Work

May 13, 2020

[1]:

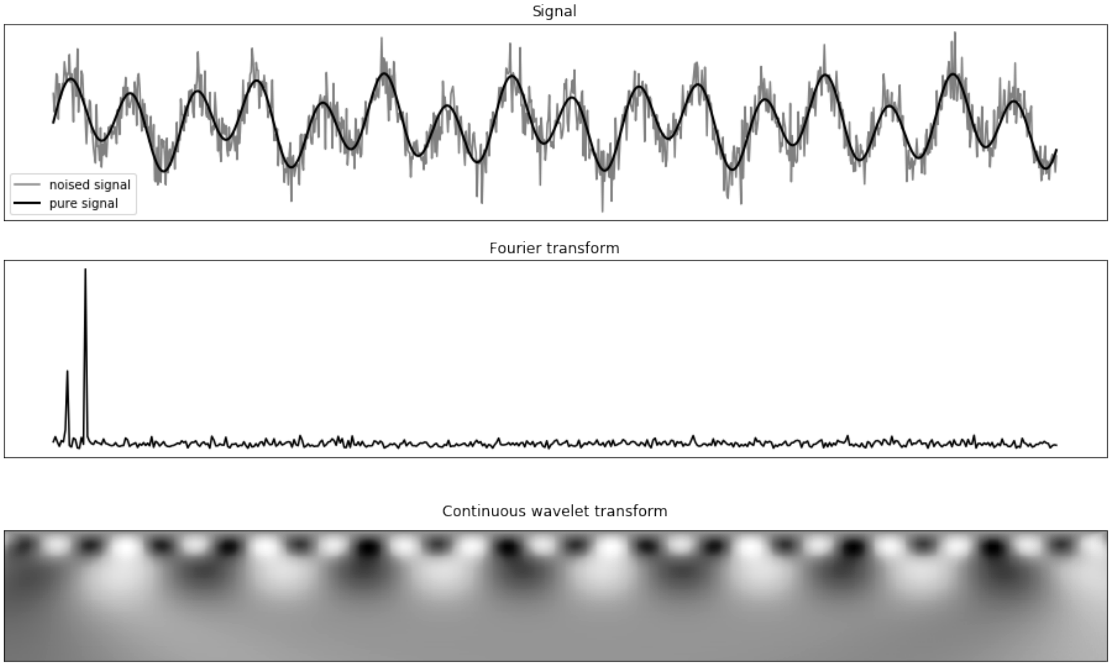
```
import pywt
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

np.random.seed(0)
```

[2]:

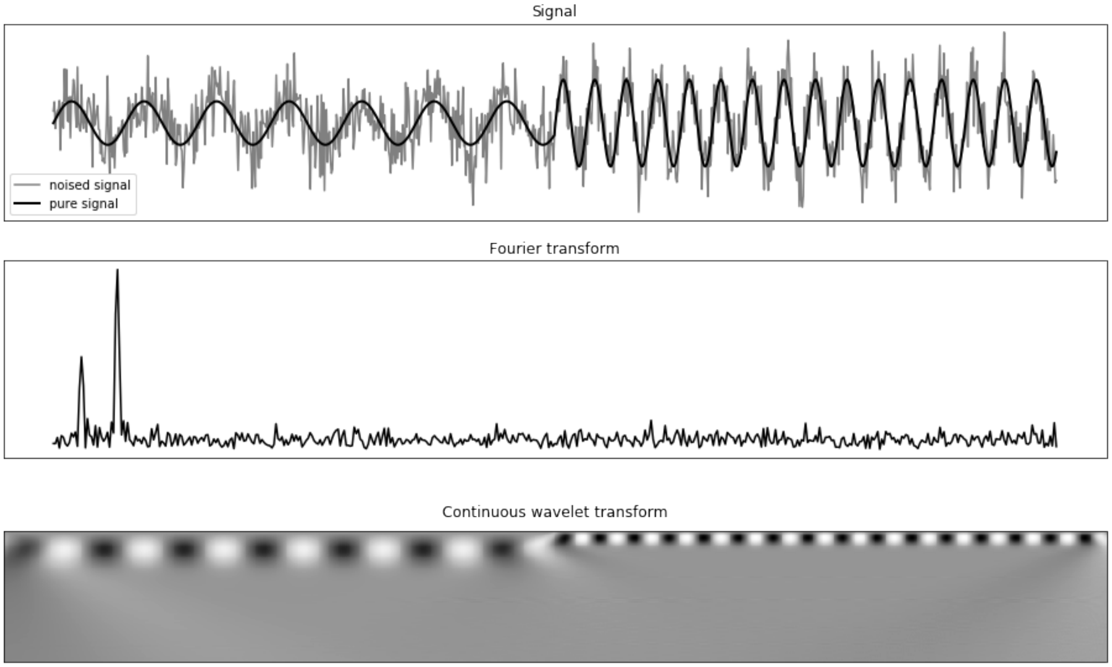
```
X_range = np.arange(1000)

def describe(signal):
    plt.figure(figsize=(16, 10))
    signal_with_noise = signal + np.random.normal(0, 1, len(signal))

    plt.subplot(3, 1, 1)
    plt.plot(signal_with_noise, 'gray', label='noised signal')
    plt.plot(signal, 'black', label='pure signal', linewidth=2)
    plt.xticks(()), plt.yticks(())
    plt.title('Signal')
    plt.legend()

    plt.subplot(3, 1, 2)
    plt.plot(np.abs(np.fft.rfft(signal_with_noise)), 'black')
    plt.title('Fourier transform')
    plt.xticks(()), plt.yticks(())

    ax = plt.subplot(3, 1, 3)
    coef, freqs=pywt.cwt(signal ,np.arange(1, 120),'mexh')
    ax.matshow(coef, cmap='Greys')
    plt.title('Đąontinuous wavelet transform')
    plt.xticks(()), plt.yticks(())
    plt.show()
```

