Proyecto de Modelado Matemático I:
Visualización 3D en VisIt

Paula Camila Wandurraga Sanabria

Maestría en Matemática Aplicada
Grupo de Investigación en Relatividad y Gravitación
Universidad Industrial de Santander

1 Introducción

En el presente trabajo se hace un proyecto integrando Python y bash para realizar gráficas en tres dimensiones en VisIt. La herramienta creada toma los datos en bruto que arrojan las simulaciones y las separa por tiempo y por variable de interés para que el visualizador la pueda leer y procesar. Se escogió VisIt como visualizador ya que permite manejar grandes cantidades de datos, tiene diferentes opciones de visualización y genera figuras con la calidad suficiente para introducirlas en artículos.

2 Objetivos

2.1 Objetivo General

Visualizar con VisIt los datos 3D resultantes de simulaciones ealizadas con el código MAGNUS acerca de la propagación de ondas en la atmósfera solar.

2.2 Objetivos Específicos

- Conocer los formatos de datos que recibe VisIt.
- Crear un convertidor de formato que pase los datos de las simulaciones a un formato válido para VisIt.
- Escribir un script en bash para hacer más eficiente el proceso.
- Visualizar los datos por bloques en VisIt.

3 Planteamiento del Problema

El código CAFE Relativista [Lora-Clavijo et al., 2015], es un código magnetohidrodinámico que modela procesos de acreción, jets, GRBs, además de propagación de ondas magnetohidrodinámicas resistivas en la atmósfera solar. De este código nació MAGNUS [Navarro, 2017] el cual resuelve las ecuaciones de la magnetohidrodinámica en el régimen no lineal, incluyendo términos de resistividad y flujo de calor. Mi proyecto de pregrado se basó en simulaciones realizadas con MAGNUS. Este código arroja un único archivo de datos 3D que contiene todas las variables (malla, densidad, componentes de la velocidad, presión, componentes del campo magnético) para todos los pasos de tiempo que se simularon.

Anteriormente, los datos eran extraídos por bloques de tiempo con un código en Fortran y visualizados con un código de Python para MayaVi. El proceso de visualización en MayaVi se complica, ya que la aplicación se buggea fácilmente y se pierde el trabajo hecho en la sesión. Por esta razón, se decidió cambiar el visualizador de datos 3D. En medio de la búsqueda, se quiso aprender a visualizar en VisIt, ya que es ampliamente utilizada en el ámbito científico, más específicamente en física solar. VisIt es una herramienta de código libre, está diseñada para soportar datos de gran tamaño,

permite visualizar campos escalares y vectoriales y realizar operaciones entre conjuntos de datos.

VisIt recibe una gran cantidad de formatos de texto, como archivos AUX, HDF4, HDF5, AMR, M3D, OVERFLOW, algunos tipos de archivo ASCII, entre otros. Los datos salvados en las simulaciones están en un formato ASCII no compatible con VisIt, por tanto, se podía cambiar la manera de salvar datos y volver a realizar las simulaciones o traducir los datos a alguno de los formatos válidos para VisIt. Debido a que las simulaciones que actualmente se están trabajando, son un grupo de 18 corridas que demoran aproximadamente 2 días cada una, no es viable volver a realizarlas, es mejor fabricar el convertidor de formatos.

4 Código

Inicialmente, la idea era crear un convertidor de ASCII a HDF5 para poder visualizar. Este código (ver figura 1) ya estaba hecho, se encontró en el siguiente enlace: Convertidor de ASCII a HDF5, y funciona. Aun así, no fue satisfactorio para el proyecto, ya que el sabor de HDF5 que arroja como resultado el código, no está entre los posibles formatos de datos para VisIt.

```
from __future__ import print_function
import os
from astropy.io import ascii

def ascii2hdf5(inputfile, outputfile, clobber=False, overwrite=True,
               verbose=False):
    """Convert a file to hdf5 using compression and path set to data"""
    if verbose:
        print('converting {} to {}'.format(inputfile, outputfile))

    tbl = ascii.read(inputfile)
    try:
        tbl.write(outputfile, format='hdf5', path='data', compression=True,
                  overwrite=overwrite)
    except:
        print('problem with {}'.format(inputfile))
        return

    if clobber:
        os.remove(inputfile)
        if verbose:
            print('removed {}'.format(inputfile))

    return
```

Figure 1: Código que toma un *inputfile* en formato ASCII y lo convierte en un *outputfile* en formato HDF5.

