Centro de investigación en Matemáticas A.C

Temas Selectos en Estadística

Hairo Ulises Miranda Belmonte

Tarea 4. Diferencias finitas: Determinante Jacobiano- determinante de Vandermonde // Resolvente caso límite pdf espectral

07 de Septiembre del 2019

Ejercicio 1. Determinante Jacobiano - Determinante de Vandermonde

Considere la descomposición espectral $H = OXO'$, donde H es una matriz simétrica de dimensión nxn, O es la matriz ortogonal que contiene los eigenvectores de H, y X es una matriz diagona que contiene sus eigenvalores.

Los diferenciales de H se pueden exprezar como:

$$dH = \Pi_{i<j}|\lambda_i - \lambda_j|(dX)(O'dO)$$

Númericamente, las perturbaciones en X y O se calculan a trevés de las perturbaciones en H. Como analistas nnuméricos siempre pensamos en H como la entrada, y en $X; O$ como la salidas, por lo que es natural hacerse preguntas en esa dirección. Asumiendo que la descomposición espectral es única después de fijar la fase de las columnas de O, la perturbación a primer orden en $X; O$ debido a la perturbación en H está dada por:

$$\frac{(dX)(O'dO)}{dH} = \frac{1}{\Pi_{i<j}|\lambda_i - \lambda_j|} = \frac{1}{\Delta(X)}$$

donde $\Delta(X)$ es el valor absoluto del determinante de Vandermonde

Basandose en la expresión anterior, escriba un código para obtener numéricamente el jacobiano, y compare el resultado con el valor exacto al calcular el determinante de Vandermonde para una matriz fija de dimensión $10x10$ de tal manera que el error relativo sea menor a 10^{-3} Se recomienda seguir los siguientes pasos:

(i) Construya una matriz simérica: H.

(ii) Obtenga los valores y vectores propios de la matriz construida: $X; O$.

(iii) Determine la dimensión de la matriz jacobiana: $n(n+1) = 2xn(n+1) = 2$.

(iv) De manera iterativa realizar los siguientes pasos:

- Genera una perturbación ϵ en el elemento i, j de la matriz H: $H' = H + \epsilon E$
- Obtener los valores y vectores propios de H' : X', O'
- Calcular los valores propios perturbados: $dX = (X - x')/\epsilon$
- Calcular los vectores propios perturbados:$= O^T dO = O^T(O' - O)/\epsilon$

(v) Llenar las primeras n columnas de la matriz jacobiana con dX, y las restantes con $O^T dO$.

(vi) Calcular el valor absoluto de la matriz jacobiana resultante y comparar el resultado con el valor del determinante de Vandermonde de H.

(vii) Asegurarse que el error relativo sea menor a 10^{-3}.

In [1]:
```
import numpy as np # importar numpy
```

Se define la siguiente función para resover el jacobiano numéricamente a través de diferencias finitas:

- Se genera una matriz cuadrada aleatoria, sea X con elementos que se distribuyen $X \sim N(0, 1)$, de tamaño $N = 10$

In [7]:
```
np.random.seed(16)
n=10 # tamaño de la matriz
X = np.random.normal(0,1, (n,n))  # se genera matriz 10 x 10
```

- Se simetriza la matriz aleatoria de la forma:

$$A = \frac{X + X'}{n}$$

In [8]:
```
A = (X + X.T)/n # matriz simetrica
```

- Se calcula la transformación de similaridad (i.e., la descomposición espectral por medio de sus valore y vectores propios)

$$A = LQL'$$

L: valores propios

Q: vectores propios

In [10]:
```
L, Q = np.linalg.eig(A) # valores y vectores propios
L = np.diag(L) # introducimos valores propios en una matriz
```

- Se genera el espacio de memoria para almacenar a los elementos de la matriz jacobiana de tamaño:

$$J_{\frac{n(n+1)}{2} x \frac{n(n+1)}{2}}$$

en este caso se tiene:

$$\frac{n(n+1)}{2} = \frac{10(10+1)}{2} = 55$$

In [13]:
```
JacMatrix = np.zeros((int(n*(n+1)/2),int(n*(n+1)/2))) # inicializa matriz jaco
biana
JacMatrix.shape
```

Out[13]: (55, 55)