[3]:

```
describe(np.sin(X_range / 23) + 2 * np.sin(X_range / 10))
```



```
[4]: describe(np.append(np.sin(X_range[::2] / 23), 2 * np.sin(X_range[::2] / 10)))
```

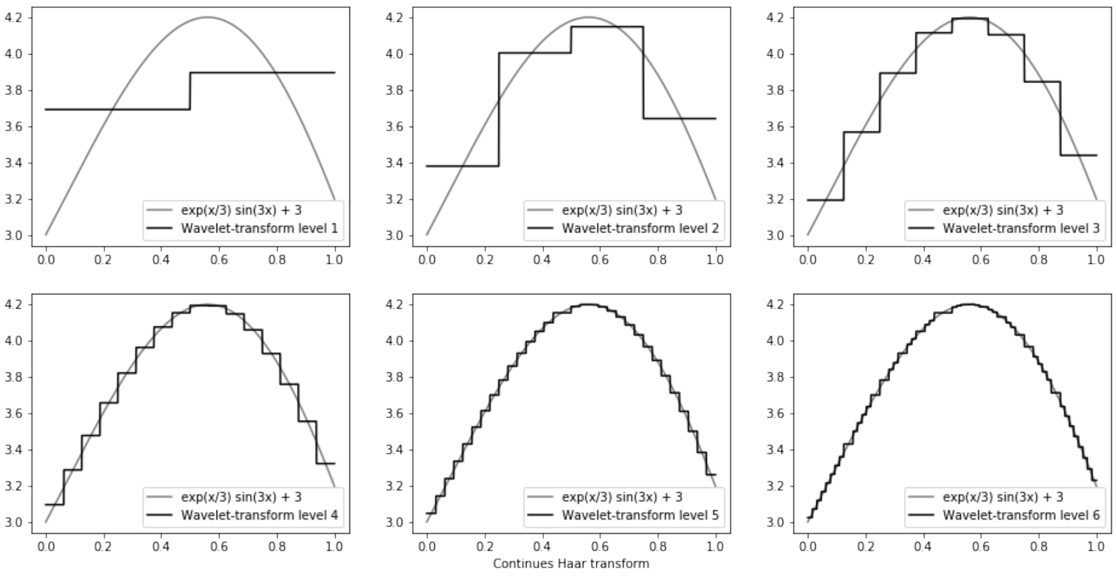


[5]:

```
from scipy import integrate

class wavelet_series:
    def __init__(self, g, levels=8):
        self.levels = levels
        mother_wavelet = lambda x: 0 if x < 0 else 1 if x < 0.5 else -1 if x < 1␣
 ↪else 0

        self.scaling = lambda x: 1 if 0 <= x < 1 else 0

        self.basis = [[
                (lambda i, j: lambda x: 2 ** (i / 2) * mother_wavelet(2**i * x -␣
 ↪j))(i, j)
            for j in range(2 ** i)] for i in range(levels)]

        self.coef = [[
                integrate.quad(lambda x: g(x) * self.basis[i][j](x), 0, 1)[0]
            for j in range(2 ** i)] for i in range(levels)]

        self.scaling_coef = integrate.quad(
            lambda x: g(x) * self.scaling(x), 0, 1)[0]

    def __call__(self, point):
        value = 0
        for i in range(self.levels):
            for j in range(2 ** i):
                value += self.coef[i][j] * self.basis[i][j](point)
        return value + self.scaling_coef * self.scaling(point)
```

[6]:

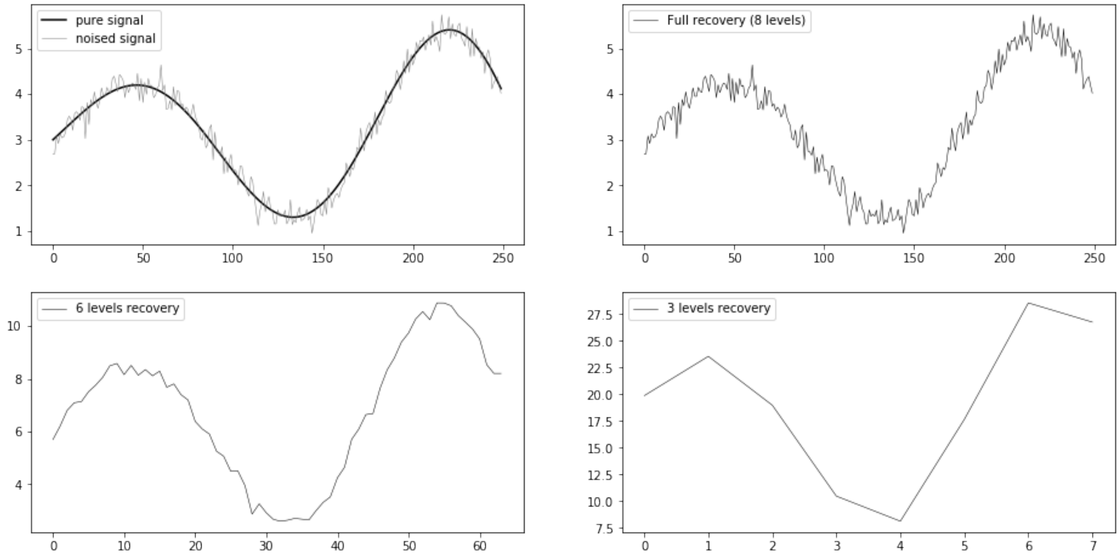
```
g = lambda x: np.exp(x / 3) * np.sin(3 * x) + 3
xs = np.linspace(1e-8, 1 - 1e-8, 1000)