Como solución alternativa, se hizo un código basado en un código de Python que genera archivos ASCII compatibles con el visualizador: Generador de archivos en formato permitido. El código creado funciona como extractor y convertidor al mismo tiempo, tomando los datos en bruto y generando datos organizados que VisIt pueda leer.

En la figura 2 se muestran las primeras líneas de código. Se importan las librerías necesarias (*sys* y *numpy* en este caso), y se abre un archivo llamado *1input_plots.par*, el cual es arrojado por la simulación, donde se encuentran los parámetros de la simulación, como número de puntos espaciales y temporales, dominio de la simulación, número de courant y otros. Y posteriormente se calcula la resolución espacial y temporal.

```
import sys
import numpy as np

# Se importa el archivo con los datos correspondientes
# a las dimensiones y tiempos de las corridas
filedata = open('1input_plots.par', 'r')

lines = filedata.readlines()
# Se asignan la variables del archivo filedata en
# las variables correspondientes
xmin = float(lines[1].split('=')[1])
xmax = float(lines[2].split('=')[1])
ymin = float(lines[3].split('=')[1])
ymax = float(lines[4].split('=')[1])
zmin = float(lines[5].split('=')[1])
zmax = float(lines[6].split('=')[1])
Nxx = int(lines[7].split('=')[1])
Nyy = int(lines[8].split('=')[1])
Nzz = int(lines[9].split('=')[1])
Ntt = int(lines[10].split('=')[1])
courant = float(lines[11].split('=')[1])
every3D = float(lines[14].split('=')[1])

# Se cierra el archivo
filedata.close()

# Se calculan los pasos espaciales y temporales
dx  = (xmax - xmin)/float(Nxx)
dy  = (ymax - ymin)/float(Nyy)
dz  = (zmax - zmin)/float(Nzz)
dt = courant * min(dx,dy,dz)
```

Figure 2: Primeras líneas del código de Pyton, donde se leen los parámetros asociados con la simulación.

Luego, como se observa en la figura 3, se abre el archivo de los datos relacionados con las 8 variables anteriormente mencionadas en cada punto de la malla. Se crea una función *extraer* con una entrada llamada *parametro* que se compone de dos partes: la primera es la abreviación de la variable que se quiere extraer y la segunda parte es el bloque de tiempo que se quiere extraer. Se calcula el *skip*, que es el número de líneas que se deben saltan para llegar al bloque de tiempo requerido y se saltan esas líneas. Por útlimo, se crea el archivo de salida cuyo nombre se compone de la variable y el bloque de tiempo y en este se escribe la línea de encabezado solicitada por VisIt, que corresponde a los nombres de las columnas (malla + variable).

```
# Se importa el archivo con los datos de las corridas correspondientes a las variables
# densidad, velocidad en x, y, z, presion y campo magnetico en x, y, z
f = open('primitivas_1.xyzl', 'r')

# Se crea la funcion 'extraer' para tomar determinado tiempo y variable del archivo f
def extraer(parametro):

    # Se leen las entradas para reconocer la variable y tiempo a extraer
    variable = str(parametro.split('-')[0])
    bloque = format(int(parametro.split('-')[1]), '02')

    # Se calcula el tiempo real que se esta extrayendo
    t = dt*float(bloque)*every3D

    # Se valida la infomacion acerca de la variable, el bloque de tiempo y el tiempo
    # real que se esta extrayendo
    print 'Extrayendo la variable',variable,'en el bloque',bloque,'en',t, 'segundos.'

    # Se saltan las lineas correspondientes a los tiempos anteriores al que se requiere
    skip = int((((Nyy+2)*(Nxx+1)+1)*(Nzz+1)+4)*int(bloque))
    print 'Se saltan', skip, 'lineas.'
    for a in range (0,skip+3):
        f.readline()

    # Se crea un archivo nuevo llamado por el nombre de la variable y el bloque a extraer
    w = open('%s_b%s.3D'%(variable,bloque), 'wt')
    # Se escribe el encabezado necesario para que VisIt pueda leer los archivos ASCII
    w.write('x y z %s\n'%variable);
```

Figure 3: Líneas de código Python preparando los documentos de salida y buscando el bloque requerido.

Después de haber creado el archivo y escrito el encabezado, en la figura 4 se muestra la tercera parte del código, donde se leen las líneas del archivo de datos tal y como se salvaron, se extraen las variables en cada línea y se escribe la variable de interés en el archivo de salida junto con la malla.

```
# Se leen las lineas de la misma manera como fueron salvadas y se reescriben
# los datos requeridos en el archivo w
for i in range (0,Nzz+1):
    f.readline()
    for j in range (0,Nxx+1):
        f.readline()
        for k in range (0,Nyy+1):
            z, x, y, rho, vx, vy, vz, press, Bx, By, Bz = f.readline().split()
            if (variable == 'v'):
                v = np.sqrt(float(vx)**2+float(vy)**2+float(vz)**2)
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(v)))
            elif (variable == 'B'):
                B = np.sqrt(float(Bx)**2+float(By)**2+float(Bz)**2)
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(B)))
            elif (variable == 'rho'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(rho)))
            elif (variable == 'vx'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(vx)))
            elif (variable == 'vy'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(vy)))
            elif (variable == 'vz'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(vz)))
            elif (variable == 'press'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(press)))
            elif (variable == 'Bx'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(Bx)))
            elif (variable == 'By'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(By)))
            elif (variable == 'Bz'):
                w.write('%g %g %g %g\n' %(float(x),float(y),float(z),float(Bz)))

# Se cierran ambos archivos
w.close()
f.close()
```

Figure 4: Líneas de código Python leyendo los datos y reescribiéndolos en un formato válido para visualizar con VisIt.

Finalmente, las tres líneas de la figura 5 relacionan el código de Python con bash. Permiten escribir la función que se quiere llamar (*method_name*) y los argumentos de entrada de esta (*parameter_name*) desde la línea de comando, posibilitando el uso de un script en bash.

```
# Se utilizan estas 3 lineas para llamar la funcion
# e ingresar la variable y bloque a salvar desde la terminal
method_name = sys.argv[1]
parameter_name = sys.argv[2]
getattr(sys.modules[__name__], method_name)(parameter_name)
```

Figure 5: Líneas de código Python que toma entradas desde la línea de comando para utilizarlas como argumento de la función.

La salida de este código en Python, va a ser un archivo ASCII de cuatro columnas, las tres primeras correspondientes a la malla numérica en tres dimensiones y la última es la variable en ese punto de la malla (ver figura 6), para un valor determinado de tiempo.

```
(base) paula@PaulaCW:~/prom/Graphs_ASCII$ head -25 vz_b15.3D
x y z vz
-0.5 -0.5 0 1e-14
-0.5 -0.48 0 1e-14
-0.5 -0.46 0 1e-14
-0.5 -0.44 0 1.00001e-14
-0.5 -0.42 0 1.00008e-14
-0.5 -0.4 0 1.00041e-14
-0.5 -0.38 0 1.00195e-14
-0.5 -0.36 0 1.00859e-14
-0.5 -0.34 0 1.03482e-14
-0.5 -0.32 0 1.13036e-14
-0.5 -0.3 0 1.45048e-14
-0.5 -0.28 0 2.437e-14
-0.5 -0.26 0 5.23151e-14
-0.5 -0.24 0 1.25024e-13
-0.5 -0.22 0 2.9863e-13
-0.5 -0.2 0 6.78572e-13
-0.5 -0.18 0 1.43959e-12
-0.5 -0.16 0 2.83184e-12
-0.5 -0.14 0 5.15173e-12
-0.5 -0.12 0 8.65852e-12
-0.5 -0.1 0 1.34386e-11
-0.5 -0.08 0 1.92576e-11
-0.5 -0.06 0 2.54771e-11
-0.5 -0.04 0 3.11156e-11
```

Figure 6: Archivo de salida del extractor.

Con el fin de automatizar ciertos pasos en la línea de comandos, se escribió un pequeño script en bash que:

- Remueve los archivos extraídos anteriormente.
- Recibe 2 valores correspondientes a la variable y bloque de tiempo de interés.
- Corre el extractor de datos.

- Copia el archivo de salida a la carpeta de VisIt para que pueda ser abierto en el programa.
- Sale de la carpeta y ejecuta VisIt.

De tal manera que, al ejecutar el script de bash, se realicen todos los procesos anteriores a salvar la imagen. Se decidió no salvar la imagen directamente, ya que como es una imagen 3D, debe haber un proceso subjetivo que corresponde al creador de las imágenes, donde se elige el plano en el que se ve la mayor cantidad de información relevante posible.

```
(base) paula@PaulaCW:~/prom/Graphs_ASCII$ cat myscript.sh
#!/usr/bin/env bash

rm *.3D

v=${1?Error: Ingrese otra variable}
b=${2?Error: Ingrese otro bloque}

python extractor.py extraer $v-$b

cp *.3D ../../../../../Visit/Proj/

cd
./visit
```

Figure 7: Script en bash que permite extraer los datos, copiarlos a la carpeta correspondiente y ejecutar VisIt.

