Taller 14: Inversión de Onda Completa (FWI): Actualización de velocidades mediante Steepest Descent y avance óptimo usando L-BFGS.

Juan Carlos Gutiérrez Barrera

Universidad Industrial de Santander

7 de junio de 2022

1. Introducción

La inversión de onda completa es una minimización repetitiva compleja entre la información observada y la calculada. Los componentes escenciales para la inversión de onda completa son: la función de costo para medir la diferencia entre los datos observados y los modelados, un propagador de onda para calcular los datos modelados y el modelo de velocidades inicial que es actualizado iterativamente hasta que la función de costo encuentra un valor adecuado (Abreo et al., 2015).

El método BFGS fue propuesto por Broyden, Fletcher, Goldfarb y Shanno en 1970 y es una fórmula de actualización Quasi-Newton que permite obtener el mínimo de una función. Por otro lado, el método L-BFGS se trata de un método que hace un uso limitado de la memoria (usa mucha menos memoria que otros algoritmos para el mismo problema). Únicamente necesita la función y su gradiente, pero no la matriz Hessiana (Acevedo Vázquez, 2019).

Se tiene un operador hacia adelante que parte de la ecuación de propagación de una onda acústica isotrópica:

$$\frac{1}{c^2}\frac{\partial^2 p(x,z)}{\partial t^2} = \frac{\partial^2 p(x,z)}{\partial x^2} + \frac{\partial^2 p(x,z)}{\partial z^2} + src(x,z,t) \tag{1}$$

y un operador hacia atrás obtenido a partir de las ecuaciones adjuntas y el gradiente:

$$\frac{1}{c^2(x,z)}\frac{\partial^2 \lambda(x,z)}{\partial t^2} = \frac{\partial^2 (x,z)}{\partial x^2} + \frac{\partial^2 (x,z)}{\partial z^2} + \frac{\partial f}{\partial s} \tag{2}$$

donde los residuales se inyectan en la posición de los receptores:

$$\frac{\partial f}{\partial s} = \sum_{R}^{N_R} \left(s(y_R) - d(y_R)\right) \tag{3}$$

Usualmente la función de costo utilizada es la función de error de mínimos cuadrados dada por la norma L2:

$$f(s,m) = \frac{1}{2}\sum_{R}^{N_R} \|s(y_R) - d(y_R)\|_2^2 \tag{4}$$

Una actualización para el modelo de velocidad puede ser obtenita usando el método steepest descent como:

$$c^{k+1} = c^k - \alpha \cdot g(c^k) \tag{5}$$

donde $g(c^k)$ es el gradiente de la función de costo, el cual está dado por:

$$g(c^k) = -\sum_{s} \frac{2}{c^k(x,z)^3} \int_0^T \lambda(x,z) \frac{\partial^2 p_s(x,z)}{\partial t^2} dt \tag{6}$$

donde λ es el campo de onda adjunto obtenido a partir de una propagación que utiliza como fuente el error entre los datos modelados y observados, en tiempo inverso. Por esta razón λ es conocido como el campo propagado hacia atrás del residual o también como campo adjunto. Al contrario de la ecuación de onda, en la posición de cada receptor se inyecta una fuente cuya firma es un residual, por lo que la fuente original desaparece.

1.1. FWI como problema de optimización

La ecuación 5 indica el funcionamiento del máximo descenso (Steepest descent) que se basa en el gradiente (máximo), en el gradiente (mínimo) y en α que indica cuanto se avanza. Este avance puede ser optimizado escribiendo la ecuación 5 de la forma:

$$c^{k+1} = c^k - \alpha [H(c^k)]^{-1} \cdot g(c^k) \tag{7}$$

donde la inversa de la Hessiana permite obtener la forma de la superficie de la función de costo. Lo malo de esto es el elevado costo computacional que implica calcular una matriz Hessiana y se le suma que se debe calcular su inversa. La aproximación de $[H(c^k)]^{-1} g(c^k)$ se llama Quasi-newton y el método L-BFGS permite obtener esta aproximación a partir del histórico del los últimos 'L' gradientes y modelos donde $2 \leq L \leq 20$. Esto reduce el costo computacional ya que permite obtener directamente el producto entre la matriz Hessiana y el vector gradiente. El siguiente es el Pseudocódigo propuesto por para L-BFGS:

Algoritmo L-BFGS

- $\mathbf{q} \leftarrow \boldsymbol{g}_k$
- **for** $i = k-1, k-2, ..., k-m$ **do**
  - $\varepsilon_i \leftarrow \sigma_i \boldsymbol{s}_i^T \boldsymbol{q}$;
  - $\boldsymbol{q} \leftarrow \boldsymbol{q} - \varepsilon_i \boldsymbol{y}_i$
- **end for**
- $\mathbf{r} \leftarrow \boldsymbol{D}_k^0 \boldsymbol{q}$;
- **for** $i = k-m, k-m+1, ..., k-1$ **do**
  - $\beta \leftarrow \sigma_i \boldsymbol{y}_i^T \boldsymbol{r}$;
  - $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{s}_i(\varepsilon_i - \beta)$;
- **end for**

En este algoritmo $s_k = v_{k+1} - v_k, y_k = g_{k+1} - g_k y \sigma_k = 1/y_k^T s_k$, donde el primero es una resta de velocidades, el segundo una resta de gradientes, y σ los relaciona a los dos. La matriz D_k^0 es aproximada por $D_k^0 = \gamma_k I$ con:

Pseudocódigo Intentos L-BFGS

- $\alpha = 2$
- $\boldsymbol{h}_k = r$;
- Compute $\phi_{(\boldsymbol{v}_k)}$;
- $\phi_(attempt) = \phi_(attempt) + 100$;
- **while** $\phi_(attempt) > \phi_{(\boldsymbol{v}_k)}$ **do**
  - $\alpha = \alpha/2$;
  - $attempt = \boldsymbol{v}_k - \alpha \boldsymbol{h}_k$;
  - Compute $\phi_{(attempt)}$;
- **end while**
- $\boldsymbol{v}_{k+1} = attempt$

Este algoritmo se ha implementado en diferentes lenguajes de programación

como Python, Fortran y C; además es posible encontrar estas implementaciones en la red.

2. Actividad

Revisar los archivos de **lbfgs** compartidos por el profesor, llevarlos a Python, y verificar su correcto funcionamiento con la implementación para FWI del máximo descenso.

2.1. Solución

Para la implementación de L-BFGS sobre el código, se trabajó con la librería Scipy, la herramienta llamada optimize, la cual cuenta con diferentes módulos programados para minimizar funciones a partir de una gran cantidad de métodos. En este caso se probaron las variantes optimize.minimize especificando el método L-BFGS-B y optimize.fmin l bfgs b que trabaja directamente con el método L-BFGS utilizado en este taller.

Lo que estos métodos necesitan como dato de entrada es una función que retorne el valor de la función de costo en un punto y el valor inicial, la función gradiente es opcional ya que los algoritmos de minimización traen funciones que implementan por defecto si no se ha especificado una, sin embargo este caso requiere de la especificación de una función que retorne el gradiente, que puede ir incluida en la función que retorna la función de costo o puede ser ingresada en otro de los argumentos de la herramienta de optimización. Las pruebas se hicieron usando las dos funciones por aparte.