plt.figure(figsize=(16, 8))

for i in range(1, 7):
    f = wavelet_series(g, i)
    plt.subplot(2, 3, i)
    plt.plot(xs, list(map(g, xs)), 'grey', label='exp(x/3) sin(3x) + 3')
    plt.plot(xs, list(map(f, xs)), 'black', label='Wavelet-transform level %d' %␣
 ↪i)
    plt.legend()

    if i == 5:
        plt.xlabel('Continues Haar transform')

plt.show()
```



```
[7]: def decomposition(signal, wavelet='Haar'):
         wavelet = pywt.Wavelet(wavelet)
         signal_with_noise = signal + np.random.normal(0, 0.2, len(signal))

         plt.figure(figsize=(16, 8))
         plt.subplot(2, 2, 1)
         plt.plot(signal, 'black', label='pure signal')
         plt.plot(signal_with_noise,'gray', label='noised signal', linewidth=0.5)
         plt.legend()

         coefs = pywt.wavedec(signal_with_noise, wavelet, level=8)

         plt.subplot(2, 2, 2)
         plt.plot(pywt.waverec(coefs, wavelet), 'black', label='Full recovery (8␣
      ↪levels)', linewidth=0.5)
         plt.legend()

         plt.subplot(2, 2, 3)
         plt.plot(pywt.waverec(coefs[:-2], wavelet), 'black', label='6 levels␣
      ↪recovery', linewidth=0.5)
         plt.legend()

         plt.subplot(2, 2, 4)
         plt.plot(pywt.waverec(coefs[:-5], wavelet), 'black', label='3 levels␣
      ↪recovery', linewidth=0.5)
         plt.legend()

         plt.show()
```

[8]:

```
decomposition(g(np.linspace(0, 3, 250)))
```

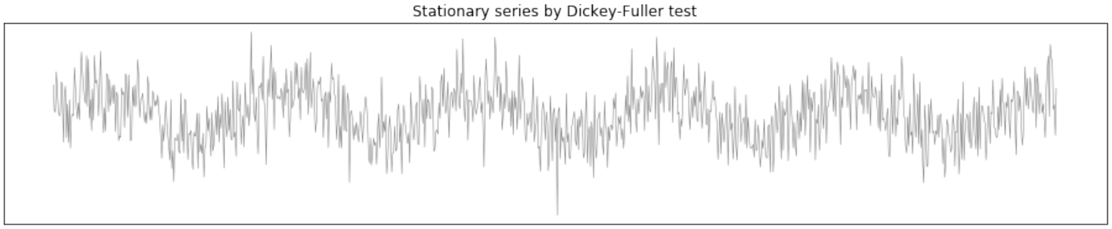


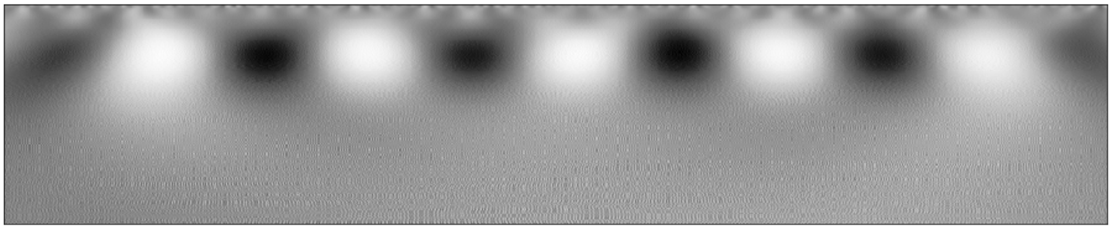
[9]:

```
def ShowSeries(series):
    test = sm.tsa.adfuller(series)

    plt.figure(figsize=(16, 3))
    plt.plot(series, 'Grey', linewidth=0.5)
    plt.xticks(())
    plt.yticks(())
    if test[0] > test[4]['5%']:
        plt.title('Non-stationary series by Dickey-Fuller test')
    else:
        plt.title('Stationary series by Dickey-Fuller test')

    coef, freqs = pywt.cwt(series ,np.arange(1, 200), 'morl')
    plt.matshow(coef, cmap='Greys')
    plt.xticks(())
    plt.yticks(())
    plt.show()

    print('adf: ', test[0])
    print('p-value: ', test[1])
    print('Critical values: ', test[4])
```

[10]:

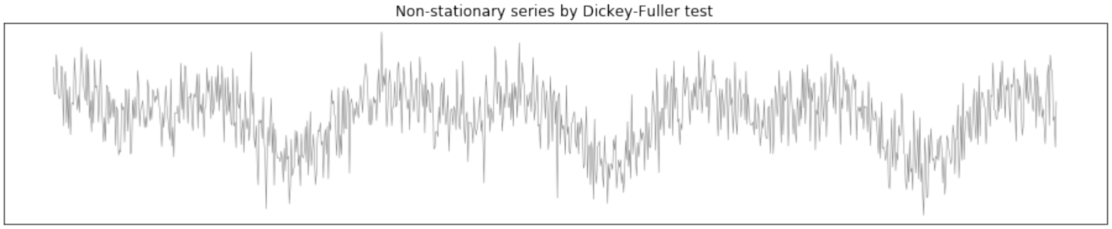
```
noise = np.random.normal(0, 1, 1000)
x_range = np.arange(1000)

ShowSeries(np.sin(x_range / 30) + noise)
```

