Técnicas de simulación numérica

Práctica 1: Fluidos compresibles en una dimensión (parte 1)

David Martín Belda
dmbelda@gmail.com

1. El código numérico

El código que se presenta está escrito en lenguaje *Python*, y hace uso de los paquetes *numpy* y *matplotlib*.

1.1. Manejo del programa

El código está pensado para poder llevar a cabo varios experimentos en el gas monodimensional sin tener que modificar demasiado los módulos, a excepción de *initcond.py*, *exp_ctrl.py* y *bcs.py*. El primero contiene diferentes formas para la condición inicial, el segundo controla el desarrollo del experimento en curso y el tercero implementa las condiciones de contorno.

Para ejecutar el programa introducimos el siguiente comando:

```
$ python main.py exp.dat
```

donde *exp.dat* es el archivo de entrada correspondiente al experimento que se quiere hacer, cuyo formato se describe a continuación:

```
256                     nint          number of intervals
-12., -8.3              z0, zf        extremes of physical domain

6000000                 itmax         maximum number of iterations
36000000.               timefinal     maximum physical time
0 0                     plottimeinterv, plotstinterv - cadence for output

0   Draw pres?          drawpres    1=True // 0=False
0   Draw rho?           drawrho     1=True // 0=False
0   Draw vz?            drawvz      1=True // 0=False
0   Draw moving dot?    drawdot     1=True // 0=False
0   Draw moving wave?   drawwave    1=True // 0=False

sound wave              inittype    init. cond: type (25 spaces)
sine                    shape       init. cond: other param (25 spaces)
1.5e-4                  amp         amplitude parameter
1e0                     p00         equilibrium pressure
1e0                     rho00       equilibrium density
0                       vz00        overall velocity
1.6666666666666667      gam         adibatic index

0.95                    fcfl        parameter for the Courant condition
```

Parámetros de la red numérica.

nint Número de intervalos de la red.

z_0 y z_f Extremos del intervalo.

Parámetros de control del programa.

itmax Máximo número de iteraciones.

timefinal Tiempo final del experimento.

plottimeinterv Número de iteraciones por cada actualización de las gráficas.

plotstinterv Número de iteraciones por cada registro en fichero (por implementar).

Parámetros de control del *plotting*.

drawpres Si es 1 (0), se representa (no se representa) la gráfica correspondiente a la presión.

drawrho Si es 1 (0), se representa (no se representa) la gráfica correspondiente a la densidad.

drawvz Si es 1 (0), se representa (no se representa) la gráfica correspondiente a la velocidad.

drawdot Si es 1 (0), se añade (no se añade) un punto moviéndose sobre el máximo de las magnitudes representadas.

drawwave Si es 1 (0), se representa (no se representa) una onda sinusoidal moviéndose a la velocidad del sonido, *cs*00, hacia la derecha.

Parámetros del problema.

initype Tipo de la condición inicial (sound wave).

shape Forma de la condición inicial (sine).

amp Amplitud de la perturbación.

*p*00 Presión de equilibrio.

$rho00$ Densidad de equilibrio.

$vz00$ Velocidad del fluido.

gam Coeficiente adiabático.

Parámetros del esquema numérico.

$fcfl$ Parámetro para la condición de *Courant*

Es importante respetar las líneas en blanco y los 25 espacios en las líneas correspondientes al tipo y forma de la condición inicial.

1.2. Breve descripción de los módulos nuevos.

El código presentado es una traducción del código *IDL* facilitado a *Python*. Se han añadido los siguientes módulos:

plotrout: Se encarga de hacer las representaciones gráficas.

initcond: Contiene la forma de la condición inicial.

read_input: He decidido colocar la lectura de datos en un módulo aparte.

routines: Importa los módulos necesarios en el módulo __main__

exp_ctrl: Controla cuándo finaliza un experimento, la salida en pantalla...

