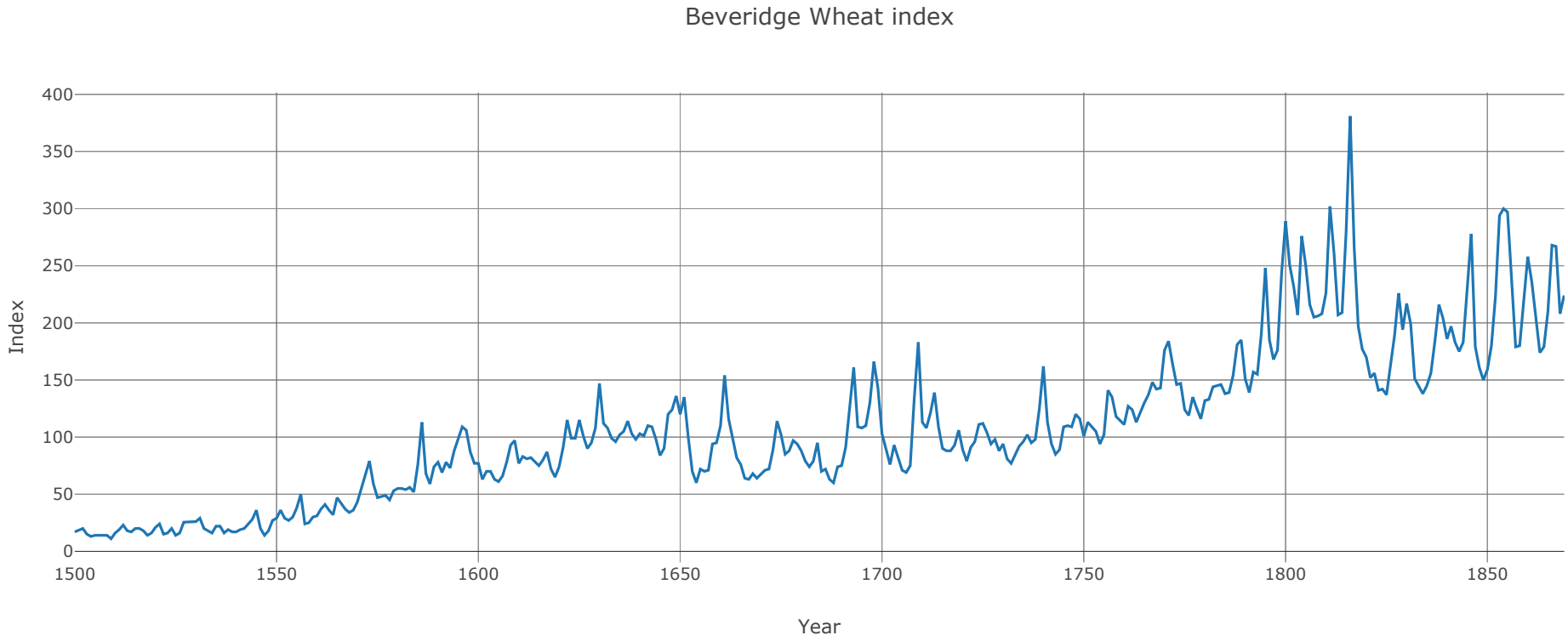
In [1]:

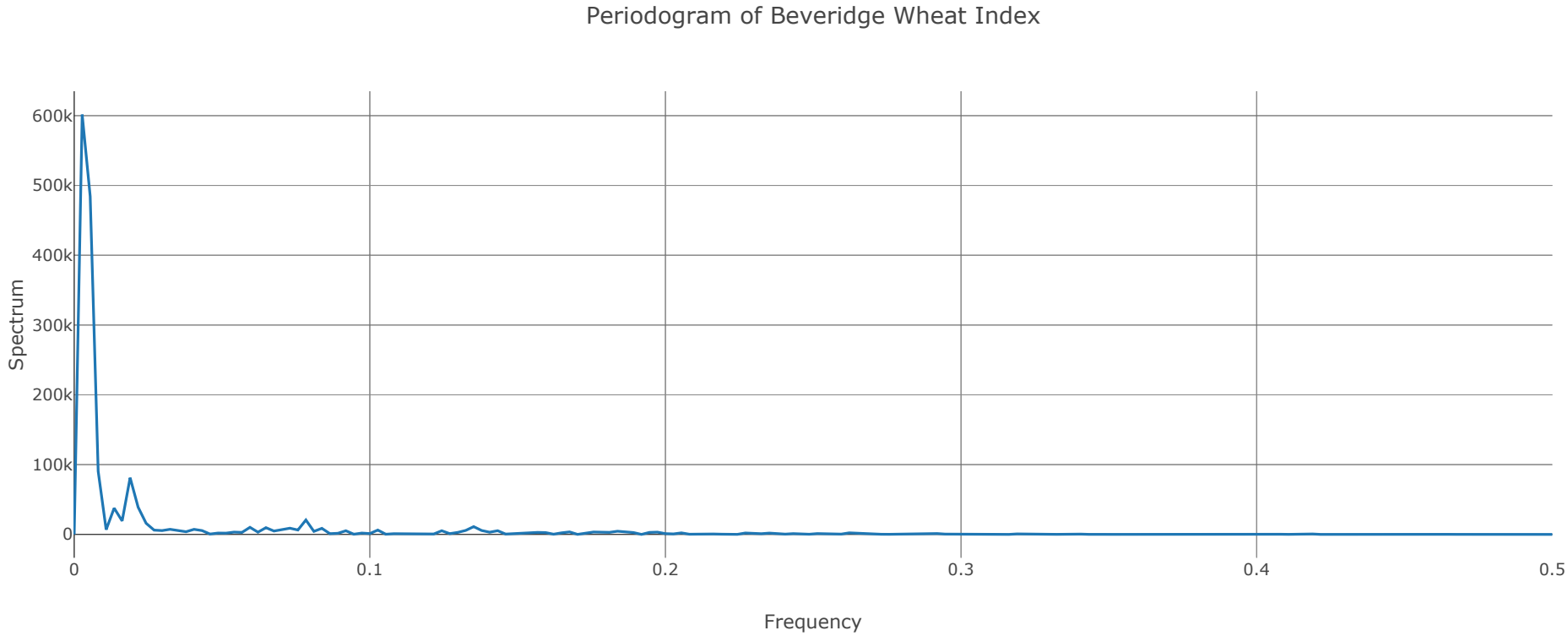
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
from pandas import DataFrame, read_csv
import matplotlib.pyplot as plt
import pandas as pd
import requests
import datetime
import matplotlib.pyplot as plt
# import plotly.graph_objects as go
import plotly.offline as py
import plotly
from plotly.offline import init_notebook_mode
from plotly import graph_objs as go
plotly.offline.init_notebook_mode(connected=True)

file = r'beveridge-wheat-price-index-1500.xls'
df = pd.read_excel(file)
df.columns = ['year', 'price']
df = df[12:]
# df.set_index('year',inplace=True)
df_price = pd.to_numeric(df['price'])
```

In [2]:

```
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt
fs = 1 # sampling rate
x = df_price
freq, espectro = signal.periodogram(x, fs)

data=[go.Scatter(x=df['year'], y=df_price, mode='lines')]
layout = go.Layout(
    title='Beveridge Wheat index',
    xaxis={'title':'Year'},
    yaxis={'title':'Index'}
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

data=[go.Scatter(x=freq, y=espectro, mode='lines')]
layout = go.Layout(
    title='Periodogram of Beveridge Wheat Index',
    xaxis={'title':'Frequency'},
    yaxis={'title':'Spectrum'}
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

index_greatest = np.where(espectro == max(espectro))[0]
print(f"PICOS DO PERIODOGRAMA")
print(f"Maior Pico: {max(espectro)} em frequência: {freq[index_greatest]}")
tmp_pico = max(espectro)
tmp_index_greatest = index_greatest
espectro[index_greatest] = 0
index_greatest = np.where(espectro == max(espectro))[0]
print(f"Segundo Maior Pico: {max(espectro)} em frequência: {freq[index_greatest]}")
espectro[tmp_index_greatest] = tmp_pico

import matplotlib.pyplot as plt
import numpy as np
from scipy import interpolate
from scipy import ndimage

print("**************************************************************************")
print(f"Estimador Suavizado de periodograma usando filtro Gaussiano")
print("**************************************************************************")

print(f"\nEstimação não paramétrica\n")

print(f"\nPara construir um estimador suavizado de periodograma,\n"+
      f"devemos aplicar o método de suavização no domínio do frequência\n"+
      f"depois de calcular o periodograma cru")

y_sm = espectro
x_sm = freq

y = y_sm.tolist()
```

```
x = x_sm.tolist()

y_sm = np.array(y)
x_sm = np.array(x)