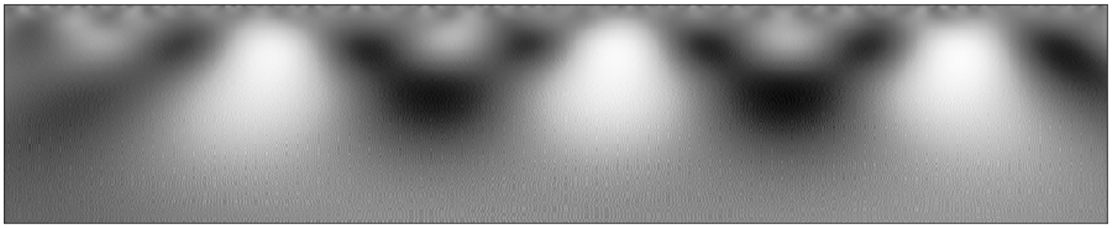

Stationary series by Dickey-Fuller test



```
adf:  -3.481848853184378
p-value:  0.008465675598229853
Critical values:  {'1%': -3.4369860032923145, '5%': -2.8644697838498376, '10%':
-2.5683299626694422}
```

[11]:
```
ShowSeries(np.sin(x_range / 50) + np.sin(x_range / 25 - 5) + noise)
```

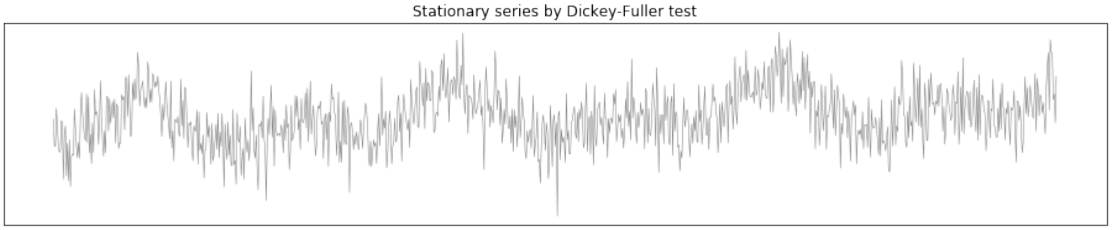

Non-stationary series by Dickey-Fuller test

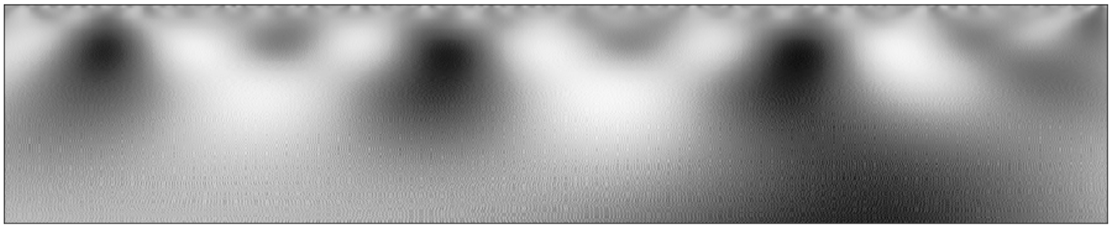


```
adf:  -2.903023162949968
```

```
p-value:  0.0449975725101061
Critical values:  {'1%': -3.4369860032923145, '5%': -2.8644697838498376, '10%':
-2.5683299626694422}
```

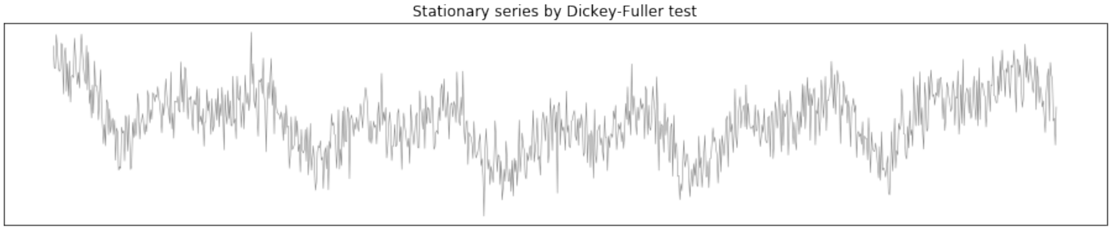
[12]:
```python
ShowSeries(np.sin(x_range / 50) + np.sin(x_range / 25 - 2) + + x_range / 500 + noise * 1.3)
```





