Projeto 1, 2 e PCA

Ricardo Tetti Camacho ricardotetti.camacho@usp.br
10728098

1. Projeto 1

1.1 Parte A

1.1.1 Autômatos Analisados

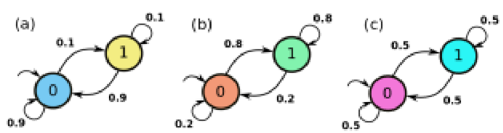

Figura 1: Figura retirada do CDT-22

1.1.2 Códigos

Parte do programa usado para gerar os padrões:

```python
class Padroes:
    def __init__(self,M):
        self.matriz = np.array(M)
        self.symbols = []
        self.padroes_gerados = []
    def deterministic(self, inter):
        self.symbols = [0,1]
        k = 0
        for i in range(inter):
            p = 0
            r = np.random.random_sample()
            for j in range(len(self.matriz)):
                p = p + self.matriz[k][j]
                if r < p:
                    self.padroes_gerados.append(self.symbols[k])
                    k = j
                    break
                else: continue
        return self.padroes_gerados

fig6a = Padroes([[0.9,0.1],[0.9,0.1]])
fig6b = Padroes([[0.2,0.8],[0.2,0.8]])
fig6c = Padroes([[0.5,0.5],[0.5,0.5]])

padroes_fig6a = fig6a.deterministic(200)
```

```
padroes_fig6b = fig6b.deterministic(200)
padroes_fig6c = fig6c.deterministic(200)
```

Parte usada para gerar o stem plot:

```
marks = np.arange(len(padroes_fig6a))

fig, (x1, x2, x3) = plt.subplots(3, sharex = True, sharey = True, figsize=(20,
    5))

x1.stem(marks, padroes_fig6a,'dimgrey', linefmt=None, markerfmt = '.')
x2.stem(marks, padroes_fig6b,'dimgrey' ,linefmt=None, markerfmt = '.')
x3.stem(marks, padroes_fig6c,'dimgrey' ,linefmt=None, markerfmt = '.')

plt.suptitle('Stem Plot')
plt.xlim(0,len(padroes_fig6a))
plt.xlabel('f')

plt.show()
```

Parte Usada para gerar o bar plot:

```
fig, (x1, x2, x3) = plt.subplots(3, sharex = True, sharey = True, figsize=(20,
    5))

x = np.arange(len(padroes_fig6a))
x1.bar(x, padroes_fig6a, color = 'dimgrey')
x2.bar(x, padroes_fig6b, color = 'dimgrey')
x3.bar(x, padroes_fig6c, color = 'dimgrey')

plt.suptitle('Barplot')
plt.xlabel('f')
pylab.xlim(0,len(padroes_fig6a))
pylab.ylim(0,1)

plt.show()
```

Parte para gerar o Square Wave plot:

```
fig, (x1, x2, x3) = plt.subplots(3, sharex = True, sharey = True, figsize=(20,
    5))

plt.suptitle("Square wave")
x1.step(arange(0,len(padroes_fig6a)),padroes_fig6a, color = 'dimgrey')
x2.step(arange(0,len(padroes_fig6b)),padroes_fig6b, color = 'dimgrey')
x3.step(arange(0,len(padroes_fig6c)),padroes_fig6c, color = 'dimgrey')

plt.show()
```

Parte para obter a média e desvio padrão da frequência relativa de símbolos $1's$.

```
def freq(automato, symbol):
    automato = np.array(automato)
    c = 0
    for i in automato:
        if i==symbol:
            c +=1
    return c/len(automato)

```

```
freq_fig6a = []
freq_fig6b = []
freq_fig6c = []
k = fig6a.deterministic(2000)
p = fig6b.deterministic(2000)
t = fig6c.deterministic(2000)
for i in range(200,2000,200):
    freq_fig6a.append(freq(k[:i],1))
    freq_fig6b.append(freq(p[:i],1))
    freq_fig6c.append(freq(t[:i],1))
    sns.distplot(freq_fig6a, hist = False, kde = True, color = 'darkblue')
    sns.distplot(freq_fig6b, hist = False, kde = True, color = 'lime')
    sns.distplot(freq_fig6c, hist = False, kde = True, color = 'tomato')

print('Media fig 6a:', np.mean(freq_fig6a))
print('Desvio fig 6a:', np.std(freq_fig6a))

print('Media fig 6b:', np.mean(freq_fig6b))
print('Desvio fig 6b:', np.std(freq_fig6b))

print('Media fig 6c:', np.mean(freq_fig6c))
print('Desvio fig 6c:', np.std(freq_fig6c))

plt.title('Density Plot')
plt.xlabel('f')
plt.xlim(0,1)
plt.ylabel('Density')
plt.show()
```

