Analysis II Sommer 2017

Analyse von Pegeldaten mittels FFT

(Text auf Englisch)

Jörn Behrens (http://www.clisap.de/behrens) (joern.behrens@uni-hamburg.de) (mailto:joern.behrens@uni-hamburg.de)

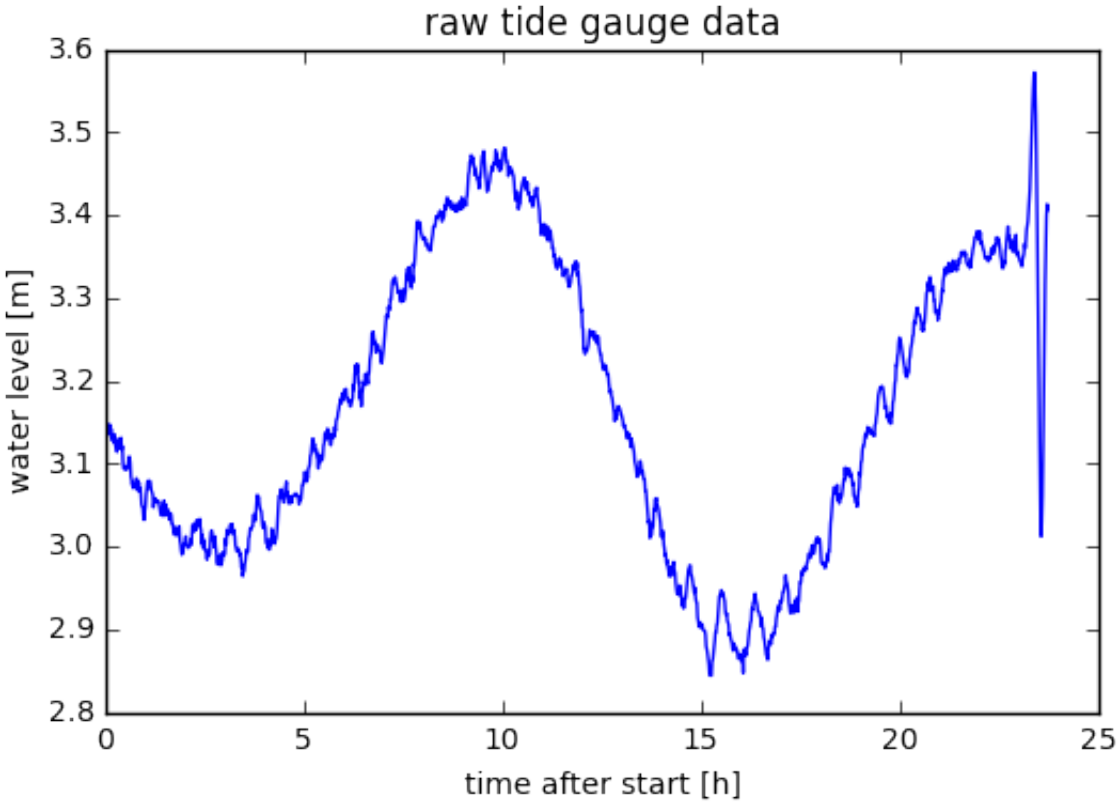
This example demonstrates the analysis of real life time series data. In order to demonstrate the power of fast Fourier transforms for data analysis, we first load time series data (sea surface elevation) at a coastal tide gauge in Padang, Western Sumatra, Indonesia. Data are given hourly from Sept. 29, 2009, 23:00 through Sept. 30, 2009, 6:00. However, we will generate our own normalized time starting with hour 1 at the first measurement.

In [1]:
```
from numpy import loadtxt, array
gauge=loadtxt('Gauge_data_pagang.txt', usecols=(1,))
hours=array(range(1,len(gauge)+1))/60.
```

Now, we want to plot the data to obtain an overview.

