KRZYSZTOF SOKÓŁ-SZOŁTYSEK
Projekt zaliczeniowy - język Python

KRÓTKI OPIS

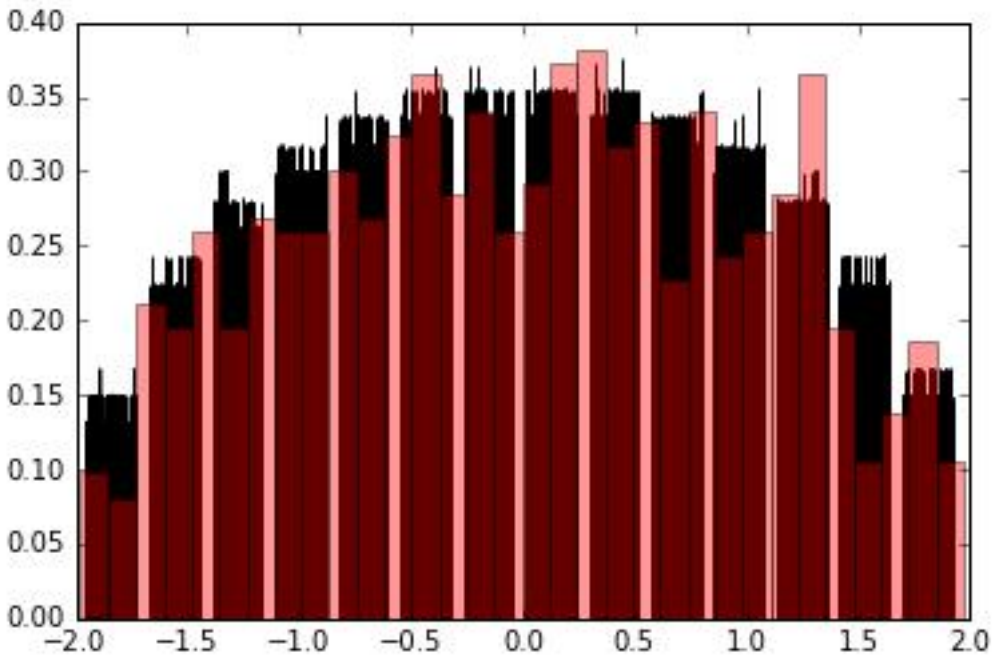
Aplikacja ma na celu zaprezentowanie w interesujący sposób pewnych własności zbiorów macierzy losowych Ginibre'a. Interfejs jest konsolowy, z interaktywnym menu pozwalającym wybrać dowolną z funkcji, a także ustawić samodzielnie parametry symulacji. Program korzysta z Pythona w wersji 3, oraz zewnętrznych bibliotek numpy, scipy, pylab oraz pyplot.

KONFIGURACJA

Program możemy uruchomić z linii komend za pomoca polecenia "python kss.py". Jeśli chcemy, możemy ustawić za pomocą klawisza 6 tryb korzystania z wlasnych parametrów symulacji, a następnie ustawić je za pomocą klawisza 7. Dostępne parametry to seed(szerzej opisany w sekcji MEAN AND STANDARD DEVIATIONS OF TRACES) oraz liczba macierzy których chcemy użyć - domyślne wartości dobrze przybliżające zaprezentowane niżej zjawiska będa używane domyślnie po wyłączeniu trybu "Use custom parameters".

PREZENTACJA FUNKCJI

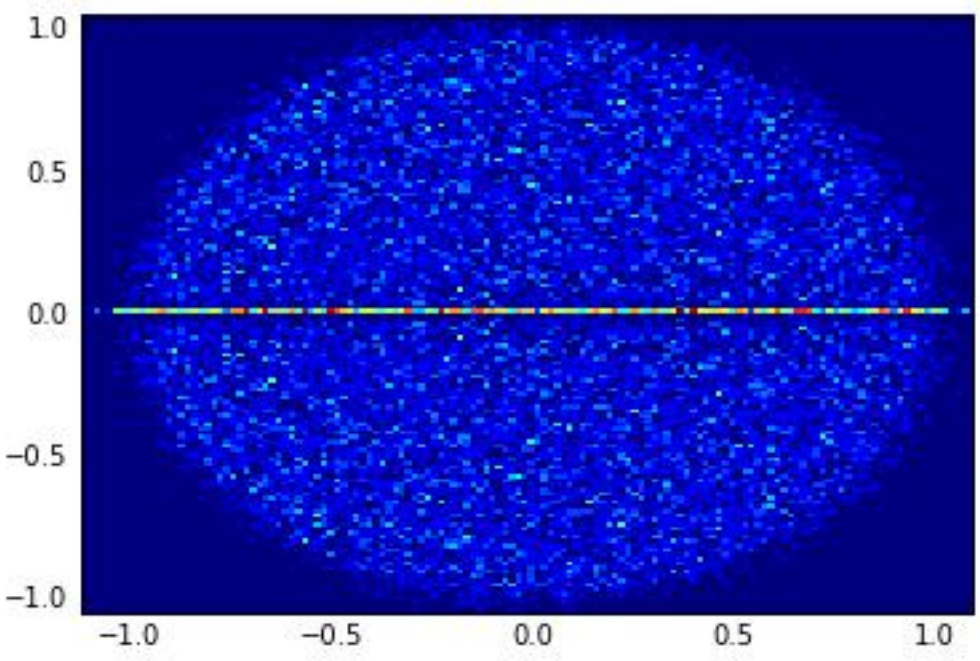
SEMICIRCULAR DISTRIBUTION



Dla 600 macierzy * 192 wartości własne = 115200 wartości własnych (bins = sqrt, czarny kolor) możemy uzyskac po unormowaniu wykres który przypomina rozklad półkolisty Wignera(100 losowych próbek z rozkładu, kolor czerwony, R = 2) - rozkład wartości własnych dużych losowych macierzy jest ograniczony właśnie w ten sposób (Winger semicircle law).

Więcej informacji - https://en.wikipedia.org/wiki/Wigner_semicircle_distribution

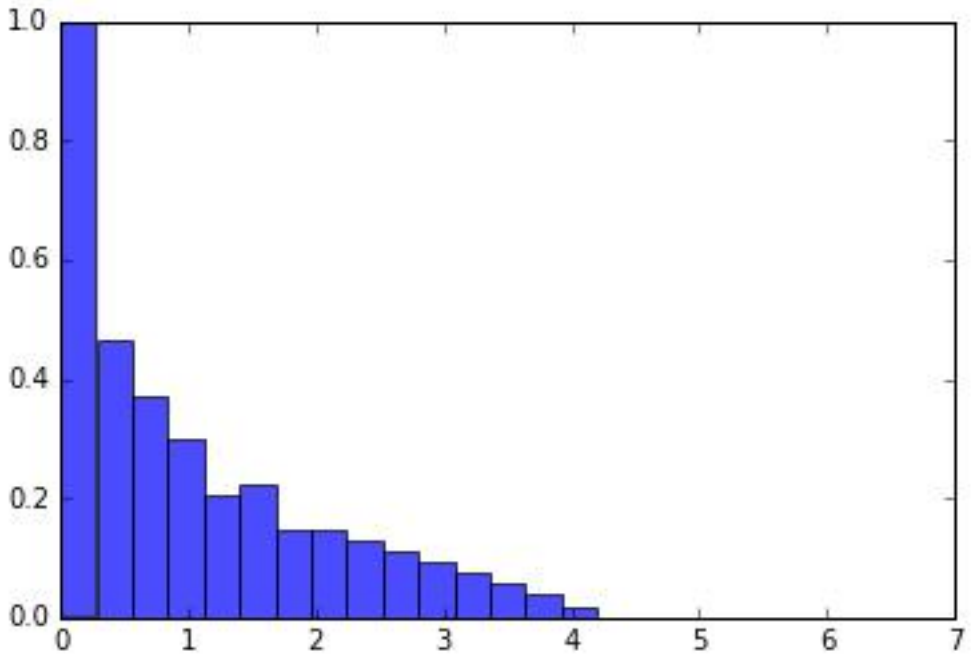
EIGENVALUES CONCENTRATION



Wartości własne losowej macierzy(rzeczywiste i zespolone) wykazują tendencje do układania sie w się w koło, a największe ich zagęszczenie występuje na osi rzeczywistej - zgodnie z Circular Law.

Więcej informacji - https://en.wikipedia.org/wiki/Circular_law

MARCHENKO-PASTUR DISTRIBUTION



Po adekwatnym przeskalowaniu możemy zaobserwować, że wartości osobliwe dużych prostokątnych macierzy losowych przybierają postać rozkładu Marchenko-Pastur.

Więcej informacji -
https://en.wikipedia.org/wiki/Marchenko%E2%80%93Pastur_distribution

KAPPA AVERAGES COMPARISON

Wyniki z kilku pomiarów :

Kappa:
mean K:
2288.7733601
std K:
6331.48387263
mean logarithm:
6.83078633814
logarithmic average:
925.918611415
std of log average:
1.1214281491

Kappa:
mean K:
3654.00596131
std K:
23376.3953658
mean logarithm:
6.81841196902
logarithmic average:
914.531551882
std of log average:
1.13851396002

Kappa:
mean K:
3793.78006404
std K:
28043.0877192
mean logarithm:
6.76298535486
logarithmic average:
865.221336774
std of log average:
1.16010263806

Korzystając z tej funkcji porównujących różne rodzaje średnich zauważalna jest duża standardowa dewiacja średniej arytmetycznej i stosunkowo duże odchyły niej samej. Dla danych o znacznym “rozrzucie” rozsądniejsza okazuje się średnia logarytmiczna, która daje bardziej powtarzalne rezultaty(kontrast szczególnie widoczny przy porównywaniu standardowych dewiacji)

MEAN AND STANDARD DEVIATION OF TRACES

Wyniki z kilku pomiarów :

mean t: 192.280612856
standard deviation t: 1.45027596812

mean t: 192.088407855
standard deviation t: 1.33132267908

mean t: 191.911635962
standard deviation t: 1.51779559528

W tej funkcji pokazujemy ciekawą własność tego rodzaju macierzy - średnia dla śladów naszych macierzy zbliża się do parametru “seed”, którego użyliśmy do tworzenia listy macierzy losowych.

LISTING Z KOMENTARZEM

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a script file.

Autor : Krzysztof Sokół-Szołtysek
"""
#import Gnuplot
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as ss
import pylab as pl
import sys
import os


seed = 100
Matricescount = 500
defaultParams = True
```

Globalne zmienne z domyślnymi wartościami oraz przełącznik trybu korzystania z własnych

```
def main_menu():
    os.system('clear')

    print ("\n\n\n\n\nWelcome,\n")
    print ("Please choose the feature you want to see:")
    print ("1. Semicircular distribution(might take a while)")
    print ("2. Eigenvalues concentration")
    print ("3. Marchenko-Pastur distribution")
    print ("4. Kappa averages comparison")
    print ("5. Mean and stadard deviation of traces")
    print ("------CONFIGURATION---------")
    print ("6. Toggle use of custom parametres - currently :" + " OFF" if defaultParams
else "6. Toggle use of custom parametres - currently : ON" )
    print ("7. Set custom parametres")
    print ("\n0. Quit")
    choice = input(" >>  ")
    exec_menu(choice)

    return
```

Czyszczenie konsoli, wyświetlenie menu w zależności od defaultParams

```
def exec_menu(ch):
    os.system('clear')

    if ch == '':
        main_menu()
    elif ch == '1':
        semicircular()
    elif ch == '2':
        eigenHeatMap()
    elif ch == '3':
        MarchenkoPastur()
    elif ch == '4':
        KappaAverages()
    elif ch == '5':
        meanAndStandard()
    elif ch == '6':
        global defaultParams
        defaultParams = not defaultParams
        main_menu()
    elif ch == '7':
        print("enter no. of matrices and seed(preferably between 100 and 200) divided
by whitespace")
```

```
            raw = input(" >>    ")
            param = [int(s) for s in raw.split() if s.isdigit()]
            print (param)
            print (type(param))
            setParametres(param[0],param[1])
            main_menu()
        elif ch == '0':
            sys.exit()
        else:
            print("invalid argument, try again")
        return
```

Czyszczenie konsoli, wywołanie odpowiedniej funkcji - w przypadku “7” konwersja stringa do tablicy intów, w przypadku “0” wywołanie systemowego wyjścia.

```
def setParametres(n,m):
    global seed
    global Matricescount
    Matricescount = n #600 for D. ; 1 for F.
    seed = m

def getMatrices():

    return [np.matrix(np.sqrt(1/seed) * np.random.randn(seed, seed)) for i in range(Matricescount)] # np.sqrt(1/n) * np.mat
```

Tworzenie listy losowych macierzy wykorzystując seed i liczbe macierzy wg przepisu z dokumentacji :

Two-by-four array of samples from N(3, 6.25):

```
>>> 2.5*np.random.randn(2,4)+3
```

```
#C.
def meanAndStandard():
    if defaultParams : setParametres(50,192)
    G = getMatrices()
    GxGt = [g * np.transpose(g) for g in G]
    t = [np.trace(gxgt) for gxgt in GxGt]
    #print(t)
    print("mean t:", end= ' ')
    print(np.mean(t))
```

```
        print("standard deviation t:", end= ' ')
        print(np.std(t))
```

Informacja o funkcji w sekcji MEAN AND STANDARD DEVIATION OF TRACES

```
#D.
def semicircular():

        if defaultParams : setParametres(600,192)
        G = getMatrices()
        H=[((g + np.transpose(g)) / np.sqrt(2)) for g in G]
        EV=[np.linalg.eigvalsh(h) for h in H]
        plt.hist(EV, normed=True, bins='sqrt')
        r = ss.semicircular.rvs(size=100, scale=2)
        plt.hist(r, bins='sqrt',normed=True,color='r',alpha=0.4)
        plt.show()
```

Informacja o funkcji w sekcji SEMICIRCULAR DISTRIBUTION

```
#E.
def eigenHeatMap():

        if defaultParams : setParametres(100,192)
        G = getMatrices()
        cEV = [np.linalg.eigvals(g) for g in G]
        Real = np.real(cEV)
        Imag = np.imag(cEV)
        plt.hist2d(Real.ravel(),Imag.ravel(),  bins=np.sqrt(seed  *  Matricescount),  normed  =
True)
        plt.show()
```

Informacja o funkcji w sekcji EIGENVALUES CONCENTRATION

```
#F.
def MarchenkoPastur():
        if defaultParams : setParametres(1,192)
        G = getMatrices()
        GxGt = [g * np.transpose(g) for g in G]
        W = [gxgt / np.trace(gxgt) for gxgt in GxGt]
        rEV = [np.linalg.eigvalsh(w) for w in W]
        x=[rev * seed for rev in rEV]
```

```
    plt.hist(x, alpha = 0.7, bins='sqrt', normed = False, range = [0,7])
    pl.ylim([0,100])
    plt.show()
```

Informacja o funkcji w sekcji MARCHENKO-PASTUR DISTRIBUTION

```
#G.
def KappaAverages():

    if defaultParams : setParametres(600,192)
    G = getMatrices()
    SV = [np.linalg.svd(g, compute_uv = False) for g in G]
    #Sigmamax = [np.amax(sv) for sv in SV]
    #Sigmamin = [np.amin(sv) for sv in SV]
    #print(Sigmamax)
    #print(Sigmamin)
    K = [np.amax(sv) / np.amin(sv) for sv in SV]
    #print(K)
    print("Kappa:")
    #print(K)
    print("mean K:")
    print(np.mean(K))
    print("std K:")
    print(np.std(K))
    #print("log(Kappa)")
    #print(np.log(K))
    print("mean logarithm:")
    print(np.mean(np.log(K)))
    print("logarithmic average:")
    print(np.exp(np.mean(np.log(K))))
    print("std of log average:")
    print(np.std(np.log(K)))
```

Informacja o funkcji w sekcji KAPPA AVERAGES COMPARISON

```
if __name__ == "__main__":
    # Launch main menu
    while True:
        main_menu()
```

Nieskończona pętla wywołująca menu