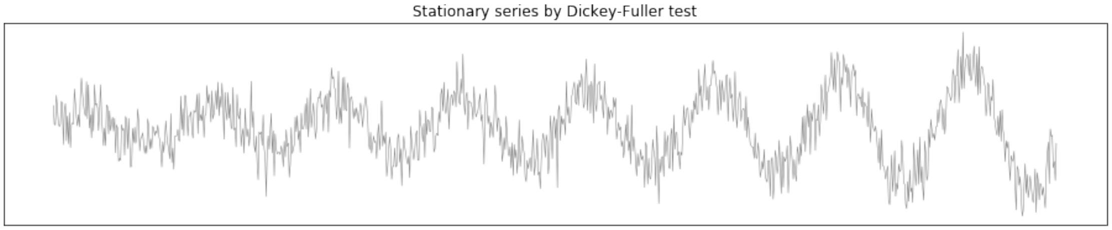
```
adf:  -3.276199205944301
p-value:  0.015976475379125093
Critical values:  {'1%': -3.4369860032923145, '5%': -2.8644697838498376, '10%':
-2.5683299626694422}
```

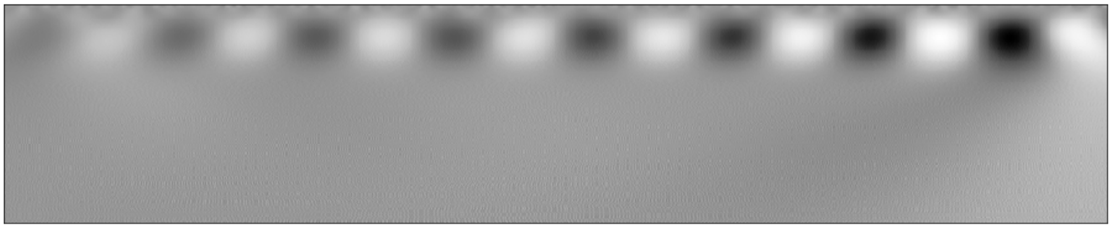
[13]:
```python
ShowSeries(np.sin(x_range / 30 + 2) + np.sin(x_range / 15) + ((x_range - len(x_range) / 2) / 400)**2 + noise)
```



```
adf:  -3.5973135357984862
p-value:  0.005812595366803051
Critical values:  {'1%': -3.4369658620871286, '5%': -2.8644609013346485, '10%':
-2.56832523159495}
```

[14]:
```
ShowSeries((np.sin(x_range / 20)) * (x_range + len(x_range))**2 + noise * 1e6)
```

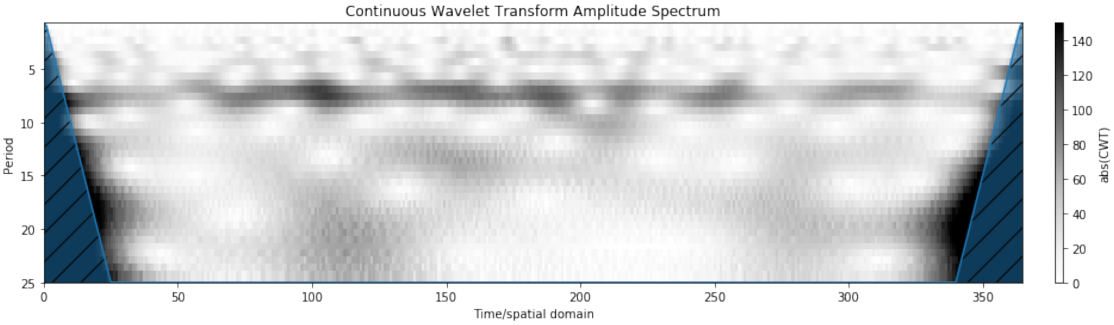




```
adf:  -5.770885118947002
p-value:  5.39587787602044e-07
Critical values:  {'1%': -3.4370471695043037, '5%': -2.8644967585264784, '10%':
-2.5683443301882956}
```

[15]:
```
df = pd.read_csv('nss15.csv')

time = pd.to_datetime(df.treatmentDate, format='%m/%d/%Y')
timeseries = pd.Series(np.ones(len(time)), index=sorted(time), dtype=np.int64).
 ↪resample('D').sum()
```

[16]:
```
from scaleogram import cws

cws(timeseries, cmap='Greys', clim=[0, 150], figsize=(16, 4))
plt.show()
```

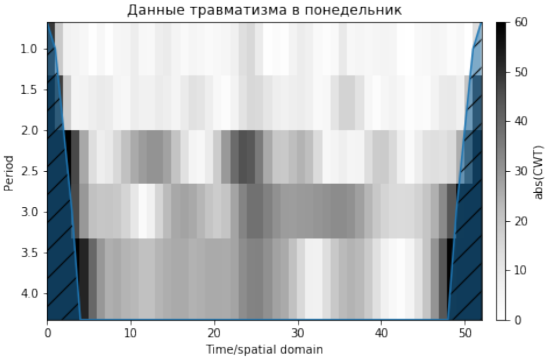


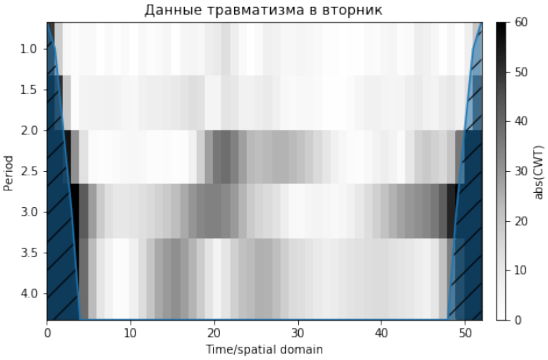
[17]:
```
timeseries.index = pd.Series(timeseries.index).apply(lambda day: day.weekday())
```

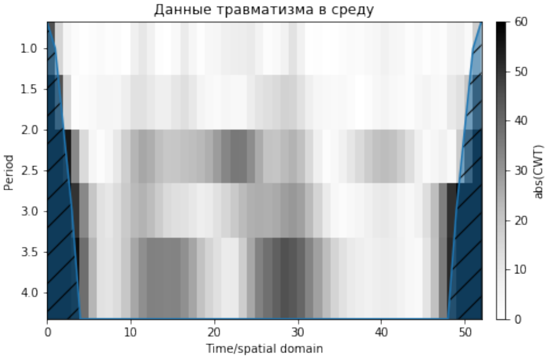
[18]:
```
dayweeks = ['Đ£Đ¿Đ¡ĐţĐťĐţĐżÑŇĐ¡ĐÿĐž', 'ĐšÑĆĐ¿ÑĂĐ¡ĐÿĐž', 'ÑĄÑĂĐţĐťÑČ',␣
 ↪'ÑĞĐţÑĆĐšĐţÑĂĐş', 'Đ£ÑŔÑĆĐ¡ĐÿÑĘÑČ', 'ÑĄÑČĐśĐśĐ¿ÑĆÑČ', 'ĐšĐ¿ÑĄĐžÑĂĐţÑĄĐţĐ¡ÑŇĐţ']

for i in range(7):
    print('ĐŠ {:12s}: {} ÑĄĐżÑČÑĞĐřĐţĐš'.format(dayweeks[i], timeseries[i].
 ↪sum()))
```

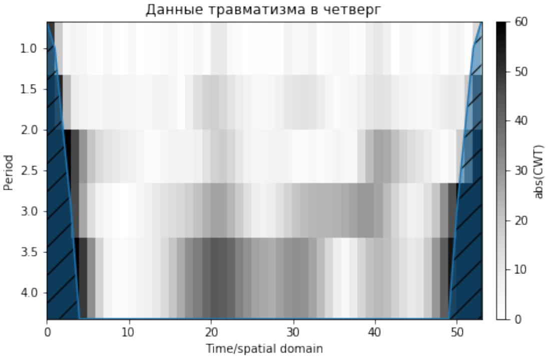
```
ĐŠ Đ£Đ¿Đ¡ĐţĐťĐţĐżÑŇĐ¡ĐÿĐž : 49868 ÑĄĐżÑČÑĞĐřĐţĐš
ĐŠ ĐšÑĆĐ¿ÑĂĐ¡ĐÿĐž     : 46629 ÑĄĐżÑČÑĞĐřĐţĐš
ĐŠ ÑĄÑĂĐţĐťÑČ       : 46727 ÑĄĐżÑČÑĞĐřĐţĐš
ĐŠ ÑĞĐţÑĆĐšĐţÑĂĐş     : 46798 ÑĄĐżÑČÑĞĐřĐţĐš
ĐŠ Đ£ÑŔÑĆĐ¡ĐÿÑĘÑČ     : 45269 ÑĄĐżÑČÑĞĐřĐţĐš
ĐŠ ÑĄÑČĐśĐśĐ¿ÑĆÑČ     : 49408 ÑĄĐżÑČÑĞĐřĐţĐš
ĐŠ ĐšĐ¿ÑĄĐžÑĂĐţÑĄĐţĐ¡ÑŇĐţ : 50140 ÑĄĐżÑČÑĞĐřĐţĐš
```

