CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS (CIMAT)
UNIDAD MONTERREY
TEMAS SELECTOS DE ESTADÍSTICA

Tarea 3

Diego Ramírez Araque

03 de septiembre de 2021

1. EJERCICIO 1

Consideramos

$$\overline{p}(s) = \frac{\pi s}{2} e^{-\frac{\pi s^2}{4}}.$$

Si definimos el cambio de variable $u = \frac{\pi s^2}{4}$ implica que $du = \frac{\pi s}{2}$, entonces la integral de $\overline{p}(s)$ se calcula como:

$$\begin{aligned}
\int_0^\infty \overline{p}(t)\,dt &= \int_0^\infty \frac{\pi t}{2} e^{-\frac{\pi t^2}{4}}\,dt \\
&= \int_0^\infty e^{-u}\,du \\
&= -e^{-u}\Big|_0^\infty \\
&= -(0-1) \\
&= 1
\end{aligned}$$

2. EJERCICIO 2.

2.1. ENSAYO SOBRE "THE STATISTICAL PROPERTIES OF THE CITY TRANSPORT IN CUERNAVACA (MEXICO) AND RANDOM MATRIX ENSEMBLES "

Se sabe que las propiedades estadística sde sistemas caoticos cuanticos se describen de manera adecuada a través de los ensambles aleatorios de Wigner/Dyson. La distribución estadística que caracteriza a los ensambles de matrices aleatorias se puede entender como la minimización de la información contenida en un sistema con la restricción de que las matrices poseen propiedades simétricas discretas. Sí $P(x_1, x_2, ..., x_n)$ denota la distribución conjunta de los eigenvalores $x_1, x_2, ..., x_n$ de las matrices dadas e

$$I = \int P(x_1, x_2, ..., x_n)\, ln(P(x_1, x_2, ..., x_n))\, dx_1 ... dx_n \tag{2.1}$$

la información de contenido. Si asumimos por instancia que las matrices son invariantes con respecto a transformaciones revertidas en el tiempo la información I es minimizada cuando la distribución $P(x_1, x_2, ..., x_n)$ describe un Ensamble Gaussiano Ortogonal (GOE). Si no existe simetría externa el mínimo total de la información I se alcanza a través del Ensamble Gaussiano Unitario (GUE), en donde la restricción es que las matrices deben ser hermiteanas.

Se sabe que los ensambles de matrices son de relavancia debido a su aplicación en variados sistemas clásicos de interacción de partículas, en donde las matrices de los eigenvalores $x_1, ..., x_n$ describen la posición de partículas. De esta forma el equilibrio térmico de un gas unidimensional que actúa a través del potencial de Coulomb (gas de Dyson) tiene propiedades estadísticas (dependiendo de la temperatura) que son idénticas al conjunto de matrices aleatoria. Lo mismo también es válido para otros potenciales. Bajo ciertos supuestos los métodos de la física estadística sigue siendo válidos para diferentes formas del potencial de partículas. Se puede demostrar que el potencial

$$V(x) \approx 1/|x|^a$$

con a siendo una constante positiva y lo cual conlleva una distribución de la matriz aleatoria de las posiciones de la partícula. La equivalencia de las propiedades estadísticas de las posiciones de la partícula de una dimensión de los gases interactuando a ensambles de matrices aleatorias y el hecho de que GUE minimiza la información (1) nos lleva a especular que, siempre y cuando la información contenida en el gas sea minimizada sus propiedades pueden describirse con GUE.

El gas unidimensional que propone el artículo se representan por autobuses que operan en la línea 4 de la Ciudad de Cuernavaca (México). El artículo muestra que las propiedades estadísticas de las llegadas de autobuses se describen mediante un Ensamble Unitario de matrices aleatorias. Para explicar el origen de la interacción entre los autobuses posteriores se requiere tomar nota de lo siguiente:

- No hay una empresa de cobertura responsable para organizar el transporte de la ciudad. En consecuencia, no existen limitaciones como un calendario, etc. que representan una influencia externa en el transporte.
- cada autobús es propiedad del conductor. Los conductores intentan maximizar sus ingresos y, por tanto, la cantidad de pasajeros que transporte.

Esto conduce a la competencia entre los conductores y a su interacción mutua. Los autores destacan que la interacción de la distribución de probabilidad de las distancias entre los siguientes autobuses es Poissoniana (esto debido a las condiciones de tráfico complicadas en la ciudad que funcionan como una urna eficaz). La distribución de Poisson implica, sin embargo, que la probabilidad de encuentros de dos autobuses es alta (agrupación de autobuses) lo cual entra en conflicto con el esfuerzo del conductor para maximizar el número de pasajeros transportados y en consecuencia maximizar la distancia al autobús precedente. Con el fin de evitar el desagradable efecto de agrupamiento de los conductores de autobuses en Cuernavaca designan colaboradores que registran el tiempo de arribo de los autobuses en lugares significativos. Estos registran los tiempos de llegada de autobuses en lugares significativos y los autobuses al llegar a un puesto de control, el conductor obtiene la información cuando el autobús anterior pasó por ese lugar. Conociendo el intervalo de tiempo del autobús precedente, el conductor intenta optimizar la distancia hasta él, ya sea disminuyendo o acelerando de tal manera que la información obtenida conduce a la interacción entre los autobuses cambiando sus propiedades estadísticas.

Para demostrar la equivalencia entre interacción de gases y los conductores de autobuses en Cuernavaca, el artículo analiza la distribución de diferencias empírica contra la distribución analítica así como también las funciones cumulativas empíricas vs. analíticas y la varianza con respecto al número de vehículos. Los autores recalcan que para un número de vehículos igual a 3, la varianza predicha y la varianza muestral se ajustan al modelo determinado por las varianzas de en un modelo GUE, insinuando que este modelo es un buen enfoque para analizar la distribución de los camiones. Este comportamiento puede entenderse como el estado de equilibrio del gas unidimensional en interacción bajo el supuesto de que información contenida en las posiciones de gas individual

las partículas se minimizan. El acuerdo del bus actual los datos con la predicción GUE son sorprendentemente buenos.

3. Siguientes ejercicios...

```
    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown
', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX incl
udes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full
'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple o
f
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containin
g::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
```

In [ ]:
```
# Librerías
import numpy as np
import matplotlib.pyplot as plt
import math as mth
from scipy.stats import multivariate_normal
import scipy.integrate as integrate
import scipy.special as special
import seaborn as sns
```

Temas Selectos de Estadística

Tarea 2

Ejercicio 1

Ejercicio 2

In [ ]:
```
def GOE(H):
  Hs = (H + np.transpose(H))/10
  evs_Hs = np.linalg.eigvals(Hs)
  return evs_Hs
```

In [ ]:
```
def gaussian_real_matrix(N):
  matrix_aux = np.zeros((N,N))
  for i in range(0,N):
    for j in range(0,N):
      a = float(multivariate_normal.rvs( mean=0, cov=1, size = 1))
      matrix_aux[i,j] = a
  return np.array(matrix_aux)
```

In [ ]:
```
samples = {}
aux_dict = {'X': None}
for i in range(0,1000):
  Z = gaussian_real_matrix(100)
  X = np.matmul(np.transpose(Z),Z)
  aux_dict['X'] =  X
  #aux_dict['eigen_values'] = np.linalg.eigvals(X)
  samples[i] = aux_dict
```

In [ ]:
```
samples_GOE = {}
aux_dict_GOE = {'eigen_values': None}
for i in range(0,1000):
  Hs = GOE(samples[i]['X'])
  aux_dict_GOE['eigen_values'] = Hs
  samples_GOE[i] = aux_dict_GOE
```

In [ ]:
```
samples_GOE[0]
```

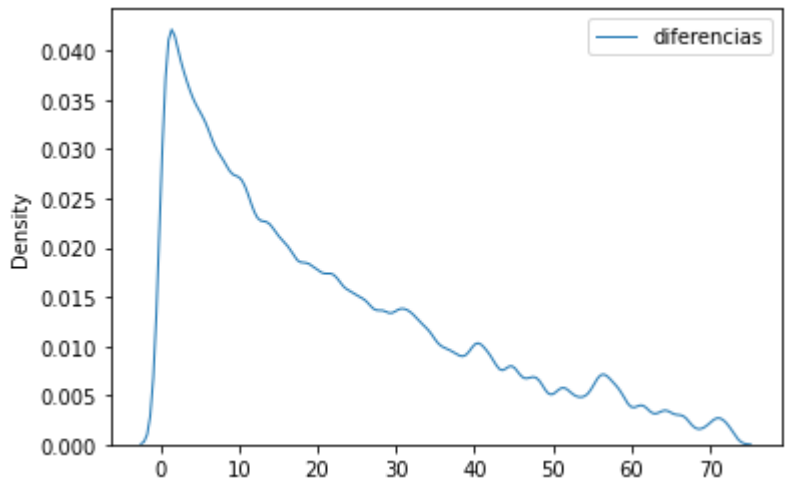
Out[ ]:
```
{'eigen_values': array([3.74001397e+02, 3.59556858e+02, 3.36461584e+0
2, 3.24001239e+02,
       3.10161857e+02, 3.04237577e+02, 2.96055964e+02, 2.87776666e+0
2,
       2.77222805e+02, 2.64608881e+02, 2.59030390e+02, 2.51045023e+0
2,
       2.40898150e+02, 2.30670067e+02, 2.24984107e+02, 2.19416210e+0
2,
       2.14830544e+02, 2.07242955e+02, 2.08066288e+02, 1.99718618e+0
2,
       1.93900400e+02, 1.91220496e+02, 1.75498049e+02, 1.81416368e+0
2,
       1.63074553e+02, 1.67651223e+02, 1.54108956e+02, 1.48321856e+0
2,
       1.45577081e+02, 1.38485031e+02, 1.35441053e+02, 1.32863141e+0
2,
       1.17140363e+02, 1.27721356e+02, 1.23384828e+02, 1.11101173e+0
2,
       1.06425791e+02, 1.04373251e+02, 1.00678888e+02, 9.34865416e+0
1,
       7.61183531e+01, 8.79717536e+01, 8.68946581e+01, 8.13536419e+0
1,
       8.49747081e+01, 7.07155840e+01, 6.95270105e+01, 6.58117722e+0
1,
       6.44882812e+01, 6.23683766e+01, 5.71345784e+01, 5.67144754e+0
1,
       4.80511685e+01, 4.94521172e+01, 5.19114468e+01, 5.39138416e+0
1,
       4.59316225e+01, 4.34559763e+01, 4.07178186e+01, 3.53467543e+0
1,
       3.70055404e+01, 3.93782318e+01, 3.89283608e+01, 3.30580239e+0
1,
       2.97007656e+01, 2.76830267e+01, 2.92543941e+01, 2.60245846e+0
1,
       2.39788860e+01, 2.26478342e+01, 2.10560742e+01, 1.90508864e+0
1,
       1.70984764e+01, 1.77247218e+01, 1.48025025e+01, 1.01817937e+0
1,
       8.37663603e+00, 8.78056362e+00, 1.33553948e+01, 1.26991005e+0
1,
       1.88183298e+01, 5.06997801e+00, 3.99378006e+00, 3.19458319e+0
0,
       6.21070776e+00, 1.63000916e+00, 1.34257007e+00, 2.43903102e+0
0,
       9.56378453e-01, 4.96406735e-04, 2.73596763e+00, 5.52558536e+0
0,
       3.66438982e-01, 1.41543275e-01, 6.99736885e-01, 1.30500441e+0
1,
       6.58787771e+00, 2.39753170e-01, 1.92450542e-01, 2.13358506e-0
1])}
```

In [ ]:
```
diferencias = []
for i in range(0,1000):
  n = len(samples_GOE[i]['eigen_values'])
  for k in range(0,n-1):
    diferencia = [abs(samples_GOE[i]['eigen_values'][k]-samples_GOE
[i]['eigen_values'][j]) for j in range(k+1,n)]
    diferencias = diferencias + diferencia
```

```
---------------------------------------------------------------------------
------
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-82-2887b32f1730> in <module>()
      4   for k in range(0,n-1):
      5     diferencia = [abs(samples_GOE[i]['eigen_values'][k]-sampl
es_GOE[i]['eigen_values'][j]) for j in range(k+1,n)]
----> 6     diferencias = diferencias + diferencia

KeyboardInterrupt:
```

In [ ]:
```
ensambles = {'diferencias':diferencias}
plot = sns.kdeplot(data=ensambles, alpha=.35, linewidth=1)
```



Ejercicio 3

In [ ]:
```
def gaussian_real_matrix(N):
  matrix_aux = np.zeros((N,N))
  for i in range(0,N):
    for j in range(0,N):
      a = float(multivariate_normal.rvs( mean=0, cov=1, size = 1))
      matrix_aux[i,j] = a
  return np.array(matrix_aux)
```

In [ ]:
```
n = 20
X = gaussian_real_matrix(n)
A = (X+np.transpose(X))/n
L,Q = np.linalg.eig(A)
#Q = Q.reshape(n,1)
L = np.diag(L)
dim_jacobiano =int((n*(n+1))/2)
m_jacobiana = np.zeros([dim_jacobiano,dim_jacobiano])
epsilon = 1e-10
idx = 0
mask = np.triu(np.ones([n,n])) - np.identity(n)
mask = mask.astype('bool')
```

In [ ]:
```
for i in range(0,n):
  for j in range(i,n):
    ### Matriz de perturbaciones
    Eij = np.zeros([n,n])
    Eij[i,j] = 1
    Eij[j,i] = 1
    Ap = A + epsilon*Eij

    ### Eigenvalores y Eigenvectores
    Lp,Qp = np.linalg.eig(Ap)
    #Qp = Qp.reshape(n,1)
    dL = (np.diag(Lp)-L)/epsilon
    QdQ = np.matmul(np.transpose(Q),(Qp-Q))/epsilon

    ### Matriz Jacobiana
    m_jacobiana[0:n,idx] = np.diagonal(dL)

    m_jacobiana[n:,idx] = QdQ[mask]
    idx = idx+1
```

Validamos con el determinante del Jacobiano y el determinante de Vandermonde:

In [ ]:
```
print('Jacobiano: ',abs(np.linalg.det(m_jacobiana)))
print('Vandermonde: ',abs(1/np.linalg.det(np.vander(np.diagonal(L)))))
print('Diferencia :' ,abs(abs(np.linalg.det(m_jacobiana)) - abs(1/np.li
nalg.det(np.vander(np.diagonal(L))))))
```

```
Jacobiano:  2.9669249288066317e+113
Vandermonde:  9.934573878147292e+95
Diferencia : 2.9669249288066317e+113
```

Podemos observar que los valores se aproximan de manera satisfactoria. En la realización del ejercicio se notó que para dimensiones chicas estos valores pueden diferir por mucho y no aproximarse para nada.

Ejercicio 4

In [ ]:
```
## Ingresar parte real
re_z = np.sqrt(2) - .0001
##
if abs(re_z) >= np.sqrt(2):
  re_z = None
  print('ERROR: Real(z) debe ser más chico que el valor absoluto de sqrt(2)')
```

In [ ]:
```
## Ingresar parte imaginaria
im_z = 1000
z = complex(re_z,-im_z)
print(z)
```

```
(1.4141135623730952-1000j)
```

In [ ]:
```
result = integrate.quad(lambda x: np.sqrt(2-pow(x,2))/(np.pi*(z-x)),-np.sqrt(2),0)
print(result)
```

```
(1.007162017711073e-06, 5.948047197904083e-09)

/usr/local/lib/python3.7/dist-packages/scipy/integrate/quadpack.py:453: ComplexWarning: Casting complex values to real discards the imaginary part
  return _quadpack._qagse(func,a,b,args,full_output,epsabs,epsrel,limit)
```

In [ ]:
```
import scipy
from scipy.integrate import quad

def complex_integral(func, a, b, **kwargs):
    def real_func(x):
        return np.real(func(x))
    def imag_func(x):
        return np.imag(func(x))
    real_integral = quad(real_func, a, b, **kwargs)
    imag_integral = quad(imag_func, a, b, **kwargs)
    return (real_integral[0] + 1j*imag_integral[0], real_integral[1:], imag_integral[1:])
```

In [ ]:
```
complex_integral(lambda x: np.sqrt(2-pow(x,2))/(np.pi*(z-x)),-np.sqrt(2),0)
```

Out[ ]:
```
((1.007162017711073e-06+0.0004999979013850756j),
 (5.948047197904083e-09,),
 (5.6236048803781635e-14,))
```

In [ ]:
```
Gav_infty = complex_integral(lambda x: np.sqrt(2-pow(x,2))/(np.pi*(z-x)),-np.sqrt(2),np.sqrt(2))
```

In [ ]:
```
Gav_infty_plus = z + np.sqrt(pow(z,2)-2)
print(Gav_infty)
```

```
((1.4141129221029666e-06+0.0009999975002933322j), (5.952493916506455e
-09,), (6.367567063902957e-13,))
```

In [ ]:
```
Gav_infty_minus = z - np.sqrt(pow(z,2)-2)
print(Gav_infty)
```

```
((1.4141129221029666e-06+0.0009999975002933322j), (5.952493916506455e
-09,), (6.367567063902957e-13,))
```

In [ ]:
```
Gav_infty[0] - Gav_infty_minus
```

Out[ ]:
```
(4.3085630005225585e-12-5.051991811000356e-14j)
```

Se puede observar que el valor de G_{∞}^{av} en este caso coincide con $G_{\infty-}^{av}$.