- se declara un valor ϵ pequeño con $\epsilon > 0$

In [15]:
```
epsilon=1e-7
```

- Se realiza la iteración para calcular las perturbaciones y con ello el jacobiano por medio de diferencias finitas

In [ ]:
```
idx = 0 # índice que recorre columnas del jacobiano
mask = np.triu(np.ones(n, dtype = bool),1) # mascara que filtra elementos de l
a parte triangular superior

for i in range(0,n):
    for j in range(i,n):
        Eij = np.zeros((n,n)) # inizializa perturbación
        Eij[i,j] = 1  # matriz perturbada
        Eij[j,i] = 1 # matriz perturbada
        Ap = A + epsilon*Eij # matriz ya perturbada
        #  Valores y vectores propios perturbados
        Lp, Qp = np.linalg.eig(Ap)
        dL= (np.diag(Lp)-L)/epsilon # Valor propio perturbado
        QdQ = Q.T@(Qp-Q)/epsilon # Vector propio perturbado
        # Matriz Jacobiana:

            # Guardando las primeras n columnas de la matriz
            # jacobiana con dX
        JacMatrix[0:n,idx]= np.diag(dL) # Valor propio parte del jacobiano

            # Guardando las n+q restantes columndad con OtdO

        JacMatrix[n:, idx] = QdQ[mask] # Vectores propios parte de  Jacobiano
        idx=idx+1 # incrementa el núммero de la columna
```

- valor absoluto del determinante de la Matriz Jacobiana

$$|det(J)| = \frac{(dX)(O'dO)}{dH} = \frac{1}{\Pi_{i<j}|\lambda_i - \lambda_j|}$$

In [20]:
```
J = np.abs(np.linalg.det(JacMatrix))
```

- Calculamos matriz Vandermont

In [23]:
```
Vandermont = np.vander(np.diag(L))
```

- Tomamos el absoluto del determinante de la matriz de vander, y realizamos uno sobre lo anterior:

$$\frac{1}{\Delta(X)}$$

X: valore spropios de la matriz aleatoria simetrizada

In [ ]:
```
det_Vandermont = 1/np.abs(np.linalg.det(Vandermont))
```

Comparando los resultados debemos tener buena aproximación de:

$$\frac{(dX)(O'dO)}{dH} = \frac{1}{\Pi_{i<j}|\lambda_i - \lambda_j|} = \frac{1}{\Delta(X)}$$

In [26]:
```
print("Diferencias finitas", J)
print("Vandemont", det_Vandermont)
```

```
Diferencias finitas 434082972123294.8
Vandemont 434082982495145.4
```

- Asegurando que el error relativo sea menor a 10^{-3}

In [31]:
```
error_relativo = np.abs((J - det_Vandermont) / np.abs(det_Vandermont))
print("El error relativo es de : ", error_relativo, "¿Es menor que 10 a la men
os tres? R =",
      error_relativo<10**(-3))
```

```
El error relativo es de :  2.389370461583574e-08 ¿Es menor que 10 a la menos
tres? R = True
```

Conclusión :

- Se tiene que la "repulsión" se escribe como el determinante de Vandermonde
- Las diferencias de la matriz A simetrica, es relativa a la diferencia de A simetrica perturbada via una transformación ortogonal, con sus respectivos valores y vectores propios perturbados.
- Se encuentra que el error relativo del valor absoluto del determinante del jacobiano, el cual se calculó por medio de diferencias finitas, es menor que uno sobre el valor absoluto del determinante de la matriz de vandermont cuya entrada fueron los valores propios de la matriz aleatoria simetrizada, esto en un valor menor a 10^{-3}.

Referencia :

- Edelman, A., & Rao, N. R. (2005). Random matrix theory. Acta Numerica, 14, 233–297. doi:10.1017/s0962492904000236

Ejercicio 2. Resolvente: comprobación del resultado

Para comprobar que la solución es consistente insertemos la fórmula del semicírculo

$$\rho(\lambda) = \frac{1}{\pi}\sqrt{(2-\lambda^2)}$$

en la definición del resolvente

$$G_{\infty}^{av} = \int dx' \frac{\rho(\lambda)}{z-\lambda'}$$

y realice la integración númerica para $z = \lambda - i\epsilon$, con $\epsilon > 0$, separando los casos:

i) $0 < \lambda < \sqrt{(2)}$

ii) $-\sqrt{(2)} < \lambda < 0$

Compare los resultados con la expresión dada por

$$G_{\infty}^{av} = z \pm \sqrt{(z^2-2)}$$

para 10 elecciones distintas de z. El ejercicio se evalúa como correcto si el error relativo porcentual promedio es menor al $1\%0$.

In [32]:
```
from scipy import integrate # evalua integrales
```

- Se declaran los soportes de la integral, o los intervalos de los valores donde λ se defina en la función, i.e.:

$$-\sqrt{(2)} < \lambda < \sqrt{(2)}$$

lowIntervalo = -np.sqrt(2) UppIntervalo = np.sqrt(2)

- Función que realiza la integral, la cual comprueba el resultado de encontrar el resolvente:

$$G_{\infty}^{av} = \int dx' \frac{\frac{1}{\pi}\sqrt{(2-\lambda^2)}}{z-\lambda'}$$

Nota: Observe que aquí la integral se le sustituye la función de densidad espectra, que dado al resultado se tiene la ley del semicírculo

Se separan las integrales, la parte real e imaginaria, y por ultimo se suman para tener el resultado de la integral compleja (i.e., el promedio del resolvente $G_{\infty}^{av}(z)$)

Nota: Input función Gavg_integral

- real: parte real del número complejo
- imaginario: parte compleja
- lowIntervalo: cota inferior del intervalo donde se define λ
- UppIntervalo: cota superior del intervalo donde se define λ

In [ ]:
```
def Gavg_integral(real, imaginario, lowIntervalo, UppIntervalo):
    z = real - 1j*imaginario # número complejo
    # integral parte real
    integral_real = integrate.quad(lambda x: np.real(np.sqrt(2-x**2)/(np.pi*(z
-x))), lowIntervalo, UppIntervalo)[0]
    # integral parte compleja
    integral_imag = integrate.quad(lambda x: np.imag(np.sqrt(2-x**2)/(np.pi*(z
-x))), lowIntervalo, UppIntervalo)[0]
    # integral compleja (suma de las anteriores)
    integral_complex = integral_real + 1j*integral_imag
    return integral_complex
```

- Función que calcula el error relativo:

$$Erelativo = \frac{|estimado - exacto|}{|exacto|}$$

In [104]:
```
erro_realativo = lambda x,y: np.abs((x - y) / np.abs(y))
```

- Se separa para dos casos, ya que los resultado se comparan con la elección de signos de :

$$G_{\infty}^{av} = z \pm \sqrt{(z^2 - 2)}$$

- Caso 1. Se verá que para un valor positivo:

$$G_{\infty}^{av} = z + \sqrt{(z^2 - 2)}$$

solo sirve si $0 < \lambda < \sqrt{(2)}$

- Caso 2. Se verá que para un valor negativo:

$$G_{\infty}^{av} = z + \sqrt{(z^2 - 2)}$$

solo sirve si $-\sqrt{(2)} < \lambda < 0$

Caso 1

$0 < \lambda < \sqrt{(2)}$

- Función del valor medio del resolvente, recive el número complejo:

$$G_{\infty}^{av} = z + \sqrt{(z^2 - 2)}$$

In [86]:
```
G_plus = lambda x: x + np.sqrt(x**2 - 2)
```

- Se requiere realizar 10 elecciones de z, para esto se generan valores aleatorios para la parte real e imaginaria del valor de z

$$z = \lambda - i\epsilon$$

- Parte real λ, 10 valores entre $-\sqrt{(2)}$ a 0.
- Parte imaginaria ϵ, valores positivos pequeños en el intervalo de [.01,.1]

In [117]:
```
np.random.seed(16) # fija semilla
real_mas = np.random.uniform(-np.sqrt(2),0,10) # genera valores de lambda
imaginario_mas  = np.random.uniform(.01,.1,10) # genera valores de epsilon
```

- Almacenamos en matrices los valores de la integral del resolvente medio al sustituir la ley del semicirculo, y la expresión algebraica que se desarrolla del valor medio del resolvente.

In [118]:
```
valor_mas = np.zeros((10,1), dtype = complex) # almacena resultado de integral
del resolvente
mas = np.zeros((10,1), dtype = complex) # almacena resultado de definición de
 resolvente
```

- Calculamos los valores:

$$G_{\infty}^{av} = \int dx' \frac{\frac{1}{\pi}\sqrt{(2-\lambda^2)}}{z-\lambda'}$$

Observe que aquí la integral se le sustituye la función de densidad espectra, que dado al resultado se tiene la ley del semicírculo

$$G_{\infty}^{av} = z + \sqrt{(z^2-2)}$$

esto para 10 elecciones distintas de $z_j = \lambda_j - i\epsilon_j$ con $j = 1, 2, \ldots, 10$

In [119]:
```
for i in range(len(real_mas)):
    # valores de la integral para distintas z's
    valor_mas[i,0] = Gavg_integral(real_mas[i], imaginario_mas[i], lowIntervalo, UppIntervalo)
    # valor del resolvente encontrado
    z_mas = real_mas[i] - 1j*imaginario_mas[i] # genera el número
    mas[i,0] = G_plus(z_mas)
print('Valor de resolvente en z:', np.real(valor_mas), np.imag(valor_mas))
print('Valor de G mas:',np.real(mas),np.imag(mas))
```

```
Valor de resolvente en z: [[-1.02397365]
 [-0.66511733]
 [-0.59774929]
 [-1.27351833]
 [-0.87704735]
 [-1.05388949]
 [-0.41643518]
 [-1.104258  ]
 [-1.24151053]
 [-0.08045785]] [[0.83519216]
 [1.22621377]
 [1.1911812 ]
 [0.405442  ]
 [1.0558003 ]
 [0.8558474 ]
 [1.27343537]
 [0.72932989]
 [0.4975556 ]
 [1.36243239]]
Valor de G mas: [[-1.02397365]
 [-0.66511733]
 [-0.59774929]
 [-1.27351833]
 [-0.87704735]
 [-1.05388949]
 [-0.41643518]
 [-1.104258  ]
 [-1.24151053]
 [-0.08045785]] [[0.83519216]
 [1.22621377]
 [1.1911812 ]
 [0.405442  ]
 [1.0558003 ]
 [0.8558474 ]
 [1.27343537]
 [0.72932989]
 [0.4975556 ]
 [1.36243239]]
```

- Se toma el valor promedio de el error relativo de cada caso

$$\hat{E}relativo = \frac{\Sigma_{j=1}^{10} Erelativo_j}{n}$$

In [120]:
```
e_mas = erro_realativo(valor_mas, mas).mean() # checar los dos casos para ver
 cual poner
print('Valor del error relativo porcentual:', e_mas,
     '\n¿Es menor el error relativo porcentual  al uno porciento? R =', e_mas
<.01)
```

```
Valor del error relativo porcentual: 2.9676148056095533e-12
¿Es menor el error relativo porcentual  al uno porciento? R = True
```

Caso 2

$-\sqrt{(2)} < \lambda < 0$

- Función del valor medio del resolvente, recive el número complejo:

$$G_{\infty}^{av} = z - \sqrt{(z^2 - 2)}$$

In [92]:
```
G_minus = lambda x: x - np.sqrt(x**2 - 2)
```

- Se requiere realizar 10 elecciones de z, para esto se generan valores aleatorios para la parte real e imaginaria del valor de z

$$z = \lambda - i\epsilon$$

- Parte real λ, 10 valores entre 0 a $\sqrt{(2)}$.
- Parte imaginaria ϵ, valores positivos pequeños en el intervalo de [.01,.1]

In [121]:
```
np.random.seed(16) # fijar semilla
real_menos = np.random.uniform(0,np.sqrt(2),10) # genera valores de lambda
imaginario_menos  = np.random.uniform(.01,.1,10) # genera valores de epsilon
```

- Almacenamos en matrices los valores de la integral del resolvente medio al sustituir la ley del semicirculo, y la expresión algebraica que se desarrolla del valor medio del resolvente.

In [122]:
```
valor_menos = np.zeros((10,1), dtype = complex)
menos = np.zeros((10,1), dtype = complex)
```

- Calculamos los valores:

$$G_{\infty}^{av} = \int dx' \frac{\frac{1}{\pi}\sqrt{(2-\lambda^2)}}{z-\lambda'}$$

Observe que aquí la integral se le sustituye la función de densidad espectra, que dado al resultado se tiene la ley del semicírculo

$$G_{\infty}^{av} = z - \sqrt{(z^2-2)}$$

esto para 10 elecciones distintas de $z_j = \lambda_j - i\epsilon_j$ con $j = 1, 2, \ldots, 10$

In [123]:
```
for i in range(len(real_menos)):
    # valores de la integral para distintas z's
    valor_menos[i,0] = Gavg_integral(real_menos[i], imaginario_menos[i], lowIn
tervalo, UppIntervalo)
    # valor del resolvente encontrado
    z_menos = real_menos[i] - 1j*imaginario_menos[i] # genera el número
    menos[i,0] = G_minus(z_menos)
print('Valor de resolvente en z:', np.real(valor_menos), np.imag(valor_menos))
print('Valor de G menos:',np.real(menos),np.imag(menos))
```

```
Valor de resolvente en z: [[0.30188344]
 [0.72941841]
 [0.72944548]
 [0.06338357]
 [0.4975722 ]
 [0.30715392]
 [0.90527648]
 [0.22296266]
 [0.09738528]
 [1.19698603]] [[1.31918303]
 [1.18838478]
 [1.10882183]
 [1.38869038]
 [1.28693118]
 [1.34266742]
 [0.95750931]
 [1.34433204]
 [1.38166349]
 [0.4492075 ]]
Valor de G menos: [[0.30188344]
 [0.72941841]
 [0.72944548]
 [0.06338357]
 [0.4975722 ]
 [0.30715392]
 [0.90527648]
 [0.22296266]
 [0.09738528]
 [1.19698603]] [[1.31918303]
 [1.18838478]
 [1.10882183]
 [1.38869038]
 [1.28693118]
 [1.34266742]
 [0.95750931]
 [1.34433204]
 [1.38166349]
 [0.4492075 ]]
```

- Se toma el valor promedio de el error relativo de cada caso

$$\hat{E}relativo = \frac{\Sigma_{j=1}^{10} Erelativo_j}{n}$$

In [124]:
```
e_menos = erro_realativo(valor_menos, menos).mean() # checar los dos casos par
a ver cual poner
print('Valor del error relativo porcentual:', e_menos,
      '\n¿Es menor el error relativo porcentual al uno porciento? R =', e_menos
<.01)
```

```
Valor del error relativo porcentual: 1.256870674872081e-12
¿Es menor el error relativo porcentual al uno porciento? R = True
```

Conclusión:

- Se encuentra que en los dos casos, el error relativo porcentual para las 10 distintas elecciones de z, es menor al 1%
- De esta forma, al sustituir el valor promedio del resolvente encontrado de forma algebraica, y encontral la función de densidad espectral en el caso límite, y como resultado la ley del semicírculo, se puede sustituirlo en la integral del valor medio del resolvente, y comprobar si el resolvente es correcto
- En este caso para 10 elecciones de Z, sí es correcto el resolvente

Github:

presentación en: https://nbviewer.jupyter.org/github/hairo1421/Maestria-Computo-Estadistico/blob/master/05%20-%20Tercer%20Semestre%20Temas%20en%20Estad%C3%ADstica/Matrices%20Aleatorias/Tarea%204.%20Dete%20determinante%20de%20Vandermonde%20%20Resolvente%20.ipynb (https://nbviewer.jupyter.org/github/hairo1421/Maestria-Computo-Estadistico/blob/master/05%20-%20Tercer%20Semestre%20Temas%20en%20Estad%C3%ADstica/Matrices%20Aleatorias/Tarea%204.%20Dete%20determinante%20de%20Vandermonde%20%20Resolvente%20.ipynb)

archivos en: https://github.com/hairo1421/Maestria-Computo-Estadistico/blob/master/05%20-%20Tercer%20Semestre%20Temas%20en%20Estad%C3%ADstica/Matrices%20Aleatorias/Tarea%204.%20Dete%20determinante%20de%20Vandermonde%20%20Resolvente%20.ipynb (https://github.com/hairo1421/Maestria-Computo-Estadistico/blob/master/05%20-%20Tercer%20Semestre%20Temas%20en%20Estad%C3%ADstica/Matrices%20Aleatorias/Tarea%204.%20Dete%20determinante%20de%20Vandermonde%20%20Resolvente%20.ipynb)