# resample to lots more points - needed for the smoothed curves
x_smooth = np.linspace(x_sm.min(), x_sm.max(), 200)

sigma = 5 #tamanho da janela do filtro gaussiano
x_g1d2 = ndimage.gaussian_filter1d(x_sm, sigma)
y_g1d2 = ndimage.gaussian_filter1d(y_sm, sigma)

sigma = 2 #tamanho da janela do filtro gaussiano
x_g1d3 = ndimage.gaussian_filter1d(x_sm, sigma)
y_g1d3 = ndimage.gaussian_filter1d(y_sm, sigma)

sigma = 1 #tamanho da janela do filtro gaussiano
x_g1d4 = ndimage.gaussian_filter1d(x_sm, sigma)
y_g1d4 = ndimage.gaussian_filter1d(y_sm, sigma)

data = [go.Scatter(x=x_sm, y=y_sm, mode='lines', name="Original Periodogram")]
data += [go.Scatter(x=x_g1d2, y=y_g1d2, mode='lines', name="Gaussian Kernel width 5")]
data += [go.Scatter(x=x_g1d3, y=y_g1d3, mode='lines', name="Gaussian Kernel width 2")]
data += [go.Scatter(x=x_g1d4, y=y_g1d4, mode='lines', name="Gaussian Kernel width 1")]

layout = go.Layout(
    title='Smoothed Periodogram of Beveridge Wheat Index',
    xaxis={'title':'Smoothed Spectrum'},
    yaxis={'title':'Frequency'}
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