2. Primer experimento: Ondas de sonido

2.1. Experimento básico

Ficheros de entrada: *exp1a.dat*, *exp1b.dat* y *exp1.dat*

exp1a.dat: Las condiciones iniciales para ρ, p y v_z son sinusoides, tal y como se describe en el guión. Empezamos representando las tres magnitudes, y usando 256 intervalos de red. Tras dejar correr unos cuantos periodos no se aprecia una difusión fuerte, al menos comparando de forma visual con la solución analítica. Bajando el número de celdas hasta 16[1], se comprueba que, en efecto, hay difusión numérica. Se observa también que el código introduce un error de desfase respecto de la solución analítica.

exp1b.dat: El archivo de entrada es una copia de *exp1a.dat*, salvo porque en éste se pide al programa que represente el punto en lugar de la solución analítica. El resultado es el mismo; para 256 celdas, pasados unos periodos, no se aprecia que el máximo se distancie verticalmente del punto. Bajando el número de puntos de red el efecto se hace más y más notorio.

exp1c.dat: Se intenta comprobar con algo más de rigor si existe difusión numérica. Para ahorrar tiempo, pedimos en el archivo de entrada que se realicen las representaciones gráficas inicial y final sólamente (dando un valor muy grande a *plottimeinterv*). En el módulo *exp_ctrl* especificamos que el programa detenga el experimento cuando el error provocado por la difusión numérica rebase cierto umbral. Por ejemplo, podemos pedir que el programa pare el código cuando la amplitud haya caído un 5 % en la siguiente línea en *exp_ctrl-py*:

```
fraclim=0.01
```

Ejecutando el programa con 64 intervalos:

```
$ python main.py exp1c.dat
```

obtenemos la sigiuente salida en pantalla:

[1]Este número no cuenta las dos celdas *guarda* usadas en el código numérico.

```
Experiment exp1c.dat on course...
Taking 64 intervals between -12.0 and -8.3.


Experiment 1c finished:
  Initial perturbation amplitude: 0.00015
  Final perturbation amplitude: 0.000157504605437
  Fraction of amplitude lost by numerical diffusion: 0.0500307029115
  Tolerance: 0.05

press enter...
```

Inmediatamente llama la atención que la amplitud ha *crecido*, en contra de lo esperado. Examinamos la gráfica de la densiadad en el instante final:

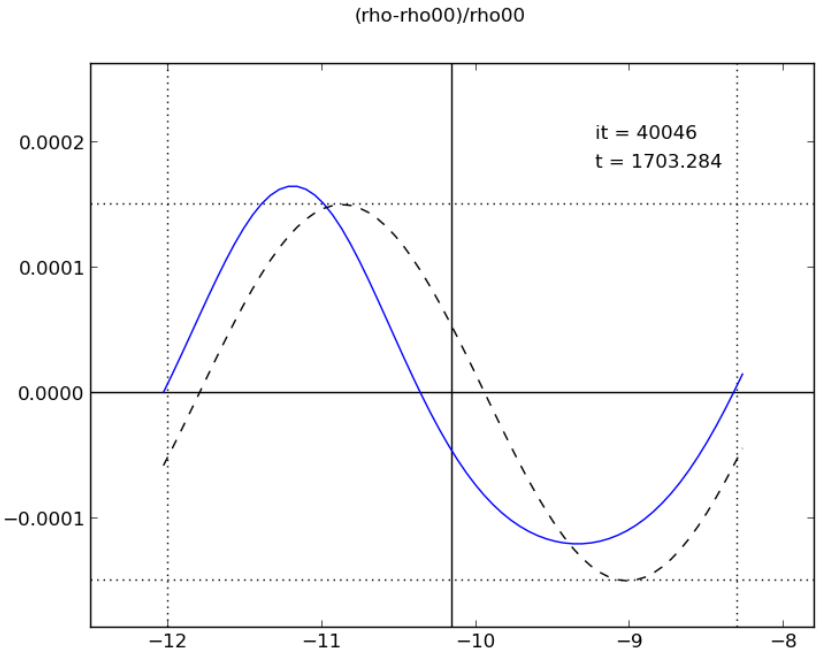

Figura 1: Deformación de la solución a baja resolución de la red.

