Informe Práctica

Diego Ramírez

March 17, 2017

Abstract

1 Adaptación del paper usando varianza de la estimación

Dado una cantidad de puntos a trabajar ($NPart$) y una cantidad de puntos a simular (n_0) se inicia con el método original del paper, el cual es básicamente ir generando una "predicción" de los valores de energía H y la derivada de energía libre F' de todos los puntos a tratar a partir de las configuraciones obtenidas en los puntos simulados más próximos (inicialmente siendo los valores extremos 0.0 y 1.0).
Luego, dada una función de "desconfianza" para cada punto, la cual depende de:

- La diferencia entre las predicciones de los puntos simulados más próximos.
- La estimación del error asociado a cada punto (el cual depende de F'').
- Una ponderación (o peso) asociada a la distancia que se tiene de estos puntos vecinos.

Luego se extrae el máximo de la función en la muestra de puntos para luego este valor sea el próximo a ser simulado, dado es el que resulta ser la estimación de mayor incertidumbre.

Se itera sobre este método hasta cumplir con la cantidad n_0 de puntos planteada y, finalmente, se considera una combinación convexa de las ultimas predicciones de cada punto que no fue simulado.
Cabe notar que en cada uno de estos pasos se guarda un registro del error estadístico asociado o su propagación en caso de los puntos cuyos valores fueron predichos.

Candidatos a función de desconfianza:

- $Diff * Peso * |tol - var|^m$ / tol= tolerancia que quiero darle al error, y estudiar m que resulte apropiado

- $Peso * Diff * var^m$ / Agregar una ponderación por el error, estudiando el m apropiado, m=0 corresponde a la implementación original
- $a * Peso + b * Diff + c * var$ Estudiar coeficientes que ajusten el peso de forma apropiada

```
from header import *
import numpy as np

#############################################################################
#numero de puntos a simular
n_0=15
#tamano particion a trabajar
NPart=100
Part=[round(0.0 + i*(1./NPart),2) for i in range(NPart+1)]
#inicializo cada uno de los puntos(objetos)
Pts = [Lambda(x) for x in Part]
#simulacion inicial
Pts[0].simular()
Pts[-1].simular()
#Realiza una primera ronda de predicciones
for i in range(1,NPart-1):
Pts[i].predict(0.0,'Ant')
Pts[i].predict(1.0,'Succ')

N=2
#Dado n0 puntos a simular
while N<= n_0:
R=[] #Lista pesos
for y in range(NPart):
if Pts[y].tipo=='p':
x=Pts[y].valor
H_xa = Pts[y].ham['Ant']
H_xs = Pts[y].ham['Succ']
Ant  = Pts[y].Ant
Succ = Pts[y].Succ
#variacion o diferencia entre predicciones de puntos "vecinos"
Diff=abs(H_xa-H_xs)
#asigno ponderacion (peso) segun distancia a ambos puntos vecinos
w=(Succ-Ant)-abs((Succ-x)-(x-Ant))
#Crear una lista del valor dado a la "desconfianza" de cada punto
R.append(desconfianza(w,Diff))
else:
R.append(-100)
#Aquel punto con mayor valor en la lista es asignado como proximo a
    simular
Q=Pts[np.argmax(R)]
Q.simular()
#Se actualizan los vecinos de aquellos puntos predichos
```

```
for y in range(NPart):
if Pts[y].tipo=='p':
if Pts[y].ant!=Pts[y].ubicar(Pts)['Ant']:
Pts[y].ant=Pts[y].ubicar(Pts)['Ant']
Pts[y].predict(Pts[y].Ant,'Ant')
if Pts[y].succ!=Pts[y].ubicar(Pts)['Succ']:
Pts[y].succ=Pts[y].ubicar(Pts)['Succ']
Pts[y].predict(Pts[y].Succ,'Succ')
N+=1

#De aqui hay dos caminos, integrar a partir de estos puntos obtenidos, o
    bien,
#Haciendo una combinacion convexa de las predicciones entre puntos
    vecinos
#para aquellos no simulados. junto con una propagacion
# del error respectivo a estos mismos

for y in range(NPart):
x=Pts[y].valor
Y=Pts[y]
if Y.tipo=='p':
Ant  = Y.Ant
Succ = Y.Succ

w_a=(x-Succ)/(Ant-Succ)
w_s=(x-Ant)/(Succ-Ant)
Y.ham= w_a*Y.ham['Ant'] + w_s*Y.ham['Succ']
Y.err=np.sqrt( (w_a*Y.err['Ant'])**2 + (w_s*Y.err['Succ'])**2)

#Escribo un archivo que guarde la informacion final:
#valor del punto # predicho/simulado # valor energia #err asociado
with open('dHdlAdapt.dat','w') as F:
F.write('#Pt\t'+'type\t'+'ham\t'+'errorEst\n')
for x in Pts:
F.write(str(x.valor)+'\t'+str(x.tipo)+'\t'+str(x.ham)+'\t'+str(x.err)+'\n')
F.close()

#integrar con trapezoide (o metodo de simpson) los puntos finales en el
    eje x
#junto a sus valores respectivos en el eje y , graficar.
ys=[x.ham for x in Pts]
es=[x.err for x in Pts]
plotCurveErrors(Part,ys,es)
```

