Fourier spectral analysis

Fourier Transform (scipy.fftpack) of mathematical model

https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html
(https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html)

La transformada de Fourier transforma entre el dominio de tiempo al dominio de la frecuencia, la transformada inversa hace lo opuesto, así que aplicar la transformada inversa después de la transformada "derecha" reproduce los datos originales. Si los datos originales son reales, la transformada de Fourier generalmente produce números complejos.

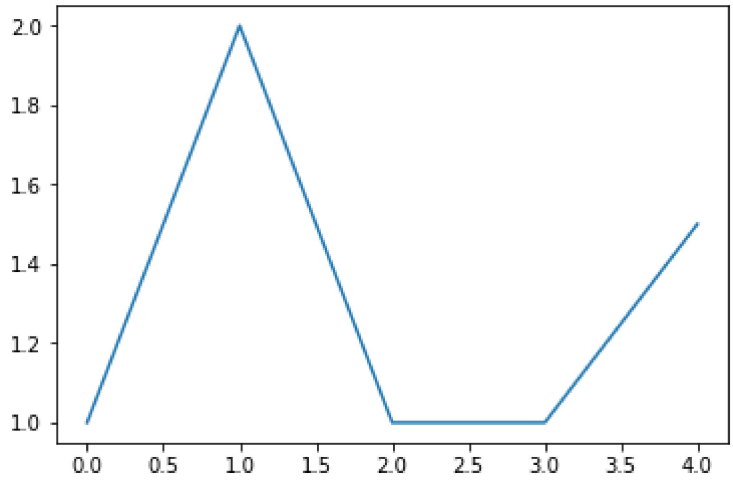
In [1]:
```
import numpy as np
from scipy.fftpack import fft, ifft
import matplotlib.pyplot as plt
import pandas as pd
```

In [23]:
```
x = np.array([1.0, 2.0, 1.0, -1.0, 1.5]) # original data
y = fft(x) # discrete Fourier transform
print(x)
print('first element of Fourier transform corresponds to sum of all elements o
f original series: ',np.sum(x))
print(y)
```

```
[ 1.   2.   1.  -1.   1.5]
first element of Fourier transform corresponds to sum of all elements of orig
inal series:  4.5
[ 4.50000000+0.j          2.08155948-1.65109876j -1.83155948+1.60822041j
 -1.83155948-1.60822041j  2.08155948+1.65109876j]
```

In [14]:
```
yinv = ifft(y) # inverse discrete Fourier transform
print(yinv)
print(abs(yinv)) # absolute values according to Pythagoras rule

plt.clf()
plt.plot(abs(yinv))
plt.show()
```

```
[ 1.0+0.j  2.0+0.j  1.0+0.j -1.0+0.j  1.5+0.j]
[ 1.   2.   1.   1.   1.5]
```



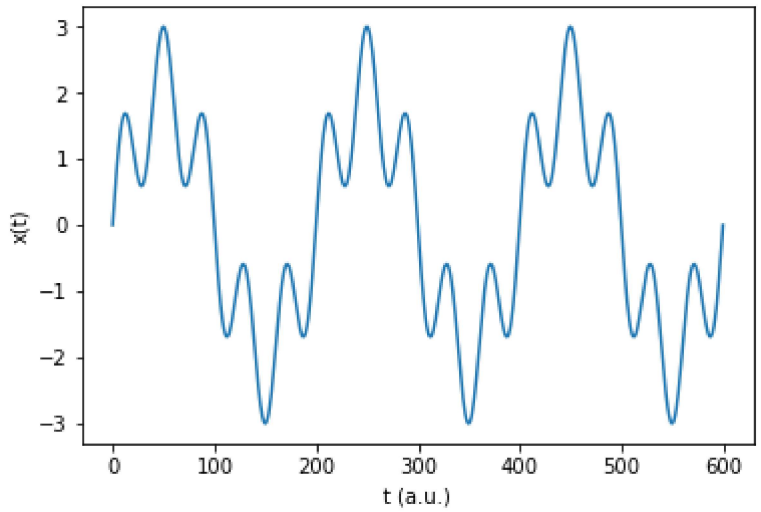
Transformada de Fourier de superposición de 2 funciones periódicas (2 senos)

In Python 3, they made the / operator do a floating-point division, and added the // operator to do integer division (i.e. quotient without remainder);