Parece que la onda, pasado cierto tiempo, sufre una deformación (Figura 1). El lóbulo negativo se ensancha, mientras que el positivo se comprime. Esta deformación se hace menos notoria si aumentamos el número de celdas, con lo que se puede concluir que se debe a la baja resolución de la red numérica utilizada.

Para estimar correctamente la amplitud perdida por difusión numérica, debemos tomar la amplitud completa, es decir, la diferencia entre el máximo y el mínimo de la onda. Las líneas de *exp_ctrl.py* que se encargan de esto son:

```
# Fraction of amplitude lost by numerical diffusion:
maxrho=np.max((rho-rho00)/rho00)
minrho=np.min((rho-rho00)/rho00)
frac=np.abs((2.*amp-(maxrho-minrho))/2./amp)
fraclim=0.05
```

Repitiendo el experimento, obtenemos:

```
Experiment exp1c.dat on course...
Taking 64 intervals between -12.0 and -8.3.
press enter...

Experiment 1c finished:
  Full initial perturbation amplitude: 0.0003
  Full final perturbation amplitude: 0.000284994771763
  Fraction of full amplitude lost by numerical diffusion: 0.0500174274581
  Tolerance: 0.05

press enter...
program finished.
     it=40046
   time=1703.28409096
```

Dejando correr un tiempo razonable el programa, con un número suficiente de puntos, queda mucho más claro el efecto de la no linealidad; las partículas más rápidas van ganando terreno a las más lentas, y el seno va adoptando una forma más bien de diente de sierra (Figura 2). Esto ya se vio cuando estudiamos la ecuación de Burgers; si se deja pasar el tiempo suficiente, se generará una discontinuidad.

Si lo dejamos correr mucho tiempo, la onda se acabará deformando severamente. Lo más probable es que, a diferencia de la tendencia hacia el diente de sierra, este efecto sea debido a algún la inestabilidad a tiempos largos del código. El hecho de que la amplitud total crezca implica que la onda gana energía, lo cual no tiene sentido físico. Se estudiará esto con más detalle en el apartado 5.

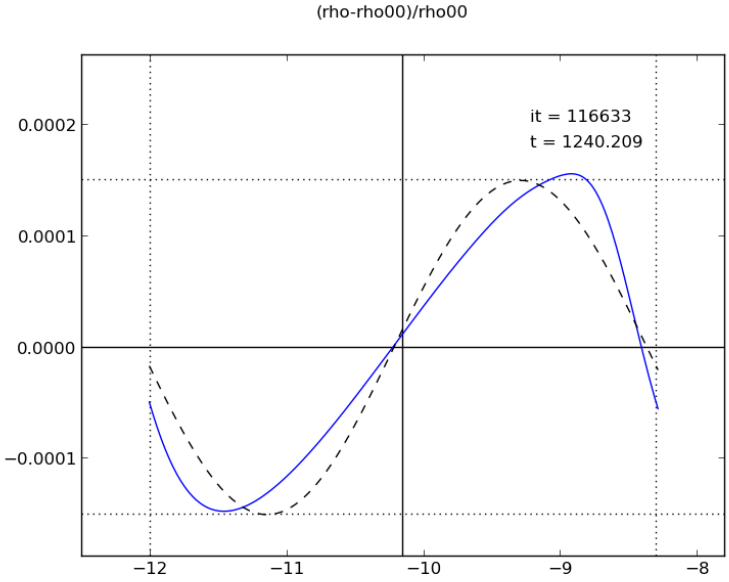

Figura 2: 256 celdas, 116633 iteraciones. El seno se va pareciendo a un diente de sierra.

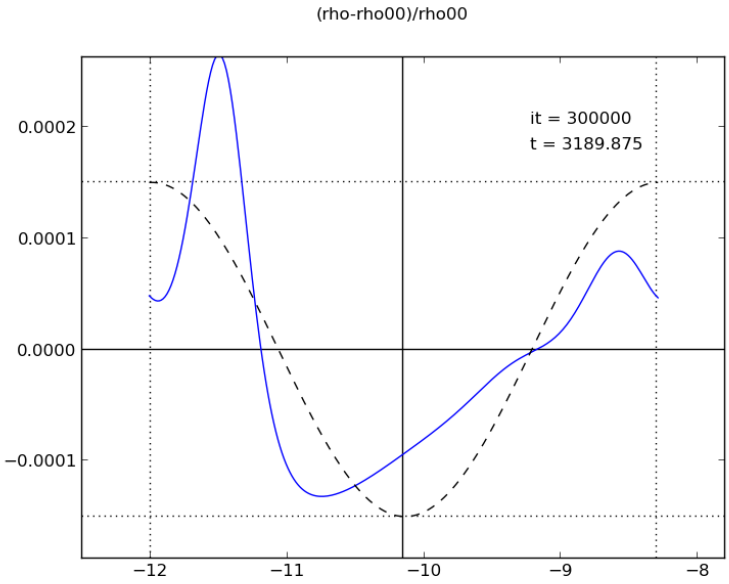

Figura 3: 256 celdas, 601903 iteraciones

Una de las mejoras en el código que sugiere esta sección es introducir la posibilidad de representar las soluciones en la misma gráfica, a escala, para poder ver si se produce desfase entre ellas.

Nota sobre las unidades:

La salida del programa da el número de iteraciones y el tiempo *computacional* en el que acaba el experimento. Por tiempo computacional entiendo aquel que uso en el código para resolver las ecuaciones.

Por ejemplo, estamos usando $\rho_{00} = 1$ y $p_{00} = 1$. Si suponemos que el fluido que estamos tratando se trata de un gas en condiciones ordinarias, es razonable pensar que la densidad viene dada en miligramos y la presión en bares. A partir de la fórmula para la velocidad podemos sacar las unidades de tiempo en este caso:

$$cs_0 = \sqrt{\gamma}\sqrt{\frac{1\,bar}{1\,mg}} = \sqrt{\gamma} \times 10^{9/2}\,cm/s$$

Es decir, si damos el dominio en cm, las unidades de tiempo serán $10^{-9/2}\,s$. Tomando la última salida del programa, encontraríamos que la solución se ha difundido un 5 % después de un tiempo computacional de 1703.28 unidades, que corresponde a un tiempo físico de $1703{,}28 \times 10^{-9/2}\,s = 0{,}054\,s$.

2.2. Equilibrio en el movimiento.

Ficheros de entrada: *exp2.dat*

Modificamos la línea correspondiente en el fichero de entrada para dar al fluido una velocidad inicial, y dibujamos la onda que se mueve con velocidad cs_{00} respecto de los ejes con una línea punteada (onda superpuesta). Si damos al fluido una velocidad de $-cs_{00}/4$, la velocidad de la onda olución será, respecto de los ejes, $3/4 \times cs_{00}$. Visualmente, vemos pasar 4 ondas superpuestas por cada tres ondas solución. Lo mismo pasa si damos al fluido una velocidad de $0{,}9\,cs_{00}$, pero en este caso la onda solución viajará más rápido respecto de los ejes que la onda superpuesta. En el caso límite, cuando la velocudad inicial del fluido es $-cs_{00}$, vemos la onda solución estática en la pantalla (lo que no quiere decir que el programa no esté haciendo nada; está resolviendo las ecuaciones del fluido todo el tiempo).

2.3. Cambiando las fases.

Ficheros de entrada: *exp1a.dat*

Con la ayuda del programa, se comprueban los siguientes hechos sobre las condiciones iniciales:

a.- Si la densidad y la presión están en fase, la forma de la onda se conserva.

b.- Si la densidad y la presión están en contrafase, la amplitud de la onda crece y decrece.

c.- Si ambas están en fase con la velocidad, la onda se propaga hacia la derecha.

d.- Si ambas están en contrafase con la velocidad, la onda se propaga hacia la iquierda.