1.1.3 Plots

Para 500 interações

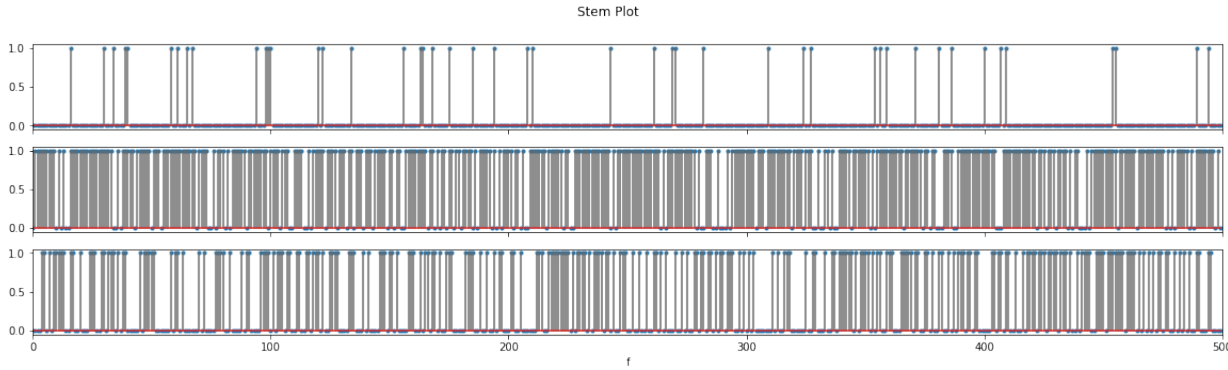

Figura 2: Stem plot, 500 interações; Figura 6a, 6b,6c

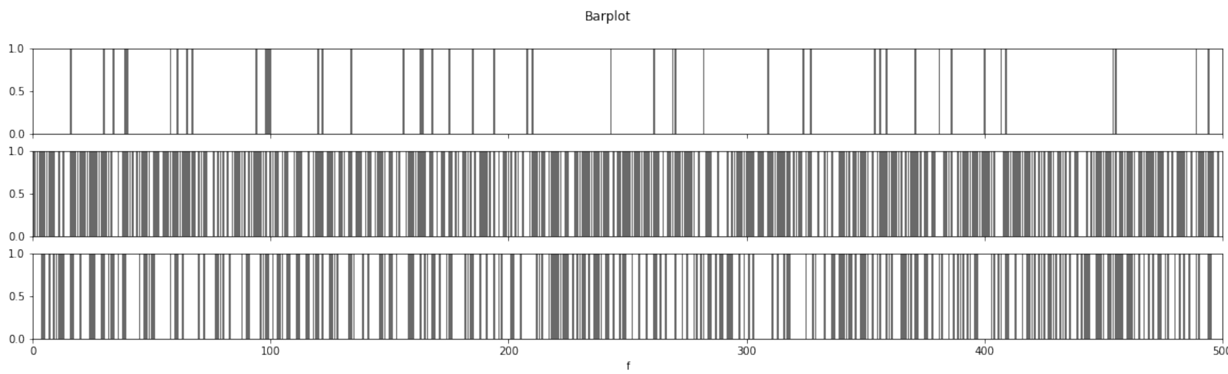

Figura 3: Bar plot, 500 interações; Figura 6a, 6b,6c

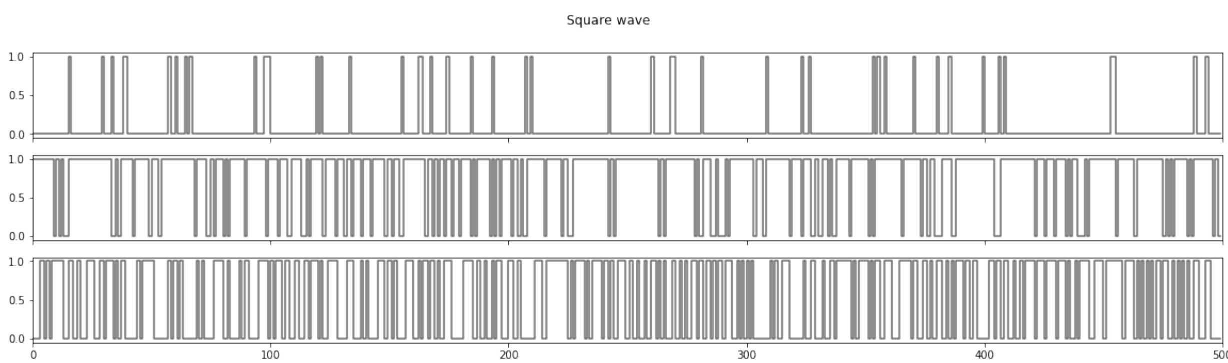

Figura 4: Square Wave, 500 interações; Figura 6a, 6b,6c

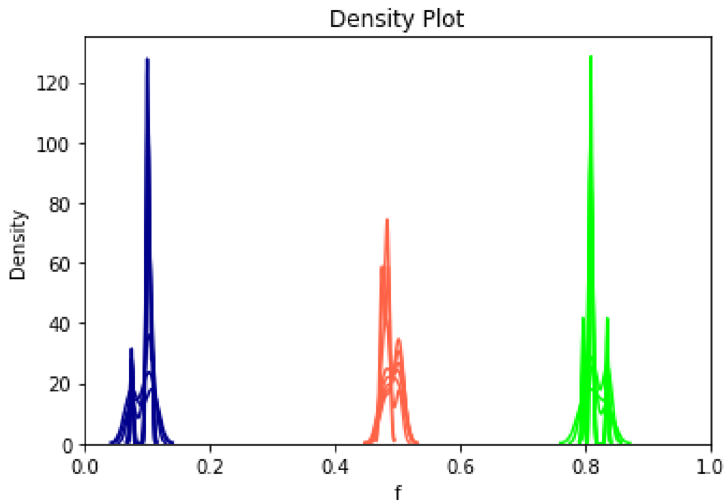

Figura 5: Densidade dos $1's$, azul representa o automato da figura 6a, vermelho a figura 6c e o verde a figura 6b

```
Média fig 6a: 0.09933492063492062
Desvio fig 6a: 0.004870384740271202
Média fig 6b: 0.7907349206349207
Desvio fig 6b: 0.005892153504033305
Média fig 6c: 0.4985365079365079
Desvio fig 6c: 0.012015512355410119
```

Figura 6: Resultados frequência dos $1's$

1.2 Parte B

1.2.1 Autômatos analisados

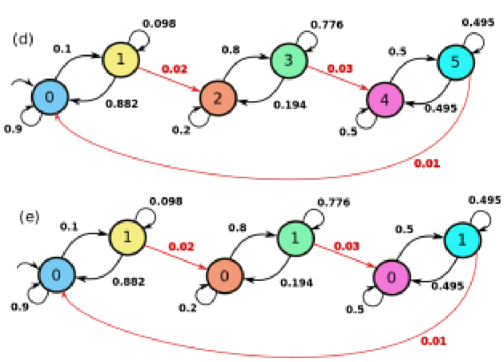

Figura 7: Figura retirada do CDT-22

1.2.2 Códigos

```
class Automatos:
    def __init__(self, M):
        self.matriz = np.array(M)
        self.symb = []
        self.padroes = []
    def symbols(self,S):
        self.symb = S
    def deterministic(self, inter):
        k = 0
        for i in range(inter):
            r = np.random.random_sample()
            a = 0
            for j in range(len(self.matriz)):
                a = a + self.matriz[k][j]
                if r < a:
                    self.padroes.append(self.symb[k])
                    k = j
                    break
        return self.padroes
fig_matrix = [
    [0.9,0.1,0,0,0,0],
    [0.882,0.098,0.02,0,0,0],
    [0,0,0.2,0.8,0,0],
    [0,0,0.194,0.776,0.03,0],
    [0,0,0,0,0.5,0.5],
    [0.01,0,0,0,0.495,0.495]
    ]
s_fig1d = [0,1,2,3,4,5]
s_fig1e = [0,1,0,1,0,1]

fig1d = Automatos(fig_matrix)
fig1d.symbols(s_fig1d)
padroes_fig1d = fig1d.deterministic(500)

fig1e = Automatos(fig_matrix)
fig1e.symbols(s_fig1e)
padroes_fig1e = fig1e.deterministic(500)
```

Para gerar o Stem Plot:

```
marks_fig1d = np.arange(len(padroes_fig1d))
fig, (x1, x2) = plt.subplots(2, sharex = True, figsize=(20, 5))
x1.stem(marks_fig1d, padroes_fig1d, 'dimgrey',markerfmt = '.')
x2.stem(marks_fig1d, padroes_fig1e, 'dimgrey',markerfmt = '.')
plt.xlim(0,len(padroes_fig1d))
plt.suptitle('Stem Plot')
plt.xlabel('f')
plt.show()
```

Para gerar o Bar Plot:

```
fig, (x1, x2) = plt.subplots(2, sharex = True, figsize=(20, 5))

x = np.arange(len(padroes_fig1d))
x1.bar(x, padroes_fig1d, color = 'dimgrey')
```

```
x2.bar(x, padroes_fig1e, color = 'dimgrey')

plt.suptitle('Barplot')
plt.xlim(0,len(padroes_fig1d))
plt.xlabel('f')

plt.show()
```

Para gerar o Square Wave plot:

```
fig, (x1, x2) = plt.subplots(2, sharex = True, figsize=(20, 5))

plt.suptitle("Square wave")
plt.xlim(0,len(padroes_fig1d))
x1.step(arange(0,len(padroes_fig1d)),padroes_fig1d, color = 'dimgrey')
x2.step(arange(0,len(padroes_fig1e)),padroes_fig1e, color = 'dimgrey')
plt.show()
```

Parte para obter a média e desvio padrão da frequência relativa de símbolos $1's$.

```
def freq(automato, symbol):
    automato = np.array(automato)
    c = 0
    for i in automato:
        if i==symbol:
            c +=1
    return c/len(automato)

freq_fig6d = []
freq_fig6e = []

k = fig1d.deterministic(2000)
p = fig1e.deterministic(2000)

for i in range(200,2000,200):
    freq_fig6d.append(freq(k[:i],1))
    freq_fig6e.append(freq(p[:i],1))
    sns.distplot(freq_fig6d, hist = False, kde = True, color = 'darkblue')
    sns.distplot(freq_fig6e, hist = False, kde = True, color = 'tomato')

print('Media fig 6d:', np.mean(freq_fig6d))
print('Desvio fig 6d:', np.std(freq_fig6d))

print('Media fig 6e:', np.mean(freq_fig6e))
print('Desvio fig 6e:', np.std(freq_fig6e))


plt.xlabel('f')
plt.ylabel('Density')
plt.title('Density Plot')
plt.show()
```

1.2.3 Plots

Para 500 interações

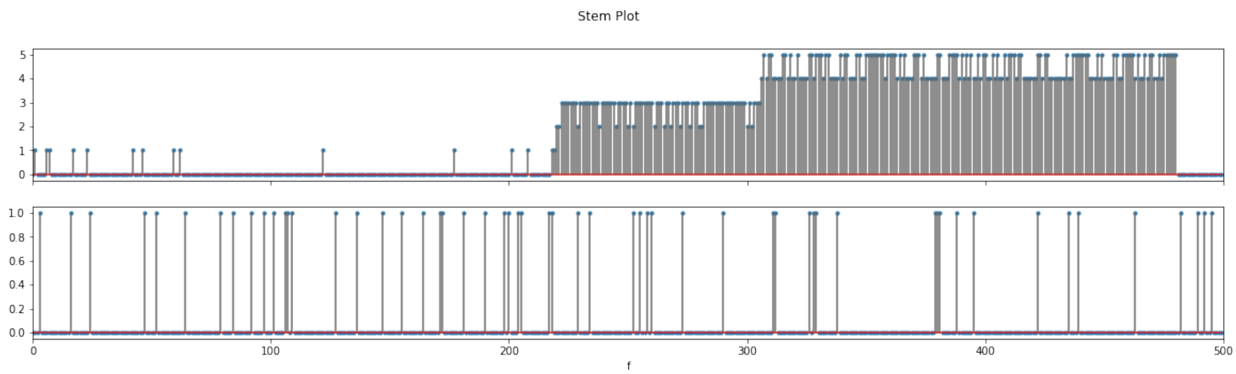

Figura 8: Stem Plot, 500 interações; Figura 6d, 6e

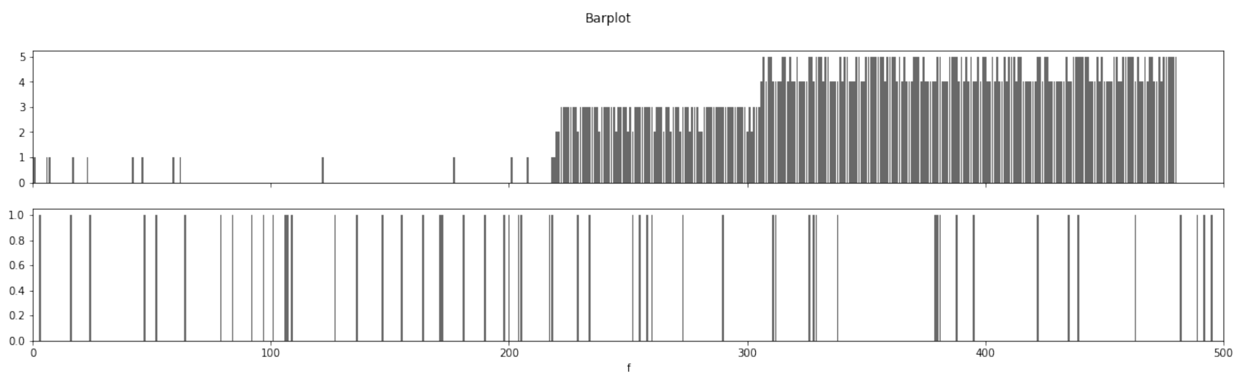

Figura 9: Bar Plot, 500 interações; Figura 6d, 6e

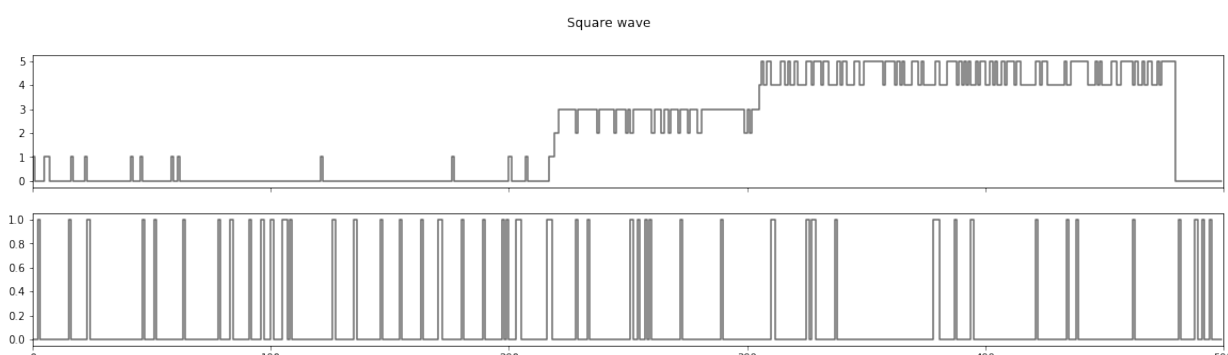

Figura 10: Square Wave, 500 interações; Figura 6d, 6e

```
Media fig 6d: 0.07111111111111112
Desvio fig 6d: 0.013375223547226103
Media fig 6e: 0.23334259259259257
Desvio fig 6e: 0.06948821336731983
```

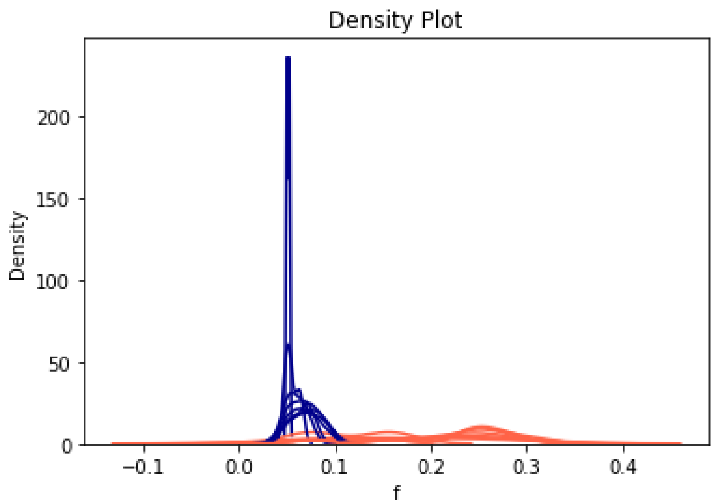

Figura 11: Densidade dos $1's$, azul representa o automato da figura 6d e o vermelho a figura 6e, junto com as suas médias e desvios padrão.

2. Projeto 2

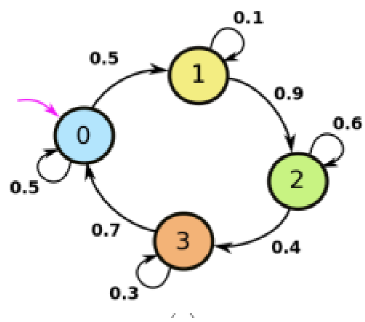

Figura 12: Automato analisado. Figura retirada do CDT-23

Bibliotecas usadas:

```
from random import random
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy import stats
import pylab
import math
```

Para gerar os padrões da série temporal foi usado o seguinte código:

```
class Automatos:
    def __init__(self, M):
        self.matriz = np.array(M)
        self.symb = []
        self.padroes = []
    def symbols(self,S):
        self.symb = S
    def deterministic(self, inter):
        k = 0
        for i in range(inter):
            r = np.random.random_sample()
            a = 0
            for j in range(len(self.matriz)):
                a = a + self.matriz[k][j]
                if r < a:
                    self.padroes.append(self.symb[k])
                    k = j
                    break
        return self.padroes

M = [[0.5,0.5,0,0],
    [0,0.1,0.9,0],
    [0,0,0.6,0.4],
    [0.7,0,0,0.3]
    ]

auto = Automatos(M)
```

```
auto.symbols([0,1,2,3])
padroes_fig2 = auto.deterministic(200)
```

Parte para gerar os split signals:

```
def main():
    split_fig2 = split_signal()
    split_fig2.split(padroes_fig2)
    split_fig2.relative_frequency()
    split_fig2.plots()

class split_signal:
    def __init__(self):
        self.list_zero = []
        self.list_one = []
        self.list_two = []
        self.list_three = []
        self.lista_burst = []
        self.relative_fre = []
    def split(self, lista):
        for i in range(len(lista)):
            if lista[i] == 0:
                self.list_zero.append(1)
            else:
                self.list_zero.append(0)
        for i in range(len(lista)):
            if lista[i] == 1:
                self.list_one.append(1)
            else:
                self.list_one.append(0)
        for i in range(len(lista)):
            if lista[i] == 2:
                self.list_two.append(1)
            else:
                self.list_two.append(0)
        for i in range(len(lista)):
            if lista[i] == 3:
                self.list_three.append(1)
            else:
                self.list_three.append(0)
    def relative_frequency(self):
        for i in range (len(self.list_zero)):
            self.relative_fre.append((self.list_zero[i])/(len(self.list_zero)))
    def plots(self):
        fig, axs = plt.subplots(4, sharex = True)
        fig.suptitle('Split signals')
        x = np.arange(len(padroes_fig2))
        axs[0].bar(x, self.list_zero, color = 'dimgrey')
        axs[1].bar(x, self.list_one, color = 'dimgrey')
        axs[2].bar(x, self.list_two, color = 'dimgrey')
        axs[3].bar(x, self.list_three, color = 'dimgrey')
        pylab.xlim(0,len(padroes_fig2))
        pylab.ylim(0,1)
```

```python
if __name__ == "__main__":
    main()
```

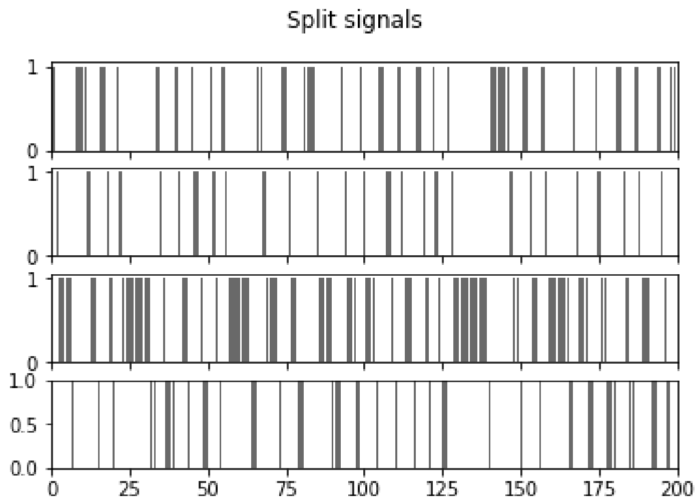

Figura 13: Split Signal

2.1 Parte 1

Número de bursts e média, desvio padrão, entropia e evenness dos tamanhos dos bursts em cada split signal;

2.1.1 Códigos

```python
def main():
    burst_fig2 = burst()
    burst_fig2.split(padroes_fig2)
    burst_fig2.relative_frequency()
    burst_fig2.burst_(len(padroes_fig2) - 1)
    burst_fig2.media_burst()
    burst_fig2.desvio_burst()
    burst_fig2.entropia()
    burst_fig2.plot()

class burst(split_signal):
    def __init__(self):
        split_signal.__init__(self)
        self.lista_burst = []
    def burst_(self, interacoes):
        i = 1
        L = self.list_zero
        while (i<=interacoes):
            if(L[i] == 1):
                i_0 = 1
                while (L[i] == 1) and (i<interacoes):
                    i+=1
```

```
                if (i == M)and(L[i] == 1):
                    i = M-1
                bs = i-i_0
                if(bs>0):
                    self.lista_burst.append(bs)
            i += 1
    def media_burst(self):
        media = np.mean(self.lista_burst)
        print("Media:",media)
    def desvio_burst(self):
        desvio = np.std(self.lista_burst)
        print("Desvio padrao:", desvio)
    def entropia(self):
        indices = []
        for i in range(len(self.relative_fre)):
            if self.relative_fre[i] != 0:
                k = (self.relative_fre[i])*(math.log(self.relative_fre[i],2))
                indices.append(k)
        epsilon = (-1)*sum(indices)
        eta = 2**((-1)*sum(indices))
        print("Entropia:", epsilon)
        print("Evenness:", eta)
    def plot(self):
        plt.title("Burst")
        contador = []
        for i in range(len(self.lista_burst)):
            contador.append(1)
        plt.stem(self.lista_burst, contador, markerfmt = '.')
        plt.show()


if __name__ == '__main__':
    main()
```

```
Média: 102.6
Desvio padrao: 60.08416319352935
Entropia: 2.3695954188301678
Evenness: 5.16796184556994
```

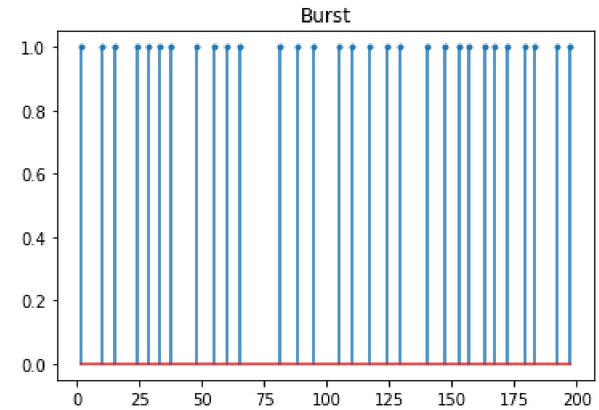

Figura 14: Burst

2.2 Parte 2

Número de distâncias intersímbolos e média, desvio padrão, entropia e evenness das distâncias intersímbolos em cada split signal;

2.2.1 Códigos

```
def main():
    intersymbol_fig2 = inter_symbol()
    intersymbol_fig2.split(padroes_fig2)
    intersymbol_fig2.distance(len(padroes_fig2))
    intersymbol_fig2.distancias()
    intersymbol_fig2.number_distance()
    intersymbol_fig2.media_desvio()
    intersymbol_fig2.entropia_evenness()
    intersymbol_fig2.plot()

class inter_symbol(split_signal):
    def __init__(self):
        split_signal.__init__(self)
        self.list_inter_symbol = []
        self.list_distancias = []
    def distance(self, interacoes):
        L = self.list_zero
        for i in range(interacoes):
            if(L[i] == 1):
                bs = 0
                while(L[i] == 0) & (i<interacoes):
                    i += 1
                if (i == interacoes) & (L[i] == 1):
                    bs = interacoes
```

```
                elif (i == interacoes) & (L[i] == 0):
                    bs = 0
                elif (i<interacoes) & (L[i] == 1):
                    bs = i
                if (bs>0):
                    self.list_inter_symbol.append(bs)
            i += 1
    def number_distance(self):
        print("Numero de distancias intersimbolos:", len(self.list_inter_symbol))
    def distancias(self):
        for i in range(0,(len(self.list_inter_symbol)-1)):
            k = self.list_inter_symbol[i+1] - self.list_inter_symbol[i]
            self.list_distancias.append(k)
    def media_desvio(self):
        media_ = sum(self.list_distancias)/len(self.list_distancias)
        desvio_ = np.std(self.list_distancias)
        print("Media:",media_)
        print("Desvio:", desvio_)
    def entropia_evenness(self):
        k = []
        indices = []
        for i in range(len(self.list_distancias)):
            k.append(self.list_distancias[i]/len(self.list_distancias))
        for i in range(len(k)):
            if k[i] != 0:
                indices_ = ((k[i])*(math.log(k[i],2)))
                indices.append(indices_)
        entropia = -(sum(indices))
        evenness = 2**(entropia)
        print("Entropia:", entropia)
        print("Evenness:", evenness)
    def plot(self):
        plt.title("Inter Symbol distance")
        contador = []
        for i in range(len(self.list_inter_symbol)):
            contador.append(1)
        plt.stem(self.list_inter_symbol,contador, markerfmt = '.')
        plt.show()


if __name__ == '__main__':
    main()
```

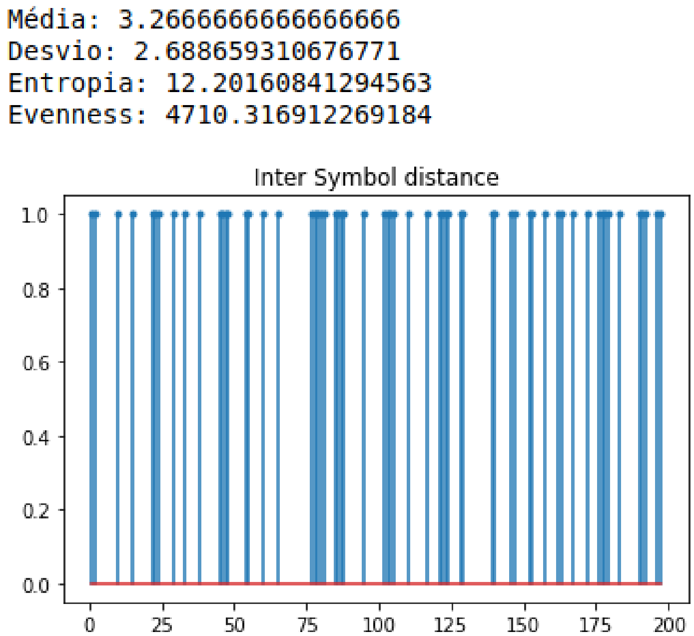

Figura 15: Intersymbol distance

2.3 Parte 3

Média e desvio padrão das magnitudes do espectro de potência da transformada de Fourier discreta de cada split signal (pode usar rotina para FFT);

2.3.1 Códigos

```
def main():
    fourier_fig2 = fourier_transform()
    fourier_fig2.split(padroes_fig2)
    fourier_fig2.transform(len(padroes_fig2))
    fourier_fig2.media()
    fourier_fig2.desvio()
    fourier_fig2.plot()

class fourier_transform(split_signal):
    def __init__(self):
        split_signal.__init__(self)
        self.transformada = []
    def transform(self, interacoes):
        L = self.list_zero
        self.transformada = np.fft.fft(L, (interacoes-1))
    def media(self):
        media = np.mean(self.transformada)
        print("Media:", media)
    def desvio(self):
        desvio_padrao = np.std(self.transformada)
        print("Desvio padrao:", desvio_padrao)
```

```
    def plot(self):
        plt.title("Fourier Transform")
        modulo = []
        for i in range(len(self.transformada)):
            modulo.append((np.abs(self.transformada[i]))**2)
        plt.stem(modulo, markerfmt = '.')
        plt.show()

if __name__ == "__main__":
    main()
```

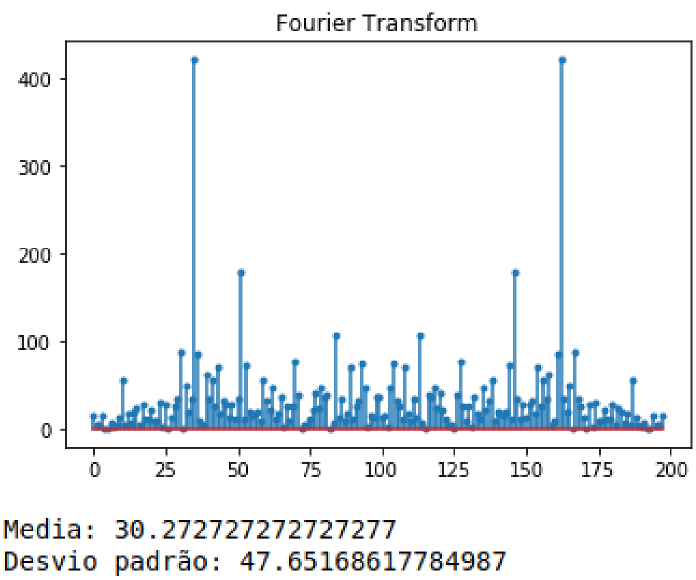

Figura 16: Power Spectrum

2.4 Parte 4

Média e desvio do grau e coeficiente de aglomeração de grafos dos sinais originais, obtidos pelo método de visibilidade, que deve ser implementado;

2.4.1 Códigos

```
def main():
    network_fig2 = network_based()
    network_fig2.visibility(padroes_fig2)
    network_fig2.average_degree()
    network_fig2.clustering_coefficient()
    #network_fig2.plot()
    network_fig2.tentativa_plot()

class network_based(split_signal):
    def __init__(self):
        split_signal.__init__(self)
        self.M = np.zeros((len(padroes_fig2),len(padroes_fig2)), dtype=int)
    def visibility(self, L):
        for j in range(1,len(padroes_fig2)):
            for i in range(0, j-1):
                flag = 1
                k = i + 1
                while (k <= j-1) and (flag == 1):
                    aux = L[j] + (L[i]-L[j])*(j-k)/(j-i)
                    if (L[k] >= aux):
                        flag = 0
                    k += 1
                if (flag == 1):
                    self.M[i][j] = 1
                    self.M[j][i] = 1
    def average_degree(self):
        edges = 0
        for i in range(len(padroes_fig2)):
            for j in range(len(padroes_fig2)):
                if self.M[i][j] == 1:
                    edges += 1
        average = edges/(len(padroes_fig2))
        standard_deviation = self.M.std()
        print("Media:",average)
        print("Desvio padrao:",standard_deviation)
    def clustering_coefficient(self):
        degree_node = []
        contador = 0
        for i in range(len(self.M)):
            degree_node.append(sum(self.M[i]))
            contador += 1
        clustering_coefficient = (sum(degree_node)*2)/(contador*(contador-1))
        print("Coeficiente de aglomeracao:",clustering_coefficient)
    def tentativa_plot(self):
        M_array = np.array(self.M)
        plt.matshow(M_array)
        plt.title("Visibility procedure")
        ax = plt.gca()
```

```python
        ax.axes.xaxis.set_visible(False)
        ax.axes.yaxis.set_visible(False)
        plt.show()
    def plot(self):
        x = []
        y = []
        for i in range(len(padroes_fig2)):
            for j in range(len(padroes_fig2)):
                if self.M[i][j] == 1:
                    x.append(i)
                    y.append(j)
        #print(x)
        plt.title("Visibility procedure")
        plt.scatter(x,y,s = 1.5)
        plt.show()

if __name__ == '__main__':
    main()
```

```
Média: 2.77
Desvio padrão: 0.1168682056848654
Coeficiente de aglomeração: 0.0278391959798995
```

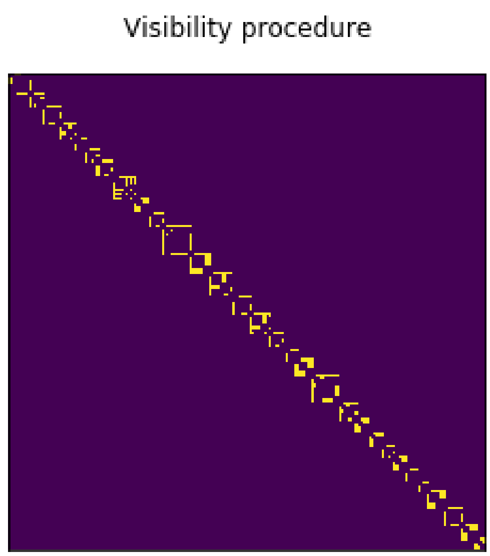

Figura 17: Visibility procedure

3. PCA

Códigos

```
def main():
    pca_preparo = PCA()
    pca_preparo.aleatorios(600)
    pca_preparo.circulo()
    pca_preparo.alongamento()
    pca_preparo.rotacao()
    pca_preparo.covariancia()
    pca_preparo.subplot()
    pca_preparo.eigen()
    pca_preparo.plot()


class PCA:
    def __init__(self):
        self.aleatorios_x = []
        self.aleatorios_y = []
        self.circulo_x = []
        self.circulo_y = []
        self.circulo_y_alongado = []
        self.rotacao_ = []
        self.cov_matrix = []
        self.eigenvalue = []
        self.eigenvector = []
        self.new_circulo_x = []
        self.new_circulo_y = []

    def aleatorios(self,interacoes):
        for i in range(interacoes):
            k = random.uniform(-1,1)
            h = random.uniform(-1,1)
            self.aleatorios_x.append(k)
            self.aleatorios_y.append(h)

    def circulo(self):
        for i in range(len(self.aleatorios_x)):
            k = math.sqrt((self.aleatorios_y[i])**2 + (self.aleatorios_x[i])
    **2)
            if k <= 1:
                self.circulo_x.append(self.aleatorios_x[i])
                self.circulo_y.append(self.aleatorios_y[i])
            else:
                continue

    def alongamento(self):
        for i in range(len(self.circulo_y)):
            self.circulo_y_alongado.append(0)
        for i in range(len(self.circulo_y)):
            self.circulo_y_alongado[i] = 0.2*self.circulo_y[i]

    def rotacao(self):
        k = np.radians(30)
        rot = [[np.cos(k),np.sin(k)],[np.sin(k),np.cos(k)]]
```

```python
        self.rotacao_ = np.dot(rot,[self.circulo_x,self.circulo_y_alongado])

    def covariancia(self):
        self.cov_matrix = np.cov(self.rotacao_)

    def eigen(self):
        self.eigenvalue,self.eigenvector = LA.eig(self.cov_matrix)
        self.eigenvalue.sort()
        self.eigenvalue = self.eigenvalue[::-1]
        eta = self.eigenvalue[0]/(sum(self.eigenvalue))
        print('Autovetor 1:',self.eigenvector[0])
        print('Lambda 1:',self.eigenvalue[0])
        print('Autovetor 2:', self.eigenvector[1])
        print('Lambda 2:',self.eigenvalue[1])
        print('Eta:', eta)

    def plot(self):
        origin = [0,0]
        plt.title("PCA")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.quiver(*origin, *self.eigenvector[:,0],color = 'b' ,
            width = 0.004, scale_units='xy', scale=2)
        plt.quiver(*origin, *self.eigenvector[:,1],color = 'r' ,
            width = 0.004, scale_units='xy', scale=2)
        plt.axis('equal')
        plt.scatter(self.rotacao_[0],self.rotacao_[1], s = 2, color = 'dimgrey')
        plt.xlim(-1,1)
        plt.ylim(-1,1)
        plt.show()

    def subplot(self):
        fig, ax = plt.subplots(1,3, figsize=(6*3, 6), sharex = True, sharey = True)
        plt.xlim(-1,1)
        plt.ylim(-1,1)
        fig.suptitle("Distribuicao")
        ax[0].scatter(self.circulo_x,self.circulo_y, s = 2)
        ax[0].plot()
        ax[1].scatter(self.circulo_x,self.circulo_y_alongado, s = 2)
        ax[2].scatter(self.rotacao_[0],self.rotacao_[1], s = 2)
        plt.show()

if __name__ == '__main__':
    main()
```

3.1 Visualização dos dados, para ver se está parecido com Figura 9, CDT-24

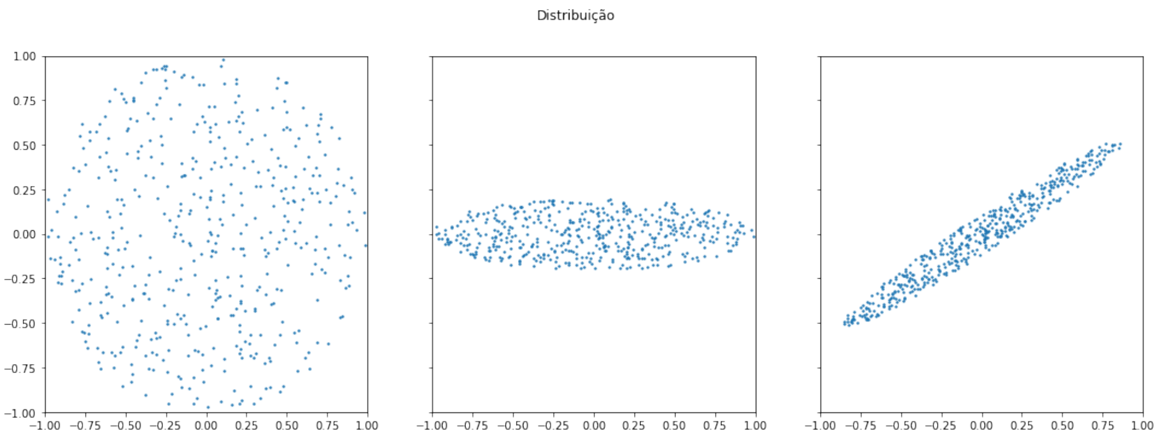

Figura 18: Distribuição

3.2 Resultado final

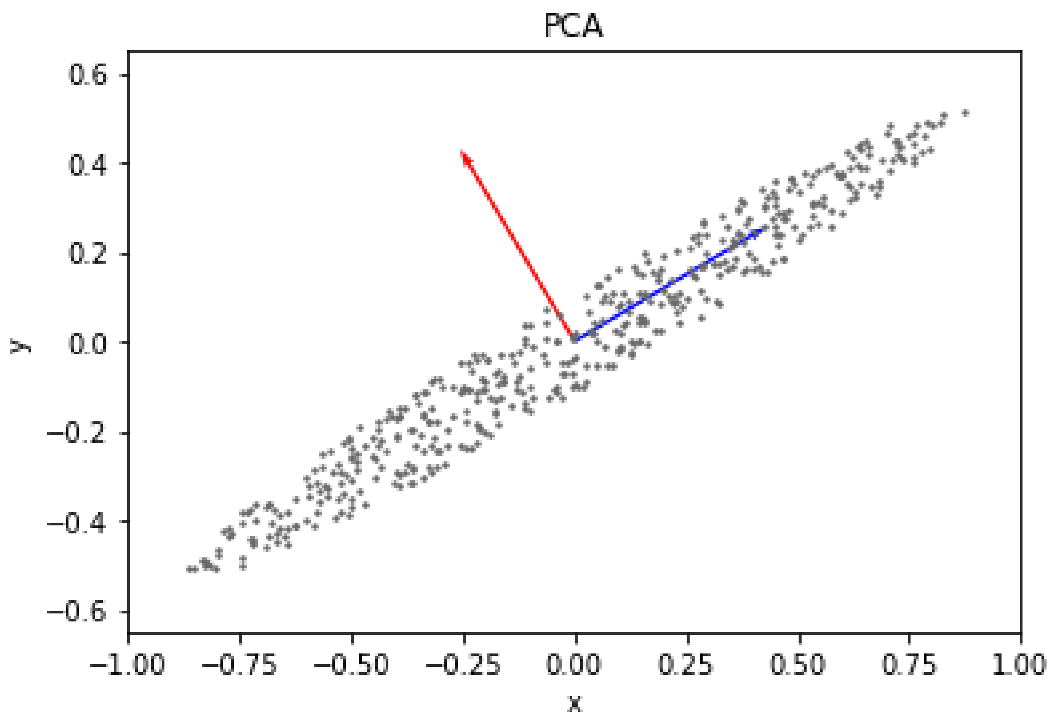

Figura 19: PCA