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(hours,gauge)
plt.title('raw tide gauge data')
plt.xlabel('time after start [h]')
plt.ylabel('water level [m]')
```

Out[2]: <matplotlib.text.Text at 0x105a429b0>



It is obvious that something happened after approx. 23 hours. But the tide is also clearly visible. So, let us analyze the data by an fft.

In [3]:
```
from numpy.fft import fft
Y=fft(gauge)
n=len(Y)
```

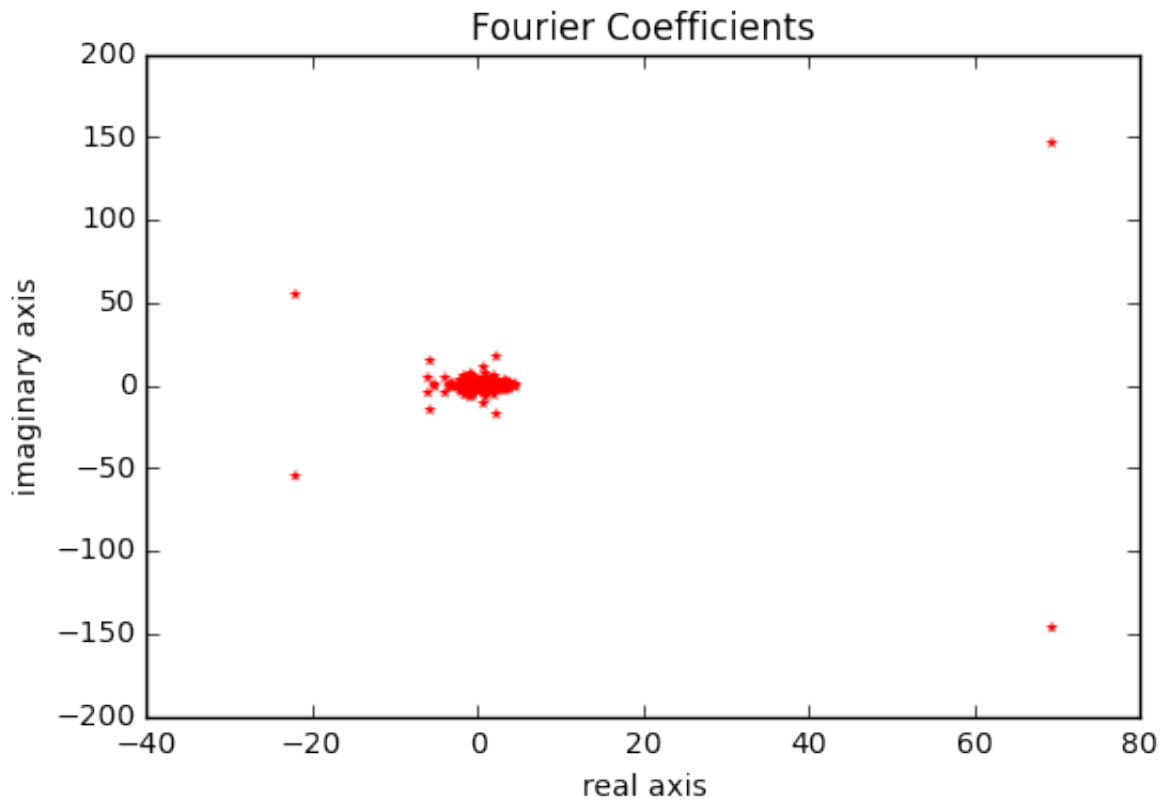
We discard the first entry of Y| since this - by definition - is just the sum of all entries.

In [4]:
```
from numpy import delete
ysav=Y[0]
Y=delete(Y,0)
```

Now that we have the coefficients, we plot them. Be aware that the coefficients are complex!

In [5]:
```
plt.scatter(Y.real,Y.imag,marker='*', c='r', linewidths=0)
plt.title('Fourier Coefficients')
plt.xlabel('real axis')
plt.ylabel('imaginary axis')
```

Out[5]: <matplotlib.text.Text at 0x105b8d208>



You can see that the coefficients are mirrowed at the real axis, so we need to consider only half of them for our analysis.