5 Resultados

Los archivos generados por el extractor se abren con la herramienta de visualización, y luego de un proceso de filtrado de datos sin información como se observa en las figuras 8 y 9.

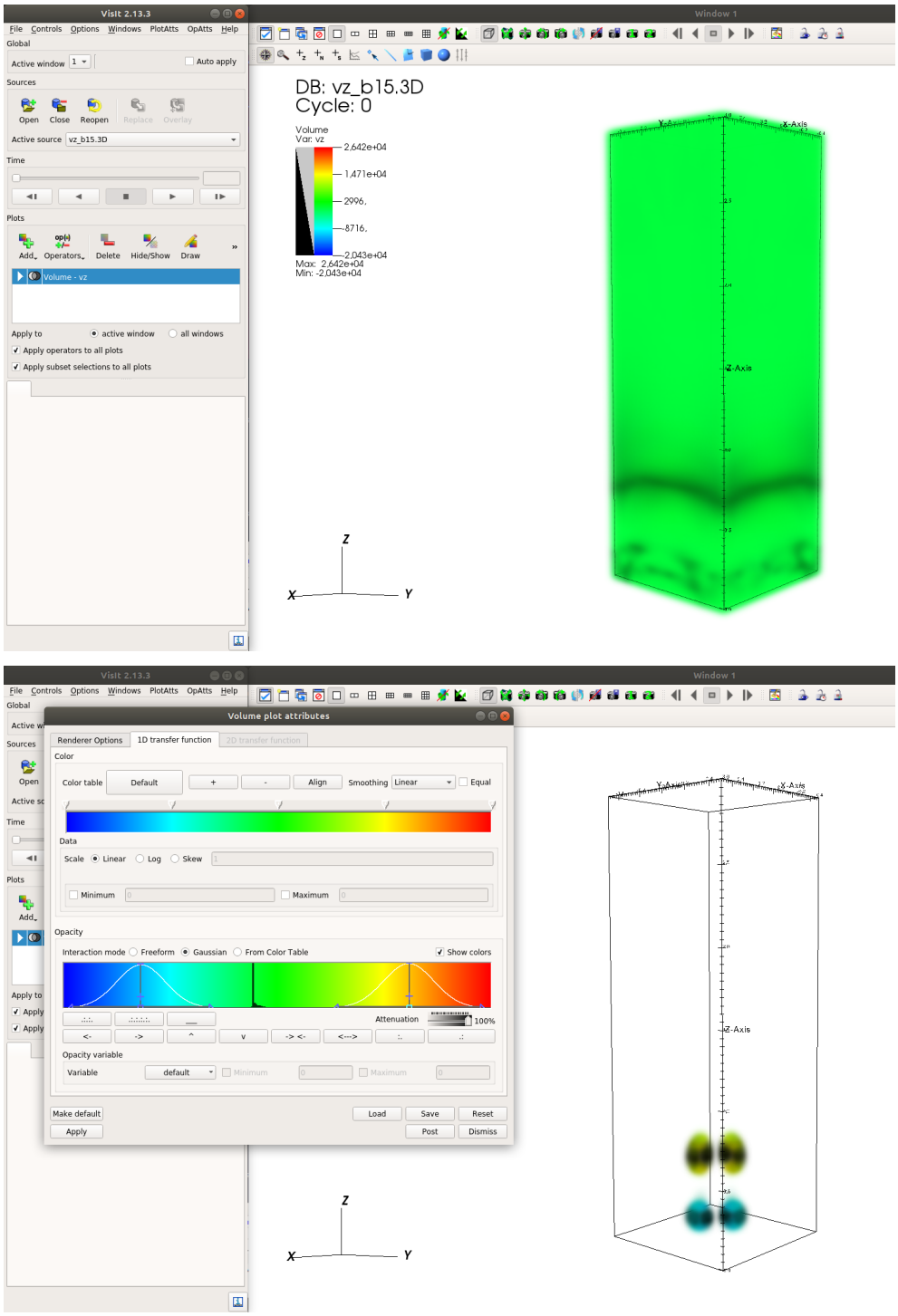

Figure 8: Gráfica 3D de la componente z de la velocidad en el bloque 15, antes y después de filtrar los datos.

