OpenFLUID
Software Environment
for Modelling Fluxes in Landscapes

TP2 : Production de variables

Objectifs:	Produire des variables sur différentes unités spatiales, utiliser des boucles de parcours de l'espace
Pré-requis:	TP1

A partir du simulateur "vide" créé lors du TP1 (`training.signal.prod`), nous allons générer un signal de pluie sous la forme d'une variable produite sur toutes les unités de la classe SU. Ce signal est généré par une sinusoïde adaptée pour produire des valeurs de pluie sur une durée correspondant aux deux premiers tiers de la simulation.

1 Code source

Les modifications du code source sont à apporter dans le fichier `SignalSim.cpp`.

1.1 Signature

Nous allons tout d'abord déclarer dans la signature que le simulateur va produire une nouvelle variable nommée `water.atm-surf.H.rain`. Cette déclaration se fait à l'aide de l'instruction `DECLARE_PRODUCED_VAR`. L'instruction `DECLARE_PRODUCED_VAR` comporte 4 paramètres :

- le nom de la variable produite
- la classe d'unité sur laquelle est produite la variable
- la description de la variable
- l'unité (SI) de la variable

Une fois complétée, la signature devrait être similaire à :

```
BEGIN_SIMULATOR_SIGNATURE("training.signal.prod");
  DECLARE_NAME("");
  DECLARE_DESCRIPTION("");

  DECLARE_VERSION("13.05");
  DECLARE_STATUS(openfluid::ware::EXPERIMENTAL);

  DECLARE_DOMAIN("");
  DECLARE_PROCESS("");
  DECLARE_METHOD("");
```

```
  DECLARE_AUTHOR("Doe J.","doe@foobar.org");


  DECLARE_PRODUCED_VAR("water.atm-surf.H.rain","SU","rainfall height on SU","m");

  //Scheduling
  DECLARE_SCHEDULING_DEFAULT;
END_SIMULATOR_SIGNATURE
```

Après construction/installation du simulateur, il est possible de vérifier si la signature a bien été modifiée en exécutant la commande `openfluid -u training.signal.prod` ou `openfluid -r` (pour l'ensemble des simulateurs disponibles).

1.2 initializeRun()

La méthode `initializeRun()` est appelée en début de simulation et permet d'initialiser les variables produites par le simulateur.

```
openfluid::base::SchedulingRequest initializeRun()
{

  openfluid::core::Unit* SU;
  OPENFLUID_UNITS_ORDERED_LOOP("SU",SU) // debut de la boucle spatiale
  {
    // Initialisation a 0 de la variable water.atm-surf.H.rain pour chaque SU
    OPENFLUID_InitializeVariable(SU,"water.atm-surf.H.rain",0.0);
  }
  return DefaultDeltaT();
}
```

1.3 runStep()

La méthode `runStep()` est appelée à chaque index de temps pour lequel le simulateur est actif. Nous allons y ajouter le code de calcul du signal qui sera produit sur chaque unité de la classe SU au travers de la variable `water.atm-surf.H.rain`. Pour cela, nous allons utiliser une boucle spatiale sur les SU, et produire la variable à l'aide de l'instruction `OPENFLUID_AppendVariable`.

Une boucle spatiale est identifiée au travers de l'instruction `OPENFLUID_UNITS_ORDERED_LOOP`, elle débute et se termine avec des accolades `{}`.

Le générateur du signal proposé pour cet exercice comporte deux phases :
- Une fonction basée sur un cosinus pendant les deux premiers tiers de la simulation,
- une valeur égale à 0.0 après les deux premiers tiers de la simulation.

Soit P_i la valeur du signal de pluie courant, P_{max} la valeur maximale du signal de pluie,
t_i l'index de temps courant, t_{max} l'index de temps maximal du signal,
t_{total} la durée de la simulation, t_0 l'index de temps 0 de la simulation

$$t_{max} = \frac{2}{3} \cdot t_{total}$$
$$0.0005 \leq P_{max} \leq 0.001$$
$$\text{Si } t_0 \leq t_i \leq t_{max} \text{ alors } P_i = P_{max} \cdot \frac{1-\cos\frac{2.\pi.t_i}{t_{max}}}{2}$$
$$\text{Si } t_i > t_{max} \text{ alors } P_i = 0.0$$

Si vous le souhaitez, vous pouvez intégrer votre propre équation de génération du signal.
Une fois complétée, la méthode runStep() devrait être similaire à :

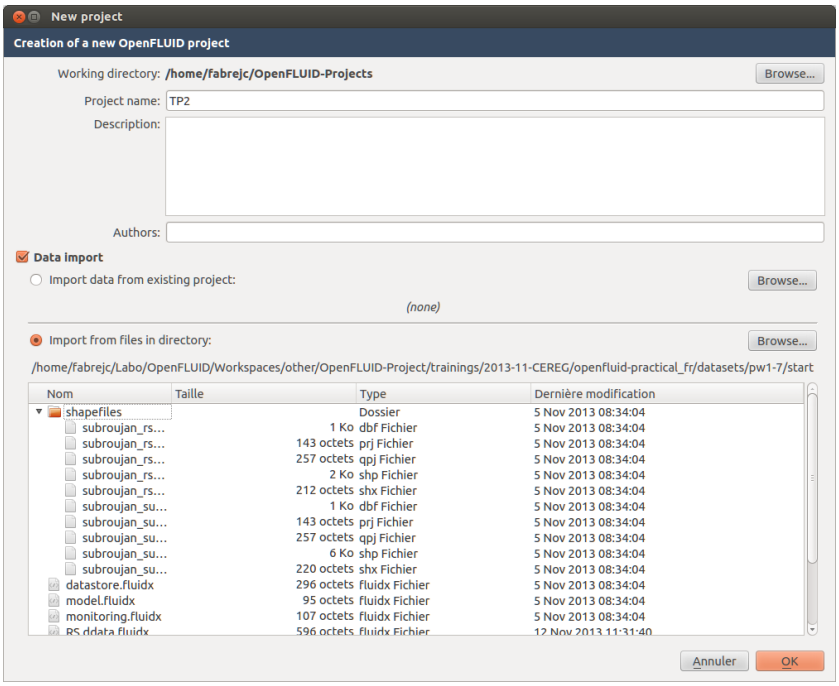
```
openfluid::base::SchedulingRequest runStep()
{
  openfluid::core::Unit* pSU;  // pointeur sur la SU courante
  openfluid::core::DoubleValue Value;  // valeur du signal à calculer

  const double MaxRainValue = 0.0008;
  openfluid::core::TimeIndex_t LastRainIndex = OPENFLUID_GetSimulationDuration()*2/3;
  openfluid::core::TimeIndex_t CurrentIndex = OPENFLUID_GetCurrentTimeIndex();

  // calcul de la valeur du signal
  Value = 0.0;
  if (CurrentIndex <= LastRainIndex)
  {
    Value = MaxRainValue * (1 - std::cos(double(2*CurrentIndex*3.1415926)
                          /(double(LastRainIndex)))) / 2;
  }

  OPENFLUID_UNITS_ORDERED_LOOP("SU",pSU)  // debut de la boucle spatiale
  {
    // production de la valeur du signal
    OPENFLUID_AppendVariable(pSU,"water.atm-surf.H.rain",Value);
  }   // fin de la boucle spatiale

  return DefaultDeltaT();
}
```

2 Simulation

Pour la simulation, nous allons utiliser le jeu de données "Bassin versant TP". Le jeu de données comporte 15 unités de classe SU, 14 unités de classe RS, et est paramétré pour une simulation sur une période du 28 avril 1998 à 6h00 au 29 avril à 22h00 avec un pas de temps d'échange de 60 secondes.

2.1 ... avec l'interface OpenFLUID-Builder

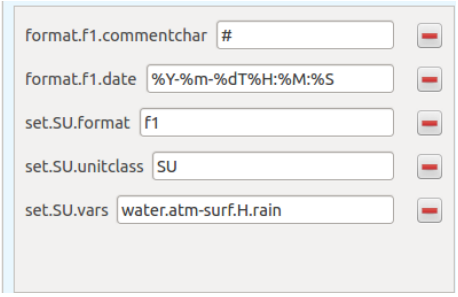
Lancer OpenFLUID-builder en cliquant sur l'icône présente sur le Bureau. Cliquez sur *Créer un projet*, vérifier que le chemin du dossier de travail pointe bien sur `/home/openfluid/Bureau/formation/projects`, nommer votre projet `TP2` et importez les données nécessaires en cliquant sur `Import de données`, `Importez des fichiers de données` en pointant vers le dossier `/home/openfluid/Bureau/formation/datasets/TP1-7`. Et cliquez sur OK.



Une fois le projet créé, il faut ajouter notre simulateur dans le modèle. Pour cela, dans l'onglet *Model*, cliquer sur *Add simulator* pour ajouter le simulateur que l'on vient de créer (`training.signal.prod`)

Vérifier le pas de temps (DeltaT) et la période de simulation (onglet *Simulation configuration*).

Ensuite, ajouter un observateur, via l'onglet *Monitoring* en cliquant sur *Add observer*. Une des fonctions types d'un observateur est d'extraire des valeurs de variables au cours de la simulation pour les sauvegarder dans un format donné.
Ajouter un observateur de type *export.vars.file.csv*. Paramétrer l'observateur pour sauvegarder la variable *water.atm-surf.H.rain* pour toutes les SU.



Note: Pour paramétrer cet observateur, il faut définir un format pour la date, le caractère de séparation de colonne, et le caractère de commentaire de ligne (qui permet d'ignorer la ligne lors d'un post-traitement). se reporter au tutoriel sur l'utilisation des observateurs pour des compléments d'information.

Enfin, lancer la simulation en cliquant sur le bouton *Exécution* de la barre d'outils.

Si tout s'est bien passé, les résultats de la simulation sont accessibles via le navigateur de projet, dans l'onglet *Outputs browser*,
Double-cliquer sur le fichier .csv correspondant à l'unité spatiale pour laquelle vous voulez visualiser les résultats.

Fermer OpenFLUID-Builder en enregistrant votre projet.

2.2 ... en ligne de commande

Note: L'utilisation de la ligne de commande est donnée ici à titre indicatif, il n'est pas nécessaire d'effectuer ces actions pour l'enchaînement des TP.

Le jeu de données (par exemple le contenu du dossier `/home/openfluid/Bureau/formation/projects/TP2` doit être déposé dans un répertoire qui sera ensuite utilisé par le moteur de calcul. (par exemple `/home/openfluid/Bureau/formation/inputs/TP1-TP7`).

Pour lancer une simulation à partir du jeu de données, nous allons utiliser la commande `openfluid` en précisant le répertoire du jeu de données en entrée et celui des résultats.

- jeu de donnée en entrée : `/home/openfluid/Bureau/formation/inputs/TP1-TP7`
- résultats : `/home/openfluid/Bureau/formation/outputs/TP2`

La commande à exécuter est donc :

```
openfluid -i /home/openfluid/Bureau/formation/inputs/TP1-TP7
  -o /home/openfluid/Bureau/formation/outputs/TP2
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `/home/openfluid/Bureau/formation/outputs/TP2`.

3 Annexe : Formats de dates

Format	Description
%a	locale's abbreviated weekday name.
%A	locale's full weekday name.
%b	locale's abbreviated month name.
%B	locale's full month name.
%c	locale's appropriate date and time representation.
%C	century number (the year divided by 100 and truncated to an integer) as a decimal number [00-99].
%d	day of the month as a decimal number [01,31].
%D	same as %m/%d/%y.
%e	day of the month as a decimal number [1,31] ; a single digit is preceded by a space.
%h	same as %b.
%H	hour (24-hour clock) as a decimal number [00,23].
%I	hour (12-hour clock) as a decimal number [01,12].
%j	day of the year as a decimal number [001,366].
%m	month as a decimal number [01,12].
%M	minute as a decimal number [00,59].
%n	is replaced by a newline character.
%p	locale's equivalent of either a.m. or p.m.
%r	time in a.m. and p.m. notation ; in the POSIX locale this is equivalent to %I :%M :%S %p.
%R	time in 24 hour notation (%H :%M).
%S	second as a decimal number [00,61].
%t	is replaced by a tab character.
%T	time (%H :%M :%S).
%u	weekday as a decimal number [1,7], with 1 representing Monday.
%U	week number of the year (Sunday as the first day of the week) as a decimal number [00,53].
%V	week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1.
%w	weekday as a decimal number [0,6], with 0 representing Sunday.
%W	week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.
%x	locale's appropriate date representation.
%X	locale's appropriate time representation.
%y	year without century as a decimal number [00,99].
%Y	year with century as a decimal number.
%Z	timezone name or abbreviation, or by no bytes if no timezone information exists.
%%	character %.

Exemples :

- `%Y-%m-%dT%H:%M:%S` ⇔ `2008-01-01T11:13:00`

- %Y-%m-%d %H:%M:%S ⇔ 2008-01-01 11:13:00
- %Y%m%d%H%M%S ⇔ 20080101111300