In [6]:
```
half=int(n/2)
```

The power spectrum consists of the absolute coefficients squared. And it tells us something about the *power* of each of the wave lengths represented in the Fourier series.

In [7]:
```
power= abs(Y[0:half])**2
```

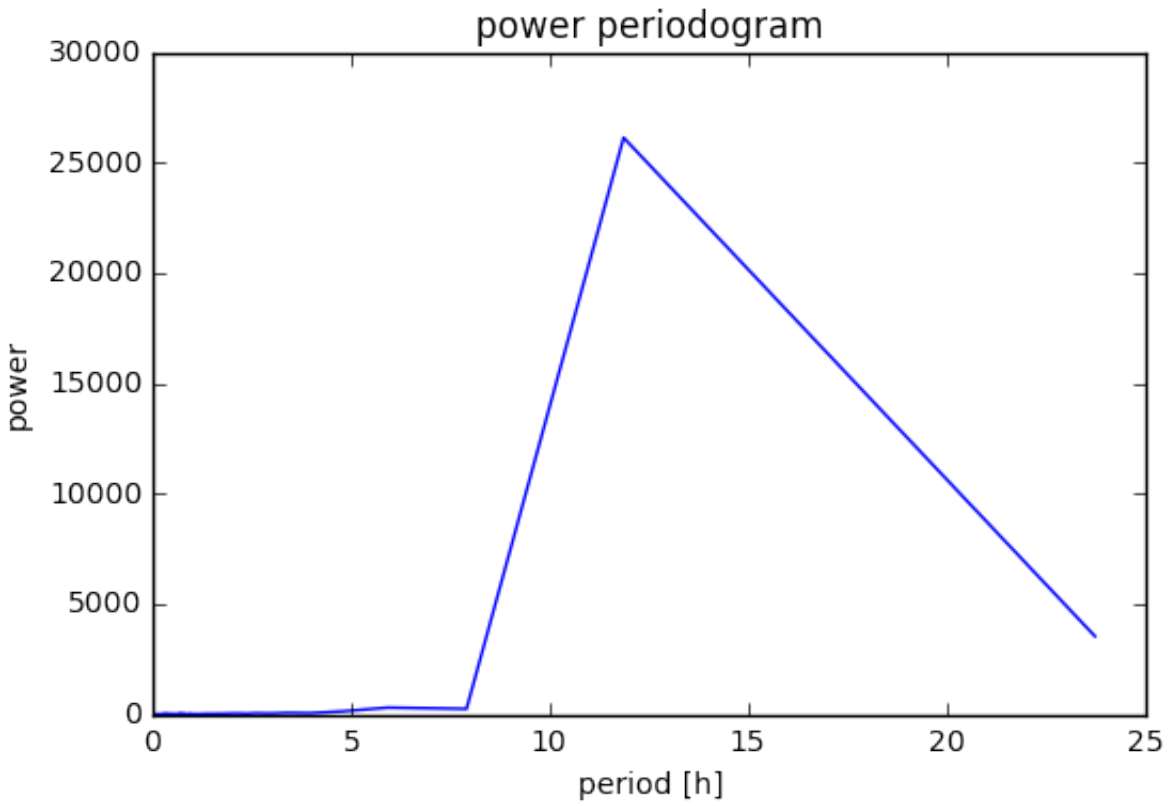
The frequency is just the wave number, while the period is 1/frequency. We also compute the period related to hours.

In [8]:
```
freq= array(range(1,int(half+1))).astype(float)/float(n)
peri= (1./freq)
perih=peri/60.
```

Now, let us see how this looks like:

In [9]:
```
plt.plot(perih,power)
plt.title('power periodogram')
plt.xlabel('period [h]')
plt.ylabel('power')
```

Out[9]: <matplotlib.text.Text at 0x105bb2438>



So, there is a clear and very strong maximum at about 11 hours period. We want to determine the exact period with the maximum:

In [10]:
```
from numpy import where, max
ind= where(power == max(power))[0][0]
period= peri[ind]/60.
print('Dominant period in hours: %0.5g' % (period))
```