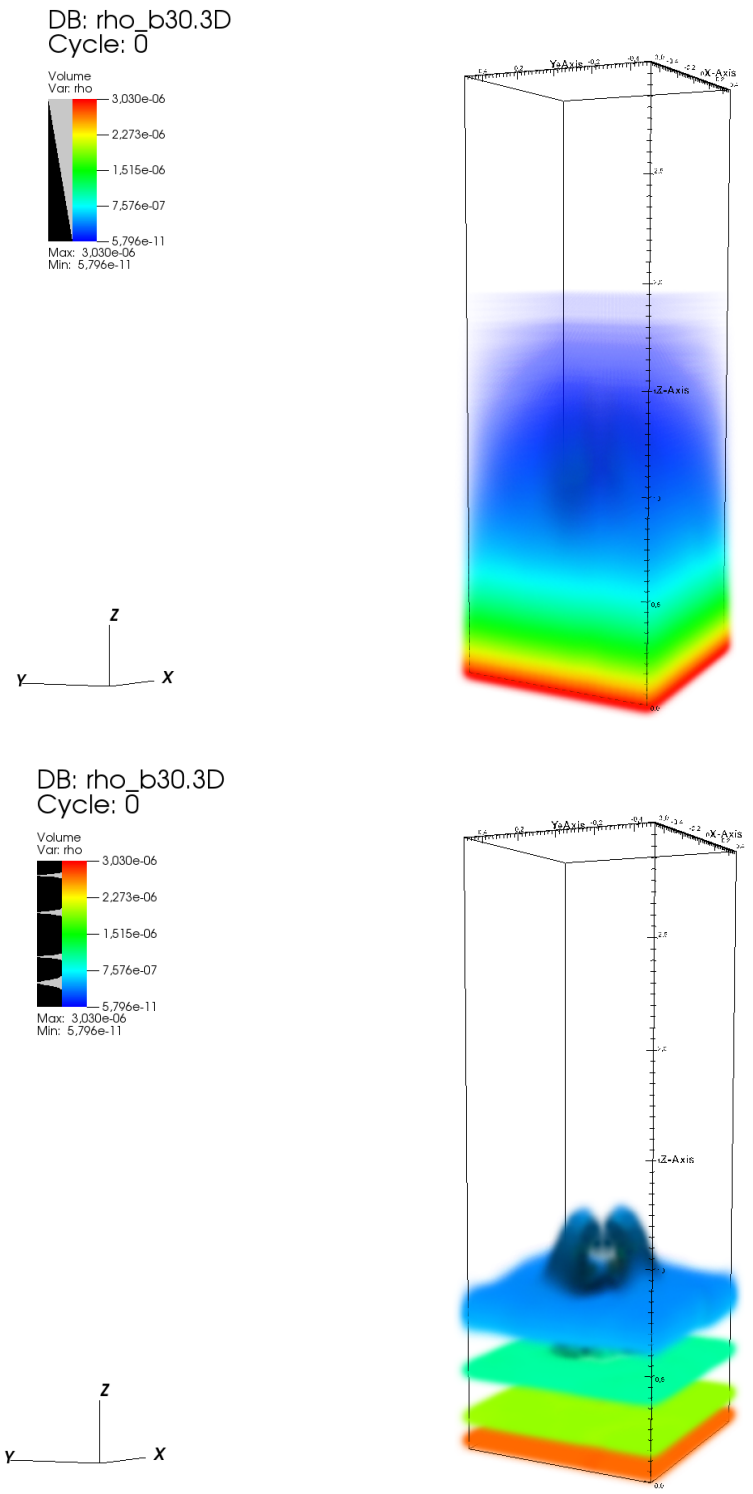

Figure 9: Gráfica 3D de densidad en el bloque 30, antes y después de filtrar los datos.

Se observa la importancia de tener una herramienta poderosa como VisIt al momento de analizar datos de simulaciones, ya que permite encontrar la verdadera morfología de las variables a analizar.

6 Conclusiones

Se cumplió safisfactoriamente con los objetivos trazados para el proyecto, ya que se lograron visualizar los datos del código MAGNUS con VisIt, utilizando el script de bash, con lo que se aumenta la eficiencia de producción y la calidad de las gráficas de salida del código.

7 Referencias

References

F. D. Lora-Clavijo, A. Cruz-Osorio, and F. S. Guzmán. CAFE: A New Relativistic MHD Code. *Astrophys. J. Supp.*, 218:24, June 2015. doi: 10.1088/0067-0049/218/2/24.

F. D. & González G. A. Navarro, A. & Lora-Clavijo. Magnus: A New Resistive MHD Code with Heat Flow Terms. *Astrophys. J.*, 844:57, July 2017. doi: 10.3847/1538-4357/aa7a13.