timeseries_correlations

June 4, 2024

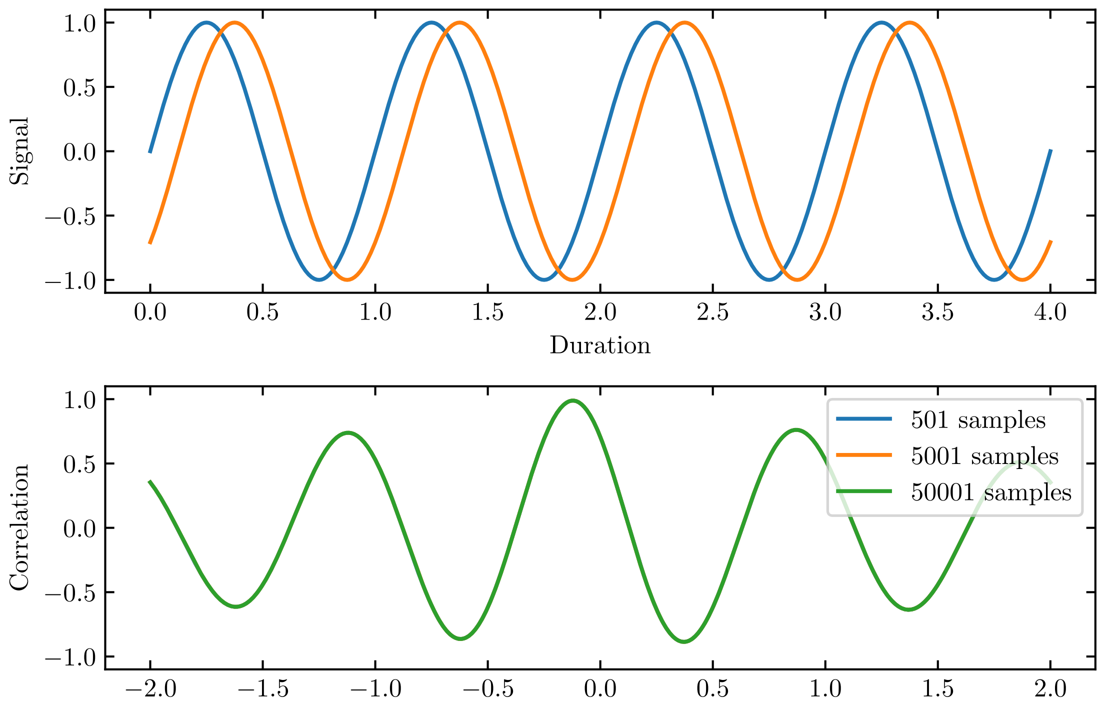
[1]:
```python
import numpy as np
import scipy.signal
import matplotlib.pyplot as plt

from matplotlib.dates import DateFormatter, MonthLocator, DayLocator
from matplotlib.ticker import MultipleLocator
from datetime import datetime as dt, timedelta as td

# from IPython.display import set_matplotlib_formats

%matplotlib inline
import matplotlib_inline
# set_matplotlib_formats('pdf')
matplotlib_inline.backend_inline.set_matplotlib_formats('pdf')

plt.rcParams['figure.dpi']  = 150
plt.rcParams['savefig.dpi'] = 150

# Definition of functions for testing
def sinusoid(x, offset, amplitude, frequency, phase):
    return offset + amplitude * np.sin(2 * np.pi * frequency * x - phase)

def gaussian(x, loc, scale):
    return np.exp( -((x - loc) / scale)**2 )

def square(x, start, stop):
    return np.where(np.logical_and(x > start, x <= stop), 1, 0)

# Simplifies processing of natural data
def detrend_normalise_data(data):
    d = scipy.signal.detrend(data)
    return d / np.linalg.norm(d)
```

1 Tests of understandability

Here, we run some simple tests to help understand the correlation functions and lags

[2]:
```
# Parameters for the sinusoids: offset, amplitude, "understandable" frequency␣
↪and phase
p0 = [0, 1, 1, 0]
p1 = [0, 1, 1, np.pi / 4]  # Play around with the phase.

n_samples = (501, 5001, 50_001)  # I get the same result regardless of number␣
↪of samples

fig, axes = plt.subplots(nrows=2)

for n_samples in n_samples:
    x  = np.linspace(0, 4, n_samples)
    dx = x[1] - x[0]       # sampling interval
    fs = 1 / dx            # sampling frequcency
    y0 = sinusoid(x, *p0)
    y1 = sinusoid(x, *p1)

    a  = y0 / np.linalg.norm(y0)
    b  = y1 / np.linalg.norm(y1)

    corr = scipy.signal.correlate(a, b, 'same')
    lags = scipy.signal.correlation_lags(len(a), len(b), 'same') * dx  # lags␣
↪multiplied by sampling interval

    axes[1].plot(lags, corr, label=f'{n_samples} samples')
    axes[1].set_ylim([-1.1, 1.1])

axes[0].plot(x, y0, '-C0', x, y1, '-C1')
axes[0].set_xlabel('Duration')
axes[0].set_ylabel('Signal')
axes[1].set_ylabel('Correlation')

print(f"Signal y1 lags signal y0 by {-lags[np.argmax(corr)]}")
axes[1].legend(loc=1)
fig.tight_layout()
plt.show()
```

```
Signal y1 lags signal y0 by 0.12184
```

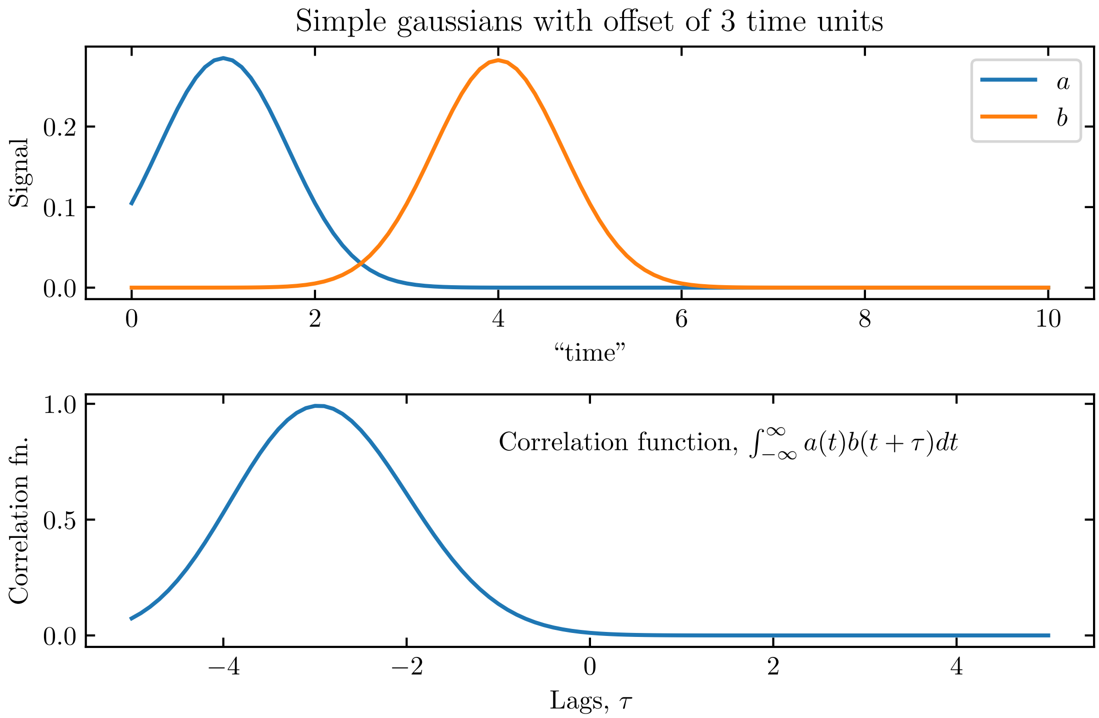


The lag with a phase of $\pi/4$ (i.e. $\frac{\pi/4}{2\pi} = 1/8$ cycles) is almost (but not quite!) 1 / 8.

[3]:
```
t = np.linspace(0, 10, 101)
dt = t[1] - t[0]
a, b = gaussian(t, 1, 1), gaussian(t, 4, 1)
a /= np.linalg.norm(a)
b /= np.linalg.norm(b)

fig, axes = plt.subplots(nrows=2)
axes[0].plot(t, a, '-', label='$a$')
axes[0].plot(t, b, '-', label='$b$')
axes[0].legend()
axes[0].set_title(f'Simple gaussians with offset of 3 time units')
axes[0].set_xlabel("``time''")
axes[0].set_ylabel('Signal')

corr = scipy.signal.correlate(a, b, 'same')
lags = scipy.signal.correlation_lags(len(a), len(b), 'same') * dt
axes[1].plot(lags, corr)
axes[1].annotate(r'Correlation function, $\int_{-\infty}^{\infty} a(t)b(t+\tau)␣
↪dt$', (-1, .8))
axes[1].set_xlabel(r'Lags, $\tau$')
axes[1].set_ylabel('Correlation fn.')
fig.tight_layout()
```

```
print(f'Signal $a$ comes after signal $b$ by {lags[np.argmax(corr)]}')
```

```
Signal $a$ comes after signal $b$ by -3.0
```



Signal a comes after signal b by -3.0, i.e. b lags a by 3.

2 Correlation of real data

Here, we're going to use the ground water level (GWL) data, self polarisation (SP) and sismic (count from WO-MC3). As the GWL data is produced once per day, the two other data sets have been subsampled to match this sampling frequency: for SP, the resampling is the mean of the daily value; for the sismic data, this is the sum of the daily data.

[4]:
```
GWL_data = pd.read_csv('../data/GWL_modelling/GWL_TAS_model_inversion.csv',
                       sep=',', parse_dates=['# Date (yyyy-mm-dd)'])
GWL_data = GWL_data.rename(mapper={'# Date (yyyy-mm-dd)': 'Date',
                                   ' GWL (m) ': 'GWL',
                                   ' error (m)': 'error'}, axis=1)
# print(GWL_data.head())

# Depth of Tarissan lake - for comparison with GWL
TAR_data = pd.read_csv('../data/OVSG_EAUX_2024-05-31.csv', sep=';',
 ↪parse_dates=['Date']).dropna(subset=['Niveau (m)'])
```

```
TAR_data =  TAR_data[['Date', 'Niveau (m)']]
# print(TAR_data)
```

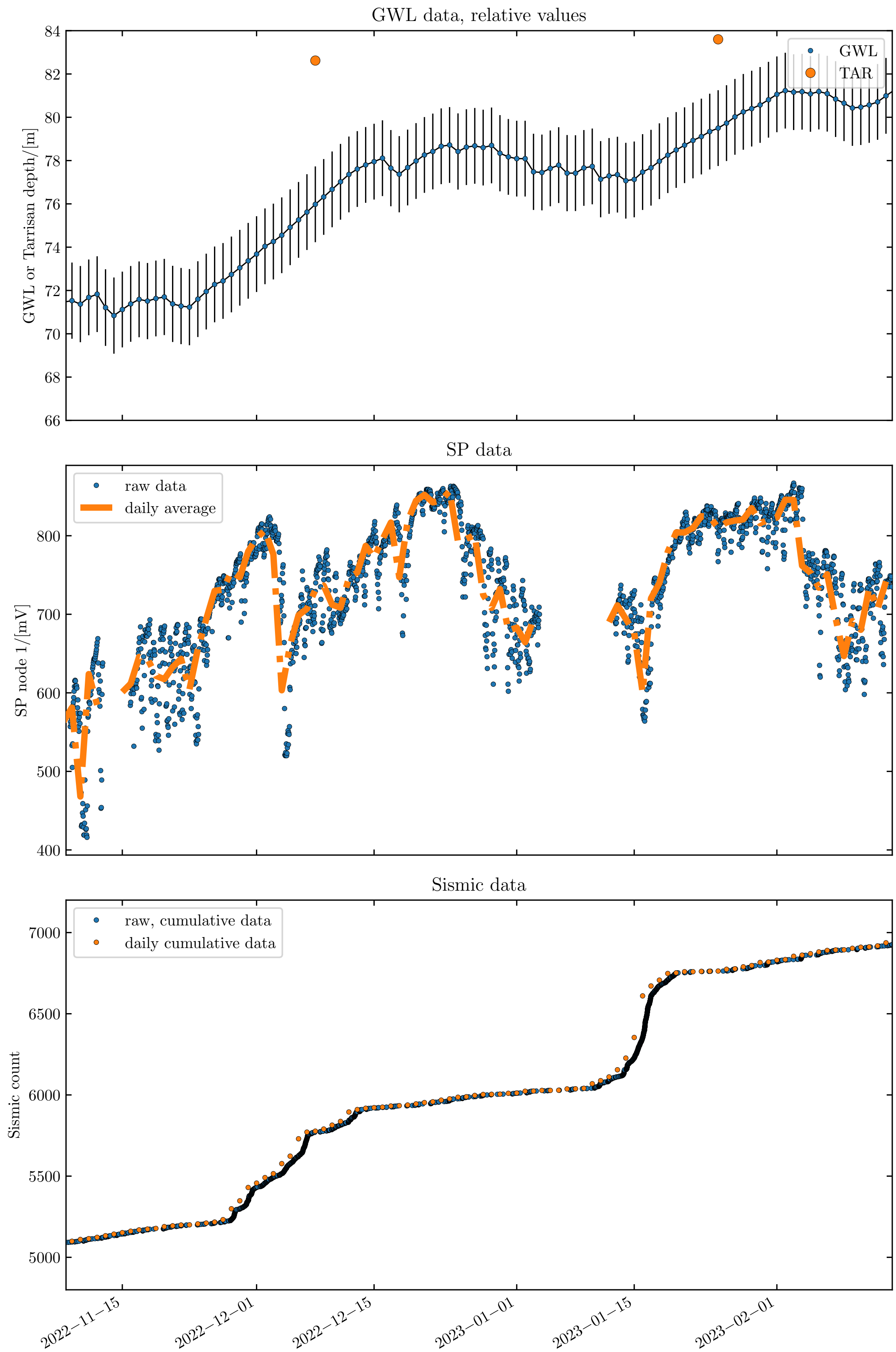
[5]:
```
SP_data  = pd.read_csv('../data/
 ↪PSGUADA_NOEUD_263_PS-data-as-joinbyfield-2024-05-06-09-51-08.csv',
                         sep=',', parse_dates=['Time'])
# print(SP_data.head())

SP_data_daily = SP_data.copy()
SP_data_daily.index = SP_data['Time']
SP_data_daily = SP_data_daily.loc[:, SP_data_daily.columns.str.
 ↪contains('Average')]
SP_data_daily = SP_data_daily.resample('1D').mean()
SP_data_daily = SP_data_daily.reset_index()

# SP_data_daily.head()
```

[6]:
```
sismo = pd.read_csv('../data/WO_OVSG_MC3_OVSG_dump_bulletin.csv',
                    sep=';', skiprows=2,
                    names=['date', 'count', 'duration', 'amplitude',
                           'magnitude', 'e(j)', 'longitude', 'latitude',
                           'depth', 'type', 'file', 'locmode', 'loctype',
                           'projection', 'operator', 'timestamp', 'id'],
                    parse_dates=['date'], date_format='%Y%m%d %H%M%S.%f')
sismo = sismo[['date', 'count', 'duration', 'amplitude',
               'longitude', 'latitude', 'depth']]
sismo['cumul'] = np.cumsum(sismo['count'])

sismo_daily = sismo.copy()
sismo_daily.index = sismo_daily['date']

sismo_daily = sismo_daily.loc[:, ~sismo_daily.columns.str.contains('date')]
sismo_daily = sismo_daily.resample('1D').sum().reset_index()
sismo_daily['cumul'] = np.cumsum(sismo_daily['count'])
```

2.1 Plots of raw data.

[7]:
```
fig, axes = plt.subplots(nrows=3, sharex=True, figsize=(8, 12))
axes[0].plot_date(GWL_data['Date'], GWL_data['GWL'], '.', zorder=3, label='GWL')
axes[0].errorbar(GWL_data['Date'], GWL_data['GWL'], yerr=GWL_data['error'],
                 marker='.', c='k', lw=.8, zorder=1)
axes[0].set_title('GWL data, relative values')
axes[0].set_ylabel('GWL or Tarrisan depth/[m]')
axes[0].set_ylim([66, 84])
axes[0].plot_date(TAR_data['Date'], -TAR_data['Niveau (m)'], 'o', label='TAR')

axes[1].plot_date(SP_data['Time'],
```

```
                  SP_data['Average json.data-tension-ps1'],
                  '.', zorder=3, label='raw data')
axes[1].plot_date(SP_data_daily['Time'],
                  SP_data_daily['Average json.data-tension-ps1'],
                  '-.', lw=4, zorder=3, label='daily average')
axes[1].set_xlim([SP_data['Time'].min(), SP_data['Time'].max()])
axes[1].set_ylabel('SP node 1/[mV]')
axes[1].set_title('SP data')

axes[2].plot_date(sismo['date'], sismo['cumul'], '.',
                  label='raw, cumulative data')
axes[2].plot_date(sismo_daily['date'], sismo_daily['cumul'],
                  '.', lw=2, label='daily cumulative data')
axes[2].set_ylim([4800, 7200])
axes[2].set_title('Sismic data')
axes[2].set_ylabel('Sismic count')
for ax in axes: ax.legend()

fig.autofmt_xdate()
fig.tight_layout()
```

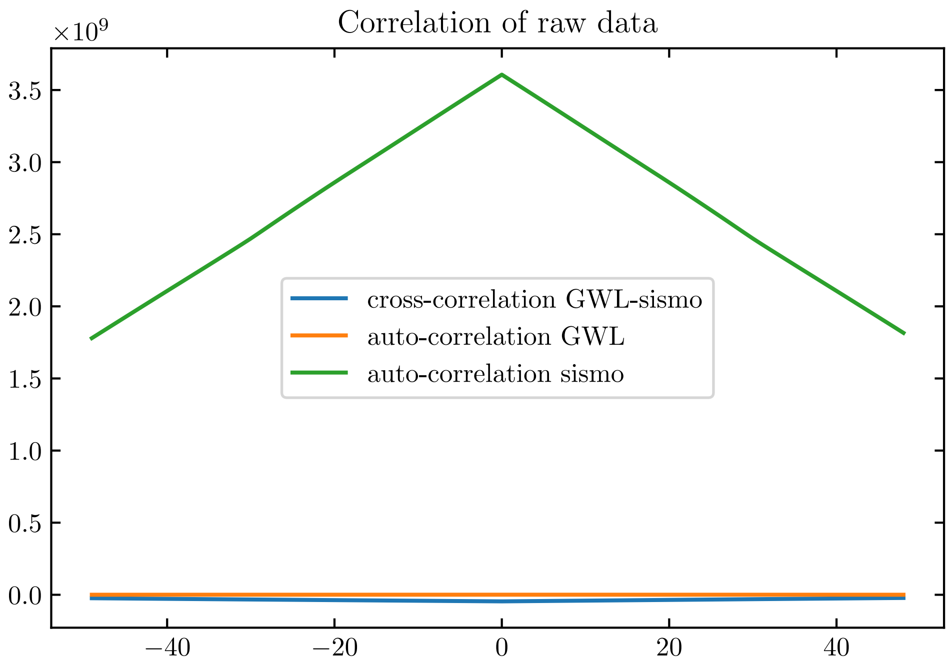

GWL data, relative values
GWL
TAR
GWL or Tarrisan depth/[m]
84
82
80
78
76
74
72
70
68
66
SP data
raw data
daily average
SP node 1/[mV]
800
700
600
500
400
Sismic data
raw, cumulative data
daily cumulative data
Sismic count
7000
6500
6000
5500
5000
2022−11−15
2022−12−01
2022−12−15
2023−01−01
2023−01−15
2023−02−01

The depth of the Tarissan lake is equivalent to the level below a reference point. As the lowest depths (i.e. largest magnitude of the -vely signed data) correspond to the highest values of the GWL data. Hence we take the -ve of the GWL signals in the following analyses

[8]:
```
## simplified variables for ease of manipulation
x0 = GWL_data['Date']
y0 = GWL_data['GWL']
x0 = x0[np.logical_and(x0 >= SP_data['Time'].min(),
                       x0 <= SP_data['Time'].max())]
y0 = y0[x0.index]

x1 = SP_data_daily['Time']
x1 = x1[np.logical_and(x1 > SP_data['Time'].min(),
                       x1 <= SP_data['Time'].max())]

xs = sismo_daily['date']
ys = np.cumsum(sismo_daily['count'])
ys = ys[np.logical_and(xs >= SP_data['Time'].min(),
                       xs <= SP_data['Time'].max())]
xs = xs[np.logical_and(xs >= SP_data['Time'].min(),
                       xs <= SP_data['Time'].max())]
# ys = detrend_normalise_data(ys)
```

2.2 Test of correlation of the raw data

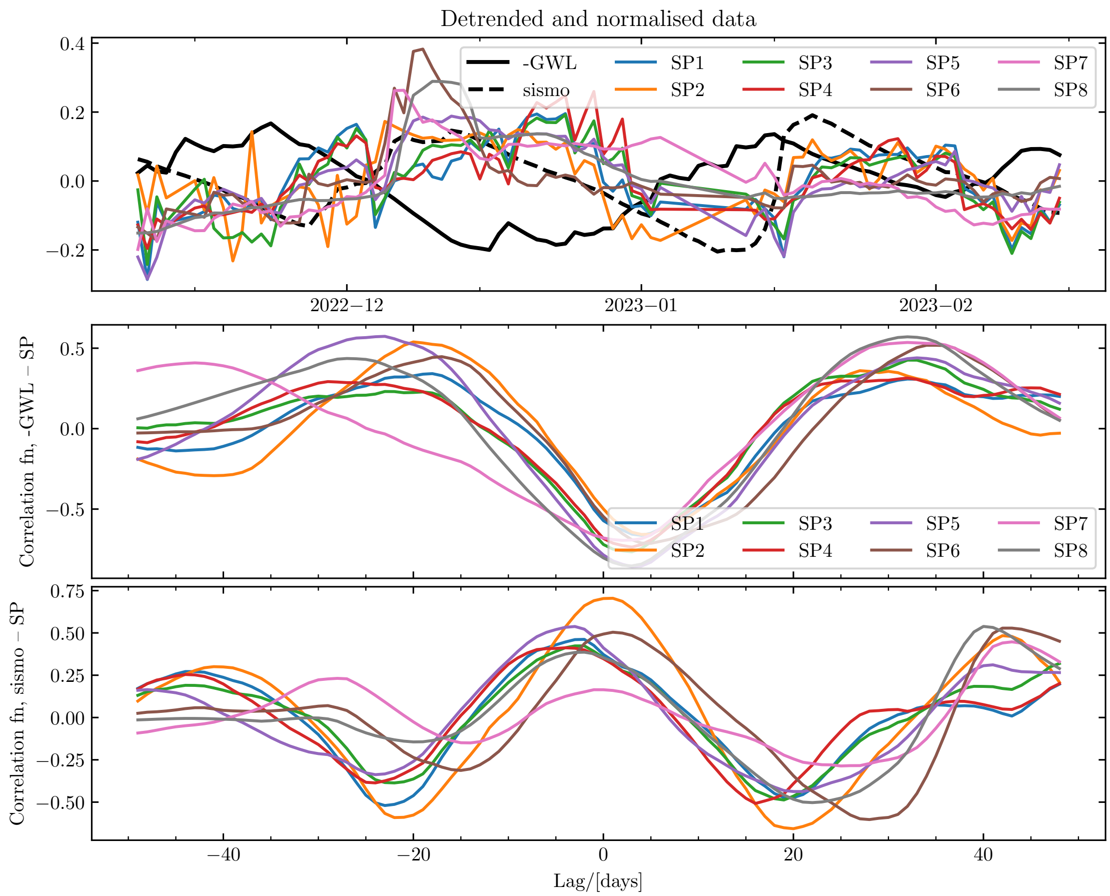
[9]:
```
corr = scipy.signal.correlate(-y0, ys, 'same')
lags = scipy.signal.correlation_lags(len(y0), len(ys), 'same')
plt.plot(lags, corr, '-', label='cross-correlation GWL-sismo')

auto_corr = scipy.signal.correlate(-y0, -y0, 'same')
auto_lags = scipy.signal.correlation_lags(len(y0), len(y0), 'same')
plt.plot(auto_lags, auto_corr, '-', label='auto-correlation GWL')

auto_corr = scipy.signal.correlate(ys, ys, 'same')
auto_lags = scipy.signal.correlation_lags(len(ys), len(ys), 'same')
plt.plot(auto_lags, auto_corr, '-', label='auto-correlation sismo')

plt.legend()
plt.title('Correlation of raw data');
```



The correlation functions are dominated by the long-period signals and, in particular, by the sismic data which is large in magnitude. We will detrend the data to remove the long-period information. We will also normalise the data (by its vector norm) to scale the correlation function to the interval [-1, 1].

[10]:
```
fig = plt.figure(figsize=(8, 6))

ax0 = fig.add_axes([.05, .75, .9, .3])
ax1 = fig.add_axes([.05, .41, .9, .3])
ax2 = fig.add_axes([.05, .10, .9, .3], sharex=ax1)


for observable, indep_var, linesty, label, ax in zip((-y0, ys), (x0, xs), ('-',
    '--'), ('-GWL', 'sismo'), (ax1, ax2)):
    a  = detrend_normalise_data(observable)
    ax0.plot_date(indep_var, a, f'{linesty}k', lw=2, label=label)
    # ax0.plot_date(xs, # detrend_normalise_data(np.
    cumsum(sismo_daily['count'][xs.index])),
                    # '--k', lw=3, label='cumul VT')
    for ind in range(1, 9):
        y1 = SP_data_daily[f'Average json.data-tension-ps{ind}']
        y1 = y1[x1.index].interpolate('linear')

        b  = detrend_normalise_data(y1)
```

```
        if label == 'sismo':
            ax0.plot_date(x1, b, f'-C{ind-1}', label=f'SP{ind}')

        corr = scipy.signal.correlate(a, b, 'same')
        lags = scipy.signal.correlation_lags(len(a), len(b), 'same')

        ax.plot(lags, corr, f'-C{ind-1}', label=f'SP{ind}')

        # print(f"SP{ind} signal lags {label} signal by {lags[np.
↪argmax(corr)]}")
        # if ind == 8:
        #     print()

ax0.legend(ncols=5)
ax1.legend(ncols=4, loc=4)

ax0.xaxis.set_major_locator(MonthLocator())
ax0.xaxis.set_minor_locator(DayLocator(15))
ax0.xaxis.set_major_formatter(DateFormatter('%Y-%m'))
ax0.set_title('Detrended and normalised data')

ax1.set_ylabel(r'Correlation fn, -GWL -- SP')
ax1.xaxis.set_tick_params(labelbottom=False)
ax1.xaxis.set_minor_locator(MultipleLocator(5))

ax2.set_ylabel(r'Correlation fn, sismo -- SP')
ax2.set_xlabel('Lag/[days]')

# ax0.plot_date(x0, ((y0 - y0.mean()) / (y0.max() - y0.min())), '-.')

plt.show()
```



2.3 What are the lags for the all peaks and troughs of the GWL-SP correlations?

[11]:
```
a = detrend_normalise_data(-y0)
for ind in range(1, 9):
    y1 = SP_data_daily[f'Average json.data-tension-ps{ind}']
    y1 = y1[x1.index].interpolate('linear')

    b  = detrend_normalise_data(y1)

    corr = scipy.signal.correlate(a, b, 'same')
    lags = scipy.signal.correlation_lags(len(a), len(b), 'same')

    pos_lags = lags[lags > 0]
    pos_corr = corr[lags > 0]

    peaks_troughs = scipy.signal.find_peaks(abs(corr), prominence=.04)[0]

    # print(f'SP{ind} : {pos_lags[np.argmin(pos_corr)]}')
```

```
    print(f'SP{ind} : {lags[peaks_troughs]}')

# sorted(np.append(lags[peaks_troughs], lags[scipy.signal.find_peaks(-corr,␣
 ↪prominence=.05)[0]]))
```

```
SP1 : [-18   4  32]
SP2 : [-41 -20   4  27]
SP3 : [-23   3  32]
SP4 : [-29   3  32  45]
SP5 : [-23   3  33]
SP6 : [-17   5  35]
SP7 : [-43   2  32]
SP8 : [-27   3  32]
```

2.3.1 Table: lag of peaks/troughs seen in -GWL–SP correlation functions

Peaks and troughs found using the scipy.signal.find_peaks function, requiring a prominence of .04

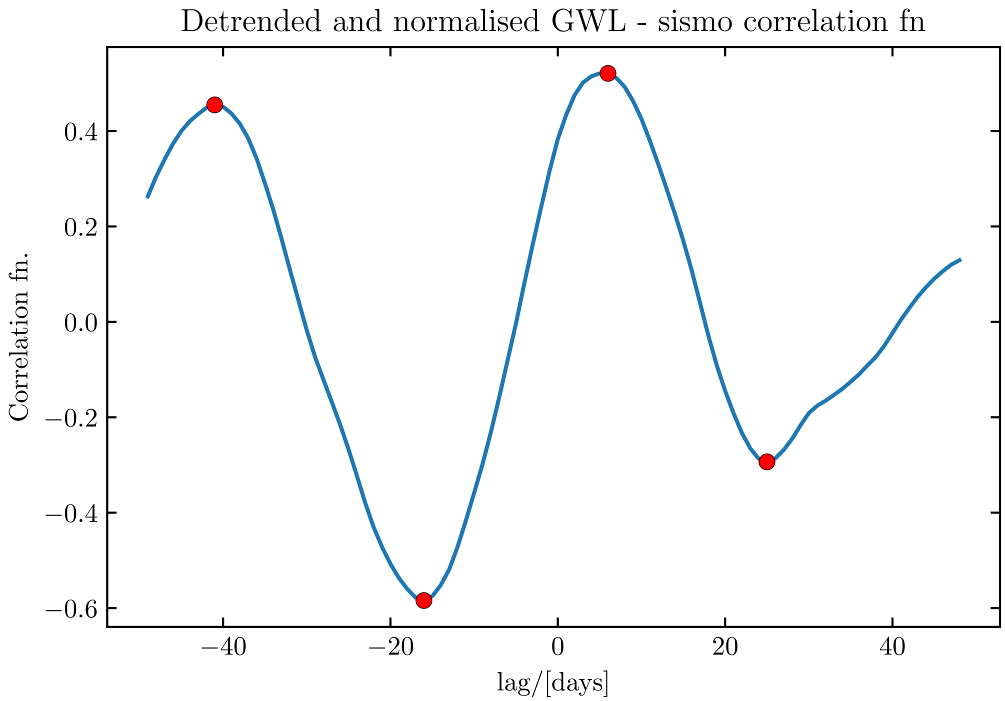
	1	2	3	4	5
SP1		-18	4	32	
SP2	-41	-20	4	27	
SP3		-23	3	32	
SP4		-29	3	32	45
SP5		-23	3	33	
SP6		-17	5	35	
SP7	-43		2	32	
SP8		-27	3	32	

2.4 Correlation of GWL and sismic data

[12]:
```
a = detrend_normalise_data(y0)  # GWL
# a /= np.linalg.norm(a)
b = detrend_normalise_data(ys)  # sismo
# b /= np.linalg.norm(b)

corr = scipy.signal.correlate(a, b, 'same')
lags = scipy.signal.correlation_lags(len(a), len(b), 'same')

plt.plot(lags, corr, '-')
plt.xlabel('lag/[days]')
plt.ylabel('Correlation fn.')
plt.title('Detrended and normalised GWL - sismo correlation fn');
# print(f'{lags[np.argmax(corr)]}')

peaks_troughs = scipy.signal.find_peaks(abs(corr), prominence=.1)[0]
plt.plot(lags[peaks_troughs], corr[peaks_troughs], 'ro')
```

```
print(f"Peaks and troughs at: {lags[peaks_troughs]} days, with max at {lags[np.
↪argmax(corr)]} days")
```

```
Peaks and troughs at: [-41 -16   6  25] days, with max at 6 days
```