Es fácil entender de forma intuitiva que, en el caso *c* la onda se propague hacia la derecha fijándonos en el siguiente esquema:

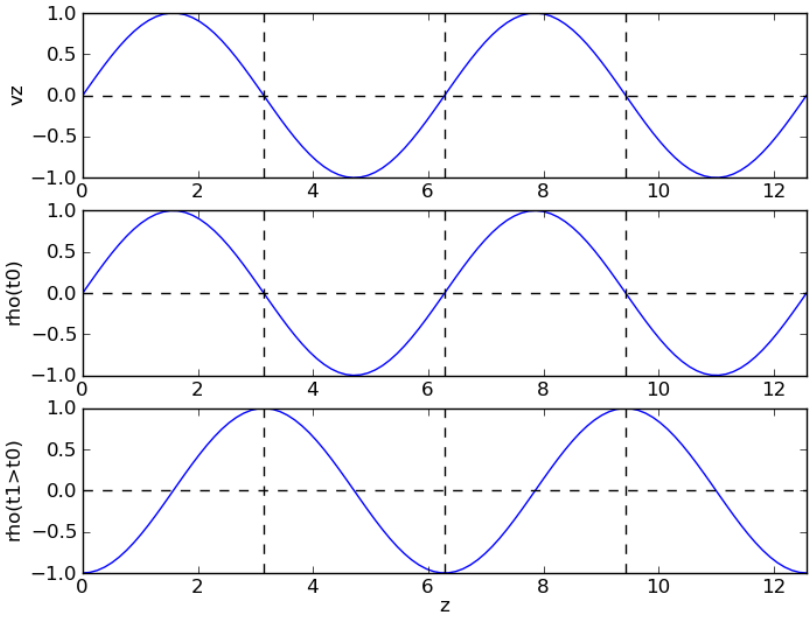

Figura 4: Densidad y velocidad iniciales en fase

Las líneas punteadas marcan, en las dos gráficas superiores, los puntos en los que la presión es la del fluído en equilibrio. Si nos fijamos en la línea vertical punteada de la izquierda en dichas gráficas, vemos que estos puntos tienen materia moviéndose hacia ellos. Tienden a concentrarla. En el punto marcado por la segunda línea punteada ocurre justamente lo contrario: el material a su izquierda se mueve hacia la izquierda, y el de su derecha se mueve hacia la derecha. De esta manera, pasado un tiempo, la materia se habrá concentrado en los primeros y escapado de los segundos, reproduciéndose el patrón que se muestra en la figura inferior. Cuando la velocidad y la densidad están inicialmente en contrafase (caso d), ocurre lo contrario, como se puede ver fácilmente en el diagrama correspondiente:

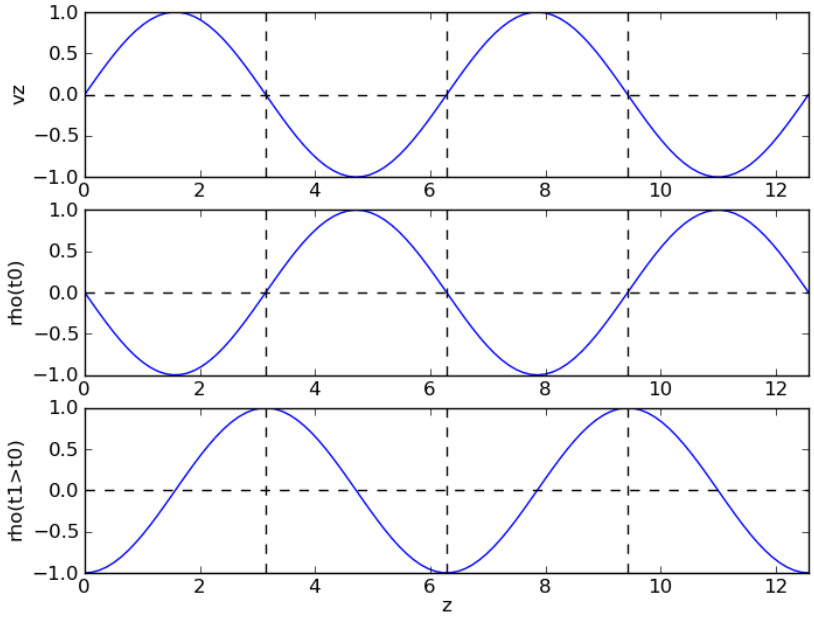

Figura 5: Densidad y velocidad iniciales en fase. La materia escapa de los puntos con un máximo de densidad a la izquierda, y va hacia los que tienen un máximo a la derecha, dando como resultado la propagación del tren de ondas hacia la izquierda.

2.4. Perturbaciones de forma arbitraria

Ficheros de entrada: *exp4.dat*

Probamos la forma de la perturbación dada en el guión de la práctica. Para ello, cambiamos el archivo de entrada de tal manera que en la línea correspondiente a la forma de la condición inicial (*shape*) figure *test 1*. Representamos cada 5 iteraciones para ver la animación. Se observa que los pulsos no pierden su forma a tiempos cortos. Esto es debido a que todas las frecuencias se propagan con la misma velocidad en aproximación lineal. Si consideramos la onda como una superposición de ondas elementales (armónicas), es fácil entender que si todas se propagan con la misma velocidad la superposición lienal no sufrirá deformaciones (en cada punto del pulso la suma de las señales se conserva). Si dejamos pasar un tiempo suficiente, el pulso se deforma debido a que se acumulan los efectos no lineales. La relación de dispersión se hace no lineal, y, además, la energía de la onda se va distribuyendo en otros modos, con lo que la superposición no coincide con la inicial.

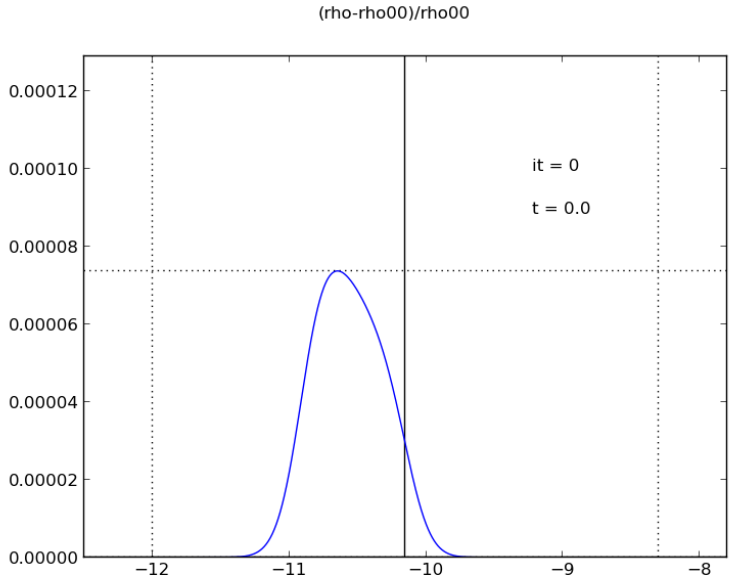

Figura 6: Forma de pulso arbitraria

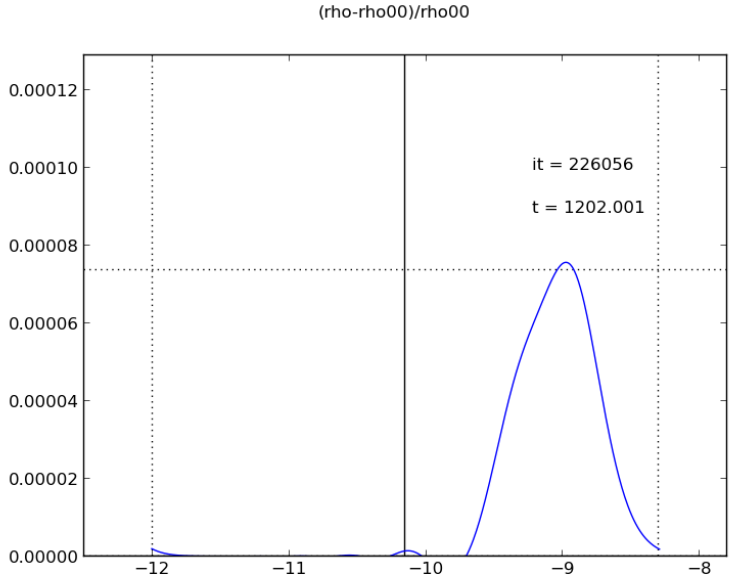

Figura 7: Forma del pulso tras un tiempo computacional t=1200 uds. 512 celdas.

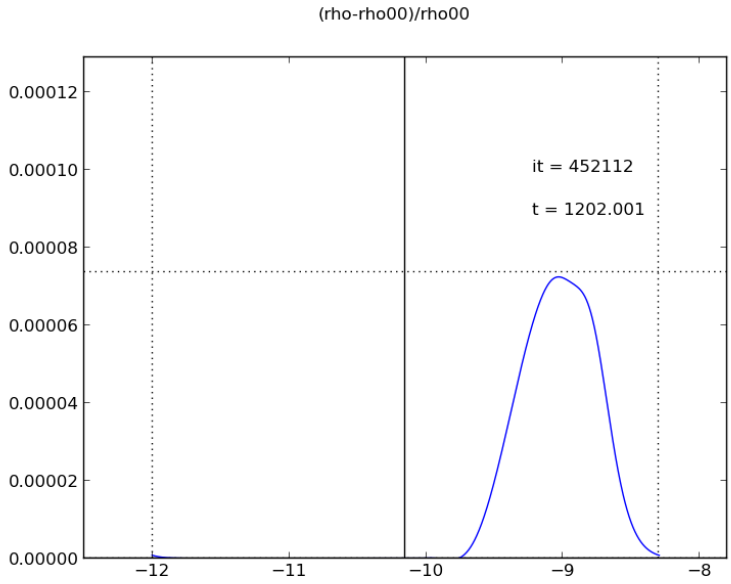

Figura 8: Forma del pulso tras un tiempo computacional t=1200 uds. 1024 celdas.

La forma del pulso 3 confirma que el aumento de amplitud observado en los experimentos anteriores se debe a la baja resolución de la red. En esta figura se aprecia la difusión numérica. Para el mismo tiempo, la amplitud con 512 puntos ha crecido.

2.5. Estabilidad del esquema numérico

Ficheros de entrada: *exp5.dat*

Tomamos una red con 1024 puntos, una forma de onda inicial sinusoidal, y empezamos probando con un valor para el parámetro $fcfl = 2$. El código explota en seguida (a las 21 iteraciones):

```
/home/david/master/3/tsn/mod1/prac/cfl.py:20: RuntimeWarning: invalid value encountered in sqr
  cs = np.sqrt(par.gam*pres/rho)
program finished.
     it=21
   time=nan
```

Si bajamos a 1.5, no tarda mucho más en explotar (a las 38 iteraciones):

```
/home/david/master/3/tsn/mod1/prac/cfl.py:20: RuntimeWarning: invalid value encountered in sqr
  cs = np.sqrt(par.gam*pres/rho)
program finished.
     it=38
   time=nan
```

Con el parámetro de *Courant* a 1, el código es estable. Además, con 1024 puntos, en la figura no se aprecia difusión numérica ni otra deformación que no sea la de la tendencia al diente de sierra debido a la no linealidad:

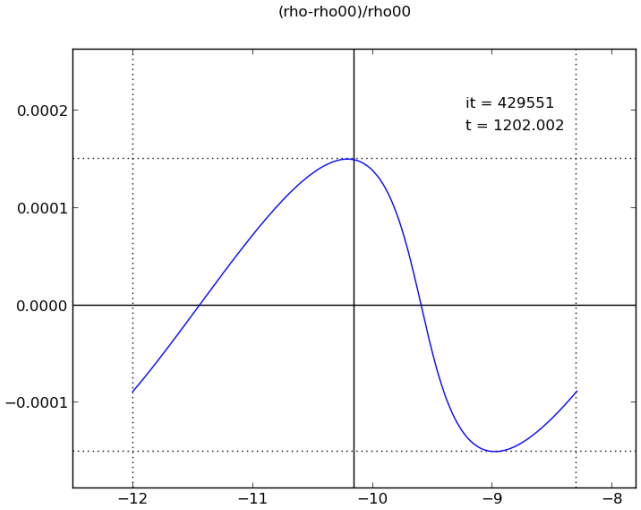

Figura 9: 1024 celdas, fcfl=1.

Si subimos a 1,25, el código no explota, pero la representación gráfica deja claro que existe inestabilidad:

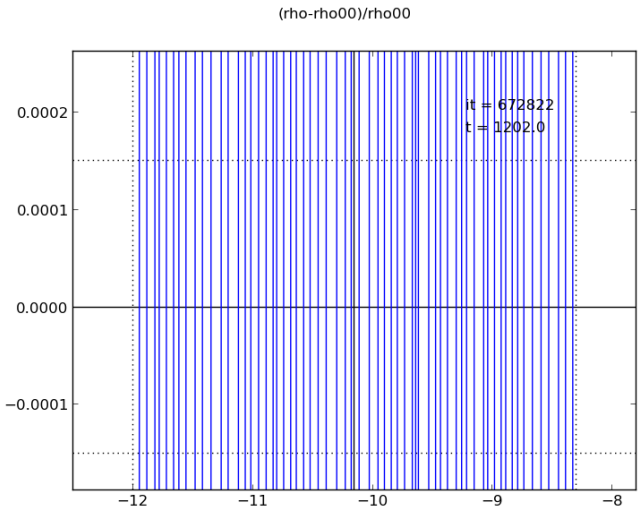

Figura 10: 1024 celdas, fcfl=1.

Lo mismo ocurre si bajamos a 0,5...

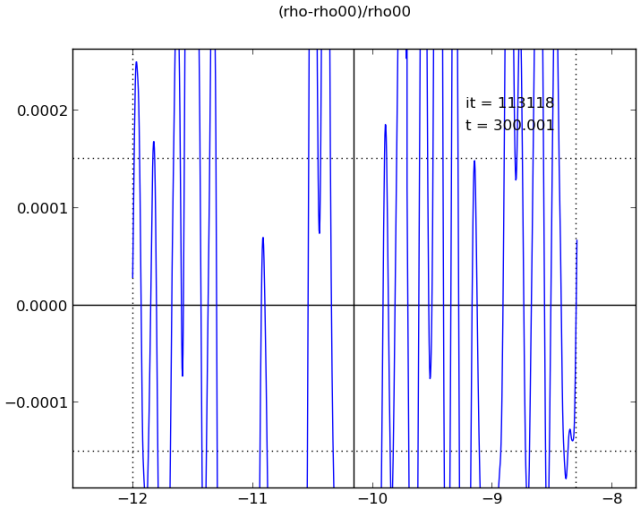

Figura 11: 1024 celdas, fcfl=1.05

... e incluso a 0.1:

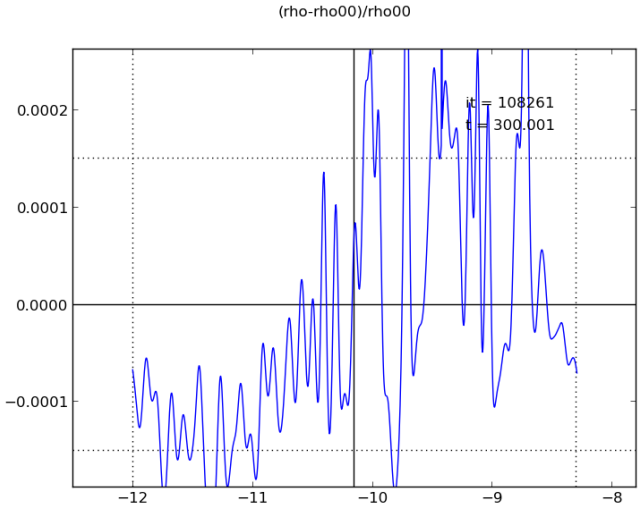

Figura 12: 1024 celdas, fcfl=1.01

de donde se concluye que el umbral de estabilidad del código es $fcfl = 1$, o que éste está en un valor del parámetro cfl entre 1 y 1,01.