```
Dominant period in hours: 11.867
```

We will now find the next most significant periods. We do this by setting the powerspectrum to zero at the index, where we find the most significant entry and finding the next maximum, and so on. We will do this for the four most significant entries.

In [11]:
```
pwanalys=power; pwanalys[ind]=0;
ind2=where(pwanalys == max(pwanalys))[0][0]
period2= peri[ind2]/60.
print('2nd dominant period in hours: %0.5g\n' % (period2))

pwanalys[ind2]=0
ind3=where(pwanalys == max(pwanalys))[0][0]
period3= peri[ind3]/60.
print('3rd dominant period in hours: %0.5g\n' % (period3))

pwanalys[ind3]=0
ind4=where(pwanalys == max(pwanalys))[0][0]
period4= peri[ind4]/60.
print('4th dominant period in hours: %0.5g\n' % (period4))
```

```
2nd dominant period in hours: 23.733

3rd dominant period in hours: 5.9333

4th dominant period in hours: 7.9111
```

Note that these correspond (more or less) to the domintant tidal components M2, K1, M4, M3...

Let us recover a smooth tidal signal from those four components. This would be an expected clean behavior at the tide gauge. All other signals would then be disturbances.

In [12]:
```
from numpy import zeros, concatenate
Yrecov=zeros(len(Y), dtype=complex)
Yrecov[ind]= Y[ind]; Yrecov[len(Y)-ind-2]= Y[len(Y)-ind-2]
Yrecov[ind2]= Y[ind2]; Yrecov[len(Y)-ind2-2]= Y[len(Y)-ind2-2]
Yrecov[ind3]= Y[ind3]; Yrecov[len(Y)-ind3-2]= Y[len(Y)-ind3-2]
Yrecov[ind4]= Y[ind4]; Yrecov[len(Y)-ind4-2]= Y[len(Y)-ind4-2]
Yrecov= concatenate(([ysav], Yrecov))
```

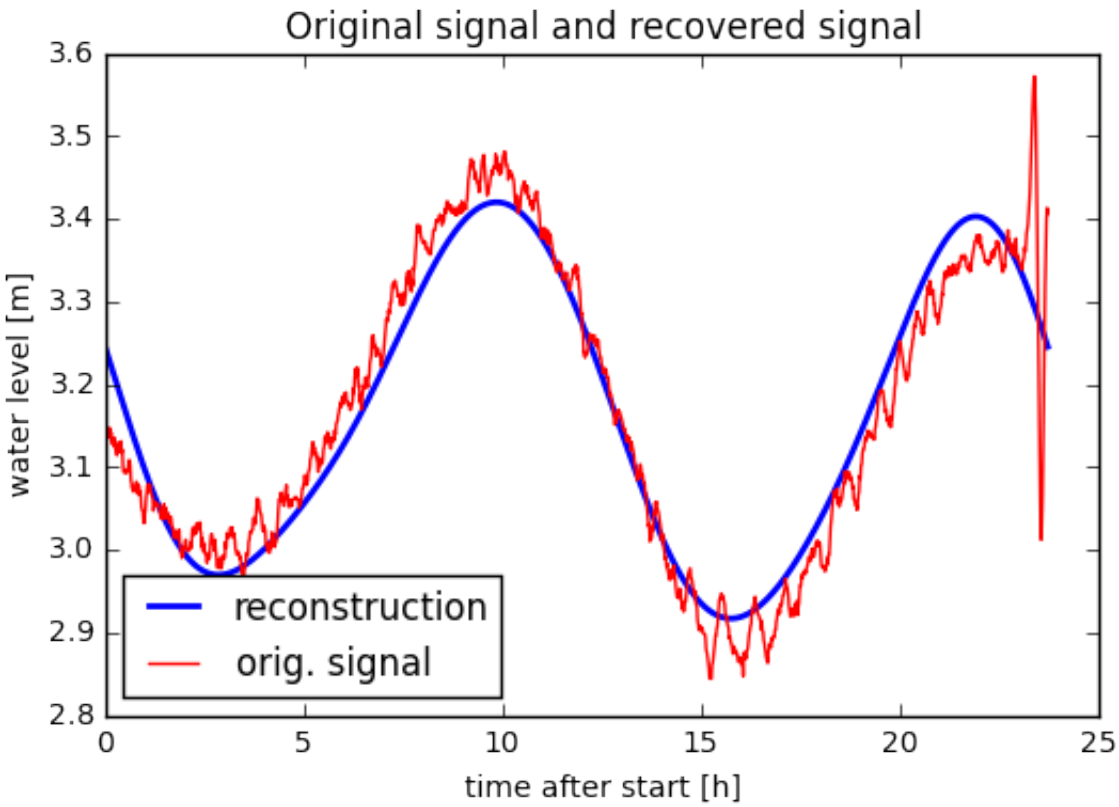
We use the inverse fft in order to reconstruct the wave from the coefficients extracted by means of the power spectrum.

In [13]:
```
from numpy.fft import ifft
x=ifft(Yrecov)
```

We now plot the real part of the reconstructed time series.

In [14]:
```
from numpy import real
gaugerecov=real(x)
h2=plt.plot(hours,gaugerecov,linewidth=2, c=[0,0,1], label='reconstruct
h3=plt.plot(hours,gauge,linewidth=1, c=[1,0,0], label='orig. signal')
plt.legend(loc='lower left')
plt.title('Original signal and recovered signal')
plt.xlabel('time after start [h]')
plt.ylabel('water level [m]')
```

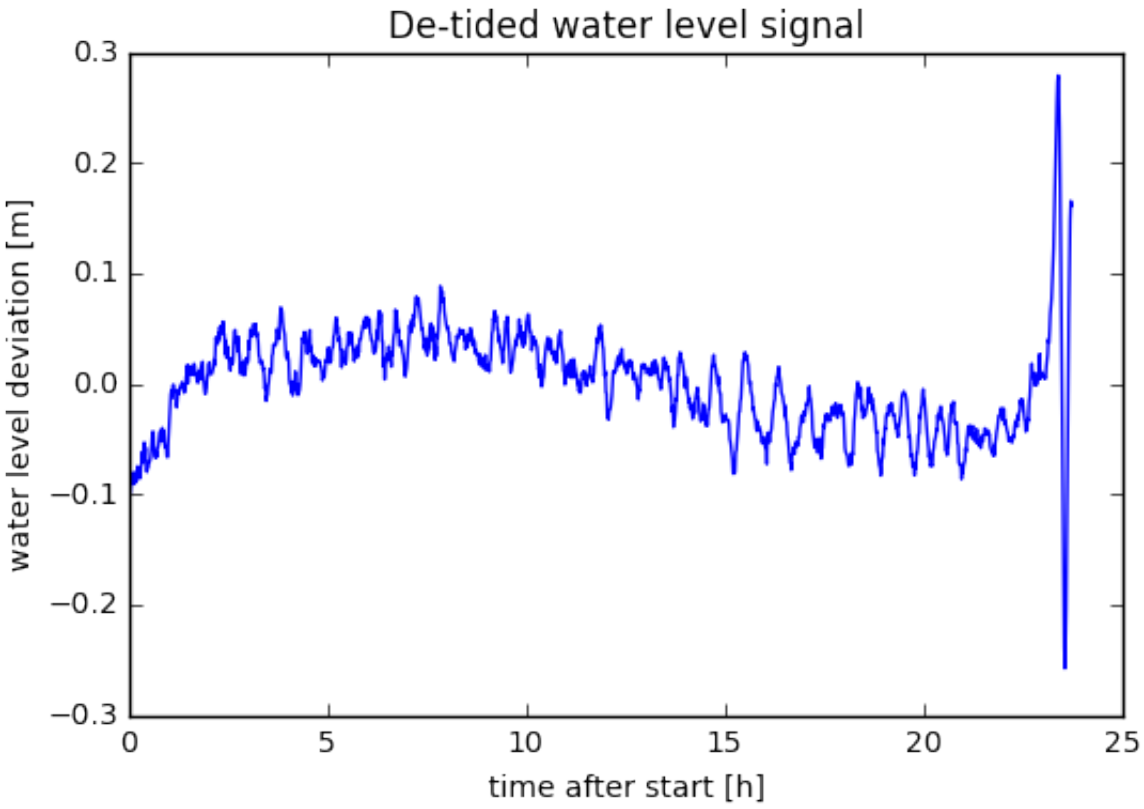
Out[14]: <matplotlib.text.Text at 0x105e56a20>



Finally we can use this smoothed recovery for de-tiding! We just subtract, what we assume to be the smooth expected period of the signal. The remainder is then the anomaly from mean (expected) sea level. This could potentially be used for automatic detection algorithms.

In [15]:
```
gaugedetide= gauge-gaugerecov
plt.figure(5)
plt.plot(hours,gaugedetide)
plt.title('De-tided water level signal')
plt.xlabel('time after start [h]')
plt.ylabel('water level deviation [m]')
```

Out[15]: <matplotlib.text.Text at 0x105f6c860>



So, to conclude: The anomaly that we saw in the beginning is indeed a small tsunami of an amplitude of approx. 0.5 m. It was generated by a magnitude 7.6 earthquake close to the city of Padang (see http://geofon.gfz-potsdam.de/eqinfo/event.php?id=gfz2009tdkv (http://geofon.gfz-potsdam.de/eqinfo/event.php?id=gfz2009tdkv)). You may find a simulation for this event at http://hdl.handle.net/10013/epic.34035.d001 (http://hdl.handle.net/10013/epic.34035.d001).

In [ ]: