

```
Dimension 2
```

or

```
Dimension 3
```

Problems

Then, you need to specify the type of problem you want to solve.

```
# Problem definition #
Pb_hydraulique my_problem
```

You can find all the available **TRUST** problems at *Problem types*.

Here are some of the available **TRUST** problem types.

- for Incompressible flow: **Pb_[Thermo]Hydraulic[_Concentration]**
- for Quasi-Compressible flow: **Pb_Thermohydraulique_QC**
- for Weakly-Compressible flow: **Pb_Thermohydraulique_WC**
- for Multi-Phase flow: **Pb_Multiphase**
- for solid: **Pb_Conduction**

where:

- **hydraulique**: means that we will solve Navier-Stokes equations without energy equation.
- **Thermo**: means that we will solve Navier-Stokes equations with energy equation.
- **Concentration**: that we will solve multiple constituent transportation equations.
- **Conduction**: resolution of the heat equation.
- **QC**: Navier-Stokes equations with energy equation for quasi-compressible fluid under low Mach approach.
- **WC**: Navier-Stokes equations with energy equation for weakly-compressible fluid under low Mach approach.

Domain Definition

To define the domain, you must name it. This is done thanks to the following block:

```
# Domain definition #
Domaine my_domain
```

Then you must add your mesh to your simulation.

Mesh

Notice the presence of the tags

```
# Mesh #
# BEGIN MESH #
... ;
# END MESH #
```

This is useful for parallel calculation if well placed in the datafile (see section *Parallel Simulations*).

Allowed meshes

TRUST allows all types of meshes if the appropriate spatial discretization is used. See the *Spatial schemes* section for the list of available discretizations and some advices regarding there usage or the *Spatial discretisations* section for more details regarding the undelying maths.

In your `.data` file, you can either import a pre-existing mesh or build one quickly if your geometry is not that complex.

Import a mesh file

If your mesh was generated with an external tool like SALOME (open source software), ICEM (commercial software), Gmsh (open source software, included in **TRUST** package) or Cast3M (CEA software), then you must use one of the following keywords into your data file:

- **Read_MED** for a MED file from SALOME or Gmsh.
- **Read_File** for a binary mesh file from ICEM.
- for another format, see the *TRUST Keyword Reference Manual*.

You can have a look too at the *Post-Processing* section.

Quickly create a mesh

Here is an example of a simple geometry (of non complex channel type) using the internal tool of **TRUST**:

```
Mailler my_domain
{
   # Define the domain with one cavity #
   # cavity 1m*2m with 5*22 cells #
   Pave box
   {
      Origine 0. 0.
      Longueurs 1 2

      # Cartesian grid #
      Nombre_de_Noeuds 6 23

      # Uniform mesh #
      Facteurs 1. 1.
   }
   {
      # Definition and names of boundary conditions #
      bord Inlet  X = 0.   0. <= Y <= 2.
      bord Outlet X = 1.   0. <= Y <= 2.
      bord Upper  Y = 2.   0. <= X <= 1.
      bord Lower  Y = 0.   0. <= X <= 1.
   }
}
```

To use this mesh in your data file, you just have to add the previous block in your data file or save it in a file named for example `my_mesh.geo` and add the line:

```
Read_file my_mesh.geo ;
```

Do not forget the semicolon at the end of the line!

Transform mesh within the data file

You can also make transformations on your mesh after the **"Mailler"** or **"Read_"** command, using the following keywords:

- **Trianguler** to triangulate your 2D cells and create an unstructured mesh
- **Tetraedriser** to tetrahedralise 3D cells and create an unstructured mesh
- **Raffiner_anisotrope** or **Raffiner_isotrope** to triangulate/tetrahedralise elements of an untructured mesh
- **ExtrudeBord** to generate an extruded mesh from a boundary of a tetrahedral or an hexahedral mesh ([doc here]

Note: ExtrudeBord in VEF generates 3 or 14 tetrahedra from extruded prisms.

- **RegroupeBord** to build a new boundary with several boundaries of the domain
- **Transformer** to transform the coordinates of the geometry

For more details regarding this commande, go find them in the *TRUST Keyword Reference Manual*. There, you will also find other commands, under the `interprete` section.

All theses keywords work on all mesh file formats (i.e. also for ***.geo** or ***.bin** or ***.med** files).

Test your mesh

The keyword **Discretiser_domaine** is useful to discretize the domain (faces will be created) without defining a problem. Indeed, you can create a minimal data file, in order to create and post-process your mesh in, for example, the lata format and visualize it with VisIt.

Note: you must name all the boundaries to descretize!

Here is an example of this kind of data file:

```
dimension 3
Domaine my_domain

Mailler my_domain
{
   Pave box
   {
      Origine 0. 0. 0.
      Longueurs 1 2 1
      Nombre_de_Noeuds 6 23 6
      Facteurs 1. 1. 1.
   }
   {
      bord Inlet X = 0. 0. <= Y <= 2. 0. <= Z <= 1.
      bord Outlet X = 1. 0. <= Y <= 2. 0. <= Z <= 1.
      bord Upper Y = 2. 0. <= X <= 1. 0. <= Z <= 1.
      bord Lower Y = 0. 0. <= X <= 1. 0. <= Z <= 1.
      bord Front Z = 0. 0. <= X <= 1. 0. <= Y <= 2.
      bord Back Z = 1. 0. <= X <= 1. 0. <= Y <= 2.
   }
}

discretiser_domaine my_domain
postraiter_domaine { domaine my_domain fichier file format lata }
End
```

To use it, launch in a bash terminal:

```
# Initialize TRUST env if not already done
source $my_path_to_TRUST_installation/env_TRUST.sh
# Run you data file
trust my_data_file
visit -o file.lata &
```

To see how to use VisIt, look at the first Quick start.

Spatial Discretization

You have to specify a discretization type to run a simulation. See the *Spatial schemes*. For example, if you want to use the **VDF** discretisation:

```
# Discretization on hexa or tetra mesh #
VDF my_discretization
```

Time Schemes

Now you can choose your time scheme to solve your problem. For this you must specify the time scheme type wanted and give it a name. then you have to specify its parameters by filling the associated **Read** block.

```
# Time scheme explicit or implicit #
Scheme_euler_explicit my_scheme
Read my_scheme
{
  # Initial time #
  # Time step #
  # Output criteria #
  # Stop Criteria #
}
```

Some available time schemes

The time schemes available in the **TRUST** platform are summarized in the *Temporal schemes* section.

Here are some available types of explicit schemes:

- **Scheme_Euler_explicit**
- **Schema_Adams_Bashforth_order_2**
- **Runge_Kutta_ordre_3**

And also some available types of implicit schemes:

- **Scheme_Euler_implicit**
- **Schema_Adams_Moulton_order_3**

For other schemes, see the `schematempsbas` section of the *TRUST Keyword Reference Manual*.

Note: you can treat implicitly the diffusion/viscous operators in a **TRUST** calculation. For that, you should activate the **diffusion_implicite** keyword in your explicit time scheme.

Calculation stopping condition

You must specify at least one stopping condition for you simulation in your time scheme block. It can be:

- the final time: **tmax**
- the maximal allowed cpu time: **tcpumax**
- the number of time step: **nb_pas_dt_max**
- the convergency treshold: **seuil_statio**

Note: if the time step reaches the minimal time step **dt_min**, **TRUST** will stop the calculation.

If you want to stop properly your running calculation (i.e. with all saves), you may use the `my_data_file.stop` file. When the simulation is running, you can see the **0** value in that file. To stop it, put a **1** instead of the **0**, save the file and at the next iteration the calculation will stop properly. When you don't change anything in that file, at the end of the calculation, you can see that it is writen **Finished correctly**.

Objects association and discretization

Association

Until now, we have created some objects, now we must associate them together. For this, we must use the **Associate** interpretor:

```
# Association between the different objects #
Associate my_problem my_domain
Associate my_problem my_time_scheme
```

Discretization

Then you must discretize your domain using the **Discretize** interpretor:

```
Discretize my_problem my_discretization
```

The problem *my_problem* is discretized according to the *my_discretization* discretization.

IMPORTANT: A number of objects must be already associated (a domain, time scheme, ...) prior to invoking the **Discretize** keyword. **Note:** when the discretization step succeeds, the mesh is validated by the code.

Read problem block

Now that you have the create and associate your object, and discrtised your equations, you will need to specify your problem:

```
# Problem description #
Read my_problem
{
```

More information regarding this block are given in the *Problem types*.

First, in this block, you will have to specify the type of equations you will solve. Your choices here depend on you problem choice, see the *TRUST Keyword Reference Manual*:

```
  # hydraulic problem #
  Navier_Stokes_standard
  {
```

Medium/Type of Fluide

Then you will specify the medium or fluid, you must add the following block.

```
    # Physical characteristics of medium #
    Fluide_Type { ... }
```

Fluid_type can be one of the following:

- **Fluide_incompressible**
- **Fluide_Quasi_compressible**
- **Fluide_Weakly_Compressible**
- **Solide**
- **Constituant**
- **Milieu_Composite**

For other types and more information see the *TRUST Keyword Reference Manual*.

Note: if you want to solve a coupled problem, each medium should be read in the corresponding problem.

Add Gravity

If needed, you can add a gravity term to your simulation. This is done by adding a uniform field, in the medium block since V1.9.1.

For example in 2D:

```
# Gravity vector definition #
Gravity Uniform_field 2 0 -9.81
```

Pressure solver

The, you need to specify your pressure solver:

```
# Choice of the pressure matrix solver #
          Solveur_Pression solver { ... }
```

Go check the *Solver and Preconditioners* section to see the options you may use.

Diffusion, convection and sources

Now you will specify how you will treat the diffusion operator, the convection operator and the source terms.

```
    # Diffusion operator #
    Diffusion { ... }

    # Convection operator #
    Convection { ... }

    # Sources #
    Sources { ... }
```

In the Diffusion block, you can but **negligeable** to deactivate the diffusion terme.

In the convection block, you need to specify a spatial scheme. Have a look at the *Spatial schemes* section for a list of the available schemes in **TRUST**.

In the sources block, you will define your source terms. You will need to go check the *TRUST Keyword Reference Manual* to pick up the good keyword for your case. Here are some available source terms:

- **Perte_Charge_Reguliere** type_perte_charge bloc_definition_pertes_charges
- **Perte_Charge_Singuliere KX | KY | KZ** coefficient_value { … }
- **Canal_perio** { … }
- **Boussinesq_temperature** { … }, defined as $\rho(T) = \rho(T_0)(1 - \beta_{th}(T - T_0))$
- **Boussinesq_concentration** { … }
-
- **Puissance_thermique** field_type bloc_lecture_champ

Initial and boundary conditions

To define the initial and boundary conditions, make sure to add the following in your **# hydraulic problem # Navier_Stokes_standard** block:

```
    # Initial conditions #
     Initial_conditions { ... }

     # Boundary conditions #
     Boundary_conditions { ... }
  }
```

This should conclude this block.

Here is a list of the most used boundary conditions:

- Bord **Frontiere_ouverte_vitesse_imposee** boundary_field_type *bloc_lecture_champ*
- Bord **Frontiere_ouverte_pression_imposee** boundary_field_type *bloc_lecture_champ*
- Bord **Paroi_fixe**
- Bord **Symetrie**
- Bord **Periodique**

- Bord **Frontiere_ouverte_temperature_imposee** boundary_field_type *bloc_lecture_champ*
- Bord **Frontiere_ouverte T_ext** boundary_field_type *bloc_lecture_champ*
- Bord **Paroi_adiabatique**
- Bord **Paroi_flux_impose** boundary_field_type *bloc_lecture_champ*

To choose your *boundary_field_type* parameters, refer to the *TRUST Keyword Reference Manual*.

Post-processing

Still in your **Read my_problem** block, you will need to add a postprocessing block in order to specify your post-processing parameters:

```
# Post_processing description #
# To know domains that can be treated directly, search in .err #
# output file: "Creating a surface domain named" #
# To know fields that can be treated directly, search in .err #
# output file: "Reading of fields to be postprocessed" #

Post_processing
{
   # Definition of new fields #
   Definition_Champs { ... }

   # Probes #
   Probes { ... }

   # Fields #
   # format default value: lml #
   # select 'lata' for VisIt tool or 'MED' for Salomé #
   format lata
   fields dt_post 1. { ... }

   # Statistical fields #
   Statistiques dt_post 1. { ... }
}
```

The post-processing options are listed in the *Post-Processing*

Stop and restart

If you want to stop or restart your computation, add the following after your post-processing block:

```
# Saving and restarting process #
[sauvegarde binaire datafile .sauv]
[resume_last_time binaire datafile .sauv]
# Don't forget to close your Read my_problem block #
}
```

Some more details regarding this step are given in *Stop and restart*.

Solve the problem

When you have specified everything else, you need to put the solving keyword at the end of your computation:

```
# The problem is solved with #
Solve my_problem
```

This is mandatory, if you forgot to put it, TRUST will not run the simulation.

Template of a `.data` file

Here is the template of a basic sequential data file:

```
# Dimension 2D or 3D #
Dimension 2

# Problem definition #
Pb_hydraulique my_problem

# Domain definition #
Domaine my_domain

# Mesh #
# BEGIN MESH #
Read_file my_mesh.geo ;
# END MESH #

# For parallel calculation only! #
# For the first run: partitioning step #
# Partition my_domain
{
   Partition_tool partitioner_name { option1 option2 ... }
   Larg_joint 2
   zones_name DOM
   ...
}
End #

# For parallel calculation only! #
# For the second run: read of the sub-domains #
# Scatter DOM .Zones my_domain #

# Discretization on hexa or tetra mesh #
VDF my_discretization

# Time scheme explicit or implicit #
Scheme_euler_explicit my_scheme
Read my_scheme
{
   # Initial time #
   # Time step #
   # Output criteria #
   # Stop Criteria #
```

(continues on next page)

(continued from previous page)

```
}

# Association between the different objects #
Associate my_problem my_domain
Associate my_problem my_scheme

# Discretization of the problem #
Discretize my_problem my_discretization

# New domains for post-treatment #
# By default each boundary condition of the domain is already extracted #
# with names such as "my_dom"_boundaries_"my_BC" #
Domaine plane
extraire_surface
{
   domaine plane
   probleme my_probleme
   condition_elements (x>0.5)
   condition_faces (1)
}

# Problem description #
Read my_problem
{

   {
      # hydraulic problem #
      Navier_Stokes_standard
      {
         # Physical characteristics of medium #
         Fluide_Incompressible
         {
         ...
         # Gravity vector definition #
         gravity Uniform_field 2 0 -9.81
         }

         # Choice of the pressure matrix solver #
         Solveur_Pression solver { ... }

         # Diffusion operator #
         Diffusion { ... }

         # Convection operator #
         Convection { ... }

         # Sources #
         Sources { ... }

         # Initial conditions #
         Initial_conditions { ... }
```

(continues on next page)

(continued from previous page)

```
         # Boundary conditions #
         Boundary_conditions { ... }
      }

      # Post_processing description #
      # To know domains that can be treated directly, search in .err #
      # output file: "Creating a surface domain named" #

      # To know fields that can be treated directly, search in .err #
      # output file: "Reading of fields to be postprocessed" #

      Post_processing
      {
         # Definition of new fields #
         Definition_Champs { ... }

         # Probes #
         Probes { ... }

         # Fields #
         # format default value: lml #
         # select 'lata' for VisIt tool or 'MED' for Salomé #
         format lata
         fields dt_post 1. { ... }

         # Statistical fields #
         Statistiques dt_post 1. { ... }
      }

      # Saving and restarting process #
      [sauvegarde binaire datafile .sauv]
      [resume_last_time binaire datafile .sauv]

   # End of the problem description block #
   }

# The problem is solved with #
Solve my_problem

# Not necessary keyword to finish #
End
```

1.1.3 Problem types

Depending on the type of problem you choose, **TRUST** will solve different sets of equation. It is therefore key to identify which problem corresponds to your application.

Inside the problem bloc, the user have to define first the medium he wants to consider. The **medium specified in the bloc should be coherent with the instantiated problem**, otherwise you will not be able to run your test case. For example, defining an incompressible medium in a conduction problem is not possible.

After defining the medium, the user should precise the equations he is willing to solve. The **equation should be also coherent with the instantiated problem** otherwise **TRUST** will exit with an error message. For example, defining a Navier-Stokes equation in a conduction problem is not possible.

Once the problem, medium and equations are defined, the user will have to specify the post-processing of the current case. Go check-out the *Post-Processing* section for more informations regarding the post-processing in **TRUST**.

In this section, the most used problems are introduced. For documentation and for complete problem sets, see the *TRUST Keyword Reference Manual*.

Conduction Problem

For this kind of problems, **TRUST** solves the heat equation:

$$\rho C_p \frac{\partial T}{\partial t} = \nabla \cdot (\lambda \nabla T) + Q \tag{1.1.1}$$

where:

- ρ: density,
- C_p: specific heat capacity at constant pressure,
- λ: thermal conductivity,
- Q is a heat source term.

Options of Conduction problems

Problem type	Possible mediums	Possible Equations
Pb_Conduction	Solide	Conduction

Example of a Pb_conduction block

Here is an example of a **Pb_conduction**, taken from the `docond_VEF.data` file:

```
# Read a conduction problem pb #
        Read pb
        {
                # Read solid medium #
            Solide
            {
                rho Champ_Uniforme 1 1000.
                lambda Champ_Uniforme 1 250.
                Cp Champ_Uniforme 1 100.
```

(continues on next page)

(continued from previous page)

```
        }

            # Read conduction equation #
        Conduction
        {
            diffusion { }
            initial_conditions { temperature Champ_Uniforme 1 30. }
            boundary_conditions
            {
                paroi_a_40 paroi_temperature_imposee Champ_Front_Uniforme 1 40.
                paroi_a_20 paroi_temperature_imposee Champ_Front_Uniforme 1 20.
                Paroi_echange paroi_contact pb2 Paroi_echange
            }
        }

        # Read post-processing #
        Post_processing
        {
            Probes
            {
                sonde_tsol temperature periode 1. points 2 0.15 0.55 0.55 0.15
                sonde_segs temperature periode 5. segment 10 0. 0.75 0.3 0.75
            }
            fields dt_post 300.
            {
                temperature elem
            }
        }

        # Save the data at the end of the calculation to resume later #
        sauvegarde formatte solide.rep
    }
```

Hydraulic and Thermohydraulics problems

The hydraulic and thermohydraulic problems solve the **incompressible Navier-Stokes** equation with more or less terms. Complementary transport equations can also be added.

$$\begin{cases} \nabla \cdot \vec{u} = 0 \\ \dfrac{\partial \vec{u}}{\partial t} + \nabla \cdot (\vec{u} \otimes \vec{u}) + \nabla P^* = \nabla \cdot (\nu \nabla \vec{u}) \end{cases} \tag{1.1.2}$$

Incompressible Navier-Stokes equation, referred as Navier_Stokes_standard in **TRUST**

$$\frac{\partial T}{\partial t} + \vec{u}\nabla T = \nabla \cdot (\alpha \nabla T) + \frac{Q}{\rho C_p} \tag{1.1.3}$$

Heat equation, referred as Convection_Diffusion_Temperature in **TRUST**

$$\frac{\partial X}{\partial t} + \vec{u}\nabla X = 0 \tag{1.1.4}$$

Passive scalar transport equation, referred as Convection_Diffusion_Concentration in **TRUST**

Where: $P^* = \dfrac{P}{\rho} + gz$, Q is the heat source term, and:

- ρ: density
- μ: dynamic viscosity
- $\nu = \frac{\mu}{\rho}$: kinematic viscosity $\vec{g} = gz$: gravity vector in cartesian coordinates
- $\alpha = \frac{\lambda}{\rho C_p}$: thermic diffusivity
- C_p: specific heat capacity at constant pressure
- λ: thermic conductivity
- X: a transported scalar quantity

Note

Red terms corresponds to convective terms and blue terms to diffusive terms.

Options of Hydraulic and Thermohydraulics problems

Problem type	Possible mediums	Possible Equations
Pb_Hydraulique	Fuilde_Incompressible or Fluide_Ostwald	Navier_Stokes_standard
Pb_Hydraulique_Concentratio	Fuilde_Incompressible or Fluide_Ostwald	Navier_Stokes_standard + Convection_Diffusion_Concentration
Pb_Hydraulique_Concentratio	Fuilde_Incompressible or Fluide_Ostwald	Navier_Stokes_standard + N additional Convection_Diffusion_Concentration
Pb_Thermohydraulique	Fuilde_Incompressible or Fluide_Ostwald	Navier_Stokes_standard + Convection_Diffusion_Temperature
Pb_Thermohydraulique_Conc	Fuilde_Incompressible or Fluide_Ostwald	Navier_Stokes_standard + Convection_Diffusion_Temperature + Convection_Diffusion_Concentration
Pb_Thermohydraulique_Conc	Fuilde_Incompressible or Fluide_Ostwald	Navier_Stokes_standard + Convection_Diffusion_Temperature + N additional Convection_Diffusion_Concentration

Important remarks:

- for Hydraulique or Thermohydraulique problems, an additional `Constituant` medium should be **secondly** defined if you ask for a problem with concentration.

Example of a Pb_Thermohydraulique block

Here is an example of Pb_Thermohydraulique, taken from the `Source_Generique_PolyMAC` test case:

```
# Read a thermal hydraulic problem pb #
Read pb
{
        # Read incompressible medium #
        fluide_incompressible
```

(continues on next page)

(continued from previous page)

```
        {
            gravite champ_uniforme 2 0 -9.81
            mu Champ_Uniforme 1 1.85e-5
            rho Champ_Uniforme 1 1.17
            lambda Champ_Uniforme 1 0.0262
            Cp Champ_Uniforme 1 1006
            beta_th Champ_Uniforme 1 3.41e-3
        }

        # Read NS equation #
        Navier_Stokes_standard
        {
            solveur_pression petsc gcp { precond null { } seuil 1e30 }
            convection { negligeable }
            diffusion { negligeable }
            initial_conditions { vitesse Champ_Uniforme 2 0. 0. }
            boundary_conditions
            {
                Obstacle paroi_fixe
                Symetrie symetrie
                Sortie frontiere_ouverte_pression_imposee Champ_front_Uniforme 1 0.
                Entree frontiere_ouverte_vitesse_imposee  Champ_front_Uniforme 2  1. 0.
            }
            sources
            {
                Source_Qdm Champ_fonc_xyz dom 2 x y
            }
        }

        # Read Temperature equation #
        Convection_Diffusion_Temperature
        {
            diffusion { negligeable }
        convection { negligeable }
        boundary_conditions
        {
            Symetrie         paroi_adiabatique
            Obstacle         paroi_adiabatique
            Entree          frontiere_ouverte_temperature_imposee Champ_front_Fonc_xyz 1␣
↪1
            Sortie        frontiere_ouverte_temperature_imposee Champ_front_Fonc_xyz 1 0
        }
        initial_conditions { Temperature Champ_Fonc_xyz dom 1 0 }
        sources
        {
            Puissance_Thermique champ_fonc_xyz dom 1 50+x+y
        }
    }

         # Read post-processings #
    Postraitement
    {
```

(continues on next page)

(continued from previous page)

```
        fields dt_post 1e10
        {
            temperature elem
            vitesse elem
            pressure elem
        }
    }
}
```

Low Mach Number Problems

The Mach number $Ma = u/c$ measures the ratio of the bulk velocity to the local speed of sound. Typically for $Ma < 0.1$, the compressibility effects can be neglected and the physical problem may be reduced by using a Low Mach Number formulation. In such cases, the aspect of the problem orients towards simulating a mass variation rather than capturing and solving for the acoustic waves. This is basically the main hypothesis of a Low Mach Number approximation where all acoustic waves are filtered out.

Following a single scale asymptotic analysis referring to the Mach number, the zero Mach order equations are considered where the total pressure is decomposed into a thermodynamic pressure and a hydrodynamic pressure that fluctuates in an order of Ma^2. Note that the hydrodynamic pressure is much smaller than the thermodynamic one. The hydrodynamic pressure alone is used in the momentum equation, while the thermodynamic is used in the equation of state.

TRUST offers two Low Mach Number modelisations; either a Weakly-Compressible (WC) or a Quasi-Compressible (QC) models. The main difference between both models is that the QC model considers a space-uniform thermodynamic pressure. This is not the case in the WC model where the total pressure, which is space/time varying, is used in the equation of state. The last can have a great impact in some situations as the pressure drop and/or the hydro-static pressure can influence significantly the density variation.

The system of equation associated with Low Mach number problem writes:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \\ \dfrac{\partial \rho \vec{u}}{\partial t} + \nabla \cdot (\rho \vec{u} \vec{u}) = \nabla \cdot (\mu \nabla \vec{u}) - \nabla P - \rho \vec{g} \\ \rho C_p \left(\dfrac{\partial T}{\partial t} + \vec{u} \nabla T \right) = \nabla \cdot (\lambda \nabla T) + \dfrac{dP_{tot}}{dt} + Q \end{cases}$$

where: $P_{tot} = \rho R T$, Q is a heat source term, and:

- ρ: density,
- μ: dynamic viscosity,
- $\vec{g} = gz$: gravity vector in cartesian coordinates,
- C_p: specific heat capacity at constant pressure,
- λ: thermal conductivity.

Note

Red terms are convective terms and blue terms are diffusive terms.

Remark: The difference between *Hydraulique* and *Thermohydraulique* type of problem is whether the last equation of the system (that drive the temperature) is taken into account or not. If it is taken into account you'll have to use a *Thermohydraulique* type of problem.

Valid `.data` options for WC and QC problems

Problem type	Possible mediums	Possible Equations
Pb_Hydraulique_Melang	Fluide_Quasi_Compr	Navier_Stokes_QC + Convection_Diffusion_Espece_Binaire_QC
Pb_Hydraulique_Melang	Fluide_Quasi_Compr	Navier_Stokes_WC + Convection_Diffusion_Espece_Binaire_WC
Pb_Thermohydraulique_	Fluide_Quasi_Compr	Navier_Stokes_QC + Convection_Diffusion_Chaleur_QC
Pb_Thermohydraulique_	Fluide_Quasi_Compr	Navier_Stokes_WC + Convection_Diffusion_Chaleur_WC
Pb_Thermohydraulique_	Fluide_Quasi_Compr	Navier_Stokes_QC + Convection_Diffusion_Chaleur_QC + N additional Convection_Diffusion_Espece_Multi_QC
Pb_Thermohydraulique_	Fluide_Quasi_Compr	Navier_Stokes_WC + Convection_Diffusion_Chaleur_WC + N additional Convection_Diffusion_Espece_Multi_WC

Example of a Pb_Thermohydraulique block

Here is an example of **Pb_thermohydraulique_qc**, taken from the `DarcyFlow_QC_VDF` test case:

```
Read pb
{

    fluide_quasi_compressible {
        pression 111582.205714
        loi_etat gaz_parfait_QC
        {
            Cp 1006.8
            gamma 1.4
            Prandtl 0.494112177122
            rho_constant_pour_debug champ_uniforme 1 1.293
        }
        traitement_pth constant
        traitement_rho_gravite moins_rho_moyen
        mu champ_uniforme 1 1.33e-05
    }

    Navier_stokes_qc
    {
        solveur_pression petsc Cholesky { }
        convection { negligeable }
        diffusion { negligeable }
        initial_conditions
        {
```

(continues on next page)

(continued from previous page)

```
            vitesse champ_uniforme 3 0.0 0.2 0.0
        }
        boundary_conditions
        {
            Entree frontiere_ouverte_vitesse_imposee champ_front_uniforme 3 0.0 0.2 0.0
            Sortie frontiere_ouverte_pression_imposee champ_front_uniforme 1 0.0
            Bord paroi_fixe
        }
        sources
        {
            darcy { porosite 0.4 modele_K ErgunDarcy { diametre 1.5e-3 } } ,
            forchheimer { porosite 0.4 Cf 1. modele_K ErgunForchheimer { diametre 1.5e-3␣
↪} }
        }
    }
    convection_diffusion_chaleur_qc
    {
        convection { negligeable }
        diffusion { negligeable }
        initial_conditions
        {
            temperature champ_uniforme 1 300.0
        }
        boundary_conditions
        {
            Entree frontiere_ouverte_temperature_imposee champ_front_uniforme 1 300.0
            Sortie frontiere_ouverte T_ext champ_front_uniforme 1 300.0
            Bord paroi_adiabatique
        }
    }
    Post_processing
    {
        Probes
        {
            sonde_vitesse                nodes         vitesse periode 0.0001 point 1 0.
↪08 0.10 0.08
            segment_pression         grav         pression_pa periode 0.0001 segment 50␣
↪0.08 0.05 0.08 0.08 0.20 0.08
        }
        Format lata fields dt_post 0.1
        {
            pression_pa elem
        }
    }
}
```

Multi-Phase Problem

The multiphase type of prblem is simply called `Pb_Multiphase`. This problem allows the resolution of N-phases via a model of 3N equations, based on a Ishii types of model :cite:`I75`.

An instance of the medium `Milieu_composite` is required at the begining of the problem's bloc. Inside the medium bloc, you can define any medium. For exmaple, you can define a `Fluide_Incompressible` or any child class of the `Fluide_reel_base` class (cf C++ API).

An additional `Correlations` bloc should be **secondly** defined. Each and every correlations derive from the base class `Corellation_base`. Check the C++ API to see the list of available correlations.

Afterwards, the reading of the equations starts. In `Pb_Multiphase`, the equations solved for each phase are the momentum, mass and energy equations; respectively `QDM_Multiphase`, `Masse_Multiphase` and `Energie_Multiphase`.

The coupling between all equations is done in a strong way: a single matrix for all equations is used to solve the problem. Available discretisations for this type of problem are **VDF**, **PolyMAC** and **PolyMAC_P0**.

You can also call the **EOS** (private CEA/EDF project) and the **CoolProp** library to compute the Thermo-Physical Properties via the `TPPI` interface. Check this tutorial for a guide to link CoolProp with **TRUST**.

Example of a Pb_Multiphase block

Here is an example of **Pb_Multiphase**, taken from the `CoolProp_water_BICUBIC_HEOS_with_sat` test case. **Warning**: this test case needs a TRUST version linked with the CoolProp library.

```
# Read a multiphase problem pb #
Read pb
{
        # Read the multi-phase + saturation medium #
        Milieu_composite
        {
                liquide_eau Fluide_generique_coolprop { model BICUBIC&heos fluid Water␣
↪phase liquid }
                gaz_eau Fluide_generique_coolprop { model BICUBIC&heos fluid Water }
                saturation_eau saturation_generique_coolprop { model BICUBIC&heos fluid␣
↪Water phase liquid }
        }

        # Read the correlations #
        correlations
        {
        flux_interfacial coef_constant { liquide_eau 1.0e10 gaz_eau 1.0e10 }
        }

    # Read the momentum equation #
    QDM_Multiphase
    {
        evanescence { homogene { alpha_res 1 alpha_res_min 0.5 } }
        solveur_pression petsc cli_quiet { -pc_type hypre -pc_hypre_type boomeramg }
        convection { amont }
        diffusion  { negligeable }
        initial_conditions
        {
           vitesse  Champ_fonc_xyz dom 4 2.0*(x>0.5)-2.0*(x[0.5) 2.0*(x>0.5)-2.0*(x[0.5)␣
```

(continues on next page)

(continued from previous page)

```
↪0.0 0.0
          pression Champ_fonc_xyz dom 1 1.0e5
       }
       boundary_conditions
       {
           haut symetrie
           bas symetrie
            gauche frontiere_ouverte_pression_imposee champ_front_uniforme 1 100000.0
           droite frontiere_ouverte_pression_imposee champ_front_uniforme 1 100000.0
       }
   }

   # Read the mass equuation #
   Masse_Multiphase
   {
       initial_conditions { alpha Champ_Fonc_xyz dom 2 0.95 0.05 }
       convection { amont }
       boundary_conditions
       {
           haut paroi
           bas paroi
           gauche frontiere_ouverte a_ext Champ_Front_Uniforme 2 0.95 0.05
           droite frontiere_ouverte a_ext Champ_Front_Uniforme 2 0.95 0.05
       }
         sources { flux_interfacial }
   }

   # Read the energy equation #
       Energie_Multiphase
   {
       diffusion { negligeable }
       convection { amont }
       initial_conditions { temperature Champ_fonc_xyz dom 2 10. 10. }
       boundary_conditions
       {
           haut paroi_adiabatique
           bas paroi_adiabatique
           gauche frontiere_ouverte T_ext Champ_Front_uniforme 2 81.578 61.578
           droite frontiere_ouverte T_ext Champ_Front_uniforme 2 71.578 51.578
       }
        sources { flux_interfacial }
   }

   # Read the post-processing #
   Post_processing
   {
       probes
       {
            rho grav masse_volumique periode 1e8 segment 1000 0 0.5 1 0.5
                  v grav         vitesse periode 1e8 segment 1000 0 0.5 1 0.5
                  p grav        pression periode 1e8 segment 1000 0 0.5 1 0.5
           eint grav energie_interne periode 1e8 segment 1000 0 0.5 1 0.5
```

(continues on next page)

(continued from previous page)

```
        }

        Format lml
        fields dt_post 1e-4
        {
            alpha elem
            vitesse elem
            pression elem
            temperature elem
            energie_interne elem
            vitesse_liquide_eau elem
            vitesse_gaz_eau elem
            alpha_gaz_eau elem
            masse_volumique elem
        }
    }
}
```

Coupled Problems

There are two ways to solve coupled problems with TRUST: either using `Probleme_Couple` or via the Interface of Code Coupling (**ICoCo**) and `ProblemeTrio`.

With Probleme_Couple

In `Probleme_Couple`, the user has to define the contact boundary/volume between several domains where a **TRUST** problem (one of the above) is to be defined on each domain. The coupling is managed by **TRUST** where a point-fixed algorithm is used to converge the coupled simulation.

Here is an example:

- First two problems *my_problem_1* and *my_problem_2* are defined. Each one associated with a separate domain *my_domain_1* and *my_domain_2*, and each one having a different medium *my_medium_1* and *my_medium_2*:

```
Dimension 2
Pb_ThermoHydraulique my_problem_1
Pb_ThermoHydraulique my_problem_2

Domaine my_domain_1
Read_file my_mesh_1.geo ;

Domaine my_domain_2
Read_file my_mesh_2.geo ;

Associate my_problem_1 my_domain_1
Associate my_problem_2 my_domain_2
```

- Then a coupled problem associated to a single time scheme is defined:

```
Probleme_Couple my_coupled_problem

```

(continues on next page)

(continued from previous page)

```
VEFPreP1B my_discretization

Scheme_euler_explicit my_scheme
Read my_scheme { ... }

Associate my_coupled_problem my_problem_1
Associate my_coupled_problem my_problem_2
Associate my_coupled_problem my_scheme
```

- Eventually the coupled problem is discretized, the two separated problems specified and the coupled problem is solved:

```
Discretize my_coupled_problem my_discretization

Read my_problem_1
{
   Fluide_Incompressible { ... }
   ...
}

Read my_problem_2
{
   Fluide_Incompressible { ... }
   ...
}

Solve my_coupled_problem
End
```

With ICoCo

On the other hand, the user have to write his own coupling algorithm when using ICoCo. In practice, this can be done either by writing a C++ supervisor to manage the coupling, or via its python interface available in **TRUST**, thanks to the swig wrapper !

1.1.4 Stop and restart

An important feature when running a simulation is the ability to stop and restart it. This section explain how you can do it with **TRUST**.

Stop a Running Calculation

When using **TRUST**, your calculation will automatically stop if it has reached:

- the end of the calculation time.
- the maximal allowed cpu time.
- the maximal number of iterations.
- the threshold of convergence.

You may use the `my_data_file.stop` file, if you want to stop properly your running calculation (i.e. with all saves).

When the simulation is running, you can see the **0** value in that file. To stop it, put a **1** instead of the **0** and at the next iteration the calculation will stop properly.

When you don't change anything to that file, at the end of the calculation, you can see that it is writen **Finished correctly**.

Save

TRUST makes automatic backups during the calculation. The unknowns (velocity, temperature,...) are saved in:

- one **.xyz** file, written:
- at the end of the calculation.
- but, user may disable it with the specific keyword **EcritureLectureSpecial 0** added just before the **Solve** keyword.
- one (or several in case of parallel calculation) **.sauv** files, written:
- at the start of the calculation.
- at the end of the calculation.
- each 23 hours of CPU, to change it, uses **periode_sauvegarde_securite_en_heure** keyword (default value 23 hours).
- user may also specify a time physical period with **dt_sauv** keyword.
- periodically using **tcpumax** keyword for which calculation stops after the specified time (default value 10^{30}). Use it for calculation on CCRT/TGCC and CINES clusters for example.

> **Note**
>
> By default, the name for the **.sauv** files is **filename_problemname.sauv** for sequential calculation, **filename_problemname_000n.sauv** for parallel calculation (one per process). The format of theses files is binary and the files are completed during successive saves.

You can change the behaviour using the following keywords just before the **solve** instruction:

```
sauvegarde binaire|xyz filename .sauv|filename .xyz
```

with **xyz**: the **.xyz** file is written instead of the **.sauv** files.

> **Note**
>
> You can use **sauvegarde_simple** instead of **sauvegarde** where the .sauv or .xyz file is deleted before saves, in order to keep disk space:

```
sauvegarde_simple binaire|xyz filename .sauv|filename .xyz
```

For more information, go see the *TRUST Keyword Reference Manual*.

Resume

To resume your calculation, you may:

- change your initial time, the new initial time will be the real final calculation time of the previous calculation (see the .err file).
- change your final calculation time to the new wanted value.
- add the following block just before the **Solve** keyword:

```
reprise binaire|xyz filename .sauv|filename .xyz
```

> **Note**
>
> Instead of **reprise** keyword, you can use **resume_last_time** where **tinit** is automatically set to the last time of saved files (but you may change **tmax**):
>
> ```
> resume_last_time binaire|xyz filename .sauv|filename .xyz
> ```

You can resume your calculation:

- from .sauv file(s) (one file per process): you can only resume the calculation with the **same number of equations** on **the same number of processes**.
- or from a .xyz file: here you can resume your calculation by **changing the number of equations solved** and/or with a **different number of processes**.

> **Note**
>
> You can run a calculation with initial condition read into a save file (.xyz or .sauv) from a previous calculation using **Champ_Fonc_reprise** or read a into a MED file with **Champ_Fonc_MED**.

1.1.5 Boundary conditions

There is no sense to talk about numerical simulations without talking about the boundary conditions ! In order to close the discretized the system, consistent boundary conditions must be defined.

The platform **TRUST** implements a huge number of boundary conditions (BC). Here is a short summary of the available classes aliases that can be used to define your BC's in a **TRUST** simulation.

Dirichlet-type Boundary Conditions

Fluid Inlet boundary conditions

Boundary Condition Keyword	Description
Frontiere_ouverte_alp	Imposed void fraction condition at an open boundary called *bord* (french for edge) (corresponding to a fluid inlet). This condition must be associated with an imposed inlet velocity condition.
Frontiere_ouverte_co	Imposed concentration condition at an open boundary (corresponding to a fluid inlet). This condition must be associated with an imposed inlet velocity condition.
Frontiere_ouverte_fra	Imposed mass fraction condition at an open boundary (corresponding to a fluid inlet). This condition must be associated with an imposed inlet velocity condition.
Frontiere_ouverte_ten	Imposed temperature condition at the open boundary (in the case of fluid inlet). This condition must be associated with an imposed inlet velocity condition. The imposed temperature value is expressed in C or K.
Entree_temperature_	Particular case of class frontiere_ouverte_temperature_imposee for enthalpy equation.
Frontiere_ouverte_vit	Class for velocity-inlet boundary condition. The imposed velocity field at the inlet is vectorial and the imposed velocity values are expressed in m.s-1.
Frontiere_ouverte_rhc	This keyword is used to designate a condition of imposed mass rate at an open boundary . The imposed mass rate field at the inlet is vectorial and the imposed velocity values are expressed in kg.s-1. This boundary condition can be used only with the dilatable model (Low Mach Number).

Dirichlet boundary conditions

Boundary Condition Keyword	Description
Dirichlet_loi_paroi	Used for Multiphase Problem with wall laws.
Paroi_defilante	Keyword to designate a condition where tangential velocity is imposed on the wall . If the velocity components set by the user is not tangential, projection is used.
Paroi_Knudser	Boundary condition for a Knudsen number (Kn) above 0.001 where slip-flow condition appears: the velocity near the wall depends on the shear stress : Kn = l/L with l is the mean-free-path of the molecules and L a characteristic length scale.
Scalaire_impos	Imposed temperature condition at the wall.
Temperature_imposee_	Imposed temperature condition at the wall.
Paroi_fixe	Keyword to designate a situation of adherence to the wall (normal and tangential velocity at the edge is zero).
Paroi	Impermeability condition at a wall (standard flux zero). This condition must be associated with a wall type hydraulic condition (Dirichlet_paroi_fixe).
Paroi_Tempera	Imposed temperature condition at the wall.

Navier boundary conditions

Boundary Condition Keyword	Description
Frottement_exter	External friction BC used in Multi-Phase problems.
Frottement_globa	Global friction BC used in Multi-Phase problems.
Symetrie	For Navier-Stokes equations, this keyword is used to designate a symmetry condition concerning the velocity at the boundary (normal velocity at the edge equal to zero and tangential velocity gradient at the edge equal to zero). For scalar transport equation, this keyword is used to set a symmetry condition on scalar on the boundary.

Periodic boundary conditions

Boundary Condition Keyword	Description
Periodique	For Navier-Stokes equations, this keyword is used to indicate that the horizontal inlet velocity values are the same as the outlet velocity values, at every moment. As regards meshing, the inlet and outlet edges bear the same name. For scalar transport equation, this keyword is used to set a periodic condition on scalar. The two edges dealing with this periodic condition bear the same name.

Neumann-type Boundary Conditions

Wall boundary conditions

Boundary Condition Keyword	Description
Neumann_paroi_adiabatique	Adiabatic wall neumann boundary condition.
Paroi_adiabatique	Normal zero flux condition at the wall.
Paroi_flux_impose	Normal flux condition at the wall. The surface area of the flux (W.m-1 in 2D or W.m-2 in 3D) is imposed at the boundary.

Outlet boundary conditions

Boundary Condition Keyword	Description
Neumann_sortie_lib	Open boundary for heat equation with enthalpy as unknown.
Frontiere_ouverte_G	Normal imposed pressure gradient condition on the open boundary. This boundary condition may be only used in VDF discretization. The imposed pressure gradient value is expressed in Pa.m-1.
Frontiere_ouverte_pr	Imposed pressure condition at the open boundary. The imposed pressure field is expressed in Pa.
Frontiere_ouverte_P	This boundary condition may only be used with VDF discretization. There is no reference for pressure for this boundary condition so it is better to add pressure condition (with Frontiere_ouverte_pression_imposee) on one or two cells (for symetry in a channel) of the boundary where Orlansky conditions are imposed.
Frontiere_ouverte_G	Class for outlet boundary condition in VEF like Orlansky. There is no reference for pressure for theses boundary conditions so it is better to add pressure condition (with Frontiere_ouverte_pression_imposee) on one or two cells (for symmetry in a channel) of the boundary where Orlansky conditions are imposed.
Frontiere_ouverte_pr	Class for open boundary with pressure mean level imposed.
Frontiere_ouverte_vi	Sub-class for velocity boundary condition. The imposed velocity field at the open boundary is vectorial and the imposed velocity values are expressed in m.s-1.

Coupling-type Boundary Conditions

Boundary Condition Keyword	Description
Echa	Thermal coupling boundary condition.
Paroi	Thermal condition between two domains. Important: the name of the boundaries in the two domains should be the same. (Warning: there is also an old limitation not yet fixed on the sequential algorithm in VDF to detect the matching faces on the two boundaries: faces should be ordered in the same way). The kind of condition depends on the discretization. In VDF, it is a heat exchange condition, and in VEF, a temperature condition. Such a coupling requires coincident meshes for the moment. In case of non-coincident meshes, run is stopped and two external files are automatically generated in VEF (`connectivity_failed_boundary_name` and `connectivity_failed_pb_name.med`). In 2D, the keyword Decouper_bord_coincident associated to the `connectivity_failed_boundary_name` file allows to generate a new coincident mesh. In 3D, for a first preliminary cut domain with HOMARD (fluid for instance), the second problem associated to pb_name (solide in a fluid/solid coupling problem) has to be submitted to HOMARD cutting procedure with `connectivity_failed_pb_name.med`.
Paroi	This keyword is derivated from paroi_contact and is especially dedicated to compute coupled fluid/solid/fluid problem in case of thin material. Thanks to this option, solid is considered as a fictitious media (no mesh, no domain associated), and coupling is performed by considering instantaneous thermal equilibrium in it (for the moment).
Echa	Boundary condition type to model the heat flux between two problems. Important: the name of the boundaries in the two problems should be the same.

Robin-type Boundary Conditions

Boundary Condition Keyword	Description
Echange_exte	External type exchange condition with a heat exchange coefficient and an imposed external temperature.
Echange_exte	Particular case of class paroi_echange_externe_impose for enthalpy equation.
Echange_inte	Internal heat exchange boundary condition with exchange coefficient.
Echange_inte	Internal heat exchange boundary condition with perfect (infinite) exchange coefficient.
Echange_glob	Global type exchange condition (internal) that is to say that diffusion on the first fluid mesh is not taken into consideration.
Echange_cont	Class to define a thermohydraulic 1D model which will apply to a boundary of 2D or 3D domain.
Echange_inte	Internal heat exchange boundary condition with global exchange coefficient.
Echange_inte	Internal heat exchange boundary condition with perfect (infinite) exchange coefficient.
Robin_VEF	Available only with the VEF Pnc-P0 discretisation. Robin flux term for Navier-Stokes equations. Uses Robin coefficients `alpha` (normal) and `beta` (tangential), and a field `flux_normal_et_tangentiel` (concatenation of normal and tangential flux components).

1.1.6 Solver and Preconditioners

Resolution of large sparse linear systems of the form of **AX = B** are required at two stages in resolution algorithm:

- when dealing with the Elliptic pressure Poisson equation,
- when dealing with the implicit resolution at second (either diffusion implicit or a full implicit time integration scheme)

The platform TRUST allows the use of a wide range of solvers/preconditioners in order to solve such linear systems. Two types of solvers are mainly available: either part of the TRUST code ones or called from external libraries, for example from the open source **PETSc** library. As stated previously, TRUST has been ported recently to support GPU acceleration (NVidia/AMD). The solvers were the first parts to be ported to GPU as they represent most of the computational time.

In cases where the user asks to treat implicitly the diffusion operator in an explicit time integration scheme, a Preconditioned Conjugate Gradient (GCP) solver will be used (by default) to solve the implicit matrix. However, it is possible to select a specific solver. These instances are optional and can be inserted in the bloc of each equation.

If a pure implicit scheme is used in a TRUST calculation, an implicit solver **must** be defined too. This is done via the keyword **Solveur**.

Linear Solvers

External solvers

External Library	Solver	Description
PETSc	GCP	Preconditioned Conjugate Gradient.
PETSc	GMRES	Generalized Minimal Residual.
PETSc	BICGSTAI	Stabilized Bi-Conjugate Gradient.
PETSc	CHOLESK	Parallel LU or Cholesky decomposition
PETSc	CHOLESK	Sequential Cholesky from UMFPACK library (seems fast).
PETSc	CLI	Command Line Interface. Should be used only by advanced users, to access the whole solver/preconditioners from the PETSC API. To find all the available options, run your calculation with the -ksp_view -help options (Krylov Method Options).
AmgX	AMGX	GPU solver via AmgX API, for Nvidia only.
PETSc GPU	PETSc_GP	GPU solver via PETSc API.
cuDSS	Cholesky or LU	GPU direct solver for Nvidia only.

Direct TRUST internal solvers

Solver	Description
Cholesky	Cholesky direct method.

Iterative TRUST internal solvers

Solver	Description
GCP	Preconditioned Conjugate Gradient.
GMRES	Gmres method (for non symetric matrix).
GEN	Generic solver.

Preconditioners

External preconditioners

Preconditioner	Parameters	Description
NULL { }	-	No preconditioner used.
DIAG { }	-	Diagonal (Jacobi) preconditioner.
BOOMERAMG { }	-	Multigrid preconditioner (no option is available yet, look at CLI command and Petsc documentation to try other options).
BLOCK_JAC { level k ordering natural | rcm }	level k (default: 1), ordering: natural or rcm	Incomplete Cholesky factorization for symmetric matrix with the PETSc implementation. The integer k is the factorization level (default value, 1). In parallel, the factorization is done by block (one per processor by default). The ordering of the local matrix is natural by default, but rcm ordering, which reduces the bandwith of the local matrix, may interestingly improves the quality of the decomposition and reduces the number of iterations.
SSOR { omega double }	omega (default: 1.5)	Symmetric Successive Over Relaxation algorithm. omega (default value, 1.5) defines the relaxation factor.

TRUST internal preconditioners

Preconditioner	Description
ILU	Can be only used with the generic GEN solver.
SSOR	Symmetric successive over-relaxation algorithm.

Implicit Solvers

Linear Solvers

Solver	Description
Solveur_lineaire_std	Standard linear solver.

Non-Linear Solvers

Solver	Full Name	Description
Implicite	-	Similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps. But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains.
Simple	Semi-Implicit Method for Pressure Linked Equations (SIMPLE)	Type algorithm, see **CT Shaw**.
Simpler	Semi-Implicit Method for Pressure Linked Equations Revised (SIMPLER)	Method for incompressible systems, see **DS Jang**.
Piso	Pressure Implicit with Split Operator (PISO)	A method to solve the implicit Navier-Stokes equation, see **RI Issa**.
SETS	Stability-Enhancing Two-Step (SETS)	Solver which is useful for a multiphase problem, see **JH Mahaffy**.
ICE	Implicit Continuous-fluid Eulerian	Solver which is useful for a multiphase problem. Robust pressure reduction resolution. See **FH Harlow**.
Solveu	-	Similar to simple.

Advices

Here are some general advices regarding **TRUST** solvers:

For constant sparse symmetric matices, such as the one of the pressure linear system of incompressible flow:

- use GCP with Algebric Multigrid (AMG GCP) preconditioning.

For non-constant sparse symmetric matices, such as the one of the pressure linear system of quasi compressible flow or two-phase flow solved by front tracking models:

- use GCP with SSOR preconditioning from PETSc (PETSc GCP).

For non-constant sparse non-symmetric matices, solved during implicit time schemes:

- use GMRES or BICGSTAB with Jacobi preconditioner (diag).

For non-constant sparse potentially ill-conditioned matrices, for instance the one obtained from multiphase flow:

- use direct method, typically PETSc Cholesky, that will either use LU or Cholesky according to the matrix conditioning.

Selected Examples

Pressure solver examples

```
Solveur_pression PETSc GCP { precond SSOR { omega 1.6 }  seuil 1.e-12 impr   }

Solveur_pression AMG GCP {  atol 1.e-12 impr   }

Solveur_pression PETSc Cholesky { }

solveur Gen { seuil 1e-8 solv_elem BiCGStab precond ILU { type 2 filling 20 }  }
```

Diffusion implicit solver example

```
Parametre_equation Parametre_diffusion_implicite
{
        Solveur PETSc GCP { precond block_jacobi_ilu { level 0 } rtol 1.e-13 impr }
}
```

Implicit solver example

```
Solveur Implicite
{
        Solveur PETCs GMRES { precond DIAG { } rtol 1e-3 impr }
}
```

```
Solveur Implicite
{
        solveur gmres { diag seuil 1.e-8 nb_it_max 3 impr }
}
```

```
Solveur Implicite
{
        Solveur PETSc CLI
        {
                -pc_type hypre
                -pc_hypre_type boomeramg
                -pc_hypre_boomeramg_strong_threshold 0.8
                -pc_hypre_boomeramg_agg_nl 4
                -pc_hypre_boomeramg_agg_num_paths 5
                -pc_hypre_boomeramg_max_levels 25
                -pc_hypre_boomeramg_coarsen_type PMIS
                -pc_hypre_boomeramg_interp_type ext+i
                -pc_hypre_boomeramg_P_max 2
                -pc_hypre_boomeramg_truncfactor 0.5
                -ksp_type fgmres
        }
}
```

```
        Solveur Sets
        {
                criteres_convergence { alpha 1e-5 pression 1. vitesse 1e-5 temperature␣
↪1e8 k 1e-5 omega 1e-5 }
```

(continues on next page)

(continued from previous page)

```
                iter_min 2
                Solveur PETSc CLI { }
                seuil_convergence_implicite 1e30
        }
```

```
        Solveur Implicite
        {
                seuil_convergence_implicite 1e+0
                seuil_convergence_solveur  1e-4
                Solveur Gen { seuil 1e-8 solv_elem BiCGStab precond ILU { type 2 filling␣
↪20 }  }
                # Solveur GMRES { diag seuil 1e-8 impr } #
        }
```

1.1.7 Temporal schemes

It is mandatory to define a time integration scheme in order to perform a TRUST calculation. Solving a stationary problem is currently not possible.

TRUST implements a large selection of time schemes where the use selection is kept as a choice for the user. It is possible to perform completely pure explicit or even pure implicit integration. The code also allows to employ a semi-implicit time integration.

The basic point that **should be** taken into account is to pay attention to the time-step and the stability criteria. For example, if one selects an explicit time scheme, both the Courant–Friedrichs–Lewy (CFL) and the Von-Neumann criteria must be respected in order to obtain stable results. This constraint can be relaxed when employing an implicit integrator.

In cases where the diffusion time step is more critcal than the convective one, TRUST offers the possibility to implicitly treat the diffusion operator (the convective remains explicit). In such a case, respecting the CFL criterion is sufficient. This can play an important role in accelerating the computational time in some cases.

The stability of the time scheme can be controled in a TRUST's data file by the parameter `facsec` and `facsec_max` (please have a look to the TRUST documentation for a detailed discription). Keep in mind that the non-linear term of the Navier-Stokes equation is not fully implicited, although if one uses a pure implicit time scheme. For this reason, we recommend you to use the `facsec` parameter with moderation !

Here is the list of the time integration schemes available in the platform.

1.1.8 Explicit schemes

Euler Scheme

Keyword	Description
Scheme_euler_explicit	This is the Euler explicit scheme of first order.

Runge Kutta Series

Keyword	Description
Runge_ku	This is a low-storage Runge-Kutta scheme of second order that uses 2 integration points. The method is presented **here** by Williamson (case 1).
Runge_ku	This is a low-storage Runge-Kutta scheme of third order that uses 3 integration points. The method is presented **here** by Williamson (case 7).
Runge_ku _ordre_4_	This is a low-storage Runge-Kutta scheme of third order that uses 3 integration points. The method is presented **here** by Williamson (case 17).
Runge_ku	This is the Runge-Kutta rational scheme of second order. The method is described in the note: Wambeck - Rational Runge-Kutta methods for solving systems of ordinary differential equations, **here**. Although rational methods require more computational work than linear ones, they can have some other properties, such as a stable behaviour with explicitness, which make them preferable. The CFD application of this RRK2 scheme is described in this **note**.
Runge_ku	This is a classical Runge-Kutta scheme of second order that uses 2 integration points.
Runge_ku	This is a classical Runge-Kutta scheme of third order that uses 3 integration points.
Runge_ku	This is a classical Runge-Kutta scheme of fourth order that uses 4 integration points.
Runge_ku	This is a classical Runge-Kutta scheme of fourth order that uses 4 integration points and the 3/8 rule.
Schema_A	This is the Adams Bashforth second order scheme.
Schema_A	This is the Adams Bashforth third order scheme.

1.1.9 Implicit schemes

Euler Scheme

Keyword	Description
Schema_Euler_Implicite	This is the Euler implicit scheme of first order.

Crank Nicholson

Keyword	Description
Sch_C	This is the Crank-Nicholson method of second order accuracy. A mid-point rule formulation is used (Euler-centered scheme). The time derivative at the mid-level is calculated iteratively with a simple under-relaxations method. Since the method is implicit, neither the cfl nor the fourier stability criteria must be respected. The time step is calculated in a way that the iterative procedure converges with the less iterations as possible.

Adams Moulton Series

Keyword	Description
Schema_Adams_Moulton_order_2	Adams Moulton second order scheme.
Schema_Adams_Moulton_order_3	Adams Moulton third order scheme.

Backward Differentiation Series

Keyword	Description
Schema_Backward_Differentiation_order_2	Backward Differentiation second order scheme.
Schema_Backward_Differentiation_order_3	Backward Differentiation third order scheme.

1.1.10 Semi-Implicit schemes

Predictor Corrector

Keyword	Description
Schema_predic	This is the predictor-corrector scheme (second order). It is more accurate and economic than MacCormack scheme. It gives best results with a second ordre convective scheme like quick, centre (VDF).

Crank Nicholson

Keyword	Description
Sch_	It describes a Crank-Nicholson (CN) method of second order accuracy but here, for scalars, because of instablities encountered when dt > dt_CFL, the Crank Nicholson scheme is not applied to scalar quantities. Scalars are treated according to Euler-Explicite scheme at the end of the CN treatment for velocity flow fields (by doing p Euler explicite under-iterations at dt <= dt_CFL). Parameters are the sames (but default values may change) compare to the Sch_CN_iterative scheme plus a relaxation keyword: niter_min (2 by default), niter_max (6 by default), niter_avg (3 by default), facsec_max (20 by default), seuil (0.05 by default)

Leap Frog

Keyword	Description
leap_frog	This is the leap-frog scheme.

1.1.11 Spatial schemes

The spatial discretization of each term in a given equation is carried out in the **TRUST** platform by what we call the **operators** (operators for convective terms, diffusion, gradient, divergence, ...) It is important to keep in mind that all these operators are **dependent** of the employed discretization; in particular on the variables localisation.

In what follows, the available convective operator schemes are summarized.

Attention: The diffusion term is more or less a Laplacien operator and is thus always discretized by a centered difference scheme.

Attention: TRUST allows the user to neglect the operator contribution. This can be done by using the keyword **negligeable** in the concevtion and/or diffusion block.

Finite Volume Difference (VDF) Schemes

Scheme	Keyword	Description
Upwind scheme	Amont	Corresponds to first order upwind scheme.
Centered scheme	Centre or Centre4	They correspond respectively to a second and fourth order centered schemes.
QUICK scheme	Quick	This is the third order Quadratic Upstream Interpolation for Convective Kinematics (Quick) scheme.

Finite Element Volume (VEF) Schemes

Scheme	Keyword	Description
Upwind scheme	Amont	Corresponds to first order upwind scheme.
Centered scheme	Centre or KCentre	Corresponds to second order centered scheme.
QUICK scheme	KQuick	This is the third order Quadratic Upstream Interpolation for Convective Kinematics (Quick) scheme.
EF-Stab scheme	EF_Stab	This scheme is an upwind/centered mixed schemes. The behavior is controlled by a parameter, alpha, where the scheme behaves as a pure upwind with alpha = 1 and centered with alpha = 0.
MUSCL scheme	Muscl	This is the second order Monotonic Upstream-centered Scheme for Conservation Laws (MUSCL) scheme.

PolyMAC-series Schemes

Scheme	Keyword	Description
Upwind scheme	Amon	Corresponds to first order upwind scheme.
Centered scheme	Centre	Corresponds to second order centered scheme.
EF-Stab scheme	EF_St	This scheme is an upwind/centered mixed schemes. The behavior is controlled by a parameter, alpha, where the scheme behaves as a pure upwind with alpha = 1 and centered with alpha = 0.

1.1.12 Post-Processing

Do you know that CFD refers to Colors For Directors ? It sure is a bad joke but it emphasis the importance of beautiful post-processing when you are running a fluid mecanics simulation. The aim of this section is to give you the key to create beautiful pictures and videos from your amazing **TRUST** simulations.

Before trying to post-processing something, make sure that the information has not already been printed out in one of the files creted by a **TRUST** run.

Automatically outpouted files

After running, you will find different files in your directory. Here is a short explaination of what you will find in each type of file depending on its extension.

Even if you don't post-process anything, you will have output files which are listed here:

File	Contents
*my_data_file***.dt_ev**	Time steps, facsec, equation residuals
*my_data_file***.stop**	Stop file ('0', '1' or 'Finished correctly')
*my_data_file***.log**	Journal logging
*my_data_file***.TU**	CPU/GPU performances
*my_data_file***_csv.TU**	CPU/GPU performance formatted as a CSV file
*my_data_file_problem_name***.sauv** or **.xyz**	Saving 2D/3D results for resume
or *specified_name***.sauv** or **.xyz**	(binary files)

and the listing of boundary fluxes where:

- *my_data_file***_Contrainte_visqueuse.out** correspond to the friction drag exerted by the fluid.
- *my_data_file***_Convection_qdm.out** contains the momentum flow rate.
- *my_data_file***_Debit.out** is the volumetric flow rate.
- *my_data_file***_Force_pression.out** correspond to the pressure drag exerted by the fluid.

If you add post-processings in your data files, you will find:

File	Contents
*my_data_file***.sons**	1D probes list
*my_data_file_probe_name***.son**	1D results with probes
*my_data_file_probe_name***.plan**	3D results with probes
*my_data_file***.lml** *(default format)*	
*my_data_file***.lata** *(with all *.lata.* files)*	
*my_data_file***.med**	2D/3D results
or *specified_name***.lml** or **.lata** or **.med**	

The sceen outputs are automatically redirected in *my_data_file***.out** and *my_data_file***.err** files if you run a parallel calculation or if you use the "**-evol**" option of the "trust" script.

You can also redirect them in two files with the following command:

```
# Source TRUST env if not already done
source $my_path_to_TRUST_installation/env_TRUST.sh

# then
trust my_data_file 1>file_out.out 2>file_err.err
```

In the .out file, you will find the listing of physical infos with mass balance and in the .err file, the listing of warnings, errors and domain infos.

Post-processing block

Several keywords can be used to create a post-processing block, into a problem. First, you can create a single post-processing task (**Post_processing** keyword). Generally, in this block, results will be printed with a specified format at a specified time period.

```
Post_processing
{
   Postraitement_definition
   ...
}
```

But you can also create a list of post-processings with **Post_processings** keyword (named with Post_name1, Post_name2, etc...), in order to print results into several formats or with different time periods, or into different results files:

```
Post_processings
{
   Post_name1 { Postraitement_definition }
   Post_name2 { Postraitement_definition }
   ...
}
```

Probes

Probes refer to sensors that allow a value or several points of the domain to be monitored over time. The probes are a set of points defined:

- one by one: **Points** keyword

  or

- by a set of points evenly
  - distributed over a straight segment: **Segment** keyword

    or

  - arranged according to a layout: **Plan** keyword

    or

  - arranged according to a parallelepiped **Volume** keyword.

Here is an example of 2D **Probes** block:

```
Probes
{
   pressure_probe [loc] pressure Periode 0.5 Points 3 1. 0. 1. 1. 1. 2.
   velocity_probe [loc] velocity Periode 0.5 Segment 10 1. 0. 1. 4.
}
```

where the use of *loc* option allow to specify the wanted location of the probes. The available values are **grav** for gravity center of the element, **nodes** for faces and **som** for vertices. There is not default location. If the point does not coincide with a calculation node, the value is extrapolated linearly according to neighbouring node values.

For complete syntax, see the *Keyword Reference Manual*.

Fields

This keyword allows to post-process fields on the whole domain, specifying the name of the backup file, its format, the post-processing time step and the name (and location) of the post-processed fields.

Here is an example of **Fields** block:

```
Fichier results
Format lata
Fields dt_post 1.
{
   velocity [faces] [som] [elem]
   pressure [elem] [som]
   temperature [elem] [som]
}
```

where **faces**, **elem** and **som** are keywords allowed to specify the location of the field.

> **Note**
>
> When you don't specify the location of the field, the default value is **som** for values at the vertices. So fields are post-processed at the vertices of the mesh.

To visualize your post-processed fields, you can use open source softwares like:

VisIt (included in **TRUST** package) or SALOME.

For complete syntax, see the *Keyword Reference Manual*.

Statistics

Using this keyword, you will compute statistics on your unknows. You must specify the begining and ending time for the statistics, the post-processing time step, the statistic method, the name (and location) of your post-processed field.

Here is an example of **Statistiques** block:

```
Statistiques dt_post 0.1
{
   t_deb 1. t_fin 5.
   moyenne velocity [faces] [elem] [som]
   ecart_type pressure [elem] [som]
   correlation pressure velocity [elem] [som]
}
```

This block will write at every **dt_post** the average of the velocity $\overline{V(t)}$:

$$\overline{V(t)} = \begin{cases} 0 & , \text{ for } t \leq t_{deb} \\ \frac{1}{t-t_{deb}} \int_{t_{deb}}^{t} V(t)dt & , \text{ for } t_{deb} < t \leq t_{fin} \\ \frac{1}{t_{fin}-t_{deb}} \int_{t_{deb}}^{t_{fin}} V(t)dt & , \text{ for } t > t_{fin} \end{cases}$$

the standard deviation of the pressure $\langle P(t) \rangle$:

$$\langle P(t) \rangle = \begin{cases} 0 & , \text{ for } t \leq t_{deb} \\ \frac{1}{t-t_{deb}} \sqrt{\int_{t_{deb}}^{t} \left[P(t) - \overline{P(t)}\right]^2 dt} & , \text{ for } t_{deb} < t \leq t_{fin} \\ \frac{1}{t_{fin}-t_{deb}} \sqrt{\int_{t_{deb}}^{t_{fin}} \left[P(t) - \overline{P(t)}\right]^2 dt} & , \text{ for } t > t_{fin} \end{cases}$$

and correlation between the pressure and the velocity $\langle P(t).V(t) \rangle$ like:

$$\langle P(t).V(t) \rangle = \begin{cases} 0 & , \text{ for } t \leq t_{deb} \\ \frac{1}{t-t_{deb}} \int_{t_{deb}}^{t} \left[P(t) - \overline{P(t)}\right] \cdot \left[V(t) - \overline{V(t)}\right] dt & , \text{ for } t_{deb} < t \leq t_{fin} \\ \frac{1}{t_{fin}-t_{deb}} \int_{t_{deb}}^{t_{fin}} \left[P(t) - \overline{P(t)}\right] \cdot \left[V(t) - \overline{V(t)}\right] dt & , \text{ for } t > t_{fin} \end{cases}$$

Remark: Statistical fields can be plotted with probes with the keyword "operator_field_name" like for example: Moyenne_Vitesse or Ecart_Type_Pression or Correlation_Vitesse_Vitesse. For that, it is mandatory to have the statistical calculation of this fields defined with the keyword **Statistiques**.

For complete syntax, see the *Keyword Reference Manual*.

Field names

Existing & predefined fields

You can post-process predefined fields and already existing fields. Here is a list of post-processable fields, but it is not the only ones.

Physical values	Keyword for field_name	Unit
Velocity	Vitesse or Velocity	$m.s^{1}$
Velocity residual	Vitesse_residu	$m.s^{2}$
Kinetic energy per elements	Energie_cinetique_elem	$kg.m^{1}.s^{2}$
Total kinetic energy	Energie_cinetique_totale	$kg.m^{1}.s^{2}$
Vorticity	Vorticite	s^{1}
Pressure in incompressible flow (P/ + gz)	Pression	$Pa.m^{3}.kg^{1}$
Pressure in incompressible flow (P+gz)	Pression_pa or Pressure	Pa
Pressure in compressible flow	Pression	Pa
Hydrostatic pressure (gz)	Pression_hydrostatique	Pa
Total pressure	Pression_tot	Pa
Pressure gradient	Gradient_pression	$m.s^{2}$
Velocity gradient	gradient_vitesse	s^{1}
Temperature	Temperature	C or K
Temperature residual	Temperature_residu	$C.s^{1}$ or $K.s^{1}$
Temperature variance	Variance_Temperature	K^{2}
Temperature dissipation rate	Taux_Dissipation_Temperature	$K^{2}.s^{1}$
Temperature gradient	Gradient_temperature	$K.m^{1}$
Heat exchange coefficient	H_echange_Tref	$W.m^{2}.K^{1}$
Turbulent viscosity	Viscosite_turbulente	$m^{2}.s^{1}$
Turbulent dynamic viscosity	Viscosite_dynamique_turbulente	$kg.m.s^{1}$
Turbulent kinetic	Energy	$K\ m^{2}.s^{2}$
Turbulent dissipation rate	Eps	$m^{3}.s^{1}$
Constituent concentration	Concentration	-
Constituent concentration residual	Concentration_residu	-
Component velocity along X	VitesseX	$m.s^{1}$
Component velocity along Y	VitesseY	$m.s^{1}$
Component velocity along Z	VitesseZ	$m.s^{1}$
Mass balance on each cell	Divergence_U	$m^{3}.s^{1}$
Irradiancy	Irradiance	$W.m^{2}$
Q-criteria	Critere_Q	s^{1}
Distance to the wall Y +	Y_plus	-
Friction velocity	U_star	$m.s^{1}$
Void fraction	Alpha	-
Cell volumes	Volume_maille	m^{3}
Source term in non Galinean referential	Acceleration_terme_source	$m.s^{2}$
Stability time steps	Pas_de_temps	s
Volumetric porosity	Porosite_volumique	-
Distance to the wall	Distance_Paroi	m
Volumic thermal power	Puissance_volumique	$W.m^{3}$
Local shear strain rate	Taux_cisaillement	s^{1}
Cell Courant number (VDF only)	Courant_maille	-
Cell Reynolds number (VDF only)	Reynolds_maille	-
Viscous force	Viscous_force	$kg.m^{2}.s^{1}$

continues on next page

Table 1.1.1 – continued from previous page

Physical values	Keyword for field_name	Unit
Pressure force	Pressure_force	$kg.m^2.s^1$
Total force	Total_force	$kg.m^2.s^1$
Viscous force along X	Viscous_force_x	$kg.m^2.s^1$
Viscous force along Y	Viscous_force_y	$kg.m^2.s^1$
Viscous force along Z	Viscous_force_z	$kg.m^2.s^1$
Pressure force along X	Pressure_force_x	$kg.m^2.s^1$
Pressure force along Y	Pressure_force_y	$kg.m^2.s^1$
Pressure force along Z	Pressure_force_z	$kg.m^2.s^1$
Total force along X	Total_force_x	$kg.m^2.s^1$
Total force along Y	Total_force_y	$kg.m^2.s^1$
Total force along Z	Total_force_z	$kg.m^2.s^1$

Note

Physical properties (conductivity, diffusivity,...) can also be post-processed.

Note

The name of the fields and components available for post-processing is displayed in the error file after the following message: "Reading of fields to be postprocessed". Of course, this list depends of the problem being solved.

Creating new fields

The **Definition_champs** keyword is used to create new or more complex fields for advanced post-processing.

```
Definition_champs { field_name_post field_type { ... } }
```

field_name_post is the name of the new created field and **field_type** is one of the following possible type:

- **refChamp**
- **Reduction_0D** using for example the **min**, **max** or **somme** methods
- **Transformation**

Refer to the *Keyword Reference Manual* for more information.

Note

You can combine several **field_type** keywords to create your field and then use your new fields to create other ones.

Here is an example of new field named *max_temperature*:

```
Read my_problem
{
   ...
   Postraitement
```

(continues on next page)

(continued from previous page)

```
   {
      Definition_champs
      {
         # Creation of a 0D field: maximal temperature of the domain #
         max_temperature Reduction_0D
         {
            methode max
            source refChamp { Pb_champ my_problem temperature }
         }
      }

      Probes
      {
         # Print max(temperature) into the datafile_TMAX.son file #
         tmax max_temperature periode 0.01 point 1 0. 0.
      }

      Champs dt_post 1.0 { ... }
   }
}
```

Complete Post-Processing Example

Here is a complete post-processing example taken from the TRUST's upwind test case.

```
Post_processing
{
    Probes
    {
        sonde_pression pression periode 0.005 points 2 0.13 0.105 0.13 0.115
        sonde_vitesse vitesse periode 0.005 points 2 0.14 0.105 0.14 0.115
        sonde_vitesse_bis vitesse periode 0.005 position_like sonde_vitesse
        sonde_vitesse_ter vitesse periode 1e-5 position_like sonde_vitesse
    }

    Definition_champs
    {
        # Calcul du produit scalaire grad(Pression).Vitesse #
        pscal_gradP_vit Transformation {
            methode produit_scalaire
            sources {
                Interpolation { localisation elem source refChamp { Pb_champ pb gradient_
↪pression } } ,
                Interpolation { localisation elem source refChamp { Pb_champ pb vitesse }
↪ }
            }
        }
        # Calcul du produit scalaire Vitesse.Vitesse #
        pscal_vit_vit_elem Transformation {
            methode produit_scalaire
            sources {
```

(continues on next page)

(continued from previous page)

```
                refChamp { Pb_champ pb vitesse } ,
                refChamp { Pb_champ pb vitesse }
            }
            localisation elem
        }

        pscal_vit_vit_elem_interp Transformation {
            methode produit_scalaire
            sources {
                Interpolation { localisation elem source refChamp { Pb_champ pb vitesse }
↪ } ,
                Interpolation { localisation elem source refChamp { Pb_champ pb vitesse }
↪ }
            }
        }

        pscal_vit_vit_som Interpolation { localisation som sources_reference { pscal_vit_
↪vit_elem_interp } }

        pscal_vit_vit_faces Interpolation { localisation faces sources_reference { pscal_
↪vit_vit_elem_interp } }

        norme_transfo_vit Transformation {
            methode norme
            source Interpolation {
                localisation elem
                source refChamp { Pb_champ pb vitesse }
            }
        }

        vecteur_unitairex transformation { methode vecteur expression 2 1. 0.␣
↪localisation faces }

        vecteur_transfo transformation { methode vecteur  expression 2 x t localisation␣
↪faces }

        vecteur_source_elem Transformation {
            methode vecteur  expression 2 pression_natif_dom y
            source refChamp { Pb_champ pb pression }
            localisation elem
        }
        vecteur_source_faces Transformation {
            methode vecteur expression  2 pression_natif_dom y
            source refChamp { Pb_champ pb pression }
            localisation Faces
        }

        compo_vitessex Transformation {
            methode composante numero 0
            source refChamp { Pb_champ pb vitesse }
            localisation elem
        }
```

(continues on next page)

(continued from previous page)

```
        compo_vitessex_elem Interpolation { localisation elem sources_reference { compo_
↪vitessex } }

        # Calcul de la composante selon X du gradient de pression #
        compo_graPx Transformation {
            methode composante numero 0
            source refChamp { Pb_champ pb gradient_pression }
            localisation elem
        }
        # Interpolation de cette composante aux elements #
        compo_graPx_elem Interpolation { localisation elem sources_reference { compo_
↪graPx } }
    }
    Format Lata
    fields dt_post 1.3 /* physical time */
    {
        pression elem
        vitesse elem
        masse_volumique elem
        gradient_pression elem
        pscal_gradP_vit elem
        pscal_vit_vit_elem
        pscal_vit_vit_elem_interp
        pscal_vit_vit_som
        norme_transfo_vit elem
        vecteur_unitairex elem
        vecteur_transfo elem
        vecteur_source_elem
        compo_vitessex_elem
        compo_graPx_elem
    }
}

liste_de_postraitements {
    test1 Post_processing {
        Definition_champs {
            test_nom reduction_0D {
                methode weighted_average
                sources { refchamp { pb_champ pb pression } }
            }
        }
        format lata
        fields dt_post 1.3
        {
            pression elem
            test_nom
        }
    }

    test2 Post_processing {
        format lata
```

(continues on next page)

(continued from previous page)

```
        fields dt_post 1.3e9
        {
            pression elem
        }
    }
}
```

Visualisation Tools

To open your 3D results in **lata** format, you can use VisIt which is an open source software included in **TRUST** package. For that you may "source" **TRUST** environment and launch VisIt:

```
# Source TRUST env if not already done
source $my_path_to_TRUST_installation/env_TRUST.sh
# then
visit -o my_data_file.lata &
```

To learn how to use it, you can do the Quick start.

To open your 3D results in **med** format, you can also use VisIt, SALOME or Paraview.

Here are some actions that you can perform when your simulation is finished:

- To visualize the positions of your probes in function of the 2D/3D mesh, you can open your .son files at the same time of the .lata file in VisIt.
- If you need more probes, you can create them with VisIt (if you have post-processed the good fields) or with MEDCoupling.
- You can use the option **-evol** of the trust script, like:

  ```
  trust -evol my_data_file
  ```

  and access to the probes or open VisIt for 2D/3D visualizations via this tool.

1.1.13 Parallel Simulations

TRUST is a plateform which allows to make parallel calculation following some basic rules:

- **S**ingle **P**rogram, **M**ultiple **D**ata model: tasks are split up and run simultaneously on multiple processors with different input in order to obtain results faster.
- messages exchange by **M**essage **P**assing **I**nterface.
- from a PC to a massively parallel computer, with shared or distributed memory.

Basic Notions

To make a parallel calculation, you have to partition your domain. Each sub-domain will be treated by one processor. In order to have good performances, ensure you that:

- sub-domains have almost the same number of cells.
- joint lengths (boundaries between sub-domains) are minimal.

Performances

You have to choose a number of processors which is in agreement with the number of cells. So, you can evaluate your speed-up (sequential time/parallel time which must be as close as possible of the number of processors) or efficiency (=1/SpeedUp) to choose the right number of processors.

From our experience, for good performance with **TRUST**, each processor has to treat between **20000 and 30000 cells**.

Partitioning

To run a parallel calculation, you must:

- make some modifications on your *my_data_file.data* file,
- do two runs:
  - the first one, to partitioning and create your 'n' sub-domains (two methods will by presented).
  - the second one, to read your 'n' sub-domains and run the calculation on 'n' processors.

The different blocks

Different blocks appear in the data file.

- **Modifications on the mesh block**

  First you may add the tags `# BEGIN MESH #` and `# END MESH #` before and after your mesh block, for example:

```
# BEGIN MESH #
Read_file my_mesh.geo ;
[Trianguler_h my_domain ]
# END MESH #
```

  You can refer to section Mesh to have more information.

- **Adding a partitioning block**

  You may now add the partitioning block which contains the cutting instruction, after your mesh block:

```
# BEGIN PARTITION
Partition my_domain
{
   Partition_tool partitioner_name { option1 option2 ... }
   Larg_joint 2
   zones_name DOM
   ...
}
```

(continues on next page)

(continued from previous page)

```
End
END PARTITION #
```

Where *partitioner_name* is the name of the chosen partitioner, one of **METIS**, **Tranche**, **Sous_Zones**, **Partition** or **Fichier_Decoupage** (see section *TRUST available partitioning tools*).

The **Larg_joint** keyword allows to specify the overlapping width value.

Note the **End** before the last line. It will be useful for the cutting step.

This block will make the partitioning of your domain into the specified number of sub-domains during the partitioning run.

- **Adding a block to read the sub-domains**

  At last, you will add a block which will be activated during the parallel calculation and will allow to read the sub-domains:

```
# BEGIN SCATTER
Scatter DOM .Zones my_domain
END SCATTER #
```

Partitionning: "Assisted" method

Here we will use the **trust -partition datafile** command line to make the partitioning step. We consider that you have correctly add the "#" in your *my_data_file.data* file with the partitioning block and cutting block.

Be careful with the hashtags "#", they are interpreted by the script!

To automatically perform the partitioning step and obtain the parallel data file, you have to run:

```
> trust -partition my_data_file [parts_number]
```

Note

Here parts_number is the number of sub-domains created but it is also the number of processors which will be used.

This command creates:

- a *SEQ_my_data_file.data* file which is a backup file of *my_data_file.data* the sequential data file,
- a *DEC_my_data_file.data* file which is the first data file to be run to make the partitioning.

  It is immediately run by the command line **trust -partition datafile** to create a partition, located in the *DOM_000n*.**Zones** files.

  Note

  The code stops reading this file at the **End** keyword just before the **# END PARTITION #** block.

- a *PAR_my_data_file.data* file which is the data file for the parallel calculation. It reads the *DOM_000n*.**Zones** files through the instruction **"Scatter"**.

  Note that the meshing and cut of the mesh are commented here.

To see your sub-domains, you can run:

```
> trust -mesh PAR_my_data_file
```

For more information, you can do the exercise of the TRUST Tutorial.

TRUST available partitioning tools

In **TRUST**, you can make partitioning with:

- the external partitionning library METIS (open source).

  It is a general algorithm that will generate a partition of the domain

```
Partition_tool Metis
{
   nb_parts N
   [use_weights]
   [pmetis | kmetis]
   [nb_essais N]
}
```

- internal **TRUST** partitioning tool **Tranche** which makes parts by cutting the domain following x, y and/or z directions.

```
partition_tool Tranche
{
   tranches nx ny [nz]
}
```

Figure 3 is an example of what you can obtain by cutting a 1m x 1m square, divided in three parts using METIS and the same square divided in Figure 4 into three slices following the x direction with **Tranche**.

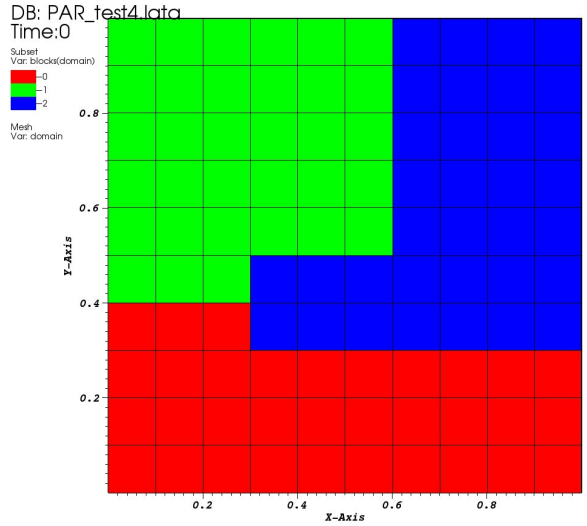


Figure 1.1.3: Figure 3: METIS partition.

For more information, see the TRUST Reference Manual.

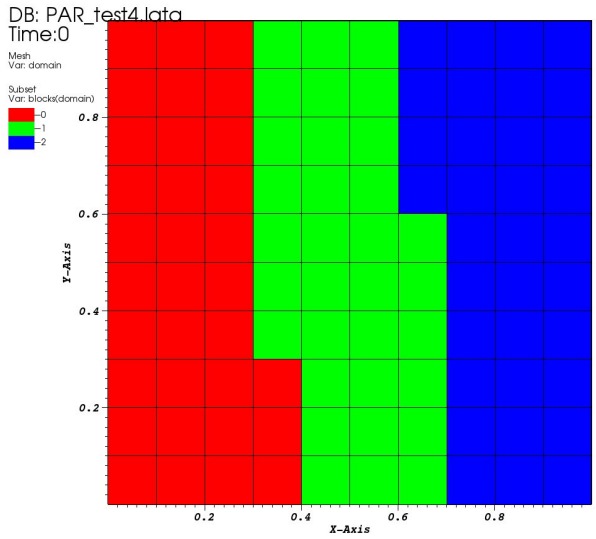


Figure 1.1.4: Figure 4: TRANCH partition.

Overlapping width value

To make the partitioning, you will have to specify the *overlapping width value*. This value corresponds to the thickness of the virtual ghost zone (data known by one processor though not owned by it) i.e. the number of vertices or elements on the remote sub-domain known by the local sub-domain (see Figure 5).

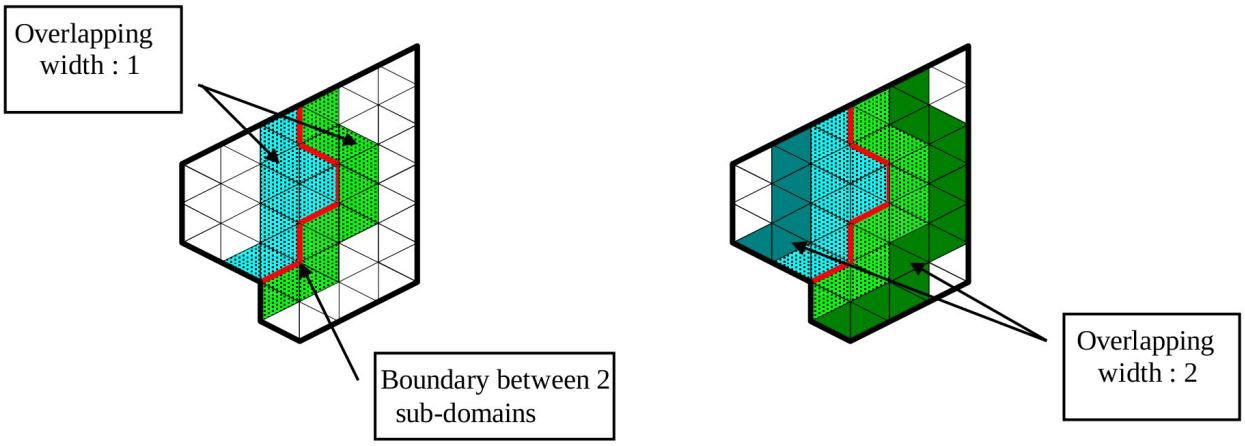


Figure 1.1.5: Figure 5: Overlapping width.

This value depends on the space discretization scheme orders:

- 1 if 1st or 2nd order.
- 2 if 3rd or 4th order.

Note that in general, you will use “2”!

Running a Parallel Calculation

On a PC

To launch the calculation, you have to run the calculation by the usual command completed by the number of processors needed:

```
> trust my_parallel_data_file procs_number
```

and *procs_number* is the number of processors used. In fact it is the same as the number of sub-domains.

You can see the **TRUST**& **TrioCFD** user slides in the "Parallel calculation" section for more information. Also work the exercises in the TRUST Tutorial.

On a cluster

You must submit your job in a queue system. For this, you must have a submission file. **TRUST** can create a submission file for you **on clusters on which the support team has done installations**.

To create this file, run:

```
> trust -create_sub_file my_parallel_data_file
```

You obtain a file named **sub_file**, you can open it and verify/change values(for example the name of the job, the name of the exe, ...).

Then you must submit you calculation with:

```
> sbatch sub_file
```

or

```
> ccc_msub sub_file
```

following the queue system of the cluster.

You can see the **TRUST**& **TrioCFD** user slides in the "Parallel calculation" section for more information.

Visualization

To visualize your probes, you can use the CurvePlot tool, with the command line:

```
> trust -evol my_parallel_data_file
```

or use Gnuplot or any software which reads values in columns in a file.

There are three ways to visualize your parallel results with VisIt:

- HPCDrive or Nice DCV on CCRT/TGCC clusters: opens a deported graphic session on dedicated nodes with more memory (on TGCC cluster: HPCDrive),
- local mode: copy your results from the cluster to your local computer and open it with a local parallel version of VisIt with:

  ```
  > visit -np 4 &
  ```

You can have a look at the **TRUST**& **TrioCFD** user slides in the "Parallel calculation description" section.

Useful Information

Modify the mesh

If you want to modify your mesh, you have two possibilities:

- modify the *my_data_file.data* file and run:

```
> trust -partition my_data_file [parts_number]
```

  Be carefull it will erase the *SEQ_my_data_file.data*, *DEC_my_data_file.data* and *PAR_my_data_file.data* files and creates new ones.

  Then it will run the new *DEC_my_data_file.data* file which gives your new *DOM_000n*.**Zones** files.

- modify the meshing part of file *DEC_my_data_file.data* and run it with:

```
> trust DEC_my_data_file
```

Then run the parallel calculation normally, on the new *DOM_000n*.**Zones** files.

```
> trust PAR_my_data_file procs_number
```

Modify calculation parameters

If you want to modify the calculation parameters, you can modify:

- the file *my_data_file.data* and run:

```
> trust -partition data_file_name [parts_number]
```

  But it will erase the *SEQ_my_data_file.data*, *DEC_my_data_file.data* and *PAR_my_data_file.data* files and create new ones.

  Then it will run the new *DEC_my_data_file.data* file which gives your new *DOM_000n*.**Zones** files.

  Note

  In that case, you don't need to re-create the mesh so you can use the second point below:

- modify the *PAR_my_data_file.data* file *without* running **trust -partition datafile** command line.

Then run the *PAR_my_data_file.data* file with:

```
> trust PAR_my_data_file procs_number
```

Note

If after a certain time, you want to reopen an old case and understand want you did in it without any doubts, you may create two files by hands:

- one "BuildMeshes.data" file only for the mesh and the cut of the mesh.
- one "calculation.data" file for the parallel calculation.

You will run it like:

```
> trust BuildMeshes
> trust calculation processors_number
```

1.2 TRUST Numerical Methods

You will find here the documentation on TRUST numerical methods.

Those pages aim at describing the numerical schemes at hand in the TRUST plaform, giving when necessary the academic references that have driven the implementation, as well as the main places in the code where the methods are implemented.

1.2.1 Spatial discretisations

TRUST provides several spatial discretisations.

Introduction

In the **TRUST** code, different numerical schemes are availabe to the user : VDF, VEF and the PolyMAC family.

- The VDF discretisation is based on the Marker and Cell scheme presented in [Harlow *et al.*, 1965].
- The VEF discretisation is based on the Crouzeix-Raviart element method [Emonot, 1992].

The PolyMAC discretisation family has been developped since 2018. Three PolyMAC are usable in **TRUST**. They have been built using a Finite Volume (FV) framework on a staggered mesh so as to extend the MAC scheme developped in [Harlow *et al.*, 1965] to complex grids:

- PolyMAC : based on a Compact Discrete Operator (CDO) approach, such as the one presented in [Bonelle, 2014] and [Milani, 2020].
- PolyMACP0 : based on MPFA approach, such as the one presented in [Agelas and Masson, 2008], [Droniou, 2014] and [Le Potier, 2017].
- PolyMACP0P1NC : based on a Hybrid Finite Volmue (HFV) approach, such as the one presented in [Eymard *et al.*, 2007] and [Droniou *et al.*, 2010].

Thereafter, for each method the core ideas and the main steps for the discretisation of the incompressible Navier-Stokes equation are presented. For now, the PolyMAC and PolyMAC_P0 parts are completed, the others are a work in progress.

Notations

Let's consider a space Ω and a certain grid $\mathcal{M}$ of non-overlapping polyhedrons that map Ω.

In the following:

- A polyhedron of the grid will be called a cell : e.
- A face f is defined as the intersection of two cells or one face and a boundary. Faces of the grid are supposed to be planar.
- An edge σ is defined as the intersection of faces or faces and boundary. This entity only exists in the 3D framework.
- A vertex v is defined as the intersection of edges or edges and a boundary.

The set of cells will be called E. The set of faces of a peculiar cell e will be denoted F_e. In the same fashion, the set of edges of a peculiar face f will be noted as Σ_f and finally, the two vertices of an edge σ will be denoted V_σ. In the rest of the document, the measure of an unknown x at a control volume cv will be denoted:

$$[x]_{cv} = \frac{1}{|cv|} \int_{cv} x \, \mathrm{d}\,(cv)$$

where $|\cdot|$ will be a global measure operator over the considered control volume. For example, $|e|$ refers to the volume of the cell e, $|f|$ to the surface of the face f and $|\sigma|$ to the length of the edge σ. Unknown u refers to the velocity and p refers to the pressure.

Bibliographie

VDF

The Finite Volume Difference (VDF) discretization (class `VDF_discretisation` and alias `VDF`) is the simplest and most efficient discretization of the TRUST platform. This discretization is compatible with conform mesh with hexahedral type of elements. Attention: do not confuse between a hexahedral mesh and a cartesian mesh. The VDF mesh is not structured and does not follow the IJK indexing!

As stated by its name, the VDF is a conservative finite volume scheme of Marker-and-Cell (MAC) type, [Harlow *et al.*, 1965]. The discretization of each term of the equation is performed by integrating over a control volume. The diffusion gradient terms are approximated by a linear difference equation. All scalars are stored at the center of each control volume except the velocity field which is defined on a staggered mesh.

This discretization **supports** 2D axi-symmetrical configurations and **is compatible** with `Pb_Multiphase`.

VEF

Initially introduced in [Liu and McCormick, 1989], *Volume Element Finis -VEF-* (Finite Volume Element) method is a variant of the standard finite element and finite volume methods. The formalism developed in [Emonot, 1992] was subsequently used for the implementation of this method in the **TRUST** code.

Finite Volume Element method

Core Idea

First, let's consider the following instationary problem, with the velocity $\boldsymbol{u}$ a flux term $\boldsymbol{F}$ and a source term $\boldsymbol{S}$.

$$\partial_t \boldsymbol{u} + \nabla \cdot \boldsymbol{F} = \boldsymbol{S} \tag{1.2.1}$$

We also introduce the control volume ω_f (see Figure Figure 1.2.2) in which we want to evaluate the velocity $\boldsymbol{u}$. We integrate on ω_f between the times t^n and t^{n+1}, regardless the regularity of $\boldsymbol{u}$ and $\boldsymbol{F}$. We also introduce a pressure p.

$$\int_{\omega_f} (\boldsymbol{u}^{n+1} - \boldsymbol{u}^n) \mathrm{d}\boldsymbol{V} + \int_{\partial\omega_f} \int_{t^n}^{t^{n+1}} \boldsymbol{F} \cdot \boldsymbol{n} \mathrm{d}\boldsymbol{s} = \int_{\omega_f} \int_{t^n}^{t^{n+1}} \boldsymbol{S} \mathrm{d}\boldsymbol{V}$$

The expression of the flux term depends on the equation : $\boldsymbol{F} = \mu \nabla \boldsymbol{u} - p\boldsymbol{I}$ for Stokes equation and $\boldsymbol{F} = \mu \nabla \boldsymbol{u} - p\boldsymbol{I} + \rho \boldsymbol{u} \otimes \boldsymbol{u}$ for Navier-Stokes equation.

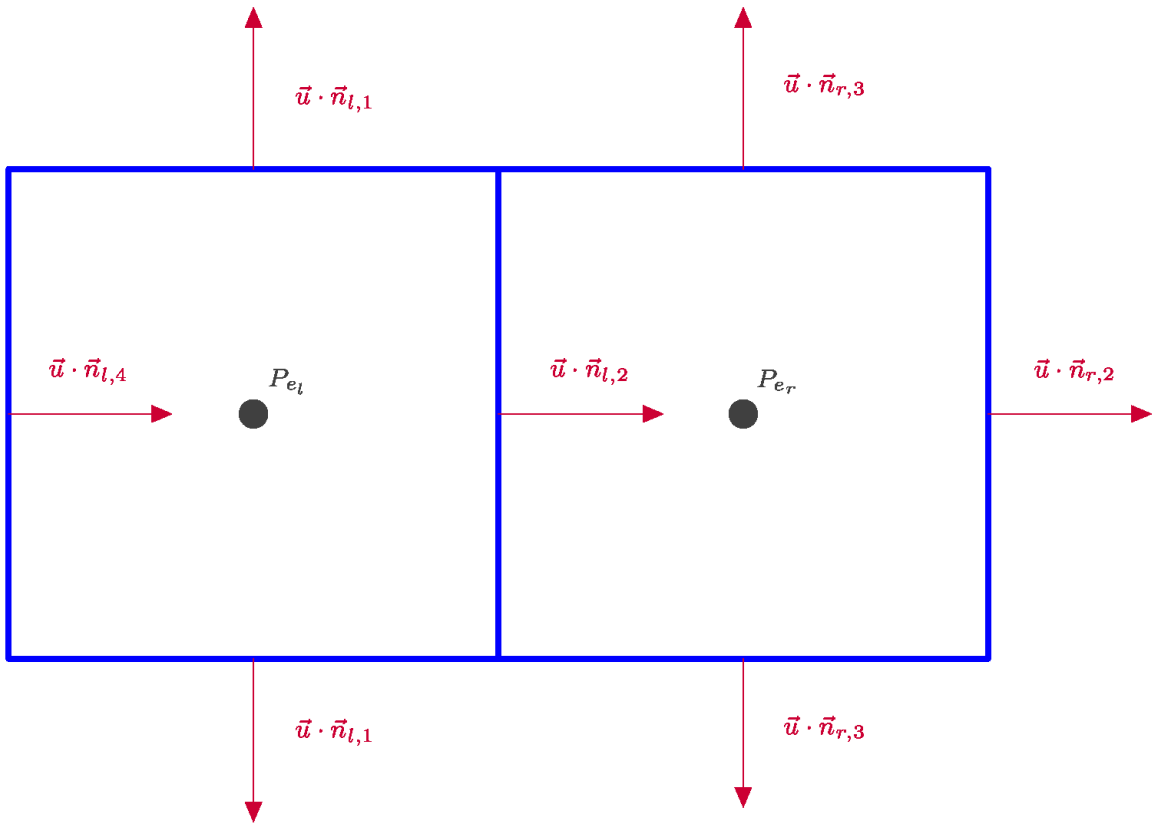


Figure 1.2.1: Scheme of a VDF grid: scalars are stored at the center of the elements and normal component of the vorticities at the faces of the element

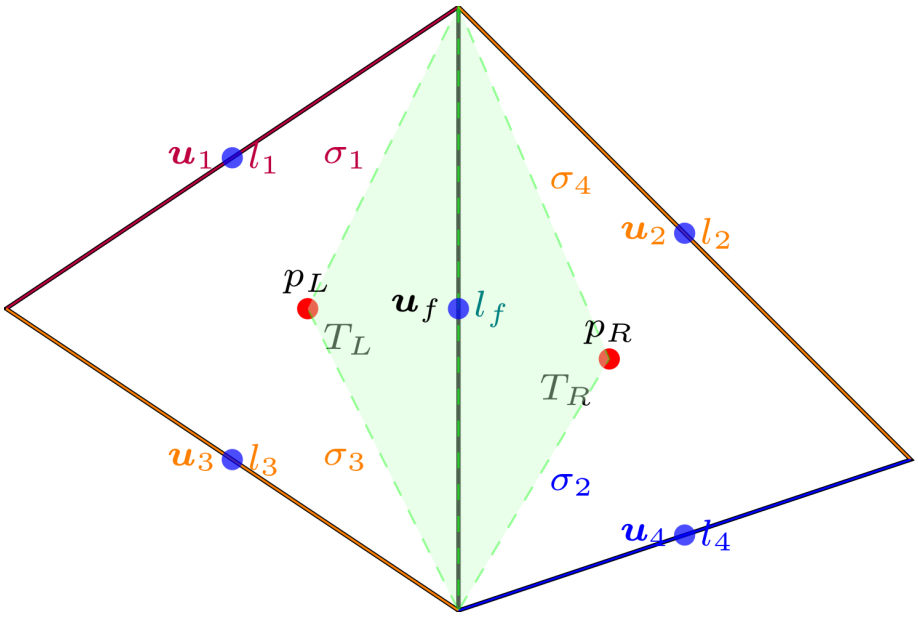


Figure 1.2.2: Control volume for velocity

Finite Volume Approach

Given a tetrahedral mesh $\mathcal{T}_h$, we define the points $\boldsymbol{x}_f$ as the barycentric center of the face f. The control volume ω_f is the polygon which links the vertex of the face $\boldsymbol{f}$ with the barycenters of the two tetrahedron that share the face $\boldsymbol{f}$. Let $\boldsymbol{u}_f^m$ be the approximation of the velocity $\boldsymbol{u}$ at the node $\boldsymbol{x}_f$ and $\Delta t^{n,n+1}\boldsymbol{S}_f^{n,n+1}$ the approximation of the right side hand term. Let's discretize the evolution term such that :

$$\int_{\omega_f} \boldsymbol{u}^m \mathrm{d}\boldsymbol{V} \approx |\omega_f| \boldsymbol{u}_f^m \qquad m \in \{n, n+1\}$$

Let's pose $\boldsymbol{F}^m = \boldsymbol{F}(t^n)$ or $\boldsymbol{F}(t^{n+1})$ or of combination of the two depending on the time scheme choosen. The discretization of the flux term leads to the following equation.

$$\int_{\partial\omega_f}\int_{t^n}^{t^{n+1}} \boldsymbol{F}\cdot\boldsymbol{n}\mathrm{d}\boldsymbol{s}\mathrm{d}t \approx \Delta t^{n,n+1}\int_{\partial\omega_f} \boldsymbol{F}^m\cdot\boldsymbol{n}\mathrm{d}\boldsymbol{s} = \Delta t^{n,n+1}|l_f|(\boldsymbol{F}_{T_R}^m - \boldsymbol{F}_{T_L}^m)\boldsymbol{n}_{T_L,T_R}$$

The discretization of the equation (1.2.1) becomes :

$$|\omega_f|(\boldsymbol{u}_f^{n+1} - \boldsymbol{u}_f) + \Delta t^{n,n+1}|l_f|(\boldsymbol{F}_{T_R}^m - \boldsymbol{F}_{T_L}^m)\boldsymbol{n}_{T_L,T_R} = \Delta t^{n,n+1}\boldsymbol{S}_f^{n,n+1}$$

At this point, the discretization method looks like a Finite Volume scheme. The main difference comes from the way the term $\boldsymbol{F}_T^m$ is discretized with the help of Finite Element basis.

Finite Element Basis

Historically, the VEF method was presented with the Crouzeix-Raviart basis. The full vector of the velocity is evaluated at the center of the faces of each tetrahedron. Within each cell, the pressure is a constant evaluated by its value at the center of the cell. Let's pose $(\phi_f)_{f\in\mathcal{I}_\mathrm{f}}$ the velocity basis (i.e. $\phi_f(\boldsymbol{x_{f'}}) = \delta_{f,f'}$) and $(\mathbb{I}_{K_k})_{k\in\mathcal{I}_K}$ the pressure basis (see Figure 1.2.3). Each discrete velocity vector $\boldsymbol{u}_h$ and pressure p_h can be expressed with the following linear combination.

$$\boldsymbol{u}_h = \sum_{f\in\mathcal{I}_\mathrm{f}} \boldsymbol{u}_f\phi_f$$
$$p_h = \sum_{k\in\mathcal{I}_K} p_k\mathbb{I}_{K_k}$$

Discretization of the flux term in the Stokes equation

For the Stokes equation, the flux term is $\boldsymbol{F} = \mu\nabla\boldsymbol{u} - p\boldsymbol{I}$. Integrating on $\partial\omega_f$, the discretization can be written with the finite element basis :

$$\int_{\partial\omega_f}\boldsymbol{F} = \sum_{f'\in\mathcal{I}_\mathrm{f}}\boldsymbol{u}_{f'}\int_{\partial\omega_f}\boldsymbol{\nabla}\phi_{f'}\cdot\boldsymbol{n}d\boldsymbol{s} + \sum_{k\in\mathcal{I}_K}p_k\int_{\partial\omega_f\cap K_k}\boldsymbol{n}d\boldsymbol{s}$$

Note that the finite element basis $(\phi_f)_{f\in\mathcal{I}_f}$ can be express with the help of barycentric coordinate (see [Crouzeix and Raviart, 1973]) and its gradient is constant per tetrahedron: $(\nabla\phi_f)_T = \frac{1}{|T|}\int_{\partial T}\boldsymbol{n}d\boldsymbol{s}$ (see [Emonot, 1992], p27).

Thus, the discrete gradient of the velocity writes:

$$\begin{aligned}\int_{\partial\omega_f}\boldsymbol{\nabla}\phi_{f'}\cdot\boldsymbol{n}d\boldsymbol{s} &= \sum_{T\in\mathcal{T}_h}(\nabla\phi_{f'})_T\cdot\int_{\omega_f\cap T}\boldsymbol{n}d\boldsymbol{s}\\ &= -\sum_{T\in\mathcal{T}_h}\frac{1}{|T|}S_T^{f'}\cdot S_T^f,\end{aligned}$$

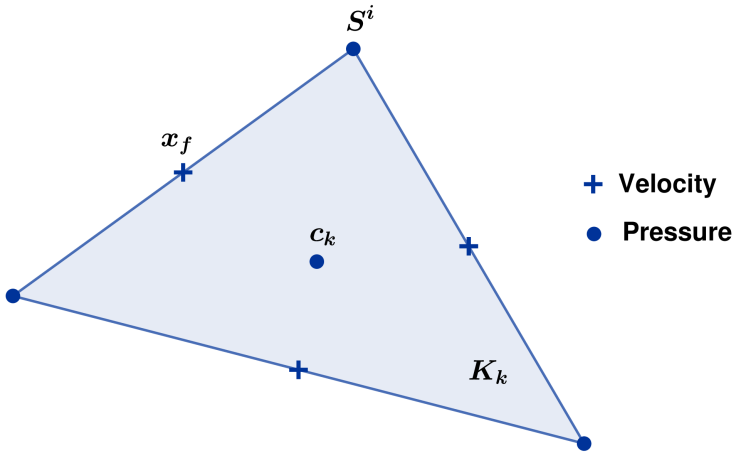


Figure 1.2.3: Control volumes for VEF-P0

with :

$$\int_{\omega_f \cap T} \boldsymbol{n} ds = -\int_{\partial T} \boldsymbol{n} ds = S_T^f$$

and the pressure part :

$$\sum_{k \in \mathcal{I}_K} p_k \int_{\partial \omega_f \cap K_k} \boldsymbol{n} ds = |l_f|(p_{T_R} - p_{T_L}) \boldsymbol{n}_{T_L,T_R}$$

Variational Formulation of the Stokes problem

Let us introduce $\mathbb{X}_h$ the finite element space for discrete velocities $\boldsymbol{u}_f$ and $\mathring{\mathbb{N}}_h$ for the discrete pressure. Then, we obtain the following *VEF* variational formulation by multiplying the mass conservation by a *test* pressure function $q_h = \sum_{k \in \mathcal{I}_K} q_k \mathbb{I}_{K_k}$ and the momentum conservation by a *test* velocity function $\boldsymbol{v}_h = \sum_{f \in \mathcal{I}_f} \boldsymbol{v}_f \phi_f$.

Find $(\boldsymbol{u}_h, p_h) \in \mathbb{X}_h \times \mathring{\mathbb{N}}_h$ such that:

$$\begin{cases} \partial_t m_h^V(\boldsymbol{u}_h, \boldsymbol{v}_h) + a_h^V(\boldsymbol{u}_h, \boldsymbol{v}_h) + b_h^V(\boldsymbol{v}_h, p_h) = L_h^V(\boldsymbol{v}_h) & \forall \boldsymbol{v}_h \in \mathbb{X}_h, \\ c_h^V(\boldsymbol{u}_h, q_h) = 0 & \forall q_h \in \mathring{\mathbb{N}}_h. \end{cases} \tag{1.2.2}$$

with:

$$m_h^V := \begin{cases} \mathbb{X}_h \times \mathbb{X}_h \to \mathbb{R}, \\ (\boldsymbol{u}_h, \boldsymbol{v}_h) \mapsto \sum_{f,f' \in \mathcal{I}_f} \boldsymbol{u}_{f'} \cdot \boldsymbol{v}_f |\omega_f| \delta_f(\boldsymbol{x}_{f'}) \end{cases}$$

$$a_h^V := \begin{cases} \mathbb{X}_h \times \mathbb{X}_h \to \mathbb{R}, \\ (\boldsymbol{u}_h, \boldsymbol{v}_h) \mapsto \sum_{f,f' \in \mathcal{I}_\mathrm{f}} \boldsymbol{u}_{f'} \boldsymbol{v}_f \int_{\partial\omega_f} \boldsymbol{\nabla}\phi_{f'} \cdot \boldsymbol{n} d\boldsymbol{s}. \end{cases}$$

$$b_h^V := \begin{cases} \mathbb{X}_h \times \mathring{\mathbb{N}}_h \to \mathbb{R}, \\ (\boldsymbol{v}_h, p_h) \mapsto \sum_{f \in \mathcal{I}_\mathrm{f}} \sum_{k \in \mathcal{I}_K} \boldsymbol{v}_f p_k \int_{\partial\omega_f \cap K_k} \boldsymbol{n} d\boldsymbol{s}. \end{cases}$$

$$c_h^V := \begin{cases} \mathbb{X}_h \times \mathring{\mathbb{N}}_h \to \mathbb{R}, \\ (\boldsymbol{u}_h, q_h) \mapsto \sum_{k \in \mathcal{I}_K} \sum_{f \in \mathcal{I}_\mathrm{f}} \boldsymbol{u}_f q_k \int_{\partial K_k} \phi_f \cdot \boldsymbol{n} d\boldsymbol{s}. \end{cases}$$

$$L_h^V := \begin{cases} \mathbb{X}_h \to \mathbb{R}, \\ \boldsymbol{v}_h \mapsto \sum_{f \in \mathcal{I}_\mathrm{f}} \boldsymbol{v}_f \int_{\omega_f} \boldsymbol{f} d\boldsymbol{V}. \end{cases}$$

This formulation looks like finite element variational formulation.

Mathematical properties

according to [Heib, 2003], there are two methods for analyzing the scheme based on the formulation (1.2.2):

- The first involves directly analyzing the scheme. It enables to prove the uniform continuity of the bilinear forms, the ellipticity of a_h^V, and establishing the inf-sup conditions.
- The second involves demonstrating the equivalence of assembly matrices derived from FEM and VEF for the same given functional spaces. Thus, numerical scheme can be analyze with the FEM formalism which is well-known for Navier-Stokes equation with Crouzeix-Raviart elements (see [Crouzeix and Raviart, 1973]).

Using these equivalence properties, the Finite Volume Element scheme satisfies the FEM properties:

- **Inf-sup condition**: Ensures the stability of the numerical scheme.
- **Continuity at edge midpoints**: Implies weak continuity of velocity and enforces local mass conservation, leading to a divergence-free condition in each cell.
- **Well-posedness of the discrete problem**: Guarantees the existence and uniqueness of the discrete solution.
- **Convergence rate for pressure**: The pressure approximation converges with order 1 in the L^2 norm.
- **Convergence rate for velocity**: The velocity approximation converges with order 2 in the $\boldsymbol{L^2}$ norm, provided that Ω is convex.

A summary of the Crouzeix-Raviart FEM properties is presented in [Brenner, 2014]. However parasite currents for low velocities can appear when using the VEF approach, see [Fortin, 2006].

New Finite element basis

In order to reduce parasite currents (usefull for low viscosities), a pressure enriched basis was studied in [Heib, 2003] and [Fortin, 2006] and implemented in **TRUST** code under the name VEF - $\mathbb{P}^{nc}/\mathbb{P}^0 + \mathbb{P}^1$. The idea is to add pressure unknows $\mathbb{P}^1$ at the vertices of each cell. This add a new control volume for the mass conservation. Figure 1.2.3 represents the two control volumes for the two pressure unknows:

- K_k for the constant part of the pressure which is $\mathbb{P}^0$
- Π_{S^i} for the $\mathbb{P}^1$ part associated with the unknown located at the center of vertex S^i.

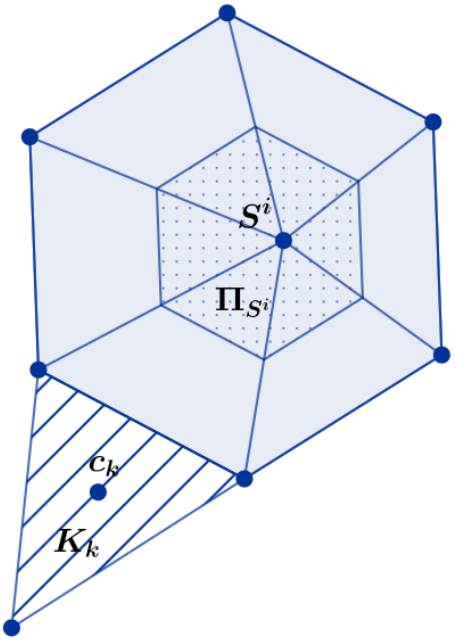


Figure 1.2.4: Control volume for pressure P0 and P1

The stability of this new finite element basis is proved in [Jamelot *et al.*, 2023] and the inf-sup condtion in [Fortin, 2006]. This scheme is the most used VEF discretization in **TRUST**.

PolyMAC_CDO

PolyMAC discretization is based on the development of Compatible Discrete Operators (CDO) schemes. This discretization should make it possible to apply finite volume schemes to non-conforming meshes, with the sole constraint of arranging star-shaped meshes.

Dual Mesh

PolyMAC introduces a rather complex dual mesh. To do so, the gravity center of each control volume $cv \in \{E, F, \Sigma, V\}$ corresponding respectively to the cells, the faces, the egdes and the points localization, called x_{cv} has to be introduce. Then we introduce:

- The dual cell $\widetilde{v}$ is located at the center of gravity of the cell : x_e, see Figure Figure 1.2.5.
- The dual face $\widetilde{\sigma}$ is the line that links the gravity center of the face x_f to the gravity center of the neighbour cells of the face, see Figure Figure 1.2.6.
- The dual edge $\widetilde{f}$ is the surface that links the gravity center of all of the neighbouring cells x_e, the gravity center of all of the neighbouring faces x_f and the gravity center of the edge x_σ, see Figure Figure 1.2.7.

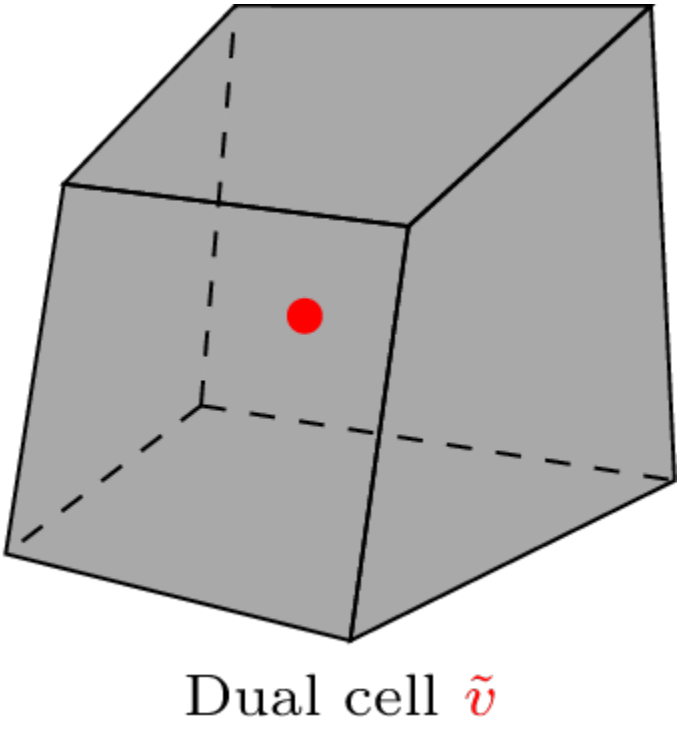


Figure 1.2.5: Dual cell when using PolyMAC_CDO

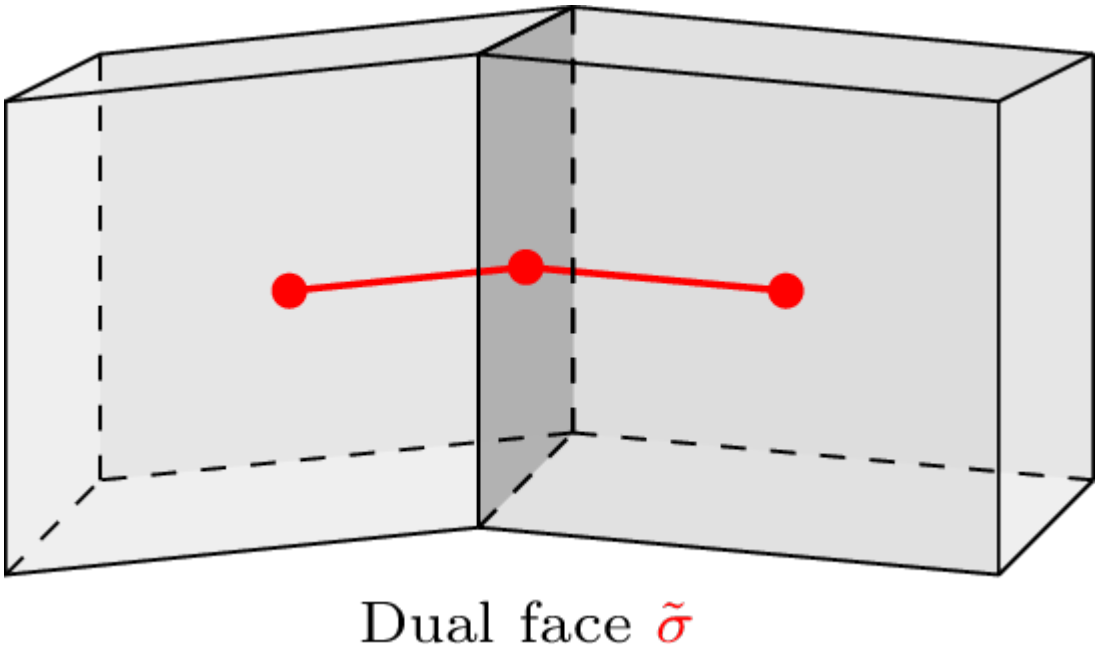


Figure 1.2.6: Dual face when using PolyMAC_CDO

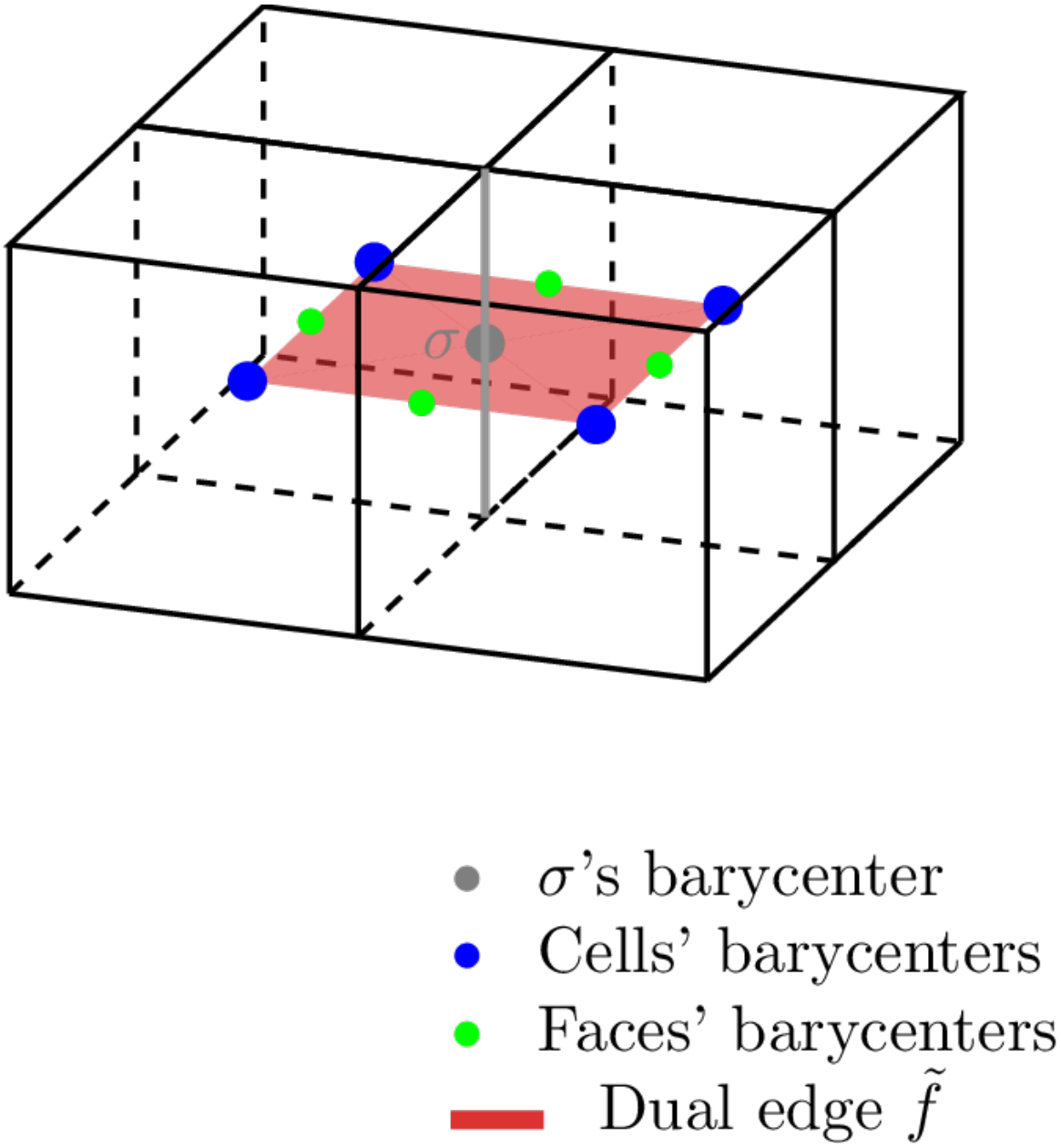


Figure 1.2.7: Dual edge when using PolyMAC_CDO

Location of the unknowns

In PolyMAC, unknowns are discretised according to their "physical" properties. A circulation is discretised over an edge, a flux over a face, a potential over the dual cell. Therefore we have:

- The pressure p is stored at the dual cell: $p_{\tilde{v}} = p(x_{ev}, t)$.
- The normal component of the velocity with respect to a face f is stored as: $v_f = \frac{1}{|f|} \int u \cdot n dS$.
- The tangential vorticity with respect to an edge σ is stored as: $\omega_\sigma = \frac{1}{|\sigma|} \int \omega \cdot \tau dl$.

CDO scheme

The CDO scheme is based on a set of exact operators that allow you to switch from one location to another on the primal and dual mesh.

Figure Figure 1.2.8 summarized the different projection between control volumes in CDO. It is useful to keep it in mind when one want to discretised an equation on a specific control volume.

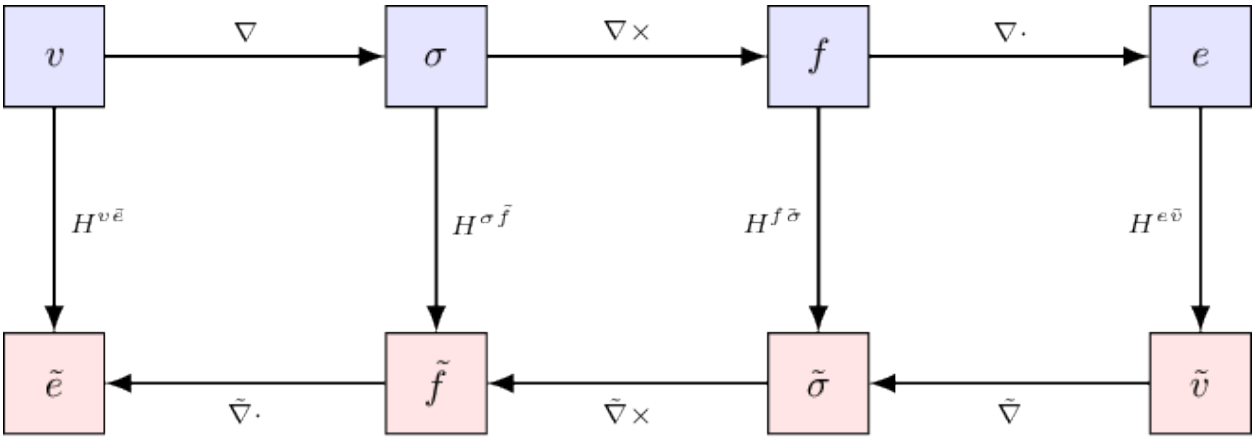


Figure 1.2.8: Paths between primal and dual mesh entities and corresponding operators.

Operators can be classified into 2 types:

- The exact operators, such as the gradient or divergence operator, which correspond to linear operators used to move from one discretization to another.
- The approximated operators that introduce approximation error. In particular, Hodge operators are used to establish discrete closure relation linking primal and dual meshes. Other interpolation operators can also be used especially for the convection operator.

Exact discrete operators

Discrete differential operators relies the entities and allow to switch from one localization to another

Exact discrete operator GRAD, DIV and CURL are defined and have their counterpart $\widetilde{\text{GRAD}}$, $\widetilde{\text{DIV}}$ and $\widetilde{\text{CURL}}$ on the dual mesh

Discrete gradient

$$\text{GRAD} : V \to \pm, \qquad \forall \sigma \in \Sigma, \forall \mathbf{p} \in V, \qquad \text{GRAD}(\mathbf{p})|_{\tilde{\sigma}} = \sum_{\tilde{v} \in V_\sigma} \iota_{v,\sigma} p_v,$$

where $V_\sigma := \{v \in V | v \subset \partial\sigma\}$ and $\iota_{v,\sigma} = +1$ if τ_σ points towards v, $\iota_{v,\sigma} = -1$ otherwise

$$\widetilde{\text{GRAD}} : \widetilde{V} \to \widetilde{\Sigma}, \qquad \forall \tilde{\sigma} \in \widetilde{\pm}, \forall \mathbf{p} \in \widetilde{\mathcal{V}}, \qquad \widetilde{\text{GRAD}}(\mathbf{p})|_{\tilde{\sigma}} = \sum_{\tilde{v} \in \widetilde{V}_{\tilde{\sigma}}} \iota_{\tilde{v},\tilde{\sigma}} p_{\tilde{v}}$$

where $\widetilde{V}_{\tilde{\sigma}} := \{\tilde{v} \in \widetilde{V} | \tilde{v} \subset \partial\tilde{\sigma}\}$ and $\iota_{\tilde{v},\tilde{\sigma}} = +1$ if $\tau_{\tilde{\sigma}}$ points towards $\tilde{v}$, $\iota_{\tilde{v},\tilde{\sigma}} = -1$ otherwise

Discrete divergence

$$\text{DIV} : F \to E \qquad \forall e \in E, \forall \phi \in F, \qquad \text{DIV}(\phi)|_e = \sum_{f \in F_e} \iota_{f,e} \phi_f$$

where $F_e := \{f \in F | f \subset \partial e\}$ and $\iota_{f,e} = +1$ if ν_f points outwards e, $\iota_{f,e} = -1$ otherwise

$$\widetilde{\text{DIV}} : \widetilde{F} \to \widetilde{E} \qquad \forall \tilde{e} \in \widetilde{E}, \forall \phi \in \widetilde{F}, \qquad \widetilde{\text{DIV}}(\phi)|_{\tilde{e}} = \sum_{\tilde{f} \in \widetilde{F}_{\tilde{e}}} \iota_{\tilde{f},\tilde{e}} \phi_{\tilde{f}}$$

where $\widetilde{F}_{\tilde{e}} := \{\tilde{f} \in \widetilde{F} | \tilde{f} \subset \partial\tilde{e}\}$ and $\iota_{\tilde{f},\tilde{e}} = +1$ if $\nu_{\tilde{f}}$ points outwards $\tilde{e}$, $\iota_{\tilde{f},\tilde{e}} = -1$ otherwise

Discrete curl

$$\text{CURL} : \Sigma \to F \qquad \forall f \in F, \forall \mathbf{u} \in \Sigma, \qquad \text{CURL}(\mathbf{u})|_f = \sum_{\sigma \in \Sigma_f} \iota_{\sigma,f} \mathbf{u}_\sigma$$

where $\Sigma_f := \{\sigma \in \Sigma | f \subset \partial f\}$ and $\iota_{\sigma,f} = +1$ if τ_σ shares the same orientation as the one induced by ν_f, $\iota_{\sigma,f} = -1$ otherwise

$$\widetilde{\text{CURL}} : \widetilde{\Sigma} \to \widetilde{F} \qquad \forall \tilde{f} \in \widetilde{F}, \forall \mathbf{u} \in \widetilde{\Sigma}, \qquad \widetilde{\text{CURL}}(\mathbf{u})|_{\tilde{f}} = \sum_{\tilde{\sigma} \in \widetilde{\Sigma}_{\tilde{f}}} \iota_{\tilde{\sigma},\tilde{f}} \mathbf{u}_{\tilde{\sigma}}$$

where $\widetilde{\Sigma}_{\tilde{f}} := \{\tilde{\sigma} \in \widetilde{\Sigma} | \tilde{f} \subset \partial\tilde{f}\}$ and $\iota_{\tilde{\sigma},\tilde{f}} = +1$ if $\tau_{\tilde{\sigma}}$ shares the same orientation as the one induced by $\nu_{\tilde{f}}$, $\iota_{\tilde{\sigma},\tilde{f}} = -1$ otherwise

Hodge operators

Closure relations between the localization in the primal cell and the dual cell are formulated using the construction of non exact global discrete operators called Hodge operators $H_{\alpha^{-1}}^{\widetilde{\mathcal{X}}\mathcal{Y}}$. They are express using geometric quantities related to the primal and dual mesh entities and some material properties such as diffusivity.

Local Hodge operator must be symmetric, locally stable and $\mathbb{P}_0$-consistency

Hodge operators are not unique, the strategy in the CDO scheme consists of defining them using a reconstruction operator $L_{\mathcal{X}_e}$

$$L_{\mathcal{X}_e}(\mathbf{a})(\overline{x}) := \sum_{x \in X_e} \mathbf{a}_X l_{x,e}(\overline{x}) \quad \forall \mathbf{a} \in \mathcal{X}_e, \forall \overline{x} \in e$$

Orthogonal decomposition of the reconstruction operator into consistent and stabilization parts

$$L_{\mathcal{X}_e} := C_{\mathcal{X}_e} + S_{\mathcal{X}_e}$$

$C_{\mathcal{X}_e} R_{\mathcal{X}_e}(K) = K$ and $S_{\mathcal{X}_e} R_{\mathcal{X}_e}(K) = 0$ for constant fields

then the local Hodge operator can be generically defined by:

$$H_\alpha^{\mathcal{X}_e \widetilde{\mathcal{Y}}_e}\Big|_{x', \widetilde{y}(x)} := \int_e l_{x,e}(\overline{x}) \alpha l_{x',e}(\overline{x}) \quad \forall x, x' \in X_e$$

We choose here according to description in Bonelle thesis [Bonelle, 2014] (section 7.3.1) and in Codecasa et al. [Codecasa *et al.*, 2010] the Piecewise constant non-conforming reconstruction.

$$L_{\mathcal{X}_e} := C_{\mathcal{X}_e} + \hat{S}_{\mathcal{X}_e}((\mathbb{I}_{\mathcal{X}_e} - R_{\mathcal{X}_e} C_{\mathcal{X}_e}))$$

with $E_{\mathcal{X}_e} : \mathcal{X}_e \to \mathbb{P}_0(e)$ and $\hat{S}_{\mathcal{X}_e} : \mathcal{X}_e \to \mathbb{P}_0(p_{x,e})$ with $p_{x,e}$ the partition of the cell associated to the mesh entities x

Then the local reconstruction of the circulation $\{\underline{l}_{\sigma,e}\}_{\sigma \in \Sigma_e}$ on the piecewise partition volume $p_{\sigma',e},\ \sigma' \in \Sigma_e$ corresponding to the subvolume of the cell attached to the edge σ', the center of the cell and the center of the adjacent face (Figure Figure 1.2.9). It written:

$$\underline{l}_{\sigma,e}|_{p_{\sigma',e}} = \frac{\beta}{|p_{\sigma,e}|} \underline{\tilde{f}}_e(\sigma) \delta_{\sigma,\sigma'} + \left(\mathbb{I} - \beta \frac{\underline{\tilde{f}}_e(\sigma') \otimes \underline{\sigma'}}{|p_{\sigma',e}|} \right) \frac{\underline{\tilde{f}}(\sigma)}{|e|}$$

Then the local reconstruction of the flux $\{\underline{l}_{f,e}\}_{f \in F_e}$ on the piecewise partition volume $p_{f',e},\ f' \in F_e$ corresponding to the subvolume of the cell attached to the face f', and the center of the cell (Figure Figure 1.2.9). It written:

$$\underline{l}_{f,e}|_{p_{f',e}} = \frac{\beta}{|p_{f,e}|} \underline{\tilde{\sigma}}_e(f) \delta_{f,f'} + \left(\mathbb{I} - \beta \frac{\underline{\tilde{\sigma}}_e(f') \otimes \underline{f'}}{|p_{f',e}|} \right) \frac{\underline{\tilde{\sigma}}(f)}{|e|}$$

The choice for the β parameter must be $\beta = \frac{1}{\text{dim}}$ to yield the DGA reconstruction while the choice $\beta = \frac{1}{\sqrt{dim}}$ corresponds to the choice made in SUSHI schemes

Additional reconstruction operator

A first-order reconstruction mapping operator $\mathbb{R} : F \to V$ will be used in the convection operator for the Navier-Stokes equations (see *Incompressible Navier-Stokes*) to interpolate a vector ϕ expressed along the normal of the faces to the center of the cell using the formula (1) from [Basumatary *et al.*, 2014].

$$\phi_\sigma = \frac{1}{|\sigma|} \sum_{f \in \sigma} |f| \phi_f (\vec{x}_f - \vec{x}_\sigma) \tag{1.2.3}$$

Elliptic equations

We first consider here the resolution of a diffusion equation using the CDO schemes

$$-\underline{\nabla} \cdot (\underline{\underline{\kappa}}\, \underline{\nabla}) = s$$

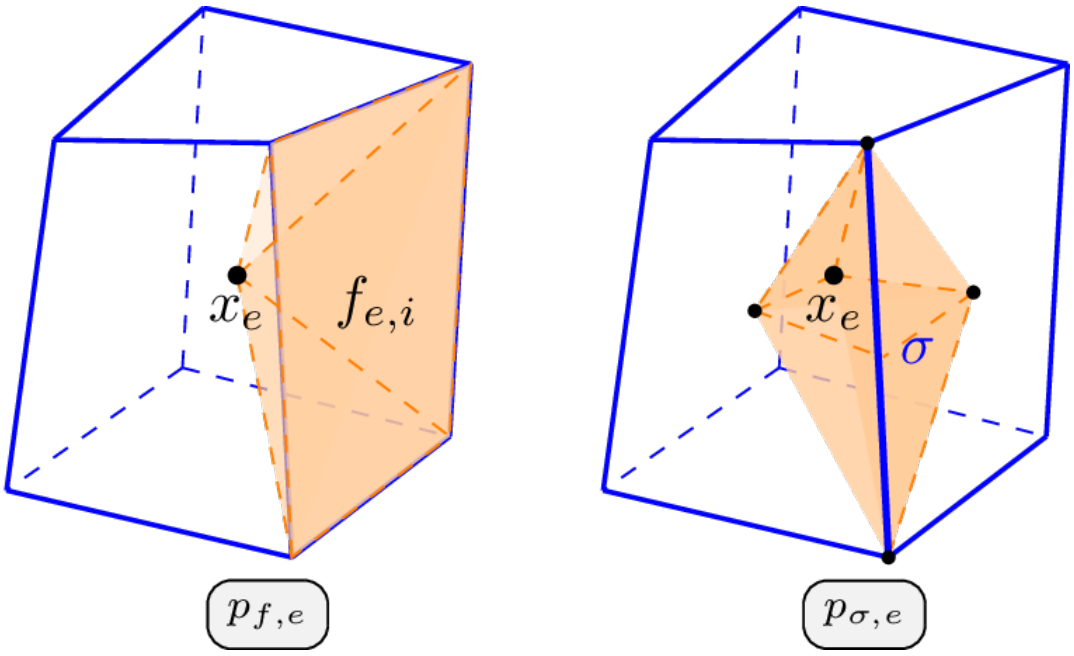


Figure 1.2.9: Partitioning of the cell into elementary sub-volumes attached to face $p_{f,c}$ (left) and to edge $p_{e,c}$ (right)

we introduce the gradient $\overline{g} = \underline{\text{grad}}\ p$ and the flux $\overline{\phi} = -\underline{\underline{\kappa}}\ g$

using cell-based schemes we obtain the discrete system of equations :

$$\begin{cases} \kappa^{-1}\phi + \nabla g = 0 \\ \nabla \cdot \phi = s, \end{cases} \qquad \begin{cases} H_{\kappa^{-1}}^{\widetilde{F\Sigma}}(\phi) + \widetilde{GRAD}(p) = 0 \\ DIV(\phi) = R_E(s) \end{cases}$$

with matricial expression

$$\begin{pmatrix} H_{\kappa^{-1}}^{\widetilde{F\Sigma}} & \widetilde{\mathbb{G}} \\ \mathbb{D} & 0 \end{pmatrix} \begin{bmatrix} \phi \\ p \end{bmatrix} = \begin{bmatrix} 0 \\ R_E(s) \end{bmatrix}$$

with $\widetilde{\mathbb{G}} = -\mathbb{D}^T$

Stokes equations

The Stokes equations model incompressible flows of viscous fluids where the advective inertial forces are negligible with respect to the viscous forces.

$$\begin{cases} -\underline{\Delta u} + \underline{\nabla} p & = \underline{f} \\ \underline{\nabla} \cdot \underline{u} & = 0 \end{cases}$$

where p is the pressure, $\underline{u}$ the velocity and $\underline{f}$ the external load.

Bonelle thesis [Bonelle, 2014] chooses to formulated the Stokes equations with the $\underline{\text{curl}}$ operator using the identity $-\underline{\Delta u} = \underline{\nabla} \times \underline{\nabla} \times \underline{u} - \underline{\nabla}(\nabla \cdot \underline{u})$

$$\begin{cases} \underline{\nabla} \times \underline{\nabla} \times \underline{u} + \underline{\nabla} p & = \underline{f} \\ \underline{\nabla} \cdot \underline{u} & = 0 \end{cases}$$

using the vorticity $\underline{\omega} := \underline{\text{curl}}\ \underline{u}$

$$\begin{cases} -\underline{\omega} + \underline{\nabla} \times \underline{u} & = \underline{0}, \\ \underline{\nabla} \times \underline{\omega} + \underline{\nabla} p & = \underline{f} \\ \underline{\nabla} \cdot \underline{u} & = 0 \end{cases}$$

Introducing the mass density ρ and the viscosity μ leads to:

$$\begin{cases} -\mu^{-1}\underline{\omega}^* + \underline{\nabla} \times (\rho^{-1}\underline{\phi}) & = \underline{0}, \\ \rho^{-1}\underline{\nabla} \times \underline{\omega}^* + \underline{\nabla} p^* & = \underline{f}^* \\ \underline{\nabla} \cdot \underline{\phi} & = 0 \end{cases}$$

with $\underline{\phi} = \rho \underline{u}$, $\underline{\omega}^* = \mu \underline{\omega}$, $p^* = \rho^{-1} p$ and $\underline{f}^* = \rho^{-1} \underline{f}$

We use here two discrete Hodge operators:

$$H^{F\widetilde{\Sigma}}_{\rho^{-1}} : F \to \widetilde{\Sigma} \quad \text{and} \quad H^{\Sigma\widetilde{F}}_{\mu^{-1}} : \Sigma \to \widetilde{F}$$

Then we have the discrete velocity located at dual edges and the discrete vorticity at dual faces.

$$\mathbf{u} = H^{F\widetilde{\Sigma}}_{\rho^{-1}}(\phi) \quad \text{and} \quad \omega := H^{\Sigma\widetilde{F}}_{\mu^{-1}}(\omega^*)$$

we also have the following relation

$$\omega = \widetilde{CURL}(\mathbf{u})$$

The cell-based pressure scheme is:

Find $(\mathbf{p}^*, \phi, \omega^*) \in \widetilde{V} \times F \times \Sigma$

$$\begin{cases} -H^{\Sigma\widetilde{F}}_{\mu^{-1}}(\omega^*) + \widetilde{CURL} \cdot H^{F\widetilde{\Sigma}}_{\rho^{-1}}(\phi) & = 0_{\widetilde{F}}, \\ H^{F\widetilde{\Sigma}}_{\rho^{-1}} \cdot CURL(\omega^*) + \widetilde{GRAD}(\mathbf{p}^*) & = S(\rho, \underline{f}^*), \\ -DIV(\phi) & = 0_E. \end{cases}$$

with matricial expression

$$\begin{pmatrix} -H^{\Sigma\widetilde{F}}_{\mu^{-1}} & \widetilde{\mathbb{C}} \cdot H^{F\widetilde{\Sigma}}_{\rho^{-1}} & 0 \\ H^{F\widetilde{\Sigma}}_{\rho^{-1}} \cdot \mathbb{C} & 0 & \widetilde{\mathbb{G}} \\ 0 & -\mathbb{D} & 0 \end{pmatrix} \begin{bmatrix} \omega^* \\ \phi \\ \mathbf{p}^* \end{bmatrix} = \begin{bmatrix} 0 \\ S(\rho, \underline{f}^*) \\ 0 \end{bmatrix}$$

with $\widetilde{\mathbb{G}} = -\mathbb{D}^T$ and $\widetilde{\mathbb{C}} \cdot H^{F\widetilde{\Sigma}}_{\rho^{-1}} = H^{F\widetilde{\Sigma}}_{\rho^{-1}} \cdot \mathbb{C}$

Incompressible Navier-Stokes

$$\begin{cases} \partial_t \underline{u} + \nabla \cdot (u \otimes u) + \nu \underline{\Delta}(\underline{u}) + \underline{\nabla} p & = \underline{f} \\ \underline{\nabla} \cdot \underline{u} & = 0 \end{cases}$$

where p is the pressure, $\underline{u}$ the velocity and $\underline{f}$ the external load.

Using the identity $-\nu\underline{\Delta} = \nu\underline{\nabla} \times \underline{\nabla} \times \underline{u} - \underline{\nabla}(\nabla \cdot \underline{u})$ we obtain

$$\begin{cases} \partial_t \underline{u} + \underline{\nabla} \times \underline{\nabla} \times \underline{u} + \underline{\nabla} \cdot (\underline{u} \otimes \underline{u}) + \underline{\nabla} p & = \underline{f} \\ \underline{\nabla} \cdot \underline{u} & = 0 \end{cases}$$

using the vorticity $\underline{\omega} := \underline{\text{curl}}\ \underline{u}$

$$\begin{cases} -\underline{\omega} + \underline{\nabla} \times \underline{u} & = \underline{0}, \\ \partial_t \underline{u} + \underline{\nabla} \times \underline{\omega} + \nabla \cdot (u \otimes u) + \underline{\nabla} p & = \underline{f} \\ \underline{\nabla} \cdot \underline{u} & = 0 \end{cases}$$

Introducing the mass density ρ and the viscosity μ leads to:

$$\begin{cases} -\mu^{-1}\underline{\omega}^* + \underline{\nabla} \times (\rho^{-1}\underline{\phi}) & = \underline{0}, \\ \partial_t(\rho^{-1}\underline{\phi}) + \nabla \cdot \left((\rho^{-1}\phi) \otimes (\rho^{-1}\phi)\right) + \rho^{-1}\underline{\nabla} \times \underline{\omega}^* + \underline{\nabla} p^* & = \underline{f}^* \\ \underline{\nabla} \cdot \underline{\phi} & = 0 \end{cases}$$

The cell-based pressure scheme is:

Find $(\mathbf{p}^*, \phi, \omega^*) \in \widetilde{\mathcal{V}} \times F \times \Sigma$

$$\begin{cases} -H_{\mu^{-1}}^{\Sigma\widetilde{F}}(\omega^*) + \widetilde{CURL} \cdot H_{\rho^{-1}}^{F\widetilde{\Sigma}}(\phi) & = 0_{\widetilde{F}}, \\ H_{\rho^{-1}}^{F\widetilde{\Sigma}} \cdot CURL(\omega^*) + \partial_t H_{\rho^{-1}}^{F\widetilde{\Sigma}}(\phi) + CONV(\phi)\phi + \widetilde{GRAD}(\mathbf{p}^*) & = S(\rho, \underline{f}^*), \\ -DIV(\phi) & = 0_E. \end{cases}$$

with matricial expression

$$\begin{pmatrix} -H_{\mu^{-1}}^{\Sigma\widetilde{F}} & \widetilde{\mathbb{C}} \cdot H_{\rho^{-1}}^{F\widetilde{\Sigma}} & 0 \\ H_{\rho^{-1}}^{F\widetilde{\Sigma}} \cdot \mathbb{C} & H_{\rho^{-1}}^{F\widetilde{\Sigma}} \partial_t + \mathbb{T}(\phi) & \widetilde{\mathbb{G}} \\ 0 & -\mathbb{D} & 0 \end{pmatrix} \begin{bmatrix} \omega^* \\ \phi \\ \mathbf{p}^* \end{bmatrix} = \begin{bmatrix} 0 \\ S(\rho, \underline{f}^*) \\ 0 \end{bmatrix}$$

with $\widetilde{\mathbb{G}} = -\mathbb{D}^T$ and $\widetilde{\mathbb{C}} \cdot H_{\rho^{-1}}^{F\widetilde{\Sigma}} = H_{\rho^{-1}}^{F\widetilde{\Sigma}} \cdot \mathbb{C}$

The non linear convection term $CONV$ described in [Beltman *et al.*, 2018] is computing on the using the reconstruction operator (1.2.3)

$$\begin{aligned} [\nabla \cdot (u \otimes u)]_{\widetilde{v}} &= \frac{1}{|e|} \sum_{f \in F_e} |f| [u \otimes u]_f \\ &\simeq \frac{1}{|e|} \sum_{f \in F_e} |f| [u]_f \left(\alpha \left(\gamma [u]_{e_{up}} + (1-\gamma) [u]_{e_{down}} \right) \right. \\ &\quad \left. + (1-\alpha) \left(\frac{[u]_{e_{up}} + [u]_{e_{down}}}{2} \right) \right), \end{aligned}$$

with $\alpha \in [0,1]$ and $\gamma \in \{0,1\}$ such that $\gamma = 1$ if $[u_f] \geq 0$ and 0 otherwise.

This convective terms must be localized on the dual face:

$$[\nabla \cdot (u \otimes u)]_{\widetilde{f}} = \lambda_{e,f} [\nabla \cdot (u \otimes u)]_e + \lambda_{e',f} [\nabla \cdot (u \otimes u)]_{e'}$$

with the penalty coefficient $\lambda_{e,f} = \frac{|\vec{x}_{e' \to f}|}{|\vec{x}_{e' \to f}| + |\vec{x}_{e \to f}|}$ and e' the neighbouring cell of e sharing the face f.

PolyMAC_MPFA

Unlike PolyMAC, PolyMAC_MPFA (previously named PolyMAC_P0) does not introduce the vorticity. Moreover, no complex dual mesh is explicitly needed. The location of the unknowns is described in Figure 1.2.10.

PolyMAC_P0 is based on Multi Point Flux Approximation (MPFA) method.

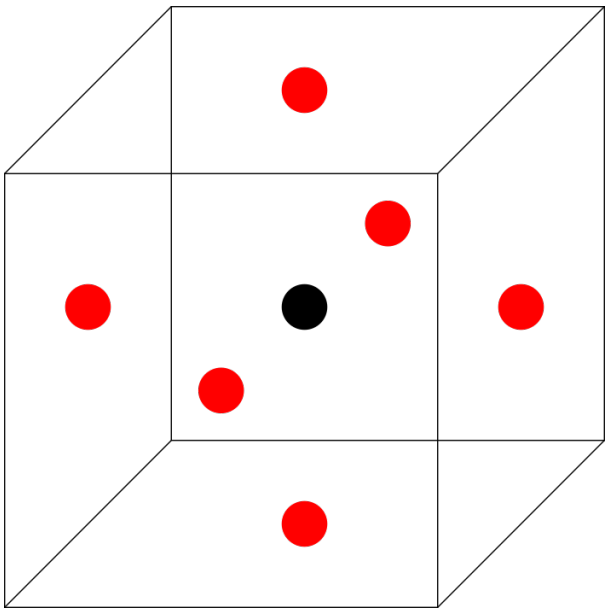


Figure 1.2.10: Location of the unknowns when using PolyMAC_P0

MPFA methods

Three MPFA methods are used in practice in PolyMAC_P0 for computing gradient:

- The MPFA-O method presented in [Aavatsmark, 2002], [Agelas and Masson, 2008], [Droniou, 2014]
- The MPFA-O(η) method presented in [Edwards and Rogers, 1998]
- The MPFA-symm method presented in [Le Potier, 2005], [Le Potier, 2005], [Le Potier, 2017]

The choice of the method is based on a coercivity condition. Let's briefly introduce the core ideas of gradient approximation using MPFA methods. First, a dual mesh is constructed. An exemple of dual mesh for a tringular mesh is presented in Figure 1.2.11, where the red dot are the primal vertices and black lines the primal faces. The procedure to build the dual mesh in Figure 1.2.11 is as follows:

- Link each cell's (e) gravity center (in purple) to the gravity center of each cell's face $f \subset e$ (in blue). Doing so, the face of the mesh are cut into two sub-faces called $\hat{f}_1$ and $\hat{f}_2$. Each cell can then be subdivided into N_i quadrilaterals (in orange), called $(S_{e,i})_{i\in\{1,\dots,N_i\}}$.
- Introduce for each sub-face $\hat{f} \subset f$, an auxiliary quantity (in green). For the MPFA-symmetric method, those auxiliary quantities are set at one third and two third of the face f. For the MPFA-O method, they are put at the center of the face, however, the value of the auxiliary unknowns at the center is not continuous. The MPFA-O(η) method can be seen as an in between, as it try compute the optimum location of the auxiliary unknown.

On S_1 in Figure 1.2.11 for example, the gradient of a potential p, $G_{S_{e,i}}([p]_e)$ is computed as:

$$G_{S_{e,i}}([p]_e) = \frac{1}{|S_{e,i}|}((p_{S_{e,1}} - p_e)\vec{n_1} + (p_{S_{e,1},2} - p_e)\vec{n_2}),$$

where $\vec{n_1}$ and $\vec{n_2}$ are the outward unit normal vectors of the respective sub-faces $\tilde{f} \subset f$ where the auxiliary elements $p_{S_{e,1}}$ and $p_{S_{e,2}}$ are located. Thus, G^{MPFA} writes:

$$G^{\text{MPFA}} : [p]_e \mapsto G^{\text{MPFA}}([p]_e)\ , \quad \forall e \in E\ , \quad i \in S_e\ : \quad G^{\text{MPFA}}_{|S_{e,i}} = G_{S_{e,i}}([p]_e). \tag{1.2.4}$$

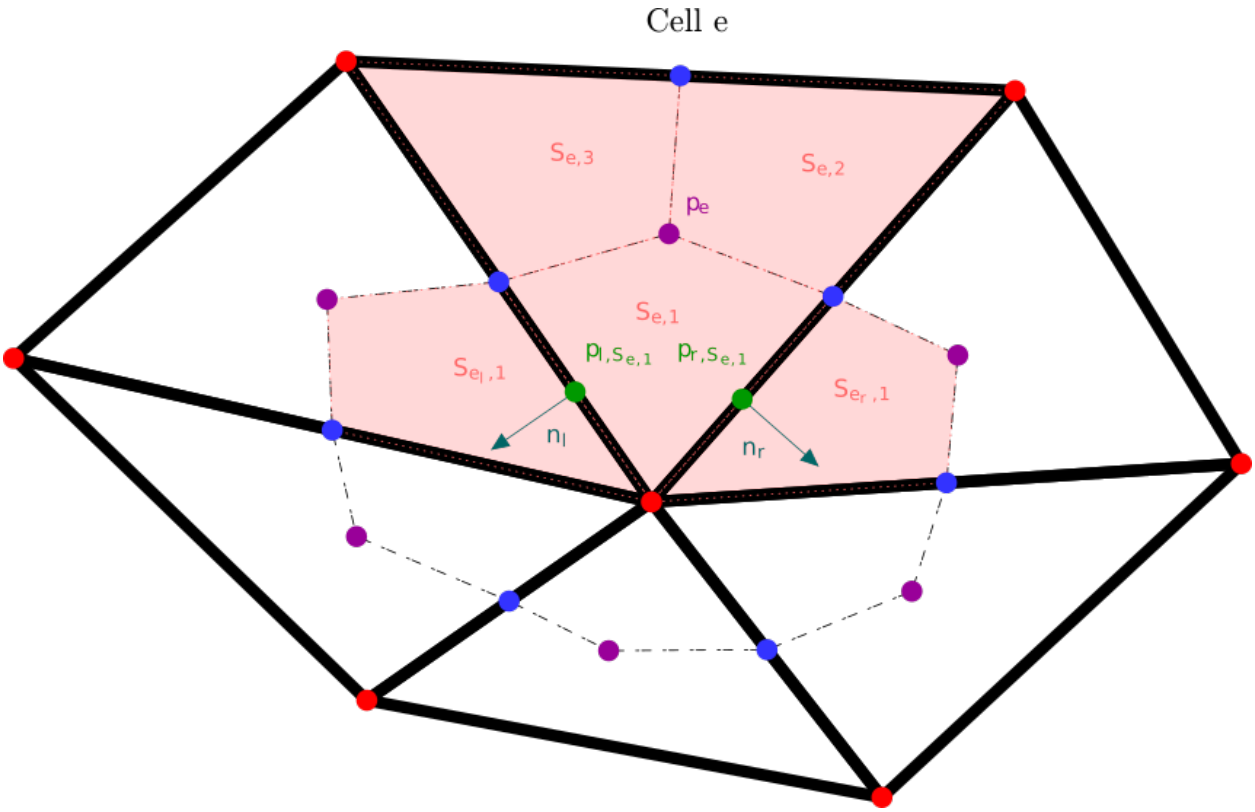


Figure 1.2.11: Construction of a gradient using MPFA method

A core assumption of the MPFA method is to suppose that $G^{\text{MPFA}}([p]_e)$ is constant on each $S_{e,i}$. When enforcing the continuity across the sub-faces that are linked by a vertex of the primal mesh, auxiliary variables can be substitute by cells unknowns.

The MPFA methods are impemented in Domaine_PolyMAC_P0::fgrad.

Incompressible Navier Stokes

The incompressible Navier-Stokes equation reads:

$$\begin{aligned} &\partial_t (u) + \nabla \cdot (u \otimes u) + \nabla p - \mu \Delta u = f \ , \\ &\nabla \cdot u = 0 \ . \end{aligned} \tag{1.2.5}$$

The mass equation is discretised at the cell using the Green-Ostrogradski theorem:

$$|e|[\nabla \cdot u]_e = |f| \sum_{F_e} [u]_f$$

The momentum equation is discretised at the face:

- For the convective term:
  - Approximate the value of the velocity at the cell:

$$[u]_e = \frac{1}{|e|} \sum_{f \in F_e} |f| [u]_f x_{e \to f}.$$

– Discretise the convective terms at the cell centers:

$$\begin{aligned}[\nabla \cdot (u \otimes u)]_e &= \frac{1}{|e|} \sum_{f \in F_e} |f| [u \otimes u]_f \\ &\simeq \frac{1}{|e|} \sum_{f \in F_e} |f| [u]_f \left(\beta \left(\gamma [u]_{e_{up}} + (1-\gamma) [u]_{e_{down}} \right) \right. \\ &\quad \left. + (1-\beta) \left(\frac{[u]_{e_{up}} + [u]_{e_{down}}}{2} \right) \right),\end{aligned}$$

with $\beta \in [0,1]$ and $\gamma \in \{0,1\}$ such that $\gamma = 1$ if $[u_f] \geq 0$ and 0 otherwise.

- The convective terms:
  - Interpolate convective terms to the face:

$$[\nabla \cdot (u \otimes u)]_f = \lambda_{e,f} [\nabla \cdot (u \otimes u)]_e + \lambda_{e',f} [\nabla \cdot (u \otimes u)]_{e'}$$

with the penalty coefficient $\lambda_{e,f} = \frac{|\vec{x}_{e' \to f}|}{|\vec{x}_{e' \to f}| + |\vec{x}_{e \to f}|}$, with e' the neighbouring cell of e sharing the face f.

- The gradient of p is computed using an MPFA scheme (1.2.4).
- The diffusive term is rewritten as :

$$\Delta u = \nabla \cdot (\nabla u + (\nabla u)^\intercal))$$

- Then a second order interpolation is used to compute the velocity at the cell, see:
- First, introducing n_f the outward normal of face f, one can write the series expansion:

$$u_f \cdot n_f \approx (u_e + (\nabla u)_e \cdot (x_f - x_e)) \cdot n_f$$

- Then considering the stencil composed of the faces that share a vertex of e, called $\mathcal{F}_e^v$, one can obtain the following system:

$$A \cdot U_e = U_{\mathcal{F}_e^v}$$

where each line i corresponds to the equation for the i^{th} face of $\mathcal{F}_e^v$. A is a matrix of geometrical quantities, U_e stores the components of u_e and $(\nabla u)_e$, and $U_{\mathcal{F}_e^v}$ the value of u_f at the face.

- However, there is a large number of equations relative to the number of unknowns. To solve this problem, we use a least squares method to find the solution that minimizes:

$$\min_{U_e} \sum_{f \in \mathcal{F}_e^v} \frac{1}{\|x_e - x_f\|} (A(f) U_e - u_f)^2$$

where $A(f)$ corresponds to the line of A associated with face f. This equation is solved using the `dgelsy` method of the **LAPACK** library.

- Afterwards, we compute:

$$[\nabla \cdot (\mu_e ((\nabla u) + (\nabla u)^\intercal))]_e = \sum_f |f| \left(G^{\text{MPFA}}([u]_e) + \left(G^{\text{MPFA}}([u]_e) \right)^\intercal \right) \cdot \vec{n}_f.$$

- Eventually, we interpolate the diffusion term at the face in the same fashion as for the convective term.

Some details regarding the discretisation of a two-phase flow model of the Ishii familly [Ishii, 1975] are given in [Gerschenfeld and Grosse, 2022].

PolyMAC_P0_P1_NC

PolyMACP0P1NC is based on a Hybrid Finite Volmue (HFV) approach, such as the one presented in [Eymard *et al.*, 2007] and [Eymard *et al.*, 2010]. PolyMAC_P0_P1_NC is mathematically close to the first PolyMAC, as HFV and CDO method are equivalent, see [Droniou *et al.*, 2010].

Discontinuous Galerkin Methods

Discontinuous Galerkin (DG) methods form a class of finite element methods particularly suited for solving partial differential equations. Unlike standard continuous Galerkin methods, DG allows for discontinuities between elements, providing greater flexibility and robustness, especially for complex geometries or highly dynamic phenomena.

DG methods have been thouroughly studied in the litterature, see for example [Hesthaven and Warburton, 2007], [Ern and Di Pietro, 2012] or [Cockburn *et al.*, 2012] , yet it remains an important field of study.

Advantages and Challenges of DG Methods

Pros	Cons
High-order accuracy	Large number of unknowns
Handles non-conforming meshes	Numerous parameters to tune
High arithmetic intensity	

Galerkin Methods are new in **TRUST** code. A Symmetric Interior Penalty (SIP) has been implemented for solving Non-Stationary Heat Equation. SIP method have been chosen as it is more performent than mixte DG method for example whan considering method of first and second order.

Before introducing the SIP formulation, the following definitions are needed, see Figure Figure 1.2.12. Considring a face f shared by two cells e_1 and e_2, let us first introduce the interface average of a quantity y

$$\{\{y\}\}_f(x) = \frac{1}{2}\left(y|_{e_1}(x) + y|_{e_2}\right)$$

Then, we introduce the interface jump:

$$[[y]]_f(x) = y|_{e_1}(x) - y|_{e_2}$$

ff the normal of f is defined from e_1 to e_2, and otherwise:

$$[[y]]_f(x) = y|_{e_1}(x) - y|_{e_2}$$

SIP DG Method for the Poisson Problem

First, let us present the SIP DG formulation for the Poisson equation, see [Ern and Di Pietro, 2012] for more details.

Mathematical formulation:

We aim to find $u \in H_0^1(\Omega)$ such that:

$$-\text{div}(k\nabla u) = HS \quad \Rightarrow \quad a_{dg}(u_h, v_h) = \int_\Omega HS v_h, \quad \forall v_h \in X_{DG}$$

Discrete bilinear form:

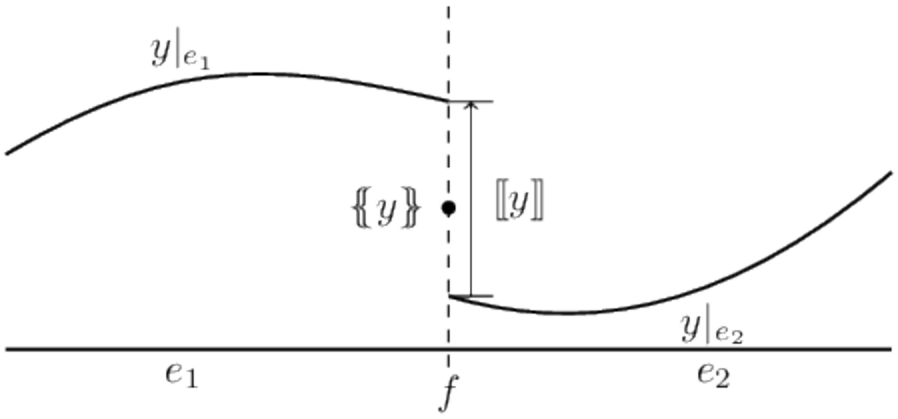


Figure 1.2.12: Definition of the average and jump notations, see [Ern and Di Pietro, 2012]

Defining N_{e_n} the number of neighbour elements of an element e and e_n^i with $i \in [0, N_{e_n}]$ the neighbour elements of e with the convention $e_n^0 = e$, we introduce the discrete bilinear form:

$$a_{dg}(u_h, v_h) = \sum_{\ell=0}^{N_{e_n}} |k|_\ell \int_{e_n^\ell} \nabla u_h \cdot \nabla v_h \tag{1.2.6}$$

$$- \sum_{f \in F_e} \int_f |k|_f \{\{\nabla u_h\}\}_f \cdot \vec{n}_f [[v_h]] \tag{1.2.7}$$

$$- \sum_{f \in F_e} \int_f |k|_f \{\{\nabla v_h\}\}_f \cdot \vec{n}_f [[u_h]] \tag{1.2.8}$$

$$+ \sum_{f \in F_e} \frac{\eta}{h_e} \int_f [[u_h]]_f [[v_h]]_f \tag{1.2.9}$$

where h_e is a geometrical parameter that corresponds to the diameter of the circumscribed circle of e.

- The first term ensures **consistency**.
- The second and third terms impose **symmetry**.
- The last term provides **stability**.

Matrix structure:

The stencil of the SIP DG method is rather small, as only element-wise interactions take place. The global stiffness matrix $\mathbf{K}$ has therefore a block-structured form reflecting this stencil:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{1,1} & \mathbf{K}_{1,2} & 0 & \cdots & \\ \mathbf{K}^e_{1,2} & \mathbf{K}_{2,2} & \mathbf{K}_{2,3} & \cdots & \mathbf{K}_{2,N_E} \\ 0 & \mathbf{K}^e_{2,3} & \mathbf{K}_{3,3} & \cdots & \\ \vdots & & & \ddots & \\ & \mathbf{K}^e_{2,N_E} & & & \mathbf{K}_{N_E,N_E} \end{bmatrix}$$

Example mesh:

The stability parameter η is not closed by default. A method has been added for automatically computing it in order to ensure coercivity.

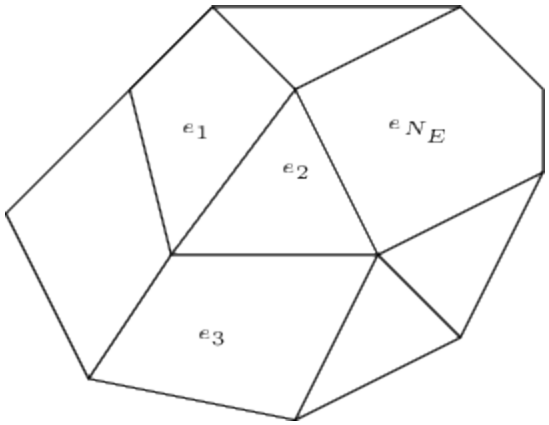


Figure 1.2.13: Possible mesh with the Discontinuous Galerkin discretisation

Non-Stationary Heat Equation

We now consider the time-dependent heat equation, which models heat transfer in a medium over time.

Problem statement:

For all $t \in [0, t_{max}]$, find $T(t) \in H_0^1(\Omega)$ such that:

$$\rho C_p \frac{dT}{dt} - \text{div}(k \nabla T) = HS$$

Physical quantities:

Quantity	Description
k	Thermal conductivity (W·m^1·K^1)
ρ	Density (kg·m^3)
C_p	Heat capacity (J·kg^1·K^1)
T	Temperature (K)
HS	Heat source (W·m^3)

DG Discretization of the Heat Equation

Using a DG formulation, we discretize the heat equation in both space and time.

Weak form:

$$\rho C_p \, m_{dg} \left(\frac{dT_h}{dt}, \theta_h \right) + a_{dg}(T_h, \theta_h) = \int_\Omega f \theta_h, \quad \forall \theta_h \in X_{DG}, \quad \forall t \in [0, t_{max}]$$

The mass bilinear form is defined as:

$$m_{dg}(\tau_h, \theta_h) := \sum_{\ell=1}^{N_{e_n}} \int_{e_n^\ell} \tau_h \theta_h$$

Time integration methods:

Different schemes are available to advance the solution in time:

- **Implicit Euler**

  - Allows for larger time steps
  - Requires solving a linear system at each time step
- **Explicit Euler**
  - Fast, straightforward iterations
  - Requires small time steps due to stability constraints

DG Options

In your data file, you can add an option DG block as follows

{
    order 2 gram_schmidt 1

}

This enables you to secify a custom order of discretisation and the use or not to the Gram-Schmidt orthonormalisation process for your base fucntion. This process is usefull when you work with an explicit scheme as it diagonalise the mass matrix.

For now only **order 1 and 2** are available.

1.2.2 Projection methods

Initially introduced in [Chorin, 1967] - [Témam, 1969], pressure projection methods are a way to separate pressure and velocity unknowns for incompressible flows. These methods are usefull to improve calculation performances, especially when the degrees of freedom are high.

The idea of these methods has two main steps:

1. **a velocity prediction**: we want to try predicting an intermediate velocity without divergence-free constraint and without solving the pressure.
2. **a pressure correction**: find the pressure which corrects the intermediate velocity to be divergence-free again.

In TRUST code, several projection methods have been implemented:

- The Chorin projection method (with or without pressure increment)
- The SIMPLE algorithm
- The SIMPLER algorithm
- The PISO algorithm

In the litterature, projection methods are presented in two opposite ways:

- *the functional formulation*: considering a good regularity for the pressure and the velocity, we take the diffential original systems with the strong formulation and manipulate the differential operators in order to enounce PDEs for the prediction and the correction steps. It often involves homogeneous Neumann conditions on Pressure for the correction step. This approach is popular in the FE community.
- *the algebraic formulation*: once the prediction and the correction steps have been discretized, we do algebraic manipulations on matrix forms to create the linear systems involved. The homogeneous Neumann conditions on pressure is replaced by homogeneous Dirichlet or normal homogeneous Dirichlet for the velocity on correction step in order to prove convergence of projection methods. This approach is often associated with Uzawa algorithm for saddle-point problems.

The convergence proof with the functional and the algebraic formulation has been done in [Guermond, 1996] for the Chorin algorithm and an overview of projection methods for incompressible flows is well presented in [Guermond *et al.*, 2006]. A performance comparison between the differents projections methods is proposed in [Jang *et al.*, 2007].

Because of the low order discretization methods used in TRUST, we present projection methods with the algebraic formulation.

Initial system

These methods do not depend on the spacial discretization (EF, VDF, VEF in TRUST). Considering the matricial problem such that :

$$\mathbb{M}\frac{U^{n+1}-U^n}{\delta t^n}+\mathbb{L}U^m+C(U^m)U^m+\mathbb{B}^TP^{n+1}=F^m$$
$$\mathbb{B}U^{n+1}=0$$

For the Navier-Stokes equation, $\mathbb{L}$ represents the laplacian matrix, $\mathbb{C}(.)$ the convection matrix, $\mathbb{B}^T$ the pressure gradient matrix, and $\mathbb{B}$ the diffusion matrix. The unknowns are U for the velocity vector, P the pressure and the source term is F for the momentum conservation equation.

The laplacian and the convective operators can be etiher explicit or implicit ($m \in \{n, n+1\}$).

For the presentation and we take the source term implicitely, we replace

$$\mathbb{L}U^m+C(U^m)U^m$$

by

$$\mathbb{L}U^{n+1}+C(U^n)U^{n+1} =: \mathbb{A}U^{n+1}.$$

Note that the pressure projection algorithm also exists with the vorticity unknown (see [Bonelle, 2014])

Initial projection

Description of the initial projection

In TRUST, initial pressure is not required. The determination of the initial pressure is done in `Navier_Stokes_std::preparer_calcul`. The idea is similar to the correction step of the Chorin algorithm.

We need to find the inital pressure which corrects the initial velocity to be divergence free. Ideed, even if the initial velocity in globally divergence free, it does not imply that the initial velocity is divergence free at each cells of the mesh. Let's define U^{ini} as the initial velocity given by the user and U^0 the real velocity used to lauch the algorithm in TRUST. The velocity U^0 verifies the following system:

$$\mathbb{M}\frac{U^0-U^{ini}}{\delta t^0}+\mathbb{B}^tP^0=0$$
$$\mathbb{B}U^0=0$$

Multiplying by $\mathbb{B}\mathbb{M}^{-1}$, the initial pressure can be computed solving the following system:

$$\delta t^0\mathbb{B}\mathbb{M}^{-1}\mathbb{B}^tP^0=\mathbb{B}U^{ini}$$

Than, we can compute the initial velocity $U^0 = U^{ini} - \delta t^0\mathbb{B}^tP^0$. Some keywords can be set in order to modify the initial pressure system (we can impose an initial pressure or use the source term to find the right pressure).

Once the initial values of velocity and pressure have been determined, we can compute the projection algorithms.

List of the available projection methods in TRUST

List of the available projection methods

Chorin algorithm

The first projection method was proposed by [Chorin, 1967] and [Témam, 1969]. In this algorithm, the pressure is not taken for the intermediate velocity.

Step 1 : velocity prediction

We call U^* the intermediate velocity vector, which verifies the following equation :

$$\mathbb{M}\frac{U^* - U^n}{\delta t^n} + \mathbb{A}U^* = F^{n+1}$$

The boundary conditions of the unknown U^* is the same as the one imposed for U^{n+1}. Note that U^* has no reason to be divergence-free.

The goal of step 2 is to find the pressure unknown which can correct the intermediate velocity U^* to be divergence-free.

Step 2 : pressure correction

In this step, the convection matrix and the laplacian matrix are volontary avoided (because the mass matrix is easy to inverse). We find the pressure P^{n+1} which verifies:

$$\begin{aligned}\mathbb{M}\frac{U^{n+1} - U^*}{\delta t^n} + \mathbb{B}^T P^{n+1} &= 0,\\ \mathbb{B}U^{n+1} &= 0.\end{aligned}$$

Left multiplying the momentum conservation equation by $\mathbb{B}\mathbb{M}^{-1}$, because of the divergence-free of U^{n+1}, we obtain a linear system on P^{n+1} which is comparable with a Poisson system on pressure :

$$\delta t^n \mathbb{B}\mathbb{M}^{-1}\mathbb{B}^T P^{n+1} = \mathbb{B}U^*$$

Note that this manipulation can not be done with a weak formulation as we usually do in finite elements methods. With more regularity, we can directly consider $\delta t^n \Delta p^{n+1} = \nabla \cdot u^*$ for the analysis.

The boundary condition for the pressure is modified at this step.

Step 3 : update

Once the pressure has been solved, the velocity can be updated with the momentum conservation equation coming from step 2. It comes:

$$U^{n+1} = U^* + \delta t^n \mathbb{M}^{-1}\mathbb{B}^T P^{n+1}$$

Note that if we sum the system from step 1 and 2, we obtain the following reconstructed system:

$$\begin{aligned}\mathbb{M}\frac{U^{n+1} - U^n}{\delta t^n} + \mathbb{A}U^* + \mathbb{B}^T P^{n+1} &= F^{n+1},\\ \mathbb{B}U^{n+1} &= 0.\end{aligned}$$

Remark that the velocity U^* is present in the final system behind the convection and the laplacian (matrix $\mathbb{A}$. It introduces a small error so-called a *splitting error* (see [Guermond, 1996]).

In TRUST, this method is used for explicit scheme.

Chorin algorithm with pressure increment

A small modification of Chorin projection can be done for the pressure.This approach has been proposed in [Goda, 1979]. It consists in taking the pressure at previous time at step 1, and solve an increment of pressure at step 2. This method is used for semi-implicited schemes (explicit schemes with diffusion implicited) or implicit schemes.

Step 1 : velocity prediction

The idea of this step is similar to the Chorin algorithm, excepted the addition of the gradient of pressure solved at time t^n. We write:

$$\mathbb{M}\frac{U^* - U^n}{\delta t^n} + \mathbb{A}U^* + \mathbb{B}^T P^n = F^{n+1}.$$

With this addition, the intermediate velocity U^* solves a semi-implicited Navier-Stokes equation, with no divergence-free constraint.

Step 2 : pressure correction

Adding the pressure at the previous times step for the velocity prediction implies that we need to substract it for the pressure correction system. It comes:

$$\begin{aligned}\mathbb{M}\frac{U^{n+1} - U^*}{\delta t^n} + \mathbb{B}^T \delta P &= 0,\\ \mathbb{B}U^{n+1} &= 0,\end{aligned}$$

with the pressure increment $\delta P := P^{n+1} - P^n$. With the same manipulation presented for the Chorin algorithm, we obtain the following system on pressure increment:

$$\delta t^n \mathbb{B}\mathbb{M}^{-1}\mathbb{B}^T \delta P = \mathbb{B}U^*$$

Step 3 : update

Once the pressure increment solved, the velocity and the pressure at time t^{n+1} are updated:

$$\begin{aligned}U^{n+1} &= U^* + \delta t^n \mathbb{M}^{-1}\mathbb{B}^T \delta P^{n+1}\\ P^{n+1} &= P^n + \delta P.\end{aligned}$$

Note that the reconstructed system is the same as the Chorin algorithm, but the approximation of U^* is closer to U^{n+1} due to the system proposed in step 1.

SIMPLE algorithm

Semi-Implicit Method for Pressure Linked Equations (SIMPLE) is a second projection method proposed by [Patankar and Spalding, 1972] and [Caretto *et al.*, 2007], it is quite similar to Chorin projection method with pressure increment, except that the mass matrix $\mathbb{M}/\delta t^n$ at step 2 and 3 is replaced by the addition of $\mathbb{M}/\delta t^n$ and the diagonal matrix of the convection-diffusion matrix. We note:

$$\mathbb{D} := diag(\mathbb{A} + \frac{\mathbb{M}}{\delta t^n})$$

Thus, the pressure correction becomes:

$$\mathbb{B}\mathbb{D}^{-1}\mathbb{B}^T\delta P = \mathbb{B}U^*$$

and the update:

$$U^{n+1} = U^* + \mathbb{D}^{-1}\mathbb{B}^T\delta P^{n+1}$$
$$P^{n+1} = P^n + \delta P.$$

A relaxation can be done at the update step for the pressure (or the velocity).

The reconstructed system obtain by summing the two steps is quite similar to the Chorin reconstructed system, except that the intermediate velocity U^* has less importance here. If we note $\mathbb{D}_{\mathbb{A}}$ the diagonal part of $\mathbb{A}$ and $\mathbb{E}_{\mathbb{A}}$ its non diagonal, it comes:

$$\mathbb{M}\frac{U^{n+1}-U^n}{\delta t^n} + \mathbb{D}_{\mathbb{A}}U^{n+1} + \mathbb{E}_{\mathbb{A}}U^* + \mathbb{B}^T P^{n+1} = F^{n+1},$$
$$\mathbb{B}U^{n+1} = 0.$$

Implementations details can be found in `Simple.h`.

An improvement of the SIMPLE algorithm has been proposed while adding a pre-computed pressure step before the prediction step: it is the SIMPLER algorithm.

SIMPLER algorithm

SIMPLE Revised algorithm (SIMPLER) consists in applying SIMPLE algorithm with a pre-computed pressure, which consider the non-diagonal term of $\mathbb{A} + \mathbb{M}/\delta t^n$. It was proposed by [Patankar, 1980].

Step 0 : pre-compute the pressure

The goal of this step is to find a pre-computed pressure P^{n+1} in which we apply the SIMPLE algorithm.

Lets define the non diagonal term of $\mathbb{A} + \mathbb{M}/\delta t^n$ such that :

$$\mathbb{E} := \mathbb{A} + \mathbb{M}/\delta t^n - \mathbb{D}$$

To find the pre-computed pressure, an intermediate velocity U^p is find, resolving the following system:

$$\mathbb{D}(U^n)U^p - \mathbb{E}U^n = F^{n+1}$$

Note that this system looks like the velocity prediction step for Chorin projection without pressure increment which would be semi-implicited (the diagonal part is implicited and the non-diagonal is explicited). This system is easy do solve because $\mathbb{D}$ is diagonal. Once U^p is determined, the pre-computed pressure is solved, verifying

$$\mathbb{B}\mathbb{D}^{-1}\mathbb{B}^t P^{n+1} = \mathbb{B}U^p.$$

The system comes from the continuity equation

$$\mathbb{D}(U^n)(U^{n+1} - U^p) + \mathbb{B}^t P^{n+1} = 0,$$
$$\mathbb{B}U^{n+1} = 0.$$

Implementations details can be found in `Simpler.h`.

Step 1 : SIMPLE algorithm on (U^{n+1}, P^{n+1})

The rest of the algorithm is the same that SIMPLE algorithm i.e.:

- velicity prediction : find the intermediate velocity U^*, solution of the following system:

$$\mathbb{M}\frac{U^* - U^n}{\delta t^n} + \mathbb{A}U^* + \mathbb{B}^T P^{n+1} = F^{n+1}.$$

- pressure correction : correct the velocity to respect the divergence-free constraint:

$$\mathbb{B}\mathbb{D}^{-1}\mathbb{B}^T \delta P = \mathbb{B}U^*$$

- update the field with the intermediate velocity:

$$U^{n+1} = U^* + \mathbb{D}^{-1}\mathbb{B}^T \delta P^{n+1}$$

Note that the pressure is not updated between step 0 and step 1, only the velocity is corrected here!

PISO algorithm

The Pressure-Implicit with Splitting of Operators algorithm (PISO) was proposed in [Issa, 1986]. It is a two steps projection method which is a SIMPLE algorithm with a second step which consider the non diagonal part of the convection-diffusion matrix $\mathbb{A}$.

Step 1 : SIMPLE algorithm

As the Chorin projection method with pressure increment, the velicity prediction consists in finding the first intermediate U^* which satisfies the momentum equation

$$\mathbb{M}\frac{U^* - U^n}{\delta t^n} + \mathbb{A}U^* + \mathbb{B}^T P^n = F^{n+1}.$$

Then, find the first pressure increment δP^{p1}, by solving the first Poisson equation:

$$\mathbb{B}\mathbb{D}^{-1}\mathbb{B}^T \delta P^{p1} = \mathbb{B}U^*$$

Then, update the first pressure P^{p1} and velocity fields U^{p1}. :

$$U^{p1} = U^* + \mathbb{D}^{-1}\mathbb{B}^T \delta P^{p1}$$
$$P^{p1} = P^n + \delta P^{p1}.$$

Step 2 : Second pressure correction

The diagonal term of the convection-diffusion matrix has been considered in the system at the SIMPLE step, the second pressure correction considers the non diagonal part.

The poisson system is:

$$\mathbb{B}\mathbb{D}^{-1}\mathbb{B}^t \delta P^{p2} = \mathbb{B}\mathbb{D}^{-1}\mathbb{E}_{\mathbb{A}} U^{p1}$$

Finally, update the velocity and the pressure fields at the next time step.

$$U^{n+1} = \mathbb{E}_{\mathbb{A}} U^{p1} - \mathbb{B}^t \delta P^{p2}$$
$$P^{n+1} = P^{p1} + \delta P^{p2}$$

Algebraic details are presented in [Issa, 1986] or in `Piso.h`

1.2.3 Boundary conditions

This page is currently under construction. Additional boundary conditions will be added in future updates.

Robin boundary conditions

Note that this boundary condition is only available with the VEF Pnc/P0 discretization.

Robin boundary conditions consist in a linear combination between the flux term $\boldsymbol{F}$ and the variables term. This type of boundary conditions can be usefull for Fluid structures interactions or domain decomposition for instance. The one implemented in **TRUST** is decomposed between the normal and the tangential part. Let's define the outward normal vector $\boldsymbol{n} = (n_x, n_y)$.

The flux term becomes:

$$\boldsymbol{F} = F_n \boldsymbol{n} + \boldsymbol{F_t}$$

with F_n the normal part of the flux term and $\boldsymbol{F_t}$ the tangential part.

For the Navier-Stokes equations, the flux term can be written like that:

$$F_n = \nu \nabla_n \boldsymbol{u} \cdot \boldsymbol{n} + \chi(\boldsymbol{u} \cdot \boldsymbol{n})(\boldsymbol{u} \cdot \boldsymbol{n}) - p$$
$$\boldsymbol{F_t} = \nu \nabla_n \boldsymbol{u} \times \boldsymbol{n} + \chi(\boldsymbol{u} \cdot \boldsymbol{n})(\boldsymbol{u} \times \boldsymbol{n})$$

with ν the viscosity, and $\chi \in \{0, 1\}$. In 2D, we project on the tangential vector $\boldsymbol{t} = (-n_y, n_x)$ instead of applying the cross product $\times \boldsymbol{n}$ (the tangential part is only composed by one vector in 2D).

Then, we define two Robin parameters: α for the normal part and β for the tangential part and the Robin data:

- a normal data g_N which is a scalar function,
- a tangential data $\boldsymbol{g_T}$ which is a scalar function in 2D and vectorial function in 3D.

The Robin boundary conditions which have been implemented in **TRUST** are defined such that:

$$\alpha F_n + \boldsymbol{u} \cdot \boldsymbol{n} = g_N$$
$$\beta \boldsymbol{F_t} + \boldsymbol{u} \times \boldsymbol{n} = \boldsymbol{g_T}$$

In **TRUST**, we use the keyword `Robin_VEF` for the Robin boundary conditions, with the following parameters:

- `alpha`, `beta`, defined bellow.
- the keyword `champ_front_normal_et_tangentiel` followed by the field data associated (consider that the followed field as a concatenation with g_N and $\boldsymbol{n} \times \boldsymbol{g_T}$)

For example, considering a 2D Navier-Stokes problem for $\boldsymbol{u} = (y, -x)$, $p = 0.5(x^2 + y^2) - 1/3$, and $\boldsymbol{n} = (1, 0)$, the Robin boundary conditions write in your *.data* file:

```
Robin_VEF {
        alpha 3
        beta 4
        champ_front_normal_et_tangentiel_robin champ_front_fonc_txyz 2 -1.5*x^2-4.5*y^
↪2+y+1.0 4*x*y-x-4
}
```

In this example, the first function in the field corresponds to g_N and the second to g_T.

In 3D, the field `champ_front_normal_et_tangentiel` will have 4 components (one for g_N and three for $\boldsymbol{n} \times \boldsymbol{g_T}$).

Note that we use $\boldsymbol{n} \times \boldsymbol{g_T}$ because we want to write the real tangential component of $\boldsymbol{u}$.

1.3 TRUST Keyword Reference Manual

This is the TRUST keyword reference manual, listing all the available TRUST keywords you can use in a dataset.

Do not forget that you can use the research bar located on the top right of your screen to quickly lookup a precise keyword.

For each keyword:

- its **synonyms** are given;
- its **parent class** is indicated;
- the **list of attributes** is provided. For each attribute in turn: - its name and synonyms are given - its type is given parenthesis - when the attribute is surrounded by square brackets, it is optional.

You will also find here the available syntax for all the mathematical expressions that you can use in TRUST.

Table Of Contents

1.3.1 Syntax to define a mathematical function

In a mathematical function, used for example in field definition, it's possible to use the predifined function (an object parser is used to evaluate the functions) :

ABS	absolute value function
COS	cosine function
SIN	sine function
TAN	tangent function
ATAN	arctangent function
EXP	exponential function
LN	natural logarithm function
SQRT	square root function
INT	integer function
ERF	error function
RND(x)	random function (values between 0 and x)
COSH	hyperbolic cosine function
SINH	hyperbolic sine function
TANH	hyperbolic tangent function
ACOS	inverse cosine function
ASIN	inverse sine function
ATANH	inverse hyperbolic tangent function
NOT(x)	NOT x (returns 1 if x is false, 0 otherwise)
SGN(x)	SGN x (returns 1 if x is positive, -1 if negative, 0 if zero)
x_AND_y	boolean logical operation AND (returns 1 if both x and y are true, else 0)
x_OR_y	boolean logical operation OR (returns 1 if x or y is true, else 0)
x_GT_y	greater than (returns 1 if $x > y$, else 0)
x_GE_y	greater than or equal to
x_LT_y	less than (returns 1 if x<y, else 0)
x_LE_y	less than or equal to
x_MIN_y	returns the smallest of x and y
x_MAX_y	returns the largest of x and y
x_MOD_y	modular division of x per y
x_EQ_y	equal to (returns 1 if x==y, else 0)
x_NEQ_y	not equal to (returns 1 if x!=y, else 0)

You can also use the following operations:

+	addition
-	subtraction
/	division
*	multiplication
%	modulo
$	max
^	power
<	less than
>	greater than
[	less than or equal to
]	greater than or equal to

You can also use the following constants:

Pi	pi value (3,1415…)

The variables which can be used are:

x,y,z	coordinates
t	time

Examples:

Champ_front_fonc_txyz 2 cos(y+x^2) t+ln(y)

Champ_fonc_xyz dom 2 tanh(4*y)*(0.95+0.1*rnd(1)) 0.

Possible errors:

Error 1:

Champ_fonc_txyz 1 cos(10*t)*(1<x<2)*(1<y<2)

Previous line is wrong. It should be written as:

Champ_fonc_txyz 1 cos(10*t)*(1<x)*(x<2)*(1<y)*(y<2)

Error 2:

Champ_front_fonc_xyz 1 20*(x<-2)+10*(y]-5)+3*(z>0)

Previous line is wrong because negative values are not written between parentheses. It should be written as:

Champ_front_fonc_xyz 1 20*(x<(-2))+10*(y](-5))+3*(z>0)

1.3.2 Keywords derived from champ_generique_base

champ_generique_base

not_set

---

champ_post_de_champs_post

not_set

Parameters are:

- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

champ_post_operateur_base

not_set

Parameters are:

- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

champ_post_operateur_eqn

Synonyms: operateur_eqn

Post-process equation operators/sources

Parameters are:

- **[numero_source]** (*type:* int) the source to be post-processed (its number). If you have only one source term, numero_source will correspond to 0 if you want to post-process that unique source
- **[numero_op]** (*type:* int) numero_op will be 0 (diffusive operator) or 1 (convective operator) or 2 (gradient operator) or 3 (divergence operator).
- **[numero_masse]** (*type:* int) numero_masse will be 0 for the mass equation operator in Pb_multiphase.
- **[sans_solveur_masse]** (*type:* flag) not_set

- **[compo]** (*type:* int) If you want to post-process only one component of a vector field, you can specify the number of the component after compo keyword. By default, it is set to -1 which means that all the components will be post-processed. This feature is not available in VDF disretization.
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

champ_post_statistiques_base

not_set

Parameters are:

- **t_deb** (*type:* float) Start of integration time
- **t_fin** (*type:* float) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

correlation

Synonyms: champ_post_statistiques_correlation

to calculate the correlation between the two fields.

Parameters are:

- **t_deb** (*type:* float) Start of integration time
- **t_fin** (*type:* float) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

divergence

Synonyms: champ_post_operateur_divergence

To calculate divergency of a given field.

Parameters are:

- **[source]** (*type:* *champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

ecart_type

Synonyms: champ_post_statistiques_ecart_type

to calculate the standard deviation (statistic rms) of the field nom_champ.

Parameters are:

- **t_deb** (*type:* float) Start of integration time
- **t_fin** (*type:* float) End of integration time
- **[source]** (*type:* *champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

extraction

Synonyms: champ_post_extraction

To create a surface field (values at the boundary) of a volume field

Parameters are:

- **domaine** (*type:* string) name of the volume field
- **nom_frontiere** (*type:* string) boundary name where the values of the volume field will be picked
- **[methode]** (*type:* string into ['trace', 'champ_frontiere']) name of the extraction method (trace by_default or champ_frontiere)
- **[source]** (*type:* *champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword

- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

gradient

Synonyms: champ_post_operateur_gradient

To calculate gradient of a given field.

Parameters are:

- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

interpolation

Synonyms: champ_post_interpolation

To create a field which is an interpolation of the field given by the keyword source.

Parameters are:

- **localisation** (*type:* string) type_loc indicate where is done the interpolation (elem for element or som for node).
- **[methode]** (*type:* string) The optional keyword methode is limited to calculer_champ_post for the moment.
- **[domaine]** (*type:* string) the domain name where the interpolation is done (by default, the calculation domain)
- **[optimisation_sous_maillage]** (*type:* string into ['default', 'yes', 'no']) not_set
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

morceau_equation

Synonyms: champ_post_morceau_equation

To calculate a field related to a piece of equation. For the moment, the field which can be calculated is the stability time step of an operator equation. The problem name and the unknown of the equation should be given by Source refChamp { Pb_Champ problem_name unknown_field_of_equation }

Parameters are:

- **type** (*type:* string) can only be operateur for equation operators.
- **[numero]** (*type:* int) numero will be 0 (diffusive operator) or 1 (convective operator) or 2 (gradient operator) or 3 (divergence operator).
- **[unite]** (*type:* string) will specify the field unit
- **option** (*type:* string into ['stabilite', 'flux_bords', 'flux_surfacique_bords']) option is stability for time steps or flux_bords for boundary fluxes or flux_surfacique_bords for boundary surfacic fluxes
- **[compo]** (*type:* int) compo will specify the number component of the boundary flux (for boundary fluxes, in this case compo permits to specify the number component of the boundary flux choosen).
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

moyenne

Synonyms: champ_post_statistiques_moyenne

to calculate the average of the field over time

Parameters are:

- **[moyenne_convergee]** (*type: field_base*) This option allows to read a converged time averaged field in a .xyz file in order to calculate, when resuming the calculation, the statistics fields (rms, correlation) which depend on this average. In that case, the time averaged field is not updated during the resume of calculation. In this case, the time averaged field must be fully converged to avoid errors when calculating high order statistics.
- **t_deb** (*type:* float) Start of integration time
- **t_fin** (*type:* float) End of integration time
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

predefini

This keyword is used to post process predefined postprocessing fields.

Parameters are:

- **pb_champ** (*type: deuxmots*) { Pb_champ nom_pb nom_champ } : nom_pb is the problem name and nom_champ is the selected field name. The available keywords for the field name are: energie_cinetique_totale, energie_cinetique_elem, viscosite_turbulente, viscous_force_x, viscous_force_y, viscous_force_z, pressure_force_x, pressure_force_y, pressure_force_z, total_force_x, total_force_y, total_force_z, viscous_force, pressure_force, total_force

---

reduction_0d

Synonyms: champ_post_reduction_0d

To calculate the min, max, sum, average, weighted sum, weighted average, weighted sum by porosity, weighted average by porosity, euclidian norm, normalized euclidian norm, L1 norm, L2 norm of a field.

Parameters are:

- **methode** (*type:* string into ['min', 'max', 'moyenne', 'average', 'moyenne_ponderee', 'weighted_average', 'somme', 'sum', 'somme_ponderee', 'weighted_sum', 'somme_ponderee_porosite', 'weighted_sum_porosity', 'euclidian_norm', 'normalized_euclidian_norm', 'l1_norm', 'l2_norm', 'valeur_a_gauche', 'left_value']) name of the reduction method: - min for the minimum value, - max for the maximum value, - average (or moyenne) for a mean, - weighted_average (or moyenne_ponderee) for a mean ponderated by integration volumes, e.g: cell volumes for temperature and pressure in VDF, volumes around faces for velocity and temperature in VEF, - sum (or somme) for the sum of all the values of the field, - weighted_sum (or somme_ponderee) for a weighted sum (integral), - weighted_average_porosity (or moyenne_ponderee_porosite) and weighted_sum_porosity (or somme_ponderee_porosite) for the mean and sum weighted by the volumes of the elements, only for ELEM localisation, - euclidian_norm for the euclidian norm, - normalized_euclidian_norm for the euclidian norm normalized, - L1_norm for norm L1, - L2_norm for norm L2
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

refchamp

Synonyms: champ_post_refchamp

Field of prolem

Parameters are:

- **[nom_source]** (*type:* string) The alias name for the field
- **pb_champ** (*type: deuxmots*) { Pb_champ nom_pb nom_champ } : nom_pb is the problem name and nom_champ is the selected field name.

---

tparoi_vef

Synonyms: champ_post_tparoi_vef

This keyword is used to post process (only for VEF discretization) the temperature field with a slight difference on boundaries with Neumann condition where law of the wall is applied on the temperature field. nom_pb is the problem name and field_name is the selected field name. A keyword (temperature_physique) is available to post process this field without using Definition_champs.

Parameters are:

- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

---

transformation

Synonyms: champ_post_transformation

To create a field with a transformation using source fields and x, y, z, t. If you use in your datafile source refChamp { Pb_champ pb pression }, the field pression may be used in the expression with the name pression_natif_dom; this latter is the same as pression. If you specify nom_source in refChamp bloc, you should use the alias given to pressure field. This is avail for all equations unknowns in transformation.

Parameters are:

- **methode** (*type:* string into ['produit_scalaire', 'norme', 'vecteur', 'formule', 'composante']) methode 0 methode norme : will calculate the norm of a vector given by a source field methode produit_scalaire : will calculate the dot product of two vectors given by two sources fields methode composante numero integer : will create a field by extracting the integer component of a field given by a source field methode formule expression 1 : will create a scalar field located to elements using expressions with x,y,z,t parameters and field names given by a source field or several sources fields. methode vecteur expression N f1(x,y,z,t) fN(x,y,z,t) : will create a vector field located to elements by defining its N components with N expressions with x,y,z,t parameters and field names given by a source field or several sources fields.
- **[unite]** (*type:* string) will specify the field unit
- **[expression]** (*type:* list of str) expression 1 see methodes formule and vecteur
- **[numero]** (*type:* int) numero 1 see methode composante
- **[localisation]** (*type:* string) localisation 1 type_loc indicate where is done the interpolation (elem for element or som for node). The optional keyword methode is limited to calculer_champ_post for the moment
- **[source]** (*type: champ_generique_base*) the source field.
- **[sources]** (*type:* list of Champ_generique_base) XXX
- **[nom_source]** (*type:* string) To name a source field with the nom_source keyword
- **[source_reference]** (*type:* string) not_set
- **[sources_reference]** (*type:* list of Nom_anonyme) List of name.

1.3.3 Keywords derived from chimie

chimie

Keyword to describe the chmical reactions

Parameters are:

- **reactions** (*type:* list of Reaction) list of reactions
- **[modele_micro_melange]** (*type:* int) modele_micro_melange (0 by default)
- **[constante_modele_micro_melange]** (*type:* float) constante of modele (1 by default)
- **[espece_en_competition_micro_melange]** (*type:* string) espece in competition in reactions

---

1.3.4 Keywords derived from class_generic

amg

Wrapper for AMG preconditioner-based solver which switch for the best one on CPU/GPU Nvidia/GPU AMD

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type: bloc_lecture*) not_set

---

amgx

Solver via AmgX API

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type: bloc_lecture*) not_set

---

cholesky

Cholesky direct method.

Parameters are:

- **[impr]** (*type:* flag) Keyword which may be used to print the resolution time.
- **[quiet]** (*type:* flag) To disable printing of information

---

class_generic

not_set

---

cudss

Solver via cuDSS API

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type:* *bloc_lecture*) not_set

---

dt_calc_dt_calc

Synonyms: dt_calc

The time step at first iteration is calculated in agreement with CFL condition.

---

dt_calc_dt_fixe

Synonyms: dt_fixe

The first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity).

Parameters are:

- **value** (*type:* float) first time step.

---

dt_calc_dt_min

Synonyms: dt_min

The first iteration is based on dt_min.

---

dt_start

not_set

---

gcp_ns

not_set

Parameters are:

- **solveur0** (*type:* *solveur_sys_base*) Solver type.
- **solveur1** (*type:* *solveur_sys_base*) Solver type.
- **seuil** (*type:* float) Value of the final residue. The gradient ceases iteration when the Euclidean residue standard ||Ax-B|| is less than this value.
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gcp.
- **[impr]** (*type:* flag) Keyword which is used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[quiet]** (*type:* flag) To not displaying any outputs of the solver.
- **[save_matrice | save_matrix]** (*type:* int) to save the matrix in a file.
- **[precond]** (*type:* *precond_base*) Keyword to define system preconditioning in order to accelerate resolution by the conjugated gradient. Many parallel preconditioning methods are not equivalent to their sequential counterpart, and you should therefore expect differences, especially when you select a high value of the final residue (seuil). The result depends on the number of processors and on the mesh splitting. It is sometimes useful to run the solver with no preconditioning at all. In particular: - when the solver does not converge during initial projection, - when comparing sequential and parallel computations. With no preconditioning, except in some particular cases (no open boundary), the sequential and the parallel computations should provide exactly the same results within fpu accuracy. If not, there might be a coding error or the system of equations is singular.
- **[precond_nul]** (*type:* flag) Keyword to not use a preconditioning method.
- **[precond_diagonal]** (*type:* flag) Keyword to use diagonal preconditioning.
- **[optimized]** (*type:* flag) This keyword triggers a memory and network optimized algorithms useful for strong scaling (when computing less than 100 000 elements per processor). The matrix and the vectors are duplicated, common items removed and only virtual items really used in the matrix are exchanged. Warning: this is experimental and known to fail in some VEF computations (L2 projection step will not converge). Works well in VDF.

---

gen

not_set

Parameters are:

- **solv_elem** (*type:* string) To specify a solver among gmres or bicgstab.
- **precond** (*type:* *precond_base*) The only preconditionner that we can specify is ilu.
- **[seuil]** (*type:* float) Value of the final residue. The solver ceases iterations when the Euclidean residue standard ||Ax-B|| is less than this value. default value 1e-12.
- **[impr]** (*type:* flag) Keyword which is used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[save_matrice | save_matrix]** (*type:* flag) To save the matrix in a file.
- **[quiet]** (*type:* flag) To not displaying any outputs of the solver.
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the GEN solver.

- **[force]** (*type:* flag) Keyword to set ipar[5]=-1 in the GEN solver. This is helpful if you notice that the solver does not perform more than 100 iterations. If this keyword is specified in the datafile, you should provide nb_it_max.

---

gmres

Gmres method (for non symetric matrix).

Parameters are:

- **[impr]** (*type:* flag) Keyword which may be used to print the convergence.
- **[quiet]** (*type:* flag) To disable printing of information
- **[seuil]** (*type:* float) Convergence value.
- **[diag]** (*type:* flag) Keyword to use diagonal preconditionner (in place of pilut that is not parallel).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* int into [0, 1]) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.
- **[save_matrice | save_matrix]** (*type:* flag) to save the matrix in a file.
- **[dim_espace_krilov]** (*type:* int) not_set

---

optimal

Optimal is a solver which tests several solvers of the previous list to choose the fastest one for the considered linear system.

Parameters are:

- **seuil** (*type:* float) Convergence threshold
- **[impr]** (*type:* flag) To print the convergency of the fastest solver
- **[quiet]** (*type:* flag) To disable printing of information
- **[save_matrice | save_matrix]** (*type:* flag) To save the linear system (A, x, B) into a file
- **[frequence_recalc]** (*type:* int) To set a time step period (by default, 100) for re-checking the fatest solver
- **[nom_fichier_solveur]** (*type:* string) To specify the file containing the list of the tested solvers
- **[fichier_solveur_non_recree]** (*type:* flag) To avoid the creation of the file containing the list

---

petsc

Solver via Petsc API

Parameters are:

- **solveur** (*type:* *solveur_petsc_deriv*) solver type and options

---

petsc_gpu

GPU solver via Petsc API

Parameters are:

- **solveur** (*type:* string) not_set
- **option_solveur** (*type:* *bloc_lecture*) not_set
- **[atol]** (*type:* float) Absolute threshold for convergence (same as seuil option)
- **[rtol]** (*type:* float) Relative threshold for convergence

---

solv_gcp

Synonyms: gcp

Preconditioned conjugated gradient.

Parameters are:

- **seuil** (*type:* float) Value of the final residue. The gradient ceases iteration when the Euclidean residue standard ||Ax-B|| is less than this value.
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gcp.
- **[impr]** (*type:* flag) Keyword which is used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[quiet]** (*type:* flag) To not displaying any outputs of the solver.
- **[save_matrice | save_matrix]** (*type:* int) to save the matrix in a file.
- **[precond]** (*type:* *precond_base*) Keyword to define system preconditioning in order to accelerate resolution by the conjugated gradient. Many parallel preconditioning methods are not equivalent to their sequential counterpart, and you should therefore expect differences, especially when you select a high value of the final residue (seuil). The result depends on the number of processors and on the mesh splitting. It is sometimes useful to run the solver with no preconditioning at all. In particular: - when the solver does not converge during initial projection, - when comparing sequential and parallel computations. With no preconditioning, except in some particular cases (no open boundary), the sequential and the parallel computations should provide exactly the same results within fpu accuracy. If not, there might be a coding error or the system of equations is singular.
- **[precond_nul]** (*type:* flag) Keyword to not use a preconditioning method.
- **[precond_diagonal]** (*type:* flag) Keyword to use diagonal preconditioning.

- **[optimized]** (*type:* flag) This keyword triggers a memory and network optimized algorithms useful for strong scaling (when computing less than 100 000 elements per processor). The matrix and the vectors are duplicated, common items removed and only virtual items really used in the matrix are exchanged. Warning: this is experimental and known to fail in some VEF computations (L2 projection step will not converge). Works well in VDF.

---

solveur_sys_base

Basic class to solve the linear system.

1.3.5 Keywords derived from comment

comment

Synonyms: #

Comments in a data file.

Parameters are:

- **comm** (*type:* string) Text to be commented.

1.3.6 Keywords derived from condlim_base

condlim_base

Basic class of boundary conditions.

---

dirichlet

Dirichlet condition at the boundary called bord (edge) : 1). For Navier-Stokes equations, velocity imposed at the boundary; 2). For scalar transport equation, scalar imposed at the boundary.

---

echange_couplage_thermique

Thermal coupling boundary condition

Parameters are:

- **[temperature_paroi]** (*type: field_base*) Temperature
- **[flux_paroi]** (*type: field_base*) Wall heat flux

---

echange_externe_radiatif

Synonyms: paroi_echange_externe_radiatif

Combines radiative $(sigma * eps * (T^4 - T_ext^4))$ and convective $(h * (T - T_ext))$ heat transfer boundary conditions, where sigma is the Stefan-Boltzmann constant, eps is the emi

Parameters are:

- **h_imp** (*type:* string into ['h_imp', 't_ext', 'emissivite']) Heat exchange coefficient value (expressed in W.m-2.K-1).
- **himpc** (*type: front_field_base*) Boundary field type.
- **emissivite** (*type:* string into ['emissivite', 'h_imp', 't_ext']) Emissivity coefficient value.
- **emissivitebc** (*type: front_field_base*) Boundary field type.
- **t_ext** (*type:* string into ['t_ext', 'h_imp', 'emissivite']) External temperature value (expressed in oC or K).
- **ch** (*type: front_field_base*) Boundary field type.
- **temp_unit** (*type:* string into ['temperature_unit']) Temperature unit
- **temp_unit_val** (*type:* string into ['kelvin', 'celsius']) Temperature unit

---

echange_interne_global_impose

Synonyms: paroi_echange_interne_global_impose

Internal heat exchange boundary condition with global exchange coefficient.

Parameters are:

- **h_imp** (*type:* string) Global exchange coefficient value. The global exchange coefficient value is expressed in W.m-2.K-1.
- **ch** (*type: front_field_base*) Boundary field type.

---

echange_interne_global_parfait

Synonyms: paroi_echange_interne_global_parfait

Internal heat exchange boundary condition with perfect (infinite) exchange coefficient.

---

echange_interne_impose

Synonyms: paroi_echange_interne_impose

Internal heat exchange boundary condition with exchange coefficient.

Parameters are:

- **h_imp** (*type:* string) Exchange coefficient value expressed in W.m-2.K-1.
- **ch** (*type: front_field_base*) Boundary field type.

---

echange_interne_parfait

Synonyms: paroi_echange_interne_parfait

Internal heat exchange boundary condition with perfect (infinite) exchange coefficient.

---

entree_temperature_imposee_h

Particular case of class frontiere_ouverte_temperature_imposee for enthalpy equation.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte

Boundary outlet condition on the boundary called bord (edge) (diffusion flux zero). This condition must be associated with a boundary outlet hydraulic condition.

Parameters are:

- **var_name** (*type:* string into ['t_ext', 'c_ext', 'y_ext', 'k_eps_ext', 'k_omega_ext', 'fluctu_temperature_ext', 'flux_chaleur_turb_ext', 'v2_ext', 'a_ext', 'tau_ext', 'k_ext', 'omega_ext', 'h_ext', 'a_i_ext']) Field name.
- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_alpha_impose

Imposed alpha condition at the open boundary.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_concentration_imposee

Imposed concentration condition at an open boundary called bord (edge) (situation corresponding to a fluid inlet). This condition must be associated with an imposed inlet velocity condition.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_fraction_massique_imposee

not_set

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_gradient_pression_impose

Normal imposed pressure gradient condition on the open boundary called bord (edge). This boundary condition may be only used in VDF discretization. The imposed $partial P/partial n$ value is expressed in Pa.m-1.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_gradient_pression_impose_vefprep1b

Keyword for an outlet boundary condition in VEF P1B/P1NC on the gradient of the pressure.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_gradient_pression_libre_vef

Class for outlet boundary condition in VEF like Orlansky. There is no reference for pressure for theses boundary conditions so it is better to add pressure condition (with Frontiere_ouverte_pression_imposee) on one or two cells (for symmetry in a channel) of the boundary where Orlansky conditions are imposed.

---

frontiere_ouverte_gradient_pression_libre_vefprep1b

Class for outlet boundary condition in VEF P1B/P1NC like Orlansky.

---

frontiere_ouverte_pression_imposee

Imposed pressure condition at the open boundary called bord (edge). The imposed pressure field is expressed in Pa.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_pression_imposee_orlansky

This boundary condition may only be used with VDF discretization. There is no reference for pressure for this boundary condition so it is better to add pressure condition (with Frontiere_ouverte_pression_imposee) on one or two cells (for symetry in a channel) of the boundary where Orlansky conditions are imposed.

---

frontiere_ouverte_pression_moyenne_imposee

Class for open boundary with pressure mean level imposed.

Parameters are:

- **pext** (*type:* float) Mean pressure.

---

frontiere_ouverte_rho_u_impose

This keyword is used to designate a condition of imposed mass rate at an open boundary called bord (edge). The imposed mass rate field at the inlet is vectorial and the imposed velocity values are expressed in kg.s-1. This boundary condition can be used only with the Quasi compressible model.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_temperature_imposee

Synonyms: frontiere_ouverte_enthalpie_imposee

Imposed temperature condition at the open boundary called bord (edge) (in the case of fluid inlet). This condition must be associated with an imposed inlet velocity condition. The imposed temperature value is expressed in oC or K.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_vitesse_imposee

Class for velocity-inlet boundary condition. The imposed velocity field at the inlet is vectorial and the imposed velocity values are expressed in m.s-1.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

frontiere_ouverte_vitesse_imposee_sortie

Sub-class for velocity boundary condition. The imposed velocity field at the open boundary is vectorial and the imposed velocity values are expressed in m.s-1.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

neumann

Neumann condition at the boundary called bord (edge) : 1). For Navier-Stokes equations, constraint imposed at the boundary; 2). For scalar transport equation, flux imposed at the boundary.

---

neumann_homogene

Homogeneous neumann boundary condition

---

neumann_paroi

Neumann boundary condition for mass equation (multiphase problem)

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

neumann_paroi_adiabatique

Adiabatic wall neumann boundary condition

---

paroi

Impermeability condition at a wall called bord (edge) (standard flux zero). This condition must be associated with a wall type hydraulic condition.

---

paroi_adiabatique

Normal zero flux condition at the wall called bord (edge).

---

paroi_contact

Thermal condition between two domains. Important: the name of the boundaries in the two domains should be the same. (Warning: there is also an old limitation not yet fixed on the sequential algorithm in VDF to detect the matching faces on the two boundaries: faces should be ordered in the same way). The kind of condition depends on the discretization. In VDF, it is a heat exchange condition, and in VEF, a temperature condition.

Such a coupling requires coincident meshes for the moment. In case of non-coincident meshes, run is stopped and two external files are automatically generated in VEF (connectivity_failed_boundary_name and connectivity_failed_pb_name.med). In 2D, the keyword Decouper_bord_coincident associated to the connectivity_failed_boundary_name file allows to generate a new coincident mesh.

In 3D, for a first preliminary cut domain with HOMARD (fluid for instance), the second problem associated to pb_name (solide in a fluid/solid coupling problem) has to be submitted to HOMARD cutting procedure with connectivity_failed_pb_name.med.

Such a procedure works as while the primary refined mesh (fluid in our example) impacts the fluid/solid interface with a compact shape as described below (values 2 or 4 indicates the number of division from primary faces obtained in fluid domain at the interface after HOMARD cutting):

2-2-2-2-2-2

2-4-4-4-4-4-2 \; 2-2-2

2-4-4-4-4-2 \; 2-4-2

2-2-2-2-2 \; 2-2

OK

2-2 \; \; 2-2-2

2-4-2 \; 2-2

2-2 \; 2-2

NOT OK

Parameters are:

- **autrepb** (*type:* string) Name of other problem.
- **nameb** (*type:* string) boundary name of the remote problem which should be the same than the local name

---

paroi_contact_fictif

This keyword is derivated from paroi_contact and is especially dedicated to compute coupled fluid/solid/fluid problem in case of thin material. Thanks to this option, solid is considered as a fictitious media (no mesh, no domain associated), and coupling is performed by considering instantaneous thermal equilibrium in it (for the moment).

Parameters are:

- **autrepb** (*type:* string) Name of other problem.
- **nameb** (*type:* string) Name of bord.
- **conduct_fictif** (*type:* float) thermal conductivity
- **ep_fictive** (*type:* float) thickness of the fictitious media

---

paroi_decalee_robin

This keyword is used to designate a Robin boundary condition (a.u+b.du/dn=c) associated with the Pironneau methodology for the wall laws. The value of given by the delta option is the distance between the mesh (where symmetry boundary condition is applied) and the fictious wall. This boundary condition needs the definition of the dedicated source terms (Source_Robin or Source_Robin_Scalaire) according the equations used.

Parameters are:

- **delta** (*type:* float) not_set

---

paroi_defilante

Keyword to designate a condition where tangential velocity is imposed on the wall called bord (edge). If the velocity components set by the user is not tangential, projection is used.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_echange_contact_correlation_vdf

Class to define a thermohydraulic 1D model which will apply to a boundary of 2D or 3D domain.

Warning : For parallel calculation, the only possible partition will be according the axis of the model with the keyword Tranche.

Parameters are:

- **[dir]** (*type:* int) Direction (0 : axis X, 1 : axis Y, 2 : axis Z) of the 1D model.
- **[tinf]** (*type:* float) Inlet fluid temperature of the 1D model (oC or K).
- **[tsup]** (*type:* float) Outlet fluid temperature of the 1D model (oC or K).
- **[lambda_ | lambda]** (*type:* string) Thermal conductivity of the fluid (W.m-1.K-1).
- **[rho]** (*type:* string) Mass density of the fluid (kg.m-3) which may be a function of the temperature T.

- **[dt_impr]** (*type:* float) Printing period in name_of_data_file_time.dat files of the 1D model results.
- **[cp]** (*type:* float) Calorific capacity value at a constant pressure of the fluid (J.kg-1.K-1).
- **[mu]** (*type:* string) Dynamic viscosity of the fluid (kg.m-1.s-1) which may be a function of thetemperature T.
- **[debit]** (*type:* float) Surface flow rate (kg.s-1.m-2) of the fluid into the channel.
- **[dh]** (*type:* float) Hydraulic diameter may be a function f(x) with x position along the 1D axis (xinf <= x <= xsup)
- **[volume]** (*type:* string) Exact volume of the 1D domain (m3) which may be a function of the hydraulic diameter (Dh) and the lateral surface (S) of the meshed boundary.
- **[nu]** (*type:* string) Nusselt number which may be a function of the Reynolds number (Re) and the Prandtl number (Pr).
- **[reprise_correlation]** (*type:* flag) Keyword in the case of a resuming calculation with this correlation.

---

paroi_echange_contact_correlation_vef

Class to define a thermohydraulic 1D model which will apply to a boundary of 2D or 3D domain.

Warning : For parallel calculation, the only possible partition will be according the axis of the model with the keyword Tranche_geom.

Parameters are:

- **[dir]** (*type:* int) Direction (0 : axis X, 1 : axis Y, 2 : axis Z) of the 1D model.
- **[tinf]** (*type:* float) Inlet fluid temperature of the 1D model (oC or K).
- **[tsup]** (*type:* float) Outlet fluid temperature of the 1D model (oC or K).
- **[lambda_ | lambda]** (*type:* string) Thermal conductivity of the fluid (W.m-1.K-1).
- **[rho]** (*type:* string) Mass density of the fluid (kg.m-3) which may be a function of the temperature T.
- **[dt_impr]** (*type:* float) Printing period in name_of_data_file_time.dat files of the 1D model results.
- **[cp]** (*type:* float) Calorific capacity value at a constant pressure of the fluid (J.kg-1.K-1).
- **[mu]** (*type:* string) Dynamic viscosity of the fluid (kg.m-1.s-1) which may be a function of thetemperature T.
- **[debit]** (*type:* float) Surface flow rate (kg.s-1.m-2) of the fluid into the channel.
- **[n]** (*type:* int) Number of 1D cells of the 1D mesh.
- **[dh]** (*type:* string) Hydraulic diameter may be a function f(x) with x position along the 1D axis (xinf <= x <= xsup)
- **[surface]** (*type:* string) Section surface of the channel which may be function f(Dh,x) of the hydraulic diameter (Dh) and x position along the 1D axis (xinf <= x <= xsup)
- **[xinf]** (*type:* float) Position of the inlet of the 1D mesh on the axis direction.
- **[xsup]** (*type:* float) Position of the outlet of the 1D mesh on the axis direction.
- **[nu]** (*type:* string) Nusselt number which may be a function of the Reynolds number (Re) and the Prandtl number (Pr).
- **[emissivite_pour_rayonnement_entre_deux_plaques_quasi_infinies]** (*type:* float) Coefficient of emissivity for radiation between two quasi infinite plates.

- **[reprise_correlation]** (*type:* flag) Keyword in the case of a resuming calculation with this correlation.

---

paroi_echange_contact_vdf

Boundary condition type to model the heat flux between two problems. Important: the name of the boundaries in the two problems should be the same.

Parameters are:

- **autrepb** (*type:* string) Name of other problem.
- **nameb** (*type:* string) Name of bord.
- **temp** (*type:* string) Name of field.
- **h** (*type:* float) Value assigned to a coefficient (expressed in W.K-1m-2) that characterises the contact between the two mediums. In order to model perfect contact, h must be taken to be infinite. This value must obviously be the same in both the two problems blocks. The surface thermal flux exchanged between the two mediums is represented by : fi = h (T1-T2) where 1/h = d1/lambda1 + 1/val_h_contact + d2/lambda2 where di : distance between the node where Ti and the wall is found.

---

paroi_echange_externe_impose

External type exchange condition with a heat exchange coefficient and an imposed external temperature.

Parameters are:

- **h_or_t | h_imp** (*type:* string into ['h_imp', 't_ext']) Heat exchange coefficient value (expressed in W.m-2.K-1).
- **himpc** (*type: front_field_base*) Boundary field type.
- **t_or_h | text** (*type:* string into ['t_ext', 'h_imp']) External temperature value (expressed in oC or K).
- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_echange_externe_impose_h

Particular case of class paroi_echange_externe_impose for enthalpy equation.

Parameters are:

- **h_or_t | h_imp** (*type:* string into ['h_imp', 't_ext']) Heat exchange coefficient value (expressed in W.m-2.K-1).
- **himpc** (*type: front_field_base*) Boundary field type.
- **t_or_h | text** (*type:* string into ['t_ext', 'h_imp']) External temperature value (expressed in oC or K).
- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_echange_global_impose

Global type exchange condition (internal) that is to say that diffusion on the first fluid mesh is not taken into consideration.

Parameters are:

- **h_imp** (*type:* string) Global exchange coefficient value. The global exchange coefficient value is expressed in W.m-2.K-1.
- **himpc** (*type: front_field_base*) Boundary field type.
- **text** (*type:* string) External temperature value. The external temperature value is expressed in oC or K.
- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_fixe

Keyword to designate a situation of adherence to the wall called bord (edge) (normal and tangential velocity at the edge is zero).

---

paroi_fixe_iso_genepi2_sans_contribution_aux_vitesses_sommets

Boundary condition to obtain iso Geneppi2, without interest

---

paroi_flux_impose

Normal flux condition at the wall called bord (edge). The surface area of the flux (W.m-1 in 2D or W.m-2 in 3D) is imposed at the boundary according to the following convention: a positive flux is a flux that enters into the domain according to convention.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

paroi_knudsen_non_negligeable

Boundary condition for number of Knudsen (Kn) above 0.001 where slip-flow condition appears: the velocity near the wall depends on the shear stress : Kn=l/L with l is the mean-free-path of the molecules and L a characteristic length scale.

U(y=0)-Uwall=k(dU/dY)

Where k is a coefficient given by several laws:

Mawxell : k=(2-s)*l/s

Bestok&Karniadakis :k=(2-s)/s*L*Kn/(1+Kn)

Xue&Fan :k=(2-s)/s*L*tanh(Kn)

s is a value between 0 and 2 named accomodation coefficient. s=1 seems a good value.

Warning : The keyword is available for VDF calculation only for the moment.

Parameters are:

- **name_champ_1** (*type:* string into ['vitesse_paroi', 'k']) Field name.
- **champ_1** (*type: front_field_base*) Boundary field type.
- **name_champ_2** (*type:* string into ['vitesse_paroi', 'k']) Field name.
- **champ_2** (*type: front_field_base*) Boundary field type.

---

paroi_temperature_imposee

Imposed temperature condition at the wall called bord (edge).

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

periodic

Synonyms: periodique

1). For Navier-Stokes equations, this keyword is used to indicate that the horizontal inlet velocity values are the same as the outlet velocity values, at every moment. As regards meshing, the inlet and outlet edges bear the same name.; 2). For scalar transport equation, this keyword is used to set a periodic condition on scalar. The two edges dealing with this periodic condition bear the same name.

---

robin_vef

Robin condition at the boundary (edge)

Parameters are:

- **alpha** (*type:* float) Robin coefficient for the normal field
- **beta** (*type:* float) Robin coefficient for the tangent field
- **champ_front_normal_et_tangentiel_robin** (*type: front_field_base*) The boundary field

---

scalaire_impose_paroi

Imposed temperature condition at the wall called bord (edge).

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

sortie_libre_temperature_imposee_h

Open boundary for heat equation with enthalpy as unknown.

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

---

symetrie

1). For Navier-Stokes equations, this keyword is used to designate a symmetry condition concerning the velocity at the boundary called bord (edge) (normal velocity at the edge equal to zero and tangential velocity gradient at the edge equal to zero); 2). For scalar transport equation, this keyword is used to set a symmetry condition on scalar on the boundary named bord (edge).

---

temperature_imposee_paroi

Synonyms: enthalpie_imposee_paroi

Imposed temperature condition at the wall called bord (edge).

Parameters are:

- **ch** (*type: front_field_base*) Boundary field type.

1.3.7 Keywords derived from discretisation_base

dg

DG discretization

---

discretisation_base

Basic class for space discretization of thermohydraulic turbulent problems.

---

ef

Element Finite discretization.

---

ef_axi

Element Finite discretization.

---

ijk

IJK discretization.

---

polymac

polymac discretization (polymac discretization that is not compatible with pb_multi).

---

polymac_p0

polymac_p0 discretization (previously covimac discretization compatible with pb_multi).

---

polymac_p0p1nc

polymac_P0P1NC discretization (previously polymac discretization compatible with pb_multi).

---

vdf

Finite difference volume discretization.

---

vef

Synonyms: vefprep1b

Finite element volume discretization (P1NC/P1-bubble element). Since the 1.5.5 version, several new discretizations are available thanks to the optional keyword Read. By default, the VEFPreP1B keyword is equivalent to the former VEFPreP1B formulation (v1.5.4 and sooner). P0P1 (if used with the strong formulation for imposed pressure boundary) is equivalent to VEFPreP1B but the convergence is slower. VEFPreP1B dis is equivalent to VEFPreP1B dis Read dis { P0 P1 Changement_de_base_P1Bulle 1 Cl_pression_sommet_faible 0 }

Parameters are:

- **[changement_de_base_p1bulle]** (*type:* int into [0, 1]) changement_de_base_p1bulle 1 This option may be used to have the P1NC/P0P1 formulation (value set to 0) or the P1NC/P1Bulle formulation (value set to 1, the default).
- **[p0]** (*type:* flag) Pressure nodes are added on element centres
- **[p1]** (*type:* flag) Pressure nodes are added on vertices
- **[pa]** (*type:* flag) Only available in 3D, pressure nodes are added on bones
- **[rt]** (*type:* flag) For P1NCP1B (in TrioCFD)
- **[modif_div_face_dirichlet]** (*type:* int into [0, 1]) This option (by default 0) is used to extend control volumes for the momentum equation.
- **[cl_pression_sommet_faible]** (*type:* int into [0, 1]) This option is used to specify a strong formulation (value set to 0, the default) or a weak formulation (value set to 1) for an imposed pressure boundary condition. The first formulation converges quicker and is stable in general cases. The second formulation should be used if there are several outlet boundaries with Neumann condition (see Ecoulement_Neumann test case for example).

1.3.8 Keywords derived from domaine

domaine

Keyword to create a domain.

---

domaineaxi1d

1D domain

---

ijk_grid_geometry

Object to define the grid that will represent the domain of the simulation in IJK discretization

Parameters are:

- **[perio_i]** (*type:* flag) flag to specify the border along the I direction is periodic
- **[perio_j]** (*type:* flag) flag to specify the border along the J direction is periodic
- **[perio_k]** (*type:* flag) flag to specify the border along the K direction is periodic

- **[nbelem_i]** (*type:* int) the number of elements of the grid in the I direction
- **[nbelem_j]** (*type:* int) the number of elements of the grid in the J direction
- **[nbelem_k]** (*type:* int) the number of elements of the grid in the K direction
- **[uniform_domain_size_i]** (*type:* float) the size of the elements along the I direction
- **[uniform_domain_size_j]** (*type:* float) the size of the elements along the J direction
- **[uniform_domain_size_k]** (*type:* float) the size of the elements along the K direction
- **[origin_i]** (*type:* float) I-coordinate of the origin of the grid
- **[origin_j]** (*type:* float) J-coordinate of the origin of the grid
- **[origin_k]** (*type:* float) K-coordinate of the origin of the grid

1.3.9 Keywords derived from domaine_base

domaine_base

base for most domains

domaine_ijk

domain for IJK simulation (used in TrioCFD)

Parameters are:

- **nbelem** (*type:* list of int) Number of elements in each direction (integers, 2 or 3 values depending on dimension)
- **size_dom** (*type:* list of float) Domain size in each direction (floats, 2 or 3 values depending on dimension)
- **perio** (*type:* list of int) Is the direction periodic ? (0 or 1, 2 or 3 values depending on dimension)
- **nproc** (*type:* list of int) Number of procs in each direction (integers, 2 or 3 values depending on dimension)
- **[origin]** (*type:* list of float) Domain origin in each direction (floats, 2 or 3 values depending on dimension)
- **[ijk_splitting_ft_extension]** (*type:* int) not_set
- **[file_coords]** (*type: troismots*) not_set

1.3.10 Keywords derived from field_base

champ_composite

Composite field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values Various pieces of the field, defined per phase. Part 1 goes to phase 1, etc…

champ_don_base

Basic class for data fields (not calculated), p.e. physics properties.

---

champ_don_lu

Field to read a data field (values located at the center of the cells) in a file.

Parameters are:

- **dom** (*type:* string) Name of the domain.
- **nb_comp** (*type:* int) Number of field components.
- **file** (*type:* string) Name of the file. This file has the following format: nb_val_lues -> Number of values readen in th file Xi Yi Zi -> Coordinates readen in the file Ui Vi Wi -> Value of the field

---

champ_fonc_fonction

Field that is a function of another field.

Parameters are:

- **problem_name** (*type:* string) Name of problem.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **expression** (*type:* list of str) Number of field components followed by the analytical expression for each field component.

---

champ_fonc_fonction_txyz

this refers to a field that is a function of another field and time and/or space coordinates

Parameters are:

- **problem_name** (*type:* string) Name of problem.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **expression** (*type:* list of str) Number of field components followed by the analytical expression for each field component.

---

champ_fonc_fonction_txyz_morceaux

Field defined by analytical functions in each sub-domaine. On each zone, the value is defined as a function of x,y,z,t and of scalar value taken from a parameter field. This values is associated to the variable 'val' in the expression.

Parameters are:

- **problem_name** (*type:* string) Name of the problem.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_domaine_1 val_1 ... sous_domaine_i val_i } By default, the value val_def is assigned to the field. It takes the sous_domaine_i identifier Sous_Domaine (sub_area) type object function, val_i. Sous_Domaine (sub_area) type objects must have been previously defined if the operator wishes to use a champ_fonc_fonction_txyz_morceaux type object.

---

champ_fonc_interp

Field that is interpolated from a distant domain via MEDCoupling (remapper).

Parameters are:

- **nom_champ** (*type:* string) Name of the field (for example: temperature).
- **pb_loc** (*type:* string) Name of the local problem.
- **pb_dist** (*type:* string) Name of the distant problem.
- **[dom_loc]** (*type:* string) Name of the local domain.
- **[dom_dist]** (*type:* string) Name of the distant domain.
- **[default_value]** (*type:* string) Name of the distant domain.
- **nature** (*type:* string) Nature of the field (knowledge from MEDCoupling is required; IntensiveMaximum, IntensiveConservation, ...).
- **[use_overlapdec]** (*type:* string) Nature of the field (knowledge from MEDCoupling is required; IntensiveMaximum, IntensiveConservation, ...).

---

champ_fonc_med

Field to read a data field in a MED-format file .med at a specified time. It is very useful, for example, to resume a calculation with a new or refined geometry. The field post-processed on the new geometry at med format is used as initial condition for the resume.

Parameters are:

- **[use_existing_domain]** (*type:* flag) whether to optimize the field loading by indicating that the field is supported by the same mesh that was initially loaded as the domain
- **[last_time]** (*type:* flag) to use the last time of the MED file instead of the specified time. Mutually exclusive with 'time' parameter.
- **[decoup]** (*type:* string) specify a partition file.

- **[mesh]** (*type:* string) Name of the mesh supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **domain** (*type:* string) Name of the domain supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **file** (*type:* string) Name of the .med file.
- **field** (*type:* string) Name of field to load.
- **[loc]** (*type:* string into ['elem', 'som']) To indicate where the field is localised. Default to 'elem'.
- **[time]** (*type:* float) Timestep to load from the MED file. Mutually exclusive with 'last_time' flag.

---

champ_fonc_med_table_temps

Field defined as a fixed spatial shape scaled by a temporal coefficient

Parameters are:

- **[table_temps]** (*type: bloc_lecture*) Table containing the temporal coefficient used to scale the field
- **[table_temps_lue]** (*type:* string) Name of the file containing the values of the temporal coefficient used to scale the field
- **[use_existing_domain]** (*type:* flag) whether to optimize the field loading by indicating that the field is supported by the same mesh that was initially loaded as the domain
- **[last_time]** (*type:* flag) to use the last time of the MED file instead of the specified time. Mutually exclusive with 'time' parameter.
- **[decoup]** (*type:* string) specify a partition file.
- **[mesh]** (*type:* string) Name of the mesh supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **domain** (*type:* string) Name of the domain supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **file** (*type:* string) Name of the .med file.
- **field** (*type:* string) Name of field to load.
- **[loc]** (*type:* string into ['elem', 'som']) To indicate where the field is localised. Default to 'elem'.
- **[time]** (*type:* float) Timestep to load from the MED file. Mutually exclusive with 'last_time' flag.

---

champ_fonc_med_tabule

not_set

Parameters are:

- **[use_existing_domain]** (*type:* flag) whether to optimize the field loading by indicating that the field is supported by the same mesh that was initially loaded as the domain
- **[last_time]** (*type:* flag) to use the last time of the MED file instead of the specified time. Mutually exclusive with 'time' parameter.
- **[decoup]** (*type:* string) specify a partition file.
- **[mesh]** (*type:* string) Name of the mesh supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **domain** (*type:* string) Name of the domain supporting the field. This is the name of the mesh in the MED file, and if this mesh was also used to create the TRUST domain, loading can be optimized with option 'use_existing_domain'.
- **file** (*type:* string) Name of the .med file.
- **field** (*type:* string) Name of field to load.
- **[loc]** (*type:* string into ['elem', 'som']) To indicate where the field is localised. Default to 'elem'.
- **[time]** (*type:* float) Timestep to load from the MED file. Mutually exclusive with 'last_time' flag.

---

champ_fonc_reprise

This field is used to read a data field in a save file (.xyz or .sauv) at a specified time. It is very useful, for example, to run a thermohydraulic calculation with velocity initial condition read into a save file from a previous hydraulic calculation.

Parameters are:

- **[format]** (*type:* string into ['binaire', 'formatte', 'xyz', 'single_hdf', 'pdi']) Type of file (the file format). If xyz format is activated, the .xyz file from the previous calculation will be given for filename, and if formatte or binaire is choosen, the .sauv file of the previous calculation will be specified for filename. In the case of a parallel calculation, if the mesh partition does not changed between the previous calculation and the next one, the binaire format should be preferred, because is faster than the xyz format. If pdi is used, the same constraints/advantages as binaire apply, but it produces one (HDF5) file per node on the filesystem instead of having one file per processor. The single_hdf format is still supported but is obsolete, the PDI format is recommended.
- **filename** (*type:* string) Name of the save file.
- **pb_name** (*type:* string) Name of the problem.
- **champ** (*type:* string) Name of the problem unknown. It may also be the temporal average of a problem unknown (like moyenne_vitesse, moyenne_temperature,...)
- **[fonction]** (*type: fonction_champ_reprise*) Optional keyword to apply a function on the field being read in the save file (e.g. to read a temperature field in Celsius units and convert it for the calculation on Kelvin units, you will use: fonction 1 273.+val)
- **temps | time** (*type:* string) Time of the saved field in the save file or last_time. If you give the keyword last_time instead, the last time saved in the save file will be used.

---

champ_fonc_t

Field that is constant in space and is a function of time.

Parameters are:

- **val** (*type:* list of str) Values of field components (time dependant functions).

---

champ_fonc_tabule

Field that is tabulated as a function of another field.

Parameters are:

- **pb_field** (*type: bloc_lecture*) block similar to { pb1 field1 } or { pb1 field1 ... pbN fieldN }
- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values (the table (the value of the field at any time is calculated by linear interpolation from this table) or the analytical expression (with keyword expression to use an analytical expression)).

---

champ_fonc_tabule_morceaux

Synonyms: champ_tabule_morceaux

Field defined by tabulated data in each sub-domaine. It makes possible the definition of a field which is a function of other fields.

Parameters are:

- **domain_name** (*type:* string) Name of the domain.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_domaine_1 val_1 ... sous_domaine_i val_i } By default, the value val_def is assigned to the field. It takes the sous_domaine_i identifier Sous_Domaine (sub_area) type object function, val_i. Sous_Domaine (sub_area) type objects must have been previously defined if the operator wishes to use a champ_fonc_tabule_morceaux type object.

---

champ_fonc_tabule_morceaux_interp

Field defined by tabulated data in each sub-domaine. It makes possible the definition of a field which is a function of other fields. Here we use MEDCoupling to interpolate fields between the two domains.

Parameters are:

- **problem_name** (*type:* string) Name of the problem.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_domaine_1 val_1 ... sous_domaine_i val_i } By default, the value val_def is assigned to the field. It takes the sous_domaine_i identifier Sous_Domaine (sub_area) type object function, val_i. Sous_Domaine (sub_area) type objects must have been previously defined if the operator wishes to use a champ_fonc_tabule_morceaux type object.

---

champ_init_canal_sinal

For a parabolic profile on U velocity with an unpredictable disturbance on V and W and a sinusoidal disturbance on V velocity.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lec_champ_init_canal_sinal*) Parameters for the class champ_init_canal_sinal.

---

champ_input_base

not_set

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list of float) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* string) not_set

---

champ_input_p0

not_set

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list of float) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* string) not_set

---

champ_input_p0_composite

Field used to define a classical champ input p0 field (for ICoCo), but with a predefined field for the initial state.

Parameters are:

- **[initial_field]** (*type: field_base*) The field used for initialization
- **[input_field]** (*type: champ_input_p0*) The input field for ICoCo
- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set

- **[initial_value]** (*type:* list of float) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* string) not_set

---

champ_musig

MUSIG field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **bloc** (*type:* *bloc_lecture*) Not set

---

champ_ostwald

This keyword is used to define the viscosity variation law:

Mu(T)= K(T)*(D:D/2)**((n-1)/2)

---

champ_parametrique

Parametric field

Parameters are:

- **fichier** (*type:* string) Filename where fields are read

---

champ_som_lu_vdf

Keyword to read in a file values located at the nodes of a mesh in VDF discretization.

Parameters are:

- **domain_name** (*type:* string) Name of the domain.
- **dim** (*type:* int) Value of the dimension of the field.
- **tolerance** (*type:* float) Value of the tolerance to check the coordinates of the nodes.
- **file** (*type:* string) name of the file This file has the following format: Xi Yi Zi -> Coordinates of the node Ui Vi Wi -> Value of the field on this node Xi+1 Yi+1 Zi+1 -> Next point Ui+1 Vi+1 Zi+1 -> Next value ...

---

champ_som_lu_vef

Keyword to read in a file values located at the nodes of a mesh in VEF discretization.

Parameters are:

- **domain_name** (*type:* string) Name of the domain.
- **dim** (*type:* int) Value of the dimension of the field.
- **tolerance** (*type:* float) Value of the tolerance to check the coordinates of the nodes.
- **file** (*type:* string) Name of the file. This file has the following format: Xi Yi Zi -> Coordinates of the node Ui Vi Wi -> Value of the field on this node Xi+1 Yi+1 Zi+1 -> Next point Ui+1 Vi+1 Zi+1 -> Next value ...

---

champ_tabule_lu

Uniform field, tabulated from a specified column file. Lines starting with # are ignored.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.
- **column_file** (*type:* string) Name of the column file.
- **dim** (*type:* int) Number of field components.

---

champ_tabule_temps

Field that is constant in space and tabulated as a function of time.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values as a table. The value of the field at any time is calculated by linear interpolation from this table.

---

champ_uniforme_morceaux

Field which is partly constant in space and stationary.

Parameters are:

- **nom_dom** (*type:* string) Name of the domain to which the sub-areas belong.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_zone_1 val_1 ... sous_zone_i val_i } By default, the value val_def is assigned to the field. It takes the sous_zone_i identifier Sous_Zone (sub_area) type object value, val_i. Sous_Zone (sub_area) type objects must have been previously defined if the operator wishes to use a Champ_Uniforme_Morceaux(partly_uniform_field) type object.

---

champ_uniforme_morceaux_tabule_temps

this type of field is constant in space on one or several sub_zones and tabulated as a function of time.

Parameters are:

- **nom_dom** (*type:* string) Name of the domain to which the sub-areas belong.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type: bloc_lecture*) { Defaut val_def sous_zone_1 val_1 ... sous_zone_i val_i } By default, the value val_def is assigned to the field. It takes the sous_zone_i identifier Sous_Zone (sub_area) type object value, val_i. Sous_Zone (sub_area) type objects must have been previously defined if the operator wishes to use a Champ_Uniforme_Morceaux(partly_uniform_field) type object.

---

field_base

Synonyms: champ_base

Basic class of fields.

---

field_func_txyz

Synonyms: champ_fonc_txyz

Field defined by analytical functions. It makes it possible the definition of a field that depends on the time and the space.

Parameters are:

- **dom** (*type:* string) Name of domain of calculation
- **val** (*type:* list of str) List of functions on (t,x,y,z).

---

field_func_xyz

Synonyms: champ_fonc_xyz

Field defined by analytical functions. It makes it possible the definition of a field that depends on (x,y,z).

Parameters are:

- **dom** (*type:* string) Name of domain of calculation.
- **val** (*type:* list of str) List of functions on (x,y,z).

---

init_par_partie

ne marche que pour n_comp=1

Parameters are:

- **n_comp** (*type:* int into [1]) not_set
- **val1** (*type:* float) not_set
- **val2** (*type:* float) not_set
- **val3** (*type:* float) not_set

---

tayl_green

Class Tayl_green.

Parameters are:

- **dim** (*type:* int) Dimension.

---

uniform_field

Synonyms: champ_uniforme

Field that is constant in space and stationary.

Parameters are:

- **val** (*type:* list of float) Values of field components.

---

valeur_totale_sur_volume

Similar as Champ_Uniforme_Morceaux with the same syntax. Used for source terms when we want to specify a source term with a value given for the volume (eg: heat in Watts) and not a value per volume unit (eg: heat in Watts/m3).

Parameters are:

- **nom_dom** (*type:* string) Name of the domain to which the sub-areas belong.
- **nb_comp** (*type:* int) Number of field components.
- **data** (*type:* *bloc_lecture*) { Defaut val_def sous_zone_1 val_1 ... sous_zone_i val_i } By default, the value val_def is assigned to the field. It takes the sous_zone_i identifier Sous_Zone (sub_area) type object value, val_i. Sous_Zone (sub_area) type objects must have been previously defined if the operator wishes to use a Champ_Uniforme_Morceaux(partly_uniform_field) type object.

1.3.11 Keywords derived from front_field_base

boundary_field_inward

this field is used to define the normal vector field standard at the boundary in VDF or VEF discretization.

Parameters are:

- **normal_value** (*type:* string) normal vector value (positive value for a vector oriented outside to inside) which can depend of the time.

---

ch_front_input

not_set

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list of float) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* string) not_set

---

ch_front_input_uniforme

for coupling, you can use ch_front_input_uniforme which is a champ_front_uniforme, which use an external value. It must be used with Problem.setInputField.

Parameters are:

- **nb_comp** (*type:* int) not_set
- **nom** (*type:* string) not_set
- **[initial_value]** (*type:* list of float) not_set
- **probleme** (*type:* string) not_set
- **[sous_zone]** (*type:* string) not_set

---

champ_front_bruite

Field which is variable in time and space in a random manner.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.

- **bloc** (*type: bloc_lecture*) { [N val L val] Moyenne m_1.....[m_i] Amplitude A_1.....[A_ i]}: Random nois: If N and L are not defined, the ith component of the field varies randomly around an average value m_i with a maximum amplitude A_i. White noise: If N and L are defined, these two additional parameters correspond to L, the domain length and N, the number of nodes in the domain. Noise frequency will be between 2*Pi/L and 2*Pi*N/(4*L). For example, formula for velocity: u=U0(t) v=U1(t)Uj(t)=Mj+2*Aj*bruit_blanc where bruit_blanc (white_noise) is the formula given in the mettre_a_jour (update) method of the Champ_front_bruite (noise_boundary_field) (Refer to the Champ_front_bruite.cpp file).

---

champ_front_calc

This keyword is used on a boundary to get a field from another boundary. The local and remote boundaries should have the same mesh. If not, the Champ_front_recyclage keyword could be used instead. It is used in the condition block at the limits of equation which itself refers to a problem called pb1. We are working under the supposition that pb1 is coupled to another problem.

Parameters are:

- **problem_name** (*type:* string) Name of the other problem to which pb1 is coupled.
- **bord** (*type:* string) Name of the side which is the boundary between the 2 domains in the domain object description associated with the problem_name object.
- **field_name** (*type:* string) Name of the field containing the value that the user wishes to use at the boundary. The field_name object must be recognized by the problem_name object.

---

champ_front_composite

Composite front field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **dim** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) Values Various pieces of the field, defined per phase. Part 1 goes to phase 1, etc...

---

champ_front_contact_vef

This field is used on a boundary between a solid and fluid domain to exchange a calculated temperature at the contact face of the two domains according to the flux of the two problems.

Parameters are:

- **local_pb** (*type:* string) Name of the problem.
- **local_boundary** (*type:* string) Name of the boundary.
- **remote_pb** (*type:* string) Name of the second problem.
- **remote_boundary** (*type:* string) Name of the boundary in the second problem.

---

champ_front_debit

This field is used to define a flow rate field instead of a velocity field for a Dirichlet boundary condition on Navier-Stokes equations.

Parameters are:

- **ch** (*type:* *front_field_base*) uniform field in space to define the flow rate. It could be, for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_txyz that depends only on time.

---

champ_front_debit_massique

This field is used to define a flow rate field using the density

Parameters are:

- **ch** (*type:* *front_field_base*) uniform field in space to define the flow rate. It could be, for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_txyz that depends only on time.

---

champ_front_debit_qc_vdf

This keyword is used to define a flow rate field for quasi-compressible fluids in VDF discretization. The flow rate is kept constant during a transient.

Parameters are:

- **dimension | dim** (*type:* int) Problem dimension
- **liste** (*type:* *bloc_lecture*) List of the mass flow rate values [kg/s/m2] with the following syntaxe: { val1 ... valdim }
- **[moyen]** (*type:* string) Option to use rho mean value
- **pb_name** (*type:* string) Problem name

---

champ_front_debit_qc_vdf_fonc_t

This keyword is used to define a flow rate field for quasi-compressible fluids in VDF discretization. The flow rate could be constant or time-dependent.

Parameters are:

- **dimension | dim** (*type:* int) Problem dimension
- **liste** (*type:* *bloc_lecture*) List of the mass flow rate values [kg/s/m2] with the following syntaxe: { val1 ... valdim } where val1 ... valdim are constant or function of time.
- **[moyen]** (*type:* string) Option to use rho mean value
- **pb_name** (*type:* string) Problem name

---

champ_front_fonc_pois_ipsn

Boundary field champ_front_fonc_pois_ipsn.

Parameters are:

- **r_tube** (*type:* float) not_set
- **umoy** (*type:* list of float) not_set
- **r_loc** (*type:* list of float) not_set

---

champ_front_fonc_pois_tube

Boundary field champ_front_fonc_pois_tube.

Parameters are:

- **r_tube** (*type:* float) not_set
- **umoy** (*type:* list of float) not_set
- **r_loc** (*type:* list of float) not_set
- **r_loc_mult** (*type:* list of int) not_set

---

champ_front_fonc_t

Boundary field that depends only on time.

Parameters are:

- **val** (*type:* list of str) Values of field components (mathematical expressions).

---

champ_front_fonc_txyz

Boundary field which is not constant in space and in time.

Parameters are:

- **val** (*type:* list of str) Values of field components (mathematical expressions).

---

champ_front_fonc_xyz

Boundary field which is not constant in space.

Parameters are:

- **val** (*type:* list of str) Values of field components (mathematical expressions).

---

champ_front_fonction

boundary field that is function of another field

Parameters are:

- **dim** (*type:* int) Number of field components.
- **inco** (*type:* string) Name of the field (for example: temperature).
- **expression** (*type:* string) keyword to use a analytical expression like 10.*EXP(-0.1*val) where val be the keyword for the field.

---

champ_front_lu

boundary field which is given from data issued from a read file. The format of this file has to be the same that the one generated by Ecrire_fichier_xyz_valeur

Example for K and epsilon quantities to be defined for inlet condition in a boundary named 'entree':

entree frontiere_ouverte_K_Eps_impose Champ_Front_lu dom 2pb_K_EPS_PERIO_1006.306198.dat

Parameters are:

- **domaine | domain** (*type:* string) Name of domain
- **dim** (*type:* int) number of components
- **file** (*type:* string) path for the read file

---

champ_front_med

Field allowing the loading of a boundary condition from a MED file using Champ_fonc_med

Parameters are:

- **champ_fonc_med** (*type: field_base*) a champ_fonc_med loading the values of the unknown on a domain boundary

---

champ_front_musig

MUSIG front field. Used in multiphase problems to associate data to each phase.

Parameters are:

- **bloc** (*type: bloc_lecture*) Not set

---

champ_front_normal_vef

Field to define the normal vector field standard at the boundary in VEF discretization.

Parameters are:

- **mot** (*type:* string into ['valeur_normale']) Name of vector field.
- **vit_tan** (*type:* float) normal vector value (positive value for a vector oriented outside to inside).

---

champ_front_parametrique

Parametric boundary field

Parameters are:

- **fichier** (*type:* string) Filename where boundary fields are read

---

champ_front_pression_from_u

this field is used to define a pressure field depending of a velocity field.

Parameters are:

- **expression** (*type:* string) value depending of a velocity (like $2*u_moy^2$).

---

champ_front_recyclage

This keyword is used on a boundary to get a field from another boundary.

It is to use, in a general way, on a boundary of a local_pb problem, a field calculated from a linear combination of an imposed field g(x,y,z,t) with an instantaneous f(x,y,z,t) and a spatial mean field <f>(t) or a temporal mean field <f>(x,y,z) extracted from a plane of a problem named pb (pb may be local_pb itself):

For each component i, the field F applied on the boundary will be:

F_i(x,y,z,t) = alpha_i*g_i(x,y,z,t) + xsi_i*[f_i(x,y,z,t)- beta_i*<fi>]

Parameters are:

- **pb_champ_evaluateur** (*type: pb_champ_evaluateur*) not_set
- **[distance_plan]** (*type:* list of float) Vector which gives the distance between the boundary and the plane from where the field F will be extracted. By default, the vector is zero, that should imply the two domains have coincident boundaries.
- **[ampli_moyenne_imposee]** (*type:* list of float) 2|3 alpha(0) alpha(1) [alpha(2)]: alpha_i coefficients (by default =1)
- **[ampli_moyenne_recyclee]** (*type:* list of float) 2|3 beta(0) beta(1) [beta(2)]}: beta_i coefficients (by default =1)
- **[ampli_fluctuation]** (*type:* list of float) 2|3 gamma(0) gamma(1) [gamma(2)]}: gamma_i coefficients (by default =1)

- **[direction_anisotrope]** (*type:* int into [1, 2, 3]) If an integer is given for direction (X:1, Y:2, Z:3, by default, direction is negative), the imposed field g will be 0 for the 2 other directions.
- **[moyenne_imposee]** (*type: moyenne_imposee_deriv*) Value of the imposed g field.
- **[moyenne_recyclee]** (*type:* string) Method used to perform a spatial or a temporal averaging of f field to specify <f>. <f> can be the surface mean of f on the plane (surface option, see below) or it can be read from several files (for example generated by the chmoy_faceperio option of the Traitement_particulier keyword to obtain a temporal mean field). The option methode_recyc can be: surfacique, Surface mean for <f> from f values on the plane ; Or one of the following methode_moy options applied to read a temporal mean field <f>(x,y,z): interpolation, connexion_approchee or connexion_exacte
- **[fichier]** (*type:* string) not_set

---

champ_front_tabule

Constant field on the boundary, tabulated as a function of time.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.
- **bloc** (*type: bloc_lecture*) {nt1 t2 t3 ....tn u1 [v1 w1 ...] u2 [v2 w2 ...] u3 [v3 w3 ...] ... un [vn wn ...] } Values are entered into a table based on n couples (ti, ui) if nb_comp value is 1. The value of a field at a given time is calculated by linear interpolation from this table.

---

champ_front_tabule_lu

Constant field on the boundary, tabulated from a specified column file. Lines starting with # are ignored.

Parameters are:

- **nb_comp** (*type:* int) Number of field components.
- **column_file** (*type:* string) Name of the column file.

---

champ_front_tangentiel_vef

Field to define the tangential velocity vector field standard at the boundary in VEF discretization.

Parameters are:

- **mot** (*type:* string into ['vitesse_tangentielle']) Name of vector field.
- **vit_tan** (*type:* float) Vector field standard [m/s].

---

champ_front_uniforme

Boundary field which is constant in space and stationary.

Parameters are:

- **val** (*type:* list of float) Values of field components.

---

champ_front_xyz_debit

This field is used to define a flow rate field with a velocity profil which will be normalized to match the flow rate chosen.

Parameters are:

- **[velocity_profil]** (*type: front_field_base*) velocity_profil 0 velocity field to define the profil of velocity.
- **flow_rate** (*type: front_field_base*) flow_rate 1 uniform field in space to define the flow rate. It could be, for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_t

---

champ_front_xyz_tabule

Space dependent field on the boundary, tabulated as a function of time.

Parameters are:

- **val** (*type:* list of str) Values of field components (mathematical expressions).
- **bloc** (*type: bloc_lecture*) {nt1 t2 t3 ....tn u1 [v1 w1 ...] u2 [v2 w2 ...] u3 [v3 w3 ...] ... un [vn wn ...] } Values are entered into a table based on n couples (ti, ui) if nb_comp value is 1. The value of a field at a given time is calculated by linear interpolation from this table.

---

front_field_base

Synonyms: champ_front_base

Basic class for fields at domain boundaries.

1.3.12 Keywords derived from interface_base

interface_base

Basic class for a liquid-gas interface (used in pb_multiphase)

Parameters are:

- **[tension_superficielle | surface_tension]** (*type:* float) surface tension

---

interface_sigma_constant

Liquid-gas interface with a constant surface tension sigma

Parameters are:

- **[tension_superficielle | surface_tension]** (*type:* float) surface tension

---

saturation_base

fluide-gas interface with phase change (used in pb_multiphase)

Parameters are:

- **[p_ref]** (*type:* float) not_set
- **[t_ref]** (*type:* float) not_set
- **[tension_superficielle | surface_tension]** (*type:* float) surface tension

---

saturation_constant

Class for saturation constant

Parameters are:

- **[p_sat]** (*type:* float) Define the saturation pressure value (this is a required parameter)
- **[t_sat]** (*type:* float) Define the saturation temperature value (this is a required parameter)
- **[lvap]** (*type:* float) Latent heat of vaporization
- **[hlsat]** (*type:* float) Liquid saturation enthalpy
- **[hvsat]** (*type:* float) Vapor saturation enthalpy
- **[p_ref]** (*type:* float) not_set
- **[t_ref]** (*type:* float) not_set
- **[tension_superficielle | surface_tension]** (*type:* float) surface tension

---

saturation_sodium

Class for saturation sodium

Parameters are:

- **[p_ref]** (*type:* float) Use to fix the pressure value in the closure law. If not specified, the value of the pressure unknown will be used
- **[t_ref]** (*type:* float) Use to fix the temperature value in the closure law. If not specified, the value of the temperature unknown will be used
- **[tension_superficielle | surface_tension]** (*type:* float) surface tension

---

1.3.13 Keywords derived from interpolation_ibm_base

interpolation_ibm_aucune

Synonyms: ibm_aucune

Immersed Boundary Method (IBM): no interpolation.

Parameters are:

- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_base

Base class for all the interpolation methods available in the Immersed Boundary Method (IBM).

Parameters are:

- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_elem_fluid

Synonyms: ibm_element_fluide, interpolation_ibm_element_fluide

Immersed Boundary Method (IBM): fluid element interpolation.

Parameters are:

- **points_fluides** (*type: field_base*) Node field giving the projection of the point below (points_solides) falling into the pure cell fluid
- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **elements_fluides** (*type: field_base*) Node field giving the number of the element (cell) containing the pure fluid point
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_hybride

Synonyms: ibm_hybride

Immersed Boundary Method (IBM): hybrid (fluid/mean gradient) interpolation.

Parameters are:

- **est_dirichlet** (*type: field_base*) Node field of booleans indicating whether the node belong to an element where the interface is
- **elements_solides** (*type: field_base*) Node field giving the element number containing the solid point
- **points_fluides** (*type: field_base*) Node field giving the projection of the point below (points_solides) falling into the pure cell fluid
- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **elements_fluides** (*type: field_base*) Node field giving the number of the element (cell) containing the pure fluid point
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_mean_gradient

Synonyms: interpolation_ibm_gradient_moyen, ibm_gradient_moyen

Immersed Boundary Method (IBM): mean gradient interpolation.

Parameters are:

- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **est_dirichlet** (*type: field_base*) Node field of booleans indicating whether the node belong to an element where the interface is
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **elements_solides** (*type: field_base*) Node field giving the element number containing the solid point
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_power_law_tbl

Synonyms: ibm_power_law_tbl

Immersed Boundary Method (IBM): power law interpolation.

Parameters are:

- **[formulation_linear_pwl]** (*type:* int) Choix formulation lineaire ou non
- **points_fluides** (*type: field_base*) Node field giving the projection of the point below (points_solides) falling into the pure cell fluid

- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **elements_fluides** (*type: field_base*) Node field giving the number of the element (cell) containing the pure fluid point
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

interpolation_ibm_power_law_tbl_u_star

Synonyms: ibm_power_law_tbl_u_star

Immersed Boundary Method (IBM): law u star.

Parameters are:

- **points_solides** (*type: field_base*) Node field giving the projection of the node on the immersed boundary
- **est_dirichlet** (*type: field_base*) Node field of booleans indicating whether the node belong to an element where the interface is
- **correspondance_elements** (*type: field_base*) Cell field giving the SALOME cell number
- **elements_solides** (*type: field_base*) Node field giving the element number containing the solid point
- **[impr]** (*type:* flag) To print IBM-related data
- **[nb_histo_boxes_impr]** (*type:* int) number of histogram boxes for printed data

---

1.3.14 Keywords derived from interprete

analyse_angle

Keyword Analyse_angle prints the histogram of the largest angle of each mesh elements of the domain named name_domain. nb_histo is the histogram number of bins. It is called by default during the domain discretization with nb_histo set to 18. Useful to check the number of elements with angles above 90 degrees.

Parameters are:

- **domain_name** (*type:* string) Name of domain to resequence.
- **nb_histo** (*type:* int) not_set

---

associate

Synonyms: associer

This interpretor allows one object to be associated with another. The order of the two objects in this instruction is not important. The object objet_2 is associated to objet_1 if this makes sense; if not either objet_1 is associated to objet_2 or the program exits with error because it cannot execute the Associate (Associer) instruction. For example, to calculate water flow in a pipe, a Pb_Hydraulique type object needs to be defined. But also a Domaine type object to represent the pipe, a Scheme_euler_explicit type object for time discretization, a discretization type object (VDF or VEF) and a Fluide_Incompressible type object which will contain the water properties. These objects must then all be associated with the problem.

Parameters are:

- **objet_1** (*type:* string) Objet_1
- **objet_2** (*type:* string) Objet_2

---

axi

This keyword allows a 3D calculation to be executed using cylindrical coordinates (R,$jolitheta$,Z). If this instruction is not included, calculations are carried out using Cartesian coordinates.

---

bidim_axi

Keyword allowing a 2D calculation to be executed using axisymetric coordinates (R, Z). If this instruction is not included, calculations are carried out using Cartesian coordinates.

---

calculer_moments

Calculates and prints the torque (moment of force) exerted by the fluid on each boundary in output files (.out) of the domain nom_dom.

Parameters are:

- **nom_dom** (*type:* string) Name of domain.
- **mot** (*type: lecture_bloc_moment_base*) Keyword.

---

corriger_frontiere_periodique

The Corriger_frontiere_periodique keyword is mandatory to first define the periodic boundaries, to reorder the faces and eventually fix unaligned nodes of these boundaries. Faces on one side of the periodic domain are put first, then the faces on the opposite side, in the same order. It must be run in sequential before mesh splitting.

Parameters are:

- **domaine** (*type:* string) Name of domain.

- **bord** (*type:* string) the name of the boundary (which must contain two opposite sides of the domain)
- **[direction]** (*type:* list of float) defines the periodicity direction vector (a vector that points from one node on one side to the opposite node on the other side). This vector must be given if the automatic algorithm fails, that is: - when the node coordinates are not perfectly periodic - when the periodic direction is not aligned with the normal vector of the boundary faces
- **[fichier_post]** (*type:* string) .

---

create_domain_from_sub_domain

Synonyms: create_domain_from_sous_zone, create_domain_from_sub_domains

This keyword fills the domain domaine_final with the subdomaine par_sous_zone from the domain domaine_init. It is very useful when meshing several mediums with Gmsh. Each medium will be defined as a subdomaine into Gmsh. A MED mesh file will be saved from Gmsh and read with Lire_Med keyword by the TRUST data file. And with this keyword, a domain will be created for each medium in the TRUST data file.

Parameters are:

- **[domaine_final]** (*type:* string) new domain in which faces are stored
- **[par_sous_zone | par_sous_dom]** (*type:* string) a sub-area (a group in a MED file) allowing to choose the elements
- **domaine_init** (*type:* string) initial domain

---

criteres_convergence

convergence criteria

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **[inco]** (*type:* string) Unknown (i.e: alpha, temperature, velocity and pressure)
- **[val]** (*type:* float) Convergence threshold
- **acof** (*type:* string into ['}']) Closing curly bracket.

---

debog

Class to debug some differences between two TRUST versions on a same data file.

If you want to compare the results of the same code in sequential and parallel calculation, first run (mode=0) in sequential mode (the files fichier1 and fichier2 will be written first) then the second run in parallel calculation (mode=1).

During the first run (mode=0), it prints into the file DEBOG, values at different points of the code thanks to the C++ instruction call. see for example in Kernel/Framework/Resoudre.cpp file the instruction: Debog::verifier(msg,value); Where msg is a string and value may be a double, an integer or an array.

During the second run (mode=1), it prints into a file Err_Debog.dbg the same messages than in the DEBOG file and checks if the differences between results from both codes are less than a given value (error). If not, it prints Ok else show the differences and the lines where it occured.

Parameters are:

- **pb** (*type:* string) Name of the problem to debug.
- **fichier1 | file1** (*type:* string) Name of the file where domain will be written in sequential calculation.
- **fichier2 | file2** (*type:* string) Name of the file where faces will be written in sequential calculation.
- **seuil** (*type:* float) Minimal value (by default 1.e-20) for the differences between the two codes.
- **mode** (*type:* int) By default -1 (nothing is written in the different files), you will set 0 for the sequential run, and 1 for the parallel run.

---

debut_bloc

Synonyms: {

Block's beginning.

---

decoupebord_pour_rayonnement

Synonyms: decoupebord

To subdivide the external boundary of a domain into several parts (may be useful for better accuracy when using radiation model in transparent medium). To specify the boundaries of the fine_domain_name domain to be splitted. These boundaries will be cut according the coarse mesh defined by either the keyword domaine_grossier (each boundary face of the coarse mesh coarse_domain_name will be used to group boundary faces of the fine mesh to define a new boundary), either by the keyword nb_parts_naif (each boundary of the fine mesh is splitted into a partition with nx*ny*nz elements), either by a geometric condition given by a formulae with the keyword condition_geometrique. If used, the coarse_domain_name domain should have the same boundaries name of the fine_domain_name domain.

A mesh file (ASCII format, except if binaire option is specified) named by default newgeom (or specified by the nom_fichier_sortie keyword) will be created and will contain the fine_domain_name domain with the splitted boundaries named boundary_name%I (where I is between from 0 and n-1). Furthermore, several files named boundary_name%I and boundary_name_xv will be created, containing the definition of the subdived boundaries. newgeom will be used to calculate view factors with geom2ansys script whereas only the boundary_name_xv files will be necessary for the radiation calculation. The file listb will contain the list of the boundaries boundary_name%I.

Parameters are:

- **domaine** (*type:* string) not_set
- **[domaine_grossier]** (*type:* string) not_set
- **[nb_parts_naif]** (*type:* list of int) not_set
- **[nb_parts_geom]** (*type:* list of int) not_set
- **[condition_geometrique]** (*type:* list of str) not_set
- **bords_a_decouper** (*type:* list of str) not_set
- **[nom_fichier_sortie]** (*type:* string) not_set

- **[binaire]** (*type:* int) not_set

---

decouper_bord_coincident

In case of non-coincident meshes and a paroi_contact condition, run is stopped and two external files are automatically generated in VEF (connectivity_failed_boundary_name and connectivity_failed_pb_name.med). In 2D, the keyword Decouper_bord_coincident associated to the connectivity_failed_boundary_name file allows to generate a new coincident mesh.

Parameters are:

- **domain_name** (*type:* string) Name of domain.
- **bord** (*type:* string) connectivity_failed_boundary_name

---

dilate

Keyword to multiply the whole coordinates of the geometry.

Parameters are:

- **domain_name** (*type:* string) Name of domain.
- **alpha** (*type:* float) Value of dilatation coefficient.

---

dimension

Keyword allowing calculation dimensions to be set (2D or 3D), where dim is an integer set to 2 or 3. This instruction is mandatory.

Parameters are:

- **dim** (*type:* int into [2, 3]) Number of dimensions.

---

disable_tu

Flag to disable the writing of the .TU files

---

discretiser_domaine

Useful to discretize the domain domain_name (faces will be created) without defining a problem.

Parameters are:

- **domain_name** (*type:* string) Name of the domain.

---

discretize

Synonyms: discretiser

Keyword to discretise a problem problem_name according to the discretization dis.

IMPORTANT: A number of objects must be already associated (a domain, time scheme, central object) prior to invoking the Discretize (Discretiser) keyword. The physical properties of this central object must also have been read.

Parameters are:

- **problem_name** (*type:* string) Name of problem.
- **dis** (*type:* string) Name of the discretization object.

---

distance_paroi

Class to generate external file Wall_length.xyz devoted for instance, for mixing length modelling. In this file, are saved the coordinates of each element (center of gravity) of dom domain and minimum distance between this point and boundaries (specified bords) that user specifies in data file (typically, those associated to walls). A field Distance_paroi is available to post process the distance to the wall.

Parameters are:

- **dom** (*type:* string) Name of domain.
- **bords** (*type:* list of str) Boundaries.
- **format** (*type:* string into ['binaire', 'formatte']) Value for format may be binaire (a binary file Wall_length.xyz is written) or formatte (moreover, a formatted file Wall_length_formatted.xyz is written).

---

ecrire_champ_med

Keyword to write a field to MED format into a file.

Parameters are:

- **nom_dom** (*type:* string) domain name
- **nom_chp** (*type:* string) field name
- **file** (*type:* string) file name

---

ecrire_fichier_formatte

Keyword to write the object of name name_obj to a file filename in ASCII format.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be written.
- **filename** (*type:* string) Name of the file.

---

ecrire_fichier_xyz_valeur

This keyword is used to write the values of a field only for some boundaries in a text file with the following format: n_valeur

x_1 y_1 [z_1] val_1

...

x_n y_n [z_n] val_n

The created files are named : pbname_fieldname_[boundaryname]_time.dat

Parameters are:

- **[binary_file]** (*type:* flag) To write file in binary format
- **[dt]** (*type:* float) File writing frequency
- **[fields]** (*type:* list of str) Names of the fields we want to write
- **[boundaries]** (*type:* list of str) Names of the boundaries on which to write fields

---

ecrire_med_32_64

Synonyms: ecrire_med, write_med

Write a domain to MED format into a file.

Parameters are:

- **nom_dom** (*type:* string) Name of domain.
- **file** (*type:* string) Name of file.

---

ecriturelecturespecial

Class to write or not to write a .xyz file on the disk at the end of the calculation.

Parameters are:

- **type** (*type:* string) If set to 0, no xyz file is created. If set to 1 (the default) the .xyz file is written at the end of the computation.

---

espece

not_set

Parameters are:

- **mu** (*type: field_base*) Species dynamic viscosity value (kg.m-1.s-1).
- **cp** (*type: field_base*) Species specific heat value (J.kg-1.K-1).
- **masse_molaire** (*type:* float) Species molar mass.

---

execute_parallel

This keyword allows to run several computations in parallel on processors allocated to TRUST. The set of processors is split in N subsets and each subset will read and execute a different data file. Error messages usualy written to stderr and stdout are redirected to .log files (journaling must be activated).

Parameters are:

- **liste_cas** (*type:* list of str) N datafile1 ... datafileN. datafileX the name of a TRUST data file without the .data extension.
- **[nb_procs]** (*type:* list of int) nb_procs is the number of processors needed to run each data file. If not given, TRUST assumes that computations are sequential.

---

export

Class to make the object have a global range, if not its range will apply to the block only (the associated object will be destroyed on exiting the block).

---

extract_2d_from_3d

Keyword to extract a 2D mesh by selecting a boundary of the 3D mesh. To generate a 2D axisymmetric mesh prefer Extract_2Daxi_from_3D keyword.

Parameters are:

- **dom3d** (*type:* string) Domain name of the 3D mesh
- **bord** (*type:* string) Boundary name. This boundary becomes the new 2D mesh and all the boundaries, in 3D, attached to the selected boundary, give their name to the new boundaries, in 2D.
- **dom2d** (*type:* string) Domain name of the new 2D mesh

---

extract_2daxi_from_3d

Keyword to extract a 2D axisymetric mesh by selecting a boundary of the 3D mesh.

Parameters are:

- **dom3d** (*type:* string) Domain name of the 3D mesh
- **bord** (*type:* string) Boundary name. This boundary becomes the new 2D mesh and all the boundaries, in 3D, attached to the selected boundary, give their name to the new boundaries, in 2D.
- **dom2d** (*type:* string) Domain name of the new 2D mesh

---

extraire_domaine

Keyword to create a new domain built with the domain elements of the pb_name problem verifying the two conditions given by Condition_elements. The problem pb_name should have been discretized.

Parameters are:

- **domaine** (*type:* string) Domain in which faces are saved
- **probleme** (*type:* string) Problem from which faces should be extracted
- **[condition_elements]** (*type:* string) not_set
- **[sous_domaine | sous_zone]** (*type:* string) not_set

---

extraire_plan

This keyword extracts a plane mesh named domain_name (this domain should have been declared before) from the mesh of the pb_name problem. The plane can be either a triangle (defined by the keywords Origine, Point1, Point2 and Triangle), either a regular quadrangle (with keywords Origine, Point1 and Point2), or either a generalized quadrangle (with keywords Origine, Point1, Point2, Point3). The keyword Epaisseur specifies the thickness of volume around the plane which contains the faces of the extracted mesh. The keyword via_extraire_surface will create a plan and use Extraire_surface algorithm. Inverse_condition_element keyword then will be used in the case where the plane is a boundary not well oriented, and avec_certains_bords_pour_extraire_surface is the option related to the Extraire_surface option named avec_certains_bords.

Parameters are:

- **domaine** (*type:* string) domain name
- **probleme** (*type:* string) pb_name
- **origine** (*type:* list of float) not_set
- **point1** (*type:* list of float) not_set
- **point2** (*type:* list of float) not_set
- **[point3]** (*type:* list of float) not_set
- **[triangle]** (*type:* flag) not_set
- **epaisseur** (*type:* float) thickness
- **[via_extraire_surface]** (*type:* flag) not_set
- **[inverse_condition_element]** (*type:* flag) not_set
- **[avec_certains_bords_pour_extraire_surface]** (*type:* list of str) name of boundaries to include when extracting plan

---

extraire_surface

This keyword extracts a surface mesh named domain_name (this domain should have been declared before) from the mesh of the pb_name problem. The surface mesh is defined by one or two conditions. The first condition is about elements with Condition_elements. For example: Condition_elements x*x+y*y+z*z<1

Will define a surface mesh with external faces of the mesh elements inside the sphere of radius 1 located at (0,0,0). The second condition Condition_faces is useful to give a restriction.

By default, the faces from the boundaries are not added to the surface mesh excepted if option avec_les_bords is given (all the boundaries are added), or if the option avec_certains_bords is used to add only some boundaries.

Parameters are:

- **domaine** (*type:* string) Domain in which faces are saved
- **probleme** (*type:* string) Problem from which faces should be extracted
- **[condition_elements]** (*type:* string) condition on center of elements
- **[condition_faces]** (*type:* string) not_set
- **[avec_les_bords]** (*type:* flag) not_set
- **[avec_certains_bords]** (*type:* list of str) not_set

---

extrudebord

Class to generate an extruded mesh from a boundary of a tetrahedral or an hexahedral mesh.

Warning: If the initial domain is a tetrahedral mesh, the boundary will be moved in the XY plane then extrusion will be applied (you should maybe use the Transformer keyword on the final domain to have the domain you really want). You can use the keyword Postraiter_domaine to generate a lata|med|... file to visualize your initial and final meshes.

This keyword can be used for example to create a periodic box extracted from a boundary of a tetrahedral or a hexaedral mesh. This periodic box may be used then to engender turbulent inlet flow condition for the main domain.

Note that ExtrudeBord in VEF generates 3 or 14 tetrahedra from extruded prisms.

Parameters are:

- **domaine_init** (*type:* string) Initial domain with hexaedras or tetrahedras.
- **direction** (*type:* list of float) Directions for the extrusion.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.
- **domaine_final** (*type:* string) Extruded domain.
- **nom_bord** (*type:* string) Name of the boundary of the initial domain where extrusion will be applied.
- **[hexa_old]** (*type:* flag) Old algorithm for boundary extrusion from a hexahedral mesh.
- **[trois_tetra]** (*type:* flag) To extrude in 3 tetrahedras instead of 14 tetrahedras.
- **[vingt_tetra]** (*type:* flag) To extrude in 20 tetrahedras instead of 14 tetrahedras.
- **[sans_passer_par_le2d]** (*type:* int) Only for non-regression

---

extrudeparoi

Keyword dedicated in 3D (VEF) to create prismatic layer at wall. Each prism is cut into 3 tetraedra.

Parameters are:

- **domaine** (*type:* string) Name of the domain.
- **nom_bord** (*type:* string) Name of the (no-slip) boundary for creation of prismatic layers.
- **[epaisseur]** (*type:* list of float) n r1 r2 .... rn : (relative or absolute) width for each layer.
- **[critere_absolu]** (*type:* flag) use absolute width for each layer instead of relative.
- **[projection_normale_bord]** (*type:* flag) keyword to project layers on the same plane that contiguous boundaries. defaut values are : epaisseur_relative 1 0.5 projection_normale_bord 1

---

extruder

Class to create a 3D tetrahedral/hexahedral mesh (a prism is cut in 14) from a 2D triangular/quadrangular mesh.

Parameters are:

- **domaine | domain_name** (*type:* string) Name of the domain.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.
- **direction** (*type:* *troisf*) Direction of the extrude operation.

---

extruder_en20

It does the same task as Extruder except that a prism is cut into 20 tetraedra instead of 3. The name of the boundaries will be devant (front) and derriere (back). But you can change these names with the keyword RegroupeBord.

Parameters are:

- **domaine | domain_name** (*type:* string) Name of the domain.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.
- **[direction]** (*type:* *troisf*) 0 Direction of the extrude operation.

---

extruder_en3

Class to create a 3D tetrahedral/hexahedral mesh (a prism is cut in 3) from a 2D triangular/quadrangular mesh. The names of the boundaries (by default, devant (front) and derriere (back)) may be edited by the keyword nom_cl_devant and nom_cl_derriere. If 'null' is written for nom_cl, then no boundary condition is generated at this place.

Recommendation : to ensure conformity between meshes (in case of fluid/solid coupling) it is recommended to extrude all the domains at the same time.

Parameters are:

- **domaine | domain_name** (*type:* list of str) List of the domains

- **[nom_cl_devant]** (*type:* string) New name of the first boundary.
- **[nom_cl_derriere]** (*type:* string) New name of the second boundary.
- **nb_tranches** (*type:* int) Number of elements in the extrusion direction.
- **direction** (*type:* *troisf*) Direction of the extrude operation.

---

facsec_expert

To parameter the safety factor for the time step during the simulation.

Parameters are:

- **[facsec_ini]** (*type:* float) Initial facsec taken into account at the beginning of the simulation.
- **[facsec_max]** (*type:* float) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value. Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice: The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: -Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 -Thermohydralic with natural convection, facsec around 300 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableThese values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[rapport_residus]** (*type:* float) Ratio between the residual at time n and the residual at time n+1 above which the facsec is increased by multiplying by sqrt(rapport_residus) (1.2 by default).
- **[nb_ite_sans_accel_max]** (*type:* int) Maximum number of iterations without facsec increases (20000 by default): if facsec does not increase with the previous condition (ration between 2 consecutive residuals too high), we increase it by force after nb_ite_sans_accel_max iterations.

---

fin

Synonyms: end

Keyword which must complete the data file. The execution of the data file stops when reaching this keyword.

---

fin_bloc

Synonyms: }

Block's end.

---

imprimer_flux

This keyword prints the flux per face at the specified domain boundaries in the data set. The fluxes are written to the .face files at a frequency defined by dt_impr, the evaluation printing frequency (refer to time scheme keywords). By default, fluxes are incorporated onto the edges before being displayed.

Parameters are:

- **domain_name** (*type:* string) Name of the domain.
- **noms_bord** (*type: bloc_lecture*) List of boundaries, for ex: { Bord1 Bord2 }

---

imprimer_flux_sum

This keyword prints the sum of the flux per face at the domain boundaries defined by the user in the data set. The fluxes are written into the .out files at a frequency defined by dt_impr, the evaluation printing frequency (refer to time scheme keywords).

Parameters are:

- **domain_name** (*type:* string) Name of the domain.
- **noms_bord** (*type: bloc_lecture*) List of boundaries, for ex: { Bord1 Bord2 }

---

integrer_champ_med

his keyword is used to calculate a flow rate from a velocity MED field read before. The method is either debit_total to calculate the flow rate on the whole surface, either integrale_en_z to calculate flow rates between z=zmin and z=zmax on nb_tranche surfaces. The output file indicates first the flow rate for the whole surface and then lists for each tranche : the height z, the surface average value, the surface area and the flow rate. For the debit_total method, only one tranche is considered.

file :z Sum(u.dS)/Sum(dS) Sum(dS) Sum(u.dS)

Parameters are:

- **champ_med** (*type:* string) not_set
- **methode** (*type:* string into ['integrale_en_z', 'debit_total']) to choose between the integral following z or over the entire height (debit_total corresponds to zmin=-DMAXFLOAT, ZMax=DMAXFLOAT, nb_tranche=1)
- **[zmin]** (*type:* float) not_set
- **[zmax]** (*type:* float) not_set
- **[nb_tranche]** (*type:* int) not_set
- **[fichier_sortie]** (*type:* string) name of the output file, by default: integrale.

---

interprete

Basic class for interpreting a data file. Interpretors allow some operations to be carried out on objects.

---

interprete_geometrique_base

Class for interpreting a data file

---

lata_to_cgns

Synonyms: lata_2_cgns

To convert results file written with LATA format to CGNS file. Warning: Fields located on faces are not supported yet.

Parameters are:

- **[format]** (*type: format_lata_to_cgns*) generated file post_CGNS.data use format (CGNS or LATA or LML keyword).
- **file** (*type:* string) LATA file to convert to the new format.
- **file_cgns** (*type:* string) Name of the CGNS file.

---

lata_to_med

Synonyms: lata_2_med

To convert results file written with LATA format to MED file. Warning: Fields located on faces are not supported yet.

Parameters are:

- **[format]** (*type: format_lata_to_med*) generated file post_med.data use format (MED or LATA or LML keyword).
- **file** (*type:* string) LATA file to convert to the new format.
- **file_med** (*type:* string) Name of the MED file.

---

lata_to_other

Synonyms: lata_2_other

To convert results file written with LATA format to CGNS, MED or LML format. Warning: Fields located at faces are not supported yet.

Parameters are:

- **[format]** (*type:* string into ['lml', 'lata', 'lata_v2', 'med', 'cgns']) Results format (CGNS, MED or LATA or LML keyword).
- **file** (*type:* string) LATA file to convert to the new format.

- **file_post** (*type:* string) Name of file post.

---

link_cgns_files

Creates a single CGNS xxxx.cgns file that links to a xxxx.grid.cgns and xxxx.solution.*.cgns files

Parameters are:

- **base_name** (*type:* string) Base name of the gid/solution cgns files.
- **output_name** (*type:* string) Name of the output cgns file.

---

lire_ideas

Read a geom in a unv file. 3D tetra mesh elements only may be read by TRUST.

Parameters are:

- **nom_dom** (*type:* string) Name of domain.
- **file** (*type:* string) Name of file.

---

lml_to_lata

Synonyms: lml_2_lata

To convert results file written with LML format to a single LATA file.

Parameters are:

- **file_lml** (*type:* string) LML file to convert to the new format.
- **file_lata** (*type:* string) Name of the single LATA file.

---

mailler

The Mailler (Mesh) interpretor allows a Domain type object domaine to be meshed with objects objet_1, objet_2, etc...

Parameters are:

- **domaine** (*type:* string) Name of domain.
- **bloc** (*type:* list of Mailler_base) List of block mesh.

---

maillerparallel

creates a parallel distributed hexaedral mesh of a parallelipipedic box. It is equivalent to creating a mesh with a single Pave, splitting it with Decouper and reloading it in parallel with Scatter. It only works in 3D at this time. It can also be used for a sequential computation (with all NPARTS=1)}

Parameters are:

- **domain** (*type:* string) the name of the domain to mesh (it must be an empty domain object).
- **nb_nodes** (*type:* list of int) dimension defines the spatial dimension (currently only dimension=3 is supported), and nX, nY and nZ defines the total number of nodes in the mesh in each direction.
- **splitting** (*type:* list of int) dimension is the spatial dimension and npartsX, npartsY and npartsZ are the number of parts created. The product of the number of parts must be equal to the number of processors used for the computation.
- **ghost_thickness** (*type:* int) the number of ghost cells (equivalent to the epaisseur_joint parameter of Decouper.
- **[perio_x]** (*type:* flag) change the splitting method to provide a valid mesh for periodic boundary conditions.
- **[perio_y]** (*type:* flag) change the splitting method to provide a valid mesh for periodic boundary conditions.
- **[perio_z]** (*type:* flag) change the splitting method to provide a valid mesh for periodic boundary conditions.
- **[function_coord_x]** (*type:* string) By default, the meshing algorithm creates nX nY nZ coordinates ranging between 0 and 1 (eg a unity size box). If function_coord_x} is specified, it is used to transform the [0,1] segment to the coordinates of the nodes. funcX must be a function of the x variable only.
- **[function_coord_y]** (*type:* string) like function_coord_x for y
- **[function_coord_z]** (*type:* string) like function_coord_x for z
- **[file_coord_x]** (*type:* string) Keyword to read the Nx floating point values used as nodes coordinates in the file.
- **[file_coord_y]** (*type:* string) idem file_coord_x for y
- **[file_coord_z]** (*type:* string) idem file_coord_x for z
- **[boundary_xmin]** (*type:* string) the name of the boundary at the minimum X direction. If it not provided, the default boundary names are xmin, xmax, ymin, ymax, zmin and zmax. If the mesh is periodic in a given direction, only the MIN boundary name is used, for both sides of the box.
- **[boundary_xmax]** (*type:* string) not_set
- **[boundary_ymin]** (*type:* string) not_set
- **[boundary_ymax]** (*type:* string) not_set
- **[boundary_zmin]** (*type:* string) not_set
- **[boundary_zmax]** (*type:* string) not_set

mass_source

Mass source used in a dilatable simulation to add/reduce a mass at the boundary (volumetric source in the first cell of a given boundary).

Parameters are:

- **bord** (*type:* string) Name of the boundary where the source term is applied
- **surfacic_flux** (*type:* *front_field_base*) The boundary field that the user likes to apply: for example, champ_front_uniforme, ch_front_input_uniform or champ_front_fonc_t

---

merge_med

This keyword allows to merge multiple MED files produced during a parallel computation into a single MED file.

Parameters are:

- **med_files_base_name** (*type:* string) Base name of multiple med files that should appear as base_name_xxxxx.med, where xxxxx denotes the MPI rank number. If you specify NOM_DU_CAS, it will automatically take the basename from your datafile's name.
- **time_iterations** (*type:* string into ['all_times', 'last_time']) Identifies whether to merge all time iterations present in the MED files or only the last one.

---

mkdir

equivalent to system mkdir

Parameters are:

- **directory** (*type:* string) directory to create

---

modif_bord_to_raccord

Keyword to convert a boundary of domain_name domain of kind Bord to a boundary of kind Raccord (named boundary_name). It is useful when using meshes with boundaries of kind Bord defined and to run a coupled calculation.

Parameters are:

- **domaine | domain** (*type:* string) Name of domain
- **nom_bord** (*type:* string) Name of the boundary to transform.

---

modifydomaineaxi1d

Synonyms: convert_1d_to_1daxi

Convert a 1D mesh to 1D axisymmetric mesh

Parameters are:

- **dom** (*type:* string) not_set
- **bloc** (*type: bloc_lecture*) not_set

---

moyenne_volumique

This keyword should be used after Resoudre keyword. It computes the convolution product of one or more fields with a given filtering function.

Parameters are:

- **nom_pb** (*type:* string) name of the problem where the source fields will be searched.
- **nom_domaine** (*type:* string) name of the destination domain (for example, it can be a coarser mesh, but for optimal performance in parallel, the domain should be split with the same algorithm as the computation mesh, eg, same tranche parameters for example)
- **noms_champs** (*type:* list of str) name of the source fields (these fields must be accessible from the postraitement) N source_field1 source_field2 ... source_fieldN
- **[format_post]** (*type:* string) gives the fileformat for the result (by default : lata)
- **[nom_fichier_post]** (*type:* string) indicates the filename where the result is written
- **fonction_filtre** (*type: bloc_lecture*) to specify the given filter Fonction_filtre { type filter_type demie-largeur l [omega w] [expression string] } type filter_type : This parameter specifies the filtering function. Valid filter_type are: Boite is a box filter, $f(x,y,z)=(abs(x)<l)*(abs(y) <l)*(abs(z) <l) / (8 l^3)$ Chapeau is a hat filter (product of hat filters in each direction) centered on the origin, the half-width of the filter being l and its integral being 1. Quadra is a 2nd order filter. Gaussienne is a normalized gaussian filter of standard deviation sigma in each direction (all field elements outside a cubic box defined by clipping_half_width are ignored, hence, taking clipping_half_width=2.5*sigma yields an integral of 0.99 for a uniform unity field). Parser allows a user defined function of the x,y,z variables. All elements outside a cubic box defined by clipping_half_width are ignored. The parser is much slower than the equivalent c++ coded function... demie-largeur l : This parameter specifies the half width of the filter [omega w] : This parameter must be given for the gaussienne filter. It defines the standard deviation of the gaussian filter. [expression string] : This parameter must be given for the parser filter type. This expression will be interpreted by the math parser with the predefined variables x, y and z.
- **[localisation]** (*type:* string into ['elem', 'som']) indicates where the convolution product should be computed: either on the elements or on the nodes of the destination domain.

---

multigrid_solver

Object defining a multigrid solver in IJK discretization

Parameters are:

- **[coarsen_operators]** (*type:* list of Coarsen_operator_uniform) not_set
- **[ghost_size]** (*type:* int) Number of ghost cells known by each processor in each of the three directions
- **[relax_jacobi]** (*type:* list of float) Parameter between 0 and 1 that will be used in the Jacobi method to solve equation on each grid. Should be around 0.7
- **[pre_smooth_steps]** (*type:* list of int) First integer of the list indicates the numbers of integers that has to be read next. Following integers define the numbers of iterations done before solving the equation on each grid. For example, 2 7 8 means that we have a list of 2 integers, the first one tells us to perform 7 pre-smooth steps on the first grid, the second one tells us to perform 8 pre-smooth steps on the second grid. If there are more than 2 grids in the solver, then the remaining ones will have as many pre-smooth steps as the last mentionned number (here, 8)
- **[smooth_steps]** (*type:* list of int) First integer of the list indicates the numbers of integers that has to be read next. Following integers define the numbers of iterations done after solving the equation on each grid. Same behavior as pre_smooth_steps
- **[nb_full_mg_steps]** (*type:* list of int) Number of multigrid iterations at each level
- **[solveur_grossier]** (*type: solveur_sys_base*) Name of the iterative solver that will be used to solve the system on the coarsest grid. This resolution must be more precise than the ones occurring on the fine grids. The threshold of this solver must therefore be lower than seuil defined above.
- **[seuil]** (*type:* float) Define an upper bound on the norm of the final residue (i.e. the one obtained after applying the multigrid solver). With hybrid precision, as long as we have not obtained a residue whose norm is lower than the imposed threshold, we keep applying the solver
- **[impr]** (*type:* flag) Flag to display some info on the resolution on eahc grid
- **[solver_precision]** (*type:* string into ['mixed', 'double']) Precision with which the variables at stake during the resolution of the system will be stored. We can have a simple or double precision or both. In the case of a hybrid precision, the multigrid solver is launched in simple precision, but the residual is calculated in double precision.
- **[iterations_mixed_solver]** (*type:* int) Define the maximum number of iterations in mixed precision solver

---

multiplefiles

Change MPI rank limit for multiple files during I/O

Parameters are:

- **type** (*type:* int) New MPI rank limit

---

my_comm_group

This keyword allows to create a user MPI Comm Group of size N using the processors allocated to TRUST. The set of processors is split in N subsets.

Parameters are:

- **group_nb** (*type:* int) Number of groups to define in your Comm Group.

---

nettoiepasnoeuds

Keyword NettoiePasNoeuds does not delete useless nodes (nodes without elements) from a domain.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

op_conv_ef_stab_polymac_face

Class Op_Conv_EF_Stab_PolyMAC_Face_PolyMAC

Parameters are:

- **[alpha]** (*type:* float) parametre ajustant la stabilisation de 0 (schema centre) a 1 (schema amont)

---

op_conv_ef_stab_polymac_p0_face

Class Op_Conv_EF_Stab_PolyMAC_P0_Face

---

op_conv_ef_stab_polymac_p0p1nc_elem

Synonyms: op_conv_ef_stab_polymac_p0_elem

Class Op_Conv_EF_Stab_PolyMAC_P0P1NC_Elem

Parameters are:

- **[alpha]** (*type:* float) parametre ajustant la stabilisation de 0 (schema centre) a 1 (schema amont)

---

op_conv_ef_stab_polymac_p0p1nc_face

Class Op_Conv_EF_Stab_PolyMAC_P0P1NC_Face

---

option_cgns

Class for CGNS options.

Parameters are:

- **[single_precision]** (*type:* flag) If used, data will be written with a single_precision format inside the CGNS file (it concerns both mesh coordinates and field values).
- **[parallel_over_zone]** (*type:* flag) If used, data will be written in separate zones (ie: one zone per processor). This is not so performant but easier to read later ...
- **[use_links]** (*type:* flag) If used, data will be written in separate files; one file for mesh, and then one file for solution time. Links will be used.
- **[file_per_comm_group]** (*type:* flag) If used, data will be written (at each comm group) in separate files; one file for mesh, and then one file for solution time. Links will be used.
- **[single_safe_file]** (*type:* flag) If used, data will be written in a single file that will be opened and closed at each dt post so that file can be visualized in live. Safer if simulation stops, the file can be used.
- **[close_every_n]** (*type:* int) Used to fix the opening/closing frequency when the SINGLE_SAFE_FILE option is used.
- **[flush_every_n]** (*type:* int) Used to fix the flush-to-disc frequency when the SINGLE_SAFE_FILE option is used.

---

option_dg

Class for DG options.

Parameters are:

- **[order]** (*type:* int) global order for the DG unknowns (1 by default)
- **[velocity_order]** (*type:* int) optional order for DG velocity unknown
- **[pressure_order]** (*type:* int) optional order for DG pressure unknown
- **[temperature_order]** (*type:* int) optional order for DG temperature unknown
- **[gram_schmidt]** (*type:* int) Gram Schmidt orthogonalization (1 by default)

---

option_ijk

Class of IJK options.

Parameters are:

- **[check_divergence]** (*type:* flag) Flag to compute and print the value of div(u) after each pressure-correction
- **[disable_diphasique]** (*type:* flag) Disable all calculations related to interfaces (phase properties, interfacial force, ...)

---

option_interpolation

Class for interpolation fields using MEDCoupling.

Parameters are:

- **[sans_dec | without_dec]** (*type:* flag) Use remapper even for a parallel calculation
- **[sharing_algo]** (*type:* int) Setting the DEC sharing algo : 0,1,2

---

option_polymac

Class of PolyMAC options.

Parameters are:

- **[use_osqp]** (*type:* flag) Flag to use the old formulation of the M2 matrix provided by the OSQP library. Only useful for PolyMAC version.
- **[maillage_vdf | vdf_mesh]** (*type:* flag) Flag used to force the calculation of the equiv tab.
- **[interp_ve1]** (*type:* flag) Flag to enable a first-order face-to-element velocity interpolation. By default, it is not activated which means a second order interpolation. Only useful for PolyMAC_P0 version.
- **[traitement_axi]** (*type:* flag) Flag used to relax the time-step stability criterion in case of a thin slice geometry while modelling an axi-symetrical case. Only useful for PolyMAC_P0 version.

---

option_vdf

Class of VDF options.

Parameters are:

- **[traitement_coins]** (*type:* string into ['oui', 'non']) Treatment of corners (yes or no). This option modifies slightly the calculations at the outlet of the plane channel. It supposes that the boundary continues after channel outlet (i.e. velocity vector remains parallel to the boundary).
- **[traitement_gradients]** (*type:* string into ['oui', 'non']) Treatment of gradient calculations (yes or no). This option modifies slightly the gradient calculation at the corners and activates also the corner treatment option.
- **[p_imposee_aux_faces]** (*type:* string into ['oui', 'non']) Pressure imposed at the faces (yes or no).
- **[deactivate_arete_mixte]** (*type:* flag) Deactivate the arete_mixte contribution in the conv op of the momentum equation.

- **[all_options | toutes_les_options]** (*type:* flag) Activates all Option_VDF options. If used, must be used alone without specifying the other options, nor combinations.

---

orientefacesbord

Keyword to modify the order of the boundary vertices included in a domain, such that the surface normals are outer pointing.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

parallel_io_parameters

Object to handle parallel files in IJK discretization

Parameters are:

- **[block_size_bytes]** (*type:* int) File writes will be performed by chunks of this size (in bytes). This parameter will not be taken into account if block_size_megabytes has been defined
- **[block_size_megabytes]** (*type:* int) File writes will be performed by chunks of this size (in megabytes). The size should be a multiple of the GPFS block size or lustre stripping size (typically several megabytes)
- **[writing_processes]** (*type:* int) This is the number of processes that will write concurrently to the file system (this must be set according to the capacity of the filesystem, set to 1 on small computers, can be up to 64 or 128 on very large systems).
- **[bench_ijk_splitting_write]** (*type:* string) Name of the splitting object we want to use to run a parallel write bench (optional parameter)
- **[bench_ijk_splitting_read]** (*type:* string) Name of the splitting object we want to use to run a parallel read bench (optional parameter)

---

partition

Synonyms: partition_64, decouper

Class for parallel calculation to cut a domain for each processor. By default, this keyword is commented in the reference test cases.

Parameters are:

- **domaine** (*type:* string) Name of the domain to be cut.
- **bloc_decouper** (*type:* *bloc_decouper*) Description how to cut a domain.

---

partition_multi

Synonyms: decouper_multi

allows to partition multiple domains in contact with each other in parallel: necessary for resolution monolithique in implicit schemes and for all coupled problems using PolyMAC_P0P1NC. By default, this keyword is commented in the reference test cases.

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **domaine1** (*type:* string into ['domaine']) not set.
- **dom** (*type:* string) Name of the first domain to be cut.
- **blocdecoupdom1** (*type: bloc_decouper*) Partition bloc for the first domain.
- **domaine2** (*type:* string into ['domaine']) not set.
- **dom2** (*type:* string) Name of the second domain to be cut.
- **blocdecoupdom2** (*type: bloc_decouper*) Partition bloc for the second domain.
- **acof** (*type:* string into ['}']) Closing curly bracket.

---

pilote_icoco

not_set

Parameters are:

- **pb_name** (*type:* string) not_set
- **main** (*type:* string) not_set

---

polyedriser

cast hexahedra into polyhedra so that the indexing of the mesh vertices is compatible with PolyMAC_P0P1NC discretization. Must be used in PolyMAC_P0P1NC discretization if a hexahedral mesh has been produced with TRUST's internal mesh generator.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

postraiter_domaine

To write one or more domains in a file with a specified format (MED,LML,LATA,SINGLE_LATA,CGNS).

Parameters are:

- **format** (*type:* string into ['lml', 'lata', 'single_lata', 'lata_v2', 'med', 'cgns']) File format.
- **[binaire]** (*type:* int into [0, 1]) Binary (binaire 1) or ASCII (binaire 0) may be used. By default, it is 0 for LATA and only ASCII is available for LML and only binary is available for MED.
- **[ecrire_frontiere]** (*type:* int into [0, 1]) This option will write (if set to 1, the default) or not (if set to 0) the boundaries as fields into the file (it is useful to not add the boundaries when writing a domain extracted from another domain)
- **[dual]** (*type:* int into [0, 1]) This option indicates whether the original mesh (default) or the dual one (the one used for postprocessing of field faces) is to be written.
- **[fichier | file]** (*type:* string) The file name can be changed with the fichier option.
- **[joints_non_postraites]** (*type:* int into [0, 1]) The joints_non_postraites (1 by default) will not write the boundaries between the partitioned mesh.
- **[domaine | domain]** (*type:* string) Name of domain
- **[domaines]** (*type: bloc_lecture*) Names of domains : { name1 name2 }

---

precisiongeom

Class to change the way floating-point number comparison is done. By default, two numbers are equal if their absolute difference is smaller than 1e-10. The keyword is useful to modify this value. Moreover, nodes coordinates will be written in .geom files with this same precision.

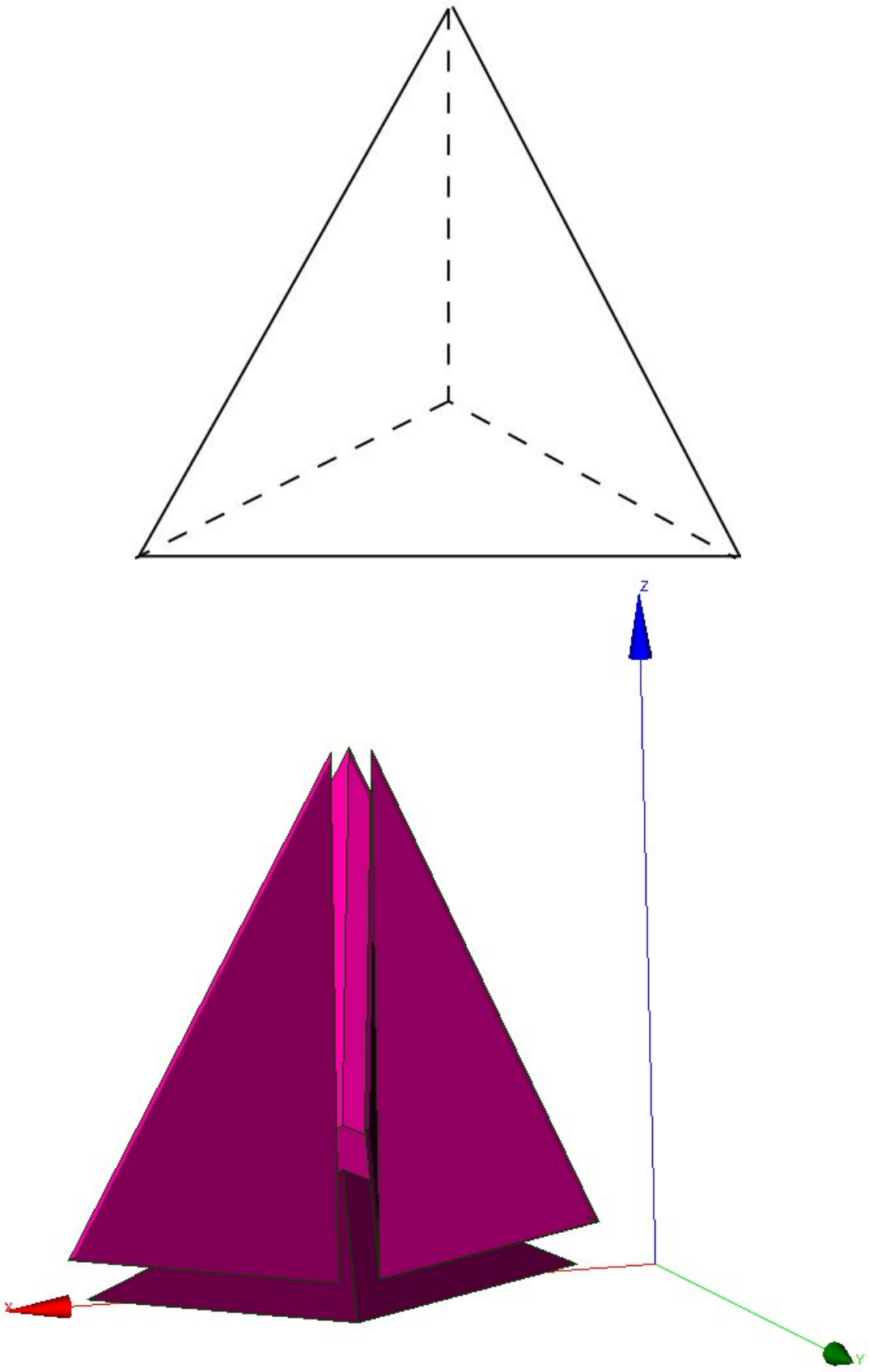
Parameters are:

- **precision** (*type:* float) New value of precision.

---

raffiner_anisotrope

Only for VEF discretizations, allows to cut triangle elements in 3, or tetrahedra in 4 parts, by defining a new summit located at the center of the element:


Z
X
Y

Note that such a cut creates flat elements (anisotropic).

Parameters are:

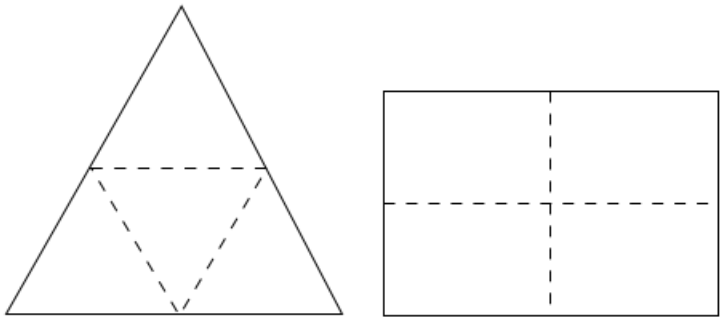
- **domain_name** (*type:* string) Name of domain.

---

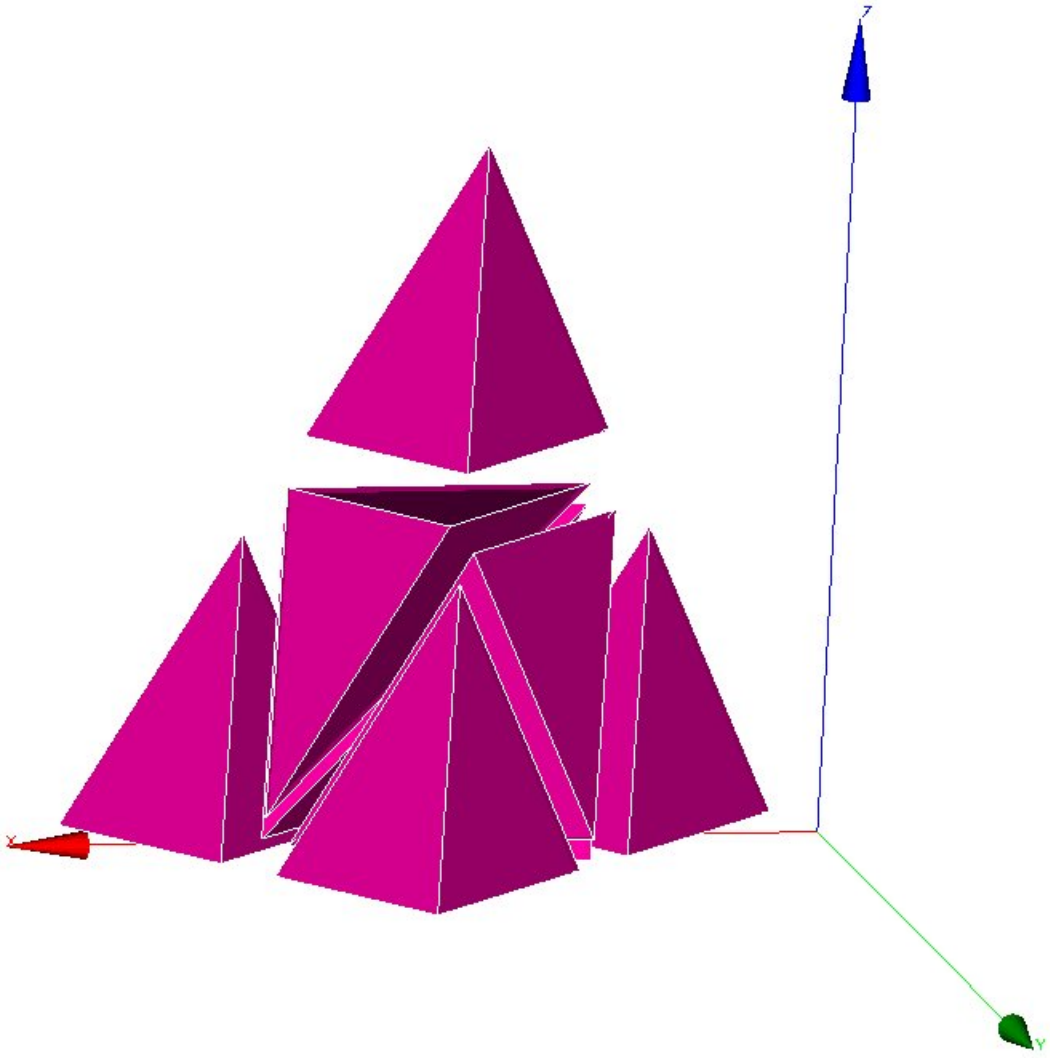
raffiner_isotrope

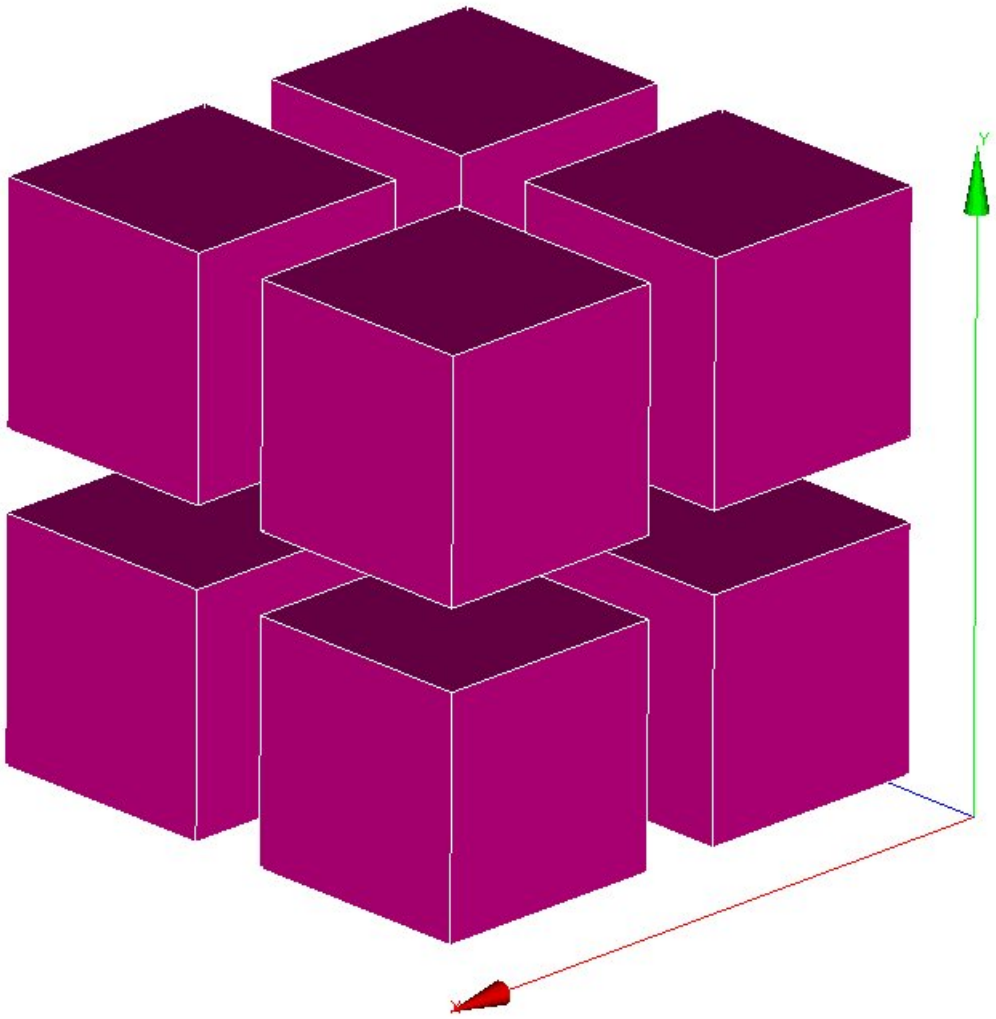
Synonyms: raffiner_simplexes

For VDF and VEF discretizations, allows to cut triangles/quadrangles or tetrahedral/hexaedras elements respectively in 4 or 8 new ones by defining new summits located at the middle of edges (and center of faces and elements for quadrangles and hexaedra). Such a cut preserves the shape of original elements (isotropic). For 2D elements:



For 3D elements:


Z
X
Y



.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

raffiner_isotrope_parallele

Refine parallel mesh in parallel

Parameters are:

- **name_of_initial_domaines | name_of_initial_zones** (*type:* string) name of initial Domaines
- **name_of_new_domaines | name_of_new_zones** (*type:* string) name of new Domaines
- **[ascii]** (*type:* flag) writing Domaines in ascii format
- **[single_hdf]** (*type:* flag) writing Domaines in hdf format

---

read

Synonyms: lire

The 'read' instruction in a TRUST dataset. Overriden from the automatic generation to make the second argument a Objet_u. See also Read_Parser class in base.py module.

Parameters are:

- **identifier** (*type:* string) Identifier of the class being read. Must match a previous forward Declaration.
- **obj** (*type:* objet_u) The object being read.

---

read_file

Synonyms: lire_fichier

Keyword to read the object name_obj contained in the file filename.

This is notably used when the calculation domain has already been meshed and the mesh contains the file filename, simply write read_file dom filename (where dom is the name of the meshed domain).

If the filename is ;, is to execute a data set given in the file of name name_obj (a space must be entered between the semi-colon and the file name).

Parameters are:

- **name_obj** (*type:* string) Name of the object to be read.
- **filename** (*type:* string) Name of the file.

---

read_file_bin

Synonyms: read_file_binary, lire_fichier_bin

Keyword to read an object name_obj in the unformatted type file filename.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be read.
- **filename** (*type:* string) Name of the file.

---

read_med

Synonyms: read_med_64, lire_med

Keyword to read MED mesh files where 'domain' corresponds to the domain name, 'file' corresponds to the file (written in the MED format) containing the mesh named mesh_name.

Note about naming boundaries: When reading 'file', TRUST will detect boundaries between domains (Raccord) when the name of the boundary begins by 'type_raccord_'. For example, a boundary named type_raccord_wall in 'file' will be considered by TRUST as a boundary named 'wall' between two domains.

NB: To read several domains from a mesh issued from a MED file, use Read_Med to read the mesh then use Create_domain_from_sub_domain keyword.

NB: If the MED file contains one or several subdomaine defined as a group of volumes, then Read_MED will read it and will create two files domain_name_ssz.geo and domain_name_ssz_par.geo defining the subdomaines for sequential and/or parallel calculations. These subdomaines will be read in sequential in the datafile by including (after Read_Med keyword) something like:

Read_Med ....

Read_file domain_name_ssz.geo ;

During the parallel calculation, you will include something:

Scatter { ... }

Read_file domain_name_ssz_par.geo ;

Parameters are:

- **[convertalltopoly]** (*type:* flag) Option to convert mesh with mixed cells into polyhedral/polygonal cells
- **domain | domaine** (*type:* string) Corresponds to the domain name.
- **file | fichier** (*type:* string) File (written in the MED format, with extension '.med') containing the mesh
- **[mesh | maillage]** (*type:* string) Name of the mesh in med file. If not specified, the first mesh will be read.
- **[exclude_groups | exclure_groupes]** (*type:* list of str) List of face groups to skip in the MED file.
- **[include_additional_face_groups | inclure_groupes_faces_additionnels]** (*type:* list of str) List of face groups to read and register in the MED file.

---

read_tgrid

Synonyms: lire_tgrid

Keyword to reaf Tgrid/Gambit mesh files. 2D (triangles or quadrangles) and 3D (tetra or hexa elements) meshes, may be read by TRUST.

Parameters are:

- **dom** (*type:* string) Name of domaine.
- **filename** (*type:* string) Name of file containing the mesh.

---

read_unsupported_ascii_file_from_icem

not_set

Parameters are:

- **name_obj** (*type:* string) Name of the object to be read.
- **filename** (*type:* string) Name of the file.

---

rectify_mesh

Synonyms: orienter_simplexes

Keyword to raffine a mesh

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

redresser_hexaedres_vdf

Keyword to convert a domain (named domain_name) with quadrilaterals/VEF hexaedras which looks like rectangles/VDF hexaedras into a domain with real rectangles/VDF hexaedras.

Parameters are:

- **domain_name** (*type:* string) Name of domain to resequence.

---

refine_mesh

not_set

Parameters are:

- **domaine** (*type:* string) not_set

---

regroupebord

Keyword to build one boundary new_bord with several boundaries of the domain named domaine.

Parameters are:

- **domaine | domain** (*type:* string) Name of domain
- **new_bord** (*type:* string) Name of the new boundary
- **bords** (*type: bloc_lecture*) { Bound1 Bound2 }

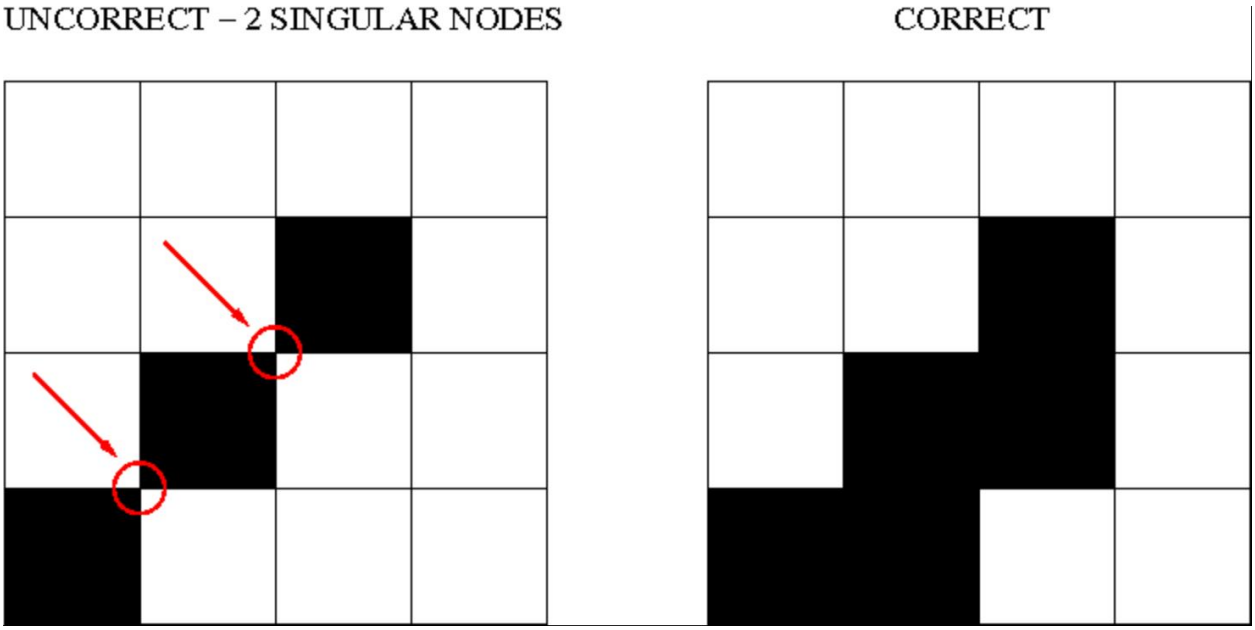
---

remove_elem

Keyword to remove element from a VDF mesh (named domaine_name), either from an explicit list of elements or from a geometric condition defined by a condition f(x,y)>0 in 2D and f(x,y,z)>0 in 3D. All the new borders generated are gathered in one boundary called : newBord (to rename it, use RegroupeBord keyword. To split it to different boundaries, use DecoupeBord_Pour_Rayonnement keyword). Example of a removed zone of radius 0.2 centered at (x,y)=(0.5,0.5):

Remove_elem dom { fonction $0.2*0.2-(x-0.5)^2-(y-0.5)^2>0$ }

Warning : the thickness of removed zone has to be large enough to avoid singular nodes as decribed below :



Parameters are:

- **domaine | domain** (*type:* string) Name of domain
- **bloc** (*type:* *remove_elem_bloc*) not_set

---

remove_invalid_internal_boundaries

Keyword to suppress an internal boundary of the domain_name domain. Indeed, some mesh tools may define internal boundaries (eg: for post processing task after the calculation) but TRUST does not support it yet.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

reorienter_tetraedres

This keyword is mandatory for front-tracking computations with the VEF discretization. For each tetrahedral element of the domain, it checks if it has a positive volume. If the volume (determinant of the three vectors) is negative, it swaps two nodes to reverse the orientation of this tetrahedron.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

reorienter_triangles

not_set

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

resequencing

Synonyms: reordonner

The Reordonner_32_64 interpretor is required sometimes for a VDF mesh which is not produced by the internal mesher. Example where this is used:

Read_file dom fichier.geom

Reordonner_32_64 dom

Observations: This keyword is redundant when the mesh that is read is correctly sequenced in the TRUST sense. This significant mesh operation may take some time... The message returned by TRUST is not explicit when the Reordonner_32_64 (Resequencing) keyword is required but not included in the data set...

Parameters are:

- **domain_name** (*type:* string) Name of domain to resequence.

---

residuals

To specify how the residuals will be computed.

Parameters are:

- **[norm]** (*type:* string into ['l2', 'max']) allows to choose the norm we want to use (max norm by default). Possible to specify L2-norm.
- **[relative]** (*type:* string into ['0', '1', '2']) This is the old keyword seuil_statio_relatif_deconseille. If it is set to 1, it will normalize the residuals with the residuals of the first 5 timesteps (default is 0). if set to 2, residual will be computed as R/(max-min).

---

rotation

Keyword to rotate the geometry of an arbitrary angle around an axis aligned with Ox, Oy or Oz axis.

Parameters are:

- **domain_name** (*type:* string) Name of domain to wich the transformation is applied.
- **dir** (*type:* string into ['x', 'y', 'z']) X, Y or Z to indicate the direction of the rotation axis
- **coord1** (*type:* float) coordinates of the center of rotation in the plane orthogonal to the rotation axis. These coordinates must be specified in the direct triad sense.
- **coord2** (*type:* float) not_set

- **angle** (*type:* float) angle of rotation (in degrees)

---

scatter

Class to read a partionned mesh from the files during a parallel calculation. The files are in binary format.

Parameters are:

- **file** (*type:* string) Name of file.
- **domaine** (*type:* string) Name of domain.

---

scattermed

This keyword will read the partition of the domain_name domain into a the MED format files file.med created by Medsplitter.

Parameters are:

- **file** (*type:* string) Name of file.
- **domaine** (*type:* string) Name of domain.

---

solve

Synonyms: resoudre

Interpretor to start calculation with TRUST.

Parameters are:

- **pb** (*type:* string) Name of problem to be solved.

---

stat_per_proc_perf_log

Keyword allowing to activate the detailed statistics per processor (by default this is false, and only the master proc will produce stats).

Parameters are:

- **flg** (*type:* int) A flag that can be either 0 or 1 to turn off (default) or on the detailed stats.

---

supprime_bord

Keyword to remove boundaries (named Boundary_name1 Boundary_name2) of the domain named domain_name.

Parameters are:

- **domaine | domain** (*type:* string) Name of domain
- **bords** (*type:* list of Nom_anonyme) List of name.

---

system

To run Unix commands from the data file. Example: System 'echo The End | mail trust@cea.fr'

Parameters are:

- **cmd** (*type:* string) command to execute.

---

test_solveur

To test several solvers

Parameters are:

- **[fichier_secmem]** (*type:* string) Filename containing the second member B
- **[fichier_matrice]** (*type:* string) Filename containing the matrix A
- **[fichier_solution]** (*type:* string) Filename containing the solution x
- **[nb_test]** (*type:* int) Number of tests to measure the time resolution (one preconditionnement)
- **[impr]** (*type:* flag) To print the convergence solver
- **[solveur]** (*type: solveur_sys_base*) To specify a solver
- **[fichier_solveur]** (*type:* string) To specify a file containing a list of solvers
- **[genere_fichier_solveur]** (*type:* float) To create a file of the solver with a threshold convergence
- **[seuil_verification]** (*type:* float) Check if the solution satisfy ||Ax-B||<precision
- **[pas_de_solution_initiale]** (*type:* flag) Resolution isn't initialized with the solution x
- **[ascii]** (*type:* flag) Ascii files

---

test_sse_kernels

Object to test the different kernel methods used in the multigrid solver in IJK discretization

Parameters are:

- **[nmax]** (*type:* int) Number of tests we want to perform

---

testeur

not_set

Parameters are:

- **data** (*type: bloc_lecture*) not_set

---

testeur_medcoupling

not_set

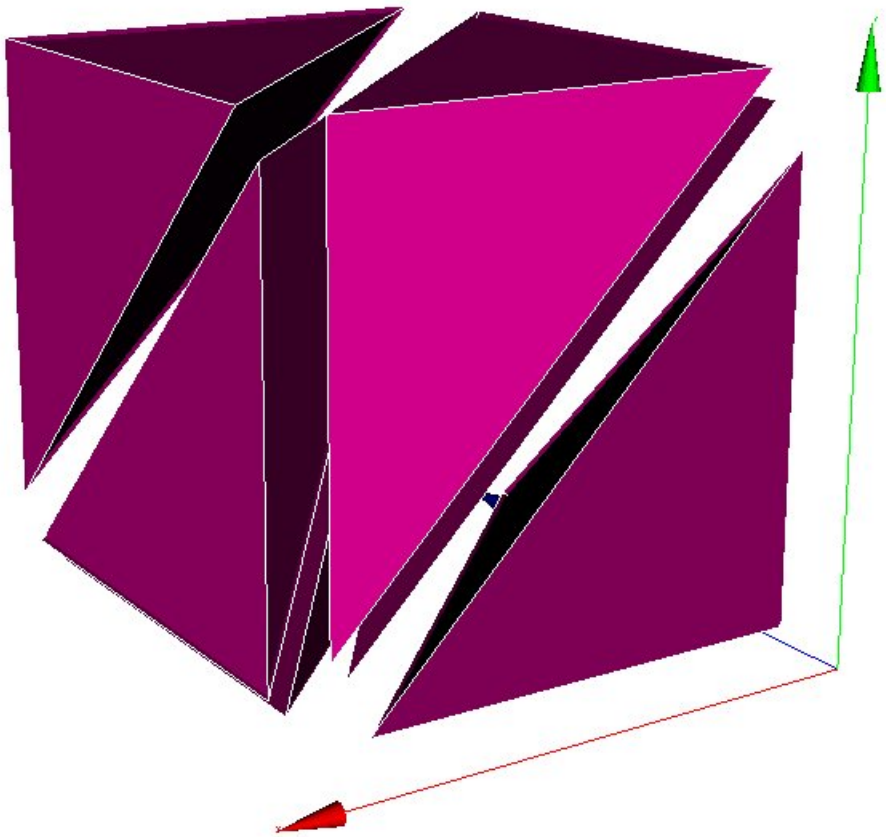
Parameters are:

- **pb_name** (*type:* string) Name of domain.
- **field_name | filed_name** (*type:* string) Name of domain.

---

tetraedriser

To achieve a tetrahedral mesh based on a mesh comprising blocks, the Tetraedriser (Tetrahedralise) interpretor is used in VEF discretization. Initial block is divided in 6 tetrahedra:

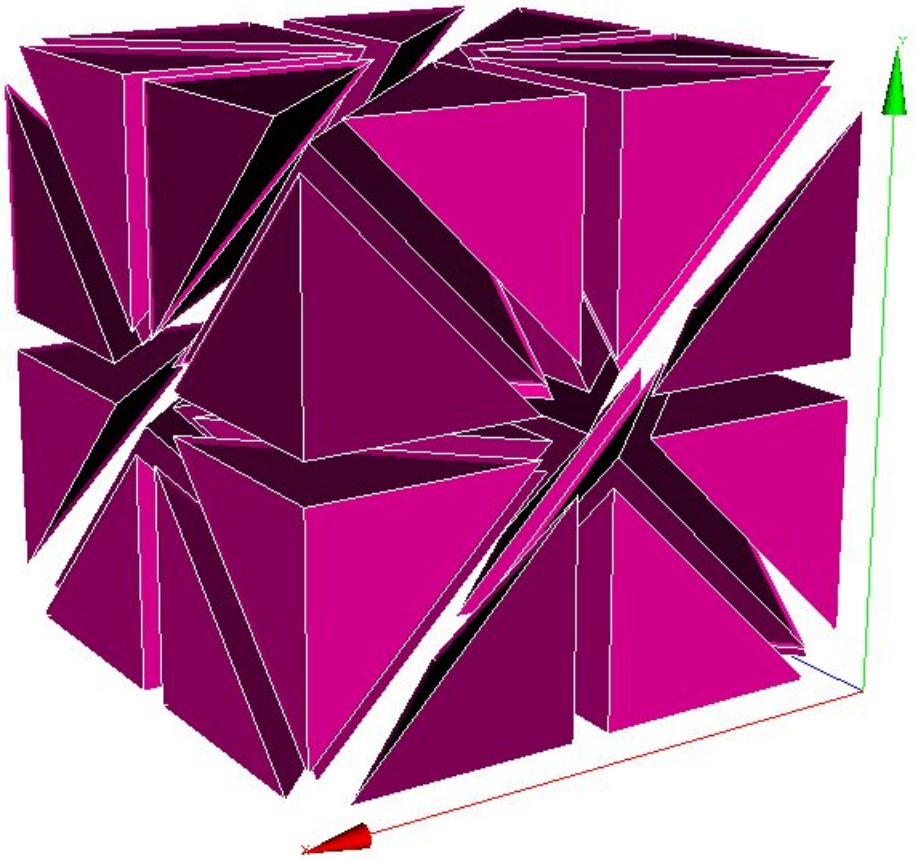


Parameters are:

- **domain_name** (*type:* string) Name of domain.

tetraedriser_homogene

Use the Tetraedriser_homogene (Homogeneous_Tetrahedralisation) interpretor in VEF discretization to mesh a block in tetrahedrals. Each block hexahedral is no longer divided into 6 tetrahedrals (keyword Tetraedriser (Tetrahedralise)), it is now broken down into 40 tetrahedrals. Thus a block defined with 11 nodes in each X, Y, Z direction will contain 10*10*10*40=40,000 tetrahedrals. This also allows problems in the mesh corners with the P1NC/P1iso/P1bulle or P1/P1 discretization items to be avoided. Initial block is divided in 40 tetrahedra:

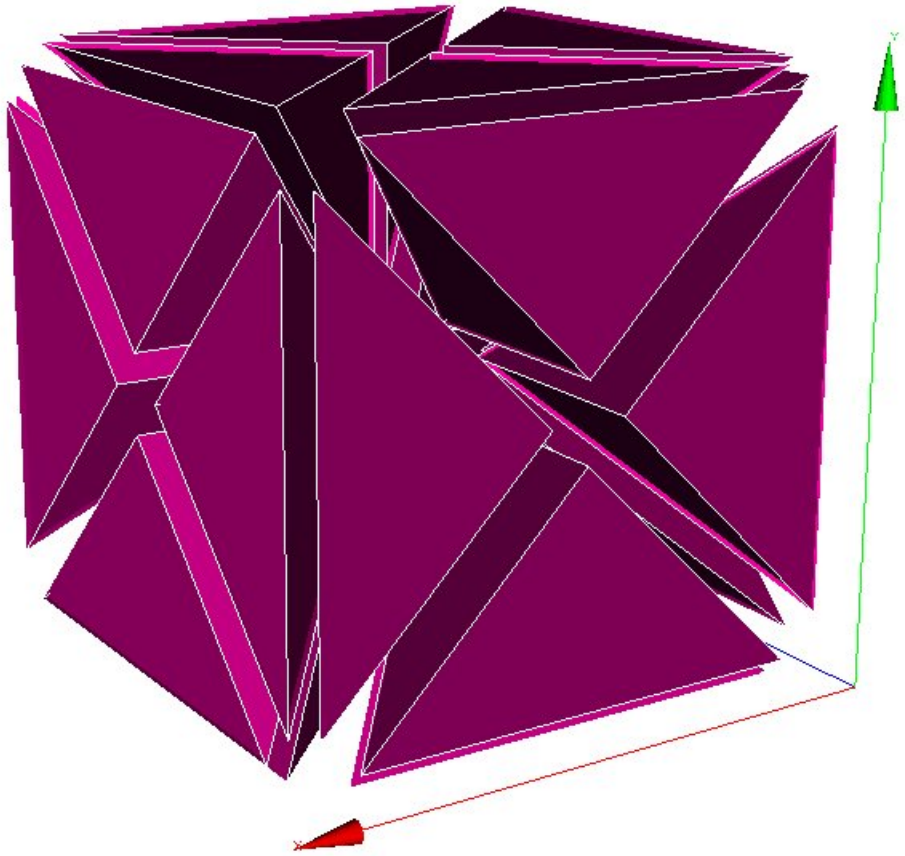


Parameters are:

- **domain_name** (*type:* string) Name of domain.

tetraedriser_homogene_compact

This new discretization generates tetrahedral elements from cartesian or non-cartesian hexahedral elements. The process cut each hexahedral in 6 pyramids, each of them being cut then in 4 tetrahedral. So, in comparison with tetra_homogene, less elements (*24 instead of*40) with more homogeneous volumes are generated. Moreover, this process is done in a faster way. Initial block is divided in 24 tetrahedra:



Parameters are:

- **domain_name** (*type:* string) Name of domain.

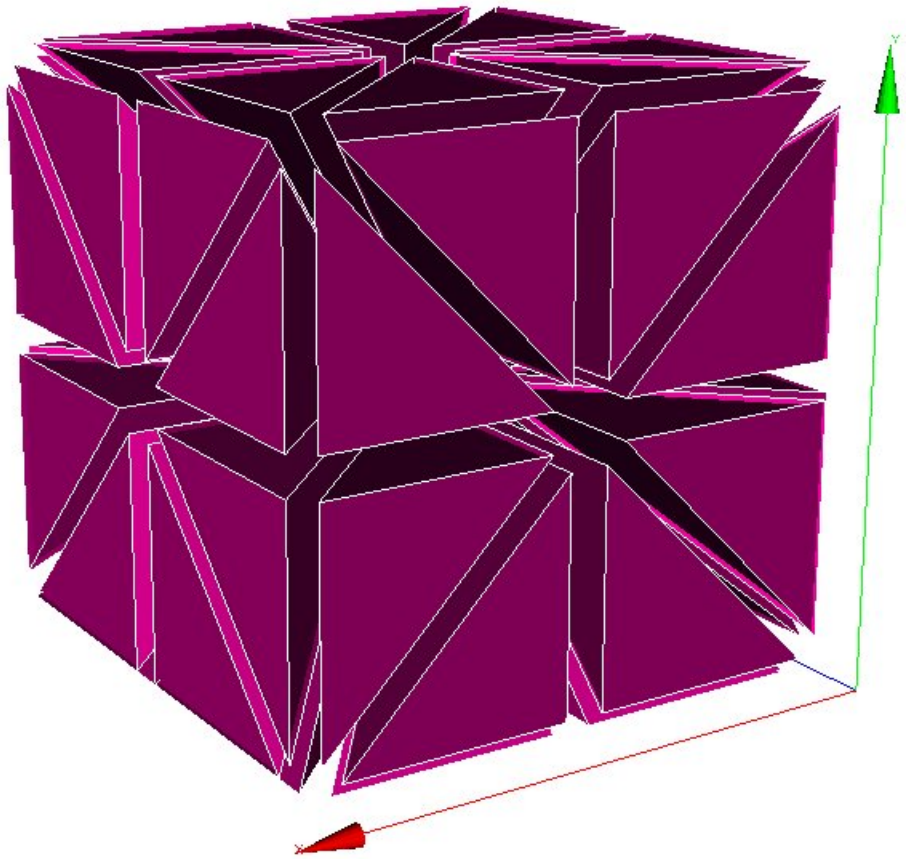
---

tetraedriser_homogene_fin

Tetraedriser_homogene_fin is the recommended option to tetrahedralise blocks. As an extension (subdivision) of Tetraedriser_homogene_compact, this last one cut each initial block in 48 tetrahedra (against 24, previously). This cutting ensures :

- a correct cutting in the corners (in respect to pressure discretization PreP1B),
- a better isotropy of elements than with Tetraedriser_homogene_compact,
- a better alignment of summits (this could have a benefit effect on calculation near

walls since first elements in contact with it are all contained in the same constant thickness and ii/ by the way, a 3D cartesian grid based on summits can be engendered and used to realise spectral analysis in HIT for instance). Initial block is divided in 48 tetrahedra:



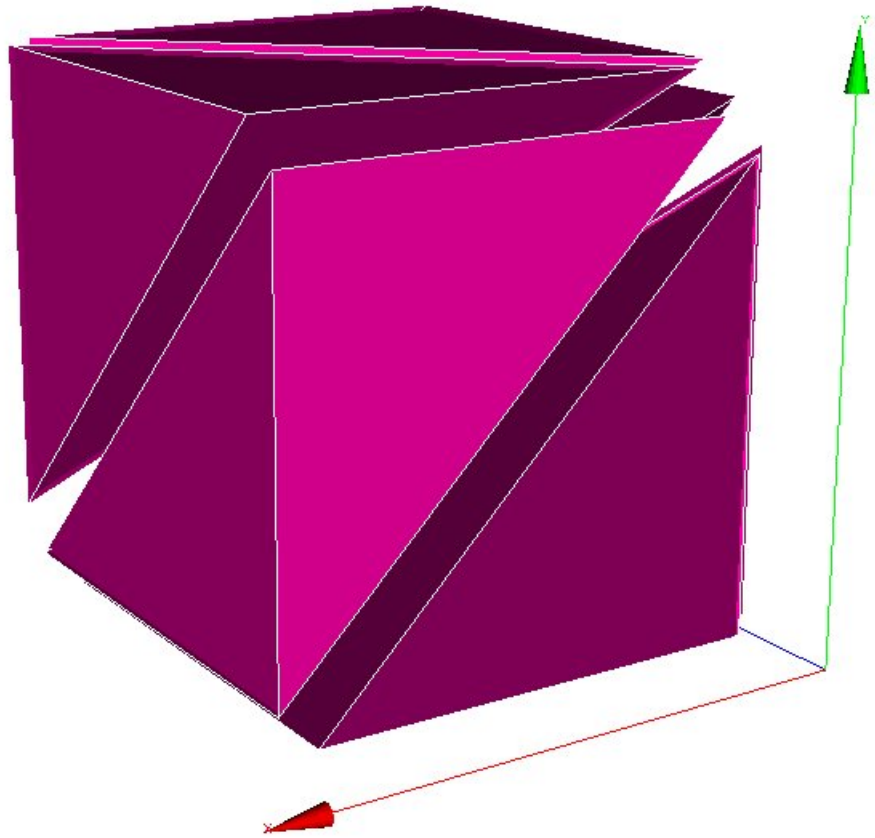
Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

tetraedriser_par_prisme

Tetraedriser_par_prisme generates 6 iso-volume tetrahedral element from primary hexahedral one (contrarily to the 5 elements ordinarily generated by tetraedriser). This element is suitable for calculation of gradients at the summit (coincident with the gravity centre of the jointed elements related with) and spectra (due to a better alignment of the points).



includepng{{tetraedriserparprisme2.jpeg}}{{5}}

Initial block is divided in 6 prismes.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

transformer

Keyword to transform the coordinates of the geometry.

Exemple to rotate your mesh by a 90o rotation and to scale the z coordinates by a factor 2: Transformer domain_name -y -x 2*z

Parameters are:

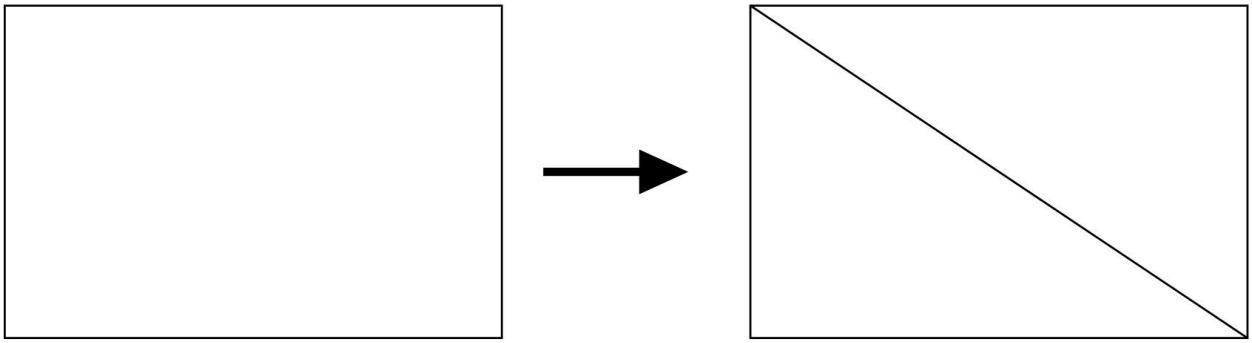
- **domain_name** (*type:* string) Name of domain.
- **formule** (*type:* list of str) Function_for_x Function_for_y [Function_for z]

---

triangulate

Synonyms: trianguler

To achieve a triangular mesh from a mesh comprising rectangles (2 triangles per rectangle). Should be used in VEF discretization. Principle:



initial rectangle Trianguler

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

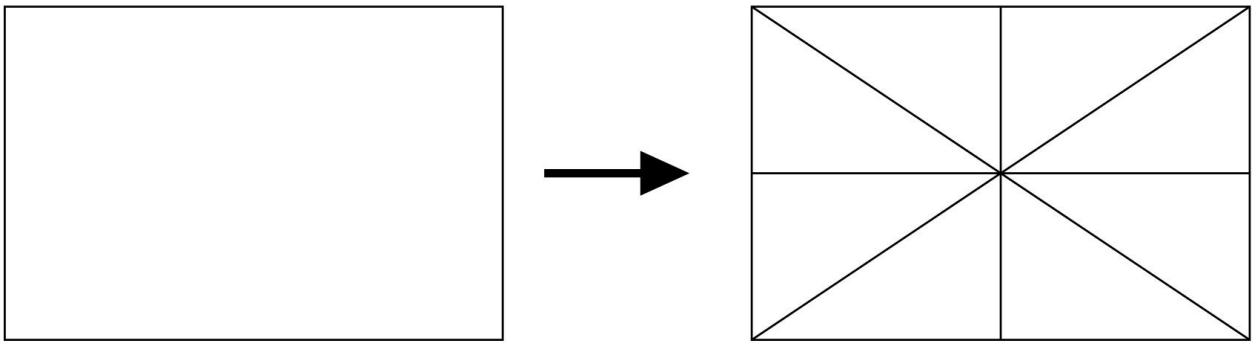
trianguler_fin

Trianguler_fin is the recommended option to triangulate rectangles.

As an extension (subdivision) of Triangulate_h option, this one cut each initial rectangle in 8 triangles (against 4, previously). This cutting ensures :

- a correct cutting in the corners (in respect to pressure discretization PreP1B).
- a better isotropy of elements than with Trianguler_h option.
- a better alignment of summits (this could have a benefit effect on calculation near

walls since first elements in contact with it are all contained in the same constant thickness, and, by this way, a 2D cartesian grid based on summits can be engendered and used to realize statistical analysis in plane channel configuration for instance). Principle:



initial rectangle Trianguler_fin

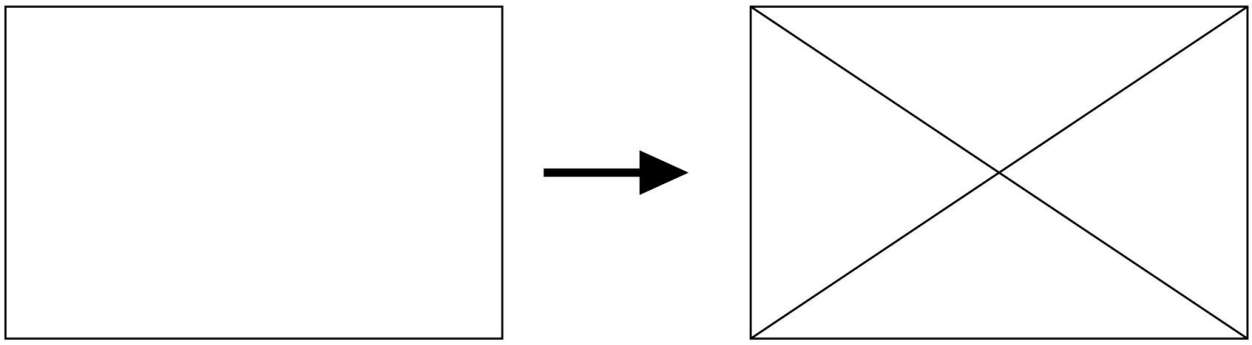
Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

trianguler_h

To achieve a triangular mesh from a mesh comprising rectangles (4 triangles per rectangle). Should be used in VEF discretization. Principle:



initial rectangle Trianguler_h

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

verifier_qualite_raffinements

not_set

Parameters are:

- **domain_names** (*type:* list of Nom_anonyme) Vect of name.

---

verifier_simplexes

Keyword to raffine a simplexes

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

verifiercoin

This keyword subdivides inconsistent 2D/3D cells used with VEFPreP1B discretization. Must be used before the mesh is discretized. The Read_file option can be used only if the file.decoupage_som was previously created by TRUST. This option, only in 2D, reverses the common face at two cells (at least one is inconsistent), through the nodes opposed. In 3D, the option has no effect.

Parameters are:

- **domain_name | dom** (*type:* string) Name of the domaine
- **bloc** (*type:* *verifiercoin_bloc*) not_set

---

write

Synonyms: ecrire

Keyword to write the object of name name_obj to a standard outlet.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be written.

---

write_file

Synonyms: ecrire_fichier, ecrire_fichier_bin

Keyword to write the object of name name_obj to a file filename. Since the v1.6.3, the default format is now binary format file.

Parameters are:

- **name_obj** (*type:* string) Name of the object to be written.
- **filename** (*type:* string) Name of the file.

1.3.15 Keywords derived from loi_etat_base

binaire_gaz_parfait_qc

Class for perfect gas binary mixtures state law used with a quasi-compressible fluid under the iso-thermal and iso-bar assumptions.

Parameters are:

- **molar_mass1** (*type:* float) Molar mass of species 1 (in kg/mol).
- **molar_mass2** (*type:* float) Molar mass of species 2 (in kg/mol).
- **mu1** (*type:* float) Dynamic viscosity of species 1 (in kg/m.s).
- **mu2** (*type:* float) Dynamic viscosity of species 2 (in kg/m.s).

- **temperature** (*type:* float) Temperature (in Kelvin) which will be constant during the simulation since this state law only works for iso-thermal conditions.
- **diffusion_coeff** (*type:* float) Diffusion coefficient assumed the same for both species (in m2/s).

---

binaire_gaz_parfait_wc

Class for perfect gas binary mixtures state law used with a weakly-compressible fluid under the iso-thermal and iso-bar assumptions.

Parameters are:

- **molar_mass1** (*type:* float) Molar mass of species 1 (in kg/mol).
- **molar_mass2** (*type:* float) Molar mass of species 2 (in kg/mol).
- **mu1** (*type:* float) Dynamic viscosity of species 1 (in kg/m.s).
- **mu2** (*type:* float) Dynamic viscosity of species 2 (in kg/m.s).
- **temperature** (*type:* float) Temperature (in Kelvin) which will be constant during the simulation since this state law only works for iso-thermal conditions.
- **diffusion_coeff** (*type:* float) Diffusion coefficient assumed the same for both species (in m2/s).

---

coolprop_qc

Class for using CoolProp with QC problem

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the CoolProp model
- **model** (*type:* string) CoolProp model name

---

coolprop_wc

Class for using CoolProp with WC problem

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the CoolProp model
- **model** (*type:* string) CoolProp model name

---

eos_qc

Class for using EOS with QC problem

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the EOS model
- **model** (*type:* string) EOS model name

---

eos_wc

Class for using EOS with WC problem

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure (J/kg/K).
- **fluid** (*type:* string) Fluid name in the EOS model
- **model** (*type:* string) EOS model name

---

loi_etat_base

Basic class for state laws used with a dilatable fluid.

---

loi_etat_gaz_parfait_base

Basic class for perfect gases state laws used with a dilatable fluid.

---

loi_etat_gaz_reel_base

Basic class for real gases state laws used with a dilatable fluid.

---

loi_etat_tppi_base

Basic class for thermo-physical properties interface (TPPI) used for dilatable problems

---

multi_gaz_parfait_qc

Class for perfect gas multi-species mixtures state law used with a quasi-compressible fluid.

Parameters are:

- **sc** (*type:* float) Schmidt number of the gas Sc=nu/D (D: diffusion coefficient of the mixing).
- **prandtl** (*type:* float) Prandtl number of the gas Pr=mu*Cp/lambda
- **[cp]** (*type:* float) Specific heat at constant pressure of the gas Cp.
- **[dtol_fraction]** (*type:* float) Delta tolerance on mass fractions for check testing (default value 1.e-6).
- **[correction_fraction]** (*type:* flag) To force mass fractions between 0. and 1.
- **[ignore_check_fraction]** (*type:* flag) Not to check if mass fractions between 0. and 1.

---

multi_gaz_parfait_wc

Class for perfect gas multi-species mixtures state law used with a weakly-compressible fluid.

Parameters are:

- **species_number** (*type:* int) Number of species you are considering in your problem.
- **diffusion_coeff** (*type: field_base*) Diffusion coefficient of each species, defined with a Champ_uniforme of dimension equals to the species_number.
- **molar_mass** (*type: field_base*) Molar mass of each species, defined with a Champ_uniforme of dimension equals to the species_number.
- **mu** (*type: field_base*) Dynamic viscosity of each species, defined with a Champ_uniforme of dimension equals to the species_number.
- **cp** (*type: field_base*) Specific heat at constant pressure of the gas Cp, defined with a Champ_uniforme of dimension equals to the species_number..
- **prandtl** (*type:* float) Prandtl number of the gas Pr=mu*Cp/lambda.

---

perfect_gaz_qc

Synonyms: gaz_parfait_qc

Class for perfect gas state law used with a quasi-compressible fluid.

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure (J/kg/K).
- **[cv]** (*type:* float) Specific heat at constant volume (J/kg/K).
- **[gamma]** (*type:* float) Cp/Cv
- **prandtl** (*type:* float) Prandtl number of the gas Pr=mu*Cp/lambda
- **[rho_constant_pour_debug]** (*type: field_base*) For developers to debug the code with a constant rho.

---

perfect_gaz_wc

Synonyms: gaz_parfait_wc

Class for perfect gas state law used with a weakly-compressible fluid.

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure (J/kg/K).
- **[cv]** (*type:* float) Specific heat at constant volume (J/kg/K).
- **[gamma]** (*type:* float) Cp/Cv
- **prandtl** (*type:* float) Prandtl number of the gas Pr=mu*Cp/lambda

---

rhot_gaz_parfait_qc

Class for perfect gas used with a quasi-compressible fluid where the state equation is defined as rho = f(T).

Parameters are:

- **cp** (*type:* float) Specific heat at constant pressure of the gas Cp.
- **[prandtl]** (*type:* float) Prandtl number of the gas Pr=mu*Cp/lambda
- **[rho_xyz]** (*type: field_base*) Defined with a Champ_Fonc_xyz to define a constant rho with time (space dependent)
- **[rho_t]** (*type:* string) Expression of T used to calculate rho. This can lead to a variable rho, both in space and in time.
- **[t_min]** (*type:* float) Temperature may, in some cases, locally and temporarily be very small (and negative) even though computation converges. T_min keyword allows to set a lower limit of temperature (in Kelvin, -1000 by default). WARNING: DO NOT USE THIS KEYWORD WITHOUT CHECKING CAREFULY YOUR RESULTS!

---

rhot_gaz_reel_qc

Class for real gas state law used with a quasi-compressible fluid.

Parameters are:

- **bloc** (*type: bloc_lecture*) Description.

1.3.16 Keywords derived from loi_fermeture_base

loi_fermeture_base

Class for appends fermeture to problem

---

loi_fermeture_test

Loi for test only

Parameters are:

- **[coef]** (*type:* float) coefficient

1.3.17 Keywords derived from loi_horaire

loi_horaire

to define the movement with a time-dependant law for the solid interface.

Parameters are:

- **position** (*type:* list of str) Vecteur position
- **vitesse** (*type:* list of str) Vecteur vitesse
- **[rotation]** (*type:* list of str) Matrice de passage
- **[derivee_rotation]** (*type:* list of str) Derivee matrice de passage
- **[verification_derivee]** (*type:* int) not_set
- **[impr]** (*type:* int) Whether to print output

1.3.18 Keywords derived from milieu_base

constituant

Constituent.

Parameters are:

- **[coefficient_diffusion]** (*type: field_base*) Constituent diffusion coefficient value (m2.s-1). If a multi-constituent problem is being processed, the diffusivite will be a vectorial and each components will be the diffusion of the constituent.
- **[is_multi_scalar | is_multi_scalar_diffusion]** (*type:* flag) Flag to activate the multi_scalar diffusion operator
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_base

Basic class for fluids.

Parameters are:

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_dilatable_base

Basic class for dilatable fluids.

Parameters are:

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_incompressible

Class for non-compressible fluids.

Parameters are:

- **[beta_th]** (*type: field_base*) Thermal expansion (K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[beta_co]** (*type: field_base*) Volume expansion coefficient values in concentration.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[porosites]** (*type: bloc_lecture*) Porosity (optional)
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).

---

fluide_ostwald

Non-Newtonian fluids governed by Ostwald's law. The law applicable to stress tensor is:

tau=K(T)*(D:D/2)**((n-1)/2)*D Where:

D refers to the deformation tensor

K refers to fluid consistency (may be a function of the temperature T)

n refers to the fluid structure index n=1 for a Newtonian fluid, n<1 for a rheofluidifier fluid, n>1 for a rheothickening fluid.

Parameters are:

- **[k]** (*type: field_base*) Fluid consistency.
- **[n]** (*type: field_base*) Fluid structure index.
- **[beta_th]** (*type: field_base*) Thermal expansion (K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[beta_co]** (*type: field_base*) Volume expansion coefficient values in concentration.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[porosites]** (*type: bloc_lecture*) Porosity (optional)

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).

---

fluide_quasi_compressible

Quasi-compressible flow with a low mach number assumption; this means that the thermo- dynamic pressure (used in state law) is uniform in space.

Parameters are:

- **[sutherland]** (*type: bloc_sutherland*) Sutherland law for viscosity and for conductivity.
- **[pression]** (*type:* float) Initial thermo-dynamic pressure used in the assosciated state law.
- **[loi_etat]** (*type: loi_etat_base*) The state law that will be associated to the Quasi-compressible fluid.
- **[traitement_pth]** (*type:* string into ['edo', 'constant', 'conservation_masse']) Particular treatment for the thermodynamic pressure Pth ; there are three possibilities: 1) with the keyword 'edo' the code computes Pth solving an O.D.E. ; in this case, the mass is not strictly conserved (it is the default case for quasi compressible computation): 2) the keyword 'conservation_masse' forces the conservation of the mass (closed geometry or with periodic boundaries condition) 3) the keyword 'constant' makes it possible to have a constant Pth ; it's the good choice when the flow is open (e.g. with pressure boundary conditions). It is possible to monitor the volume averaged value for temperature and density, plus Pth evolution in the .evol_glob file.
- **[traitement_rho_gravite]** (*type:* string into ['standard', 'moins_rho_moyen']) It may be :1) `standard` : the gravity term is evaluted with rho*g (It is the default). 2) `moins_rho_moyen` : the gravity term is evaluated with (rho-rhomoy) *g. Unknown pressure is then P*=P+rhomoy*g*z. It is useful when you apply uniforme pressure boundary condition like P*=0.
- **[temps_debut_prise_en_compte_drho_dt]** (*type:* float) While time<value, dRho/dt is set to zero (Rho, volumic mass). Useful for some calculation during the first time steps with big variation of temperature and volumic mass.
- **[omega_relaxation_drho_dt]** (*type:* float) Optional option to have a relaxed algorithm to solve the mass equation. value is used (1 per default) to specify omega.
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_reel_base

Class for real fluids.

Parameters are:

- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_sodium_gaz

Class for Fluide_sodium_liquide

Parameters are:

- **[p_ref]** (*type:* float) Use to set the pressure value in the closure law. If not specified, the value of the pressure unknown will be used
- **[t_ref]** (*type:* float) Use to set the temperature value in the closure law. If not specified, the value of the temperature unknown will be used
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).

- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_sodium_liquide

Class for Fluide_sodium_liquide

Parameters are:

- **[p_ref]** (*type:* float) Use to set the pressure value in the closure law. If not specified, the value of the pressure unknown will be used
- **[t_ref]** (*type:* float) Use to set the temperature value in the closure law. If not specified, the value of the temperature unknown will be used
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_stiffened_gas

Class for Stiffened Gas

Parameters are:

- **[gamma]** (*type:* float) Heat capacity ratio (Cp/Cv)
- **[pinf]** (*type:* float) Stiffened gas pressure constant (if set to zero, the state law becomes identical to that of perfect gases)
- **[mu]** (*type:* float) Dynamic viscosity
- **[lambda_ | lambda_u | lambda]** (*type:* float) Thermal conductivity
- **[cv]** (*type:* float) Thermal capacity at constant volume
- **[q]** (*type:* float) Reference energy
- **[q_prim]** (*type:* float) Model constant
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).

- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

fluide_weakly_compressible

Weakly-compressible flow with a low mach number assumption; this means that the thermo- dynamic pressure (used in state law) can vary in space.

Parameters are:

- **[loi_etat]** (*type: loi_etat_base*) The state law that will be associated to the Weakly-compressible fluid.
- **[sutherland]** (*type: bloc_sutherland*) Sutherland law for viscosity and for conductivity.
- **[traitement_pth]** (*type:* string into ['constant']) Particular treatment for the thermodynamic pressure Pth ; there is currently one possibility: 1) the keyword 'constant' makes it possible to have a constant Pth but not uniform in space ; it's the good choice when the flow is open (e.g. with pressure boundary conditions).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[mu]** (*type: field_base*) Dynamic viscosity (kg.m-1.s-1).
- **[pression_thermo]** (*type:* float) Initial thermo-dynamic pressure used in the assosciated state law.
- **[pression_xyz]** (*type: field_base*) Initial thermo-dynamic pressure used in the assosciated state law. It should be defined with as a Champ_Fonc_xyz.
- **[use_total_pressure]** (*type:* int) Flag (0 or 1) used to activate and use the total pressure in the assosciated state law. The default value of this Flag is 0.
- **[use_hydrostatic_pressure]** (*type:* int) Flag (0 or 1) used to activate and use the hydro-static pressure in the assosciated state law. The default value of this Flag is 0.
- **[use_grad_pression_eos]** (*type:* int) Flag (0 or 1) used to specify whether or not the gradient of the thermo-dynamic pressure will be taken into account in the source term of the temperature equation (case of a non-uniform pressure). The default value of this Flag is 1 which means that the gradient is used in the source.
- **[time_activate_ptot]** (*type:* float) Time (in seconds) at which the total pressure will be used in the assosciated state law.
- **[indice]** (*type: field_base*) Refractivity of fluid.
- **[kappa]** (*type: field_base*) Absorptivity of fluid (m-1).
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

milieu_base

Basic class for medium (physics properties of medium).

Parameters are:

- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.
- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).

---

solide

Solid with cp and/or rho non-uniform.

Parameters are:

- **[rho]** (*type: field_base*) Density (kg.m-3).
- **[cp]** (*type: field_base*) Specific heat (J.kg-1.K-1).
- **[lambda_ | lambda_u | lambda]** (*type: field_base*) Conductivity (W.m-1.K-1).
- **[user_field]** (*type: field_base*) user defined field.
- **[gravite]** (*type: field_base*) Gravity field (optional).
- **[porosites_champ]** (*type: field_base*) The porosity is given at each element and the porosity at each face, Psi(face), is calculated by the average of the porosities of the two neighbour elements Psi(elem1), Psi(elem2) : Psi(face)=2/(1/Psi(elem1)+1/Psi(elem2)). This keyword is optional.
- **[diametre_hyd_champ]** (*type: field_base*) Hydraulic diameter field (optional).
- **[porosites]** (*type: porosites*) Porosities.

1.3.19 Keywords derived from modele_turbulence_scal_base

modele_turbulence_scal_base

Basic class for turbulence model for energy equation.

Parameters are:

- **[dt_impr_nusselt]** (*type:* float) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[dt_impr_nusselt_mean_only]** (*type:* *dt_impr_nusselt_mean_only*) This keyword is used to print the mean values of Nusselt (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_nusselt_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values, then you have to specify their names.
- **[turbulence_paroi]** (*type:* *turbulence_paroi_scalaire_base*) Keyword to set the wall law.

---

modele_turbulence_scal_null

Synonyms: null

Null scalar turbulence model (turbulent diffusivity = 0) which can be used with a turbulent problem.

Parameters are:

- **[dt_impr_nusselt]** (*type:* float) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[dt_impr_nusselt_mean_only]** (*type:* *dt_impr_nusselt_mean_only*) This keyword is used to print the mean values of Nusselt (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_nusselt_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values, then you have to specify their names.

---

prandtl

The Prandtl model. For the scalar equations, only the model based on Reynolds analogy is available. If K_Epsilon was selected in the hydraulic equation, Prandtl must be selected for the convection-diffusion temperature equation coupled to the hydraulic equation and Schmidt for the concentration equations.

Parameters are:

- **[prdt]** (*type:* string) Keyword to modify the constant (Prdt) of Prandtl model : Alphat=Nut/Prdt Default value is 0.9
- **[prandt_turbulent_fonction_nu_t_alpha]** (*type:* string) Optional keyword to specify turbulent diffusivity (by default, alpha_t=nu_t/Prt) with another formulae, for example: alpha_t=nu_t2/(0,7*alpha+0,85*nu_tt) with the string nu_t*nu_t/(0,7*alpha+0,85*nu_t) where alpha is the thermal diffusivity.
- **[dt_impr_nusselt]** (*type:* float) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[dt_impr_nusselt_mean_only]** (*type:* *dt_impr_nusselt_mean_only*) This keyword is used to print the mean values of Nusselt (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_nusselt_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values, then you have to specify their names.
- **[turbulence_paroi]** (*type:* *turbulence_paroi_scalaire_base*) Keyword to set the wall law.

---

schmidt

The Schmidt model. For the scalar equations, only the model based on Reynolds analogy is available. If K_Epsilon was selected in the hydraulic equation, Schmidt must be selected for the convection-diffusion temperature equation coupled to the hydraulic equation and Schmidt for the concentration equations.

Parameters are:

- **[scturb]** (*type:* float) Keyword to modify the constant (Sct) of Schmlidt model : Dt=Nut/Sct Default value is 0.7.
- **[dt_impr_nusselt]** (*type:* float) Keyword to print local values of Nusselt number and temperature near a wall during a turbulent calculation. The values will be printed in the _Nusselt.face file each dt_impr_nusselt time period. The local Nusselt expression is as follows : Nu = ((lambda+lambda_t)/lambda)*d_wall/d_eq where d_wall is the distance from the first mesh to the wall and d_eq is given by the wall law. This option also gives the value of d_eq and h = (lambda+lambda_t)/d_eq and the fluid temperature of the first mesh near the wall. For the Neumann boundary conditions (flux_impose), the <<equivalent>> wall temperature given by the wall law is also printed (Tparoi equiv.) preceded for VEF calculation by the edge temperature <<T face de bord>>.
- **[dt_impr_nusselt_mean_only]** (*type:* *dt_impr_nusselt_mean_only*) This keyword is used to print the mean values of Nusselt (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_nusselt_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values, then you have to specify their names.

- **[turbulence_paroi]** (*type: turbulence_paroi_scalaire_base*) Keyword to set the wall law.

1.3.20 Keywords derived from mor_eqn

conduction

Heat equation.

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

conduction_ibm

IBM Heat equation.

Parameters are:

- **[correction_variable_initiale]** (*type:* int) Modify initial variable
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file

- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_chaleur_qc

Temperature equation for a quasi-compressible fluid.

Parameters are:

- **[mode_calcul_convection]** (*type:* string into ['ancien', 'divut_moins_tdivu', 'divrhout_moins_tdivrhou']) Option to set the form of the convective operator divrhouT_moins_Tdivrhou (the default since 1.6.8): rho.u.gradT = div(rho.u.T)- Tdiv(rho.u.1) ancien: u.gradT = div(u.T) - T.div(u) divuT_moins_Tdivu : u.gradT = div(u.T) - Tdiv(u.1)
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_chaleur_turbulent_qc

Temperature equation for a quasi-compressible fluid as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model for the temperature (energy) conservation equation.
- **[mode_calcul_convection]** (*type:* string into ['ancien', 'divut_moins_tdivu', 'divrhout_moins_tdivrhou']) Option to set the form of the convective operator divrhouT_moins_Tdivrhou (the default since 1.6.8): rho.u.gradT = div(rho.u.T)- Tdiv(rho.u.1) ancien: u.gradT = div(u.T) - T.div(u) divuT_moins_Tdivu : u.gradT = div(u.T) - Tdiv(u.1)

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_chaleur_wc

Temperature equation for a weakly-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_concentration

Constituent transport vectorial equation (concentration diffusion convection).

Parameters are:

- **[nom_inconnue]** (*type:* string) Keyword Nom_inconnue will rename the unknown of this equation with the given name. In the postprocessing part, the concentration field will be accessible with this name. This is usefull if you want to track more than one concentration (otherwise, only the concentration field in the first concentration equation can be accessed).
- **[alias]** (*type:* string) not_set
- **[masse_molaire]** (*type:* float) not_set
- **[is_multi_scalar | is_multi_scalar_diffusion]** (*type:* flag) Flag to activate the multi_scalar diffusion operator
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_concentration_turbulent

Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model to be used in the constituent transport equations. The only model currently available is Schmidt.
- **[nom_inconnue]** (*type:* string) Keyword Nom_inconnue will rename the unknown of this equation with the given name. In the postprocessing part, the concentration field will be accessible with this name. This is usefull if you want to track more than one concentration (otherwise, only the concentration field in the first concentration equation can be accessed).
- **[alias]** (*type:* string) not_set
- **[masse_molaire]** (*type:* float) not_set
- **[is_multi_scalar | is_multi_scalar_diffusion]** (*type:* flag) Flag to activate the multi_scalar diffusion operator

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_espece_binaire_qc

Species conservation equation for a binary quasi-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_espece_binaire_turbulent_qc

Species conservation equation for a binary quasi-compressible fluid as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type:* *modele_turbulence_scal_base*) Turbulence model for the species conservation equation.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type:* *bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type:* *bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type:* *ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type:* *parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_espece_binaire_wc

Species conservation equation for a binary weakly-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type:* *bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type:* *bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type:* *ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type:* *parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_espece_multi_qc

Species conservation equation for a multi-species quasi-compressible fluid.

Parameters are:

- **[espece]** (*type: espece*) Associate a species (with its properties) to the equation
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_espece_multi_turbulent_qc

not_set

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model to be used.
- **espece** (*type: espece*) not_set
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file

- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_espece_multi_wc

Species conservation equation for a multi-species weakly-compressible fluid.

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_temperature

Energy equation (temperature diffusion convection).

Parameters are:

- **[penalisation_l2_ftd]** (*type: bloc_lecture*) to activate or not (the default is Direct Forcing method) the Penalized Direct Forcing method to impose the specified temperature on the solid-fluid interface.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.

- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_temperature_ibm

IBM Energy equation (temperature diffusion convection).

Parameters are:

- **[correction_variable_initiale]** (*type:* int) Modify initial variable
- **[penalisation_l2_ftd]** (*type: bloc_lecture*) to activate or not (the default is Direct Forcing method) the Penalized Direct Forcing method to impose the specified temperature on the solid-fluid interface.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_temperature_ibm_turbulent

IBM Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model for the energy equation.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

convection_diffusion_temperature_turbulent

Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_scal_base*) Turbulence model for the energy equation.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }

- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

echelle_temporelle_turbulente

Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

energie_cinetique_turbulente

Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation

- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

energie_cinetique_turbulente_wit

Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

energie_multiphase

Internal energy conservation equation for a multi-phase problem where the unknown is the temperature

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file

- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

energie_multiphase_enthalpie

Synonyms: energie_multiphase_h

Internal energy conservation equation for a multi-phase problem where the unknown is the enthalpy

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

eqn_base

Basic class for equations.

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.

- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

masse_multiphase

Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

mor_eqn

Class of equation pieces (morceaux d'equation).

---

navier_stokes_ibm

IBM Navier-Stokes equations.

Parameters are:

- **[correction_matrice_projection_initiale]** (*type:* int) (IBM advanced) fix matrix of initial projection for PDF
- **[correction_calcul_pression_initiale]** (*type:* int) (IBM advanced) fix initial pressure computation for PDF
- **[correction_vitesse_projection_initiale]** (*type:* int) (IBM advanced) fix initial velocity computation for PDF
- **[correction_matrice_pression]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[matrice_pression_penalisee_h1]** (*type:* int) (IBM advanced) fix pressure matrix for PDF
- **[correction_vitesse_modifie]** (*type:* int) (IBM advanced) fix velocity for PDF
- **[correction_pression_modifie]** (*type:* int) (IBM advanced) fix pressure for PDF
- **[gradient_pression_qdm_modifie]** (*type:* int) (IBM advanced) fix pressure gradient
- **[correction_variable_initiale]** (*type:* int) Modify initial variable
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing
- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.

- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

navier_stokes_ibm_turbulent

IBM Navier-Stokes equations as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_hyd_deriv*) Turbulence model for Navier-Stokes equations.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing
- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved

as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

navier_stokes_qc

Navier-Stokes equation for a quasi-compressible fluid.

Parameters are:

- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing

- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

navier_stokes_standard

Navier-Stokes equations.

Parameters are:

- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).

- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing
- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

navier_stokes_turbulent

Navier-Stokes equations as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_hyd_deriv*) Turbulence model for Navier-Stokes equations.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor

of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.

- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing
- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

navier_stokes_turbulent_qc

Navier-Stokes equations under low Mach number as well as the associated turbulence model equations.

Parameters are:

- **[modele_turbulence]** (*type: modele_turbulence_hyd_deriv*) Turbulence model for Navier-Stokes equations.
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.

- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing
- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

navier_stokes_wc

Navier-Stokes equation for a weakly-compressible fluid.

Parameters are:

- **[mass_source]** (*type: mass_source*) Mass source used in a dilatable simulation to add/reduce a mass at the boundary (volumetric source in the first cell of a given boundary).
- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[dt_projection]** (*type: deuxmots*) nb value : This keyword checks every nb time-steps the equality of velocity divergence to zero. value is the criteria convergency for the solver used.
- **[traitement_particulier]** (*type: traitement_particulier*) Keyword to post-process particular values.
- **[seuil_divu]** (*type: floatfloat*) value factor : this keyword is intended to minimise the number of iterations during the pressure system resolution. The convergence criteria during this step ('seuil' in solveur_pression) is dynamically adapted according to the mass conservation. At tn , the linear system Ax=B is considered as solved if the residual ||Ax-B||<seuil(tn). For tn+1, the threshold value seuil(tn+1) will be evualated as: If (|max(DivU)*dt|<value) Seuil(tn+1)= Seuil(tn)*factor Else Seuil(tn+1)= Seuil(tn)*factor Endif The first parameter (value) is the mass evolution the user is ready to accept per timestep, and the second one (factor) is the factor of evolution for 'seuil' (for example 1.1, so 10% per timestep). Investigations has to be lead to know more about the effects of these two last parameters on the behaviour of the simulations.
- **[solveur_bar]** (*type: solveur_sys_base*) This keyword is used to define when filtering operation is called (typically for EF convective scheme, standard diffusion operator and Source_Qdm_lambdaup). A file (solveur.bar) is then created and used for inversion procedure. Syntax is the same then for pressure solver (GCP is required for multi-processor calculations and, in a general way, for big meshes).
- **[projection_initiale]** (*type:* int) Keyword to suppress, if boolean equals 0, the initial projection which checks DivU=0. By default, boolean equals 1.
- **[postraiter_gradient_pression_sans_masse]** (*type:* flag) Avoid mass matrix multiplication for the gradient postprocessing
- **[methode_calcul_pression_initiale]** (*type:* string into ['avec_les_cl', 'avec_sources', 'avec_sources_et_operateurs', 'sans_rien']) Keyword to select an option for the pressure calculation before the fist time step. Options are : avec_les_cl (default option lapP=0 is solved with Neuman boundary conditions on pressure if any), avec_sources (lapP=f is solved with Neuman boundaries conditions and f integrating the source terms of the Navier-Stokes equations) and avec_sources_et_operateurs (lapP=f is solved as with the previous option avec_sources but f integrating also some operators of the Navier-Stokes equations). The two last options are useful and sometime necessary when source terms are implicited when using an implicit time scheme to solve the Navier-Stokes equations.
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation

- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

qdm_multiphase

Momentum conservation equation for a multi-phase problem where the unknown is the velocity

Parameters are:

- **[solveur_pression]** (*type: solveur_sys_base*) Linear pressure system resolution method.
- **[evanescence]** (*type: bloc_lecture*) Management of the vanishing phase (when alpha tends to 0 or 1)
- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.
- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

---

taux_dissipation_turbulent

Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)

Parameters are:

- **[disable_equation_residual]** (*type:* int) The equation residual will not be used for the problem residual used when checking time convergence or computing dynamic time-step
- **[convection]** (*type: bloc_convection*) Keyword to alter the convection scheme.
- **[diffusion]** (*type: bloc_diffusion*) Keyword to specify the diffusion operator.
- **[conditions_limites | boundary_conditions]** (*type:* list of Condlimlu) Boundary conditions.
- **[conditions_initiales | initial_conditions]** (*type:* list of Condinit) Initial conditions.
- **[sources]** (*type:* list of Source_base) The sources.

- **[ecrire_fichier_xyz_valeur]** (*type: ecrire_fichier_xyz_valeur*) This keyword is used to write the values of a field only for some boundaries in a text file
- **[parametre_equation]** (*type: parametre_equation_base*) Keyword used to specify additional parameters for the equation
- **[equation_non_resolue]** (*type:* string) The equation will not be solved while condition(t) is verified if equation_non_resolue keyword is used. Exemple: The Navier-Stokes equations are not solved between time t0 and t1. Navier_Sokes_Standard { equation_non_resolue (t>t0)*(t<t1) }
- **[renommer_equation | rename_equation]** (*type:* string) Rename the equation with a specific name.

1.3.21 Keywords derived from moyenne_imposee_deriv

moyenne_imposee_connexion_approchee

Synonyms: connexion_approchee

To read the imposed field from a file where positions and values are given (it is not necessary that the coordinates of points match the coordinates of the boundary faces, indeed, the nearest point of each face of the boundary will be used).

Parameters are:

- **fichier** (*type:* string into ['fichier']) not_set
- **file1** (*type:* string) filename. The format of the file is: N x(1) y(1) [z(1)] valx(1) valy(1) [valz(1)] x(2) y(2) [z(2)] valx(2) valy(2) [valz(2)] ... x(N) y(N) [z(N)] valx(N) valy(N) [valz(N)]

---

moyenne_imposee_connexion_exacte

Synonyms: connexion_exacte

To read the imposed field from two files.

Parameters are:

- **fichier** (*type:* string into ['fichier']) not_set
- **file1** (*type:* string) first file, contains the points coordinates (which should be the same as the coordinates of the boundary faces). The format of this file is: N 1 x(1) y(1) [z(1)] 2 x(2) y(2) [z(2)] ... N x(N) y(N) [z(N)]
- **[file2]** (*type:* string) second file, contains the mean values. The format of this file is: N 1 valx(1) valy(1) [valz(1)] 2 valx(2) valy(2) [valz(2)] ... N valx(N) valy(N) [valz(N)]

---

moyenne_imposee_deriv

not_set

---

moyenne_imposee_interpolation

Synonyms: champ_post_interpolation, interpolation

To create an imposed field built by interpolation of values read from a file. The imposed field is applied on the direction given by the keyword direction_anisotrope (the field is zero for the other directions).

Parameters are:

- **fichier** (*type:* string into ['fichier']) The format of the file is: pos(1) val(1) pos(2) val(2) ... pos(N) val(N) If direction given by direction_anisotrope is 1 (or 2 or 3), then pos will be X (or Y or Z) coordinate and val will be X value (or Y value, or Z value) of the imposed field.
- **file1** (*type:* string) name of geom_face_perio

---

moyenne_imposee_logarithmique

Synonyms: logarithmique

To specify the imposed field (in this case, velocity) by an analytical logarithmic law of the wall:

g(x,y,z) = u_tau * (log(0.5*diametre*u_tau/visco_cin)/Kappa + 5.1)

with g(x,y,z)=u(x,y,z) if direction is set to 1, g=v(x,y,z) if direction is set to 2 and g=w(w,y,z) if it is set to 3

Parameters are:

- **diametre** (*type:* string into ['diametre']) not_set
- **val** (*type:* float) diameter
- **u_tau** (*type:* string into ['u_tau']) not_set
- **val_u_tau | val_u_taul** (*type:* float) value of u_tau
- **visco_cin** (*type:* string into ['visco_cin']) not_set
- **val_visco_cin** (*type:* float) value of visco_cin
- **direction** (*type:* string into ['direction']) not_set
- **val_direction** (*type:* int) direction

---

moyenne_imposee_profil

Synonyms: profil

To specify analytic profile for the imposed g field.

Parameters are:

- **profile** (*type:* list of str) specifies the analytic profile: 2|3 valx(x,y,z,t) valy(x,y,z,t) [valz(x,y,z,t)]

1.3.22 Keywords derived from nom

nom

Class to name the TRUST objects.

Parameters are:

- **[mot]** (*type:* string) Chain of characters.

---

nom_anonyme

not_set

Parameters are:

- **[mot]** (*type:* string) Chain of characters.

1.3.23 Keywords derived from objet_lecture

binaire

Format of the file - binary version

Parameters are:

- **checkpoint_fname** (*type:* string) Name of file.

---

bloc_convection

not_set

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **operateur** (*type: convection_deriv*) not_set
- **acof** (*type:* string into ['}']) Closing curly bracket.

---

bloc_couronne

Class to create a couronne (2D).

Parameters are:

- **name** (*type:* string into ['origine']) Keyword to define the center of the circle.
- **origin | origine** (*type:* list of float) Center of the circle.
- **name3** (*type:* string into ['ri']) Keyword to define the interior radius.
- **ri** (*type:* float) Interior radius.
- **name4** (*type:* string into ['re']) Keyword to define the exterior radius.
- **re** (*type:* float) Exterior radius.

---

bloc_criteres_convergence

Not set

Parameters are:

- **bloc_lecture** (*type:* string) not_set

---

bloc_decouper

Auxiliary class to cut a domain.

Parameters are:

- **[partitionneur | partition_tool]** (*type: partitionneur_deriv*) Defines the partitionning algorithm (the effective C++ object used is 'Partitionneur_ALGORITHM_NAME').
- **[larg_joint]** (*type:* int) This keyword specifies the thickness of the virtual ghost domaine (data known by one processor though not owned by it). The default value is 1 and is generally correct for all algorithms except the QUICK convection scheme that require a thickness of 2. Since the 1.5.5 version, the VEF discretization imply also a thickness of 2 (except VEF P0). Any non-zero positive value can be used, but the amount of data to store and exchange between processors grows quickly with the thickness.
- **[nom_zones | zones_name]** (*type:* string) Name of the files containing the different partition of the domain. The files will be : name_0001.Zones name_0002.Zones ... name_000n.Zones. If this keyword is not specified, the geometry is not written on disk (you might just want to generate a 'ecrire_decoupage' or 'ecrire_lata').
- **[ecrire_decoupage]** (*type:* string) After having called the partitionning algorithm, the resulting partition is written on disk in the specified filename. See also partitionneur Fichier_Decoupage. This keyword is useful to change the partition numbers: first, you write the partition into a file with the option ecrire_decoupage. This file contains the domaine number for each element's mesh. Then you can easily permute domaine numbers in this file. Then read the new partition to create the .Zones files with the Fichier_Decoupage keyword.
- **[ecrire_lata]** (*type:* string) Save the partition field in a LATA format file for visualization
- **[ecrire_med]** (*type:* string) Save the partition field in a MED format file for visualization

- **[nb_parts_tot]** (*type:* int) Keyword to generates N .Domaine files, instead of the default number M obtained after the partitionning algorithm. N must be greater or equal to M. This option might be used to perform coupled parallel computations. Supplemental empty domaines from M to N-1 are created. This keyword is used when you want to run a parallel calculation on several domains with for example, 2 processors on a first domain and 10 on the second domain because the first domain is very small compare to second one. You will write Nb_parts 2 and Nb_parts_tot 10 for the first domain and Nb_parts 10 for the second domain.
- **[periodique]** (*type:* list of str) N BOUNDARY_NAME_1 BOUNDARY_NAME_2 ... : N is the number of boundary names given. Periodic boundaries must be declared by this method. The partitionning algorithm will ensure that facing nodes and faces in the periodic boundaries are located on the same processor.
- **[reorder]** (*type:* int) If this option is set to 1 (0 by default), the partition is renumbered in order that the processes which communicate the most are nearer on the network. This may slighlty improves parallel performance.
- **[single_hdf]** (*type:* flag) Optional keyword to enable you to write the partitioned domaines in a single file in hdf5 format.
- **[print_more_infos]** (*type:* int) If this option is set to 1 (0 by default), print infos about number of remote elements (ghosts) and additional infos about the quality of partitionning. Warning, it slows down the cutting operations.

---

bloc_diffusion

not_set

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **[operateur]** (*type: diffusion_deriv*) if none is specified, the diffusive scheme used is a 2nd-order scheme.
- **[op_implicite]** (*type: op_implicite*) To have diffusive implicitation, it use Uzawa algorithm. Very useful when viscosity has large variations.
- **acof** (*type:* string into ['}']) Closing curly bracket.

---

bloc_diffusion_standard

grad_Ubar 1 makes the gradient calculated through the filtered values of velocity (P1-conform).

nu 1 (respectively nut 1) takes the molecular viscosity (eddy viscosity) into account in the velocity gradient part of the diffusion expression.

nu_transp 1 (respectively nut_transp 1) takes the molecular viscosity (eddy viscosity) into account according in the TRANSPOSED velocity gradient part of the diffusion expression.

filtrer_resu 1 allows to filter the resulting diffusive fluxes contribution.

Parameters are:

- **mot1** (*type:* string into ['grad_ubar', 'nu', 'nut', 'nu_transp', 'nut_transp', 'filtrer_resu']) not_set
- **val1** (*type:* int into [0, 1]) not_set
- **mot2** (*type:* string into ['grad_ubar', 'nu', 'nut', 'nu_transp', 'nut_transp', 'filtrer_resu']) not_set
- **val2** (*type:* int into [0, 1]) not_set
- **mot3** (*type:* string into ['grad_ubar', 'nu', 'nut', 'nu_transp', 'nut_transp', 'filtrer_resu']) not_set

- **val3** (*type:* int into [0, 1]) not_set
- **mot4** (*type:* string into ['grad_ubar', 'nu', 'nut', 'nu_transp', 'nut_transp', 'filtrer_resu']) not_set
- **val4** (*type:* int into [0, 1]) not_set
- **mot5** (*type:* string into ['grad_ubar', 'nu', 'nut', 'nu_transp', 'nut_transp', 'filtrer_resu']) not_set
- **val5** (*type:* int into [0, 1]) not_set
- **mot6** (*type:* string into ['grad_ubar', 'nu', 'nut', 'nu_transp', 'nut_transp', 'filtrer_resu']) not_set
- **val6** (*type:* int into [0, 1]) not_set

---

bloc_ef

not_set

Parameters are:

- **mot1** (*type:* string into ['transportant_bar', 'transporte_bar', 'filtrer_resu', 'antisym']) not_set
- **val1** (*type:* int into [0, 1]) not_set
- **mot2** (*type:* string into ['transportant_bar', 'transporte_bar', 'filtrer_resu', 'antisym']) not_set
- **val2** (*type:* int into [0, 1]) not_set
- **mot3** (*type:* string into ['transportant_bar', 'transporte_bar', 'filtrer_resu', 'antisym']) not_set
- **val3** (*type:* int into [0, 1]) not_set
- **mot4** (*type:* string into ['transportant_bar', 'transporte_bar', 'filtrer_resu', 'antisym']) not_set
- **val4** (*type:* int into [0, 1]) not_set

---

bloc_fichier

Block containing the name of the file

Parameters are:

- **fichier | file** (*type:* string) File name

---

bloc_lec_champ_init_canal_sinal

Parameters for the class champ_init_canal_sinal.

in 2D:

U=ucent*y(2h-y)/h/h

V=ampli_bruit*rand+ampli_sin*sin(omega*x)

rand: unpredictable value between -1 and 1.

in 3D:

U=ucent*y(2h-y)/h/h

V=ampli_bruit*rand1+ampli_sin*sin(omega*x)

W=ampli_bruit*rand2

rand1 and rand2: unpredictables values between -1 and 1.

Parameters are:

- **ucent** (*type:* float) Velocity value at the center of the channel.
- **h** (*type:* float) Half hength of the channel.
- **ampli_bruit** (*type:* float) Amplitude for the disturbance.
- **[ampli_sin]** (*type:* float) Amplitude for the sinusoidal disturbance (by default equals to ucent/10).
- **omega** (*type:* float) Value of pulsation for the of the sinusoidal disturbance.
- **[dir_flow]** (*type:* int into [0, 1, 2]) Flow direction for the initialization of the flow in a channel. - if dir_flow=0, the flow direction is X - if dir_flow=1, the flow direction is Y - if dir_flow=2, the flow direction is Z Default value for dir_flow is 0
- **[dir_wall]** (*type:* int into [0, 1, 2]) Wall direction for the initialization of the flow in a channel. - if dir_wall=0, the normal to the wall is in X direction - if dir_wall=1, the normal to the wall is in Y direction - if dir_wall=2, the normal to the wall is in Z direction Default value for dir_flow is 1
- **[min_dir_flow]** (*type:* float) Value of the minimum coordinate in the flow direction for the initialization of the flow in a channel. Default value for dir_flow is 0.
- **[min_dir_wall]** (*type:* float) Value of the minimum coordinate in the wall direction for the initialization of the flow in a channel. Default value for dir_flow is 0.

---

bloc_lecture

to read between two braces

Parameters are:

- **bloc_lecture** (*type:* string) not_set

---

bloc_lecture_poro

Surface and volume porosity values.

Parameters are:

- **volumique** (*type:* float) Volume porosity value.
- **surfacique** (*type:* list of float) Surface porosity values (in X, Y, Z directions).

---

bloc_origine_cotes

Class to create a rectangle (or a box).

Parameters are:

- **name** (*type:* string into ['origine']) Keyword to define the origin of the rectangle (or the box).
- **origin | origine** (*type:* list of float) Coordinates of the origin of the rectangle (or the box).
- **name2** (*type:* string into ['cotes']) Keyword to define the length along the axes.
- **cotes** (*type:* list of float) Length along the axes.

---

bloc_pave

Class to create a pave.

Parameters are:

- **[origine]** (*type:* list of float) Keyword to define the pave (block) origin, that is to say one of the 8 block points (or 4 in a 2D coordinate system).
- **[longueurs]** (*type:* list of float) Keyword to define the block dimensions, that is to say knowing the origin, length along the axes.
- **[nombre_de_noeuds]** (*type:* list of int) Keyword to define the discretization (nodenumber) in each direction.
- **[facteurs]** (*type:* list of float) Keyword to define stretching factors for mesh discretization in each direction. This is a real number which must be positive (by default 1.0). A stretching factor other than 1 allows refinement on one edge in one direction.
- **[symx]** (*type:* flag) Keyword to define a block mesh that is symmetrical with respect to the YZ plane (respectively Y-axis in 2D) passing through the block centre.
- **[symy]** (*type:* flag) Keyword to define a block mesh that is symmetrical with respect to the XZ plane (respectively X-axis in 2D) passing through the block centre.
- **[symz]** (*type:* flag) Keyword defining a block mesh that is symmetrical with respect to the XY plane passing through the block centre.
- **[xtanh]** (*type:* float) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the X-direction.
- **[xtanh_dilatation]** (*type:* int into [-1, 0, 1]) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the X-direction. xtanh_dilatation: The value may be -1,0,1 (0 by default): 0: coarse mesh at the middle of the channel and smaller near the walls -1: coarse mesh at the left side of the channel and smaller at the right side 1: coarse mesh at the right side of the channel and smaller near the left side of the channel.
- **[xtanh_taille_premiere_maille]** (*type:* float) Size of the first cell of the mesh with tanh (hyperbolic tangent) variation in the X-direction.
- **[ytanh]** (*type:* float) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Y-direction.
- **[ytanh_dilatation]** (*type:* int into [-1, 0, 1]) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Y-direction. ytanh_dilatation: The value may be -1,0,1 (0 by default): 0: coarse mesh at the middle of the channel and smaller near the walls -1: coarse mesh at the bottom of the channel and smaller near the top 1: coarse mesh at the top of the channel and smaller near the bottom.
- **[ytanh_taille_premiere_maille]** (*type:* float) Size of the first cell of the mesh with tanh (hyperbolic tangent) variation in the Y-direction.

- **[ztanh]** (*type:* float) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Z-direction.
- **[ztanh_dilatation]** (*type:* int into [-1, 0, 1]) Keyword to generate mesh with tanh (hyperbolic tangent) variation in the Z-direction. tanh_dilatation: The value may be -1,0,1 (0 by default): 0: coarse mesh at the middle of the channel and smaller near the walls -1: coarse mesh at the back of the channel and smaller near the front 1: coarse mesh at the front of the channel and smaller near the back.
- **[ztanh_taille_premiere_maille]** (*type:* float) Size of the first cell of the mesh with tanh (hyperbolic tangent) variation in the Z-direction.

---

bloc_pdf_model

not_set

Parameters are:

- **eta** (*type:* float) penalization coefficient
- **[bilan_pdf]** (*type:* int) type de bilan du terme PDF (seul/avec temps/avec convection)
- **[temps_relaxation_coefficient_pdf]** (*type:* float) time relaxation on the forcing term to help
- **[echelle_relaxation_coefficient_pdf]** (*type:* float) time relaxation on the forcing term to help convergence
- **[local]** (*type:* flag) whether the prescribed velocity is expressed in the global or local basis
- **[vitesse_imposee_data]** (*type: field_base*) Prescribed velocity as a field
- **[vitesse_imposee_fonction]** (*type:* list of str) Prescribed velocity as a set of ananlytical component
- **[variable_imposee_data]** (*type: field_base*) Prescribed variable as a field
- **[variable_imposee_fonction]** (*type:* list of str) Prescribed variable as a set of ananlytical component

---

bloc_sutherland

Sutherland law for viscosity mu(T)=mu0*((T0+C)/(T+C))*(T/T0)**1.5 and (optional) for conductivity lambda(T)=mu0*Cp/Prandtl*((T0+Slambda)/(T+Slambda))*(T/T0)**1.5

Parameters are:

- **problem_name** (*type:* string) Name of problem.
- **mu0** (*type:* string into ['mu0']) not_set
- **mu0_val** (*type:* float) not_set
- **t0** (*type:* string into ['t0']) not_set
- **t0_val** (*type:* float) not_set
- **[slambda]** (*type:* string into ['slambda']) not_set
- **[s]** (*type:* float) not_set
- **c** (*type:* string into ['c']) not_set
- **c_val** (*type:* float) not_set

---

bloc_tube

Class to create a tube (3D).

Parameters are:

- **name** (*type:* string into ['origine']) Keyword to define the center of the tube.
- **origin | origine** (*type:* list of float) Center of the tube.
- **name2** (*type:* string into ['dir']) Keyword to define the direction of the main axis.
- **direction** (*type:* string into ['x', 'y', 'z']) direction of the main axis X, Y or Z
- **name3** (*type:* string into ['ri']) Keyword to define the interior radius.
- **ri** (*type:* float) Interior radius.
- **name4** (*type:* string into ['re']) Keyword to define the exterior radius.
- **re** (*type:* float) Exterior radius.
- **name5** (*type:* string into ['hauteur']) Keyword to define the heigth of the tube.
- **h** (*type:* float) Heigth of the tube.

---

bord

The block side is not in contact with another block and boundary conditions are applied to it.

Parameters are:

- **nom** (*type:* string) Name of block side.
- **defbord** (*type: defbord*) Definition of block side.

---

bord_base

Basic class for block sides. Block sides that are neither edges nor connectors are not specified. The duplicate nodes of two blocks in contact are automatically recognized and deleted.

---

calcul

The centre of gravity will be calculated.

---

canal

Keyword for statistics on a periodic plane channel.

Parameters are:

- **[dt_impr_moy_spat]** (*type:* float) Period to print the spatial average (default value is 1e6).
- **[dt_impr_moy_temp]** (*type:* float) Period to print the temporal average (default value is 1e6).
- **[debut_stat]** (*type:* float) Time to start the temporal averaging (default value is 1e6).
- **[fin_stat]** (*type:* float) Time to end the temporal averaging (default value is 1e6).
- **[pulsation_w]** (*type:* float) Pulsation for phase averaging (in case of pulsating forcing term) (no default value).
- **[nb_points_par_phase]** (*type:* int) Number of samples to represent phase average all along a period (no default value).
- **[reprise]** (*type:* string) val_moy_temp_xxxxxx.sauv : Keyword to resume a calculation with previous averaged quantities. Note that for thermal and turbulent problems, averages on temperature and turbulent viscosity are automatically calculated. To resume a calculation with phase averaging, val_moy_temp_xxxxxx.sauv_phase file is required on the directory where the job is submitted (this last file will be then automatically loaded by TRUST).

---

centre_de_gravite

To specify the centre of gravity.

Parameters are:

- **point** (*type:* *un_point*) A centre of gravity.

---

champ_a_post

Field to be post-processed.

Parameters are:

- **champ** (*type:* string) Name of the post-processed field.
- **[localisation]** (*type:* string into ['elem', 'som', 'faces']) Localisation of post-processed field values: The two available values are elem, som, or faces (LATA format only) used respectively to select field values at mesh centres (CHAMPMAILLE type field in the lml file) or at mesh nodes (CHAMPPOINT type field in the lml file). If no selection is made, localisation is set to som by default.

---

champs_posts

Field's write mode.

Parameters are:

- **[format]** (*type:* string into ['binaire', 'formatte']) Type of file.
- **[mot]** (*type:* string into ['dt_post', 'nb_pas_dt_post']) Keyword to set the kind of the field's write frequency. Either a time period or a time step period. it can be specified either here, or at the begining of the postprocessing bloc.
- **[period]** (*type:* string) Value of the period which can be like (2.*t).
- **champs | fields** (*type:* list of Champ_a_post) Fields to be post-processed.

---

champs_posts_fichier

Fields read from file.

Parameters are:

- **[format]** (*type:* string into ['binaire', 'formatte']) Type of file.
- **[mot]** (*type:* string into ['dt_post', 'nb_pas_dt_post']) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **[period]** (*type:* string) Value of the period which can be like (2.*t).
- **fichier | file** (*type: bloc_fichier*) name of file

---

chmoy_faceperio

non documente

Parameters are:

- **bloc** (*type: bloc_lecture*) not_set

---

circle

Keyword to define several probes located on a circle.

Parameters are:

- **nbr** (*type:* int) Number of probes between teta1 and teta2 (angles given in degrees).
- **point_deb** (*type: un_point*) Center of the circle.
- **[direction]** (*type:* int into [0, 1, 2]) Axis normal to the circle plane (0:x axis, 1:y axis, 2:z axis).
- **radius** (*type:* float) Radius of the circle.
- **theta1** (*type:* float) First angle.
- **theta2** (*type:* float) Second angle.

---

circle_3

Keyword to define several probes located on a circle (in 3-D space).

Parameters are:

- **nbr** (*type:* int) Number of probes between teta1 and teta2 (angles given in degrees).
- **point_deb** (*type: un_point*) Center of the circle.
- **direction** (*type:* int into [0, 1, 2]) Axis normal to the circle plane (0:x axis, 1:y axis, 2:z axis).
- **radius** (*type:* float) Radius of the circle.
- **theta1** (*type:* float) First angle.
- **theta2** (*type:* float) Second angle.

---

coarsen_operator_uniform

Object defining the uniform coarsening process of the given grid in IJK discretization

Parameters are:

- **[coarsen_operator_uniform]** (*type:* string) not_set
- **aco** (*type:* string into ['{']) opening curly brace
- **[coarsen_i]** (*type:* string into ['coarsen_i']) not_set
- **[coarsen_i_val]** (*type:* int) Integer indicating the number by which we will divide the number of elements in the I direction (in order to obtain a coarser grid)
- **[coarsen_j]** (*type:* string into ['coarsen_j']) not_set
- **[coarsen_j_val]** (*type:* int) Integer indicating the number by which we will divide the number of elements in the J direction (in order to obtain a coarser grid)
- **[coarsen_k]** (*type:* string into ['coarsen_k']) not_set
- **[coarsen_k_val]** (*type:* int) Integer indicating the number by which we will divide the number of elements in the K direction (in order to obtain a coarser grid)
- **acof** (*type:* string into ['}']) closing curly brace

---

condinit

Initial condition.

Parameters are:

- **nom** (*type:* string) Name of initial condition field.
- **ch** (*type: field_base*) Type field and the initial values.

---

condlimlu

Boundary condition specified.

Parameters are:

- **bord** (*type:* string) Name of the edge where the boundary condition applies.
- **cl** (*type:* *condlim_base*) Boundary condition at the boundary called bord (edge).

---

convection_ale

Synonyms: ale

A convective scheme for ALE (Arbitrary Lagrangian-Eulerian) framework.

Parameters are:

- **opconv** (*type:* *bloc_convection*) Choice between: amont and muscl Example: convection { ALE { amont } }

---

convection_amont

Synonyms: amont

Keyword for upwind scheme for VDF or VEF discretizations. In VEF discretization equivalent to generic amont for TRUST version 1.5 or later. The previous upwind scheme can be used with the obsolete in future amont_old keyword.

---

convection_amont_old

Synonyms: amont_old

Only for VEF discretization, obsolete keyword, see amont.

---

convection_btd

Synonyms: btd

Only for EF discretization.

Parameters are:

- **btd** (*type:* float) not_set
- **facteur** (*type:* float) not_set

---

convection_centre

Synonyms: centre

For VDF and VEF discretizations.

---

convection_centre4

Synonyms: centre4

For VDF and VEF discretizations.

---

convection_centre_old

Synonyms: centre_old

Only for VEF discretization.

---

convection_deriv

not_set

---

convection_di_l2

Synonyms: di_l2

Only for VEF discretization.

---

convection_ef

Synonyms: ef

For VEF calculations, a centred convective scheme based on Finite Elements formulation can be called through the following data:

Convection { EF transportant_bar val transporte_bar val antisym val filtrer_resu val }

This scheme is 2nd order accuracy (and get better the property of kinetic energy conservation). Due to possible problems of instabilities phenomena, this scheme has to be coupled with stabilisation process (see Source_Qdm_lambdaup).These two last data are equivalent from a theoretical point of view in variationnal writing to : div((u. grad ub , vb) - (u. grad vb, ub)), where vb corresponds to the filtered reference test functions.

Remark:

This class requires to define a filtering operator : see solveur_bar

Parameters are:

- **[mot1]** (*type:* string into ['defaut_bar']) equivalent to transportant_bar 0 transporte_bar 1 filtrer_resu 1 antisym 1
- **[bloc_ef]** (*type: bloc_ef*) not_set

---

convection_ef_stab

Synonyms: ef_stab

Keyword for a VEF convective scheme.

Parameters are:

- **[alpha]** (*type:* float) To weight the scheme centering with the factor double (between 0 (full centered) and 1 (mix between upwind and centered), by default 1). For scalar equation, it is adviced to use alpha=1 and for the momentum equation, alpha=0.2 is adviced.
- **[test]** (*type:* int) Developer option to compare old and new version of EF_stab
- **[tdivu]** (*type:* flag) To have the convective operator calculated as div(TU)-TdivU(=UgradT).
- **[old]** (*type:* flag) To use old version of EF_stab scheme (default no).
- **[volumes_etendus]** (*type:* flag) Option for the scheme to use the extended volumes (default, yes).
- **[volumes_non_etendus]** (*type:* flag) Option for the scheme to not use the extended volumes (default, no).
- **[amont_sous_zone]** (*type:* string) Option to degenerate EF_stab scheme into Amont (upwind) scheme in the sub zone of name sz_name. The sub zone may be located arbitrarily in the domain but the more often this option will be activated in a zone where EF_stab scheme generates instabilities as for free outlet for example.
- **[alpha_sous_zone]** (*type:* list of Sous_zone_valeur) List of groups of two words.

---

convection_generic

Synonyms: generic

Keyword for generic calling of upwind and muscl convective scheme in VEF discretization. For muscl scheme, limiters and order for fluxes calculations have to be specified. The available limiters are : minmod - vanleer -vanalbada - chakravarthy - superbee, and the order of accuracy is 1 or 2. Note that chakravarthy is a non-symmetric limiter and superbee may engender results out of physical limits. By consequence, these two limiters are not recommended.

Examples:

convection { generic amont }

convection { generic muscl minmod 1 }

convection { generic muscl vanleer 2 }

In case of results out of physical limits with muscl scheme (due for instance to strong non-conformal velocity flow field), user can redefine in data file a lower order and a smoother limiter, as : convection { generic muscl minmod 1 }

Parameters are:

- **type** (*type:* string into ['amont', 'muscl', 'centre']) type of scheme
- **[limiteur]** (*type:* string into ['minmod', 'vanleer', 'vanalbada', 'chakravarthy', 'superbee']) type of limiter
- **[ordre]** (*type:* int into [1, 2, 3]) order of accuracy

- **[alpha]** (*type:* float) alpha

---

convection_kquick

Synonyms: kquick

Only for VEF discretization.

---

convection_muscl

Synonyms: muscl

Keyword for muscl scheme in VEF discretization equivalent to generic muscl vanleer 2 for the 1.5 version or later. The previous muscl scheme can be used with the obsolete in future muscl_old keyword.

---

convection_muscl3

Synonyms: muscl3

Keyword for a scheme using a ponderation between muscl and center schemes in VEF.

Parameters are:

- **[alpha]** (*type:* float) To weight the scheme centering with the factor double (between 0 (full centered) and 1 (muscl), by default 1).

---

convection_muscl_new

Synonyms: muscl_new

Only for VEF discretization.

---

convection_muscl_old

Synonyms: muscl_old

Only for VEF discretization.

---

convection_negligeable

Synonyms: negligeable

For VDF and VEF discretizations. Suppresses the convection operator.

---

convection_quick

Synonyms: quick

Only for VDF discretization.

---

convection_supg

Synonyms: supg

Only for EF discretization.

Parameters are:

- **facteur** (*type:* float) not_set

---

corps_postraitement

not_set

Parameters are:

- **[t_debut_statistiques]** (*type:* float) not_set (for IJK)
- **[nb_pas_dt_post_stats_plans]** (*type:* float) not_set (for IJK)
- **[nb_pas_dt_post_stats_bulles]** (*type:* float) not_set (for IJK)
- **[expression_vx_ana]** (*type:* string) not_set (for IJK)
- **[expression_vy_ana]** (*type:* string) not_set (for IJK)
- **[expression_vz_ana]** (*type:* string) not_set (for IJK)
- **[expression_p_ana]** (*type:* string) not_set (for IJK)
- **[interfaces]** (*type: interface_posts*) Keyword to read all the caracteristics of the interfaces. Different kind of interfaces exist as well as different interface intitialisations.
- **[fichier]** (*type:* string) Name of file.
- **[format]** (*type:* string into ['lml', 'lata', 'single_lata', 'lata_v2', 'med', 'cgns']) This optional parameter specifies the format of the output file. The basename used for the output file is the basename of the data file. For the fmt parameter, choices are lml or lata. A short description of each format can be found below. The default value is lml.
- **[dt_post]** (*type:* string) Field's write frequency (as a time period) - can also be specified after the 'field' keyword.
- **[nb_pas_dt_post]** (*type:* int) Field's write frequency (as a number of time steps) - can also be specified after the 'field' keyword.

- **[domaine]** (*type:* string) This optional parameter specifies the domain on which the data should be interpolated before it is written in the output file. The default is to write the data on the domain of the current problem (no interpolation).
- **[sous_domaine | sous_zone]** (*type:* string) This optional parameter specifies the sub_domaine on which the data should be interpolated before it is written in the output file. It is only available for sequential computation.
- **[parallele]** (*type:* string into ['simple', 'multiple', 'mpi-io']) Select simple (single file, sequential write), multiple (several files, parallel write), or mpi-io (single file, parallel write) for LATA format
- **[definition_champs]** (*type:* list of Definition_champ) List of definition champ
- **[definition_champs_fichier | definition_champs_file]** (*type:* *definition_champs_fichier*) Definition_champs read from file.
- **[sondes | probes]** (*type:* list of Sonde) List of probes.
- **[sondes_fichier | probes_file]** (*type:* *sondes_fichier*) Probe read from a file.
- **[sondes_mobiles | mobile_probes]** (*type:* list of Sonde) List of probes.
- **[sondes_mobiles_fichier | mobile_probes_file]** (*type:* *sondes_fichier*) Mobile probes read in a file
- **[deprecatedkeepduplicatedprobes]** (*type:* int) Flag to not remove duplicated probes in .son files (1: keep duplicate probes, 0: remove duplicate probes)
- **[champs | fields]** (*type:* *champs_posts*) Field's write mode.
- **[champs_fichier | fields_file]** (*type:* *champs_posts_fichier*) Fields read from file.
- **[statistiques | statistics]** (*type:* *stats_posts*) Statistics between two points fixed : start of integration time and end of integration time.
- **[statistiques_fichier | statistics_file]** (*type:* *stats_posts_fichier*) Statistics read from file.
- **[statistiques_en_serie | serial_statistics]** (*type:* *stats_serie_posts*) Statistics between two points not fixed : on period of integration.
- **[statistiques_en_serie_fichier | serial_statistics_file]** (*type:* *stats_serie_posts_fichier*) Serial_statistics read from a file
- **[suffix_for_reset]** (*type:* string) Suffix used to modify the postprocessing file name if the ICoCo resetTime() method is invoked.

---

defbord

Class to define an edge.

---

defbord_2

1-D edge (straight line) in the 2-D space.

Parameters are:

- **dir** (*type:* string into ['x', 'y']) Edge is perpendicular to this direction.
- **eq** (*type:* string into ['=']) Equality sign.
- **pos** (*type:* float) Position value.

- **pos2_min** (*type:* float) Minimal value.
- **inf1** (*type:* string into ['<=']) Less than or equal to sign.
- **dir2** (*type:* string into ['x', 'y']) Edge is parallel to this direction.
- **inf2** (*type:* string into ['<=']) Less than or equal to sign.
- **pos2_max** (*type:* float) Maximal value.

---

defbord_3

2-D edge (plane) in the 3-D space.

Parameters are:

- **dir** (*type:* string into ['x', 'y', 'z']) Edge is perpendicular to this direction.
- **eq** (*type:* string into ['=']) Equality sign.
- **pos** (*type:* float) Position value.
- **pos2_min** (*type:* float) Minimal value.
- **inf1** (*type:* string into ['<=']) Less than or equal to sign.
- **dir2** (*type:* string into ['x', 'y']) Edge is parallel to this direction.
- **inf2** (*type:* string into ['<=']) Less than or equal to sign.
- **pos2_max** (*type:* float) Maximal value.
- **pos3_min** (*type:* float) Minimal value.
- **inf3** (*type:* string into ['<=']) Less than or equal to sign.
- **dir3** (*type:* string into ['y', 'z']) Edge is parallel to this direction.
- **inf4** (*type:* string into ['<=']) Less than or equal to sign.
- **pos3_max** (*type:* float) Maximal value.

---

definition_champ

Keyword to create new complex field for advanced postprocessing.

Parameters are:

- **name** (*type:* string) The name of the new created field.
- **champ_generique** (*type: champ_generique_base*) not_set

---

definition_champs_fichier

Keyword to read definition_champs from a file

Parameters are:

- **fichier | file** (*type:* string) name of file

---

deuxentiers

Two integers.

Parameters are:

- **int1** (*type:* int) First integer.
- **int2** (*type:* int) Second integer.

---

deuxmots

Two words.

Parameters are:

- **mot_1** (*type:* string) First word.
- **mot_2** (*type:* string) Second word.

---

diffusion_deriv

not_set

---

diffusion_negligeable

Synonyms: negligeable

the diffusivity will not taken in count

---

diffusion_option

Synonyms: option

not_set

Parameters are:

- **bloc_lecture** (*type:* *bloc_lecture*) not_set

---

diffusion_p1ncp1b

Synonyms: p1ncp1b

not_set

---

diffusion_stab

Synonyms: stab

keyword allowing consistent and stable calculations even in case of obtuse angle meshes.

Parameters are:

- **[standard]** (*type:* int) to recover the same results as calculations made by standard laminar diffusion operator. However, no stabilization technique is used and calculations may be unstable when working with obtuse angle meshes (by default 0)
- **[info]** (*type:* int) developer option to get the stabilizing ratio (by default 0)
- **[new_jacobian]** (*type:* int) when implicit time schemes are used, this option defines a new jacobian that may be more suitable to get stationary solutions (by default 0)
- **[nu]** (*type:* int) (respectively nut 1) takes the molecular viscosity (resp. eddy viscosity) into account in the velocity gradient part of the diffusion expression (by default nu=1 and nut=1)
- **[nut]** (*type:* int) not_set
- **[nu_transp]** (*type:* int) (respectively nut_transp 1) takes the molecular viscosity (resp. eddy viscosity) into account in the transposed velocity gradient part of the diffusion expression (by default nu_transp=0 and nut_transp=1)
- **[nut_transp]** (*type:* int) not_set

---

diffusion_standard

Synonyms: standard

A new keyword, intended for LES calculations, has been developed to optimise and parameterise each term of the diffusion operator. Remark:

1. This class requires to define a filtering operator : see solveur_bar

2. The former (original) version: diffusion { } -which omitted some of the term of the diffusion operator- can be recovered by using the following parameters in the new class :

diffusion { standard grad_Ubar 0 nu 1 nut 1 nu_transp 0 nut_transp 1 filtrer_resu 0}.

Parameters are:

- **[mot1]** (*type:* string into ['defaut_bar']) equivalent to grad_Ubar 1 nu 1 nut 1 nu_transp 1 nut_transp 1 filtrer_resu 1
- **[bloc_diffusion_standard]** (*type: bloc_diffusion_standard*) not_set

---

diffusion_turbulente_multiphase

Synonyms: turbulente

Turbulent diffusion operator for multiphase problem

Parameters are:

- **[type]** (*type: type_diffusion_turbulente_multiphase_deriv*) Turbulence model for multiphase problem

---

difusion_p1b

Synonyms: p1b

not_set

---

domain

Class to reuse a domain.

Parameters are:

- **domain_name** (*type:* string) Name of domain.

---

dt_impr_nusselt_mean_only

not_set

Parameters are:

- **dt_impr** (*type:* float) not_set
- **[boundaries]** (*type:* list of str) not_set

---

dt_impr_ustar_mean_only

not_set

Parameters are:

- **dt_impr** (*type:* float) not_set
- **[boundaries]** (*type:* list of str) not_set

---

ec

Keyword to print total kinetic energy into the referential linked to the domain (keyword Ec). In the case where the domain is moving into a Galilean referential, the keyword Ec_dans_repere_fixe will print total kinetic energy in the Galilean referential whereas Ec will print the value calculated into the moving referential linked to the domain

Parameters are:

- **[ec]** (*type:* flag) not_set
- **[ec_dans_repere_fixe]** (*type:* flag) not_set
- **[periode]** (*type:* float) periode is the keyword to set the period of printing into the file datafile_Ec.son or datafile_Ec_dans_repere_fixe.son.

---

entierfloat

An integer and a real.

Parameters are:

- **the_int** (*type:* int) Integer.
- **the_float** (*type:* float) Real.

---

epsilon

Two points will be confused if the distance between them is less than eps. By default, eps is set to 1e-12. The keyword Epsilon allows an alternative value to be assigned to eps.

Parameters are:

- **eps** (*type:* float) New value of precision.

---

floatfloat

Two reals.

Parameters are:

- **a** (*type:* float) First real.
- **b** (*type:* float) Second real.

---

fonction_champ_reprise

not_set

Parameters are:

- **mot** (*type:* string into ['fonction']) not_set
- **fonction** (*type:* list of str) n f1(val) f2(val) ... fn(val)] time

---

form_a_nb_points

The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.

Parameters are:

- **nb** (*type:* int into [4]) Number of points.
- **dir1** (*type:* int) First direction.
- **dir2** (*type:* int) Second direction.

---

format_file_base

Format of the file

Parameters are:

- **checkpoint_fname** (*type:* string) Name of file.

---

format_lata_to_cgns

not_set

Parameters are:

- **mot** (*type:* string into ['format_post_sup']) not_set
- **[format]** (*type:* string into ['lml', 'lata', 'lata_v2', 'med', 'cgns']) generated file post_CGNS.data use format (CGNS or LATA or LML keyword).

---

format_lata_to_med

not_set

Parameters are:

- **mot** (*type:* string into ['format_post_sup']) not_set
- **[format]** (*type:* string into ['lml', 'lata', 'lata_v2', 'med']) generated file post_med.data use format (MED or LATA or LML keyword).

---

formatte

Format of the file - formatte version

Parameters are:

- **checkpoint_fname** (*type:* string) Name of file.

---

info_med

not_set

Parameters are:

- **file_med** (*type:* string) Name of the MED file.
- **domaine** (*type:* string) Name of domain.
- **pb_post** (*type: pb_post*) not_set

---

interface_posts

not set

Parameters are:

- **[nom_interf]** (*type:* string) name of the interface to post process
- **blocs** (*type:* list of Champ_a_post) Fields to be post-processed.

---

internes

To indicate that the block has a set of internal faces (these faces will be duplicated automatically by the program and will be processed in a manner similar to edge faces).

Two boundaries with the same boundary conditions may have the same name (whether or not they belong to the same block).

The keyword Internes (Internal) must be used to execute a calculation with plates, followed by the equation of the surface area covered by the plates.

Parameters are:

- **nom** (*type:* string) Name of block side.
- **defbord** (*type: defbord*) Definition of block side.

---

lecture_bloc_moment_base

Auxiliary class to compute and print the moments.

---

longitudinale

Class to define the pressure loss in the direction of the tube bundle.

Parameters are:

- **dir** (*type:* string into ['x', 'y', 'z']) Direction.
- **dd** (*type:* float) Tube bundle hydraulic diameter value. This value is expressed in m.
- **ch_a** (*type:* string into ['a', 'cf']) Keyword to be used to set law coefficient values for the coefficient of regular pressure losses.
- **a** (*type:* float) Value of a law coefficient for regular pressure losses.
- **[ch_b]** (*type:* string into ['b']) Keyword to be used to set law coefficient values for regular pressure losses.
- **[b]** (*type:* float) Value of a law coefficient for regular pressure losses.

---

longueur_melange

This model is based on mixing length modelling. For a non academic configuration, formulation used in the code can be expressed basically as :

$nu_t=(Kappa.y)^2$.dU/dy

Till a maximum distance (dmax) set by the user in the data file, y is set equal to the distance from the wall (dist_w) calculated previously and saved in file Wall_length.xyz. [see Distance_paroi keyword]

Then (from y=dmax), y decreases as an exponential function : y=dmax*exp[-2.*(dist_w- dmax)/dmax]

Parameters are:

- **[canalx]** (*type:* float) [height] : plane channel according to Ox direction (for the moment, formulation in the code relies on fixed heigh : H=2).
- **[tuyauz]** (*type:* float) [diameter] : pipe according to Oz direction (for the moment, formulation in the code relies on fixed diameter : D=2).
- **[dmax]** (*type:* float) Maximum distance.
- **[fichier]** (*type:* string) not_set
- **[fichier_ecriture_k_eps]** (*type:* string) When a resume with k-epsilon model is envisaged, this keyword allows to generate external MED-format file with evaluation of k and epsilon quantities (based on eddy turbulent viscosity and turbulent characteristic length returned by mixing length model). The frequency of the MED file print is set equal to dt_impr_ustar. Moreover, k-eps MED field is automatically saved at the last time step. MED file is then used for resuming a K-Epsilon calculation with the Champ_Fonc_Med keyword.
- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.

- **[longueur_maille]** (*type:* string into ['volume', 'volume_sans_lissage', 'scotti', 'arrete']) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure). scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* float) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* float) Upper limitation of turbulent viscosity (default value 1.e8).
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* float) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

---

mailler_base

Basic class to mesh.

---

mod_turb_hyd_ss_maille

Class for sub-grid turbulence model for Navier-Stokes equations.

Parameters are:

- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.
- **[longueur_maille]** (*type:* string into ['volume', 'volume_sans_lissage', 'scotti', 'arrete']) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure). scotti : Characteristic length is based on the cubic root

of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.

- **[turbulence_paroi]** (*type:* *turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* float) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type:* *dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* float) Upper limitation of turbulent viscosity (default value 1.e8).
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* float) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

---

modele_turbulence_hyd_deriv

Basic class for turbulence model for Navier-Stokes equations.

Parameters are:

- **[turbulence_paroi]** (*type:* *turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* float) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type:* *dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* float) Upper limitation of turbulent viscosity (default value 1.e8).
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.

- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* float) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

---

modele_turbulence_hyd_null

Synonyms: null

Null turbulence model (turbulent viscosity = 0) which can be used with a turbulent problem.

Parameters are:

- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* float) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* float) Upper limitation of turbulent viscosity (default value 1.e8).
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* float) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

---

nom_postraitement

not_set

Parameters are:

- **nom** (*type:* string) Name of the post-processing.
- **post** (*type: postraitement_base*) the post

---

numero_elem_sur_maitre

Keyword to define a probe at the special element. Useful for min/max sonde.

Parameters are:

- **numero** (*type:* int) element number

---

objet_lecture

Auxiliary class for reading.

---

op_implicite

not_set

Parameters are:

- **implicite** (*type:* string into ['implicite']) not_set
- **mot** (*type:* string into ['solveur']) not_set
- **solveur** (*type: solveur_sys_base*) not_set

---

parametre_diffusion_implicite

To specify additional parameters for the equation when using impliciting diffusion

Parameters are:

- **[crank]** (*type:* int into [0, 1]) Use (1) or not (0, default) a Crank Nicholson method for the diffusion implicitation algorithm. Setting crank to 1 increases the order of the algorithm from 1 to 2.
- **[preconditionnement_diag]** (*type:* int into [0, 1]) The CG used to solve the implicitation of the equation diffusion operator is not preconditioned by default. If this option is set to 1, a diagonal preconditionning is used. Warning: this option is not necessarily more efficient, depending on the treated case.
- **[niter_max_diffusion_implicite]** (*type:* int) Change the maximum number of iterations for the CG (Conjugate Gradient) algorithm when solving the diffusion implicitation of the equation.
- **[seuil_diffusion_implicite]** (*type:* float) Change the threshold convergence value used by default for the CG resolution for the diffusion implicitation of this equation.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Conjugate Gradient) to solve the linear system.

---

parametre_equation_base

Basic class for parametre_equation

---

parametre_implicite

Keyword to change for this equation only the parameter of the implicit scheme used to solve the problem.

Parameters are:

- **[seuil_convergence_implicite]** (*type:* float) Keyword to change for this equation only the value of seuil_convergence_implicite used in the implicit scheme.
- **[seuil_convergence_solveur]** (*type:* float) Keyword to change for this equation only the value of seuil_convergence_solveur used in the implicit scheme
- **[solveur]** (*type:* *solveur_sys_base*) Keyword to change for this equation only the solver used in the implicit scheme
- **[resolution_explicite]** (*type:* flag) To solve explicitly the equation whereas the scheme is an implicit scheme.
- **[equation_non_resolue]** (*type:* flag) Keyword to specify that the equation is not solved.
- **[equation_frequence_resolue]** (*type:* string) Keyword to specify that the equation is solved only every n time steps (n is an integer or given by a time-dependent function f(t)).

---

pave

Class to create a pave (block) with boundaries.

Parameters are:

- **name** (*type:* string) Name of the pave (block).
- **bloc** (*type:* *bloc_pave*) Definition of the pave (block).
- **list_bord** (*type:* list of Bord_base) The block sides.

---

pdi

Format of the file - pdi version

Parameters are:

- **checkpoint_fname** (*type:* string) Name of file.

---

pdi_expert

Format of the file - PDI expert version

Parameters are:

- **yaml_fname** (*type:* string) YAML file name
- **checkpoint_fname** (*type:* string) Name of file.

---

plan

Keyword to set the number of probe layout points. The file format is type .lml

Parameters are:

- **nbr** (*type:* int) Number of probes in the first direction.
- **nbr2** (*type:* int) Number of probes in the second direction.
- **point_deb** (*type: un_point*) First point defining the angle. This angle should be positive.
- **point_fin** (*type: un_point*) Second point defining the angle. This angle should be positive.
- **point_fin_2** (*type: un_point*) Third point defining the angle. This angle should be positive.

---

point

Point as class-daughter of Points.

Parameters are:

- **points** (*type:* list of Un_point) Points.

---

points

Keyword to define the number of probe points. The file is arranged in columns.

Parameters are:

- **points** (*type:* list of Un_point) Points.

---

position_like

Keyword to define a probe at the same position of another probe named autre_sonde.

Parameters are:

- **autre_sonde** (*type:* string) Name of the other probe.

---

postraitement

Synonyms: post_processing

An object of post-processing (without name).

Parameters are:

- **[t_debut_statistiques]** (*type:* float) not_set (for IJK)
- **[nb_pas_dt_post_stats_plans]** (*type:* float) not_set (for IJK)
- **[nb_pas_dt_post_stats_bulles]** (*type:* float) not_set (for IJK)
- **[expression_vx_ana]** (*type:* string) not_set (for IJK)
- **[expression_vy_ana]** (*type:* string) not_set (for IJK)
- **[expression_vz_ana]** (*type:* string) not_set (for IJK)
- **[expression_p_ana]** (*type:* string) not_set (for IJK)
- **[interfaces]** (*type: interface_posts*) Keyword to read all the caracteristics of the interfaces. Different kind of interfaces exist as well as different interface intitialisations.
- **[fichier]** (*type:* string) Name of file.
- **[format]** (*type:* string into ['lml', 'lata', 'single_lata', 'lata_v2', 'med', 'cgns']) This optional parameter specifies the format of the output file. The basename used for the output file is the basename of the data file. For the fmt parameter, choices are lml or lata. A short description of each format can be found below. The default value is lml.
- **[dt_post]** (*type:* string) Field's write frequency (as a time period) - can also be specified after the 'field' keyword.
- **[nb_pas_dt_post]** (*type:* int) Field's write frequency (as a number of time steps) - can also be specified after the 'field' keyword.
- **[domaine]** (*type:* string) This optional parameter specifies the domain on which the data should be interpolated before it is written in the output file. The default is to write the data on the domain of the current problem (no interpolation).
- **[sous_domaine | sous_zone]** (*type:* string) This optional parameter specifies the sub_domaine on which the data should be interpolated before it is written in the output file. It is only available for sequential computation.
- **[parallele]** (*type:* string into ['simple', 'multiple', 'mpi-io']) Select simple (single file, sequential write), multiple (several files, parallel write), or mpi-io (single file, parallel write) for LATA format
- **[definition_champs]** (*type:* list of Definition_champ) List of definition champ
- **[definition_champs_fichier | definition_champs_file]** (*type: definition_champs_fichier*) Definition_champs read from file.
- **[sondes | probes]** (*type:* list of Sonde) List of probes.
- **[sondes_fichier | probes_file]** (*type: sondes_fichier*) Probe read from a file.
- **[sondes_mobiles | mobile_probes]** (*type:* list of Sonde) List of probes.
- **[sondes_mobiles_fichier | mobile_probes_file]** (*type: sondes_fichier*) Mobile probes read in a file
- **[deprecatedkeepduplicatedprobes]** (*type:* int) Flag to not remove duplicated probes in .son files (1: keep duplicate probes, 0: remove duplicate probes)
- **[champs | fields]** (*type: champs_posts*) Field's write mode.
- **[champs_fichier | fields_file]** (*type: champs_posts_fichier*) Fields read from file.

- **[statistiques | statistics]** (*type: stats_posts*) Statistics between two points fixed : start of integration time and end of integration time.
- **[statistiques_fichier | statistics_file]** (*type: stats_posts_fichier*) Statistics read from file.
- **[statistiques_en_serie | serial_statistics]** (*type: stats_serie_posts*) Statistics between two points not fixed : on period of integration.
- **[statistiques_en_serie_fichier | serial_statistics_file]** (*type: stats_serie_posts_fichier*) Serial_statistics read from a file
- **[suffix_for_reset]** (*type:* string) Suffix used to modify the postprocessing file name if the ICoCo resetTime() method is invoked.

---

postraitement_base

not_set

---

profils_thermo

non documente

Parameters are:

- **bloc** (*type: bloc_lecture*) not_set

---

quatremots

Three words.

Parameters are:

- **mot_1** (*type:* string) First word.
- **mot_2** (*type:* string) Snd word.
- **mot_3** (*type:* string) Third word.
- **mot_4** (*type:* string) Fourth word.

---

raccord

The block side is in contact with the block of another domain (case of two coupled problems).

Parameters are:

- **type1** (*type:* string into ['local', 'distant']) Contact type.
- **type2** (*type:* string into ['homogene']) Contact type.
- **nom** (*type:* string) Name of block side.
- **defbord** (*type: defbord*) Definition of block side.

---

radius

not_set

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **radius** (*type:* float) not_set
- **teta1** (*type:* float) not_set
- **teta2** (*type:* float) not_set

---

reaction

Keyword to describe reaction:

w =K pow(T,beta) exp(-Ea/(R T)) Pi pow(Reactif_i,activitivity_i).

If K_inv >0,

w= K pow(T,beta) exp(-Ea/(R T)) (Pi pow(Reactif_i,activitivity_i) - Kinv/exp(-c_r_Ea/(R T)) Pi pow(Produit_i,activitivity_i))

Parameters are:

- **reactifs** (*type:* string) LHS of equation (ex CH4+2*O2)
- **produits** (*type:* string) RHS of equation (ex CO2+2*H20)
- **[constante_taux_reaction]** (*type:* float) constante of cinetic K
- **enthalpie_reaction** (*type:* float) DH
- **energie_activation** (*type:* float) Ea
- **exposant_beta** (*type:* float) Beta
- **[coefficients_activites]** (*type:* *bloc_lecture*) coefficients od ativity (exemple { CH4 1 O2 2 })
- **[contre_reaction]** (*type:* float) K_inv
- **[contre_energie_activation]** (*type:* float) c_r_Ea

---

remove_elem_bloc

not_set

Parameters are:

- **[liste]** (*type:* list of int) not_set
- **[fonction]** (*type:* string) not_set

---

segment

Keyword to define the number of probe segment points. The file is arranged in columns.

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentfacesx

Segment probe where points are moved to the nearest x faces

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentfacesy

Segment probe where points are moved to the nearest y faces

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentfacesz

Segment probe where points are moved to the nearest z faces

Parameters are:

- **nbr** (*type:* int) Number of probe points of the segment, evenly distributed.
- **point_deb** (*type:* *un_point*) First outer probe segment point.
- **point_fin** (*type:* *un_point*) Second outer probe segment point.

---

segmentpoints

This keyword is used to define a probe segment from specifics points. The nom_champ field is sampled at ns specifics points.

Parameters are:

- **points** (*type:* list of Un_point) Points.

---

single_hdf

Format of the file - single_hdf version

Parameters are:

- **checkpoint_fname** (*type:* string) Name of file.

---

solveur_petsc_option_cli

solver

Parameters are:

- **bloc_lecture** (*type:* string) not_set

---

sonde

Keyword is used to define the probes. Observations: the probe coordinates should be given in Cartesian coordinates (X, Y, Z), including axisymmetric.

Parameters are:

- **nom_sonde** (*type:* string) Name of the file in which the values taken over time will be saved. The complete file name is nom_sonde.son.
- **[special]** (*type:* string into ['grav', 'som', 'nodes', 'chsom', 'gravcl']) Option to change the positions of the probes. Several options are available: grav : each probe is moved to the nearest cell center of the mesh; som : each probe is moved to the nearest vertex of the mesh nodes : each probe is moved to the nearest face center of the mesh; chsom : only available for P1NC sampled field. The values of the probes are calculated according to P1-Conform corresponding field. gravcl : Extend to the domain face boundary a cell-located segment probe in order to have the boundary condition for the field. For this type the extreme probe point has to be on the face center of gravity.
- **nom_inco** (*type:* string) Name of the sampled field.
- **mperiode** (*type:* string into ['periode']) Keyword to set the sampled field measurement frequency.
- **prd** (*type:* float) Period value. Every prd seconds, the field value calculated at the previous time step is written to the nom_sonde.son file.
- **type** (*type:* *sonde_base*) Type of probe.

---

sonde_base

Basic probe. Probes refer to sensors that allow a value or several points of the domain to be monitored over time. The probes may be a set of points defined one by one (keyword Points) or a set of points evenly distributed over a straight segment (keyword Segment) or arranged according to a layout (keyword Plan) or according to a parallelepiped (keyword Volume). The fields allow all the values of a physical value on the domain to be known at several moments in time.

---

sondes_fichier

Keyword to read probes from a file

Parameters are:

- **fichier | file** (*type:* string) name of file

---

sous_maille_smago

Smagorinsky sub-grid turbulence model.

Nut=Cs1*Cs1*l*l*sqrt(2*S*S)

K=Cs2*Cs2*l*l*2*S

Parameters are:

- **[cs]** (*type:* float) This is an optional keyword and the value is used to set the constant used in the Smagorinsky model (This is currently only valid for Smagorinsky models and it is set to 0.18 by default) .
- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.
- **[longueur_maille]** (*type:* string into ['volume', 'volume_sans_lissage', 'scotti', 'arrete']) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure). scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* float) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.

- **[nut_max]** (*type:* float) Upper limitation of turbulent viscosity (default value 1.e8).
- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* float) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

---

sous_maille_wale

This is the WALE-model. It is a new sub-grid scale model for eddy-viscosity in LES that has the following properties :

- it goes naturally to 0 at the wall (it doesn't need any information on the wall

position or geometry)

- it has the proper wall scaling in o(y3) in the vicinity of the wall
- it reproduces correctly the laminar to turbulent transition.

Parameters are:

- **[cw]** (*type:* float) The unique parameter (constant) of the WALE-model (by default value 0.5).
- **[formulation_a_nb_points]** (*type: form_a_nb_points*) The structure fonction is calculated on nb points and we should add the 2 directions (0:OX, 1:OY, 2:OZ) constituting the homegeneity planes. Example for channel flows, planes parallel to the walls.
- **[longueur_maille]** (*type:* string into ['volume', 'volume_sans_lissage', 'scotti', 'arrete']) Different ways to calculate the characteristic length may be specified : volume : It is the default option. Characteristic length is based on the cubic root of the volume cells. A smoothing procedure is applied to avoid discontinuities of this quantity in VEF from a cell to another. volume_sans_lissage : For VEF only. Characteristic length is based on the cubic root of the volume cells (without smoothing procedure). scotti : Characteristic length is based on the cubic root of the volume cells and the Scotti correction is applied to take into account the stretching of the cell in the case of anisotropic meshes. arete : For VEF only. Characteristic length relies on the max edge (+ smoothing procedure) is taken into account.
- **[turbulence_paroi]** (*type: turbulence_paroi_base*) Keyword to set the wall law.
- **[dt_impr_ustar]** (*type:* float) This keyword is used to print the values (U +, d+, u$star$) obtained with the wall laws into a file named datafile_ProblemName_Ustar.face and periode refers to the printing period, this value is expressed in seconds.
- **[dt_impr_ustar_mean_only]** (*type: dt_impr_ustar_mean_only*) This keyword is used to print the mean values of u* (obtained with the wall laws) on each boundary, into a file named datafile_ProblemName_Ustar_mean_only.out. periode refers to the printing period, this value is expressed in seconds. If you don't use the optional keyword boundaries, all the boundaries will be considered. If you use it, you must specify nb_boundaries which is the number of boundaries on which you want to calculate the mean values of u*, then you have to specify their names.
- **[nut_max]** (*type:* float) Upper limitation of turbulent viscosity (default value 1.e8).

- **[correction_visco_turb_pour_controle_pas_de_temps]** (*type:* flag) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is calculated so that diffusive time-step is equal or higher than convective time-step. For a stationary flow, the correction for turbulent viscosity should apply only during the first time steps and not when permanent state is reached. To check that, we could post process the corr_visco_turb field which is the correction of turbulent viscosity: it should be 1. on the whole domain.
- **[correction_visco_turb_pour_controle_pas_de_temps_parametre]** (*type:* float) Keyword to set a limitation to low time steps due to high values of turbulent viscosity. The limit for turbulent viscosity is the ratio between diffusive time-step and convective time-step is higher or equal to the given value [0-1]

---

sous_zone_valeur

Two words.

Parameters are:

- **sous_zone** (*type:* string) sous zone
- **valeur** (*type:* float) value

---

spec_pdcr_base

Class to read the source term modelling the presence of a bundle of tubes in a flow. Cf=A Re-B.

---

stat_post_correlation

Synonyms: correlation, champ_post_statistiques_correlation

correlation between the two fields

Parameters are:

- **first_field** (*type:* string) first field
- **second_field** (*type:* string) second field
- **[localisation]** (*type:* string into ['elem', 'som', 'faces']) Localisation of post-processed field value

---

stat_post_deriv

not_set

---

stat_post_ecart_type

Synonyms: ecart_type, champ_post_statistiques_ecart_type

to calculate the standard deviation (statistic rms) of the field

Parameters are:

- **field** (*type:* string) name of the field on which statistical analysis will be performed. Possible keywords are Vitesse (velocity), Pression (pressure), Temperature, Concentration, ...
- **[localisation]** (*type:* string into ['elem', 'som', 'faces']) Localisation of post-processed field value

---

stat_post_moyenne

Synonyms: champ_post_statistiques_moyenne, moyenne

to calculate the average of the field over time

Parameters are:

- **field** (*type:* string) name of the field on which statistical analysis will be performed. Possible keywords are Vitesse (velocity), Pression (pressure), Temperature, Concentration, ...
- **[localisation]** (*type:* string into ['elem', 'som', 'faces']) Localisation of post-processed field value

---

stat_post_t_deb

Synonyms: t_deb

Start of integration time

Parameters are:

- **val** (*type:* float) not_set

---

stat_post_t_fin

Synonyms: t_fin

End of integration time

Parameters are:

- **val** (*type:* float) not_set

---

stats_posts

Post-processing for statistics. Example:

Statistiques Dt_post dtst {

t_deb 0.1 *t_fin* 0.12

Moyenne Pression

Ecart_type Pression

Correlation Vitesse Vitesse

}

will write every dt_post the mean, standard deviation and correlation value:

if $t < t\_deb$ or $t > t\_fin$

$$average : \overline{P(t)} = 0$$

$$std\ deviation :< P(t) >= 0$$

$$correlation :< U(t).V(t) >= 0$$

if $t > t\_deb$ and $t < t\_fin$

$$average : \overline{P(t)} = \frac{1}{t - t_{deb}} \int_{t_{deb}}^{t} P(s)ds$$

$$std\ deviation :< P(t) >= \sqrt{\frac{1}{t - t_{deb}} \int_{t_{deb}}^{t} [P(s) - \overline{P(t)}]^2 ds}$$

$$correlation :< U(t).V(t) >= \frac{1}{t - t_{deb}} \int_{t_{deb}}^{t} [U(s) - \overline{U(t)}].[V(s) - \overline{V(t)}]ds$$

Parameters are:

- **[mot]** (*type:* string into ['dt_post', 'nb_pas_dt_post']) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **[period]** (*type:* string) Value of the period which can be like (2.*t).
- **champs | fields** (*type:* list of Stat_post_deriv) Post-processing for statistics

---

stats_posts_fichier

Statistics read from file.. Example:

Statistiques Dt_post dtst {

t_deb 0.1 *t_fin* 0.12

Moyenne Pression

Ecart_type Pression

Correlation Vitesse Vitesse

}

will write every dt_post the mean, standard deviation and correlation value:

if $t < t\_deb$ or $t > t\_fin$

$$average : \overline{P(t)} = 0$$

$$std\ deviation :< P(t) >= 0$$

$$correlation :< U(t).V(t) >= 0$$

if $t > t\_deb$ and $t < t\_fin$

$$average : \overline{P(t)} = \frac{1}{t - t_{deb}} \int_{t_{deb}}^{t} P(s)ds$$

$$std\ deviation :< P(t) >= \sqrt{\frac{1}{t - t_{deb}} \int_{t_{deb}}^{t} [P(s) - \overline{P(t)}]^2 ds}$$

$$correlation :< U(t).V(t) >= \frac{1}{t - t_{deb}} \int_{t_{deb}}^{t} [U(s) - \overline{U(t)}].[V(s) - \overline{V(t)}]ds$$

Parameters are:

- **mot** (*type:* string into ['dt_post', 'nb_pas_dt_post']) Keyword to set the kind of the field's write frequency. Either a time period or a time step period.
- **period** (*type:* string) Value of the period which can be like (2.*t).
- **fichier | file** (*type:* *bloc_fichier*) name of file

---

stats_serie_posts

This keyword is used to set the statistics. Average on dt_integr time interval is post- processed every dt_integr seconds. Example:

Statistiques_en_serie Dt_integr dtst {

Moyenne Pression

}

will calculate and write every dtst seconds the mean value:

$$(n+1)dt\_integr > t > n.dt\_integr, \overline{P(t)} = \frac{1}{t - n.dt_{integr}} \int_{n.dt_{integr}}^{t} P(t)dt$$

Parameters are:

- **mot** (*type:* string into ['dt_integr']) Keyword is used to set the statistics period of integration and write period.
- **dt_integr** (*type:* float) Average on dt_integr time interval is post-processed every dt_integr seconds.
- **stat** (*type:* list of Stat_post_deriv) Post-processing for statistics

---

stats_serie_posts_fichier

This keyword is used to set the statistics read from a file. Average on dt_integr time interval is post-processed every dt_integr seconds. Example:

Statistiques_en_serie Dt_integr dtst {

Moyenne Pression

}

will calculate and write every dtst seconds the mean value:

$$(n+1)dt\_integr > t > n.dt\_integr, \overline{P(t)} = \frac{1}{t - n.dt_{integr}} \int_{n.dt_{integr}}^{t} P(t)dt$$

Parameters are:

- **mot** (*type:* string into ['dt_integr']) Keyword is used to set the statistics period of integration and write period.
- **dt_integr** (*type:* float) Average on dt_integr time interval is post-processed every dt_integr seconds.
- **fichier | file** (*type:* *bloc_fichier*) name of file

---

temperature

not_set

Parameters are:

- **bord** (*type:* string) not_set
- **direction** (*type:* int) not_set

---

thi

Keyword for a THI (Homogeneous Isotropic Turbulence) calculation.

Parameters are:

- **init_ec** (*type:* int) Keyword to renormalize initial velocity so that kinetic energy equals to the value given by keyword val_Ec.
- **[val_ec]** (*type:* float) Keyword to impose a value for kinetic energy by velocity renormalizated if init_Ec value is 1.
- **[facon_init]** (*type:* int into [0, 1]) Keyword to specify how kinetic energy is computed (0 or 1).
- **[calc_spectre]** (*type:* int into [0, 1]) Calculate or not the spectrum of kinetic energy. Files called Sorties_THI are written with inside four columns : time:t global_kinetic_energy:Ec enstrophy:D skewness:S If calc_spectre is set to 1, a file Sorties_THI2_2 is written with three columns : time:t kinetic_energy_at_kc=32 enstrophy_at_kc=32 If calc_spectre is set to 1, a file spectre_xxxxx is written with two columns at each time xxxxx : frequency:k energy:E(k).
- **[periode_calc_spectre]** (*type:* float) Period for calculating spectrum of kinetic energy
- **[spectre_3d]** (*type:* int into [0, 1]) Calculate or not the 3D spectrum

- **[spectre_1d]** (*type:* int into [0, 1]) Calculate or not the 1D spectrum
- **[conservation_ec]** (*type:* flag) If set to 1, velocity field will be changed as to have a constant kinetic energy (default 0)
- **[longueur_boite]** (*type:* float) Length of the calculation domain

---

traitement_particulier

Auxiliary class to post-process particular values.

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **trait_part** (*type: traitement_particulier_base*) Type of traitement_particulier.
- **acof** (*type:* string into ['}']) Closing curly bracket.

---

traitement_particulier_base

Basic class to post-process particular values.

---

transversale

Class to define the pressure loss in the direction perpendicular to the tube bundle.

Parameters are:

- **dir** (*type:* string into ['x', 'y', 'z']) Direction.
- **dd** (*type:* float) Value of the tube bundle step.
- **chaine_d** (*type:* string into ['d']) Keyword to be used to set the value of the tube external diameter.
- **d** (*type:* float) Value of the tube external diameter.
- **ch_a** (*type:* string into ['a', 'cf']) Keyword to be used to set law coefficient values for the coefficient of regular pressure losses.
- **a** (*type:* float) Value of a law coefficient for regular pressure losses.
- **[ch_b]** (*type:* string into ['b']) Keyword to be used to set law coefficient values for regular pressure losses.
- **[b]** (*type:* float) Value of a law coefficient for regular pressure losses.

---

troisf

Auxiliary class to extrude.

Parameters are:

- **lx** (*type:* float) X direction of the extrude operation.
- **ly** (*type:* float) Y direction of the extrude operation.
- **lz** (*type:* float) Z direction of the extrude operation.

---

troismots

Three words.

Parameters are:

- **mot_1** (*type:* string) First word.
- **mot_2** (*type:* string) Snd word.
- **mot_3** (*type:* string) Third word.

---

type_diffusion_turbulente_multiphase_aire_interfaciale

Synonyms: interfacial_area, aire_interfaciale

not_set

Parameters are:

- **[cstdiff]** (*type:* float) Kataoka diffusion model constant. By default it is se to 0.236.
- **[ng2]** (*type:* float) not_set

---

type_diffusion_turbulente_multiphase_deriv

not_set

---

type_diffusion_turbulente_multiphase_l_melange

Synonyms: l_melange

not_set

Parameters are:

- **l_melange** (*type:* float) not_set

---

type_diffusion_turbulente_multiphase_prandtl

Synonyms: prandtl

Scalar Prandtl model.

Parameters are:

- **[pr_t | prandtl_turbulent]** (*type:* float) Prandtl's model constant. By default it is se to 0.9.

---

type_diffusion_turbulente_multiphase_sgdh

Synonyms: sgdh

not_set

Parameters are:

- **[pr_t | prandtl_turbulent]** (*type:* float) not_set
- **[sigma | sigma_turbulent]** (*type:* float) not_set
- **[no_alpha]** (*type:* flag) not_set
- **[gas_turb]** (*type:* flag) not_set

---

type_diffusion_turbulente_multiphase_smago

Synonyms: smago

LES Smagorinsky type.

Parameters are:

- **[cs]** (*type:* float) Smagorinsky's model constant. By default it is se to 0.18.

---

type_diffusion_turbulente_multiphase_wale

Synonyms: wale

LES WALE type.

Parameters are:

- **[cw]** (*type:* float) WALE's model constant. By default it is se to 0.5.

---

type_perte_charge_deriv

not_set

---

type_perte_charge_dp

Synonyms: dp

DP field should have 3 components defining dp, dDP/dQ, Q0

Parameters are:

- **dp_field** (*type: field_base*) the parameters of the previous formula (DP = dp + dDP/dQ * (Q - Q0)): uniform_field 3 dp dDP/dQ Q0 where Q0 is a mass flow rate (kg/s).

---

type_perte_charge_dp_regul

Synonyms: dp_regul

Keyword used to regulate the DP value in order to match a target flow rate. Syntax : dp_regul { DP0 d deb d eps e }

Parameters are:

- **dp0** (*type:* float) initial value of DP
- **deb** (*type:* string) target flow rate in kg/s
- **eps** (*type:* string) strength of the regulation (low values might be slow to find the target flow rate, high values might oscillate around the target value)

---

type_postraitement_ft_lata

not_set

Parameters are:

- **type** (*type:* string into ['postraitement_ft_lata']) not_set
- **nom** (*type:* string) Name of the post-processing.
- **bloc** (*type: bloc_lecture*) not_set

---

type_un_post

not_set

Parameters are:

- **type** (*type:* string into ['postraitement', 'post_processing']) not_set
- **post** (*type: un_postraitement*) not_set

---

un_pb

pour les groupes

Parameters are:

- **mot** (*type:* string) the string

---

un_point

A point.

Parameters are:

- **pos** (*type:* list of float) Point coordinates.

---

un_postraitement

An object of post-processing (with name).

Parameters are:

- **nom** (*type:* string) Name of the post-processing.
- **post** (*type: corps_postraitement*) Definition of the post-processing.

---

un_postraitement_spec

An object of post-processing (with type +name).

Parameters are:

- **[type_un_post]** (*type: type_un_post*) not_set
- **[type_postraitement_ft_lata]** (*type: type_postraitement_ft_lata*) not_set

---

verifiercoin_bloc

not_set

Parameters are:

- **[read_file | filename | lire_fichier]** (*type:* string) name of the *.decoupage_som file

---

volume

Keyword to define the probe volume in a parallelepiped passing through 4 points and the number of probes in each direction.

Parameters are:

- **nbr** (*type:* int) Number of probes in the first direction.
- **nbr2** (*type:* int) Number of probes in the second direction.
- **nbr3** (*type:* int) Number of probes in the third direction.
- **point_deb** (*type:* *un_point*) Point of origin.
- **point_fin** (*type:* *un_point*) Point defining the first direction (from point of origin).
- **point_fin_2** (*type:* *un_point*) Point defining the second direction (from point of origin).
- **point_fin_3** (*type:* *un_point*) Point defining the third direction (from point of origin).

---

xyz

Format of the file - xyz version

Parameters are:

- **checkpoint_fname** (*type:* string) Name of file.

1.3.24 Keywords derived from partitionneur_deriv

partitionneur_deriv

not_set

Parameters are:

- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_fichier_decoupage

Synonyms: fichier_decoupage

This algorithm reads an array of integer values on the disc, one value for each mesh element. Each value is interpreted as the target part number n>=0 for this element. The number of parts created is the highest value in the array plus one. Empty parts can be created if some values are not present in the array.

The file format is ASCII, and contains space, tab or carriage-return separated integer values. The first value is the number nb_elem of elements in the domain, followed by nb_elem integer values (positive or zero).

This algorithm has been designed to work together with the 'ecrire_decoupage' option. You can generate a partition with any other algorithm, write it to disc, modify it, and read it again to generate the .Zone files.

Contrary to other partitioning algorithms, no correction is applied by default to the partition (eg. element 0 on processor 0 and corrections for periodic boundaries). If 'corriger_partition' is specified, these corrections are applied.

Parameters are:

- **fichier** (*type:* string) File name
- **[corriger_partition]** (*type:* flag) not_set
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_fichier_med

Synonyms: fichier_med

Partitioning a domain using a MED file containing an integer field providing for each element the processor number on which the element should be located.

Parameters are:

- **file** (*type:* string) file name of the MED file to load
- **[field]** (*type:* string) field name of the integer (or double) field to load
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_metis

Synonyms: metis

Metis is an external partitionning library. It is a general algorithm that will generate a partition of the domain.

Parameters are:

- **[kmetis]** (*type:* flag) The default values are pmetis, default parameters are automatically chosen by Metis. 'kmetis' is faster than pmetis option but the last option produces better partitioning quality. In both cases, the partitioning quality may be slightly improved by increasing the nb_essais option (by default N=1). It will compute N partitions and will keep the best one (smallest edge cut number). But this option is CPU expensive, taking N=10 will multiply the CPU cost of partitioning by 10. Experiments show that only marginal improvements can be obtained with non default parameters.
- **[use_weights]** (*type:* flag) If use_weights is specified, weighting of the element-element links in the graph is used to force metis to keep opposite periodic elements on the same processor. This option can slightly improve the partitionning quality but it consumes more memory and takes more time. It is not mandatory since a correction algorithm is always applied afterwards to ensure a correct partitionning for periodic boundaries.
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_partition

Synonyms: partition_64, decouper, partition

This algorithm re-use the partition of the domain named DOMAINE_NAME. It is useful to partition for example a post processing domain. The partition should match with the calculation domain.

Parameters are:

- **domaine** (*type:* string) domain name
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_sous_dom

Synonyms: sous_dom

Given a global partition of a global domain, 'sous-domaine' allows to produce a conform partition of a sub-domain generated from the bigger one using the keyword create_domain_from_sub_domain. The sub-domain will be partitionned in a conform fashion with the global domain.

Parameters are:

- **fichier** (*type:* string) fichier
- **[fichier_ssz]** (*type:* string) fichier sous zonne
- **[name_ssz]** (*type:* string) nom sous zonne
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_sous_domaines

Synonyms: partitionneur_sous_zones, sous_zones

This algorithm will create one part for each specified subdomaine/domain. All elements contained in the first subdomaine/domain are put in the first part, all remaining elements contained in the second subdomaine/domain in the second part, etc…

If all elements of the current domain are contained in the specified subdomaines/domain, then N parts are created, otherwise, a supplemental part is created with the remaining elements.

If no subdomaine is specified, all subdomaines defined in the domain are used to split the mesh.

Parameters are:

- **[sous_zones]** (*type:* list of str) N SUBZONE_NAME_1 SUBZONE_NAME_2 …
- **[domaines]** (*type:* list of str) N DOMAIN_NAME_1 DOMAIN_NAME_2 …
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_tranche

Synonyms: tranche

This algorithm will create a geometrical partitionning by slicing the mesh in the two or three axis directions, based on the geometric center of each mesh element. nz must be given if dimension=3. Each slice contains the same number of elements (slices don't have the same geometrical width, and for VDF meshes, slice boundaries are generally not flat except if the number of mesh elements in each direction is an exact multiple of the number of slices). First, nx slices in the X direction are created, then each slice is split in ny slices in the Y direction, and finally, each part is split in nz slices in the Z direction. The resulting number of parts is nx*ny*nz. If one particular direction has been declared periodic, the default slicing (0, 1, 2, . . . , n-1)is replaced by (0, 1, 2, . . . n-1, 0), each of the two '0' slices having twice less elements than the other slices.

Parameters are:

- **[tranches]** (*type:* list of int) Partitioned by nx in the X direction, ny in the Y direction, nz in the Z direction. Works only for structured meshes. No warranty for unstructured meshes.
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

---

partitionneur_union

Synonyms: union

Let several local domains be generated from a bigger one using the keyword create_domain_from_sub_domain, and let their partitions be generated in the usual way. Provided the list of partition files for each small domain, the keyword 'union' will partition the global domain in a conform fashion with the smaller domains.

Parameters are:

- **liste** (*type: bloc_lecture*) List of the partition files with the following syntaxe: {sous_domaine1 decoupage1 . . . sous_domaineim decoupageim } where sous_domaine1 . . . sous_zomeim are small domains names and decoupage1 . . . decoupageim are partition files.
- **[nb_parts]** (*type:* int) The number of non empty parts that must be generated (generally equal to the number of processors in the parallel run).

1.3.25 Keywords derived from pb_champ_evaluateur

pb_champ_evaluateur

specifies problem name, the field name beloging to the problem and number of field components.

Parameters are:

- **pb** (*type:* string) name of the problem where the source fields will be searched.
- **champ** (*type:* string) name of the field
- **ncomp** (*type:* int) number of components

1.3.26 Keywords derived from pb_gen_base

coupled_problem

Synonyms: probleme_couple

This instruction causes a probleme_couple type object to be created. This type of object has an associated problem list, that is, the coupling of n problems among them may be processed. Coupling between these problems is carried out explicitly via conditions at particular contact limits. Each problem may be associated either with the Associate keyword or with the Read/groupes keywords. The difference is that in the first case, the four problems exchange values then calculate their timestep, rather in the second case, the same strategy is used for all the problems listed inside one group, but the second group of problem exchange values with the first group of problems after the first group did its timestep. So, the first case may then also be written like this:

Probleme_Couple pbc

Read pbc { groupes { { pb1 , pb2 , pb3 , pb4 } } }

There is a physical environment per problem (however, the same physical environment could be common to several problems).

Each problem is resolved in a domain.

Warning : Presently, coupling requires coincident meshes. In case of non-coincident meshes, boundary condition 'paroi_contact' in VEF returns error message (see paroi_contact for correcting procedure).

Parameters are:

- **[groupes]** (*type:* list of List_un_pb) pour les groupes

---

pb_avec_liste_conc

Class to create a classical problem with a list of scalar concentration equations.

Parameters are:

- **list_equations** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.

- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_avec_passif

Class to create a classical problem with a scalar transport equation (e.g: temperature or concentration) and an additional set of passive scalars (e.g: temperature or concentration) equations.

Parameters are:

- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_base

Resolution of equations on a domain. A problem is defined by creating an object and assigning the problem type that the user wishes to resolve. To enter values for the problem objects created, the Lire (Read) interpretor is used with a data block.

Parameters are:

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_conduction

Resolution of the heat equation.

Parameters are:

- **[solide]** (*type: solide*) The medium associated with the problem.
- **[conduction]** (*type: conduction*) Heat equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)

- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_conduction_ibm

Resolution of the IBM heat equation.

Parameters are:

- **[solide]** (*type: solide*) The medium associated with the problem.
- **[conduction_ibm]** (*type: conduction_ibm*) IBM Heat equation.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.

- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_gen_base

Basic class for problems.

---

pb_hydraulique

Resolution of the Navier-Stokes equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_standard** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_cloned_concentration

Resolution of Navier-Stokes/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport vectorial equation (concentration diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_cloned_concentration_turbulent

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration

Resolution of Navier-Stokes/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport vectorial equation (concentration diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.

- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration_scalaires_passifs

Resolution of Navier-Stokes/multiple constituent transport equations with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration_turbulent

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_concentration_turbulent_scalaires_passifs

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling and with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.

- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_ibm

Resolution of the IBM Navier-Stokes equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_ibm** (*type: navier_stokes_ibm*) IBM Navier-Stokes equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)

- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_ibm_turbulent

Resolution of Navier-Stokes equations with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_ibm_turbulent** (*type: navier_stokes_ibm_turbulent*) IBM Navier-Stokes equations as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_list_concentration

Resolution of Navier-Stokes/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **list_equations** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_list_concentration_turbulent

Resolution of Navier-Stokes/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **list_equations** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).

- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_melange_binaire_qc

Resolution of a binary mixture problem for a quasi-compressible fluid with an iso-thermal condition.

Keywords for the unknowns other than pressure, velocity, fraction_massique are :

masse_volumique : density

pression : reduced pressure

pression_tot : total pressure.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) The various constituants associated to the problem.
- **navier_stokes_qc** (*type: navier_stokes_qc*) Navier-Stokes equation for a quasi-compressible fluid.
- **convection_diffusion_espece_binaire_qc** (*type: convection_diffusion_espece_binaire_qc*) Species conservation equation for a binary quasi-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)

- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_melange_binaire_turbulent_qc

Resolution of a turbulent binary mixture problem for a quasi-compressible fluid with an iso-thermal condition.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent_qc** (*type: navier_stokes_turbulent_qc*) Navier-Stokes equation for a quasi-compressible fluid as well as the associated turbulence model equations.
- **convection_diffusion_espece_binaire_turbulent_qc** (*type: convection_diffusion_espece_binaire_turbulent_qc*) Species conservation equation for a quasi-compressible fluid as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.

- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_melange_binaire_wc

Resolution of a binary mixture problem for a weakly-compressible fluid with an iso-thermal condition.

Keywords for the unknowns other than pressure, velocity, fraction_massique are :

masse_volumique : density

pression : reduced pressure

pression_tot : total pressure

pression_hydro : hydro-static pressure

pression_eos : pressure used in state equation.

Parameters are:

- **fluide_weakly_compressible** (*type: fluide_weakly_compressible*) The fluid medium associated with the problem.
- **navier_stokes_wc** (*type: navier_stokes_wc*) Navier-Stokes equation for a weakly-compressible fluid.
- **convection_diffusion_espece_binaire_wc** (*type: convection_diffusion_espece_binaire_wc*) Species conservation equation for a binary weakly-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_hydraulique_turbulent

Resolution of Navier-Stokes equations with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_multiphase

A problem that allows the resolution of N-phases with 3*N equations

Parameters are:

- **[milieu_composite]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[milieu_musig]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[correlations]** (*type: bloc_lecture*) List of correlations used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **[models]** (*type: bloc_lecture*) List of models used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **qdm_multiphase** (*type: qdm_multiphase*) Momentum conservation equation for a multi-phase problem where the unknown is the velocity

- **masse_multiphase** (*type: masse_multiphase*) Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)
- **energie_multiphase** (*type: energie_multiphase*) Internal energy conservation equation for a multi-phase problem where the unknown is the temperature
- **[echelle_temporelle_turbulente]** (*type: echelle_temporelle_turbulente*) Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente]** (*type: energie_cinetique_turbulente*) Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente_wit]** (*type: energie_cinetique_turbulente_wit*) Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)
- **[taux_dissipation_turbulent]** (*type: taux_dissipation_turbulent*) Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_multiphase_enthalpie

Synonyms: pb_multiphase_h

A problem that allows the resolution of N-phases with 3*N equations

Parameters are:

- **[milieu_composite]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[correlations]** (*type: bloc_lecture*) List of correlations used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **qdm_multiphase** (*type: qdm_multiphase*) Momentum conservation equation for a multi-phase problem where the unknown is the velocity
- **masse_multiphase** (*type: masse_multiphase*) Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)
- **energie_multiphase_h | energie_multiphase_enthalpie** (*type: energie_multiphase_enthalpie*) Internal energy conservation equation for a multi-phase problem where the unknown is the enthalpy
- **[milieu_musig]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[models]** (*type: bloc_lecture*) List of models used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **[echelle_temporelle_turbulente]** (*type: echelle_temporelle_turbulente*) Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente]** (*type: energie_cinetique_turbulente*) Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente_wit]** (*type: energie_cinetique_turbulente_wit*) Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)
- **[taux_dissipation_turbulent]** (*type: taux_dissipation_turbulent*) Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see

schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.

- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_multiphase_hem

Synonyms: pb_hem

A problem that allows the resolution of 2-phases mechanicaly and thermally coupled with 3 equations

Parameters are:

- **[milieu_composite]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[milieu_musig]** (*type: bloc_lecture*) The composite medium associated with the problem.
- **[correlations]** (*type: bloc_lecture*) List of correlations used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **[models]** (*type: bloc_lecture*) List of models used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **qdm_multiphase** (*type: qdm_multiphase*) Momentum conservation equation for a multi-phase problem where the unknown is the velocity
- **masse_multiphase** (*type: masse_multiphase*) Mass consevation equation for a multi-phase problem where the unknown is the alpha (void fraction)
- **energie_multiphase** (*type: energie_multiphase*) Internal energy conservation equation for a multi-phase problem where the unknown is the temperature
- **[echelle_temporelle_turbulente]** (*type: echelle_temporelle_turbulente*) Turbulent Dissipation time scale equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente]** (*type: energie_cinetique_turbulente*) Turbulent kinetic Energy conservation equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[energie_cinetique_turbulente_wit]** (*type: energie_cinetique_turbulente_wit*) Bubble Induced Turbulent kinetic Energy equation for a turbulent multi-phase problem (available in TrioCFD)
- **[taux_dissipation_turbulent]** (*type: taux_dissipation_turbulent*) Turbulent Dissipation frequency equation for a turbulent mono/multi-phase problem (available in TrioCFD)
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)

- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_post

not_set

Parameters are:

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique

Resolution of thermohydraulic problem.

Parameters are:

- **[fluide_incompressible]** (*type: fluide_incompressible*) The fluid medium associated with the problem (only one possibility).
- **[fluide_ostwald]** (*type: fluide_ostwald*) The fluid medium associated with the problem (only one possibility).
- **[fluide_sodium_liquide]** (*type: fluide_sodium_liquide*) The fluid medium associated with the problem (only one possibility).
- **[fluide_sodium_gaz]** (*type: fluide_sodium_gaz*) The fluid medium associated with the problem (only one possibility).
- **[correlations]** (*type: bloc_lecture*) List of correlations used in specific source terms (i.e. interfacial flux, interfacial friction, ...)
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

pb_thermohydraulique_cloned_concentration

Resolution of Navier-Stokes/energy/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_cloned_concentration_turbulent

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.

- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_concentration

Resolution of Navier-Stokes/energy/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).

- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_concentration_scalaires_passifs

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_concentration]** (*type: convection_diffusion_concentration*) Constituent transport equations (concentration diffusion convection).
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equations (temperature diffusion convection).
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.

- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_concentration_turbulent

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.

- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_concentration_turbulent_scalaires_passifs

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling and with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_concentration_turbulent]** (*type: convection_diffusion_concentration_turbulent*) Constituent transport equations (concentration diffusion convection) as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equations (temperature diffusion convection) as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_especes_qc

Resolution of thermo-hydraulic problem for a multi-species quasi-compressible fluid.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_qc** (*type: navier_stokes_qc*) Navier-Stokes equation for a quasi-compressible fluid.
- **convection_diffusion_chaleur_qc** (*type: convection_diffusion_chaleur_qc*) Temperature equation for a quasi-compressible fluid.
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_especes_turbulent_qc

Resolution of turbulent thermohydraulic problem under low Mach number with passive scalar equations.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent_qc** (*type: navier_stokes_turbulent_qc*) Navier-Stokes equations under low Mach number as well as the associated turbulence model equations.
- **convection_diffusion_chaleur_turbulent_qc** (*type: convection_diffusion_chaleur_turbulent_qc*) Energy equation under low Mach number as well as the associated turbulence model equations.

- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_especes_wc

Resolution of thermo-hydraulic problem for a multi-species weakly-compressible fluid.

Parameters are:

- **fluide_weakly_compressible** (*type: fluide_weakly_compressible*) The fluid medium associated with the problem.
- **navier_stokes_wc** (*type: navier_stokes_wc*) Navier-Stokes equation for a weakly-compressible fluid.
- **convection_diffusion_chaleur_wc** (*type: convection_diffusion_chaleur_wc*) Temperature equation for a weakly-compressible fluid.
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)

- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_ibm

Resolution of IBM thermohydraulic problem.

Parameters are:

- **[fluide_incompressible]** (*type: fluide_incompressible*) The fluid medium associated with the problem (only one possibility).
- **[fluide_ostwald]** (*type: fluide_ostwald*) The fluid medium associated with the problem (only one possibility).
- **[navier_stokes_ibm]** (*type: navier_stokes_ibm*) IBM Navier-Stokes equations.
- **[convection_diffusion_temperature_ibm]** (*type: convection_diffusion_temperature_ibm*) IBM Energy equation (temperature diffusion convection).
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.

- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_ibm_turbulent

Resolution of thermohydraulic problem, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_ibm_turbulent** (*type: navier_stokes_ibm_turbulent*) IBM Navier-Stokes equations as well as the associated turbulence model equations.
- **convection_diffusion_temperature_ibm_turbulent** (*type: convection_diffusion_temperature_ibm_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_list_concentration

Resolution of Navier-Stokes/energy/multiple constituent transport equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equation (temperature diffusion convection).
- **list_equations** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_list_concentration_turbulent

Resolution of Navier-Stokes/energy/multiple constituent transport equations, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.

- **list_equations** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_qc

Resolution of thermo-hydraulic problem for a quasi-compressible fluid.

Keywords for the unknowns other than pressure, velocity, temperature are :

masse_volumique : density

enthalpie : enthalpy

pression : reduced pressure

pression_tot : total pressure.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_qc** (*type: navier_stokes_qc*) Navier-Stokes equation for a quasi-compressible fluid.
- **convection_diffusion_chaleur_qc** (*type: convection_diffusion_chaleur_qc*) Temperature equation for a quasi-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).

- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_scalaires_passifs

Resolution of thermohydraulic problem, with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_standard]** (*type: navier_stokes_standard*) Navier-Stokes equations.
- **[convection_diffusion_temperature]** (*type: convection_diffusion_temperature*) Energy equations (temperature diffusion convection).
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.

- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_turbulent

Resolution of thermohydraulic problem, with turbulence modelling.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.
- **convection_diffusion_temperature_turbulent** (*type: convection_diffusion_temperature_turbulent*) Energy equation (temperature diffusion convection) as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_turbulent_qc

Resolution of turbulent thermohydraulic problem under low Mach number.

Warning : Available for VDF and VEF P0/P1NC discretization only.

Parameters are:

- **fluide_quasi_compressible** (*type: fluide_quasi_compressible*) The fluid medium associated with the problem.
- **navier_stokes_turbulent_qc** (*type: navier_stokes_turbulent_qc*) Navier-Stokes equations under low Mach number as well as the associated turbulence model equations.
- **convection_diffusion_chaleur_turbulent_qc** (*type: convection_diffusion_chaleur_turbulent_qc*) Energy equation under low Mach number as well as the associated turbulence model equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_turbulent_scalaires_passifs

Resolution of thermohydraulic problem, with turbulence modelling and with the additional passive scalar equations.

Parameters are:

- **fluide_incompressible** (*type: fluide_incompressible*) The fluid medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituents.
- **[navier_stokes_turbulent]** (*type: navier_stokes_turbulent*) Navier-Stokes equations as well as the associated turbulence model equations.

- **[convection_diffusion_temperature_turbulent]** (*type: convection_diffusion_temperature_turbulent*) Energy equations (temperature diffusion convection) as well as the associated turbulence model equations.
- **equations_scalaires_passifs** (*type:* list of Eqn_base) List of equations.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pb_thermohydraulique_wc

Resolution of thermo-hydraulic problem for a weakly-compressible fluid.

Keywords for the unknowns other than pressure, velocity, temperature are :

masse_volumique : density

pression : reduced pressure

pression_tot : total pressure

pression_hydro : hydro-static pressure

pression_eos : pressure used in state equation.

Parameters are:

- **fluide_weakly_compressible** (*type: fluide_weakly_compressible*) The fluid medium associated with the problem.
- **navier_stokes_wc** (*type: navier_stokes_wc*) Navier-Stokes equation for a weakly-compressible fluid.
- **convection_diffusion_chaleur_wc** (*type: convection_diffusion_chaleur_wc*) Temperature equation for a weakly-compressible fluid.
- **[milieu]** (*type: milieu_base*) The medium associated with the problem.

- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).
- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).

---

pbc_med

Allows to read med files and post-process them.

Parameters are:

- **list_info_med** (*type:* list of Info_med) not_set

---

problem_read_generic

The probleme_read_generic differs rom the rest of the TRUST code : The problem does not state the number of equations that are enclosed in the problem. As the list of equations to be solved in the generic read problem is declared in the data file and not pre-defined in the structure of the problem, each equation has to be distinctively associated with the problem with the Associate keyword.

Parameters are:

- **[milieu]** (*type: milieu_base*) The medium associated with the problem.
- **[constituant]** (*type: constituant*) Constituent.
- **[postraitement | post_processing]** (*type: corps_postraitement*) One post-processing (without name).
- **[postraitements | post_processings]** (*type:* list of Un_postraitement) Keyword to use several results files. List of objects of post-processing (with name).

- **[liste_de_postraitements]** (*type:* list of Nom_postraitement) Keyword to use several results files. List of objects of post-processing (with name)
- **[liste_postraitements]** (*type:* list of Un_postraitement_spec) Keyword to use several results files. List of objects of post-processing (with name)
- **[sauvegarde]** (*type: format_file_base*) Keyword used when calculation results are to be backed up. When a coupling is performed, the backup-recovery file name must be well specified for each problem. In this case, you must save to different files and correctly specify these files when resuming the calculation.
- **[sauvegarde_simple]** (*type: format_file_base*) The same keyword than Sauvegarde except, the last time step only is saved.
- **[reprise]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file (see the class format_file). If format_reprise is xyz, the name_file file should be the .xyz file created by the previous calculation. With this file, it is possible to resume a parallel calculation on P processors, whereas the previous calculation has been run on N (N<>P) processors. Should the calculation be resumed, values for the tinit (see schema_temps_base) time fields are taken from the name_file file. If there is no backup corresponding to this time in the name_file, TRUST exits in error.
- **[resume_last_time]** (*type: format_file_base*) Keyword to resume a calculation based on the name_file file, resume the calculation at the last time found in the file (tinit is set to last time of saved files).
- **liste_equations** (*type:* list of Eqn_base) None

1.3.27 Keywords derived from porosites

porosites

To define the volume porosity and surface porosity that are uniform in every direction in space on a sub-area.

Porosity was only usable in VDF discretization, and now available for VEF P1NC/P0.

Observations :

- Surface porosity values must be given in every direction in space (set this value to 1

if there is no porosity),

- Prior to defining porosity, the problem must have been discretized.

Can 't be used in VEF discretization, use Porosites_champ instead.

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **sous_zone | sous_zone1** (*type:* string) Name of the sub-area to which porosity are allocated.
- **bloc** (*type: bloc_lecture_poro*) Surface and volume porosity values.
- **[sous_zone2]** (*type:* string) Name of the 2nd sub-area to which porosity are allocated.
- **[bloc2]** (*type: bloc_lecture_poro*) Surface and volume porosity values.
- **acof** (*type:* string into ['}']) Closing curly bracket.

1.3.28 Keywords derived from precond_base

ilu

This preconditionner can be only used with the generic GEN solver.

Parameters are:

- **[type]** (*type:* int) values can be 0|1|2|3 for null|left|right|left-and-right preconditionning (default value = 2)
- **[filling]** (*type:* int) default value = 1.

---

precond_base

Basic class for preconditioning.

---

precondsolv

not_set

Parameters are:

- **solveur** (*type: solveur_sys_base*) Solver type.

---

ssor

Symmetric successive over-relaxation algorithm.

Parameters are:

- **[omega]** (*type:* float) Over-relaxation facteur (between 1 and 2, default value 1.6).

---

ssor_bloc

not_set

Parameters are:

- **[precond0]** (*type: precond_base*) not_set
- **[precond1]** (*type: precond_base*) not_set
- **[preconda]** (*type: precond_base*) not_set
- **[alpha_0]** (*type:* float) not_set
- **[alpha_1]** (*type:* float) not_set
- **[alpha_a]** (*type:* float) not_set

1.3.29 Keywords derived from preconditionneur_petsc_deriv

preconditionneur_petsc_block_jacobi_icc

Synonyms: block_jacobi_icc

Incomplete Cholesky factorization for symmetric matrix with the PETSc implementation.

Parameters are:

- **[level]** (*type:* int) factorization level (default value, 1). In parallel, the factorization is done by block (one per processor by default).
- **[ordering]** (*type:* string into ['natural', 'rcm']) The ordering of the local matrix is natural by default, but rcm ordering, which reduces the bandwith of the local matrix, may interestingly improves the quality of the decomposition and reduces the number of iterations.

---

preconditionneur_petsc_block_jacobi_ilu

Synonyms: block_jacobi_ilu

preconditionner

Parameters are:

- **[level]** (*type:* int) not_set

---

preconditionneur_petsc_boomeramg

Synonyms: boomeramg

Multigrid preconditioner (no option is available yet, look at CLI command and Petsc documentation to try other options).

---

preconditionneur_petsc_c_amg

Synonyms: c-amg

preconditionner

---

preconditionneur_petsc_deriv

Preconditioners available with petsc solvers

---

preconditionneur_petsc_diag

Synonyms: diag

Diagonal (Jacobi) preconditioner.

---

preconditionneur_petsc_eisentat

Synonyms: eisentat

SSOR version with Eisenstat trick which reduces the number of computations and thus CPU cost…

Parameters are:

- **[omega]** (*type:* float) relaxation factor

---

preconditionneur_petsc_jacobi

Synonyms: jacobi

preconditionner

---

preconditionneur_petsc_lu

Synonyms: lu

preconditionner

---

preconditionneur_petsc_null

Synonyms: null

No preconditioner used

---

preconditionneur_petsc_pilut

Synonyms: pilut

Dual Threashold Incomplete LU factorization.

Parameters are:

- **[level]** (*type:* int) factorization level
- **[epsilon]** (*type:* float) drop tolerance

---

preconditionneur_petsc_sa_amg

Synonyms: sa-amg

preconditionner

---

preconditionneur_petsc_spai

Synonyms: spai

Spai Approximate Inverse algorithm from Parasails Hypre library.

Parameters are:

- **[level]** (*type:* int) first parameter
- **[epsilon]** (*type:* float) second parameter

---

preconditionneur_petsc_ssor

Synonyms: ssor

Symmetric Successive Over Relaxation algorithm.

Parameters are:

- **[omega]** (*type:* float) relaxation factor (default value, 1.5)

1.3.30 Keywords derived from schema_temps_base

euler_scheme

Synonyms: scheme_euler_explicit, schema_euler_explicite

This is the Euler explicit scheme.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).

- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.

- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

leap_frog

This is the leap-frog scheme.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.

- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_2

This is a low-storage Runge-Kutta scheme of second order that uses 2 integration points. The method is presented by Williamson (case 1) in https://www.sciencedirect.com/science/article/pii/0021999180900339

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.

- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_ordre_2_classique

This is a classical Runge-Kutta scheme of second order that uses 2 integration points.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set

- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_3

This is a low-storage Runge-Kutta scheme of third order that uses 3 integration points. The method is presented by Williamson (case 7) in https://www.sciencedirect.com/science/article/pii/0021999180900339

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_ordre_3_classique

This is a classical Runge-Kutta scheme of third order that uses 3 integration points.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set

- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_4

Synonyms: runge_kutta_ordre_4_d3p

This is a low-storage Runge-Kutta scheme of fourth order that uses 3 integration points. The method is presented by Williamson (case 17) in https://www.sciencedirect.com/science/article/pii/0021999180900339

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.

- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_ordre_4_classique

This is a classical Runge-Kutta scheme of fourth order that uses 4 integration points.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set

- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

runge_kutta_ordre_4_classique_3_8

This is a classical Runge-Kutta scheme of fourth order that uses 4 integration points and the 3/8 rule.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

runge_kutta_rationnel_ordre_2

This is the Runge-Kutta rational scheme of second order. The method is described in the note: Wambeck - Rational Runge-Kutta methods for solving systems of ordinary differential equations, at the link: https://link.springer.com/article/10.1007/BF02252381. Although rational methods require more computational work than linear ones, they can have some other properties, such as a stable behaviour with explicitness, which make them preferable. The CFD application of this RRK2 scheme is described in the note: https://link.springer.com/content/pdf/10.1007%2F3-540-13917-6_112.pdf.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.

- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

sch_cn_ex_iteratif

This keyword also describes a Crank-Nicholson method of second order accuracy but here, for scalars, because of instablities encountered when dt>dt_CFL, the Crank Nicholson scheme is not applied to scalar quantities. Scalars are treated according to Euler- Explicite scheme at the end of the CN treatment for velocity flow fields (by doing p Euler explicite under-iterations at dt<=dt_CFL). Parameters are the sames (but default values may change) compare to the Sch_CN_iterative scheme plus a relaxation keyword: niter_min (2 by default), niter_max (6 by default), niter_avg (3 by default), facsec_max (20 by default), seuil (0.05 by default)

Parameters are:

- **[omega]** (*type:* float) relaxation factor (0.1 by default)
- **[seuil]** (*type:* float) criteria for ending iterative process (Max(|| u(p) - u(p-1)||/Max || u(p) ||) < seuil) (0.001 by default)
- **[niter_min]** (*type:* int) minimal number of p-iterations to satisfy convergence criteria (2 by default)
- **[niter_max]** (*type:* int) number of maximum p-iterations allowed to satisfy convergence criteria (6 by default)
- **[niter_avg]** (*type:* int) threshold of p-iterations (3 by default). If the number of p-iterations is greater than niter_avg, facsec is reduced, if lesser than niter_avg, facsec is increased (but limited by the facsec_max value).
- **[facsec_max]** (*type:* float) maximum ratio allowed between dynamical time step returned by iterative process and stability time returned by CFL condition (2 by default).
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).

- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.

- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

sch_cn_iteratif

The Crank-Nicholson method of second order accuracy. A mid-point rule formulation is used (Euler-centered scheme). The basic scheme is: $$u(t+1) = u(t) + du/dt(t+1/2)*dt$$ The estimation of the time derivative du/dt at the level (t+1/2) is obtained either by iterative process. The time derivative du/dt at the level (t+1/2) is calculated iteratively with a simple under-relaxations method. Since the method is implicit, neither the cfl nor the fourier stability criteria must be respected. The time step is calculated in a way that the iterative procedure converges with the less iterations as possible.

Remark : for stationary or RANS calculations, no limitation can be given for time step through high value of facsec_max parameter (for instance : facsec_max 1000). In counterpart, for LES calculations, high values of facsec_max may engender numerical instabilities.

Parameters are:

- **[seuil]** (*type:* float) criteria for ending iterative process (Max(|| u(p) - u(p-1)||/Max || u(p) ||) < seuil) (0.001 by default)
- **[niter_min]** (*type:* int) minimal number of p-iterations to satisfy convergence criteria (2 by default)
- **[niter_max]** (*type:* int) number of maximum p-iterations allowed to satisfy convergence criteria (6 by default)
- **[niter_avg]** (*type:* int) threshold of p-iterations (3 by default). If the number of p-iterations is greater than niter_avg, facsec is reduced, if lesser than niter_avg, facsec is increased (but limited by the facsec_max value).
- **[facsec_max]** (*type:* float) maximum ratio allowed between dynamical time step returned by iterative process and stability time returned by CFL condition (2 by default).
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).

- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.

- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_adams_bashforth_order_2

not_set

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.

- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_adams_bashforth_order_3

not_set

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.

- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_adams_moulton_order_2

not_set

Parameters are:

- **[facsec_max]** (*type:* float) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value. Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice: The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: -Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 -Thermohydralic with natural convection, facsec around 300 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableThese values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does

not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_adams_moulton_order_3

not_set

Parameters are:

- **[facsec_max]** (*type:* float) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value. Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice: The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: -Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 -Thermohydralic with natural convection, facsec around 300 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableThese values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does

not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_backward_differentiation_order_2

not_set

Parameters are:

- **[facsec_max]** (*type:* float) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value. Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice: The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: -Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 -Thermohydralic with natural convection, facsec around 300 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableThese values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does

not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_backward_differentiation_order_3

not_set

Parameters are:

- **[facsec_max]** (*type:* float) Maximum ratio allowed between time step and stability time returned by CFL condition. The initial ratio given by facsec keyword is changed during the calculation with the implicit scheme but it couldn't be higher than facsec_max value. Warning: Some implicit schemes do not permit high facsec_max, example Schema_Adams_Moulton_order_3 needs facsec=facsec_max=1. Advice: The calculation may start with a facsec specified by the user and increased by the algorithm up to the facsec_max limit. But the user can also choose to specify a constant facsec (facsec_max will be set to facsec value then). Faster convergence has been seen and depends on the kind of calculation: -Hydraulic only or thermal hydraulic with forced convection and low coupling between velocity and temperature (Boussinesq value beta low), facsec between 20-30-Thermal hydraulic with forced convection and strong coupling between velocity and temperature (Boussinesq value beta high), facsec between 90-100 -Thermohydralic with natural convection, facsec around 300 -Conduction only, facsec can be set to a very high value (1e8) as if the scheme was unconditionally stableThese values can also be used as rule of thumb for initial facsec with a facsec_max limit higher.
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type:* *solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does

not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_euler_implicite

Synonyms: scheme_euler_implicit

This is the Euler implicit scheme.

Parameters are:

- **[facsec_max]** (*type:* float) For old syntax, see the complete parameters of facsec for details
- **[facsec_expert]** (*type: facsec_expert*) Advanced facsec specification
- **[facsec_func]** (*type:* string) Advanced facsec specification as a function
- **[resolution_monolithique]** (*type: bloc_lecture*) Activate monolithic resolution for coupled problems. Solves together the equations corresponding to the application domains in the given order. All aplication domains of the coupled equations must be given to determine the order of resolution. If the monolithic solving is not wanted for a specific application domain, an underscore can be added as prefix. For example, resolution_monolithique { dom1 { dom2 dom3 } _dom4 } will solve in a single matrix the equations having dom1 as application domain, then the equations having dom2 or dom3 as application domain in a single matrix, then the equations having dom4 as application domain in a sequential way (not in a single matrix).
- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type: solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does

not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.

- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

schema_implicite_base

Basic class for implicite time scheme.

Parameters are:

- **[max_iter_implicite]** (*type:* int) Maximum number of iterations allowed for the solver (by default 200).
- **solveur** (*type: solveur_implicite_base*) This keyword is used to designate the solver selected in the situation where the time scheme is an implicit scheme. solver is the name of the solver that allows equation diffusion and convection operators to be set as implicit terms. Keywords corresponding to this functionality are Simple (SIMPLE type algorithm), Simpler (SIMPLER type algorithm) for incompressible systems, Piso (Pressure Implicit with Split Operator), and Implicite (similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps, and ICE (for PB_multiphase). But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains. Advice: Since the 1.6.0 version, we recommend to use first the Implicite or Simple, then Piso, and at least Simpler. Because the two first give a fastest convergence (several times) than Piso and the Simpler has not been validated. It seems also than Implicite and Piso schemes give better results than the Simple scheme when the flow is not fully stationary. Thus, if the solution obtained with Simple is not stationary, it is recommended to switch to Piso or Implicite scheme.
- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec

value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.

- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_predictor_corrector

This is the predictor-corrector scheme (second order). It is more accurate and economic than MacCormack scheme. It gives best results with a second ordre convective scheme like quick, centre (VDF).

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).

- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.
- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.

- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

---

schema_temps_base

Basic class for time schemes. This scheme will be associated with a problem and the equations of this problem.

Parameters are:

- **[tinit]** (*type:* float) Value of initial calculation time (0 by default).
- **[tmax]** (*type:* float) Time during which the calculation will be stopped (1e30s by default).
- **[tcpumax]** (*type:* float) CPU time limit (must be specified in hours) for which the calculation is stopped (1e30s by default).
- **[dt_min]** (*type:* float) Minimum calculation time step (1e-16s by default).
- **[dt_max]** (*type:* string) Maximum calculation time step as function of time (1e30s by default).
- **[dt_sauv]** (*type:* float) Save time step value (1e30s by default). Every dt_sauv, fields are saved in the .sauv file. The file contains all the information saved over time. If this instruction is not entered, results are saved only upon calculation completion. To disable the writing of the .sauv files, you must specify 0. Note that dt_sauv is in terms of physical time (not cpu time).
- **[nb_sauv_max]** (*type:* int) Maximum number of timesteps that will be stored in backup file (10 by default). This value is only useful when doing a complete backup of the calculation with parallel PDI (as it needs to allocate the proper amount of dataspace in advance). If this number is reached (ie we already stored the data of nb_sauv_max timesteps in the file), the next checkpoints will overwrite the first ones
- **[dt_impr]** (*type:* float) Scheme parameter printing time step in time (1e30s by default). The time steps and the flux balances are printed (incorporated onto every side of processed domains) into the .out file.
- **[facsec]** (*type:* string) Value assigned to the safety factor for the time step (1. by default). It can also be a function of time. The time step calculated is multiplied by the safety factor. The first thing to try when a calculation does not converge with an explicit time scheme is to reduce the facsec to 0.5. Warning: Some schemes needs a facsec lower than 1 (0.5 is a good start), for example Schema_Adams_Bashforth_order_3.
- **[seuil_statio]** (*type:* float) Value of the convergence threshold (1e-12 by default). Problems using this type of time scheme converge when the derivatives dGi/dt of all the unknown transported values Gi have a combined absolute value less than this value. This is the keyword used to set the permanent rating threshold.
- **[residuals]** (*type: residuals*) To specify how the residuals will be computed (default max norm, possible to choose L2-norm instead).
- **[diffusion_implicite]** (*type:* int) Keyword to make the diffusive term in the Navier-Stokes equations implicit (in this case, it should be set to 1). The stability time step is then only based on the convection time step (dt=facsec*dt_convection). Thus, in some circumstances, an important gain is achieved with respect to the time step (large diffusion with respect to convection on tightened meshes). Caution: It is however recommended that the user avoids exceeding the convection time step by selecting a too large facsec value. Start with a facsec value of 1 and then increase it gradually if you wish to accelerate calculation. In addition, for a natural convection calculation with a zero initial velocity, in the first time step, the convection time is infinite and therefore dt=facsec*dt_max.

- **[seuil_diffusion_implicite]** (*type:* float) This keyword changes the default value (1e-6) of convergency criteria for the resolution by conjugate gradient used for implicit diffusion.
- **[impr_diffusion_implicite]** (*type:* int) Unactivate (default) or not the printing of the convergence during the resolution of the conjugate gradient.
- **[impr_extremums]** (*type:* int) Print unknowns extremas
- **[no_error_if_not_converged_diffusion_implicite]** (*type:* int) not_set
- **[no_conv_subiteration_diffusion_implicite]** (*type:* int) not_set
- **[dt_start]** (*type: dt_start*) dt_start dt_min : the first iteration is based on dt_min. dt_start dt_calc : the time step at first iteration is calculated in agreement with CFL condition. dt_start dt_fixe value : the first time step is fixed by the user (recommended when resuming calculation with Crank Nicholson temporal scheme to ensure continuity). By default, the first iteration is based on dt_calc.
- **[nb_pas_dt_max]** (*type:* int) Maximum number of calculation time steps (1e9 by default).
- **[niter_max_diffusion_implicite]** (*type:* int) This keyword changes the default value (number of unknowns) of the maximal iterations number in the conjugate gradient method used for implicit diffusion.
- **[precision_impr]** (*type:* int) Optional keyword to define the digit number for flux values printed into .out files (by default 3).
- **[periode_sauvegarde_securite_en_heures]** (*type:* float) To change the default period (23 hours) between the save of the fields in .sauv file.
- **[no_check_disk_space]** (*type:* flag) To disable the check of the available amount of disk space during the calculation.
- **[disable_progress]** (*type:* flag) To disable the writing of the .progress file.
- **[disable_dt_ev]** (*type:* flag) To disable the writing of the .dt_ev file.
- **[gnuplot_header]** (*type:* int) Optional keyword to modify the header of the .out files. Allows to use the column title instead of columns number.

1.3.31 Keywords derived from solveur_implicite_base

ice

Implicit Continuous-fluid Eulerian solver which is useful for a multiphase problem. Robust pressure reduction resolution.

Parameters are:

- **[pression_degeneree]** (*type:* int) Set to 1 if the pressure field is degenerate (ex. : incompressible fluid with no imposed-pressure BCs). Default: autodetected
- **[reduction_pression | pressure_reduction]** (*type:* int) Set to 1 if the user wants a resolution with a pressure reduction. Otherwise, the flag is to be set to 0 so that the complete matrix is considered. The default value of this flag is 1.
- **[criteres_convergence]** (*type: bloc_criteres_convergence*) Set the convergence thresholds for each unknown (i.e: alpha, temperature, velocity and pressure). The default values are respectively 0.01, 0.1, 0.01 and 100
- **[iter_min]** (*type:* int) Number of minimum iterations (default value 1)
- **[iter_max]** (*type:* int) Number of maximum iterations (default value 10)

- **[seuil_convergence_implicite]** (*type:* float) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[facsec_diffusion_for_sets]** (*type:* float) facsec to impose on the diffusion time step in sets while the total time step stays smaller than the convection time step.
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

implicite

similar to PISO, but as it looks like a simplified solver, it will use fewer timesteps. But it may run faster because the pressure matrix is not re-assembled and thus provides CPU gains.

Parameters are:

- **[seuil_convergence_implicite]** (*type:* float) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.

- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

piso

Piso (Pressure Implicit with Split Operator) - method to solve N_S.

Parameters are:

- **[seuil_convergence_implicite]** (*type:* float) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

sets

Stability-Enhancing Two-Step solver which is useful for a multiphase problem. Ref : J. H. MAHAFFY, A stability-enhancing two-step method for fluid flow calculations, Journal of Computational Physics, 46, 3, 329 (1982).

Parameters are:

- **[criteres_convergence]** (*type: bloc_criteres_convergence*) Set the convergence thresholds for each unknown (i.e: alpha, temperature, velocity and pressure). The default values are respectively 0.01, 0.1, 0.01 and 100
- **[iter_min]** (*type:* int) Number of minimum iterations (default value 1)

- **[iter_max]** (*type:* int) Number of maximum iterations (default value 10)
- **[seuil_convergence_implicite]** (*type:* float) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[facsec_diffusion_for_sets]** (*type:* float) facsec to impose on the diffusion time step in sets while the total time step stays smaller than the convection time step.
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

simple

SIMPLE type algorithm

Parameters are:

- **[relax_pression]** (*type:* float) Value between 0 and 1 (by default 1), this keyword is used only by the SIMPLE algorithm for relaxing the increment of pressure.
- **[seuil_convergence_implicite]** (*type:* float) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.

- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

simpler

Simpler method for incompressible systems.

Parameters are:

- **seuil_convergence_implicite** (*type:* float) Keyword to set the value of the convergence criteria for the resolution of the implicit system build to solve either the Navier_Stokes equation (only for Simple and Simpler algorithms) or a scalar equation. It is adviced to use the default value (1e6) to solve the implicit system only once by time step. This value must be decreased when a coupling between problems is considered.
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

---

solveur_implicite_base

Class for solver in the situation where the time scheme is the implicit scheme. Solver allows equation diffusion and convection operators to be set as implicit terms.

---

solveur_lineaire_std

not_set

Parameters are:

- **[solveur]** (*type: solveur_sys_base*) not_set

---

solveur_u_p

similar to simple.

Parameters are:

- **[relax_pression]** (*type:* float) Value between 0 and 1 (by default 1), this keyword is used only by the SIMPLE algorithm for relaxing the increment of pressure.
- **[seuil_convergence_implicite]** (*type:* float) Convergence criteria.
- **[nb_corrections_max]** (*type:* int) Maximum number of corrections performed by the PISO algorithm to achieve the projection of the velocity field. The algorithm may perform less corrections then nb_corrections_max if the accuracy of the projection is sufficient. (By default nb_corrections_max is set to 21).
- **[seuil_convergence_solveur]** (*type:* float) value of the convergence criteria for the resolution of the implicit system build by solving several times per time step the Navier_Stokes equation and the scalar equations if any. This value MUST be used when a coupling between problems is considered (should be set to a value typically of 0.1 or 0.01).
- **[seuil_generation_solveur]** (*type:* float) Option to create a GMRES solver and use vrel as the convergence threshold (implicit linear system Ax=B will be solved if residual error ||Ax-B|| is lesser than vrel).
- **[seuil_verification_solveur]** (*type:* float) Option to check if residual error ||Ax-B|| is lesser than vrel after the implicit linear system Ax=B has been solved.
- **[seuil_test_preliminaire_solveur]** (*type:* float) Option to decide if the implicit linear system Ax=B should be solved by checking if the residual error ||Ax-B|| is bigger than vrel.
- **[solveur]** (*type: solveur_sys_base*) Method (different from the default one, Gmres with diagonal preconditioning) to solve the linear system.
- **[no_qdm]** (*type:* flag) Keyword to not solve qdm equation (and turbulence models of these equation).
- **[nb_it_max]** (*type:* int) Keyword to set the maximum iterations number for the Gmres.
- **[controle_residu]** (*type:* flag) Keyword of Boolean type (by default 0). If set to 1, the convergence occurs if the residu suddenly increases.

1.3.32 Keywords derived from solveur_petsc_deriv

solveur_petsc_bicgstab

Synonyms: bicgstab

Stabilized Bi-Conjugate Gradient

Parameters are:

- **[precond]** (*type: preconditionneur_petsc_deriv*) not_set
- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_cholesky

Synonyms: cholesky

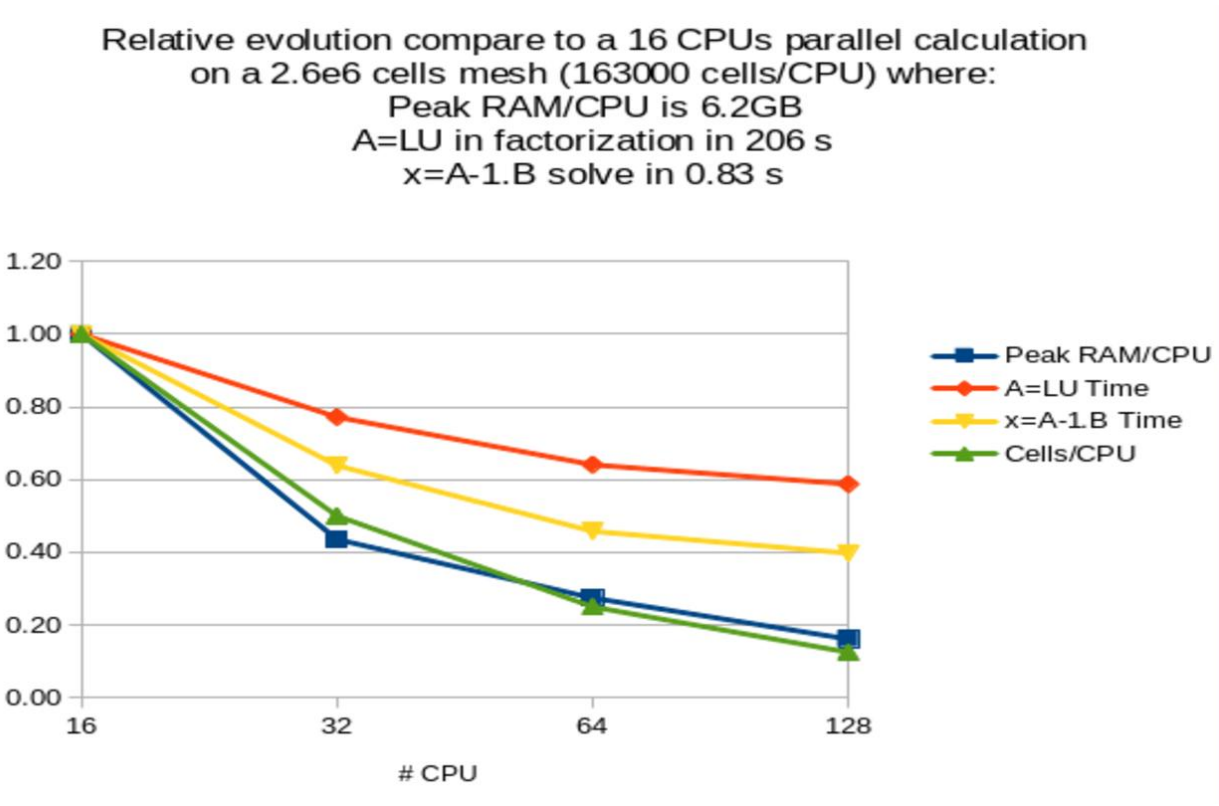
Parallelized version of Cholesky from MUMPS library. This solver accepts an option to select a different ordering than the automatic selected one by MUMPS (and printed by using the impr option). The possible choices are Metis, Scotch, PT-Scotch or Parmetis. The two last options can only be used during a parallel calculation, whereas the two first are available for sequential or parallel calculations. It seems that the CPU cost of A=LU factorization but also of the backward/forward elimination steps may sometimes be reduced by selecting a different ordering (Scotch seems often the best for b/f elimination) than the default one.

Notice that this solver requires a huge amont of memory compared to iterative methods. To know how much RAM you will need by core, then use the impr option to have detailled informations during the analysis phase and before the factorisation phase (in the following output, you will learn that the largest memory is taken by the zeroth CPU with 108MB):

Rank of proc needing largest memory in IC facto : 0

Estimated corresponding MBYTES for IC facto : 108

Thanks to the following graph, you read that in order to solve for instance a flow on a mesh with 2.6e6 cells, you will need to run a parallel calculation on 32 CPUs if you have cluster nodes with only 4GB/core (6.2GB*0.42~2.6GB) :



Parameters are:

- **[save_matrice | save_matrix]** (*type:* flag) not_set
- **[save_matrix_petsc_format]** (*type:* flag) not_set
- **[reduce_ram]** (*type:* flag) not_set
- **[cli_quiet]** (*type:* *solveur_petsc_option_cli*) not_set
- **[cli]** (*type:* *solveur_petsc_option_cli*) not_set
- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

solveur_petsc_cholesky_mumps_blr

Synonyms: cholesky_mumps_blr

BLR for (Block Low-Rank)

Parameters are:

- **[reduce_ram]** (*type:* flag) not_set
- **[dropping_parameter]** (*type:* float) not_set
- **[cli]** (*type: solveur_petsc_option_cli*) not_set
- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_cholesky_out_of_core

Synonyms: cholesky_out_of_core

Same as the previous one but with a written LU decomposition of disk (save RAM memory but add an extra CPU cost during Ax=B solve).

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_cholesky_pastix

Synonyms: cholesky_pastix

Parallelized Cholesky from PASTIX library.

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_cholesky_superlu

Synonyms: cholesky_superlu

Parallelized Cholesky from SUPERLU_DIST library (less CPU and RAM, efficient than the previous one)

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_cholesky_umfpack

Synonyms: cholesky_umfpack

Sequential Cholesky from UMFPACK library (seems fast).

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).

- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_cli

Synonyms: cli

Command Line Interface. Should be used only by advanced users, to access the whole solver/preconditioners from the PETSC API. To find all the available options, run your calculation with the -ksp_view -help options:

trust datafile [N] –ksp_view –help

-pc_type Preconditioner:(one of) none jacobi pbjacobi bjacobi sor lu shell mg eisenstat ilu icc cholesky asm ksp composite redundant nn mat fieldsplit galerkin openmp spai hypre tfs (PCSetType)

HYPRE preconditioner options:

-pc_hypre_type pilut (choose one of) pilut parasails boomeramg

HYPRE ParaSails Options

-pc_hypre_parasails_nlevels 1: Number of number of levels (None)

-pc_hypre_parasails_thresh 0.1: Threshold (None)

-pc_hypre_parasails_filter 0.1: filter (None)

-pc_hypre_parasails_loadbal 0: Load balance (None)

-pc_hypre_parasails_logging: FALSE Print info to screen (None)

-pc_hypre_parasails_reuse: FALSE Reuse nonzero pattern in preconditioner (None)

-pc_hypre_parasails_sym nonsymmetric (choose one of) nonsymmetric SPD nonsymmetric,SPD

Krylov Method (KSP) Options

-ksp_type Krylov method:(one of) cg cgne stcg gltr richardson chebychev gmres tcqmr bcgs bcgsl cgs tfqmr cr lsqr preonly qcg bicg fgmres minres symmlq lgmres lcd (KSPSetType)

-ksp_max_it 10000: Maximum number of iterations (KSPSetTolerances)

-ksp_rtol 0: Relative decrease in residual norm (KSPSetTolerances)

-ksp_atol 1e-12: Absolute value of residual norm (KSPSetTolerances)

-ksp_divtol 10000: Residual norm increase cause divergence (KSPSetTolerances)

-ksp_converged_use_initial_residual_norm: Use initial residual residual norm for computing relative convergence

-ksp_monitor_singular_value stdout: Monitor singular values (KSPMonitorSet)

-ksp_monitor_short stdout: Monitor preconditioned residual norm with fewer digits (KSPMonitorSet)

-ksp_monitor_draw: Monitor graphically preconditioned residual norm (KSPMonitorSet)

-ksp_monitor_draw_true_residual: Monitor graphically true residual norm (KSPMonitorSet)

Example to use the multigrid method as a solver, not only as a preconditioner:

Solveur_pression Petsc CLI {-ksp_type richardson -pc_type hypre -pc_hypre_type boomeramg -ksp_atol 1.e-7 }

Parameters are:

- **cli_bloc** (*type: bloc_lecture*) bloc

---

solveur_petsc_cli_quiet

Synonyms: cli_quiet

solver

Parameters are:

- **cli_quiet_bloc** (*type: bloc_lecture*) bloc

---

solveur_petsc_deriv

Additional information is available in the PETSC documentation: https://petsc.org/release/manual/

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_gcp

Synonyms: gcp

Preconditioned Conjugate Gradient

Parameters are:

- **[precond]** (*type: preconditionneur_petsc_deriv*) preconditioner
- **[precond_nul]** (*type:* flag) No preconditioner used, equivalent to precond null { }
- **[rtol]** (*type:* float) not_set
- **[reuse_preconditioner_nb_it_max]** (*type:* int) not_set
- **[cli]** (*type: solveur_petsc_option_cli*) not_set
- **[reorder_matrix]** (*type:* int) not_set

- **[read_matrix]** (*type:* flag) save_matrix|read_matrix are the keywords to save|read into a file the constant matrix A of the linear system Ax=B solved (eg: matrix from the pressure linear system for an incompressible flow). It is useful when you want to minimize the MPI communications on massive parallel calculation. Indeed, in VEF discretization, the overlapping width (generaly 2, specified with the largeur_joint option in the partition keyword partition) can be reduced to 1, once the matrix has been properly assembled and saved. The cost of the MPI communications in TRUST itself (not in PETSc) will be reduced with length messages divided by 2. So the strategy is: I) Partition your VEF mesh with a largeur_joint value of 2 II) Run your parallel calculation on 0 time step, to build and save the matrix with the save_matrix option. A file named Matrix_NBROWS_rows_NCPUS_cpus.petsc will be saved to the disk (where NBROWS is the number of rows of the matrix and NCPUS the number of CPUs used). III) Partition your VEF mesh with a largeur_joint value of 1 IV) Run your parallel calculation completly now and substitute the save_matrix option by the read_matrix option. Some interesting gains have been noticed when the cost of linear system solve with PETSc is small compared to all the other operations.
- **[save_matrice | save_matrix]** (*type:* flag) see read_matrix
- **[petsc_decide]** (*type:* int) not_set
- **[pcshell]** (*type:* string) not_set
- **[aij]** (*type:* flag) not_set
- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_gmres

Synonyms: gmres

Generalized Minimal Residual

Parameters are:

- **[precond]** (*type: preconditionneur_petsc_deriv*) not_set
- **[reuse_preconditioner_nb_it_max]** (*type:* int) not_set
- **[save_matrix_petsc_format]** (*type:* flag) not_set
- **[nb_it_max]** (*type:* int) In order to specify a given number of iterations instead of a condition on the residue with the keyword seuil. May be useful when defining a PETSc solver for the implicit time scheme where convergence is very fast: 5 or less iterations seems enough.
- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set

- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_ibicgstab

Synonyms: ibicgstab

Improved version of previous one for massive parallel computations (only a single global reduction operation instead of the usual 3 or 4).

Parameters are:

- **[precond]** (*type: preconditionneur_petsc_deriv*) not_set
- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_lu

Synonyms: lu

Several solvers through PETSc API are available.

TIPS:

A) Solver for symmetric linear systems (e.g: Pressure system from Navier-Stokes equations):

-The CHOLESKY parallel solver is from MUMPS library. It offers better performance than all others solvers if you have enough RAM for your calculation. A parallel calculation on a cluster with 4GBytes on each processor, 40000 cells/processor seems the upper limit. Seems to be very slow to initialize above 500 cpus/cores.

-When running a parallel calculation with a high number of cpus/cores (typically more than 500) where preconditioner scalabilty is the key for CPU performance, consider BICGSTAB with BLOCK_JACOBI_ICC(1) as preconditioner or if not converges, GCP with BLOCK_JACOBI_ICC(1) as preconditioner.

-For other situations, the first choice should be GCP/SSOR. In order to fine tune the solver choice, each one of the previous list should be considered. Indeed, the CPU speed of a solver depends of a lot of parameters. You may give a try to the OPTIMAL solver to help you to find the fastest solver on your study.

B) Solver for non symmetric linear systems (e.g.: Implicit schemes):

The BICGSTAB/DIAG solver seems to offer the best performances.

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.

- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

---

solveur_petsc_pipecg

Synonyms: pipecg

Pipelined Conjugate Gradient (possible reduced CPU cost during massive parallel calculation due to a single non-blocking reduction per iteration, if TRUST is built with a MPI-3 implementation)... no example in TRUST

Parameters are:

- **[seuil]** (*type:* float) corresponds to the iterative solver convergence value. The iterative solver converges when the Euclidean residue standard ||Ax-B|| is less than seuil.
- **[quiet]** (*type:* flag) is a keyword which is used to not displaying any outputs of the solver.
- **[impr]** (*type:* flag) used to request display of the Euclidean residue standard each time this iterates through the conjugated gradient (display to the standard outlet).
- **[rtol]** (*type:* float) not_set
- **[atol]** (*type:* float) not_set
- **[save_matrix_mtx_format]** (*type:* flag) not_set

1.3.33 Keywords derived from source_base

acceleration

Momentum source term to take in account the forces due to rotation or translation of a non Galilean referential R' (centre 0') into the Galilean referential R (centre 0).

Parameters are:

- **[vitesse]** (*type: field_base*) Keyword for the velocity of the referential R' into the R referential (dOO'/dt term [m.s-1]). The velocity is mandatory when you want to print the total cinetic energy into the non-mobile Galilean referential R (see Ec_dans_repere_fixe keyword).
- **[acceleration]** (*type: field_base*) Keyword for the acceleration of the referential R' into the R referential (d2OO'/dt2 term [m.s-2]). field_base is a time dependant field (eg: Champ_Fonc_t).
- **[omega]** (*type: field_base*) Keyword for a rotation of the referential R' into the R referential [rad.s-1]. field_base is a 3D time dependant field specified for example by a Champ_Fonc_t keyword. The time_field field should have 3 components even in 2D (In 2D: 0 0 omega).
- **[domegadt]** (*type: field_base*) Keyword to define the time derivative of the previous rotation [rad.s-2]. Should be zero if the rotation is constant. The time_field field should have 3 components even in 2D (In 2D: 0 0 domegadt).

- **[centre_rotation]** (*type: field_base*) Keyword to specify the centre of rotation (expressed in R' coordinates) of R' into R (if the domain rotates with the R' referential, the centre of rotation is 0'=(0,0,0)). The time_field should have 2 or 3 components according the dimension 2 or 3.
- **[option]** (*type:* string into ['terme_complet', 'coriolis_seul', 'entrainement_seul']) Keyword to specify the kind of calculation: terme_complet (default option) will calculate both the Coriolis and centrifugal forces, coriolis_seul will calculate the first one only, entrainement_seul will calculate the second one only.

---

boussinesq_concentration

Class to describe a source term that couples the movement quantity equation and constituent transport equation with the Boussinesq hypothesis.

Parameters are:

- **c0** (*type:* list of float) Reference concentration field type. The only field type currently available is Champ_Uniforme (Uniform field).

---

boussinesq_temperature

Class to describe a source term that couples the movement quantity equation and energy equation with the Boussinesq hypothesis.

Parameters are:

- **t0** (*type:* string) Reference temperature value (oC or K). It can also be a time dependant function since the 1.6.6 version.
- **[verif_boussinesq]** (*type:* int) Keyword to check (1) or not (0) the reference value in comparison with the mean value in the domain. It is set to 1 by default.

---

canal_perio

Momentum source term to maintain flow rate. The expression of the source term is:

S(t) = (2*(Q(0) - Q(t))-(Q(0)-Q(t-dt))/(coeff*dt*area)

Where:

coeff=damping coefficient

area=area of the periodic boundary

Q(t)=flow rate at time t

dt=time step

Three files will be created during calculation on a datafile named DataFile.data. The first file contains the flow rate evolution. The second file is useful for resuming a calculation with the flow rate of the previous stopped calculation, and the last one contains the pressure gradient evolution:

-DataFile_Channel_Flow_Rate_ProblemName_BoundaryName

-DataFile_Channel_Flow_Rate_repr_ProblemName_BoundaryName

-DataFile_Pressure_Gradient_ProblemName_BoundaryName

Parameters are:

- **[u_etoile]** (*type:* float) not_set
- **[coeff]** (*type:* float) Damping coefficient (optional, default value is 10).
- **[h]** (*type:* float) Half heigth of the channel.
- **bord** (*type:* string) The name of the (periodic) boundary normal to the flow direction.
- **[debit_impose]** (*type:* float) Optional option to specify the aimed flow rate Q(0). If not used, Q(0) is computed by the code after the projection phase, where velocity initial conditions are slighlty changed to verify incompressibility.

---

coriolis

Keyword for a Coriolis term in hydraulic equation. Warning: Only available in VDF.

Parameters are:

- **omega** (*type:* list of float) Value of omega.

---

correction_antal

Antal correction source term for multiphase problem

---

correction_tomiyama

Tomiyama correction source term for multiphase problem

---

darcy

Class for calculation in a porous media with source term of Darcy -nu/K*V. This keyword must be used with a permeability model. For the moment there are two models : permeability constant or Ergun's law. Darcy source term is available for quasi compressible calculation. A new keyword is aded for porosity (porosite).

Parameters are:

- **bloc** (*type: bloc_lecture*) Description.

---

dirac

Class to define a source term corresponding to a volume power release in the energy equation.

Parameters are:

- **position** (*type:* list of float) not_set
- **ch** (*type:* *field_base*) Thermal power field type. To impose a volume power on a domain sub-area, the Champ_Uniforme_Morceaux (partly_uniform_field) type must be used. Warning : The volume thermal power is expressed in W.m-3.

---

dispersion_bulles

Base class for source terms of bubble dispersion in momentum equation.

Parameters are:

- **[beta]** (*type:* float) Mutliplying factor for the output of the bubble dispersion source term.

---

dp_impose

Source term to impose a pressure difference according to the formula : DP = dp + dDP/dQ * (Q - Q0)

Parameters are:

- **aco** (*type:* string into ['{']) Opening curly bracket.
- **dp_type** (*type:* *type_perte_charge_deriv*) mass flow rate (kg/s).
- **surface** (*type:* string into ['surface']) not_set
- **bloc_surface** (*type:* *bloc_lecture*) Three syntaxes are possible for the surface definition block: For VDF and VEF: { X|Y|Z = location subzone_name } Only for VEF: { Surface surface_name }. For polymac { Surface surface_name Orientation champ_uniforme }.
- **acof** (*type:* string into ['}']) Closing curly bracket.

---

flux_interfacial

Source term of mass transfer between phases connected by the saturation object defined in saturation_xxxx

---

forchheimer

Class to add the source term of Forchheimer -Cf/sqrt(K)*V2 in the Navier-Stokes equations. We must precise a permeability model : constant or Ergun's law. Moreover we can give the constant Cf : by default its value is 1. Forchheimer source term is available also for quasi compressible calculation. A new keyword is aded for porosity (porosite).

Parameters are:

- **bloc** (*type: bloc_lecture*) Description.

---

frottement_interfacial

Source term which corresponds to the phases friction at the interface

Parameters are:

- **[a_res]** (*type:* float) void fraction at which the gas velocity is forced to approach liquid velocity (default alpha_evanescence*100)
- **[dv_min]** (*type:* float) minimal relative velocity used to linearize interfacial friction at low velocities
- **[exp_res]** (*type:* int) exponent that callibrates intensity of velocity convergence (default 2)

---

perte_charge_anisotrope

Anisotropic pressure loss.

Parameters are:

- **lambda_ | lambda_u | lambda** (*type:* string) Function for loss coefficient which may be Reynolds dependant (Ex: 64/Re).
- **lambda_ortho** (*type:* string) Function for loss coefficient in transverse direction which may be Reynolds dependant (Ex: 64/Re).
- **diam_hydr** (*type: champ_don_base*) Hydraulic diameter value.
- **direction** (*type: champ_don_base*) Field which indicates the direction of the pressure loss.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_circulaire

New pressure loss.

Parameters are:

- **lambda_ | lambda_u | lambda** (*type:* string) Function f(Re_tot, Re_long, t, x, y, z) for loss coefficient in the longitudinal direction
- **diam_hydr** (*type: champ_don_base*) Hydraulic diameter value.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

- **[lambda_ortho]** (*type:* string) function: Function f(Re_tot, Re_ortho, t, x, y, z) for loss coefficient in transverse direction
- **diam_hydr_ortho** (*type: champ_don_base*) Transverse hydraulic diameter value.
- **direction** (*type: champ_don_base*) Field which indicates the direction of the pressure loss.

---

perte_charge_directionnelle

Directional pressure loss (available in VEF and PolyMAC).

Parameters are:

- **lambda_ | lambda_u | lambda** (*type:* string) Function for loss coefficient which may be Reynolds dependant (Ex: 64/Re).
- **diam_hydr** (*type: champ_don_base*) Hydraulic diameter value.
- **direction** (*type: champ_don_base*) Field which indicates the direction of the pressure loss.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_isotrope

Isotropic pressure loss (available in VEF and PolyMAC).

Parameters are:

- **lambda_ | lambda_u | lambda** (*type:* string) Function for loss coefficient which may be Reynolds dependant (Ex: 64/Re).
- **diam_hydr** (*type: champ_don_base*) Hydraulic diameter value.
- **[sous_zone]** (*type:* string) Optional sub-area where pressure loss applies.

---

perte_charge_reguliere

Source term modelling the presence of a bundle of tubes in a flow.

Parameters are:

- **spec** (*type: spec_pdcr_base*) Description of longitudinale or transversale type.
- **zone_name | name_of_zone** (*type:* string) Name of the sub-area occupied by the tube bundle. A Sous_Zone (Sub-area) type object called zone_name should have been previously created.

---

perte_charge_singuliere

Source term that is used to model a pressure loss over a surface area (transition through a grid, sudden enlargement) defined by the faces of elements located on the intersection of a subzone named subzone_name and a X,Y, or Z plane located at X,Y or Z = location.

Parameters are:

- **dir** (*type:* string into ['kx', 'ky', 'kz', 'k']) KX, KY or KZ designate directional pressure loss coefficients for respectively X, Y or Z direction. Or in the case where you chose a target flow rate with regul. Use K for isotropic pressure loss coefficient
- **[coeff]** (*type:* float) Value (float) of friction coefficient (KX, KY, KZ).
- **[regul]** (*type:* *bloc_lecture*) option to have adjustable K with flowrate target { K0 valeur_initiale_de_k deb debit_cible eps intervalle_variation_mutiplicatif}.
- **surface** (*type:* *bloc_lecture*) Three syntaxes are possible for the surface definition block: For VDF and VEF: { X|Y|Z = location subzone_name } Only for VEF: { Surface surface_name }. For polymac { Surface surface_name Orientation champ_uniforme }

---

portance_interfaciale

Base class for source term of lift force in momentum equation.

Parameters are:

- **[beta]** (*type:* float) Multiplying factor for the bubble lift force source term.

---

puissance_thermique

Class to define a source term corresponding to a volume power release in the energy equation.

Parameters are:

- **ch** (*type:* *field_base*) Thermal power field type. To impose a volume power on a domain sub-area, the Champ_Uniforme_Morceaux (partly_uniform_field) type must be used. Warning : The volume thermal power is expressed in W.m-3 in 3D (in W.m-2 in 2D). It is a power per volume unit (in a porous media, it is a power per fluid volume unit).

---

radioactive_decay

Radioactive decay source term of the form $-lambda_i c_i$, where $0 leq i leq N$, N is the number of component of the constituent, c_i and $lambda_i$ are the concentration and the decay constant of the i-th component of the constituant.

Parameters are:

- **val** (*type:* list of float) n is the number of decay constants to read (int), and val1, val2... are the decay constants (double)

---

source_base

Basic class of source terms introduced in the equation.

---

source_constituant

Keyword to specify source rates, in [[C]/s], for each one of the nb constituents. [C] is the concentration unit.

Parameters are:

- **ch** (*type: field_base*) Field type.

---

source_dep_inco_base

Synonyms: source_dep_inco_bases

Basic class of source terms depending of inknown.

---

source_generique

to define a source term depending on some discrete fields of the problem and (or) analytic expression. It is expressed by the way of a generic field usually used for post- processing.

Parameters are:

- **champ** (*type: champ_generique_base*) the source field

---

source_pdf

Source term for Penalised Direct Forcing (PDF) method.

Parameters are:

- **aire** (*type: field_base*) volumic field: a boolean for the cell (0 or 1) indicating if the obstacle is in the cell
- **rotation** (*type: field_base*) volumic field with 9 components representing the change of basis on cells (local to global). Used for rotating cases for example.
- **[transpose_rotation]** (*type:* flag) whether to transpose the basis change matrix.
- **modele** (*type: bloc_pdf_model*) model used for the Penalized Direct Forcing
- **[interpolation]** (*type: interpolation_ibm_base*) interpolation method

---

source_pdf_base

Basic class of source_PDF terms introduced in the equation.

Parameters are:

- **aire** (*type: field_base*) volumic field: a boolean for the cell (0 or 1) indicating if the obstacle is in the cell
- **rotation** (*type: field_base*) volumic field with 9 components representing the change of basis on cells (local to global). Used for rotating cases for example.
- **[transpose_rotation]** (*type:* flag) whether to transpose the basis change matrix.
- **modele** (*type: bloc_pdf_model*) model used for the Penalized Direct Forcing
- **[interpolation]** (*type: interpolation_ibm_base*) interpolation method

---

source_qdm

Momentum source term in the Navier-Stokes equations.

Parameters are:

- **ch | champ** (*type: field_base*) Field type.

---

source_qdm_lambdaup

This source term is a dissipative term which is intended to minimise the energy associated to non-conformscales u' (responsible for spurious oscillations in some cases). The equation for these scales can be seen as: du'/dt= -lambda. u' + grad P' where -lambda. u' represents the dissipative term, with lambda = a/Delta t For Crank-Nicholson temporal scheme, recommended value for a is 2.

Remark : This method requires to define a filtering operator.

Parameters are:

- **lambda_ | lambda_u | lambda** (*type:* float) value of lambda
- **[lambda_min]** (*type:* float) value of lambda_min
- **[lambda_max]** (*type:* float) value of lambda_max
- **[ubar_umprim_cible]** (*type:* float) value of ubar_umprim_cible

---

source_th_tdivu

This term source is dedicated for any scalar (called T) transport. Coupled with upwind (amont) or muscl scheme, this term gives for final expression of convection : div(U.T)-T.div (U)=U.grad(T) This ensures, in incompressible flow when divergence free is badly resolved, to stay in a better way in the physical boundaries.

Warning: Only available in VEF discretization.

---

terme_puissance_thermique_echange_impose

Source term to impose thermal power according to formula : P = himp * (T - Text). Where T is the Trust temperature, Text is the outside temperature with which energy is exchanged via an exchange coefficient himp

Parameters are:

- **himp** (*type: field_base*) the exchange coefficient
- **text** (*type: field_base*) the outside temperature
- **[pid_controler_on_targer_power]** (*type: bloc_lecture*) PID_controler_on_targer_power bloc with parameters target_power (required), Kp, Ki and Kd (at least one of them should be provided)

---

travail_pression

Source term which corresponds to the additional pressure work term that appears when dealing with compressible multiphase fluids

---

vitesse_derive_base

Source term which corresponds to the drift-velocity between a liquid and a gas phase

---

vitesse_relative_base

Basic class for drift-velocity source term between a liquid and a gas phase

---

1.3.34 Keywords derived from sous_zone

sous_zone

Synonyms: sous_domaine

It is an object type describing a domain sub-set.

A Sous_Zone (Sub-area) type object must be associated with a Domaine type object. The Read (Lire) interpretor is used to define the items comprising the sub-area.

Caution: The Domain type object nom_domaine must have been meshed (and triangulated or tetrahedralised in VEF) prior to carrying out the Associate (Associer) nom_sous_zone nom_domaine instruction; this instruction must always be preceded by the read instruction.

Parameters are:

- **[restriction]** (*type:* string) The elements of the sub-area nom_sous_zone must be included into the other sub-area named nom_sous_zone2. This keyword should be used first in the Read keyword.
- **[rectangle]** (*type: bloc_origine_cotes*) The sub-area will include all the domain elements whose centre of gravity is within the Rectangle (in dimension 2).

- **[segment]** (*type: bloc_origine_cotes*) not_set
- **[boite | box]** (*type: bloc_origine_cotes*) The sub-area will include all the domain elements whose centre of gravity is within the Box (in dimension 3).
- **[liste]** (*type:* list of int) The sub-area will include n domain items, numbers No. 1 No. i No. n.
- **[fichier | filename]** (*type:* string) The sub-area is read into the file filename.
- **[intervalle]** (*type: deuxentiers*) The sub-area will include domain items whose number is between n1 and n2 (where n1<=n2).
- **[polynomes]** (*type: bloc_lecture*) A REPRENDRE
- **[couronne]** (*type: bloc_couronne*) In 2D case, to create a couronne.
- **[tube]** (*type: bloc_tube*) In 3D case, to create a tube.
- **[fonction_sous_zone | fonction_sous_domaine]** (*type:* string) Keyword to build a sub-area with the the elements included into the area defined by fonction>0.
- **[union | union_with]** (*type:* string) The elements of the sub-area nom_sous_zone3 will be added to the sub-area nom_sous_zone. This keyword should be used last in the Read keyword.

1.3.35 Keywords derived from turbulence_paroi_base

negligeable

Keyword to suppress the calculation of a law of the wall with a turbulence model. The wall stress is directly calculated with the derivative of the velocity, in the direction perpendicular to the wall (tau_tan /rho= nu dU/dy).

Warning: This keyword is not available for k-epsilon models. In that case you must choose a wall law.

---

turbulence_paroi_base

Basic class for wall laws for Navier-Stokes equations.

1.3.36 Keywords derived from turbulence_paroi_scalaire_base

negligeable_scalaire

Keyword to suppress the calculation of a law of the wall with a turbulence model for thermohydraulic problems. The wall stress is directly calculated with the derivative of the velocity, in the direction perpendicular to the wall.

---

turbulence_paroi_scalaire_base

Basic class for wall laws for energy equation.

CHAPTER

TWO

REFERENCES

BIBLIOGRAPHY

[Aav02] Ivar Aavatsmark. An introduction to multipoint flux approximations for quadrilateral grids. *Computational Geosciences*, 6:405–432, 2002.

[AM08] L Agelas and R Masson. Convergence of the finite volume mpfa o scheme for heterogeneous anisotropic diffusion problems on general meshes. *Acad. Sci. Paris, Ser. I*, 2008.

[BNM14] Mantulal Basumatary, Ganesh Natarajan, and Subhash C Mishra. Defect correction based velocity reconstruction for physically consistent simulations of non-newtonian flows on unstructured grids. *Journal of Computational Physics*, 272:227–244, 2014.

[BAK18] R Beltman, M J H Anthonissen, and B Koren. Conservative polytopal mimetic discretization of the incompressible navier–stokes equations. *Journal of Computational and Applied Mathematics*, 340:443–473, 2018.

[Bon14] Jerome Bonelle. *Compatible Discrete Operator schemes on polyhedral meshes for elliptic and Stokes equations*. PhD thesis, Université Paris-Est, 2014.

[Bre14] S Brenner. Forty years of the crouzeix-raviart element. *Wiley Online Library*, 2014.

[CGPS07] Larry S Caretto, AD Gosman, Suhas V Patankar, and DB Spalding. Two calculation procedures for steady, three-dimensional flows with recirculation. In *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics: Vol. II Problems of Fluid Mechanics*, 60–68. Springer, 2007.

[Cho67] Alexandre Joel Chorin. The numerical solution of the navier-stokes equationsfor an incompressible fluid. *Bulletin of the American Mathematical Society*, pages 928–931, 1967.

[CKS12] B Cockburn, G E Karniadakis, and C W Shu. *Discontinuous Galerkin Methods: Theory, Computation and Applications*. Springer, 2012.

[CST10] Lorenzo Codecasa, Ruben Specogna, and Francesco Trevisan. A new set of basis functions for the discrete geometric approach. *Journal of Computational Physics*, 229(19):7401–7410, 2010.

[CR73] M Crouzeix and P-A Raviart. Conforming and nonconforming finite element methods for solving the stationary stokes equations i. *Revue française d'automatique, informatique, recherche opérationnelle. Mathématiques*, 7(3):33–75, 1973.

[DEGH10] J Droniou, R Eymard, T Gallouët, and R Herbin. A unified approach to mimetic finite difference, hybrid finite volume and mixed finite volume methods. *Mathematical Models and Methods in Applied Sciences*, 20:265–295, 2010.

[Dro14] Jerome Droniou. Finite volume schemes for diffusion equations: introduction to and review of modern methods. *Mathematical Models and Methods in Applied Sciences*, 24(08):1575–1619, 2014.

[ER98] M G Edwards and C F Rogers. Finite volume discretization with imposed flux continuity for the general tensor pressure equation. *Computational Geosciences*, 2:259–290, 1998.

[Emo92] Philippe Emonot. *Méthodes de volumes éléments finis : applications aux équations de Navier Stokes et résultats de convergence*. PhD thesis, Université Claude Bernard, 1992.

[EDP12] A Ern and D Di Pietro. *Mathematical Aspects of Discontinuous Galerkin Methods*. Springer, 2012.

[EGH07] Robert Eymard, Thierry Gallouet, and Raphaele Herbin. A new finite volume scheme for anisotropic diffusion problems on general grids: convergence analysis. *Comptes rendus. Mathematique*, 344(6):403–406, 2007.

[EGH10] Robert Eymard, Thierry Gallouet, and Raphaele Herbin. Discretization of heterogeneous and anisotropic diffusion problems on general nonconforming meshes sushi: a scheme using stabilization and hybrid interfaces. *IMA Journal of Numerical Analysis*, 30(4):1009–1043, 2010.

[For06] Thomas Fortin. *Une méthode éléments finis à décomposition L2 d'ordre élevé motivée par la simulation d'écoulement diphasique bas Mach*. PhD thesis, Univ. Pierre et Marie Curie, 2006.

[GG22] Antoine Gerschenfeld and Yannick Grosse. Development of a robust multiphase flow solver on general meshes; application to sodium boiling at the subchannel scale. In *NURETH 2022*. 2022.

[God79] K Goda. A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows. *Journal of Computational Physics*, 30(1):76–95, 1979.

[GMS06] J. Guermond, P. Minev, and J. Shen. An overview of projection methods for incompressible flows. *Computational Methods in Applied Mechanics and Engineering*, 195:6011–6045, 2006.

[Gue96] J.L. Guermond. Some implementations of projection methods for navier-stokes equations. *M2AN*, 30(5):374–378, 1996.

[HW+65] Francis H Harlow, J Eddie Welch, and others. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 1965.

[Hei03] Sébastien Heib. *Nouvelles discrétisations non structurées pour des écoulements de fluides à incompressibilité renforcée*. PhD thesis, Université Paris 6, 2003.

[HW07] J S Hesthaven and T Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer, 2007.

[Ish75] Mamoru Ishii. *Thermo-fluid dynamic theory of two-phase flow*. Eyrolles-Collection de la Direction des Etudes et Recherches EDF, 1975.

[Iss86] R. I. Issa. The computational of compressible and incompressible recirculation flows by non-iterative implicite scheme. *Journal of Computational Physics*, 62:66–82, 1986.

[JCS23] Erell Jamelot, Patrick Ciarlet, and Stefan Sauter. Stability of the p1nc element. In *ENUMATH 2023 - The European Conference on Numerical Mathematics and Advanced Applications*. Lisbon, Portugal, 2023.

[JJA07] D. S. Jang, R. Jetli, and S. Acharya. Comparison of the piso, simpler, and simplec for the treatment of the pressure-velocity coupling in steady flow problems. *Numerical Heat Transfer*, 2007.

[LP05a] Christophe Le Potier. Finite volume monotone scheme for highly anisotropic diffusion operators on unstructured triangular meshes. *Comptes Rendus de l'Académie des Sciences*, 2005.

[LP05b] Christophe Le Potier. Schéma volumes finis pour des opérateurs de diffusion fortement anisotropes sur des maillages non structurés. *Comptes Rendus Mathematique Acad. Sci. Paris*, 2005.

[LP17] Christophe Le Potier. *Construction et developpement de nouveaux schemas pour des problemes elliptiques ou paraboliques*. PhD thesis, Université Paris-Est, 2017. Habilitation à diriger des recherches.

[LM89] C. Liu and S. McCormick. The finite volume-element method (fve) for planar cavity flow. In D. L. Dwoyer, M. Y. Hussaini, and R. G. Voigt, editors, *11th International Conference on Numerical Methods in Fluid Dynamics*, 374–378. Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[Mil20] Riccardo Milani. *Compatible Discrete Operator schemes for the unsteady incompressible Navier–Stokes equations*. PhD thesis, Université Paris-Est, 2020.

[Pat80] S Patankar. *Numerical Heat Transfer and Fluid Flows*. Hemishhere Publishing Corporation, 1980.

[PS72] S. Patankar and D. Spalding. A calculation procedure for heat, mass and momentum transfer in trhee dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15:1787–1806, 1972.

[Tem69] Régine Témam. Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires. *Archive for Rational Mechanics*, pages 377–385, 1969.

[Aav02] Ivar Aavatsmark. An introduction to multipoint flux approximations for quadrilateral grids. *Computational Geosciences*, 6:405–432, 2002.

[AM08] L Agelas and R Masson. Convergence of the finite volume mpfa o scheme for heterogeneous anisotropic diffusion problems on general meshes. *Acad. Sci. Paris, Ser. I*, 2008.

[ADPM08] Leo Agelas, Daniele Antonio Di Pietro, and Roland Masson. A symmetric and coercive finite volume scheme for multiphase porous media flow with applications in the oil industry. *Finite volumes for complex applications V*, pages 35–52, 2008.

[BNM14] Mantulal Basumatary, Ganesh Natarajan, and Subhash C Mishra. Defect correction based velocity reconstruction for physically consistent simulations of non-newtonian flows on unstructured grids. *Journal of Computational Physics*, 272:227–244, 2014.

[BAK18] R Beltman, M J H Anthonissen, and B Koren. Conservative polytopal mimetic discretization of the incompressible navier–stokes equations. *Journal of Computational and Applied Mathematics*, 340:443–473, 2018.

[BE79] Michel Bercovier and Michael Engelman. A finite element for the numerical solution of viscous incompressible flows. *Journal of Computational Physics*, 30(2):181–201, 1979.

[Bon14] Jerome Bonelle. *Compatible Discrete Operator schemes on polyhedral meshes for elliptic and Stokes equations*. PhD thesis, Université Paris-Est, 2014.

[BO17] Franck Boyer and Pascal Omnes. Benchmark proposal for the fvca8 conference: finite volume methods for the stokes and navier–stokes equations. *Finite Volumes for Complex Applications VIII-Methods and Theoretical Aspects: FVCA 8, Lille, France, June 2017*, pages 59–71, 2017.

[Bre14] S Brenner. Forty years of the crouzeix-raviart element. *Wiley Online Library*, 2014.

[CGPS07] Larry S Caretto, AD Gosman, Suhas V Patankar, and DB Spalding. Two calculation procedures for steady, three-dimensional flows with recirculation. In *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics: Vol. II Problems of Fluid Mechanics*, 60–68. Springer, 2007.

[CJLPP23] Eric Chenier, Erell Jamelot, Christophe Le Potier, and Andrew Peitavy. Improved crouzeix-raviart scheme for the stokes problem. *Finite Volumes for Complex Applications X—Volume 1, Elliptic and Parabolic Problems*, pages 245–253, 2023.

[Cho67] Alexandre Joel Chorin. The numerical solution of the navier-stokes equationsfor an incompressible fluid. *Bulletin of the American Mathematical Society*, pages 928–931, 1967.

[CKS12] B Cockburn, G E Karniadakis, and C W Shu. *Discontinuous Galerkin Methods: Theory, Computation and Applications*. Springer, 2012.

[CST10] Lorenzo Codecasa, Ruben Specogna, and Francesco Trevisan. A new set of basis functions for the discrete geometric approach. *Journal of Computational Physics*, 229(19):7401–7410, 2010.

[CR73] M Crouzeix and P-A Raviart. Conforming and nonconforming finite element methods for solving the stationary stokes equations i. *Revue française d'automatique, informatique, recherche opérationnelle. Mathématiques*, 7(3):33–75, 1973.

[DEGH10] J Droniou, R Eymard, T Gallouët, and R Herbin. A unified approach to mimetic finite difference, hybrid finite volume and mixed finite volume methods. *Mathematical Models and Methods in Applied Sciences*, 20:265–295, 2010.

[Dro14] Jerome Droniou. Finite volume schemes for diffusion equations: introduction to and review of modern methods. *Mathematical Models and Methods in Applied Sciences*, 24(08):1575–1619, 2014.

[DEG+18] Jerome Droniou, Robert Eymard, Thierry Gallouet, Cindy Guichard, and Raphaele Herbin. *The gradient discretisation method*. Volume 82. Springer, 2018.

[ER98] M G Edwards and C F Rogers. Finite volume discretization with imposed flux continuity for the general tensor pressure equation. *Computational Geosciences*, 2:259–290, 1998.

[Emo92] Philippe Emonot. *Méthodes de volumes éléments finis : applications aux équations de Navier Stokes et résultats de convergence*. PhD thesis, Université Claude Bernard, 1992.

[EDP12] A Ern and D Di Pietro. *Mathematical Aspects of Discontinuous Galerkin Methods*. Springer, 2012.

[EGH07] Robert Eymard, Thierry Gallouet, and Raphaele Herbin. A new finite volume scheme for anisotropic diffusion problems on general grids: convergence analysis. *Comptes rendus. Mathematique*, 344(6):403–406, 2007.

[EGH10] Robert Eymard, Thierry Gallouet, and Raphaele Herbin. Discretization of heterogeneous and anisotropic diffusion problems on general nonconforming meshes sushi: a scheme using stabilization and hybrid interfaces. *IMA Journal of Numerical Analysis*, 30(4):1009–1043, 2010.

[For06] Thomas Fortin. *Une méthode éléments finis à décomposition L2 d'ordre élevé motivée par la simulation d'écoulement diphasique bas Mach*. PhD thesis, Univ. Pierre et Marie Curie, 2006.

[GG22] Antoine Gerschenfeld and Yannick Grosse. Development of a robust multiphase flow solver on general meshes; application to sodium boiling at the subchannel scale. In *NURETH 2022*. 2022.

[God79] K Goda. A multistep technique with implicit difference schemes for calculating two- or three-dimensional cavity flows. *Journal of Computational Physics*, 30(1):76–95, 1979.

[GMS06] J. Guermond, P. Minev, and J. Shen. An overview of projection methods for incompressible flows. *Computational Methods in Applied Mechanics and Engineering*, 195:6011–6045, 2006.

[Gue96] J.L. Guermond. Some implementations of projection methods for navier-stokes equations. *M2AN*, 30(5):374–378, 1996.

[HA68] Francis H Harlow and Anthony Amsden. Numerical calculation of almost incompressible flow. *Journal of Computational Physics*, 3:80–93, 1968.

[HA71] Francis H Harlow and Anthony Amsden. A numerical fluid dynamics calculation method for all flow speeds. *Journal of Computational Physics*, 8(2):197–213, 1971.

[HW+65] Francis H Harlow, J Eddie Welch, and others. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 1965.

[Hei03] Sébastien Heib. *Nouvelles discrétisations non structurées pour des écoulements de fluides à incompressibilité renforcée*. PhD thesis, Université Paris 6, 2003.

[HW07] J S Hesthaven and T Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer, 2007.

[Ish75] Mamoru Ishii. *Thermo-fluid dynamic theory of two-phase flow*. Eyrolles-Collection de la Direction des Etudes et Recherches EDF, 1975.

[Iss86] R. I. Issa. The computational of compressible and incompressible recirculation flows by non-iterative implicite scheme. *Journal of Computational Physics*, 62:66–82, 1986.

[JCS23] Erell Jamelot, Patrick Ciarlet, and Stefan Sauter. Stability of the p1nc element. In *ENUMATH 2023 - The European Conference on Numerical Mathematics and Advanced Applications*. Lisbon, Portugal, 2023.

[JJA07] D. S. Jang, R. Jetli, and S. Acharya. Comparison of the piso, simpler, and simplec for the treatment of the pressure-velocity coupling in steady flow problems. *Numerical Heat Transfer*, 2007.

[LP05a] Christophe Le Potier. Finite volume monotone scheme for highly anisotropic diffusion operators on unstructured triangular meshes. *Comptes Rendus de l'Académie des Sciences*, 2005.

[LP05b] Christophe Le Potier. Schéma volumes finis pour des opérateurs de diffusion fortement anisotropes sur des maillages non structurés. *Comptes Rendus Mathematique Acad. Sci. Paris*, 2005.

[LP17] Christophe Le Potier. *Construction et developpement de nouveaux schemas pour des problemes elliptiques ou paraboliques*. PhD thesis, Université Paris-Est, 2017. Habilitation à diriger des recherches.

[LMS14] Konstantin Lipnikov, Gianmarco Manzini, and Mikhail Shashkov. Mimetic finite difference method. *Journal of Computational Physics*, 257:1163–1227, 2014.

[LM05] C Liu and S McCormick. The finite volume-element method (fve) for planar cavity flow. In *11th International Conference on Numerical Methods in Fluid Dynamics*, 374–378. Springer, 2005.

[LM89] C. Liu and S. McCormick. The finite volume-element method (fve) for planar cavity flow. In D. L. Dwoyer, M. Y. Hussaini, and R. G. Voigt, editors, *11th International Conference on Numerical Methods in Fluid Dynamics*, 374–378. Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[Mil20] Riccardo Milani. *Compatible Discrete Operator schemes for the unsteady incompressible Navier–Stokes equations*. PhD thesis, Université Paris-Est, 2020.

[Pat80] S Patankar. *Numerical Heat Transfer and Fluid Flows*. Hemishhere Publishing Corporation, 1980.

[PS72] S. Patankar and D. Spalding. A calculation procedure for heat, mass and momentum transfer in trhee dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15:1787–1806, 1972.

[Per00] Blair Perot. Conservation properties of unstructured staggered mesh schemes. *Journal of Computational Physics*, 159(1):58–89, 2000.

[Tem69] Régine Témam. Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires. *Archive for Rational Mechanics*, pages 377–385, 1969.