En el script **funcfwi.py** se encuentra definida la función funcfwi, que retorna el valor de la función de costo. Además de esto retorna una lista con el registro de los valores de la función de costo. En el script **gradfwi.py** se encuentra definida la función gradfwi, la cual retorna el valor del gradiente. Finalmente, se escribió un script base de nombre **l-bfgs-fwi.py** mostrado en el *Listing* 1, en donde está el código para realizar las optimizaciones junto con las gráficas del modelo de velocidades.

```
1  from scipy import optimize
2  import numpy as np
3  from funcfwi import funcfwi
4  from gradfwi import gradfwi
5  import matplotlib.pyplot as plt
6  from matplotlib import cm
7  import timeit
8
9  Nx = 210
10 Nz = 68
11 dz = 25
12 vp_ite = np.zeros((Nx,Nz))
13
14 # Modelo inicial
15 for iz in range(0,Nz):
16     vp_ite[:,Nz-iz-1] = 3172.5-0.6*dz*(iz)
17
18 vp_ite1 = np.reshape(vp_ite,(Nx*Nz), order='F')
19
20 v11,v22 = np.shape(vp_ite)
21 X,Y = np.meshgrid(np.arange(0,v22,1), np.arange(0,v11,1))
22 fig, ax = plt.subplots(subplot_kw={"projection": "3d"},figsize=(10,10))
```

```
surf = ax.plot_surface(X,Y, vp_ite ,cmap=cm.coolwarm ,linewidth=5)
plt.show()

#inicial=funcfwi(vp_ite)
#print(inicial)

#ginicial=gradfwi(vp_ite)
#print(ginicial)

#gview=np.reshape(ginicial[:],(Nx,Nz),order='F')
#v33,v44 = np.shape(gview)
#X1,Y1 = np.meshgrid(np.arange(0,v44,1), np.arange(0,v33,1))
#fig, ax = plt.subplots(subplot_kw={"projection": "3d"},figsize=(10,10))
#surf = ax.plot_surface(X1,Y1,gview,cmap=cm.coolwarm,linewidth=5)
#plt.show()

start = timeit.default_timer()
minimo=optimize.fmin_l_bfgs_b(funcfwi,vp_ite1,fprime=gradfwi,iprint=1)
#minimo = optimize.minimize(funcfwi, vp_ite1, method='L-BFGS-B',
jac=gradfwi)
stop = timeit.default_timer()

print('Tiempo L-BFGS-B:',stop-start)
print(minimo)

#mview=np.reshape(minimo[0],(Nx,Nz),order='F')
mview=np.reshape(minimo.x,(Nx,Nz),order='F')
v33,v44 = np.shape(mview)
X1,Y1 = np.meshgrid(np.arange(0,v44,1), np.arange(0,v33,1))
fig, ax = plt.subplots(subplot_kw={"projection": "3d"},figsize=(10,10))
surf = ax.plot_surface(X1,Y1,mview,cmap=cm.nipy_spectral,linewidth=5)
#surf = ax.plot_surface(X,Y,vp_ori,cmap=cm.coolwarm,linewidth=5,
alpha=0.8)
plt.show()
```

Listing 1: Código de Python empleado para modelar el avance de la propagación de onda implementando el método "L-BFGS".

Los resultados de la actualización del modelo de velocidades mediante el avance óptimo de L-BFGS se compararon con el mejor resultado del método de *Steepest descent* que se trabajó en el taller anterior.

2.1.1. L-BFGS vs Steepest descent

El tiempo para obtener el modelo de velocidades final del método L-BFGS por parte de la función **optimize.minimize** fué de 1073.0815518 segundos (17.88 minutos) y el valor mínimo al que llegó la función de costo fué de 392.46197531, realizando 21 iteraciones del algoritmo.

La Figura 1 muestra el ajuste del modelo de velocidades entre el modelo ajustado por L-BFGS y el modelo inicial, en la cual se observa la deformación del modelo de velocidades en su búsqueda de optimización al valor deseado. Por su parte, la Figura 2 muestra el ajuste del modelo de velocidades entre el modelo ajustado por Steepest descent y el modelo inicial. Cabe destacar que el valor mínimo al que llegó la función de costo fué de 9.4153, realizando 2600 iteraciones del algoritmo en un tiempo estimado de 20 minutos.

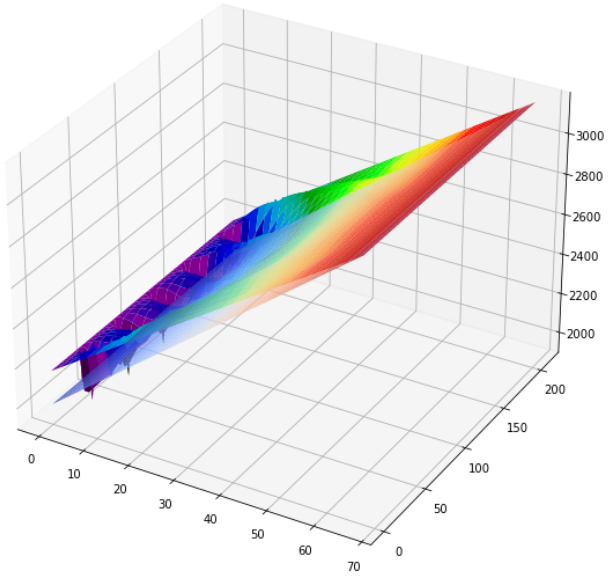

Figura 1: Modelo de velocidades obtenido mediante el método L-BFGS.

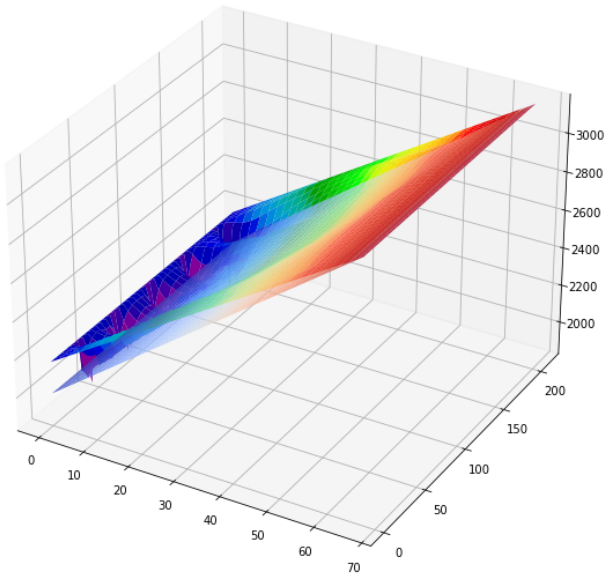

Figura 2: Modelo de velocidades obtenido mediante el método Steepest descent.

Discusión

A pesar de que se probaron los dos modulos de Scipy relacionados con el método L-BFGS, y que contaban con buenas referencias de que permiten obtener buenos resultados aproximándose a la solución de una manera óptima, lastimosamente el modulo optimize.fmin.l-bfgs-b no se logró ajustar adecuadamente al modelo de velocidades a optimizar y tuvo que ser descartado.

Por su parte, el modulo optimize.minimize logró realizar una disminución progresiva de la función de costo, llegando a valores cercanos a 390 con un avance relativamente rápido y en menos iteraciones. No obstante, la optimización sigue siendo muy inferior comparado con los resultados obtenidos con Steepest Descent, que logró minimizar mucho más la función de costo (valores aproximados a 10), aunque si se reconoce que el costo computacional disminuyó. Esto puede deberse a que al usar este algoritmo, el código puede trabajar de manera más efectiva y aprovecha de mejor manera la capacidad de la GPU.

La mejora del código del Steepest Descent con la condición que divide el beta cada vez que la función de costo aumenta, logró dar mejorar los resultados. Sin embargo, esta incrementa demasiado el costo computacional, llegando a tardar más de 20 horas en un procesamiento. Esto al final marcó la diferencia con el método L-BFGS.

Referencias

[Abreo, S. A., Silva, A., Reyes, O., Carrillo, D. L., and Alvarez, H. G.,(2015)] Abreo-Carrillo Sergio-Alberto, Ramírez Ana-B, Reyes Oscar, Abreo-Carrillo David-Leonardo and González-Álvarez Herling. A practical implementation of acoustic full waveform inversion on graphical processing units. Journal Ciencia, Tecnología y Futuro. 2015.

[Acevedo,(2019)] Acevedo Vázquez Julio Andrés. Implementación de los métodos Cuasi-Newton. TESIS PARA OBTENER EL TÍTULO DE: LICENCIADO EN MATEMÁTICAS APLICADAS. 2019.

[Virieux J. and Operto S, 2009] Virieux J. and Operto S. (2009). An overview of full-waveform inversion in exploration geophysics. GEOPHYSICS, VOL. 74, NO. 6. NOVEMBER-DECEMBER 2009; P. WCC127WCC152, 15 FIGS., 1 TABLE.10.1190/1.3238367