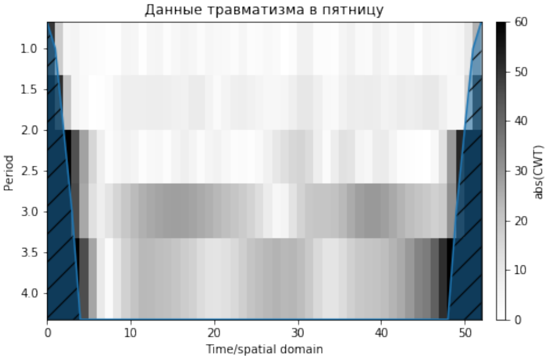
[19]:
```
plt.figure(figsize=(16, 16))

for i in range(6):
    ax = plt.subplot(3, 2, i + 1)
    cws(timeseries[i], cmap='Greys', ax=ax, clim=[0, 60], title='ĐŤĐřĐ¡Đ¡ÑŃĐţ␣
 ↪ÑĆÑĂĐřĐšĐijĐřÑĆĐÿĐůĐijĐř ĐŠ %s' % dayweeks[i])
plt.show()

plt.figure(figsize=(16, 4))
ax = plt.subplot(1, 1, 1)
cws(timeseries[6], cmap='Greys', ax=ax, clim=[0, 60], title='ĐŤĐřĐ¡Đ¡ÑŃĐţ␣
 ↪ÑĆÑĂĐřĐšĐijĐřÑĆĐÿĐůĐijĐř ĐŠ %s' % dayweeks[i])
plt.show()
```

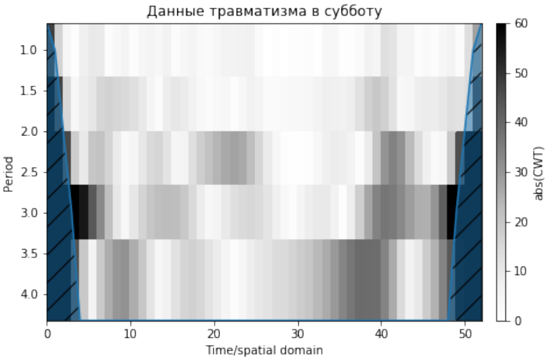

Данные травматизма в понедельник

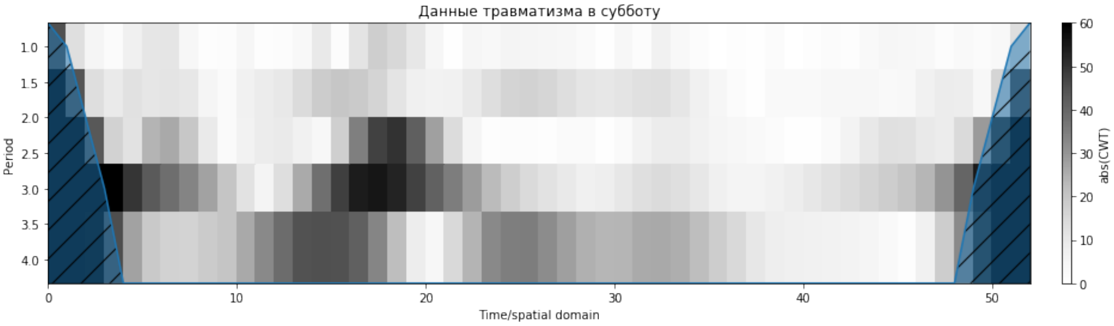

Данные травматизма в вторник


Данные травматизма в среду


Данные травматизма в четверг


Данные травматизма в пятницу


Данные травматизма в субботу


Данные травматизма в субботу