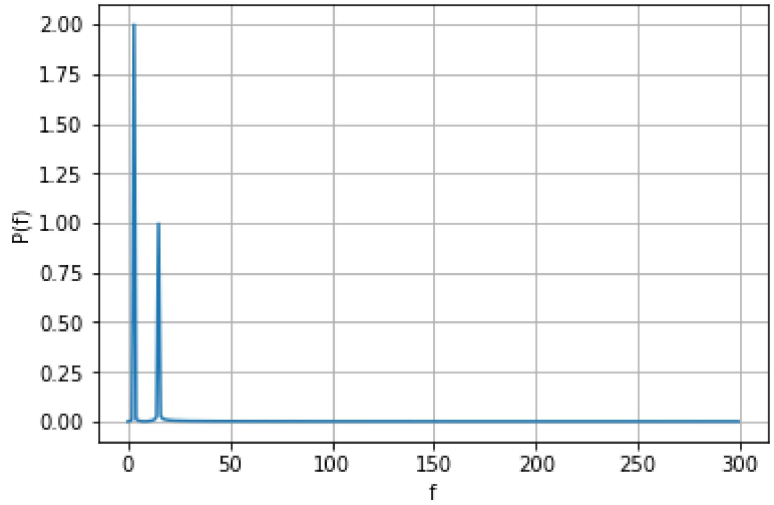
10/3=3.3333333333333335

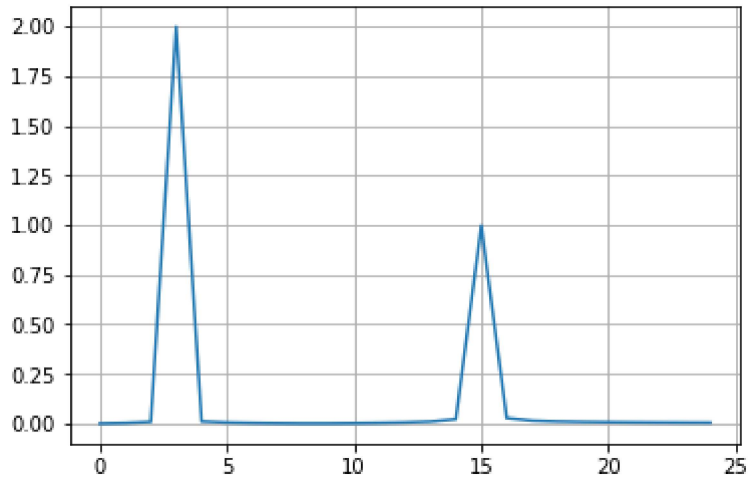
10//3=3

In [54]:
```
# Number of sample points
N = 600
# sample spacing
dt = 1.0 / 800.0
# Period of periodic function
T1=0.05
T2=0.25
# Amplitude of periodic function
A1=1
A2=2
# Periodic function
t = np.linspace(0.0, N*dt, N) # desde tMin=0 hasta tMax=N*dt en N pasos (con l
ongitud dt)
x = A1*np.sin(2.0*np.pi*t/T1) + A2*np.sin(2.0*np.pi*t/T2)
# Fourier transform
# Represented as a density (normalization factor 1/N), factor 2 to make up for
 imaginary part not shown
# Squared to transform amplitude into power P=A**2
xf = (2.0/N*np.abs(fft(x)))
# plotting
plt.clf()
plt.plot(x)
plt.xlabel('t (a.u.)')
plt.ylabel('x(t)')
plt.show()
print('length x: ',x.shape)
print('time shown in arbitrary units (a.u.)')
print('15 small and rapid oscillations superposed on 3 large and slow oscillat
ions')
print('Var(x)=',np.var(x))
print('Var(x)=',np.std(x)**2)
plt.clf()
plt.plot(xf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.show()
plt.plot(xf[0:25]) # show first 25 frequencies
plt.grid()
plt.show()
print('length xf: ',xf.shape)
print('xf=',xf[0:25])
print('frequencies represented as number of oscillations during the whole dura
tion of the time series')
print('Area of Fourier power spectrum: ',np.sum(xf[1:25]))
```




```
length x:  (600,)
time shown in arbitrary units (a.u.)
15 small and rapid oscillations superposed on 3 large and slow oscillations
Var(x)= 2.49583333333
Var(x)= 2.49583333333
```





```
length xf:  (600,)
xf= [  9.31107043e-16   2.71698028e-03   8.41571474e-03   1.99894101e+00
   1.05437412e-02   5.02745747e-03   2.87655050e-03   1.52914374e-03
   4.43489729e-04   6.02743468e-04   1.77329885e-03   3.28230380e-03
   5.55903396e-03   9.82853325e-03   2.20384046e-02   9.96739474e-01
   2.77677072e-02   1.46695383e-02   1.03169162e-02   8.11609304e-03
   6.77377929e-03   5.86181243e-03   5.19712867e-03   4.68818230e-03
   4.28401730e-03]
frequencies represented as number of oscillations during the whole duration o
f the time series
Area of Fourier power spectrum:  3.15799305045
```

Fourier transform of experimental data

In [2]:
```
xl=pd.ExcelFile("/Users/ruben/Dropbox/ESTUDIANTES/Consejos/Fermin y Sandra/201
7-10-24_sBP dBP mBP_SignificadoFisiologico/Cony.xlsx")
```

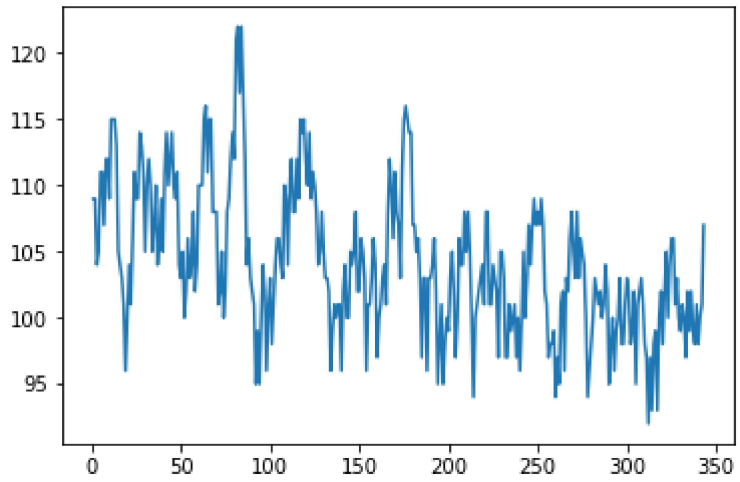
In [3]:
```
print(xl.sheet_names)
df=xl.parse("ConySupino")
```

```
['ConySupino', 'ConyDePie', 'ConyRR']
```

