Tutoriel de réalisation d'une application BALTIK basée sur le logiciel TRUST

May 6, 2022

1 résumé

Ce tutoriel a comme objectif de décrire la démarche à suivre pour la réalisatiion d'une application BALTIK sur la base du logiciel TRUST. Pour ce faire, on choisit à titre d'exercice une équation de convection-diffusion comme problème mathématique à résoudre. Dans cette équation, la vitesse de convection et le coefficient de diffusion sont des données utilisateur.

On décrit en particulier les spécifications des développements à réaliser et on donne quelques recommandations pour résoudre cette équation de convection-diffusion.

Les développements sont ensuite décrits et réalisés dans un environnement proposé par le logiciel TRUST appelé application BALTIK. La mise en place de cet environnement est également décrit.

Deux tests de vérification des développements et de non-régression sont enfin réalisés en utilisant les outils proposés par le logiciel TRUST (genererCourbes, make check_optim).

2 Introduction

Ce tutoriel a comme objectif de décrire la démarche à suivre pour la réalisation d'une application `BALTIK` sur la base du logiciel `TRUST`. Pour ce faire, on choisit à titre d'exercice l'équation de convection-diffusion (1) comme problème mathématique à résoudre:

$$\frac{\partial c}{\partial t} + \nabla \cdot (\overrightarrow{V} c) + \nabla \cdot (D \nabla c) = 0 \text{ sur } \Omega = [a, b] \times [c, d] \quad (1)$$

conditions aux limites périodiques

$c(\overrightarrow{X}, t)$: variable d'intérêt (la concentration d'une espèce, la température,...),
$\overrightarrow{V}(\overrightarrow{X}, t)$: vitesse de convection de la variable c (donnée utilisateur),
$D(\overrightarrow{X}, t)$: coefficient de diffusion de la variable c (donnée utilisateur),
a, b, c et d : les bornes du domaine de calcul en deux dimensions.

La discrétisation en espace et en temps de l'équation (1) sera réalisée avec les schémas numériques disponibles dans le logiciel `TRUST` [1].

Dans la section 3, nous décrivons les spécifications des développements à réaliser et nous donnons quelques recommandations pour résoudre l'équation (1).

Les développements spécifiés dans la section 3 sont décrits dans la section 4. Ces développements seront réalisés dans un environnement proposé par le logiciel `TRUST` appelé application `BALTIK`. La mise en place de cet environnement sera décrit dans cette même section 4.

La section 5 sera consacrée à la réalisés deux tests de vérification des développements et de non-régression en utilisant les outils proposés par le logiciel `TRUST` (`genererCourbes`, `make check_optim`).

3 Spécification

3.1 Notions

- Notion de problème : le rôle d'un problème est la résolution sur un domaine des équations qui composent le problème. Un schéma de discrétisation en temps ainsi qu'un schéma de discrétisation en espace lui sont associés.

- Notion d'équation : le rôle d'une équation est le calcul d'un ou plusieurs champs compte tenu des choix suivants:
  - un schéma de discrétisation en temps,
  - un schéma de discrétisation en espace,

  - des conditions aux limites,
  - des termes sources et des opérateurs.

  Une équation est portée par un problème et possède une référence qui permet de remonter au problème qui la porte.

- Notion d'opérateur : les opérateurs sont des parties d'une équation. Parmi les opérateurs les plus utilisés, on cite les opérateurs de convection et de diffusion.
- Notion de milieu : Description du milieu fluide ou solide qu'on modélise. Le Milieu est associé au problème et aux équations. Parmi les propriétés qui caractérisent le milieu, on a:
  - la masse volumique,
  - la diffusivité,
  - la conductivité,
  - la capacité calorifique,
  - la variation de la masse volumique en fonction de la température (dilatabilité).

Dans notre cas, le problème qu'on souhaite résoudre comprend une seule équation (1). Cette équation est composée de:

- un terme instationnaire $\frac{\partial c}{\partial t}$,
- un opérateur de convection $\nabla \cdot (\overrightarrow{V} c)$,
- un opérateur de diffusion $\nabla \cdot (D \nabla c)$,
- un terme source ou puit $f = 0$.

3.2 Jeu de données - analyse des besoins

Dans le cadre des développements des applications basées sur le logiciel `TRUST`, il est recommandé de commencer les spécifications par écrire le jeu de données. La recommandation précédente est liée au fait que les mots clés du jeu données du logiciel `TRUST` correspondent à des objets (classes) C++ et donc commencer par le jeu de données sert, d'une part, à se mettre dans la position d'un utilisateur et, d'autre part, à définir les objets (problème, équation(s), ...) à spécifier et à développer.

Nous décrivons ci-dessous les parties de jeu de données spécifiques à notre problème:

- **Déclaration du problème**

  On notera ce nouveau problème `Probleme_Convection_Diffusion`

```
Probleme_Convection_Diffusion pb
```

déclaration du problème

- **Déclaration de l'équation de convection-diffusion**

  On notera cette nouvelle équation `Convection_Diffusion`. Cette équation est composée de deux opérateurs (convection et diffusion), des conditions initiales et des conditions aux limites

```
Convection_Diffusion
{
  diffusion { }
  convection { amont }
  initial_conditions
  {
   Concentration champ_fonc_xyz dom 1 (x-0.2)^2+(y-0.2)^2<(0.1)
   ^2
  }
  boundary_conditions
  {
   Gauche  periodique
   Bas     periodique
  }
}
```

déclaration de l'équation

- **Données utilisateur**

  Pour l'équation (1), la vitesse de convection $\overrightarrow{V}(\overrightarrow{X},t)$ et le coefficient de diffusion $D(\overrightarrow{X},t)$ sont donnés par l'utilisateur. Dans les pratiques des développements `TRUST`, ces paramètres doivent être associés à un objet (équation, problème, ...) du jeu de données. on notera ces paramètres `coefficient_diffusion` et `vitesse_convection`.

```
coefficient_diffusion  Champ_Uniforme 1 0.001
vitesse_convection Champ_Fonc_txyz dom 2 x*t y*t
```

paramètres d'entrées

- **Post-traitements**

  Il nous semble intéressant de post-traiter la variable d'intérêt c, la vitesse de convection $\overrightarrow{V}(\overrightarrow{X},t)$ et le coefficient de diffusion $D(\overrightarrow{X},t)$. Le bloc de post-traitement de ces variables se présente sous la forme suivante:

```
Post_processing
{
  format lata
  fields dt_post 1.e-7
  {
    Concentration elem
    vitesse_convection elem
    coefficient_diffusion elem
  }
}
```

déclaration du champs à post-traiter

3.3 Fonctionnalités TRUST

Il est fortement recommandé d'utiliser au maximum les fonctionnalités existantes dans le logiciel TRUST et d'éviter de dupliquer les lignes de codes. Ceci permet, d'une part, de minimiser le temps de développement de l'application et, d'autre part, de profiter des fonctionnalités couvertes par les tests du logiciel TRUST.

L'identification des fonctionnalités peut se faire via la documentation doxygen du logiciel TRUST. Nous faisons ci-dessous cet exercice d'identification par rapport à nos besoins (3.2).

- **problème**

  En analysant le graphe doxygen de la classe `Probleme_base` (cf. figure 1), aucun problème nous parait proche de nos besoins (3.2).

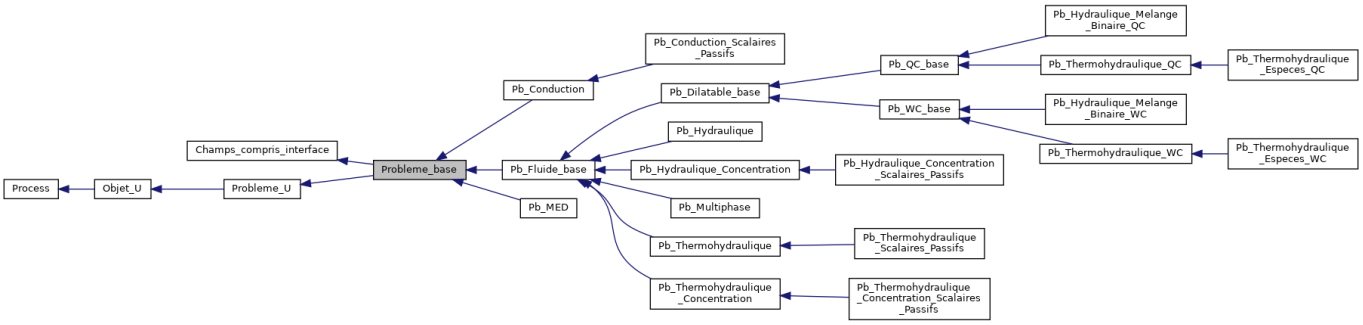

Figure 1: classe Probleme_base

- **équation de convection-diffusion**

En analysant le graphe doxygen de la classe `Equation_base` (cf. figure 2), on constante qu'il existe une équation de type convection-diffusion nommée `Convection_Diffusion_std` (cf. figure 3). C'est une classe de base pour l'équation de transport d'un scalaire en régime laminaire et elle porte une référence au champ de la vitesse transportante [2]. Cette classe d'équation est abstraite pure et donc non instanciable. L'implémentation de cette classe n'est pas complète et elle sert de base à d'autres classes dérivées (héritées).

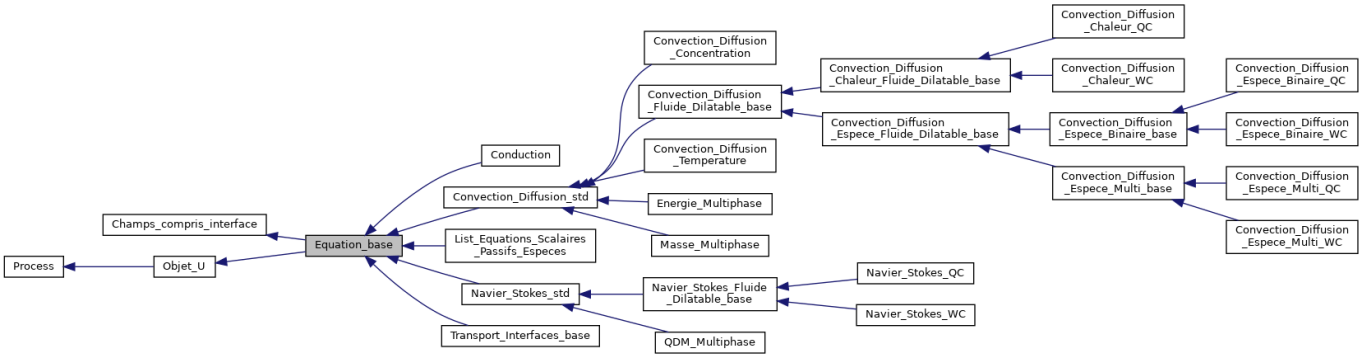

Figure 2: classe `Equation_base`

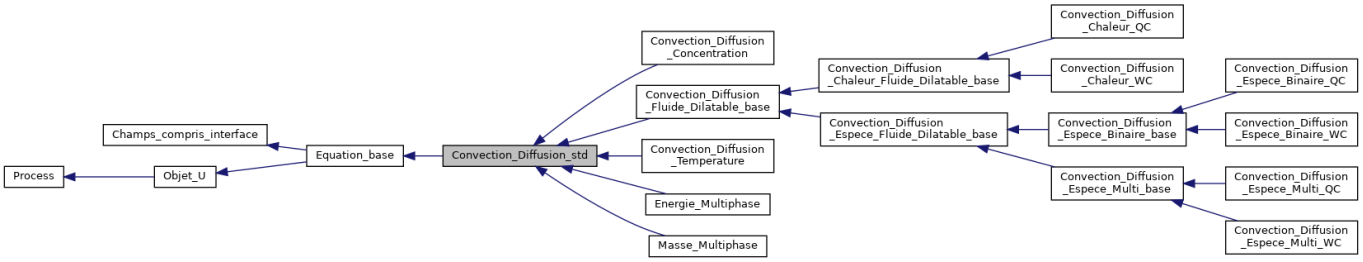

Figure 3: classe `Convection_Diffusion_std`

En se basant sur le graphe doxygen de la classe `Convection_Diffusion_Concentration` (cf. figure 4), la classe `Convection_Diffusion_Concentration` fille (dérivée) de la classe `Convection_Diffusion_std` parait très proche de l'équation (1). Cette classe est un cas particulier de `Convection_Diffusion_std` pour le transport d'un ou plusieurs constituants [2].

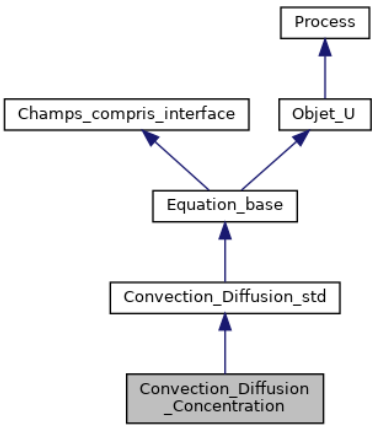

Figure 4: classe Convection_Diffusion_Concentration

Les spécificités de l'équation `Convection_Diffusion_Concentration` sont [2]:

- coefficient de diffusion [3]: ce coefficient constitue une propriété d'un constituant associé à cette équation. Cette propriété est donnée par l'utilisateur dans le jeu de données de la manière suivante:

```
Constituant constituant
Read constituant
{
  coefficient_diffusion  Champ_Uniforme 1 0.001
}
```

déclaration du constituant standard `TRUST`

- vitesse de convection [2]: cette vitesse est obtenue par la méthode `vitesse_pour_transport` qui récupéré cette vitesse à partir de l'équation du problème qui la porte. En effet, dans une utilisation standard du logiciel `TRUST`, cette vitesse de convection est obtenue par l'équation de l'hydraulique Navier-Stokes.

3.4 Fonctionnalités spécifiques

L'analyse des fonctionnalités du logiciel `TRUST` dans la section précédente a permis d'identifier une équation `Convection_Diffusion_Concentration` très proche de l'équation (1) avec un coefficient de diffusion donné par l'utilisateur à travers un constituant. Nous faisons donc le choix d'utiliser cette équation comme base pour développer une nouvelle équation qu'on notera `Convection_Diffusion`. A cette équation, on associe un nouveau constituant `Constituant_Avec_Vitesse` basé sur l'ancien en lui rajoutant comme propriété la vitesse de convection.

La nouvelle classe de constituant `Constituant_Avec_Vitesse` sera appelée dans le jeu de données de la manière suivante:

```
Constituant_Avec_Vitesse constituant
Read constituant
{
  coefficient_diffusion  Champ_Uniforme 1 0.001
  vitesse_convection Champ_Fonc_txyz dom 2 x*t y*t
}
```

déclaration du constituant spécifique à l'application BALTIK

4 Développements

4.1 Environnement de développement : Application BALTIK

Un environnement de développement Application BALTIK est mis à disposition pour les applications reposant, soit sur le logiciel TRUST soit, sur son noyau numérique « Kernel » soit sur une autre application BALTIK. Cet environnement permet une gestion de compilation de ces applications (sources spécifiques) ainsi que les sources modifiées et/ou rajoutées du logiciel TRUST. Dans ce cadre, une application est constituée :

- du noyau numérique « Kernel » ou du logiciel TRUST ou d'une autre application BALTIK,
- des sources spécifiques à l'application.

En pratique, les sources spécifiques à l'application sont rassemblées dans un répertoire de travail appelé src[1], celui-ci est compilé suivant le processus classique de compilation du logiciel TRUST. On peut inclure à cette application des sources du « Kernel » ou de TRUST que l'on souhaite modifier par rapport à leurs versions standards.

4.1.1 Création et configuration de l'application BALTIK

La création d'une application BALTIK consiste à créer un répertoire qui prend le nom de cette application (convection_diffusion par exemple).

```
$ mkdir convection_diffusion
```

La mise en place d'une application BALTIK se traduit par la création dans le répertoire convection_diffusion d'un fichier de configuration project.cfg.

```
# description section (required)
[description]
name        : convection_diffusion     # name of the project    (
   required)
```

[1] L'emplacement du répertoire src dans l'arborescence de l'application BALTIK est indiqué dans la section 4.1.1

```
author     : MEKKAS                    # author of the project  (
   optional)
executable : convection_diffusion_exe # name of the executable (
   optional)
```

project.cfg

Dans ce fichier de configuration, seul le nom de l'application est obligatoire. En revanche, il existe un certain nombre de paramètres optionnels tels que l'auteur, le nom de l'exécutable à générer, etc.

Pour configurer notre application `BALTIK`, il faut exécuter les instructions sivantes:

- initialisation de l'environnement du logiciel:

```
$ cd convection_diffusion
$ mkdir src

src est le répertoire contenant soit des sources modifiées du
   logiciel
TRUST soit du Kernel TRUST soit des sources supplémentaires
   spécifiques
à notre application.
```

- configuration de l'application `BALTIK`:

```
$ source $TRUST_ROOT/env_TRUST.sh
$ cd convection_diffusion
$ baltik_build_configure
La variable d'environnement $TRUST_ROOT est le chemin où le
   logiciel
TRUST est installé.
La première instruction sert à initialiser l'environnement
   TRUST.
La deuxième instruction génère un fichier configure. Il est
   important de
noter que cette instruction doit être exécuter dans le
   répertoire
convection_diffusion.
```

4.1.2 Compilation de l'application BALTIK

La compilation de notre application BALTIK est assez standard. En effet, la génération du fichier Makefile se fait via l'exécution dans le répertoire convection_diffusion de l'instruction suivante:

```
$ cd convection_diffusion
$ ./configure

Il est important de noter que lors du rajout de nouvelles sources
   dans
le répertoire src, il est impératif de re-exécuter cette
   instruction
pour mettre à jour le fichier Makefile.
```

Une fois le fichier Makefile généré, deux modes de compilation de l'application BALTIK sont possibles:

- compilation en mode debug:

```
$ cd convection_diffusion
$ make debug
```

- compilation en mode optimisé:

```
$ cd convection_diffusion
$ make optim
```

4.2 Développements

4.3 Classe constituant Constituant_Avec_Vitesse

Comme spécifié dans la section 3, les paramètres utilisateurs vitesse de convection et coefficient de diffusion sont attribués à la nouvelle classe de constituant Constituant_Avec_Vitesse basée (héritage) sur la classe Constituant du logiciel TRUST. La spécificité de cette nouvelle classe est que cette dernière possède un champ de vitesse de convection du constituant.

Le traitement de cet attribut supplémentaire nécessite les actions suivantes:

1. le rajout d'attributs:
   - C: il contiendra le contenu du vitesse_convection du jeu de données (cf. section 3).
   - vitesse_transport: il contiendra la vitesse de convection discrétisée.

2. la surcharge des méthodes suivantes[2]:

   - `initialiser`: évaluer la vitesse de convection en espace au temps initial.
   - `set_param`: affecter le contenu de `vitesse_convection` du jeu de données (cf. section 3) à l'attribut `C` de cette classe.
   - `mettre_a_jour`: évaluer la vitesse de convection en espace au temps courant.
   - `discretiser`: discrétise la vitesse de convection à partir de la vitesse de convection donné par l'utilisateur.

3. le développement de nouvelles méthodes:

   - `vit_convection_constituant`: renvoie l'attribut contenant la vitesse de convection non discrète.
   - `vitesse_pour_transport`: renvoie la vitesse de convection discrétisée sur les faces du maillage.

4. le développement des méthodes `printOn` et `readOn`: la classe `Constituant_Avec_Vitesse` est utilisable dans le jeu de données et donc instantiable. Pour déclarer ce type de classe, il suffit d'utiliser la macro `Declare_instanciable`. Dans cette macro, on déclare les entêtes des méthodes `printOn` et `readOn` qui sont nécessaires à la lecture des objets sur les flux d'entrée (jeu de données) et l'écriture des objets sur les flux de sortie.

4.3.1 Fichier entête

```
#ifndef Constituant_Avec_Vitesse_included
#define Constituant_Avec_Vitesse_included

#include <Constituant.h>
#include <Champ_Inc.h>

class Constituant_Avec_Vitesse : public Constituant
{
  Declare_instanciable(Constituant_Avec_Vitesse);
public :
  int   initialiser(const double temps) override;
  void  set_param(Param& param) override;
  void  mettre_a_jour(double temps) override;
  const Champ_Don& vit_convection_constituant() const;
  const Champ_base& vitesse_pour_transport() ;
  void discretiser(const Probleme_base& pb, const
   Discretisation_base& dis) override;
protected :
```

[2]Seules les fonctionnalités spécifiques de la nouvelle classe `Constituant_Avec_Vitesse` sont décrites.

```
  Champ_Don C;
private :
  Champ_Inc vitesse_transport;
};
#endif
```

Constituant_Avec_Vitesse.h

4.3.2 Fichier source

Il est à noter que dans le fichier source, qui contient le corps des méthodes définies dans le fichier entête, la macro `Implemente_instanciable` sert, d'une part, à définir le mot clé utilisable dans le jeu données pour représenter la classe `Constituant_Avec_Vitesse` et, d'autre part, la classe mère de de `Constituant_Avec_Vitesse`. Dans notre cas, on choisit le nom de la classe `Constituant_Avec_Vitesse` comme mot clé dans le jeu de données.

```
#include <Constituant_Avec_Vitesse.h>
#include <Probleme_base.h>
#include <Discretisation_base.h>
#include <Equation_base.h>
#include <Param.h>
#include <Schema_Temps_base.h>

Implemente_instanciable(Constituant_Avec_Vitesse,"
   Constituant_Avec_Vitesse",Constituant);
// XD constituant_avec_vitesse constituant constituant_avec_vitesse
    -1 not_set
Sortie& Constituant_Avec_Vitesse::printOn(Sortie& os) const
{
  return os;
}

Entree& Constituant_Avec_Vitesse::readOn(Entree& is)
{
  Constituant::readOn(is);
  return is;
}

void Constituant_Avec_Vitesse::set_param(Param& param)
{
  Constituant::set_param(param);
  param.ajouter("vitesse_convection",&C,Param::REQUIRED); //
   XD_ADD_P field_base not_set
}

```

```
void Constituant_Avec_Vitesse::discretiser(const Probleme_base& pb,
                                           const Discretisation_base&
    dis)
{
  const Zone_dis_base& zone_dis=pb.equation(0).zone_dis();
  dis.nommer_completer_champ_physique(zone_dis,"
   coefficient_diffusion","m2/s",D.valeur(),pb);
  dis.nommer_completer_champ_physique(zone_dis,"
   vitesse_convection_lu","m/s",C.valeur(),pb);
  champs_compris_.ajoute_champ(D.valeur());
  champs_compris_.ajoute_champ(C.valeur());
  const Schema_Temps_base& sch=pb.schema_temps();
  dis.discretiser_champ("vitesse",zone_dis,"vitesse_convection","m/
   s",
                        dimension,sch.nb_valeurs_temporelles(),
                        sch.temps_courant(),vitesse_transport);
  champs_compris_.ajoute_champ(vitesse_transport.valeur());
  Milieu_base::discretiser(pb,dis);
  if (C.non_nul())
    vitesse_transport.valeur().affecter_(C.valeur());
  else
  {
    Cerr<<"To must read "<<que_suis_je()<<" before discretiser "<<
   finl;
    exit();
  }
}

const Champ_base& Constituant_Avec_Vitesse::vitesse_pour_transport
    ()
{
  return vitesse_transport;
}

const Champ_Don& Constituant_Avec_Vitesse::
    vit_convection_constituant() const
{
  return C;
}

void Constituant_Avec_Vitesse::mettre_a_jour(double temps)
{
  Constituant::mettre_a_jour(temps);
  if (C.non_nul())
```

```
64      C.mettre_a_jour(temps);
    vitesse_transport.valeur().affecter_(C.valeur());
66  vitesse_transport.changer_temps(temps);
}
68
int Constituant_Avec_Vitesse::initialiser(const double temps)
70{
  Constituant::initialiser(temps);
72  if (C.non_nul())
      C.initialiser(temps);
74  return 1;
}
```

Constituant_Avec_Vitesse.cpp

4.4 Classe équation `Convection_Diffusion`

La spécificité de cette nouvelle classe d'équation `Convection_Diffusion` par rapport à la classe de base `Convection_Diffusion_Concentration` (3) est que la classe `Convection_Diffusion` possède une référence sur la nouvelle classe de constituant `Constituant_Avec_Vitesse` et que la vitesse de convection est donnée par cette dernière classe et non pas donnée par une résolution d'une équation de l'hydraulique Navier-Stokes.

Le traitement de cette nouvelle référence nécessite les actions suivantes:

1. le rajout de la référence `le_constituant`. Pour ce faire, il est nécessaire d'implémenter les deux sources ci-dessous. Cette technique, non standard C++, est utilisée dans le logiciel `TRUST` pour éviter aux développeurs de manipuler des pointeurs qui peuvent être source des fuites mémoires.

```
1 #ifndef Ref_Constituant_Avec_Vitesse_included
  #define Ref_Constituant_Avec_Vitesse_included
3
  #include <Ref.h>
5 class Constituant_Avec_Vitesse;
  Declare_ref(Constituant_Avec_Vitesse);
7 #endif
```

Ref_Constituant_Avec_Vitesse.h

```
1 #include <Ref_Constituant_Avec_Vitesse.h>
  #include <Constituant_Avec_Vitesse.h>
3 Implemente_ref(Constituant_Avec_Vitesse);
```

Ref_Constituant_Avec_Vitesse.cpp

2. la surcharge des méthodes suivantes [3]:

   - `associer_milieu_base`: Associate le milieu à l'équation de convection diffusion et plus précisément associer le nouveau constituant `Constituant_Avec_Vitesse` à l'équation.
   - `associer_constituant`: Associate le constituant à l'équation `Convection_Diffusion`. Cette méthode est appelée par la méthode précédente `associer_milieu_base`.
   - `lire_motcle_non_standard`: Read les mots clé non standard du jeu de données tel que `convection` pour Associate la vitesse de convection à l'équation de convection diffusion

3. le développement des méthodes `printOn` et `readOn`.

4.4.1 Fichier entête

```
1
#ifndef Eq_Conv_Diff_included
3 #define Eq_Conv_Diff_included

5 #include <Convection_Diffusion_Concentration.h>
#include <Ref_Constituant_Avec_Vitesse.h>
7
class Eq_Conv_Diff : public Convection_Diffusion_Concentration
9 {
  Declare_instanciable(Eq_Conv_Diff);
11 public :
  void associer_milieu_base(const Milieu_base& un_milieu) override;
13  void associer_constituant(const Constituant_Avec_Vitesse&
   un_constituant);
  int lire_motcle_non_standard(const Motcle& mot, Entree& is)
   override;
15 protected :
  REF(Constituant_Avec_Vitesse) le_constituant;
17 };
#endif
```

Eq_Conv_Diff.h

4.4.2 Fichier source

```
#include <Eq_Conv_Diff.h>
2 #include <Constituant_Avec_Vitesse.h>
#include <Discret_Thyd.h>
```

[3] Seul les fonctionnalités spécifiques de la nouvelle classe `Convection_Diffusion` sont décrites.

```
#include <Probleme_base.h>

Implemente_instanciable(Eq_Conv_Diff,"Convection_Diffusion",
   Convection_Diffusion_Concentration);
// XD  convection_diffusion  eqn_base convection_diffusion -1
   not_set

Sortie& Eq_Conv_Diff::printOn(Sortie& is) const
{
  return Convection_Diffusion_Concentration::printOn(is);
}

Entree& Eq_Conv_Diff::readOn(Entree& is)
{
  Convection_Diffusion_Concentration::readOn(is);
  return is;
}

int Eq_Conv_Diff::lire_motcle_non_standard(const Motcle& mot,Entree
   & is)
{
 if (mot=="convection")
 {
  Cerr <<"Reading and typing of the convection operator : "<< finl;
  terme_convectif.associer_vitesse(le_constituant->
   vitesse_pour_transport());
  is >> terme_convectif;
  return 1;
 }else
 {
  Cout << mot << finl;
  return Convection_Diffusion_Concentration::
   lire_motcle_non_standard(mot,is);
 }
 return 1;
}

void Eq_Conv_Diff::associer_milieu_base(const Milieu_base&
   un_milieu)
{
  Convection_Diffusion_Concentration::associer_milieu_base(
   un_milieu);
  const Constituant_Avec_Vitesse& un_constituant=ref_cast(
   Constituant_Avec_Vitesse,un_milieu);
```

```
  if (un_constituant.vit_convection_constituant().non_nul())
    associer_constituant(un_constituant);
  else
  {
    Cerr <<"The dye (constituant) convection has not been defined."
   << finl;
    exit();
  }
}

void Eq_Conv_Diff::associer_constituant(const
   Constituant_Avec_Vitesse& un_constituant)
{
  le_constituant = un_constituant;
}
```

Eq_Conv_Diff.cpp

4.5 Classe Problème `Probleme_Convection_Diffusion`

Dans la section 3 dédiée à la spécification, aucun problème n'a été identifié pour servir de base au nouveau problème `Probleme_Convection_Diffusion`. Nous prenons donc la classe abstraite pure `Probleme_base` comme classe de base (mère) pour la nouvelle classe `Probleme_Convection_Diffusion`. Pour utiliser cette nouvelle classe de problème `Probleme_Convection_Diffusion`, il faut:

1. rajouter l'attribut `eq_conv_diff` qui représente l'équation de convection-diffusion (1).
2. compléter l'implémentation de la classe `Probleme_base` en développant les méthodes suivantes:
   - `nombre_d_equations`: renvoie le nombre d'équation qui compose ce problème. Dans notre cas c'est une seule équation.
   - `equation`: renvoie l'attribut contenant l'équation de convection-diffusion (`eq_conv_diff`).
3. surcharger la méthode `associer_milieu_base` pour Associate le milieu à l'équation de convection de diffusion `eq_conv_diff`.
4. le développement des méthodes `printOn` et `readOn`.

4.5.1 Fichier entête

```
#ifndef Pb_Conv_Diff_included
#define Pb_Conv_Diff_included

```

```
#include <Probleme_base.h>
#include <Eq_Conv_Diff.h>

class Pb_Conv_Diff : public Probleme_base
{
  Declare_instanciable(Pb_Conv_Diff);
public:
  int nombre_d_equations() const override;
  const Equation_base& equation(int i) const override;
  Equation_base& equation(int i) override;
  void associer_milieu_base(const Milieu_base& un_milieu) override;
protected:
  Eq_Conv_Diff eq_conv_diff;
};
#endif
```

Pb_Conv_Diff.h

4.5.2 Fichier source

```
#include <Pb_Conv_Diff.h>

Implemente_instanciable(Pb_Conv_Diff,"Probleme_Convection_Diffusion
   ",Probleme_base);
// XD probleme_convection_diffusion Pb_base
   probleme_convection_diffusion -1 not_set
// XD   attr convection_diffusion  convection_diffusion
   convection_diffusion 1 not_set
Sortie& Pb_Conv_Diff::printOn(Sortie& os) const
{
  return Probleme_base::printOn(os);
}

Entree& Pb_Conv_Diff::readOn(Entree& is)
{
  return Probleme_base::readOn(is);
}

int Pb_Conv_Diff::nombre_d_equations() const
{
  return 1;
}

const Equation_base& Pb_Conv_Diff::equation(int i) const
```

```
{
23   assert (i==0);
   return eq_conv_diff;
25 }

27 Equation_base& Pb_Conv_Diff::equation(int i)
{
29   assert (i==0);
   return eq_conv_diff;
31 }

33 void Pb_Conv_Diff::associer_milieu_base(const Milieu_base&
    un_milieu)
{
35   eq_conv_diff.associer_milieu_base(un_milieu);
}
```

Pb_Conv_Diff.cpp

5 Processus de vérification des développements

Dans cette section, nous présentons:

- la mise en place des tests de vérification,
- la génération automatique d'un rapport de vérification des développements via l'outil de génération `genererCourbes`,
- la vérification de la non-régression.

5.1 Mise en place des tests de vérification

La mise en place des tests de vérification consiste à créer:

- un répertoire pour chaque test de vérification,

```
$ cd convection_diffusion
2 $ mkdir -p share/Validation/Rapports_automatiques/Cas-tests/src
    /cas-test1
  $ mkdir -p share/Validation/Rapports_automatiques/Cas-tests/src
    /cas-test2
4
  Cas-tests est le répertoire contenant les tests de vérification
    .
6 cas-test1 est le répertoire contenant le premier cas de
    vérification.
```

```
cas-test2 est le répertoire contenant le deuxième cas de
   vérification.
```

- un fichier `PRM` dans le répertoire `convection_diffusion/share/Validation/Rapports_automatiques/Ca` qui sert à générer automatiquement un rapport de vérification (section 6). Ce fichier `PRM` est présenté dans la section **??**.
- un jeu de donnée pour chaque test de vérification (sections **??** et **??**).

5.2 Génération de rapport de vérification

La génération consiste à exécuter la commande `make validation` dans le répertoire `convection_diffusion`. Dans ce même répertoire, l'exécution de cette commande générera le rapport de vérification `validation.pdf`. Ce rapport est inclus dans cette note technique à la section 6.

5.3 Vérification de la non-régression

Pour utiliser les tests mis en place précédemment dans le rapport de vérification comme des tests de non-régression, il suffit de:

- créer un répertoire qui contiendra les résultats de référence dans le répertoire `convection_diffusion`:

```
$ cd convection_diffusion
$ mkdir -p tests/Reference/Validation
```

- lancement des tests et vérification de la non-régression en exécutant la commande `make check_optim` dans le répertoire `convection_diffusion`. Le résultat de cette commande est illustré dans la figure (cf. figure 5.3). Cette figure montre que les résultats en séquentiel et en parallèle ne présentent aucune régression.

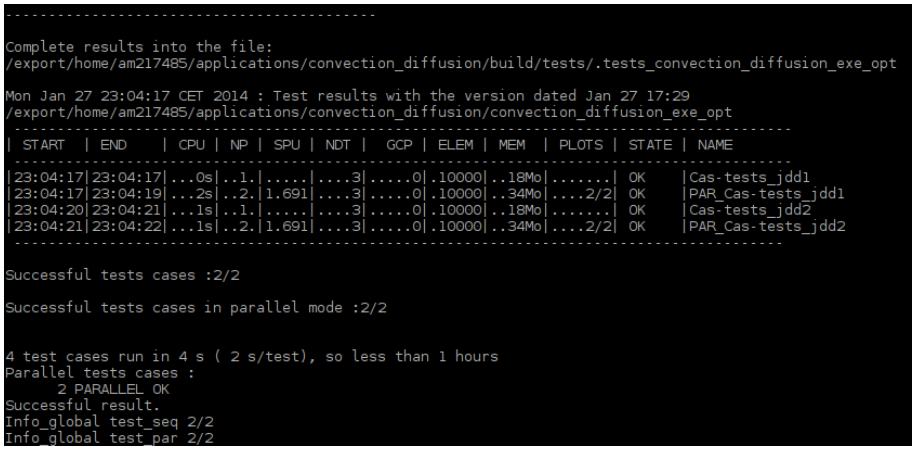

Figure 5: Résultats des tests de non-régression

6 Rapport de vérification

7 Conclusion

Dans ce tutoriel, nous avons décrit à travers la résolution d'une équation de convection-diffusion les aspects suivants:

- l'environnement de développement du logiciel `TRUST` en réalisant une nouvelle application `BALTIK`,
- la démarche de spécifications et de développements. Dans cette démarche, il est nécessaire de réaliser des choix techniques tels que l'association de la vitesse de convection à un constituant et non pas à l'équation ou problème). Ces choix ne sont ni uniques ni meilleurs que d'autres. Il est à noter que le but principal de ce tutoriel est de présenter une démarche accompagnée de ces propres choix techniques en donnant quelques recommandations ou méthodologies à suivre lors de sa phase de spécification.
- la mise en place des tests de non-régression et la génération automatique d'un rapport de vérification des développement via des outils proposés par le logiciel `TRUST`.

References

[1] U. Bieder, V. Barthel, F. Ducros, F. Perdu, and P. Quéméré. Trio_u users manual : Methodology for incompressible single phase flow in industrial applications. Note Technique DER/SSTH/LMDL/2007-058, CEA, 2007.

[2] M. Farvaque, Ph. Emonot, and O. Cueto. Document de conception trio_u version 1. Note Technique STR/LTML 96-20, CEA, 1996.

[3] Trio_U teams. Trio_u user's manual v1.6.8. Note technique, CEA, 2013.