2 Aproximar mínimo en metodo variacional.

Dado se busca obtener el valor de $F(1) - F(0)$, mediante una integral en F' al hacer integración numérica se busca tener algun control sobre el error como se

ha mencionado anteriormente.
Una segunda forma de afrontar este problema es verlo directamente como un problema de minimización en los estimadores de F'.
Asumiendo inicialmente una partición uniforme de los ($\lambda_i = i\Delta\lambda$, con $\Delta\lambda = 1/n$), una estimación de la varianza se puede estudiar de la siguiente forma:

$$Var\left(\Delta\lambda \sum_{i=1}^{n} \frac{\partial H_\lambda}{\partial \lambda}\right) = \frac{1}{n^2} \sum_{i=1}^{n} \mathbb{E}_{\mu_\lambda}\left(\left(\frac{\partial H_\lambda}{\partial \lambda} - \mathbb{E}_{\mu_\lambda}\left(\frac{\partial H_\lambda}{\partial \lambda}\right)\right)^2\right)|_{\lambda=\lambda_i}$$

Luego, para n suficientemente grande

$$\simeq \frac{1}{n}\int_0^1 \mathbb{E}_{\mu_\lambda}\left(\left(\frac{\partial H_\lambda}{\partial \lambda} - \mathbb{E}_{\mu_\lambda}\left(\frac{\partial H_\lambda}{\partial \lambda}\right)\right)^2\right) d\lambda.$$

Ya que se busca escoger una partición de $[0,1]$ de forma tal que la varianza de mi estimador sea mínima, es necesario plantear el problema con tal de "redistribuir" los puntos de la partición equiespaciada.
Mediante un cambio de variable se estudia una función creciente
$\phi : [0,1] \to [0,1]$, (con condiciones de borde $\phi(0) = 0, \phi(1) = 1$), se obtiene el problema variacional:

$$\min \text{ Var}(\phi) \; = \; \int_0^1 \phi'(\lambda)^2 \sigma^2(\phi(\lambda))\, d\lambda$$

que busca una función ϕ tal que minimice la varianza de mi estimación, donde:

$$\sigma^2(\lambda) = \mathbb{E}_{\mu_\lambda}\left(\left(\frac{\partial H_\lambda}{\partial \lambda} - \mathbb{E}_{\mu_\lambda}\left(\frac{\partial H_\lambda}{\partial \lambda}\right)\right)^2\right)$$

Corresponde a la varianza en el punto λ.
Resolviendo las ecuaciones de Euler-Lagrange se llega al óptimo:

$$\phi(\lambda) = \Sigma^{-1}(\kappa\lambda) \tag{1}$$

Donde $\Sigma(s) = \int_0^s \sigma(r)dr$ y $\kappa = \Sigma(1)$ el cual depende de la desviacion estandar, cabe notar que en el caso alquémico (el Hamiltoniano es una combinación convexa de los estados borde) la varianza es $\sigma^2(\lambda) = -\beta^{-1}F''(\lambda)$.
Luego para resolver la solución se debe, además de estimar puntualmente el valor de F'' mediante diferencias finitas, resolver una EDO numéricamente a fin de obtener el operador inverso de Σ y la integral entre 0 y 1 de la desviación estándar.

$$\frac{\sigma(c)}{\kappa} c' = 1 \quad s.a. \quad c(0) = 0 \tag{2}$$

El método se reduce finalmente a:
Estimar empíricamente la desviación estándar(σ) mediante diferencias finitas

sobre F', esto se traduce en simular una cantidad inicial N de puntos de forma equiespaciada e integrar σ en esta muestra de valores entre 0 y 1, luego, resolviendo la EDO (2), y llamando ϕ a la solución de ésta, se obtiene la imagen $\phi(x)$ de los 10 puntos equiespaciados.

Se construye una matriz de distancias de cada $\phi(x)$, a cada punto x dándose una tolerancia ϵ de cercanía entre puntos, de forma tal que si dos puntos están más cerca que mi tolerancia se considera su distancia como infinito.

Finalmente, se crea una matriz de pesos que contenga el valor de $\sigma(\phi(x))/d(x, \phi(X))$. De ahi se puede ir escogiendo un número de pts de a lo más N tal que se agreguen a las simulaciones según su peso en la matriz.
La implementación de este método, si bien para una cantidad grande de puntos debiese resultar en una gran reducción del error desde el punto de vista estadístico, trae a su vez errores deterministas de distintas fuentes (resolución de una EDO, integrar sigma, y calcular la energía libre).