Lista de Exercícios - Matemática Computacional I

Adriano Pereira Almeida

In [1]:
```
%matplotlib inline
from cappy239 import powernoise, logistic_map
from cappy239 import plot_chaotic, pmodel
from cappy239 import power_spectrum_density
import matplotlib.pyplot as plt
import numpy as np
import math
import statistics as s
import seaborn as sns
import warnings
import os
warnings.filterwarnings('ignore')
import waipy
import os
from IPython.display import Image
import scipy.stats
import matplotlib.image as mpimg
from matplotlib import rcParams
import pylab
```

1 - Simulação de Sinais Estocásticos com $N = 2^{12}$ valores de medidas.

1.1 - Utilize o algoritmo powenoise.m e gere os seguintes ruídos $1/f^{\beta}$:

S1: $\beta = 0$ (white noise)

S2: $\beta = 1$ (pink noise)

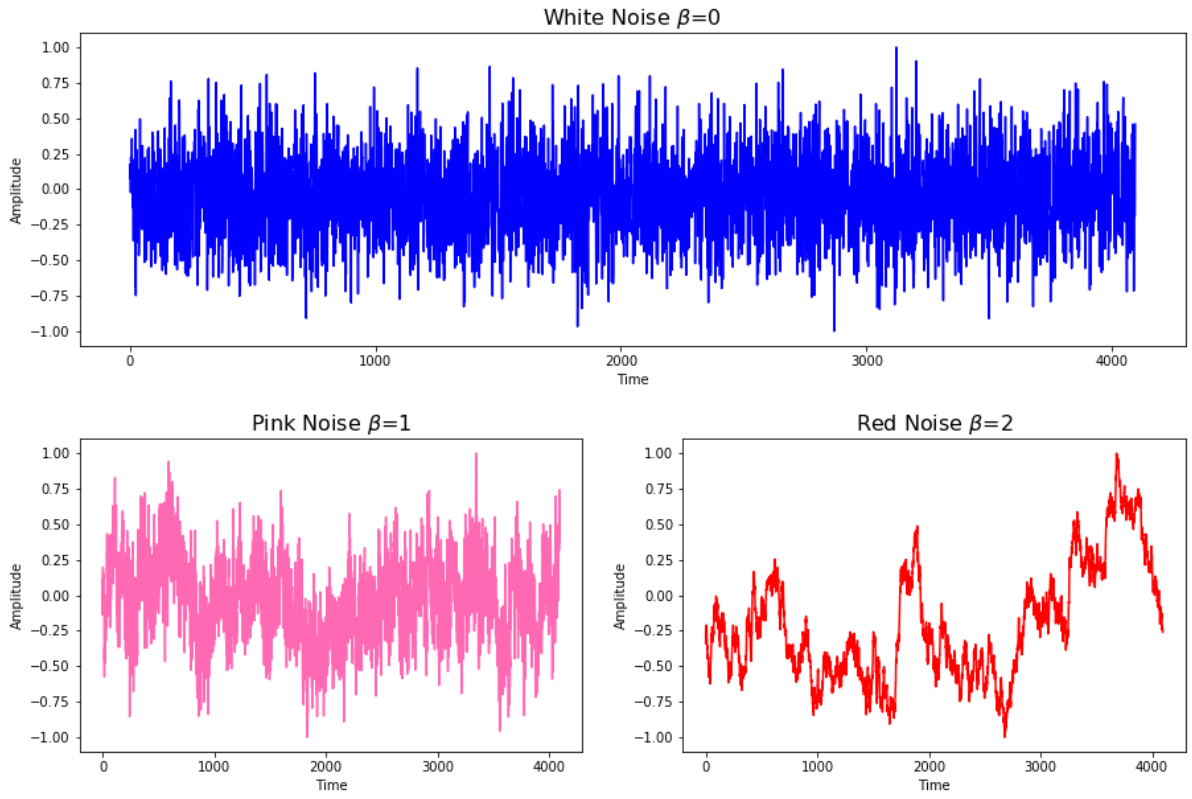
S3: $\beta = 2$ (red noise)

In [2]:
```
# S1 = powernoise(beta=0, N=2**12)
# S2 = powernoise(beta=1, N=2**12)
# S3 = powernoise(beta=2, N=2**12)

# S4 = logistic_map(n=2**12, rho=4, a0=0.001)

# S5 = S1+S4
# S5 = (S5-S5.mean())/s.stdev(S5)

# S6 = S2+S4
# S6 = (S6-S6.mean())/s.stdev(S6)

# S7 = S3+S4
# S7 = (S7-S7.mean())/s.stdev(S7)

# S8 = pmodel(noValues=4096, p=0.52, slope=-1.66)
# S9 = pmodel(noValues=4096, p=0.62, slope=-0.45)
# S10 = pmodel(noValues=4096, p=0.72, slope=-0.75)
```

In [3]:
```
# if not os.path.exists('series'):
#     os.makedirs('series')
# np.savetxt('./series/S1.txt', S1, newline='\n')
# np.savetxt('./series/S2.txt', S2, newline='\n')
# np.savetxt('./series/S3.txt', S3, newline='\n')
# np.savetxt('./series/S4.txt', S4, newline='\n')
# np.savetxt('./series/S5.txt', S5, newline='\n')
# np.savetxt('./series/S6.txt', S6, newline='\n')
# np.savetxt('./series/S7.txt', S7, newline='\n')
# np.savetxt('./series/S8.txt', S8, newline='\n')
# np.savetxt('./series/S9.txt', S9, newline='\n')
# np.savetxt('./series/S10.txt', S10, newline='\n')
```

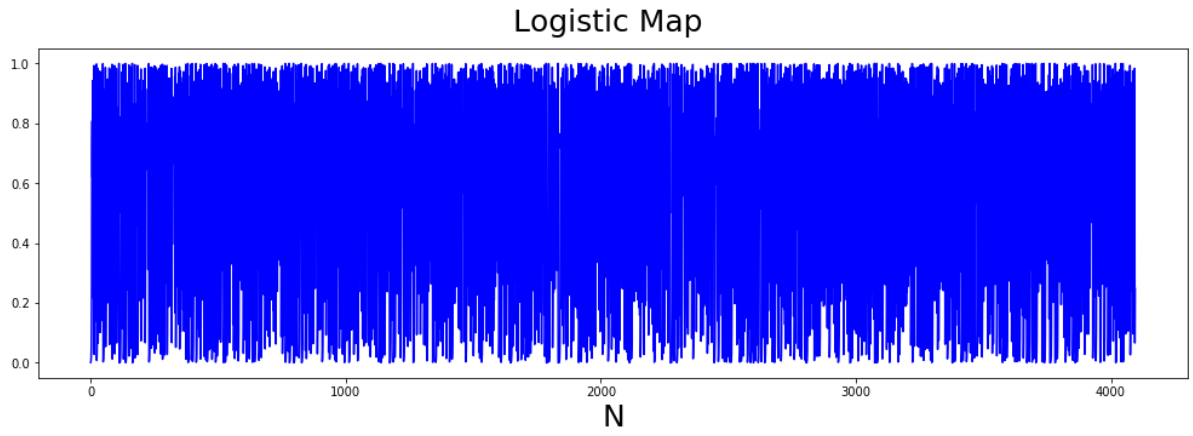
In [4]:
```
S1 = np.loadtxt('./series/S1.txt')
S2 = np.loadtxt('./series/S2.txt')
S3 = np.loadtxt('./series/S3.txt')
S4 = np.loadtxt('./series/S4.txt')
S5 = np.loadtxt('./series/S5.txt')
S6 = np.loadtxt('./series/S6.txt')
S7 = np.loadtxt('./series/S7.txt')
S8 = np.loadtxt('./series/S8.txt')
S9 = np.loadtxt('./series/S9.txt')
S10 = np.loadtxt('./series/S10.txt')

fig = plt.figure(figsize=(15, 10))
fig.subplots_adjust(hspace = .3, wspace = .2)
W = fig.add_subplot(2, 1, 1)
W.set_ylabel('Amplitude')
W.set_xlabel('Time')
W.plot(S1, color='#0100FF')
W.set_title(r'White Noise $\beta$=0', fontsize='16')

P = fig.add_subplot(2, 2, 3)
P.set_ylabel('Amplitude')
P.set_xlabel('Time')
P.plot(S2, color='hotpink')
P.set_title(r'Pink Noise $\beta$=1', fontsize='16')

R = fig.add_subplot(2, 2, 4)
R.set_ylabel('Amplitude')
R.set_xlabel('Time')
R.plot(S3, color='red')
R.set_title(r'Red Noise $\beta$=2', fontsize='16')
```

Out[4]:
```
Text(0.5, 1.0, 'Red Noise $\\beta$=2')
```



1.2 - S4: Série Caótica a partir do Mapeamento Quadrático (Logístico) para $\rho = 4$, considerando $Ao = 0.001$.

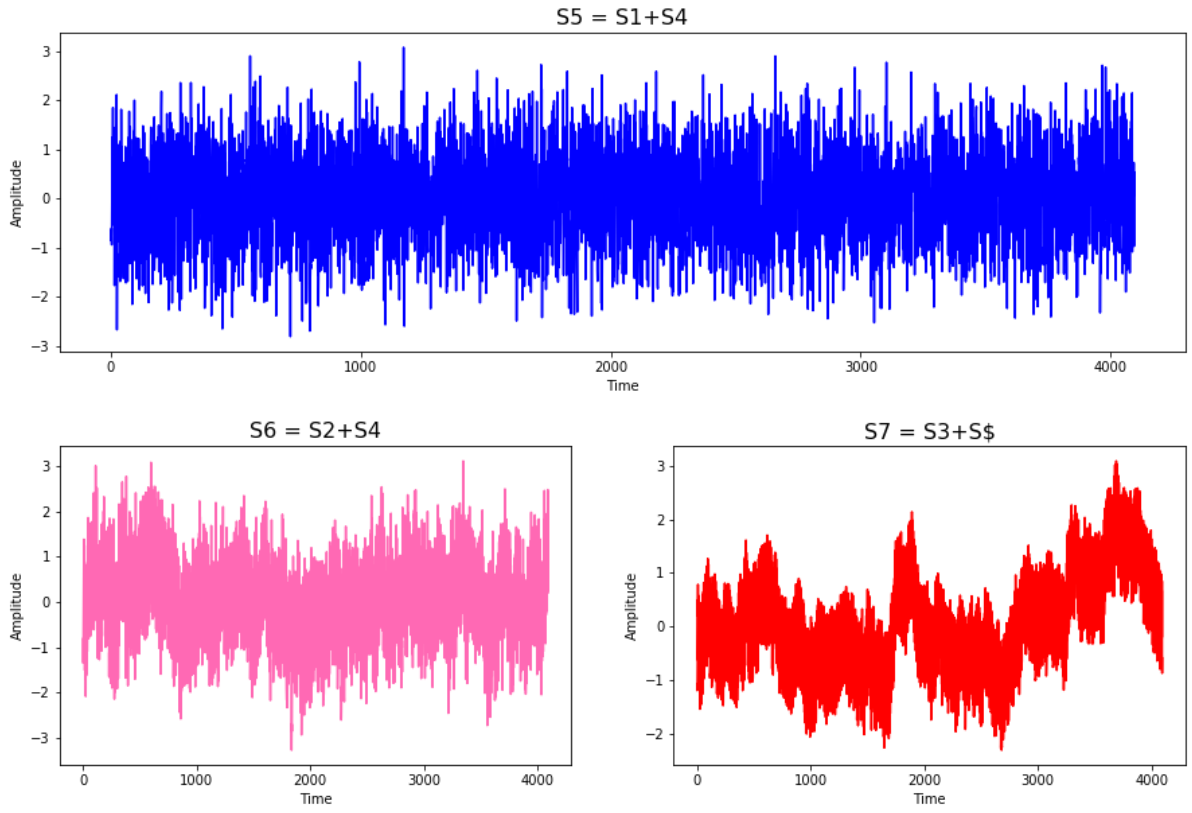
In [5]:
```
plot_chaotic(S4)
```



1.3. Gere o algoritmo em python para somar e normalizar com $< A >= 0$, dois sinais com o mesmo tamanho e construa a série S5=S1+S4, S6=S2+S4 e S7=S3+S4.

In [6]:
```
fig = plt.figure(figsize=(15, 10))
fig.subplots_adjust(hspace = .3, wspace = .2)
W = fig.add_subplot(2, 1, 1)
W.set_ylabel('Amplitude')
W.set_xlabel('Time')
W.plot(S5, color='#0100FF')
W.set_title(r'S5 = S1+S4', fontsize='16')

P = fig.add_subplot(2, 2, 3)
P.set_ylabel('Amplitude')
P.set_xlabel('Time')
P.plot(S6, color='hotpink')
P.set_title(r'S6 = S2+S4', fontsize='16')

R = fig.add_subplot(2, 2, 4)
R.set_ylabel('Amplitude')
R.set_xlabel('Time')
R.plot(S7, color='red')
R.set_title(r'S7 = S3+S$', fontsize='16')
```

Out[6]:
```
Text(0.5, 1.0, 'S7 = S3+S$')
```



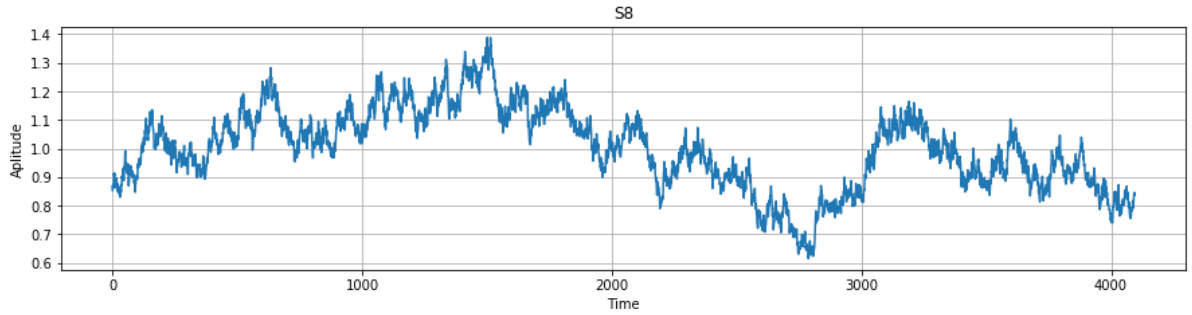
1.4 Utilize o algoritmo `pmodel.m` e gere os seguintes sinais com $N = 2^{12}$ valores de medidas:

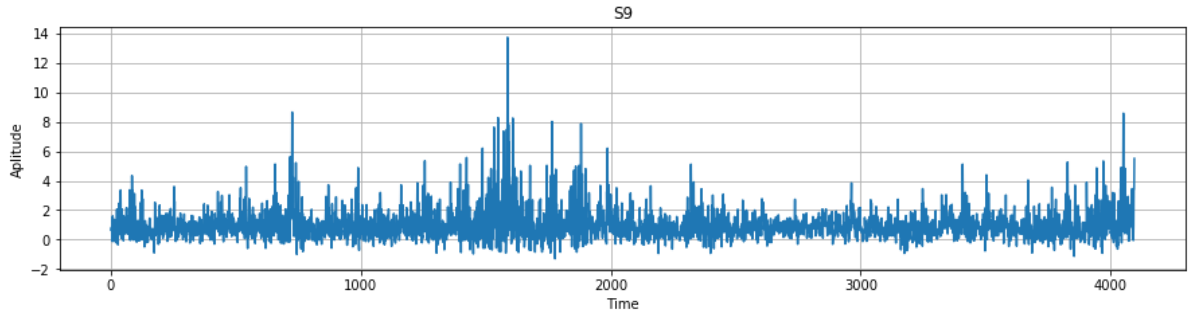
S8: $p = 0.52, \beta = -1.66$

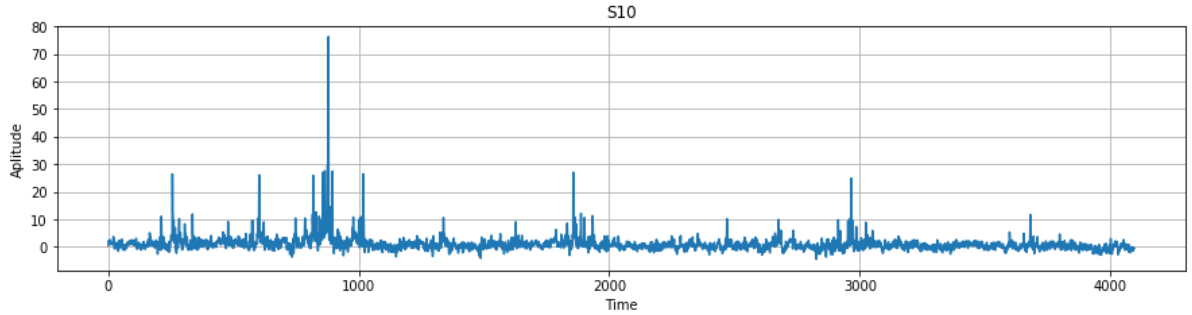
S9: $p = 0.62, \beta = -0.45$

S10: $p = 0.72, \beta = -0.75$

In [7]:
```
fig = plt.figure(figsize=(15, 12))
fig.subplots_adjust(hspace = .3, wspace = .2)
ps8 = fig.add_subplot(3, 1, 1)
ps8.set_title('S8')
ps8.set_ylabel('Aplitude')
ps8.set_xlabel('Time')
ps8.grid()
ps8.plot(S8)

fig = plt.figure(figsize=(15, 12))
fig.subplots_adjust(hspace = .3, wspace = .2)
ps9 = fig.add_subplot(3, 1, 1)
ps9.set_title('S9')
ps9.set_ylabel('Aplitude')
ps9.set_xlabel('Time')
ps9.grid()
ps9.plot(S9)

fig = plt.figure(figsize=(15, 12))
fig.subplots_adjust(hspace = .3, wspace = .2)
ps10 = fig.add_subplot(3, 1, 1)
ps10.set_title('S10')
ps10.set_ylabel('Aplitude')
ps10.set_xlabel('Time')
ps10.grid()
ps10.plot(S10)

fig.show()
```



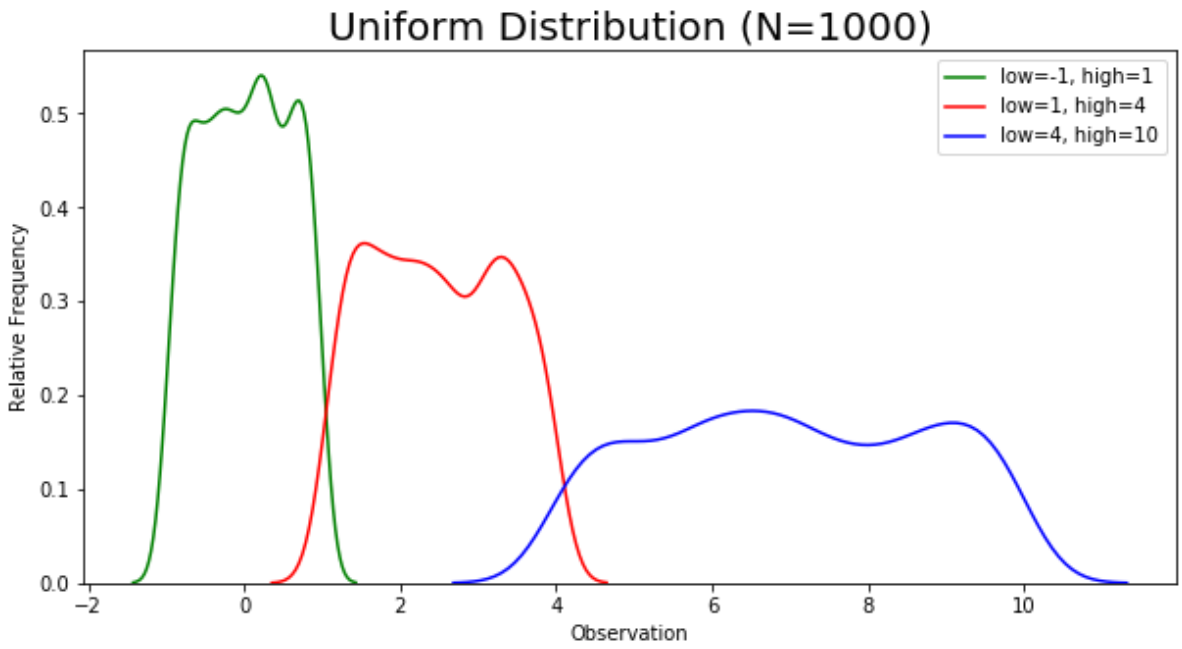




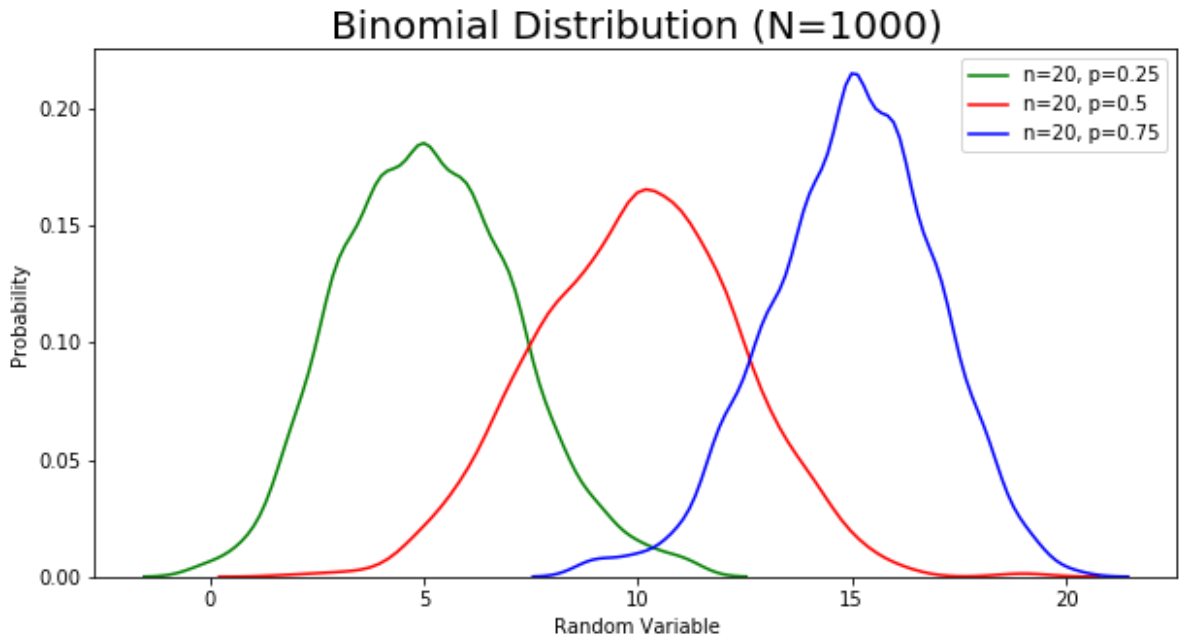
2 - Distribuições de Probabilidades 2.1 - Escreva um programa em python que, para uma amostra com N resultados, ajuste as seguintes PDFs:

(i)Uniforme, (ii) Binomial, (iii) Beta, (iv) Laplace, (v) Gamma, (vi) Expoencial (v) Qui-quadrado, (vi) Cauchy, (vii) Beta e (viii) Gaussiana (Normal)

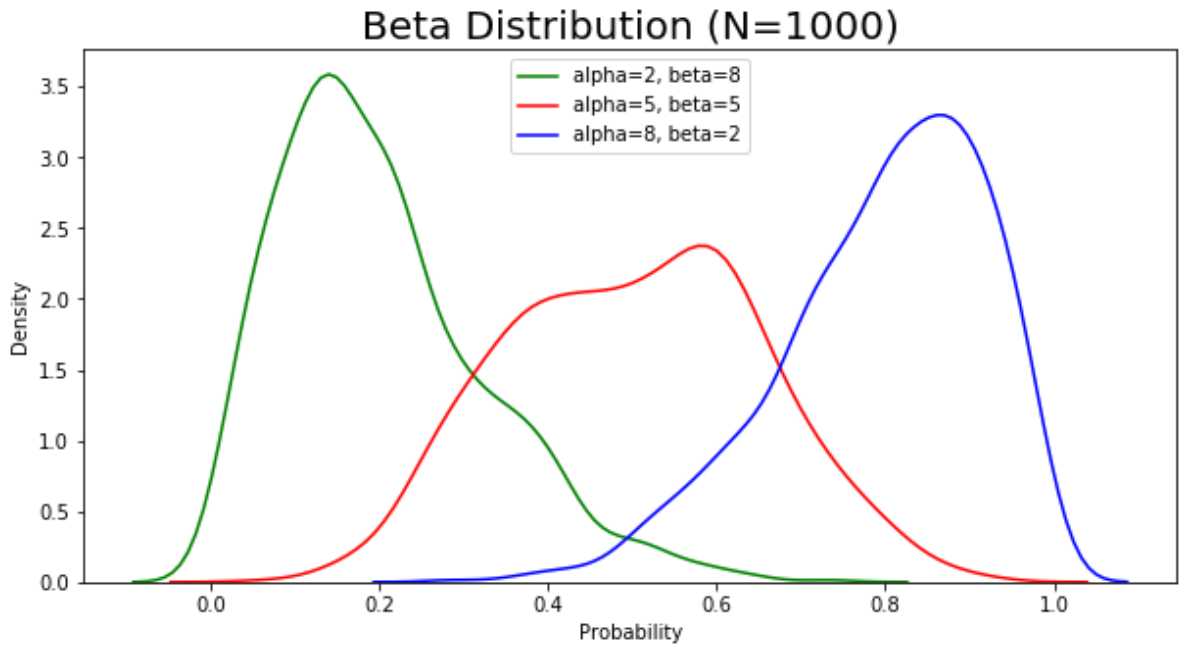
In [8]:
```
plt.figure(figsize=(10, 5))

low, high = -1, 1
uniform = np.random.uniform(low, high, 1000)
sns.distplot(uniform, color='green', label='low=-1, high=1', hist=False

low, high = 1, 4
uniform = np.random.uniform(low, high, 1000)
sns.distplot(uniform, color='red', label='low=1, high=4', hist=False)

low, high = 4, 10
uniform = np.random.uniform(low, high, 1000)
sns.distplot(uniform, color='blue', label='low=4, high=10', hist=False)

plt.title('Uniform Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Observation')
plt.ylabel('Relative Frequency')
plt.legend(loc='best')
plt.show()
# Reference: https://www.researchgate.net/figure/Histogram-of-uniform-di
# line-for-the-fitted-distribution_fig1_228390422
```



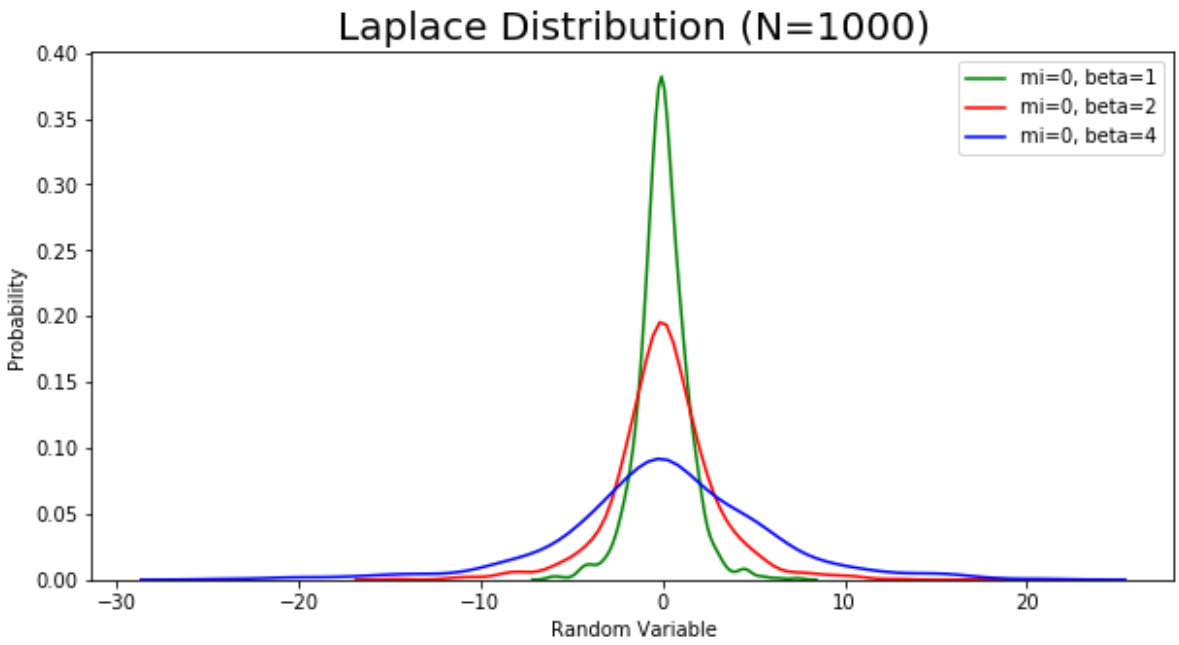
In [9]:
```
plt.figure(figsize=(10, 5))

n, p = 20, 0.25
binomial = np.random.binomial(n, p, 1000)
sns.distplot(binomial, color='green', label='n=20, p=0.25', hist=False)

n, p = 20, 0.5
binomial = np.random.binomial(n, p, 1000)
sns.distplot(binomial, color='red', label='n=20, p=0.5', hist=False)

n, p = 20, 0.75
binomial = np.random.binomial(n, p, 1000)
sns.distplot(binomial, color='blue', label='n=20, p=0.75', hist=False)

plt.title('Binomial Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://www.statisticshowto.datasciencecentral.com/negative
```



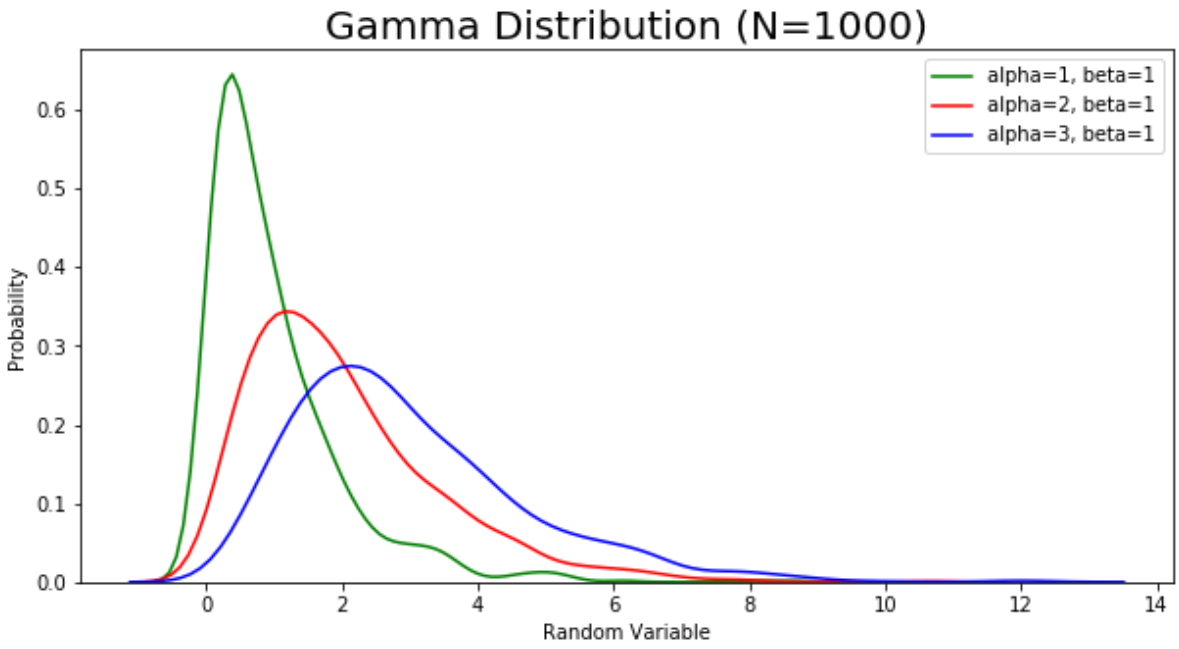
In [10]:
```
plt.figure(figsize=(10, 5))

alpha, beta = 2, 8
beta = np.random.beta(alpha, beta, 1000)
sns.distplot(beta, color='green', label='alpha=2, beta=8', hist=False)

alpha, beta = 5, 5
beta = np.random.beta(alpha, beta, 1000)
sns.distplot(beta, color='red', label='alpha=5, beta=5', hist=False)

alpha, beta = 8, 2
beta = np.random.beta(alpha, beta, 1000)
sns.distplot(beta, color='blue', label='alpha=8, beta=2', hist=False)

plt.title('Beta Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Probability')
plt.ylabel('Density')
plt.legend(loc='best')
plt.show()
# Reference: https://www.vosesoftware.com/riskwiki/Betadistribution.php
```



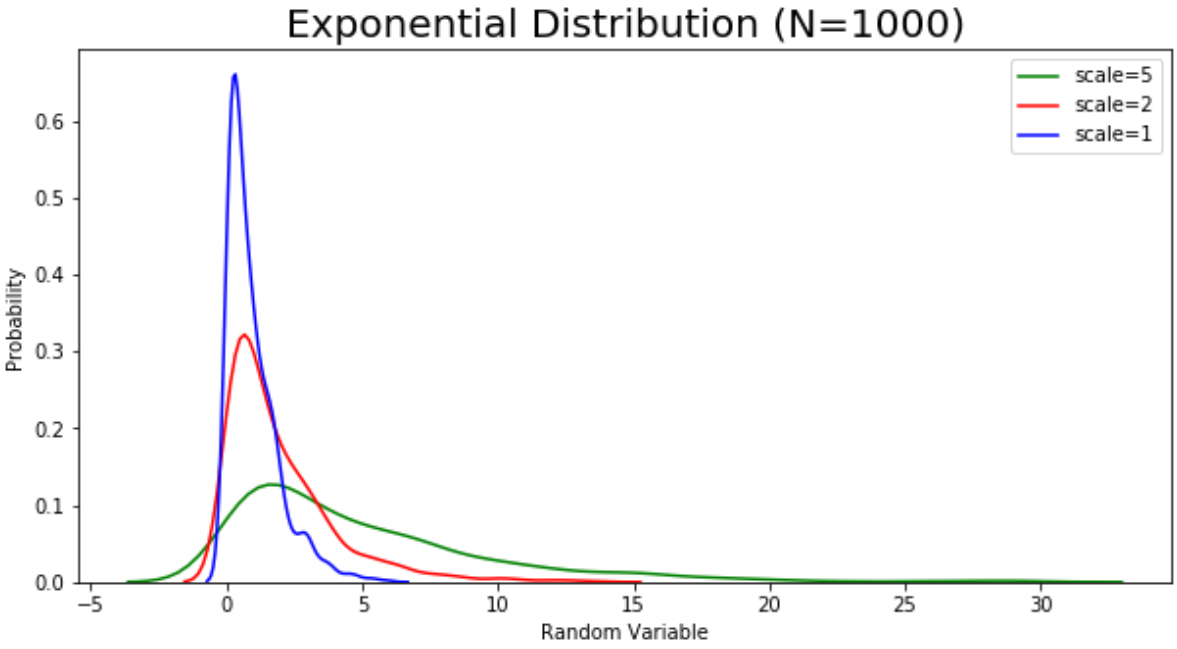
In [11]:
```
plt.figure(figsize=(10, 5))

loc, scale = 0, 1
laplace = np.random.laplace(loc, scale, 1000)
sns.distplot(laplace, color='green', label='mi=0, beta=1', hist=False)

loc, scale = 0, 2
laplace = np.random.laplace(loc, scale, 1000)
sns.distplot(laplace, color='red', label='mi=0, beta=2', hist=False)

loc, scale = 0, 4
laplace = np.random.laplace(loc, scale, 1000)
sns.distplot(laplace, color='blue', label='mi=0, beta=4', hist=False)

plt.title('Laplace Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://www.statisticshowto.datasciencecentral.com/laplace
# distribution-double-exponential/
```



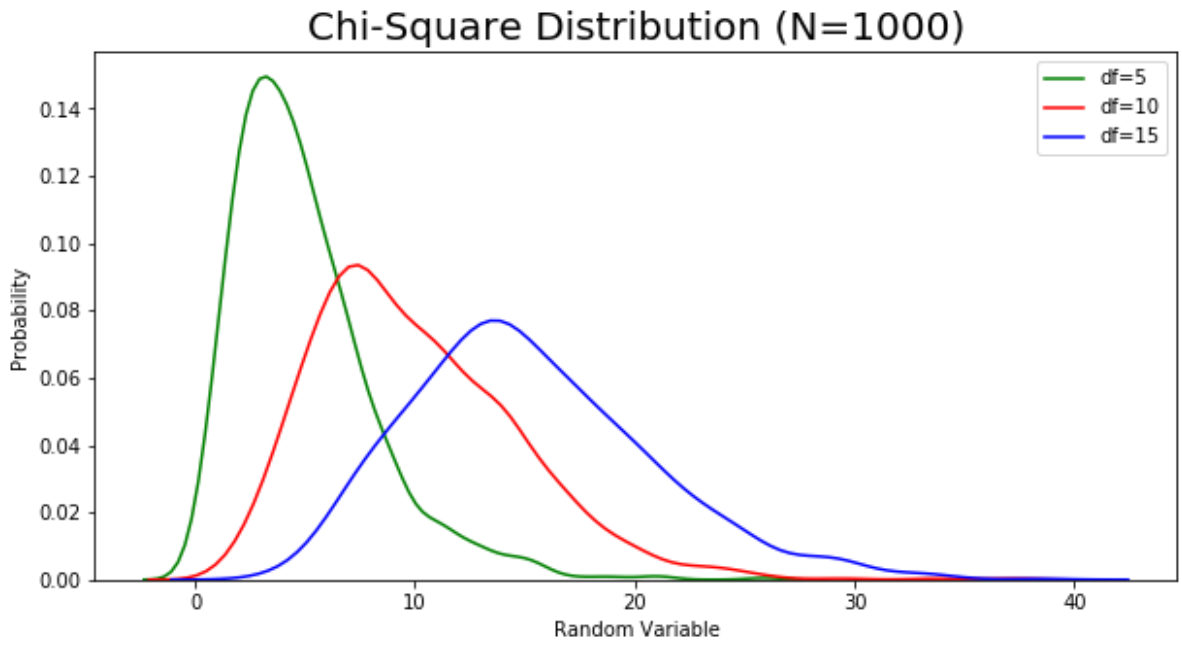
In [12]:
```
plt.figure(figsize=(10, 5))

alpha, beta = 1, 1
gamma = np.random.gamma(alpha, beta, 1000)
sns.distplot(gamma, color='green', label='alpha=1, beta=1', hist=False)

alpha, beta = 2, 1
gamma = np.random.gamma(alpha, beta, 1000)
sns.distplot(gamma, color='red', label='alpha=2, beta=1', hist=False)

alpha, beta = 3, 1
gamma = np.random.gamma(alpha, beta, 1000)
sns.distplot(gamma, color='blue', label='alpha=3, beta=1', hist=False)

plt.title('Gamma Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://en.wikipedia.org/wiki/Inverse-gamma_distribution
```



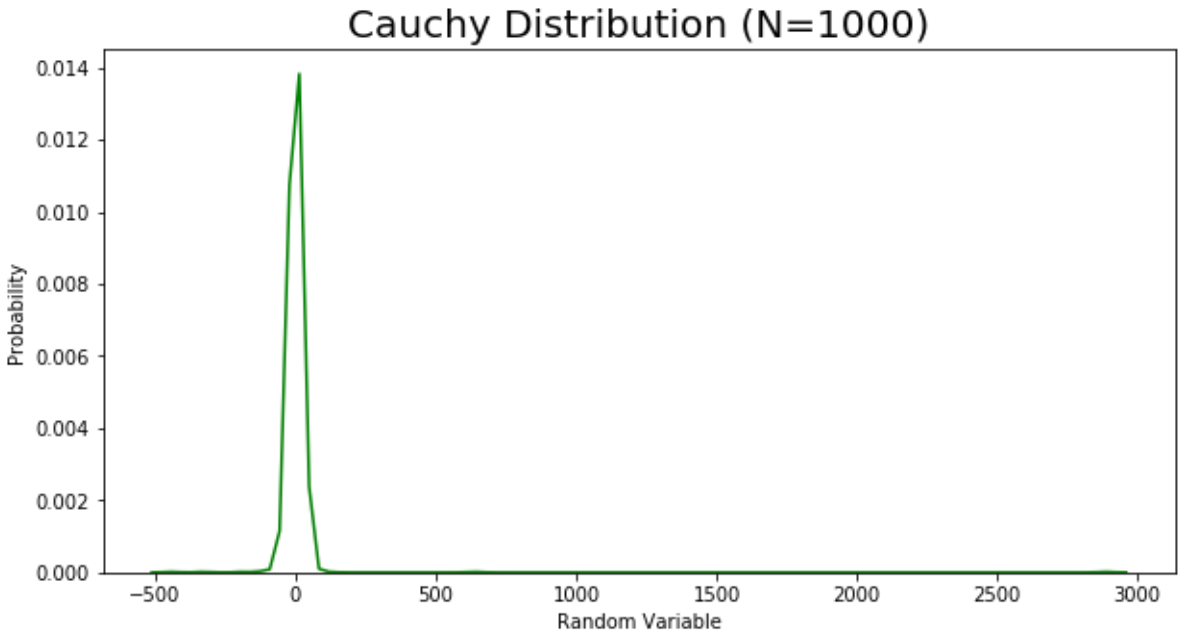
In [13]:
```
plt.figure(figsize=(10, 5))

scale = 5
exponential = np.random.exponential(scale, 1000)
sns.distplot(exponential, color='green', label='scale=5', hist=False)

scale = 2
exponential = np.random.exponential(scale, 1000)
sns.distplot(exponential, color='red', label='scale=2', hist=False)

scale = 1
exponential = np.random.exponential(scale, 1000)
sns.distplot(exponential, color='blue', label='scale=1', hist=False)

plt.title('Exponential Distribution (N=1000)', fontdict={'fontsize': 20}
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://www.vosesoftware.com/riskwiki/Exponentialdistributi
```



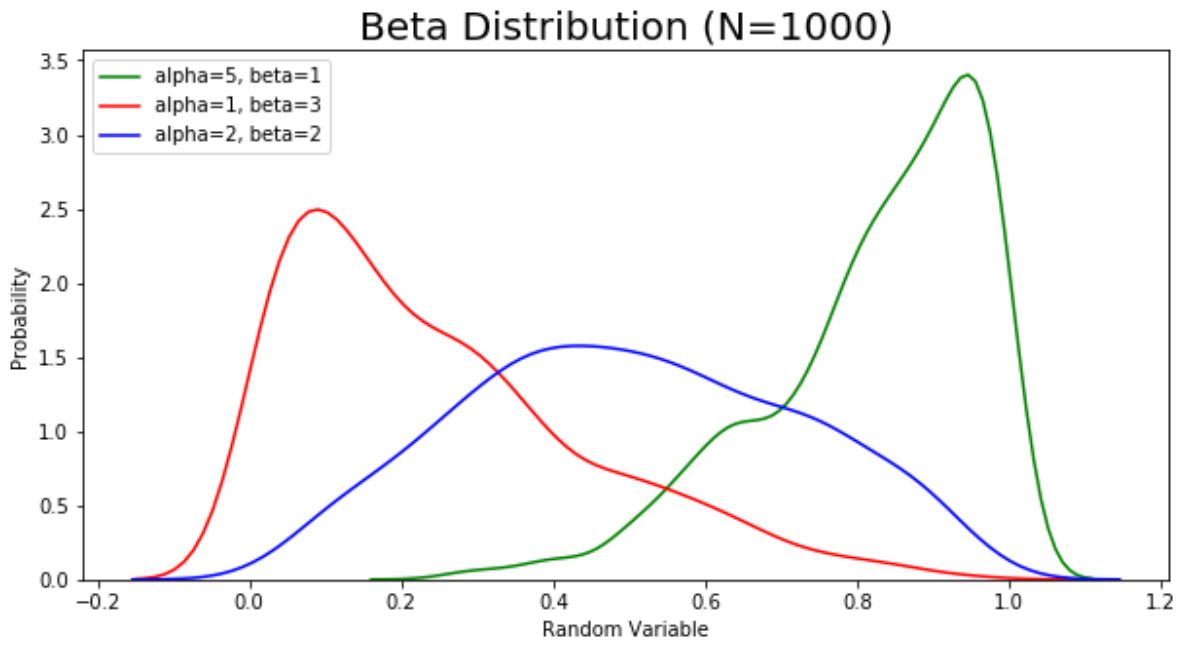
In [14]:
```
plt.figure(figsize=(10, 5))

df = 5
quiquadrado = np.random.chisquare(df, 1000)
sns.distplot(quiquadrado, color='green', label='df=5', hist=False)

df = 10
quiquadrado = np.random.chisquare(df, 1000)
sns.distplot(quiquadrado, color='red', label='df=10', hist=False)

df = 15
quiquadrado = np.random.chisquare(df, 1000)
sns.distplot(quiquadrado, color='blue', label='df=15', hist=False)

plt.title('Chi-Square Distribution (N=1000)', fontdict={'fontsize': 20}
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://www.globalspec.com/reference/69594/203279/10-8-
# the-chi-square-distribution
```



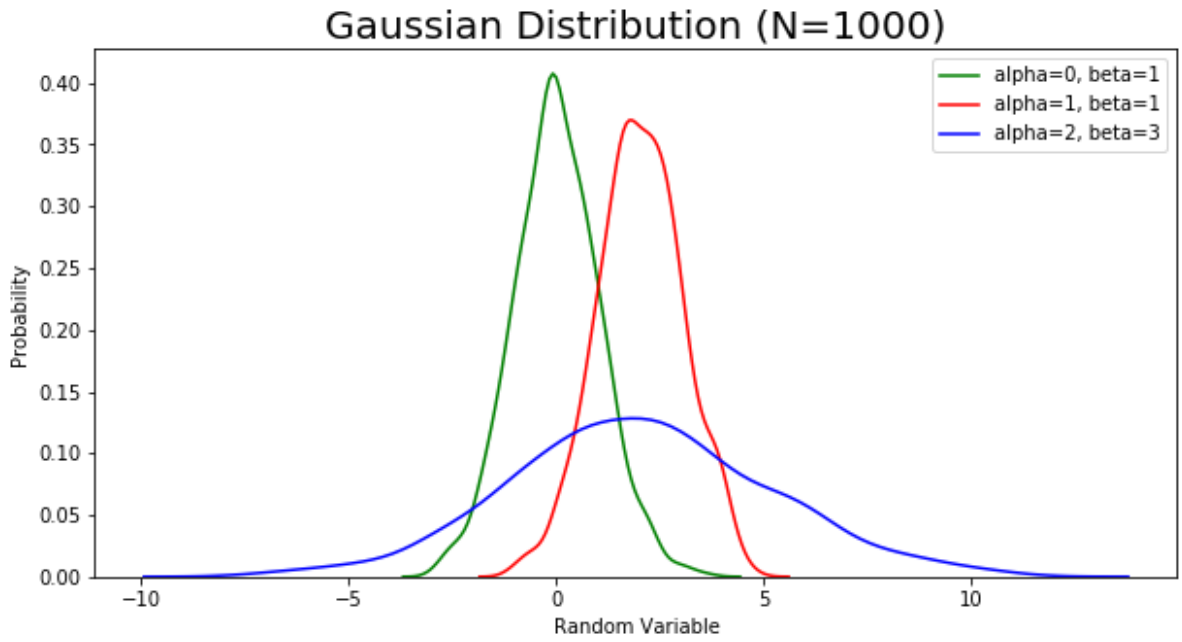
In [15]:
```
plt.figure(figsize=(10, 5))

df = 1000
cauchy = np.random.standard_cauchy(df)
sns.distplot(cauchy, color='green', hist=False)


plt.title('Cauchy Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.show()
# Reference: https://www.globalspec.com/reference/69594/203279/10-
# 8-the-chi-square-distribution
```



In [16]:
```
plt.figure(figsize=(10, 5))

alpha, beta = 5, 1
beta = np.random.beta(alpha, beta, 1000)
sns.distplot(beta, color='green', label='alpha=5, beta=1', hist=False)

alpha, beta = 1, 3
beta = np.random.beta(alpha, beta, 1000)
sns.distplot(beta, color='red', label='alpha=1, beta=3', hist=False)

alpha, beta = 2, 2
beta = np.random.beta(alpha, beta, 1000)
sns.distplot(beta, color='blue', label='alpha=2, beta=2', hist=False)


plt.title('Beta Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://en.wikipedia.org/wiki/Beta_distribution
```



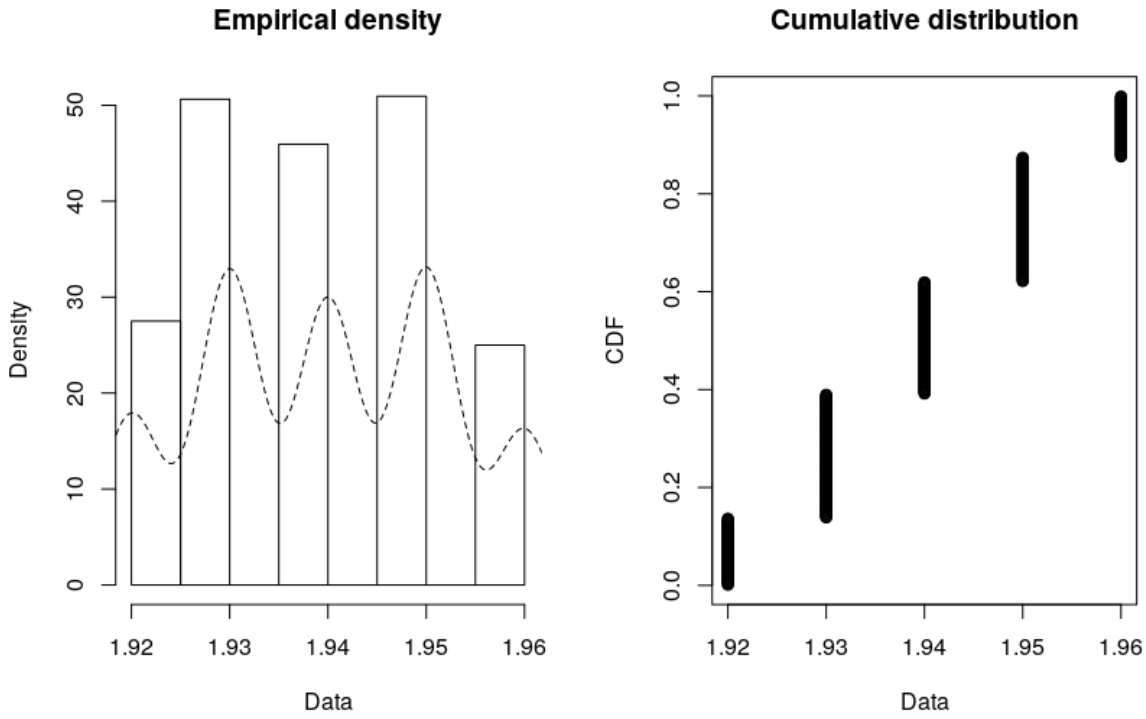
In [17]:
```
plt.figure(figsize=(10, 5))

alpha, beta = 0, 1
gaussian = np.random.normal(alpha, beta, 1000)
sns.distplot(gaussian, color='green', label='alpha=0, beta=1', hist=Fals

alpha, beta = 2, 1
gaussian = np.random.normal(alpha, beta, 1000)
sns.distplot(gaussian, color='red', label='alpha=1, beta=1', hist=False

alpha, beta = 2, 3
gaussian = np.random.normal(alpha, beta, 1000)
sns.distplot(gaussian, color='blue', label='alpha=2, beta=3', hist=Fals

plt.title('Gaussian Distribution (N=1000)', fontdict={'fontsize': 20})
plt.xlabel('Random Variable')
plt.ylabel('Probability')
plt.legend(loc='best')
plt.show()
# Reference: https://en.wikipedia.org/wiki/Q-Gaussian_distribution
```



2.2 - Considere o seguinte experimento: Lançamento de 3 dados simultâneos com registro de quantas vezes um determinado resultado pode ser obtido. Mostre que a distribuição limite é binomial e que com N tendendo a infinito ela converge para uma Gaussiana.

In [18]:
```
Image(filename='./plots/eliptical1.png')
```

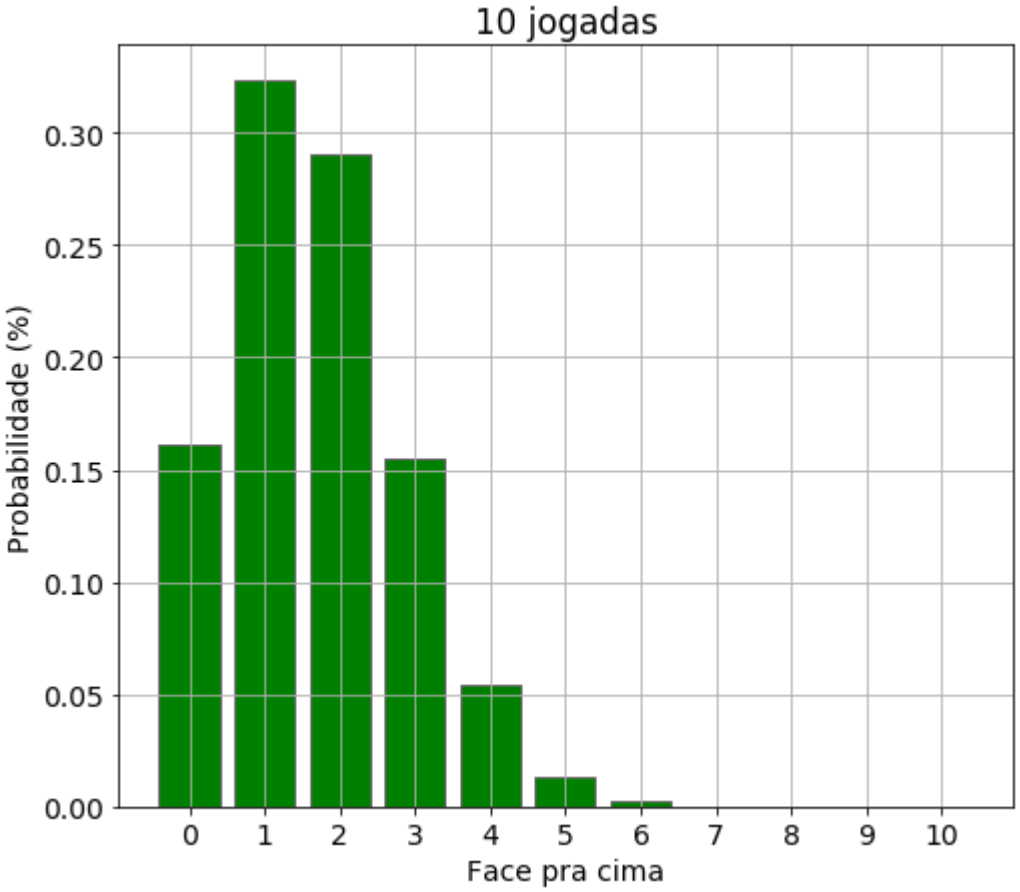
Out[18]:

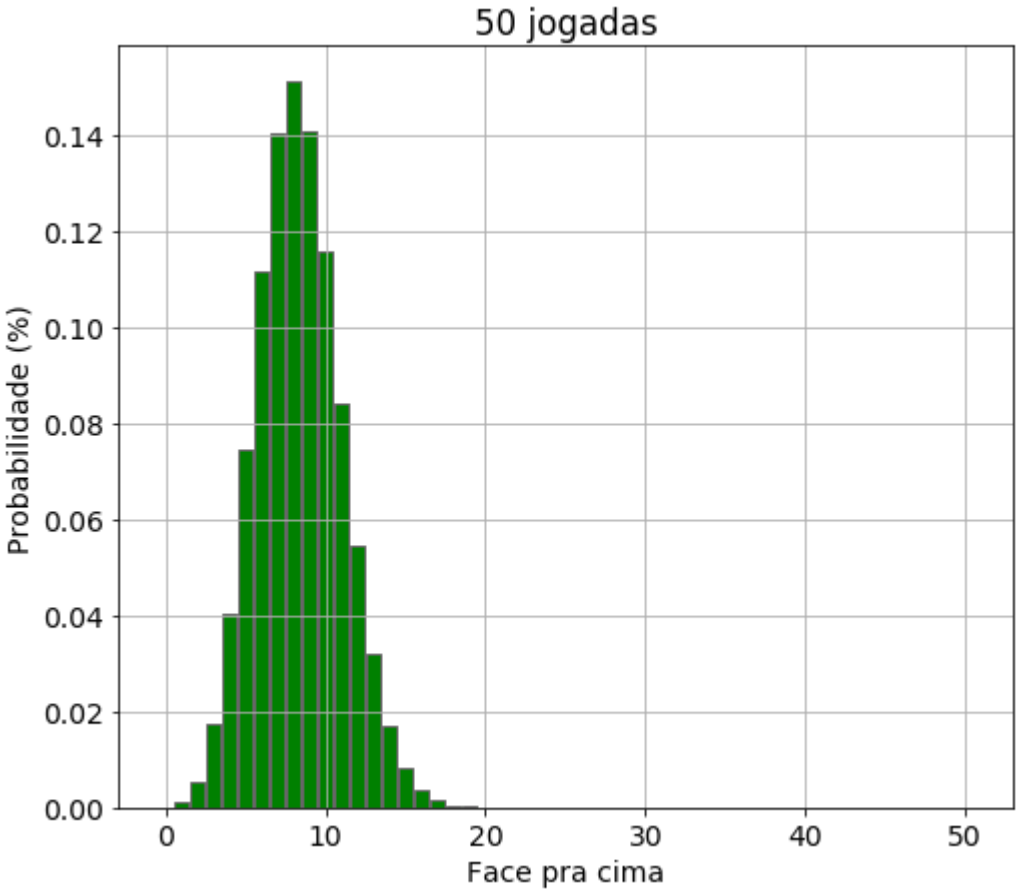


In [19]:

```
def fatorial(n):
    if(n<=1):
        return 1
    else:
        return (n * fatorial(n-1))


def geraProbabilidades(n, p):
    probabilidades = np.zeros(n+1)

    q = 1 - p
    for numP in range(0, n + 1):
        numQ = n - numP

        combinacao_n_numP = fatorial(n) / (fatorial(numP) * fatorial(n
        probabilidades[numP] = combinacao_n_numP * np.power(p, numP) *

    return probabilidades


def plot(probabilidades, n):
    bins = np.arange(0, n+1, 1)

    pylab.plt.subplots()
    pylab.clf()
    pylab.plt.bar(bins, probabilidades, color='green', ec='0.4', lw=1)

    if n <= 10:
        pylab.plt.xticks(np.arange(0, n+1, 1))
    elif n <= 20:
        pylab.plt.xticks(np.arange(0, n+1, 2))
    elif n <= 100:
        pylab.plt.xticks(np.arange(0, n+1, 10))
    else:
        pylab.plt.xticks(np.arange(0, n+1, 20))

    pylab.rc('figure', figsize=(8,7))
    pylab.rcParams.update({'font.size': 14})
    pylab.grid(True)
    title = '{} jogadas'.format(n)
    pylab.plt.title(title)
    pylab.plt.xlabel('Face pra cima')
    pylab.plt.ylabel('Probabilidade (%)')



lances = [2, 10, 50]
p = 1/6
for n in lances:
    probabilidades = geraProbabilidades(n, p)
    plot(probabilidades, n)
pylab.plt.show()
```


2 jogadas
Probabilidade (%)
0.0
0.2
0.4
0.6
0
1
2
Face pra cima


10 jogadas
Probabilidade (%)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
1
2
3
4
5
6
7
8
9
10
Face pra cima



3 - Probabilidade Condicional

3.1 - Considere 3 regiões do céu contendo aproximadamente o mesmo número (N) de galáxias, cujas Distribuições morfológicas dada por um modelo seja aquela apresentada na Tabela abaixo.

Região/Tipo	S1	S2	S3
Irregulares	10%	25%	15%
Espirais	60%	40%	55%
Elípticas	30%	35%	30%

Em cada região será realizado um survey (S1, S2 e S3) considerando para cada uma um telescópio. Supondo que os telescópios são equivalentes e que as observações serão aleatórias calcule as seguintes probabilidades:

i) A primeira galáxia observada ser espiral ou elíptica.

$$P_{Irregular} = \frac{10\% + 25\% + 15\%}{100\%} = 16.66\%$$

$$P_{Espiral} \cup P_{Elíptica} = 100\% - 16.66 = 83.34\%$$

ii) Se a primeira galáxia observada for irregular, qual a probabilidade dela pertencer à região do survey S1.

$$Irregular_S1 = \frac{10}{3} = 0.03$$

$$Ans = \frac{0.03}{0.16} = 0.20 = 20\%$$

3.2 - Considere o exercício anterior e crie um "bootstrap" para gerar 10 amostras contendo 200 galáxias cada uma. Considere os valores de morfologia caracterizados pelo parâmetro g1 (da técnica gradient pattern analysis) dado na tabela abaixo.

Irregulares: 1.97-1.99
Espirais: 1.96-1.98
Elípticas: 1.92-1.96

Aplique o Teorema de Bayes para encontrar a máxima verossimilhança considerando os modelos Gaussiano.

In [20]:
```
def generate(up, down, galaxies):
    ans = []
    for i in range(galaxies):
        rand = round(np.random.uniform(up,down),2)
        ans.append(rand)
    return ans

# irregular = []
# espiral = []
# elipitica = []

# for x in range(10):
#     temp = generate(1.97, 1.99, 33)
#     for y in temp:
#         irregular.append(y)

#     temp = generate(1.96, 1.98, 103)
#     for y in temp:
#         espiral.append(y)

#     temp = generate(1.92, 1.96, 64)
#     for y in temp:
#         elipitica.append(y)

# np.savetxt('./series/irregular.txt', irregular, newline='\n')
# np.savetxt('./series/espiral.txt', espiral, newline='\n')
# np.savetxt('./series/elipitica.txt', elipitica, newline='\n')
```

~~3.3 - Supondo que SNIa ... ...estime tempo de telescopio....~~

In [ ]:
```
```

4 - Teorema do Valor Extremo

Reconsidere o exercício anterior e aplique o Teorema de Bayes para encontrar a máxima verossimilhança considerando o modelo GEV.

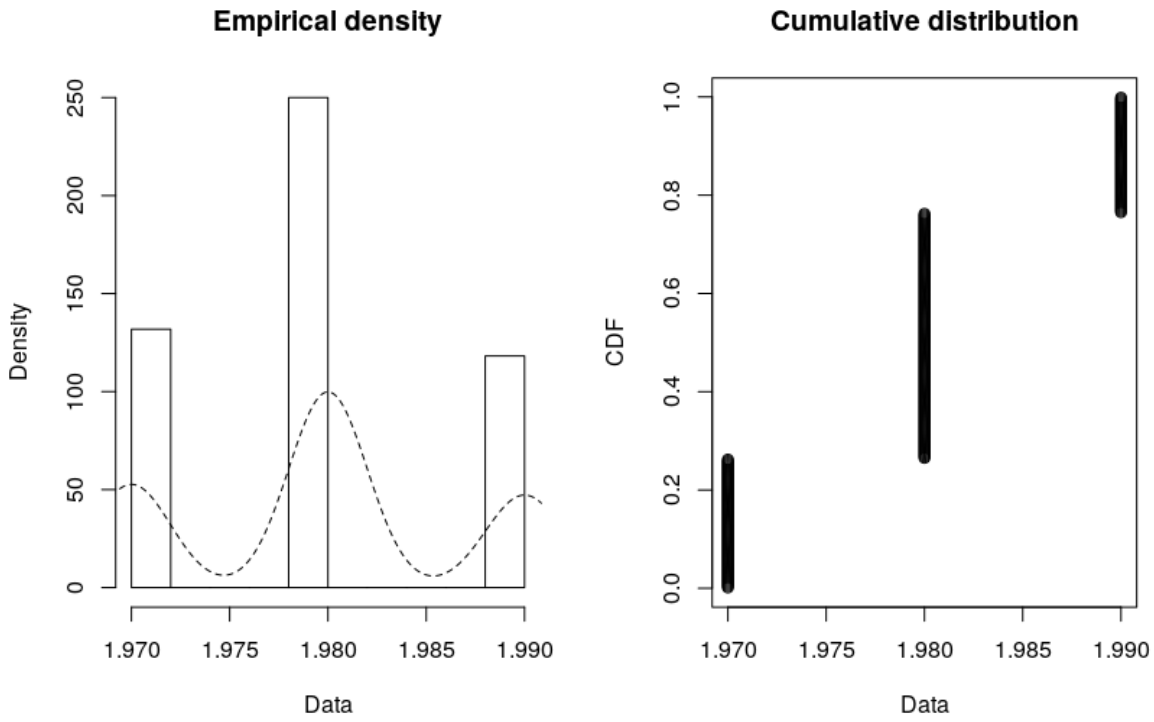
In [ ]:

5 - Classificação de Cullen-Frey

Classifique a população de amostras geradas no exercício 3.2. no espaço de Cullen-Frey, calcule os desvios e compare os desempenhos dos modelos.

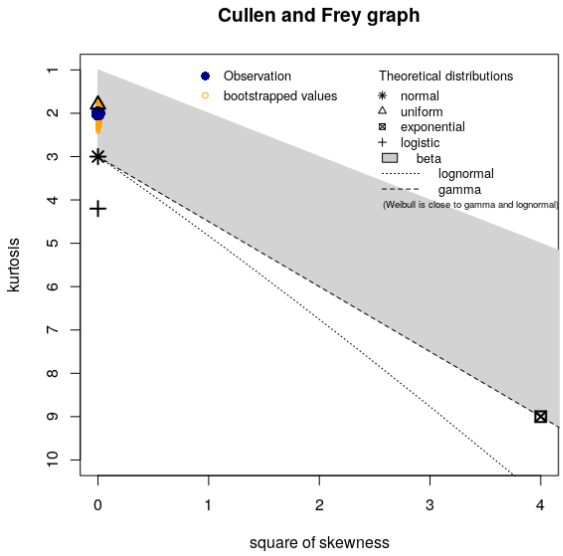
In [21]:
```
Image(filename='./plots/irregular1.png')
```

Out[21]:



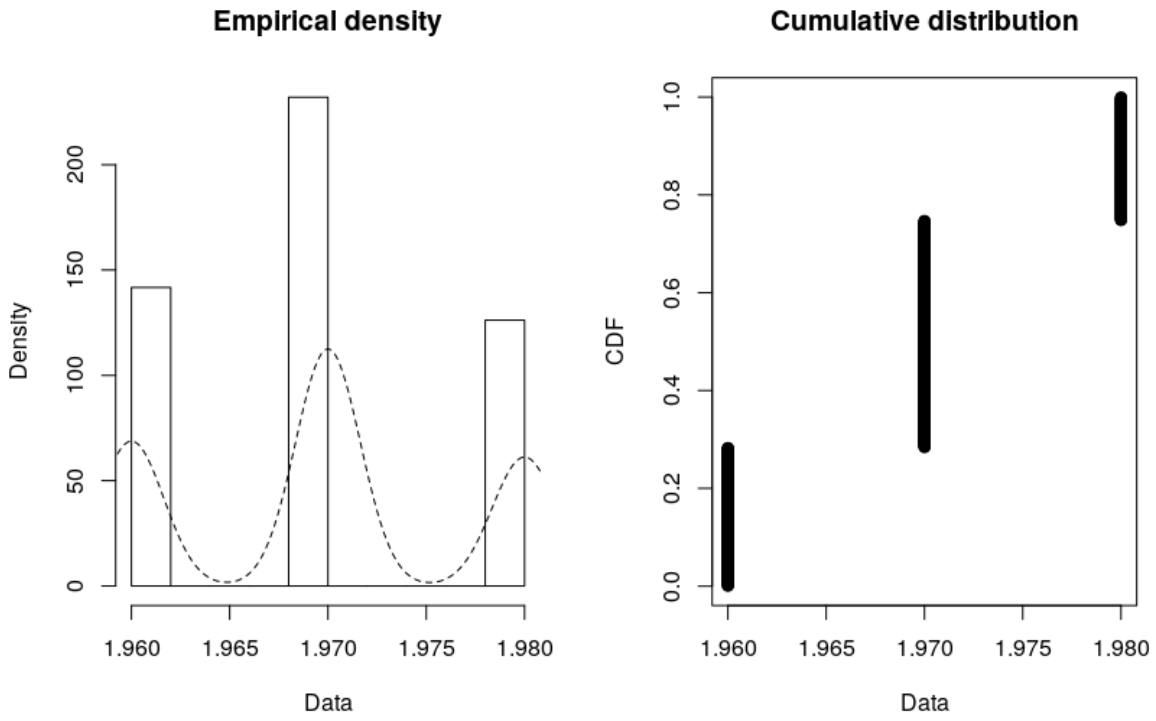
In [22]:
```
Image(filename='./plots/irregular2.png')
```

Out[22]:



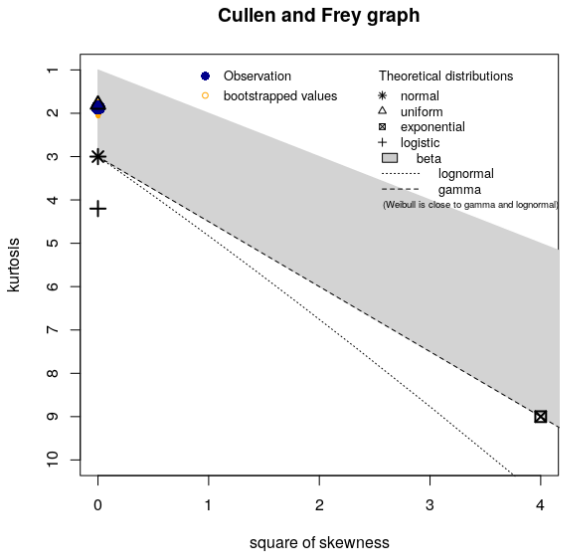
In [23]:
```
Image(filename='./plots/espiral1.png')
```

Out[23]:



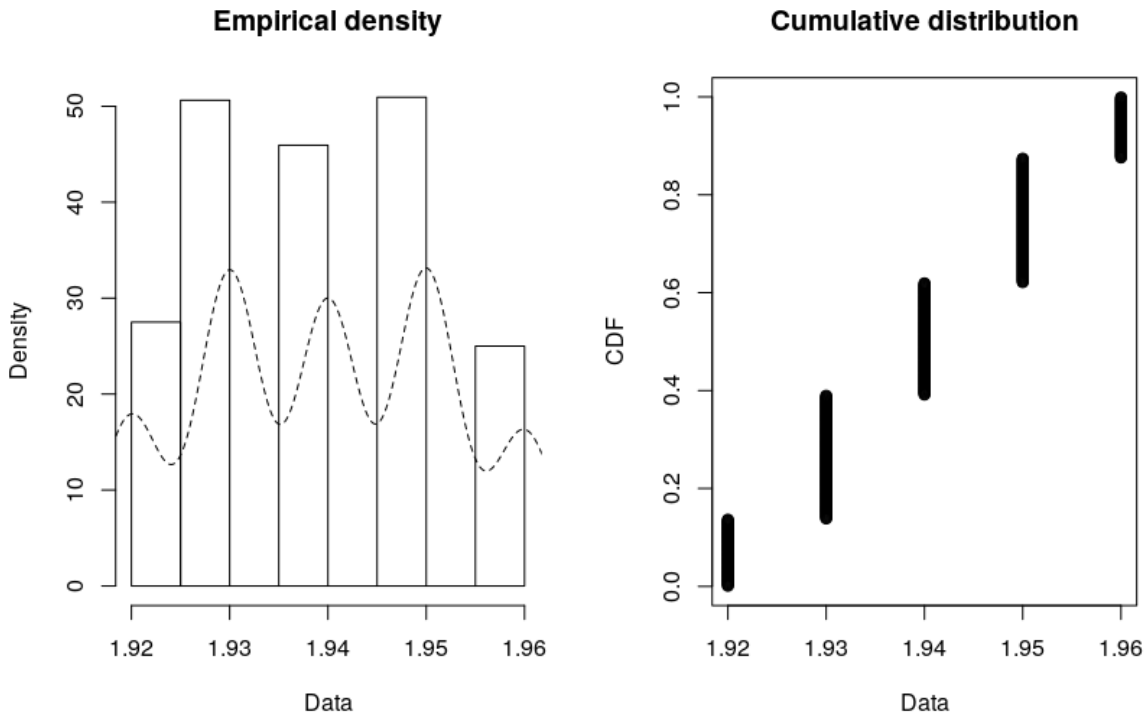
In [24]:
```
Image(filename='./plots/espiral2.png')
```

Out[24]:



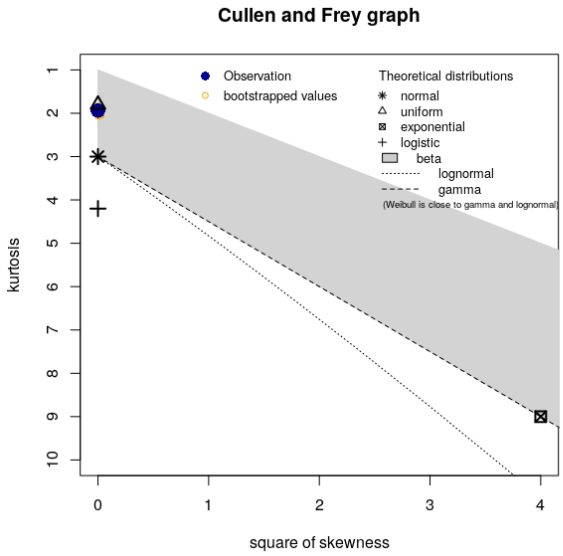
In [25]:
```
Image(filename='./plots/eliptica1.png')
```

Out[25]:



In [26]:
```
Image(filename='./plots/eliptica2.png')
```

Out[26]:

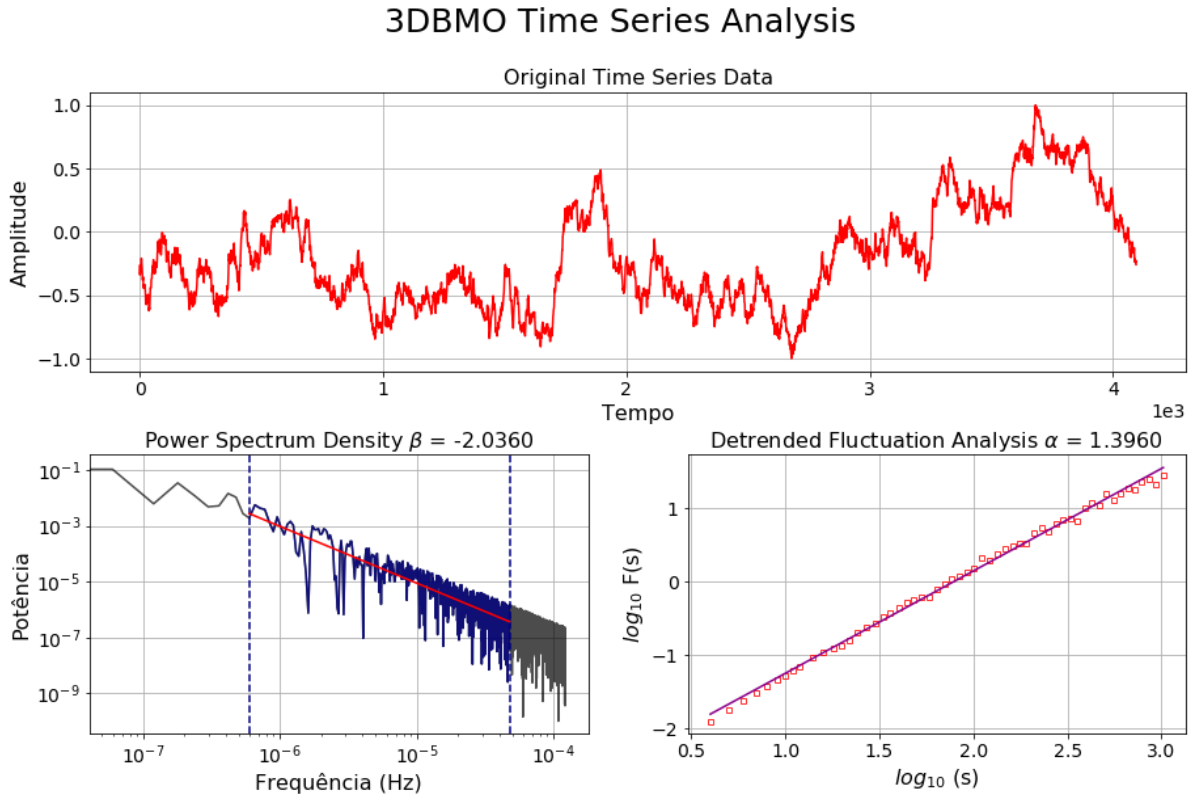


6 - PSD & DFA: S3, S7, S8.

6.1 - Considere as séries temporais listadas acima e obtenha os valores dos respectivos índices espectrais: β (via PSD) e α (via DFA).

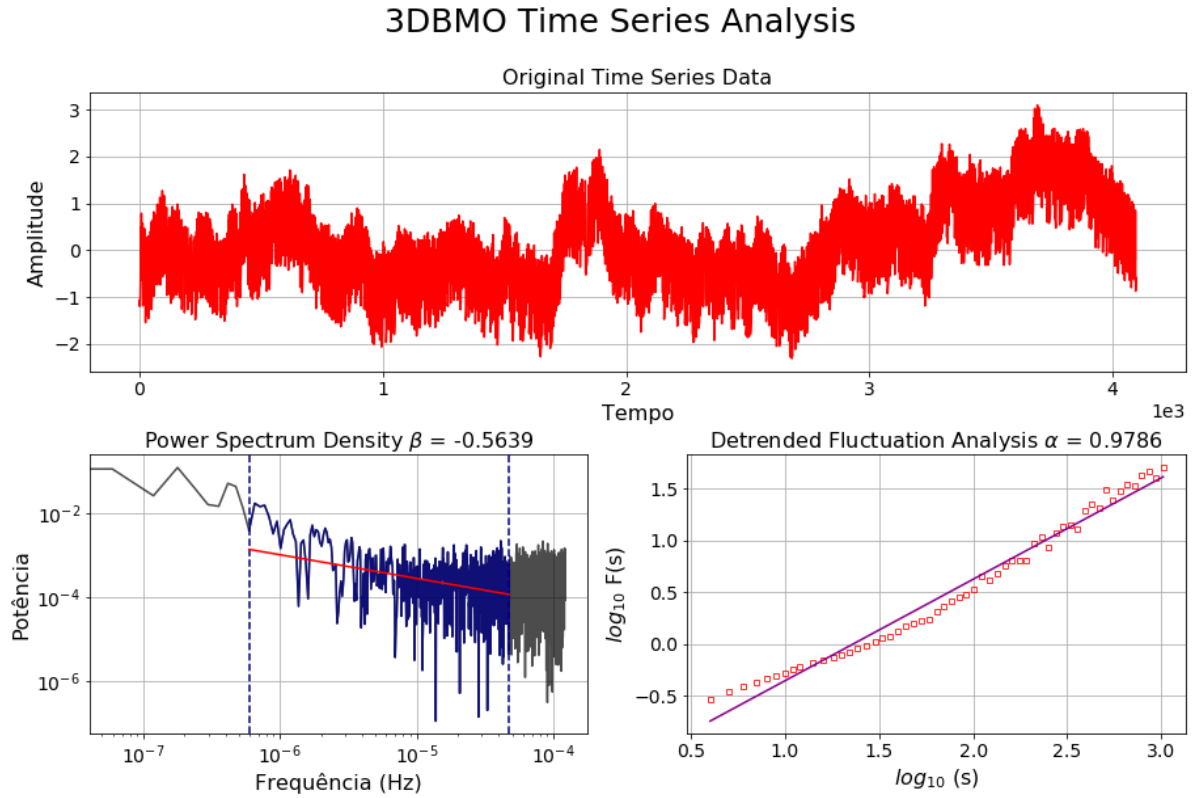
In [27]:
```
power_spectrum_density(S3)
```

```
Data Analysis for 3DBMO simulations...

Original time series data (4096 points):

First 10 points: [-0.26854399 -0.33984295 -0.32844949 -0.28747909 -0.2
803743  -0.26374457
 -0.23963736 -0.20956527 -0.22033328 -0.25287976]


1. Plotting time series data...
2. Plotting Power Spectrum Density...
3. Plotting Detrended Fluctuation Analysis...
```



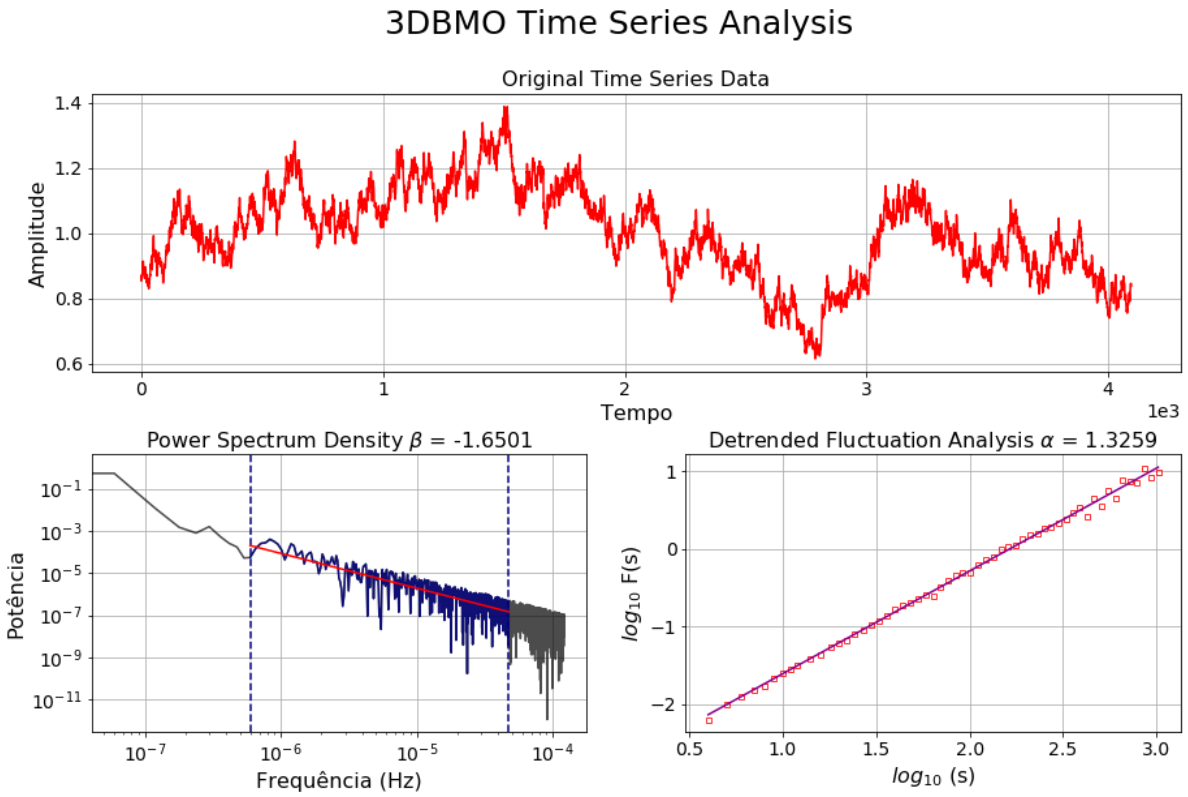
In [28]:
```
power_spectrum_density(S7)
```

```
Data Analysis for 3DBMO simulations...

Original time series data (4096 points):

First 10 points: [-1.0730866  -1.20134317 -1.15755832 -0.99284688 -0.6
559834   0.28417934
  0.49679976  0.20038954  0.78478553 -0.6364924 ]


1. Plotting time series data...
2. Plotting Power Spectrum Density...
3. Plotting Detrended Fluctuation Analysis...
```



In [29]:
```
power_spectrum_density(S8)
```

```
Data Analysis for 3DBMO simulations...

Original time series data (4096 points):

First 10 points: [0.85355973 0.87135986 0.85941593 0.8615829  0.867493
66 0.89562414
 0.91330332 0.90401942 0.91317688 0.89925285]


1. Plotting time series data...
2. Plotting Power Spectrum Density...
3. Plotting Detrended Fluctuation Analysis...
```



6.2 - Confira se o PSD está bem ajustado a partir da formula WKP: $\beta = 2\ \alpha - 1$.

In [30]:
```
2*1.3960-1 #Betal pra S3
```

Out[30]: 1.7919999999999998

In [31]:
```
2*0.9786-1 #Betal pra S7
```

Out[31]: 0.9572

In [32]:
```
2*1.3259-1 #Betal pra S8
```

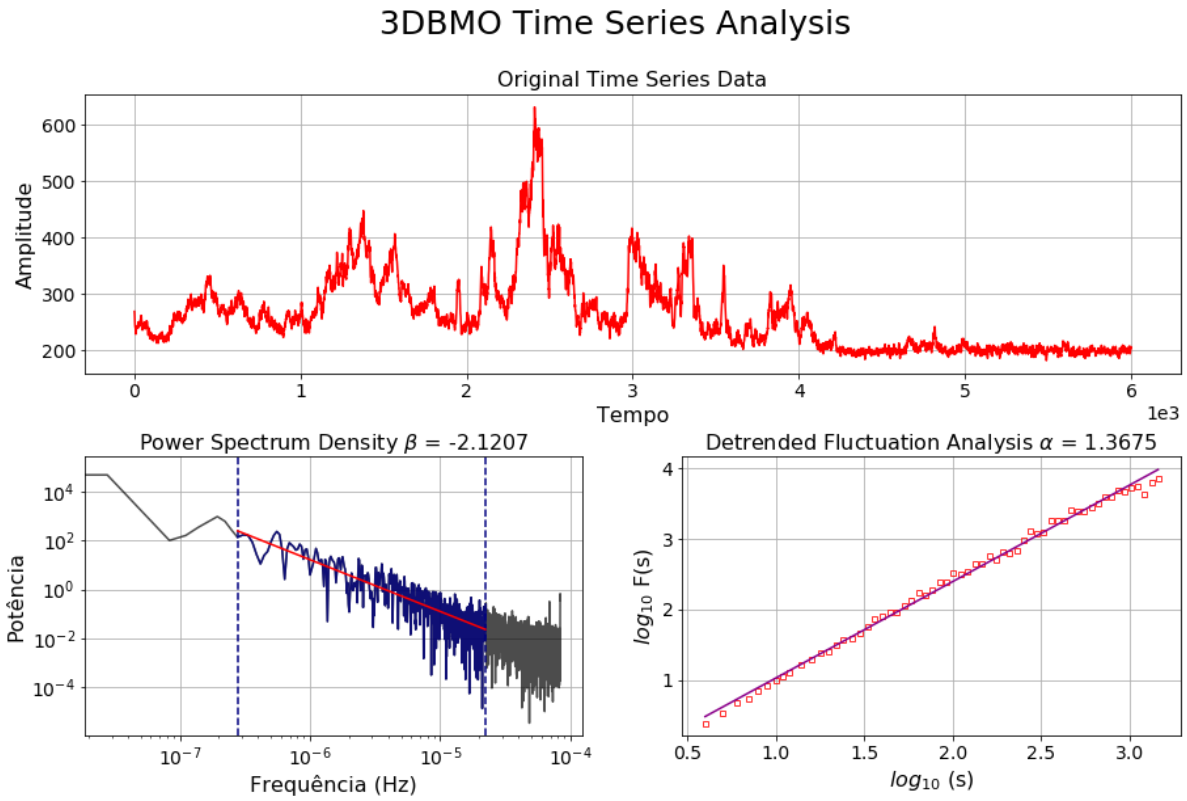
Out[32]: 1.6518000000000002

6.3 - Repita 6.1. para (a) ST-Sol3GHz, (b) ST-surftemp504 e (c) uma ST de sua escolha (alternativa).

In [33]:
```
sol = np.loadtxt('./series/sol.txt')
```

In [34]:
```
power_spectrum_density(sol)
```

```
Data Analysis for 3DBMO simulations...

Original time series data (6000 points):

First 10 points: [269. 264. 252. 246. 242. 242. 238. 233. 229. 233.]


1. Plotting time series data...
2. Plotting Power Spectrum Density...
3. Plotting Detrended Fluctuation Analysis...
```



6.4 - Com base nos valores de β (obtido via PSD e calculado via formula WKP).

In [ ]:
```
```

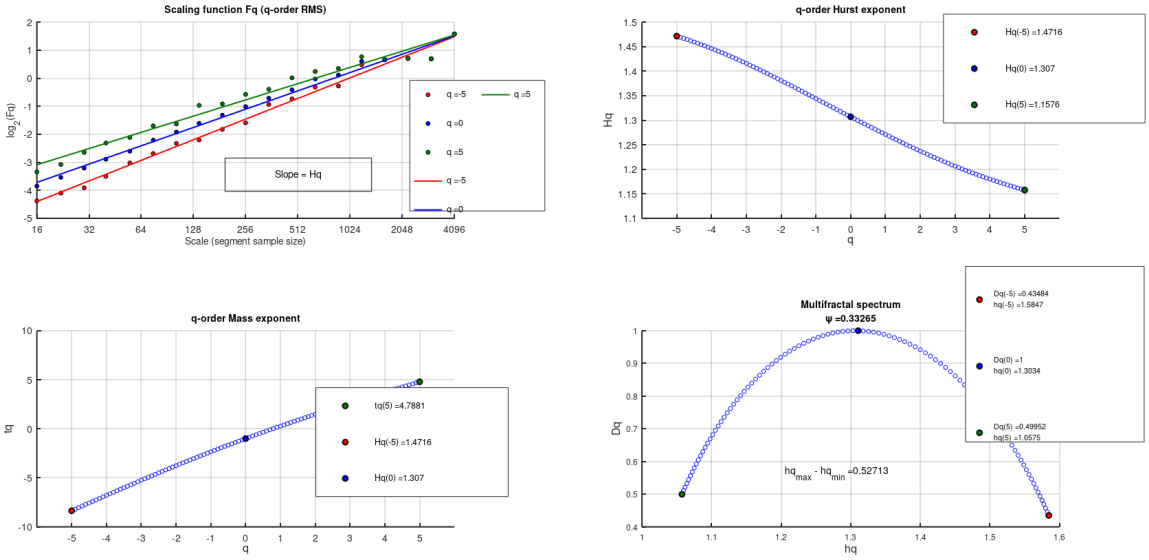
7 - Singularity Multifractal Spectra (SMS), também conhecido como MDFA.

7.1. Considere o programa `mfdfa.py` Aprimore o programa para o mesmo calcule o índice $\psi = \delta\alpha/\alpha_{max}$.

7.2 - Obtenha o espectro de singularidade para todos os sinais do exercício 6.

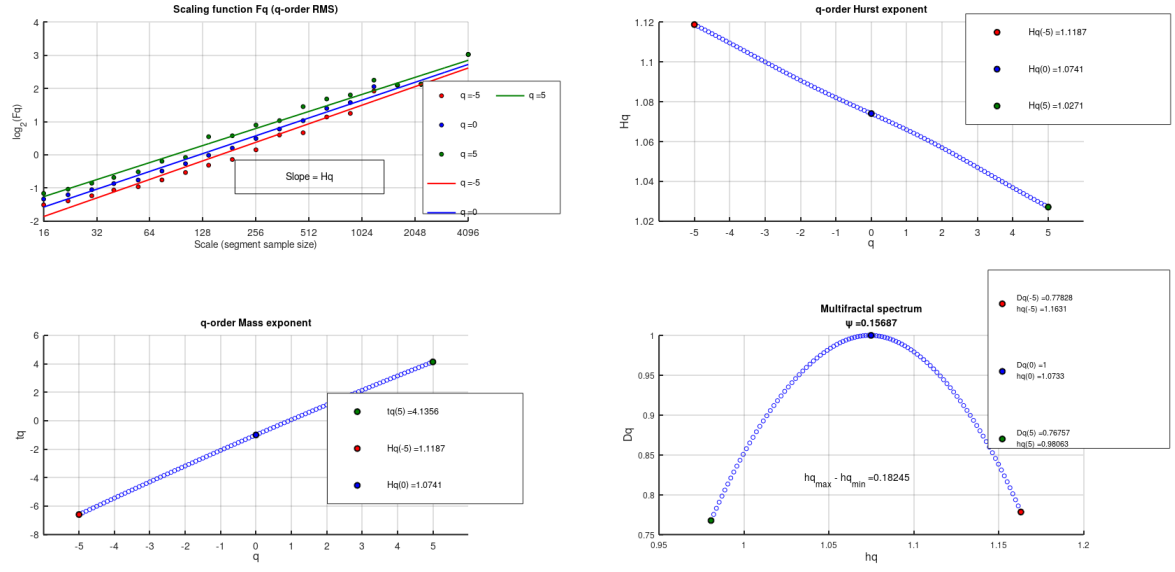
In [36]:
```
Image(filename='./plots/mfdfas3.png')
```

Out[36]:



In [37]:
```
Image(filename='./plots/mfdfas7.png')
```

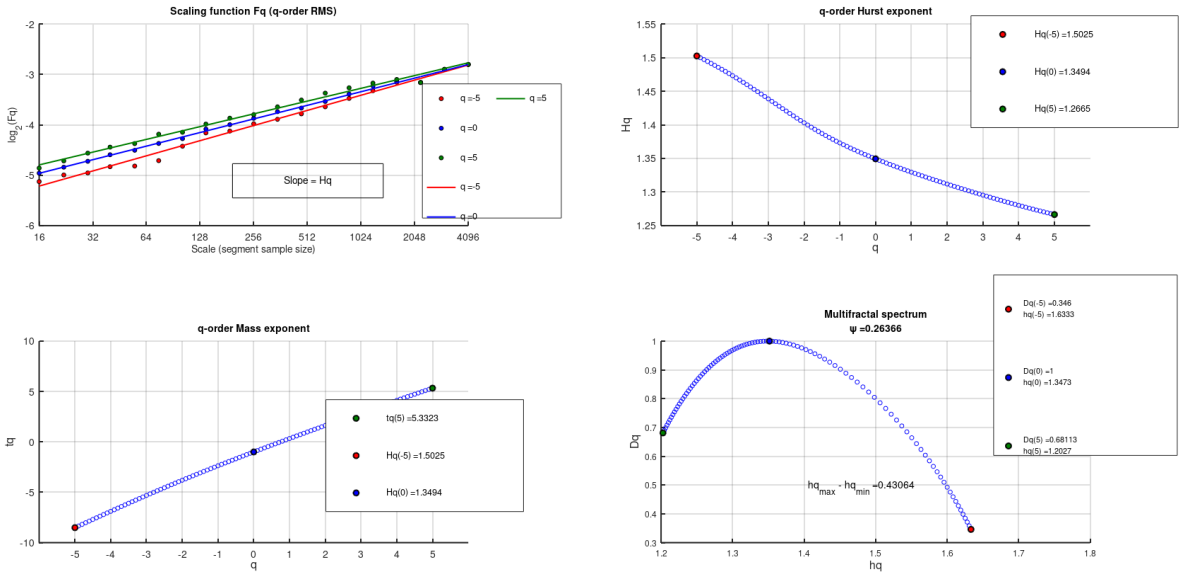
Out[37]:



7.3 - Com base nos valores obtidos em 7.2., discuta os possíveis processos subjacentes para cada ST.

In [38]:
```
Image(filename='./plots/mfdfas8.png')
```

Out[38]:



8 - Considere o software SPECTRUM e discuta, de forma comparativa e sucinta, outros métodos para análise espectral de sinais não abordados no curso.

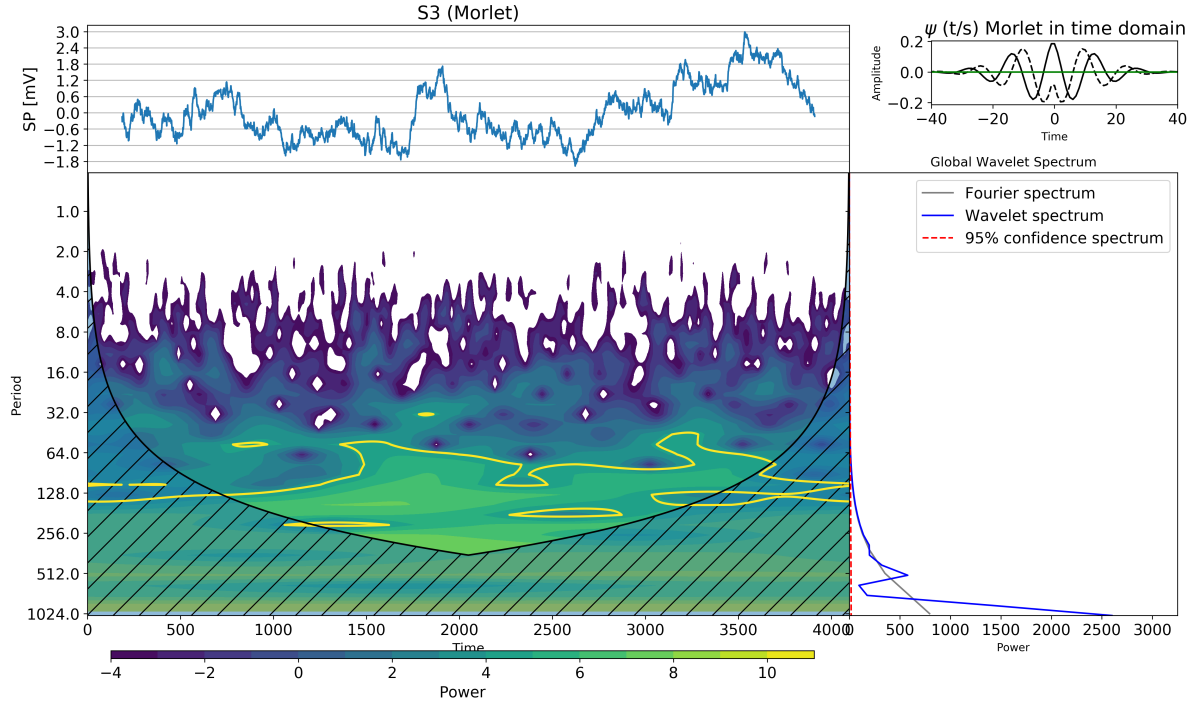
O SPECTRUM é um módulo em python que contém ferramentas para estimar o PSD de séries temporaris, baseando-se na Transformada de Fourier. s métodos baseados em Fourier são baseados em correlogramas, periodogramas e estimadores de Welch. unções de janelamento padrão (Hann, Hamming, Blackman) e outras mais exóticas estão disponíveis como exemplo a DPSS e Taylor. Os métodos paramétricos são baseados em Yule-Walker, BURG, MA e ARMA, métodos de covariância e métodos de covariância modificada.

9 - Global Wavelet Spectrum

9.1 - Utilize o Waipy para obter o GWS (Morlet) de todos as ST do exercício 6.1.

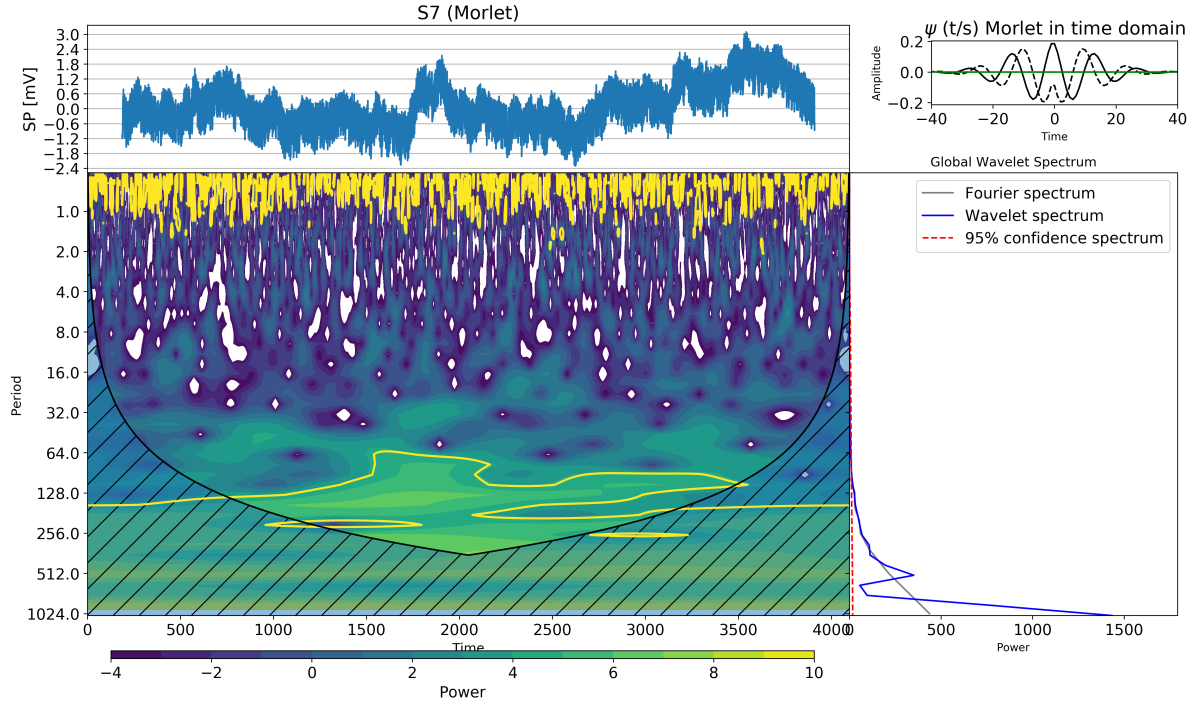
In [39]:
```
data = S3
time = np.linspace(0, len(data), len(data))
data_norm = waipy.normalize(data)
result = waipy.cwt(data_norm, dt=0.25, pad=1, dj=0.25, s0=2*0.25, j1=7/(
waipy.wavelet_plot(var='S3 (Morlet)', time=time, data=data_norm, dtmin=(
```

```
2048 2048
```



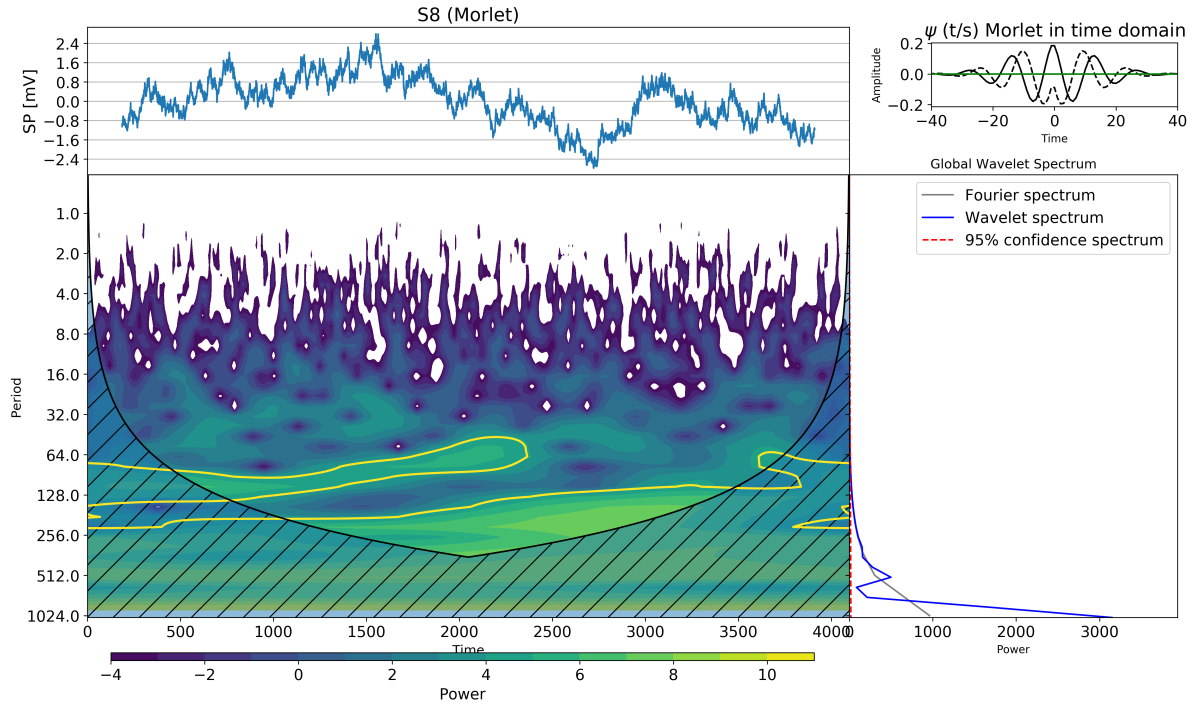
In [40]:
```
data = S7
time = np.linspace(0, len(data), len(data))
data_norm = waipy.normalize(data)
result = waipy.cwt(data_norm, dt=0.25, pad=1, dj=0.25, s0=2*0.25, j1=7/(
waipy.wavelet_plot(var='S7 (Morlet)', time=time, data=data_norm, dtmin=(
```

```
2048 2048
```



In [41]:
```
data = S8
time = np.linspace(0, len(data), len(data))
data_norm = waipy.normalize(data)
result = waipy.cwt(data_norm, dt=0.25, pad=1, dj=0.25, s0=2*0.25, j1=7/(
waipy.wavelet_plot(var='S8 (Morlet)', time=time, data=data_norm, dtmin=(
```

```
2048 2048
```

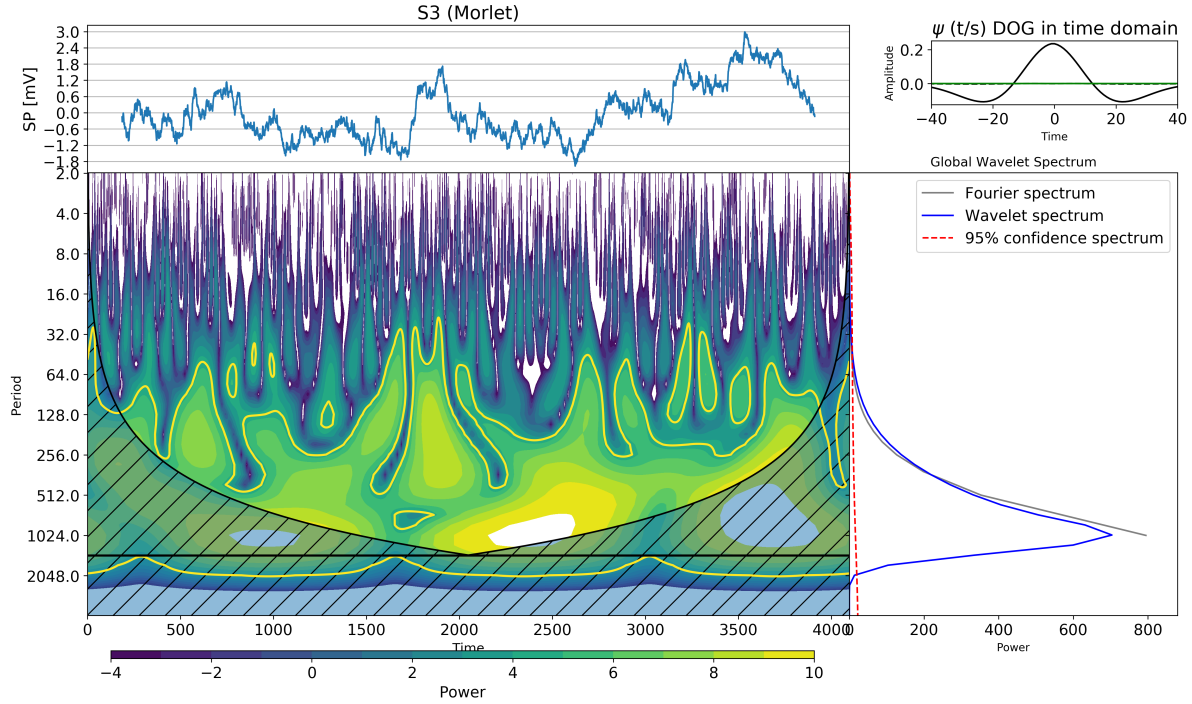


9.2 Repita 9.1. utilizando uma Db8.

(Qualquer outra transormação, Paul, DOG)Não é possível aplicar transforamções discretas utilizando a biblioteca no `waipy`, pois ela só faz transformações continuas. Por isso irei aplicar a mãe `DOG`, já que nao possui a `daubechies 8`.

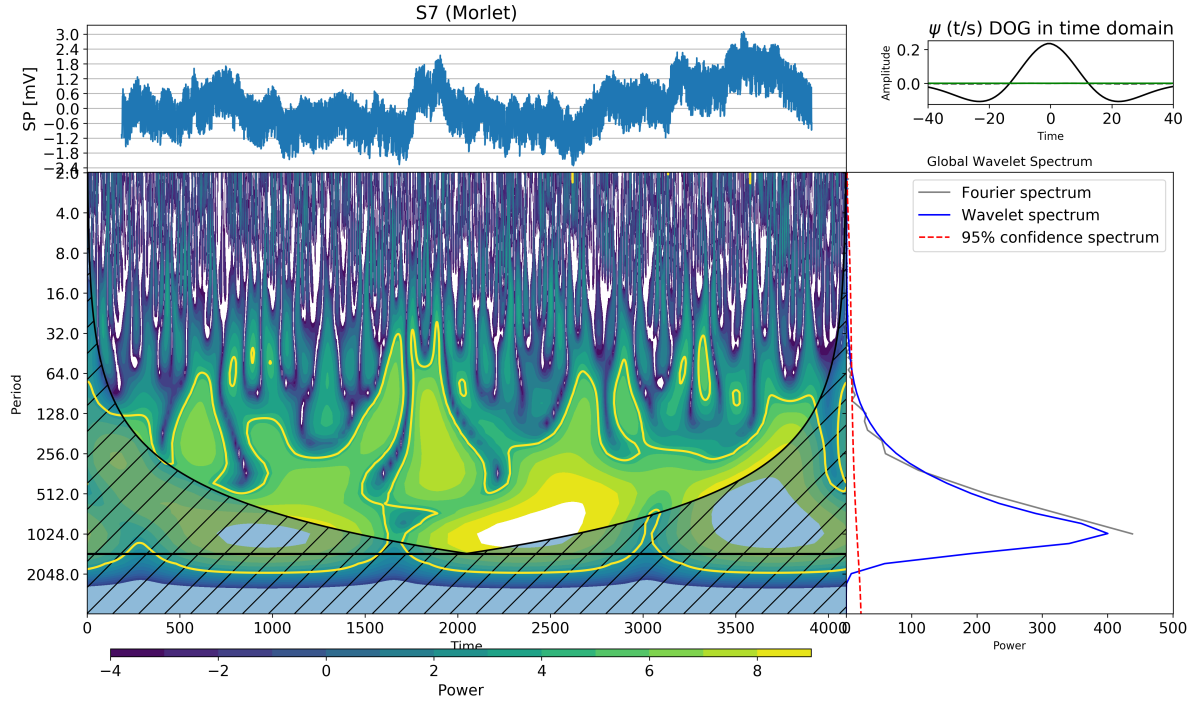
In [42]:
```
data = S3
time = np.linspace(0, len(data), len(data))
data_norm = waipy.normalize(data)
result = waipy.cwt(data_norm, dt=0.25, pad=1, dj=0.25, s0=2*0.25, j1=7/(
waipy.wavelet_plot(var='S3 (Morlet)', time=time, data=data_norm, dtmin=(
```

```
2048 2048
```



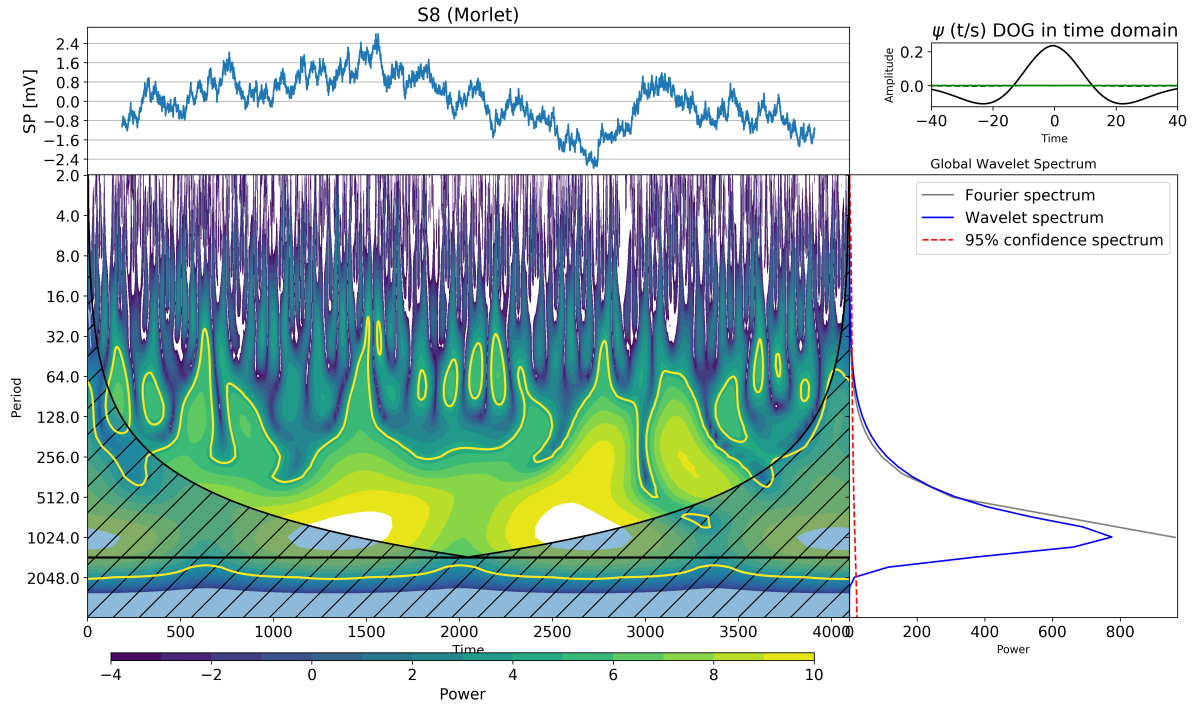
In [43]:
```
data = S7
time = np.linspace(0, len(data), len(data))
data_norm = waipy.normalize(data)
result = waipy.cwt(data_norm, dt=0.25, pad=1, dj=0.25, s0=2*0.25, j1=7/(
waipy.wavelet_plot(var='S7 (Morlet)', time=time, data=data_norm, dtmin=(
```

```
2048 2048
```



In [44]:
```
data = S8
time = np.linspace(0, len(data), len(data))
data_norm = waipy.normalize(data)
result = waipy.cwt(data_norm, dt=0.25, pad=1, dj=0.25, s0=2*0.25, j1=7/(
waipy.wavelet_plot(var='S8 (Morlet)', time=time, data=data_norm, dtmin=(
```

```
2048 2048
```



10 - Self-Organized Criticality (SOC)

i) Calcule a Taxa Local de Flutuação [Yi] para cada valor da ST

ii) Calcule P[γi] = counts(ni) / N

iii) Plot logP[γi] x log ni (e ajuste uma lei de potencia).

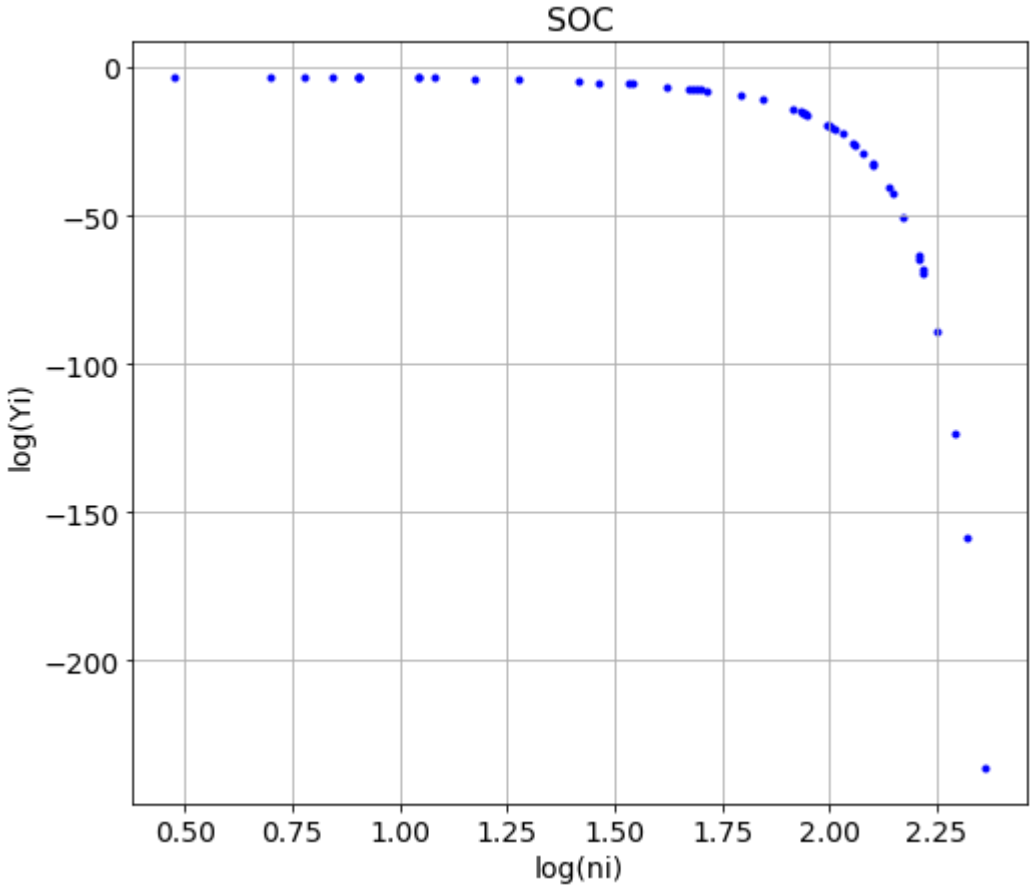
10.1 - Implemente um algoritmo em Python para caracterização de SOC a partir de uma ST.

10.2 - Aplique o SOC.py para todas as ST do exercício 6.1.

In [45]:
```
def SOC(data, n_bins=50):
    n = len(data)
    mean = np.mean(data)
    var = np.var(data)
    std = np.std(data)
    #print("mean: ", mean, " var: ", var)
    """ Computa a Taxa Local de Flutuação para cada valor da ST """
    Gamma = []
    for i in range(0,n):
        #Gamma.append((data[i] - mean)/var)
        Gamma.append((data[i] - mean)/std)

    """ Computa P[Psi_i] """
    # Retorna o número de elementos em cada bin, bem como os delimitares
    counts, bins = np.histogram(Gamma, n_bins)
    Prob_Gamma = []
    for i in range(0, n_bins):
        Prob_Gamma.append(counts[i]/n)
    #plt.plot(Gamma)
    return Prob_Gamma, counts
```

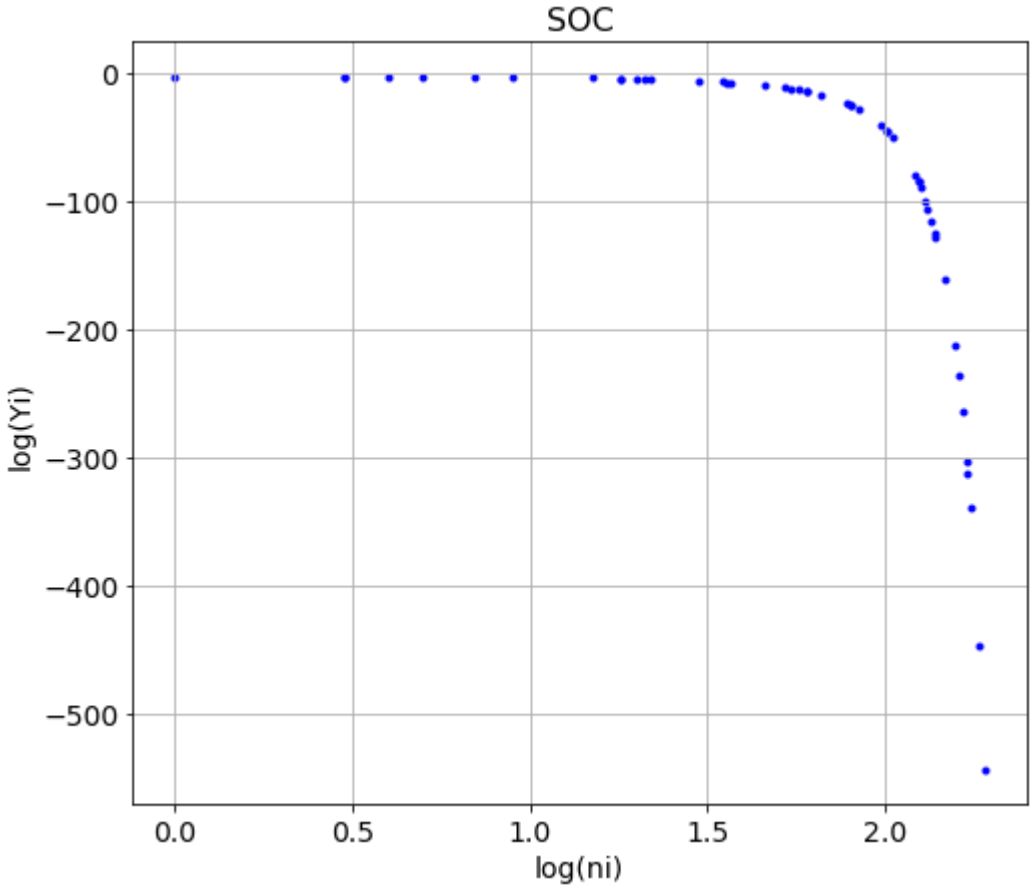
In [46]:

```
data = np.genfromtxt('./series/S3.txt')

Prob_Gamma, counts = SOC(data)

x = np.linspace(1, len(counts), len(counts))

log_Prob = np.log10(Prob_Gamma)
log_counts = np.log10(counts)

p = np.array(Prob_Gamma)
p = p[np.nonzero(p)]
c = counts[np.nonzero(counts)]
log_p = np.log10(p)
log_c = np.log10(c)

a = (log_p[np.argmax(c)] - log_p[np.argmin(c)]) / (np.max(c) - np.min(c
b = log_Prob[0]
y = b * np.power(10, (a*counts))

""" Plotagem """
plt.clf()
plt.scatter(np.log10(counts), y, marker=".", color="blue")

plt.title('SOC', fontsize = 16)
plt.xlabel('log(ni)')
plt.ylabel('log(Yi)')
plt.grid()

# plt.savefig('s7plot_novo.pdf')

plt.show()
```


SOC
0
−50
−100
−150
−200
log(Yi)
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
log(ni)

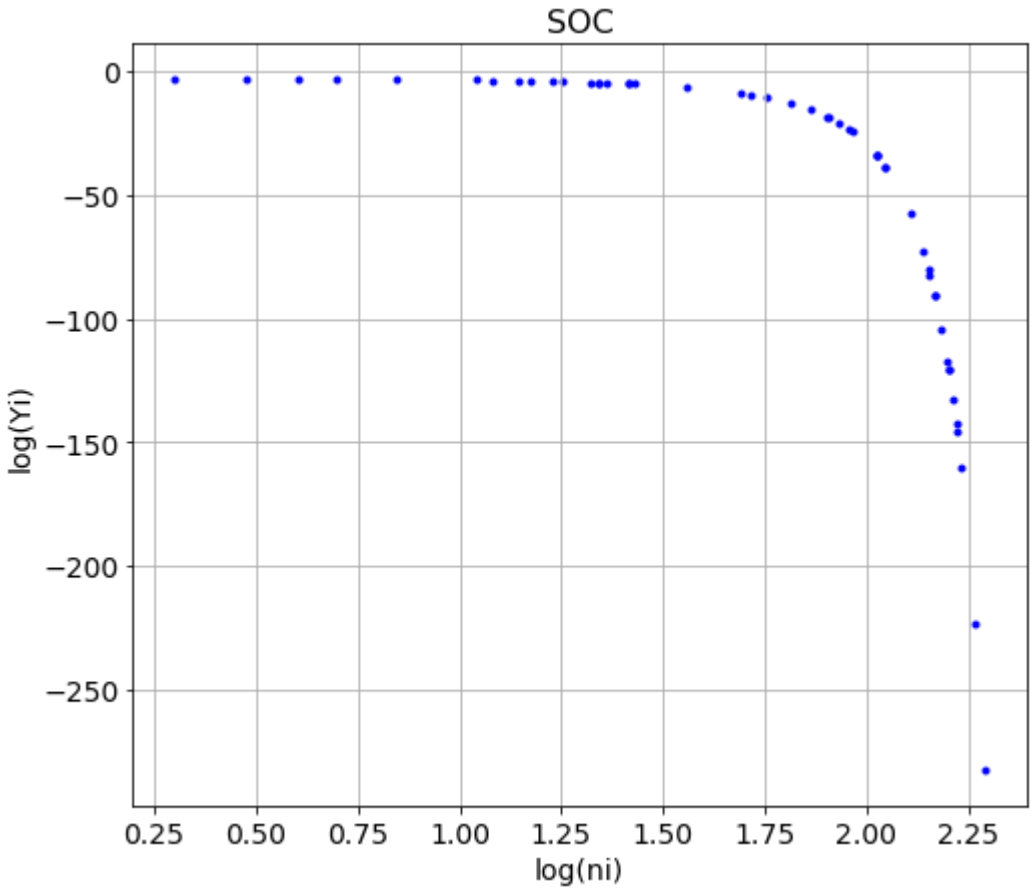
In [47]:

```
data = np.genfromtxt('./series/S7.txt')

Prob_Gamma, counts = SOC(data)

x = np.linspace(1, len(counts), len(counts))

log_Prob = np.log10(Prob_Gamma)
log_counts = np.log10(counts)

p = np.array(Prob_Gamma)
p = p[np.nonzero(p)]
c = counts[np.nonzero(counts)]
log_p = np.log10(p)
log_c = np.log10(c)

a = (log_p[np.argmax(c)] - log_p[np.argmin(c)]) / (np.max(c) - np.min(c
b = log_Prob[0]
y = b * np.power(10, (a*counts))

""" Plotagem """
plt.clf()
plt.scatter(np.log10(counts), y, marker=".", color="blue")

plt.title('SOC', fontsize = 16)
plt.xlabel('log(ni)')
plt.ylabel('log(Yi)')
plt.grid()

# plt.savefig('s7plot_novo.pdf')

plt.show()
```


SOC
0
−100
−200
−300
−400
−500
log(Yi)
0.0
0.5
1.0
1.5
2.0
log(ni)

In [48]:

```
data = np.genfromtxt('./series/S8.txt')

Prob_Gamma, counts = SOC(data)

x = np.linspace(1, len(counts), len(counts))

log_Prob = np.log10(Prob_Gamma)
log_counts = np.log10(counts)

p = np.array(Prob_Gamma)
p = p[np.nonzero(p)]
c = counts[np.nonzero(counts)]
log_p = np.log10(p)
log_c = np.log10(c)

a = (log_p[np.argmax(c)] - log_p[np.argmin(c)]) / (np.max(c) - np.min(c
b = log_Prob[0]
y = b * np.power(10, (a*counts))

""" Plotagem """
plt.clf()
plt.scatter(np.log10(counts), y, marker=".", color="blue")

plt.title('SOC', fontsize = 16)
plt.xlabel('log(ni)')
plt.ylabel('log(Yi)')
plt.grid()

# plt.savefig('s7plot_novo.pdf')

plt.show()
```


SOC
0
−50
−100
−150
−200
−250
log(Yi)
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
log(ni)