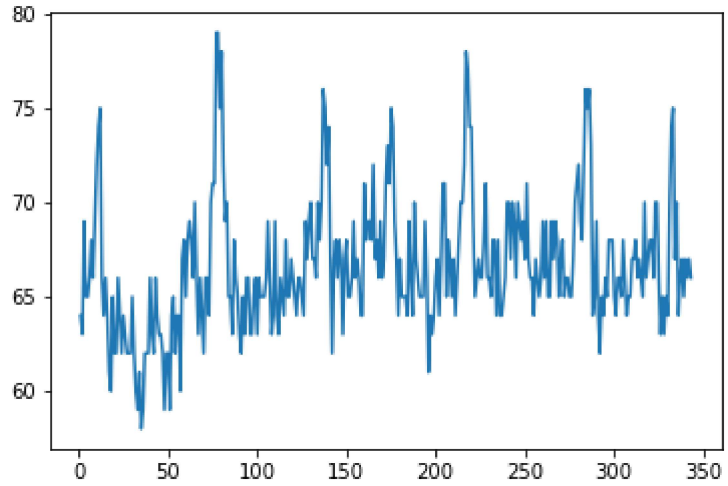
In [4]:
```
Time=df['Time'][1:df.shape[0]]
SBP=df['reSYS'][1:df.shape[0]]
DBP=df['reDIA'][1:df.shape[0]]
MBP=df['reMAP'][1:df.shape[0]]
HR=df['HR'][1:df.shape[0]]
```

In [5]:
```
N=SBP.shape[0]
print(N)
SBPf=(2.0/N*np.abs(fft(SBP)))
plt.clf()
plt.plot(SBP)
plt.show()

N=HR.shape[0]
print(N)
HRf=(2.0/N*np.abs(fft(HR)))
plt.clf()
plt.plot(HR)
plt.show()
```

