OpenFLUID
Software Environment
for Modelling Fluxes in Landscapes

TP2 : Production de variables en sortie

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

A partir de la fonction de simulation "vide" créée lors du TP1 (`formation.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.

1 Code source

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

1.1 Signature

Nous allons tout d'abord déclarer dans la signature que la fonction 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_SIGNATURE_HOOK
  DECLARE_SIGNATURE_ID("formation.signal.prod");
  DECLARE_SIGNATURE_NAME("");
  DECLARE_SIGNATURE_DESCRIPTION("");

  DECLARE_SIGNATURE_VERSION("1.0");
  DECLARE_SIGNATURE_SDKVERSION;
  DECLARE_SIGNATURE_STATUS(openfluid::base::EXPERIMENTAL);

  DECLARE_SIGNATURE_DOMAIN("");
```

```
  DECLARE_SIGNATURE_PROCESS("");
  DECLARE_SIGNATURE_METHOD("");
  DECLARE_SIGNATURE_AUTHORNAME("Chuck Norris");
  DECLARE_SIGNATURE_AUTHOREMAIL("norris@gmail.com");

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

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

1.2 runStep()

La méthode `runStep()` est appelée à chaque pas de temps. 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 déclarée et identifiée au travers de l'instruction `DECLARE_UNITS_ORDERED_LOOP`, elle débute avec `BEGIN_UNITS_ORDERED_LOOP` et se termine avec `END_LOOP`.
Le générateur du signal proposé pour cet exercice est basé sur une fonction sinus, paramétrée par le numéro de pas de temps courant, et modifiée par un coefficient :

Soit V la valeur du signal, t_i le numéro de pas de temps courant, k un coefficient d'amplitude

$$V = \sin(\frac{t_i}{k} \times \frac{\pi}{180}) + 1$$

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

```
bool runStep(const openfluid::base::SimulationStatus* SimStatus)
{
  openfluid::core::Unit* pSU;  // pointeur sur la SU courante
  openfluid::core::ScalarValue Value;  // valeur du signal à calculer
  DECLARE_UNITS_ORDERED_LOOP(1);  // declaration d'une boucle spatiale d'identifiant 1

  BEGIN_UNITS_ORDERED_LOOP(1,"SU",pSU);  // debut de la boucle spatiale

    // calcul de la valeur du signal
    Value = (std::sin((SimStatus->getCurrentStep()/5)*3.1415926/180) + 1);

    // production de la valeur du signal
    OPENFLUID_AppendVariable(pSU,"water.atm-surf.H.rain",Value);

  END_LOOP;  // fin de la boucle spatiale

  return true;
}
```

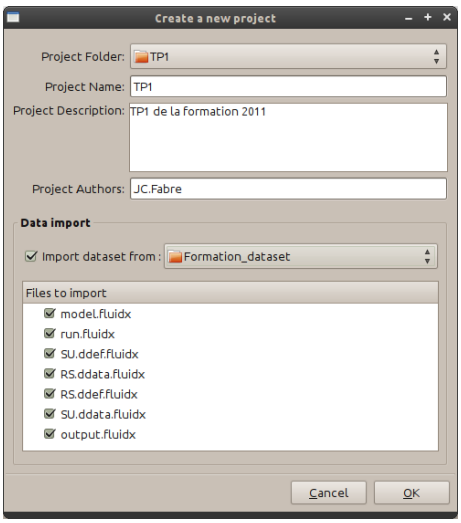
Attention: Il est nécessaire d'inclure le header <cmath> pour pouvoir utiliser les fonctions mathématiques. Pour cela, rajouter `include <cmath>` avec les autres commandes `include` en début de fichier `.cpp`.

2 Simulation

Pour la simulation, nous allons utiliser le jeu de données "Bassin versant virtuel". Le jeu de données comporte 24 unités de classe SU, 7 unités de classe RS, et est paramétré pour une simulation sur une période du 28 avril au 2 mai 1998 avec un pas de temps d'échange de 60 secondes.

2.1 ... avec l'interface OpenFLUID-Builder

Il est tout d'abord nécessaire de créer un projet OpenFLUID (par exemple dans `/home/nomutilisateur/formationOF/projects/TP2`) et d'y importer le jeu de données.



Une fois le projet créé, Il faut ajouter notre fonction de simulation dans le modèle. Pour cela, éditer la définition du modèle (double-cliquer sur *Model* dans le navigateur de projet à gauche) en y rajoutant la fonction que l'on vient de créer (formation.signal.prod)

Puis vérifier le pas de temps et la période de simulation (double-cliquer sur *Simulation > Run* dans le navigateur de projet).

Ensuite, créer un jeu de résultats en sortie si celui-ci n'est pas déjà présent, via l'onglet *Output* du navigateur de projet. Un jeu de résultat permet de sélectionner les variables dont on veut sauvegarder les valeurs au cours de la simulation, pour une ou plusieurs unités spatiales d'une classe donnée, dans un format donné.
Un format définit le format de 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). Des exemples et les spécifications des format de date sont disponibles en annexe de cette fiche TP.

Attention: Afin de faciliter le traitement des résultats en post-simulation, il est préférable de ne pas choisir un séparateur de colonne qui serait présent dans le format de date.

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

Si tout s'est bien passé, les résultats de la simulation sont accessibles via le navigateur de projet, dans la rubrique *Results*, classés par jeu de résultat.
Sélectionnez l'unité spatiale pour laquelle vous voulez visualiser les résultats, en gardant cochée la case *Show file(s)*. Depuis l'onglet du ou des fichiers de résultats (`.out`), vous pouvez accéder à une visualisation via GNUplot.

2.2 ... en ligne de commande

Ce jeu de données doit être déposé dans un répertoire qui sera ensuite utilisé par le moteur de calcul. (par exemple `/home/nomutilisateur/formationOF/dataset`).

Afin que notre fonction de simulation soit prise en compte dans le modèle que l'on va exécuter, elle doit être déclarée dans le fichier `model.fluidx`. Pour cela, il faut modifier ce fichier et rajouter cette fonction entre les balises `<model>` et `</model>` comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<openfluid>
  <model>
    <function fileID="formation.signal.prod"/>
  </model>
</openfluid>
```

Note: En XML, les commentaires commencent par `<!--` et se terminent par `-->`

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

- jeu de donnée en entrée : `/home/nomutilisateur/formationOF/dataset`
- résultats : `/home/nomutilisateur/formationOF/outputs/TP2`

La commande à exécuter est donc :

```
openfluid-engine -i /home/nomutilisateur/formationOF/dataset
  -o /home/nomutilisateur/formationOF/outputs/TP2
```

(à taper sur une seule ligne)

Si tout s'est bien passé, les résultats de la simulation sont accessibles dans `/home/nomutilisateur/formationOF/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