print(f"\nObs2.: O tamanho de janela 1 parece mais adequado\n\n")
```

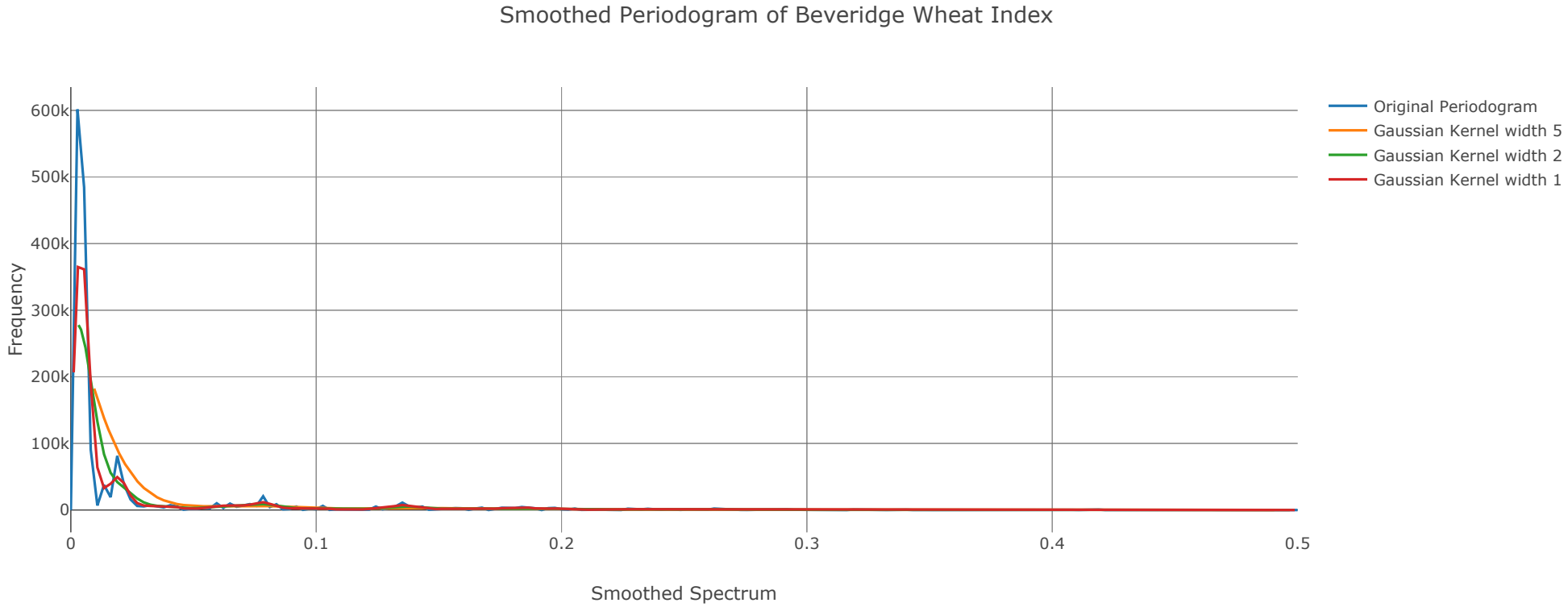

Beveridge Wheat index
Index
Year
400
350
300
250
200
150
100
50
0
1500
1550
1600
1650
1700
1750
1800
1850




PICOS DO PERIODOGRAMA
Maior Pico: 601545.3047090125 em frequência: [0.0027027]
Segundo Maior Pico: 484475.4537271647 em frequência: [0.00540541]
**********
Estimador Suavizado de periodograma usando filtro Gaussiano
**********

Estimação não paramétrica

Para construir um estimador suavizado de periodograma,
devemos aplicar o método de suavização no domínio do frequência
depois de calcular o periodograma cru

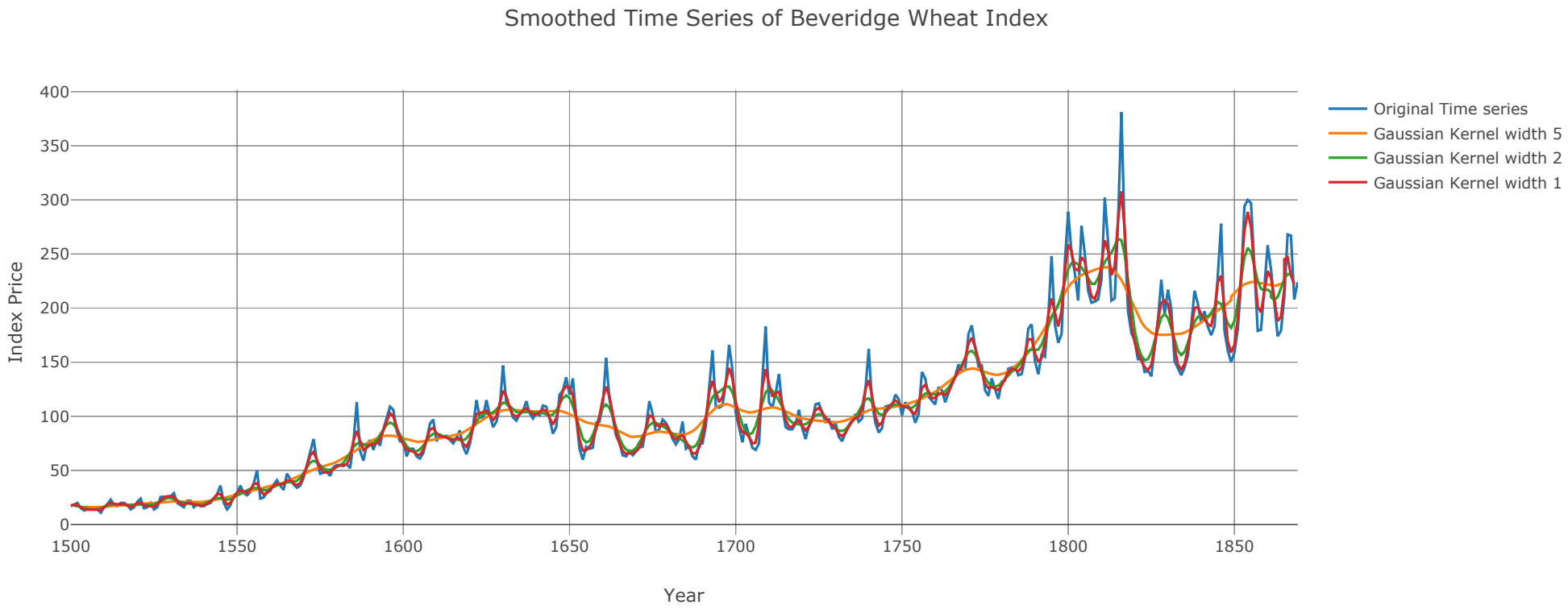



Obs2.: O tamanho de janela 1 parece mais adequado

In [3]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import interpolate
from scipy import ndimage

print("************************************************************************")
print(f"Estimador Suavizado de covariância usando filtro Gaussiano")
print("************************************************************************")
print(f"Estimação não paramétrica")

y_sm = df_price.values
x_sm = df["year"].values

y = y_sm.tolist()
x = x_sm.tolist()

y_sm = np.array(y)
x_sm = np.array(x)

# resample to lots more points - needed for the smoothed curves
x_smooth = np.linspace(x_sm.min(), x_sm.max(), 200)

sigma = 5 #tamanho da janela do filtro gaussiano
x_g1d2 = ndimage.gaussian_filter1d(x_sm, sigma)
y_g1d2 = ndimage.gaussian_filter1d(y_sm, sigma)

sigma = 2 #tamanho da janela do filtro gaussiano
x_g1d3 = ndimage.gaussian_filter1d(x_sm, sigma)
y_g1d3 = ndimage.gaussian_filter1d(y_sm, sigma)

sigma = 1 #tamanho da janela do filtro gaussiano
x_g1d4 = ndimage.gaussian_filter1d(x_sm, sigma)
y_g1d4 = ndimage.gaussian_filter1d(y_sm, sigma)

data = [go.Scatter(x=x_sm, y=y_sm, mode='lines', name="Original Time series")]
# data += [go.Scatter(x=x_g1d, y=y_g1d, mode='lines', name="Gaussian Kernel width 10")]
data += [go.Scatter(x=x_g1d2, y=y_g1d2, mode='lines', name="Gaussian Kernel width 5")]
data += [go.Scatter(x=x_g1d3, y=y_g1d3, mode='lines', name="Gaussian Kernel width 2")]
data += [go.Scatter(x=x_g1d4, y=y_g1d4, mode='lines', name="Gaussian Kernel width 1")]
# data += [go.Scatter(x=x_g1d5, y=y_g1d5, mode='lines', name="Gaussian Kernel width 0.5")]

layout = go.Layout(
    title='Smoothed Time Series of Beveridge Wheat Index',
    xaxis={'title':'Year'},
    yaxis={'title':'Index Price'}
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)

print(f"Para construir um estimador suavizado de covariância,\n"+
      f"devemos aplicar o método de suavização no domínio do tempo\n"+
      f"para então calcular o periodograma")
print(f"Obs.: Para comparação, mostra-se o spline da série")
```

```
print(f"\nObs2.: O tamanho de janela 2 parece mais adequado\n\n")

print("**********************************************************************")
print(f"Calculando periodograma a partir da série suavizada no tempo")
print("**********************************************************************")

fs = 1
freq, espectro = signal.periodogram(x_g1d3, fs)
data = [go.Scatter(x=freq, y=espectro, mode='lines', name="Smoothed Resulting Periodogram")]
# data += [go.Scatter(x=x_g1d5, y=y_g1d5, mode='lines', name="Gaussian Kernel width 0.5")]

layout = go.Layout(
    title='Estimador suavizado de covariância da série "Beveridge Wheat index"',
    xaxis={'title':'Frequency'},
    yaxis={'title':'Espectro'}
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig)
```

```
**********************************************************************
Estimador Suavizado de covariância usando filtro Gaussiano
**********************************************************************
Estimação não paramétrica
```




```
Para construir um estimador suavizado de covariância,
devemos aplicar o método de suavização no domínio do tempo
para então calcular o periodograma
Obs.: Para comparação, mostra-se o spline da série

Obs2.: O tamanho de janela 2 parece mais adequado


**********************************************************************
Calculando periodograma a partir da série suavizada no tempo
**********************************************************************
```


Estimador suavizado de covariância da série "Beveridge Wheat index"
Espectro
2.5M
2M
1.5M
1M
0.5M
0
0
0.1
0.2
0.3
0.4
0.5
Frequency

In [4]:
```
from pylab import legend
import scipy.signal
from spectrum import Periodogram, pyule
print("**********************************************************************")
print(f"Estimador espectral autorregressivo baseado em Yule-Walker usando AR(4)")
print("**********************************************************************")
print(f"Estimação paramétrica - Hipótese Autorregressiva")
y = df_price.values
p = Periodogram(y, sampling=2)
p.plot()
p = pyule(y, 4, sampling=2, scale_by_freq=False)
p.plot()
legend(['PSD of model output','PSD estimate of x using Yule-Walker AR(4)'])
```

```
**********************************************************************
Estimador espectral autorregressivo baseado em Yule-Walker usando AR(4)
**********************************************************************
Estimação paramétrica - Hipótese Autorregressiva
```

Out[4]:
```
<matplotlib.legend.Legend at 0x1c1e55d4a8>
```