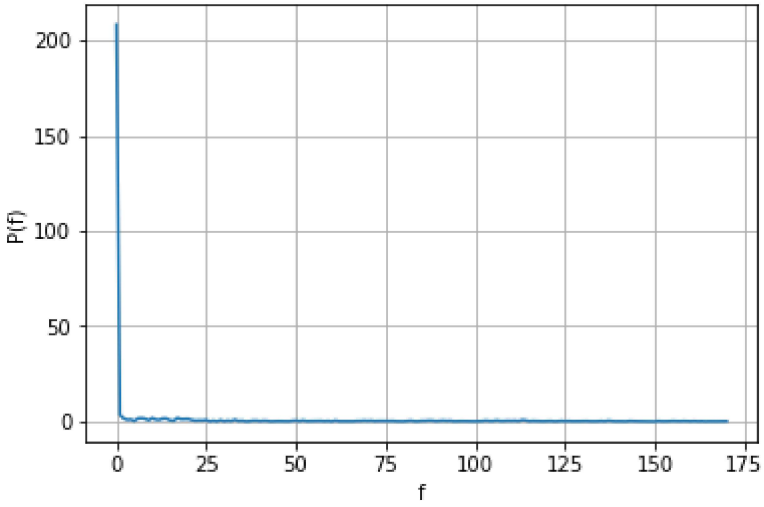
```
343
```

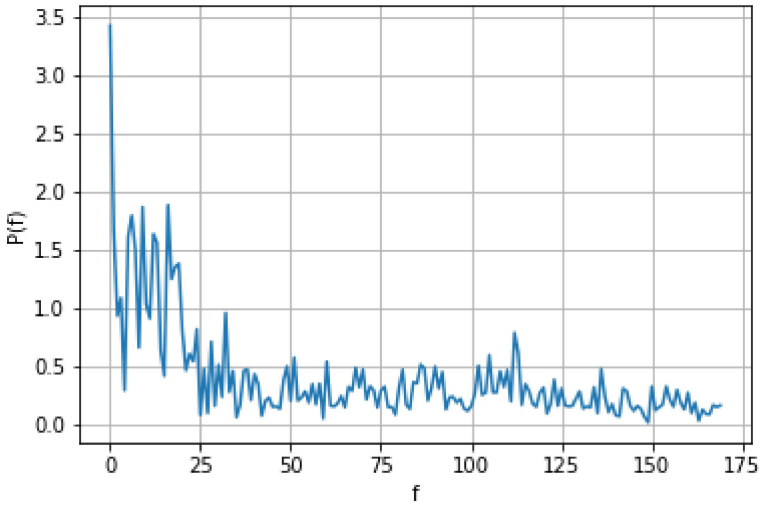


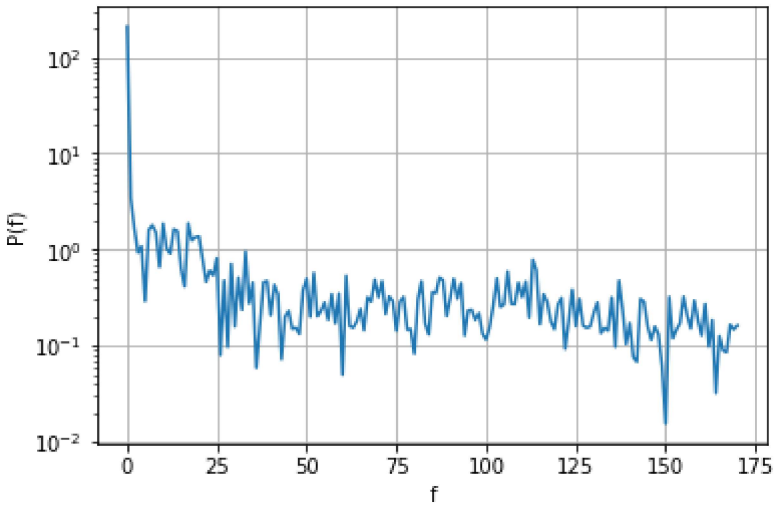

```
343
```

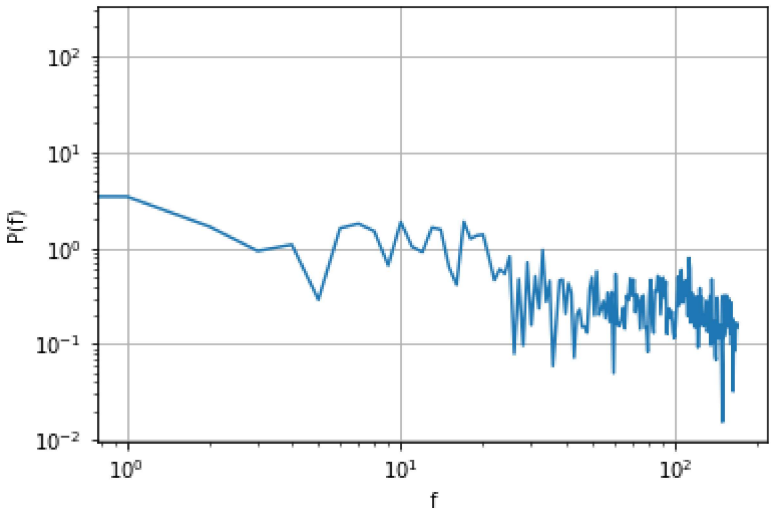


In [15]:
```
plt.clf()
plt.plot(SBPf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.show()

plt.clf()
plt.plot(SBPf[1:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.show()

plt.clf()
plt.plot(SBPf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.yscale('log')
plt.show()

plt.clf()
plt.plot(SBPf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.xscale('log')
plt.yscale('log')
plt.show()
```

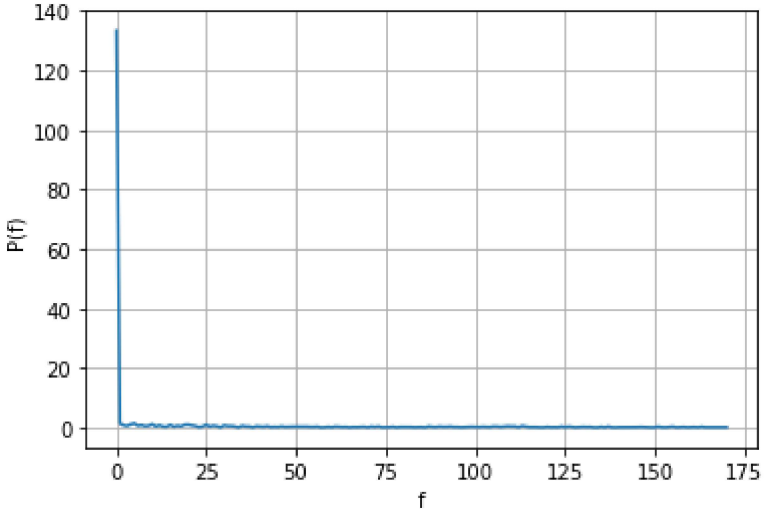

P(f)
f
0
50
100
150
200
0
25
50
75
100
125
150
175

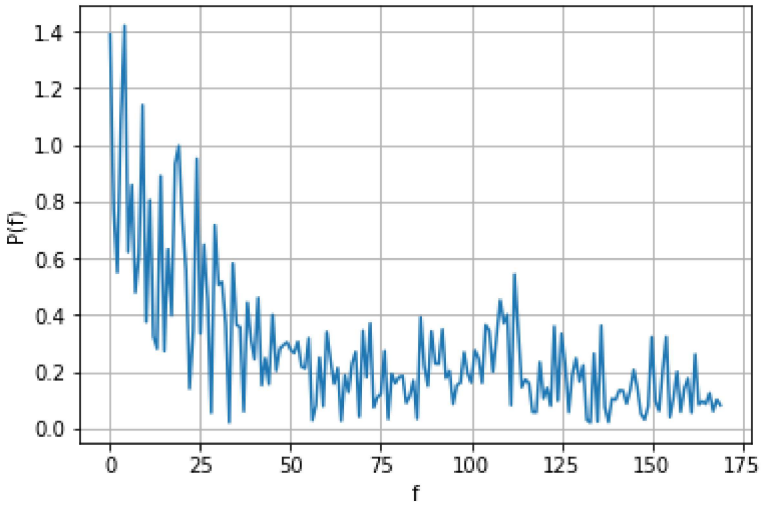

P(f)
f
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
25
50
75
100
125
150
175

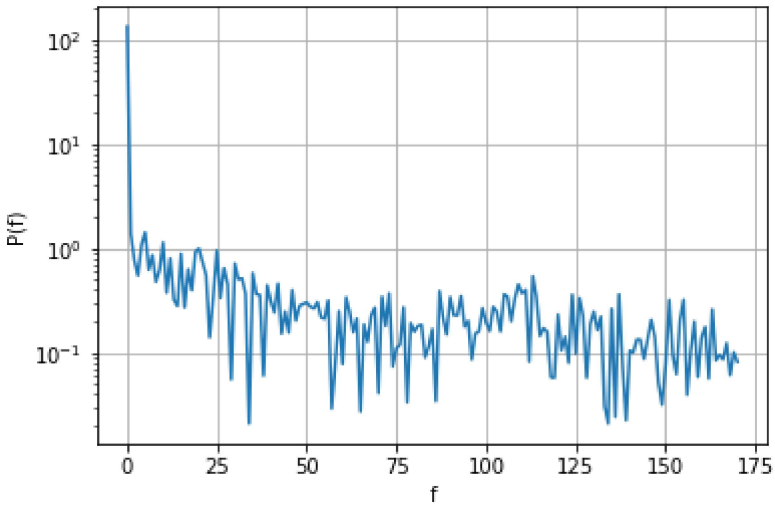

P(f)
f
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
0
25
50
75
100
125
150
175

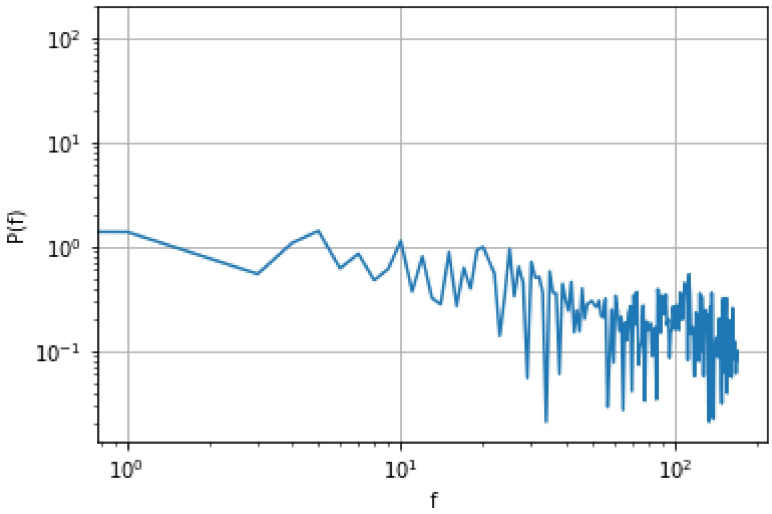

10^2
10^1
10^0
10^{-1}
10^{-2}
P(f)
10^0
10^1
10^2
f

In [19]:
```
plt.clf()
plt.plot(HRf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.show()

plt.clf()
plt.plot(HRf[1:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.show()

plt.clf()
plt.plot(HRf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.yscale('log')
plt.show()

plt.clf()
plt.plot(HRf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f')
plt.ylabel('P(f)')
plt.xscale('log')
plt.yscale('log')
plt.show()
```


P(f)
f
0
20
40
60
80
100
120
140
0
25
50
75
100
125
150
175


P(f)
f
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
25
50
75
100
125
150
175


P(f)
f
10^{2}
10^{1}
10^{0}
10^{-1}
0
25
50
75
100
125
150
175



Interpolation of mathematical model

Interpolation function

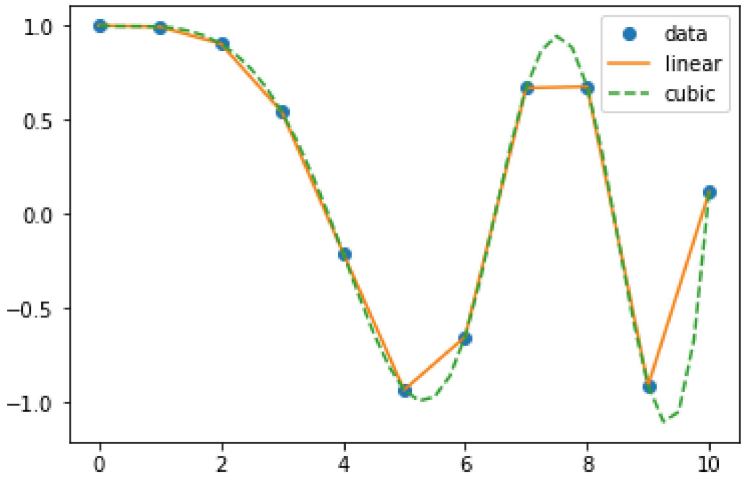
- only interpolation (no extrapolation)
- can be linear or quadratic
- no deviations, restrained

https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html
(https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html)

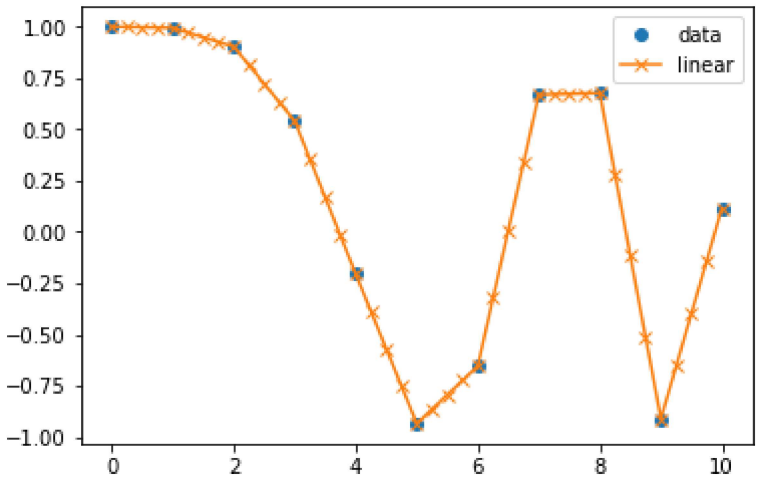
In [12]:
```
from scipy.interpolate import interp1d
```

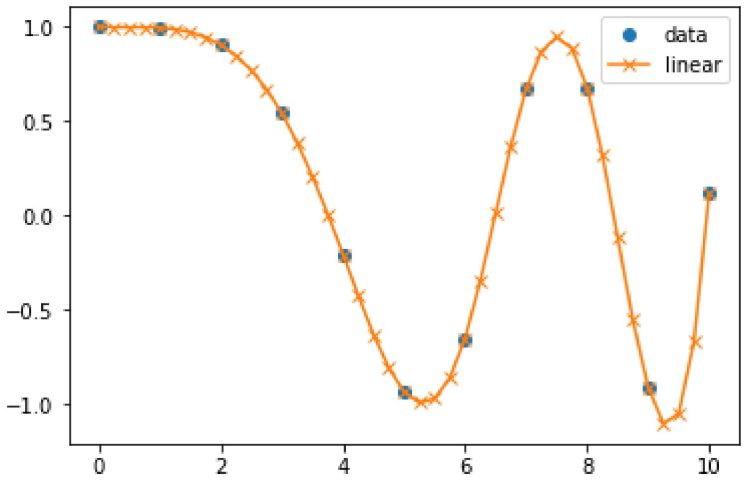
In [11]:
```
x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x**2/9.0)
f1 = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
```

In [8]:
```
xnew = np.linspace(0, 10, num=41, endpoint=True)
plt.plot(x, y, 'o', xnew, f1(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```



In [12]:
```
plt.clf()
plt.plot(x, y, 'o', xnew, f1(xnew), '-x')
plt.legend(['data', 'linear'], loc='best')
plt.show()

plt.clf()
plt.plot(x, y, 'o', xnew, f2(xnew), '-x')
plt.legend(['data', 'cubic'], loc='best')
plt.show()
```





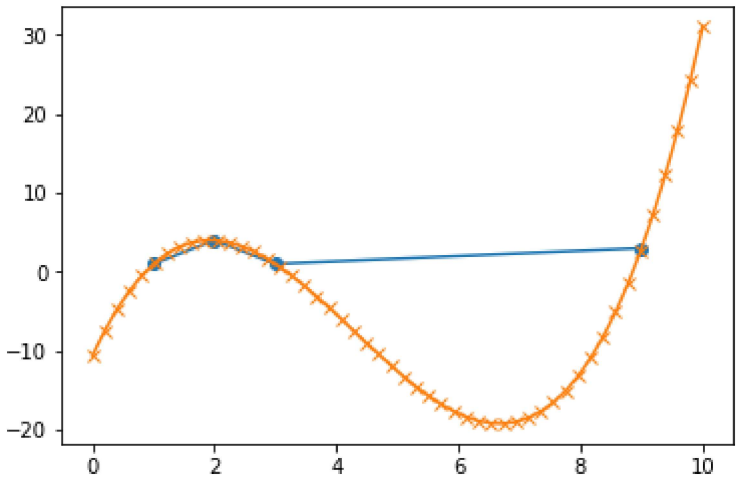
Polynomial fit

- Possibilities to interpolate and extrapolate
- Large deviations/oscillations possible

https://plot.ly/python/interpolation-and-extrapolation-in-1d/ (https://plot.ly/python/interpolation-and-extrapolation-in-1d/)

In [25]:
```
points = np.array([(1, 1), (2, 4), (3, 1), (9, 3)])

x = points[:,0]
y = points[:,1]

z = np.polyfit(x, y, 3)
f = np.poly1d(z)

x_new = np.linspace(0, 10, 50)
y_new = f(x_new)
```

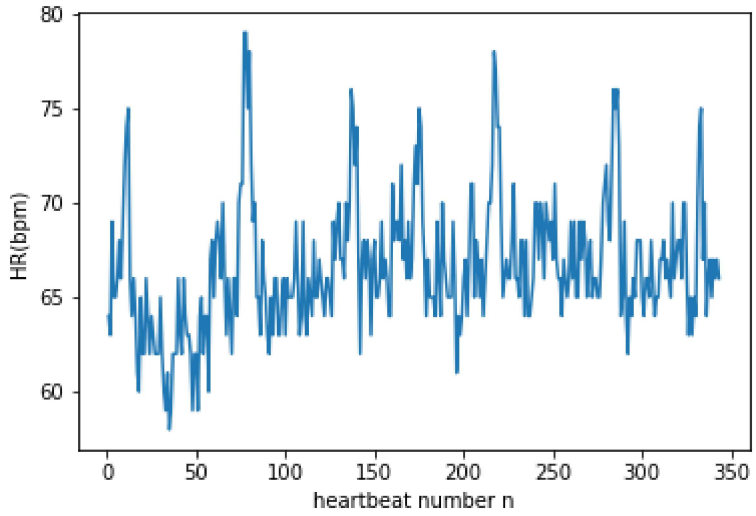
In [26]:
```
plt.clf()
plt.plot(x,y,'-o',x_new,y_new,'-x')
plt.show()
```

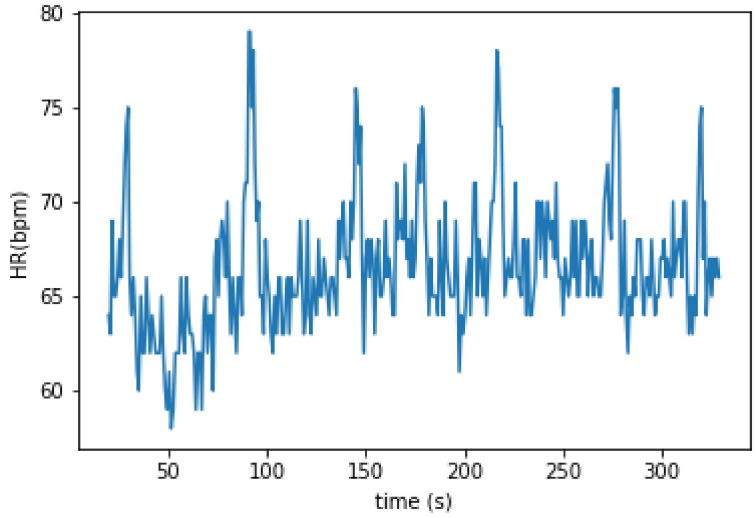


Interpolation of hear rate data

In [7]:
```
print(HR.shape)
plt.clf()
plt.plot(HR)
plt.xlabel('heartbeat number n')
plt.ylabel('HR(bpm)')
plt.show()

plt.clf()
plt.plot(Time,HR)
plt.xlabel('time (s)')
plt.ylabel('HR(bpm)')
plt.show()
```

```
(343,)
```





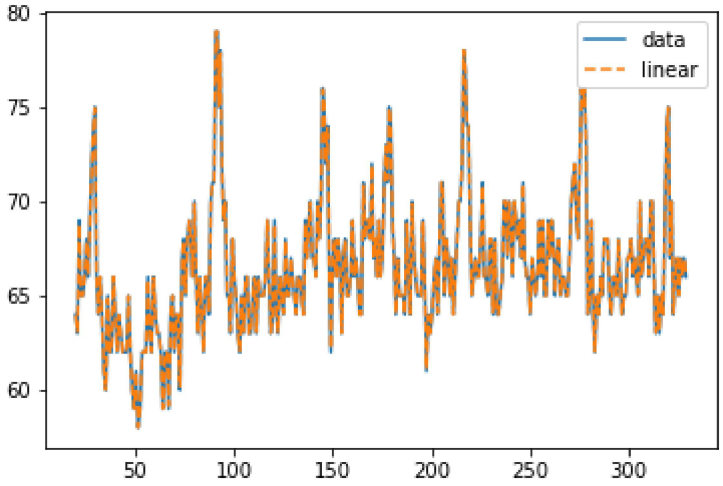
In [8]:
```
tMin=np.min(Time)
tMax=np.max(Time)
t_new=np.arange(20.1,328.8,0.1)
print(tMin)
print(tMax)
print(t_new)
```

```
20.047
328.729
[  20.1   20.2   20.3 ...,  328.5  328.6  328.7]
```

In [15]:
```
f1 = interp1d(Time, HR)
HR_new=f1(t_new)
print(HR_new)
```

```
[63.94355697550586 63.8370607028754 63.73056443024494 ...,
 66.25501113585251 66.1436525612423 66.03229398663203]
```

In [16]:
```
plt.clf()
plt.plot(Time,HR,'-',t_new, HR_new,'--')
plt.legend(['data', 'linear'], loc='best')
plt.show()
```



In [17]:
```
print(HR.shape)
print(HR_new.shape)
```

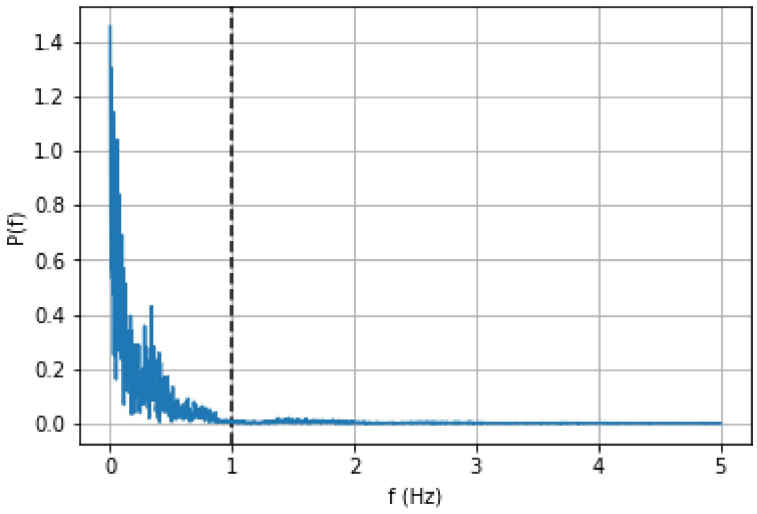
```
(343,)
(3087,)
```

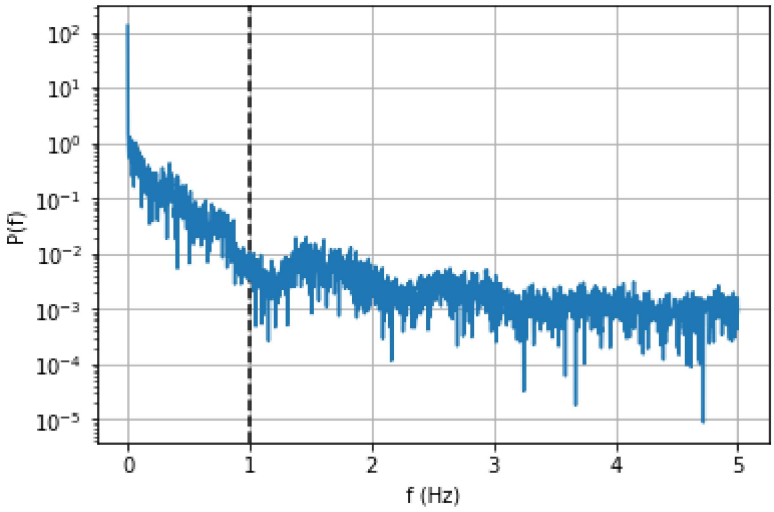
In [18]:
```
N=HR_new.shape[0]
HRf=(2.0/N*np.abs(fft(HR_new)))
```

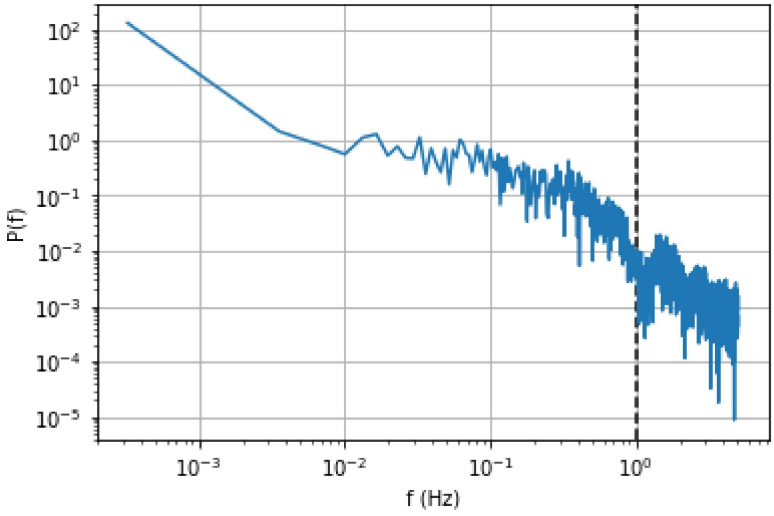
In [19]:
```
fMin=1/HR_new.shape[0] # units [Hz] , slowest oscillation is once in the whole
 duration of the time series
fMax=1/(2*0.1) # units [Hz], most rapid oscillation is once every 2 datapoints

#fMin=60/HR_new.shape[0] # units [bpm], 60 bpm = 1Hz
#fMax=60/(2*0.1) # units [bpm], 60 bpm = 1Hz
f_new=np.linspace(fMin,fMax,N//2)
print(f_new.shape)
print(HRf[0:N//2].shape)
print(fMin)
print(fMax)
print(f_new)
```

```
(1543,)
(1543,)
0.00032393909944930353
5.0
[  3.23939099e-04   3.56627118e-03   6.80860325e-03 ...,   4.99351534e+00
   4.99675767e+00   5.00000000e+00]
```

In [21]:
```
### linlin ###

plt.clf()

xposition = [1]
for xc in xposition:
    plt.axvline(x=xc, color='k', linestyle='--')


plt.plot(f_new[1:N//2],HRf[1:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f (Hz)')
plt.ylabel('P(f)')
plt.show()

#### linlog ###

plt.clf()

xposition = [1]
for xc in xposition:
    plt.axvline(x=xc, color='k', linestyle='--')

plt.plot(f_new,HRf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f (Hz)')
plt.ylabel('P(f)')
plt.yscale('log')
plt.show()

### loglog ###

plt.clf()

xposition = [1]
for xc in xposition:
    plt.axvline(x=xc, color='k', linestyle='--')

plt.plot(f_new,HRf[0:N//2]) # show all frequencies
plt.grid()
plt.xlabel('f (Hz)')
plt.ylabel('P(f)')
plt.yscale('log')
plt.xscale('log')
plt.show()
```







In [ ]: