Resumen sobre la utilización de Simflowny

Martini Christophe

September 5, 2023

Contents

1 Principios de utilización

Este documento tiene el propósito de ayudar al lector con la utilización de Simflowny, por ello se recomienda que el lector consulte la documentación y los códigos de ejemplo.

Simflowny está basado en una interfaz gráfica que permite manejar documentos fácilmente. Para generar un código se necesitan de dos archivos principales. Primero, se requiere de un modelo en el cual declaran todas las ecuaciones de evolución a utilizar (el modelo físico). Posteriormente, es necesario definir el problema donde se estipulan las condiciones iniciales, las condiciones de frontera del problema, las cantidades adicionales que uno

quiera analizar y todas las variables que se necesiten para definir el problema. Adicionalmente, se generará una póliza de discretización en la que se deberán precisar los métodos de discretización a utilizar.

Simflowny utiliza un lenguaje ML propio llamado SIML, lamentablemente no se encontró una documentación que explique claramente su sintaxis. Por ello, para familiarizarse con el lenguaje es necesario comprender los ejemplos de códigos de modelos y problemas. En esta carpeta adjuntamos los códigos comentados del modelo de ecuaciones de Einstein en vacío y el problema de las ondas de Brill explicando su estructura. En *https://bitbucket.org/iac3/simflowny/wiki/UserGuide* se puede encontrar la guía de usuario de Simflowny para más información. De la misma manera, sugerimos al lector instalar Simflowny con Docker en una máquina local y, mediante las interfaces gráficas, pasar todos los archivos que se encuentren en el servidor de Docker al servidor de Simflowny que se vaya a utilizar debido a que en la versión de Docker se encuentran ejemplos y archivos necesarios para la generación de los códigos.

La generación del código en Simflowny cuenta con varios pasos:

- Crear el modelo
- Crear el problema
- Generar la póliza de discretización mediante la interfaz gráfica
- Editar la póliza de discretización
- Generar el problema discretizado mediante la interfaz gráfica
- Generar el código SAMRAI mediate la interfaz gráfica

2 Inicializar la interfaz de usuario desde el cluster del ICN

La instalación de Simflowny en el cluster del ICN consta de un archivo y de una carpeta. Normalmente, una vez que Simflowny está instalado, el usuario debería encontrar un archivo llamado *simflowny_modules.sh* y una carpeta llamada *mathMS-3.1*. Para cargar los modulos de Simflowny (esto solamente es necesario para correr) se corre el comando *source simflowny_modules.sh*. Posteriormente, se tiene que ir a la carpeta *mathMS-3.1* y ejecutar el comando *./bin/start* para iniciar el programa. A continuación, se tiene que ir al navegador internet e ingresar a la dirección

http://localhost:8181/mathMSGui/index.html (después de unos 15 segundos), esto debería de abrir la interfaz de usuario. Adicionalmente, para parar el programa se puede ejecutar *./bin/stop* en la carpeta *mathMS-3.1.*

3 Descripción de los pasos y documentos

Describiremos rápidamente los archivos del modelo y del problema puesto que estos vienen comentados en la carpeta. También es necesario precisar que, a veces, cuando se sube un documento al servidor aparece un error porque puede haber algún campo faltante. No obstante, el documento es cargado por la interfaz gráfica y este se puede modificar posteriormente para añadir el campo faltante.[1] Un ejemplo de esto es cuando se sube el archivo del problema y no precisa el número de identificación del modelo, el cual es mejor llenar directamente en la interfaz gráfica.

3.1 Modelo

El archivo del modelo tiene que tener la estructura siguiente:

- head: En esta parte del archivo se tiene que declarar el nombre del archivo, el autor y la version. La fecha y el número de identificación del documento se generan automáticamente cuando el documento es subido al servidor.

- coordinates: Se precisan los nombres de las coordenadas espaciales y temporales que tienen que ser, una variable temporal y dos o tres variables espaciales (el código no acepta simulaciones con menos de 2 variables espaciales).

- fields: Se declaran todos los campos del modelo físico. En Simflowny, los campos son todas las variables que habrá que evolucionar en el tiempo y las únicas variables que podrán ser derivadas temporalmente.

- auxiliaryFields: Se declaran todos los campos auxiliares. Estos campos no evolucionan en el tiempo. Simflowny acepta en sus ecuaciones un orden máximo en derivadas de uno en el tiempo y un orden dos en el espacio. En consecuencia, se necesitan campos auxiliares para utilizar ordenes superiores en el espacio.

- auxiliaryVariables: Se declaran todas las variables que se van a utilizar. Las variables son cantidades que no pueden ser derivadas.

[1] El lector tendrá que diferenciar cuando hablemos de los campos numéricos (como en este caso) e.g. el nombre de un archivo y, de los campos relativos a la física e.g. la curvatura extrínseca.

- Parameters: Se declaran todos los parámetros que se van a utilizar. Los parámetros tampoco son cantidades derivables.

- evolutionEquations: Se declaran todas las ecuaciones de evolución de los campos. Las ecuaciones de evolución se presentan bajo la forma:

$$\partial_t \boldsymbol{u} + \partial_i \boldsymbol{F}^i(\boldsymbol{u}) = \boldsymbol{S}(\boldsymbol{u}) + \boldsymbol{O}(\boldsymbol{u}, \partial_i \boldsymbol{u}), \tag{1}$$

donde $\boldsymbol{u}$ es una colección de campos, $\boldsymbol{F}$ es un término de flujo, $\boldsymbol{S}$ es un término de fuentes y $\boldsymbol{O}(\boldsymbol{u}, \partial_i \boldsymbol{u})$ es cualquier operador función de $\boldsymbol{u}$ y de sus derivadas espaciales.

En en los ejemplos adjuntos solamente utilizaron la derivada temporal de los campos y los operadores. Las ecuaciones se programan de la siguiente manera, primero se define el campo a variar en el tiempo y luego se introducen los operadores. En cuanto se declara el campo en la ecuación, el programa interpreta esto directamente como $\partial_t \boldsymbol{u} =$, posteriormente, al declarar el operador se tiene $\partial_t \boldsymbol{u} = \boldsymbol{O}(\boldsymbol{u}, \partial_i \boldsymbol{u})$, y de esta manera se completa la ecuación.

Adicionalmente, cada operador debe de tener un nombre. A cada nombre se le puede aplicar una dicretización diferente (en la póliza de discretización), esto es, diferencias finitas de cuarto orden, centradas, hacia adelante o hacia atrás.

- auxiliaryFieldEquations: Se declaran las ecuaciones de los campos auxiliares. Estos campos no evolucionan en el tiempo por lo que, al ser declarado el campo, la ecuación será directamente la definición del campo auxiliar y no la de su variación temporal como en el caso de las ecuaciones de evolución.

- auxiliaryVariableEquations: Se declaran todas las ecuaciones de las variables. Como en las ecuaciones de los campos auxiliares, las variables no evolucionan en el tiempo.

3.2 Problema

El archivo del problema tiene que tener la siguiente estructura:

- head: En esta parte del archivo se tiene que declarar el nombre del archivo, el autor y la version. La fecha y el número de identificación del documento se generan automáticamente cuando el documento es subido al servidor.

- coordinates: Se precisan los nombres de las coordenadas espaciales y temporales que tienen que ser, una variable temporal y dos o tres variables espaciales (el código no acepta simulaciones con menos de 2 variables espaciales).

- fields: Se declaran los campos que van a ser usados en el problema, estos tienen que corresponder a los campos definidos en el modelo.

- auxiliaryVariables: Se declaran todas las variables que se vayan a utilizar en el problema. De la misma manea, tienen que corresponder a las variables utilizada en el modelo y, es posible introducir nuevas variables pero se tendrán que definir posteriormente.

- analysisFields: Se declaran campos de análisis. Los campos de análisis corresponden a propiedades que no evolucionan en el tiempo pero que se quieren estudiar. En consecuencia, los campos de análisis constituyen todas las variables sin ecuaciones de evolución temporales que serán guardadas a lo largo de las simulaciones.

- auxiliaryAnalysisVariables: Se declaran las variables auxiliares de análisis. Este tipo de variables es similar a las variables auxiliares, no se guardan en la memoria y son las variables utilizadas para simplificar las ecuaciones declaradas en el problema.

- parameters: Se declaran todos lo parámetros que vaya a utilizar el problema. Estos tienen que corresponder a los parámetros del modelo. Se pueden declarar nuevos parámetros en esta sección.

- models: Se declaran los modelos que se van a utilizar para la resolución del problema. Se pueden declarar varios modelos. La declaración se hace mediante el número de identificación de los modelos, por eso se recomienda añadir los modelos directamente en la interfaz de usuario donde, haciendo click en el campo correspondiente se puede seleccionar el modelo que se quiere utilizar y la identificación se completa automáticamente. Adicionalmente, los modelos tienen que nombrarse para poder referenciarlos más adelante.

- regions: Se declaran las regiones del problema las cuales corresponden a los límites del dominio de simulación. Se pueden definir varias regiones donde cada una tiene que corresponder a un modelo (el modelo se especifica por su nombre). De esta manera, es posible aplicar diferentes leyes de la física a diferentes regiones del dominio de simulación.

En esta parte también se deben de definir las condiciones iniciales del problema. Todos los campos definidos en el modelo tienen que estar inicializados. De la misma manera, las condiciones de frontera pueden precisarse con una función *if*, de esta forma es posible aplicar diferentes condiciones iniciales en diferentes partes del dominio de simulación lo que especialmente útil cuando se tienen singularidades en las funciones.

- auxiliaryFieldEquations: Se declaran todas las ecuaciones de los campos de ánalisis. Como en las ecuaciones de los campos auxiliares, las variables no evolucionan en el tiempo.

- auxiliaryAnalysisVariableEquations: Se declaran todas las ecuaciones de las variables auxiliares de análisis.

- boundaryConditions: Se explicitan las condiciones de frontera para cada región. De la misma manera, se precisan los ejes y los extremos sobre los cuales se aplican. Para utilizar las condiciones de fronteras pre-programadas de Simflowny, se recomienda completar los campos en la interfaz de usuario puesto que las condiciones se pueden seleccionar directamente.

- boundaryPrecedence: Se define el orden de cálculo para las condiciones de frontera. Esto es importante cuando se tienen diferentes tipos de condiciones de frontera y se debe seleccionar una por encima de la otra.

- finalizationConditions: Se precisa la condición para detener la simulación en función de las variables, los campos y los parámetros previamente declarados.

3.3 Generación de la póliza de discretización

Al haber terminado el problema y el modelo, estos tienen que subirse al servidor (interfaz de usuario). Una vez que los dos archivos estén en el servidor, se selecciona el problema y en la barra de herramientas superior hay un botón que genera la póliza de discretización.

3.4 Póliza de discretización

Una vez que la póliza de discretización fue generada se tiene que editar para precisar como discretizar los operadores y el integrador temporal a utilizar. Se recomienda llenar el documento directamente en la interfaz de usuario debido a que todos los campos se tienen que llenar con el número de identificación de los documentos.

- head: Es declara la cabecera del archivo que tiene la misma estructura que las de los archivos pasados.

- problem: Se indica el problema al que se le va a aplicar la póliza.

- meshVariables: Se nombran los campos y los campos auxiliares a discretizar.

- regionDiscretizations: Se declaran los operadores de discretización para cada operador. Para ello, se seleccionan los operadores que vienen en los archivos (pre-guardados en la versión de Docker) de Simflowny. Existen tres operadores de discretización, el operador crossed, el opreador forward y el operador backward que, respectivamente corresponden a diferencias finitas, centradas, hacia adelante y hacia atrás.

Es importante precisar que a veces la póliza de discretización puede tener un *bug* en este paso. El problema puede aparecer al seleccionar el operador al que se le quiere asignar el operador de discretización, esto es, a veces al querer seleccionar el nombre del operador, éste último aparece duplicado y, una de las dos opciones causa un problema mientras que la otra funciona correctamente.

En la póliza también se tiene que definir el integrador temporal. Al igual que para las discretizaciones espaciales, se busca el integrador en los archivos de Simflowny, específicamente en la carpeta de los resolvedores hyperbólicos.

- analysisDiscretizations: En esta sección se tiene que seleccionar nuevamente el operador de discretización pero esta vez para los operadores de las variables de análisis. Adicionalmente, se tienen que declarar dos *input variables*: $cc y $f_n.[2]

- extrapolationInformation: En esta sección se selecciona el método de extrapolación.

3.5 Generación del código

Cuando la póliza de discretización esté completada, se utiliza un botón de la barra de herramientas superior para generar el problema discretizado. Una vez que se tenga el problema discretizado, se selecciona este último y se vuelve a utilizar un botón en la barra superior para generar el código SAMRAI, finalmente este se descarga y se descomprime para obtener la carpeta con el código.

4 Datos externos

Para incluir datos externos en Simflowny, estos se tienen que declarar en el archivo del problema. Para ello se incluyen los nombres de las variables externas con la opción *External Initial Condition* de la interfaz gráfica.

Ya que se haya abierto la carpeta del código, el código espera dos archivos llamados *ExternalInitialData.cpp* y *ExternalInitialData.h*, estos son el código para leer datos iniciales externos y su cabecera. En la carpeta generada vienen varios ejemplos ya hechos de como cargar los datos de forma externa, no obstante, estos están pobremente comentados y utilizan funciones de la librería externa *LORENE*. Dicho esto, se adjunta en la carpeta de este documento un código que muestra como se cargan datos iniciales para las

[2] No se llegó a encontrar una explicación clara para la inclusión de estas dos variables.

ondas de Brill.[3]

Finalmente, en el *Makefile* se tienen que eliminar tres librerías (o instalarlas) para que el código compile, las librerías a eliminar son las dependencias de *LORENE* y serán indicadas por el compilador.

5 Correr el código

Cuando todos los pasos anteriores hayan sido completados se compila la carpeta con el *Makefile*. Posteriormente, uno puede modificar los parámetros de la simulación en el archivo *problem.input*. En dicho archivo el lector encontrará los timesteps, el número de celdas, el método de refinamiento de mallas, los parámetros del problema etc... que podrá modificar si le es necesario.

Finalmente, para ejecutar la simulación el usuario deberá utilizar de la siguiente manera el ejecutable generado *./nombre_del_ejecutable problem.input*.

6 Lista de los archivos incluidos con este documento

- Einstein.yaml: Archivo del modelo para las ecuaciones de Einstein en el Vacío.
- Brill.yaml: El archivo del problema de Ondas de Brill. Este archivo no es la versión con datos externos, para incluir datos externos se tiene que eliminar ϕ de los datos iniciales y declararse como datos externos.
- numerical_schemes.pdf: Archivo que explica los métodos numéricos de Simflowny.
- Brill_external: Carpeta generada por Simflowny. Es la carpeta relativa al problema de las ondas de Brill con datos externos. Los documentos *phi.h5*, *ExternalInitialData.cpp* y *ExternalInitialData.h*, son archivos creados a parte. Estos archivos corresponden respectivamente a datos iniciales para las ondas de Brill, al código para leer los datos y a su cabecera.

[3]No sé logró calibrar bien la lectura de datos por lo que él código no es correcto, pero esperamos que ayude al lector a entende mejor como se cargan los datos.

- transform.py: Código de Python utilizado para convertir datos iniciales del código Ollin-axis a datos tridimensionales en coordenadas cartesianas.