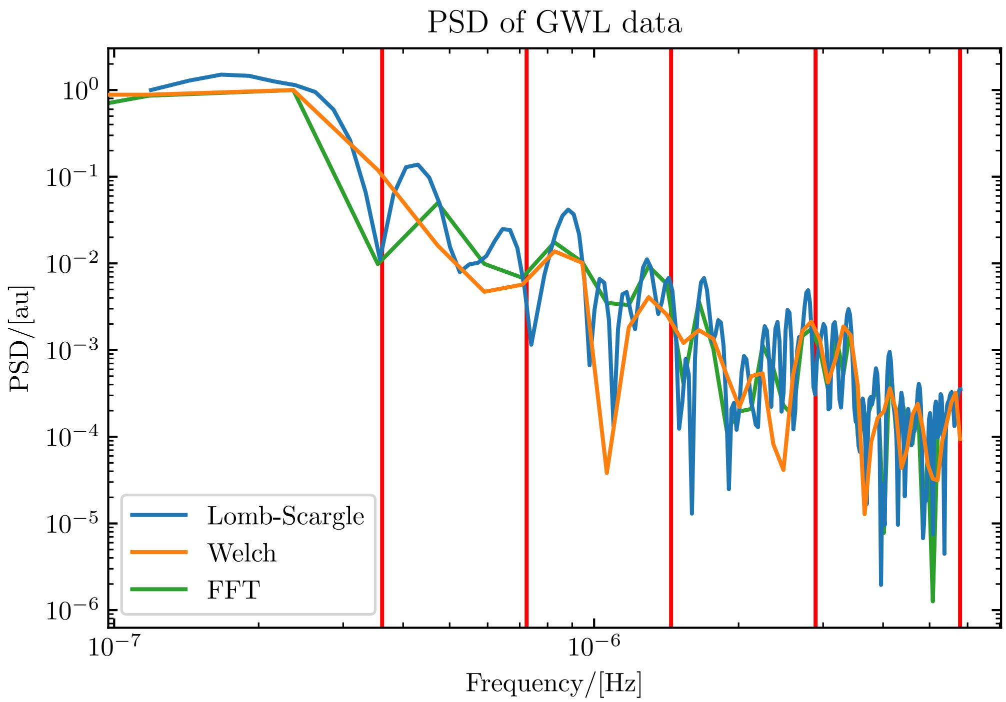


[13]:
```
from astropy.timeseries import LombScargle
from scipy.fft import fft, ifft, fftshift, ifftshift, fftfreq

y_ = detrend_normalise_data(y0)
x_ = (x0 - x0.iloc[0]).dt.total_seconds().values
dx = x_[1] - x_[0]
fs = 1 / dx
period = x_[-1] - x_[0]
fp = 1 / period
ls = LombScargle(x_, y_)
freq, power = ls.autopower(minimum_frequency=fp, maximum_frequency=fs/2)

f_w, p_w = scipy.signal.periodogram(y_, fs, window='hann')

days = [2, 4, 8, 16, 32]

for day in days:
    f_day = 1 / (day * 24 * 3600)
    plt.axvline(f_day, linestyle='-', c='r', zorder=1)
```

```
plt.loglog(freq, power / power[0], '-', zorder=3, label='Lomb-Scargle')
plt.loglog(f_w, p_w / p_w.max(), '-', zorder=3, label="Welch")


# Do it the hard way - by FFT!
y_fft = fftshift(fft(y_))
freqs = fftshift(fftfreq(y0.size, d=dx))
# print(freqs, y_fft * np.conj(y_fft))

y_fft2 = y_fft[freqs >= 0]
freqs2 = freqs[freqs >= 0]
psd    = y_fft * np.conj(y_fft)
psd2   = y_fft2 * np.conj(y_fft2)
plt.loglog(freqs2, (psd2 - psd2.min()) / (psd2.max() - psd2.min()),
           '-', label='FFT')

plt.legend()
plt.xlabel('Frequency/[Hz]');
plt.ylabel('PSD/[au]');
plt.title('PSD of GWL data')
```

```
/home/david/.local/lib/python3.10/site-packages/matplotlib/cbook.py:1699:
ComplexWarning: Casting complex values to real discards the imaginary part
  return math.isfinite(val)
/home/david/.local/lib/python3.10/site-packages/matplotlib/cbook.py:1345:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

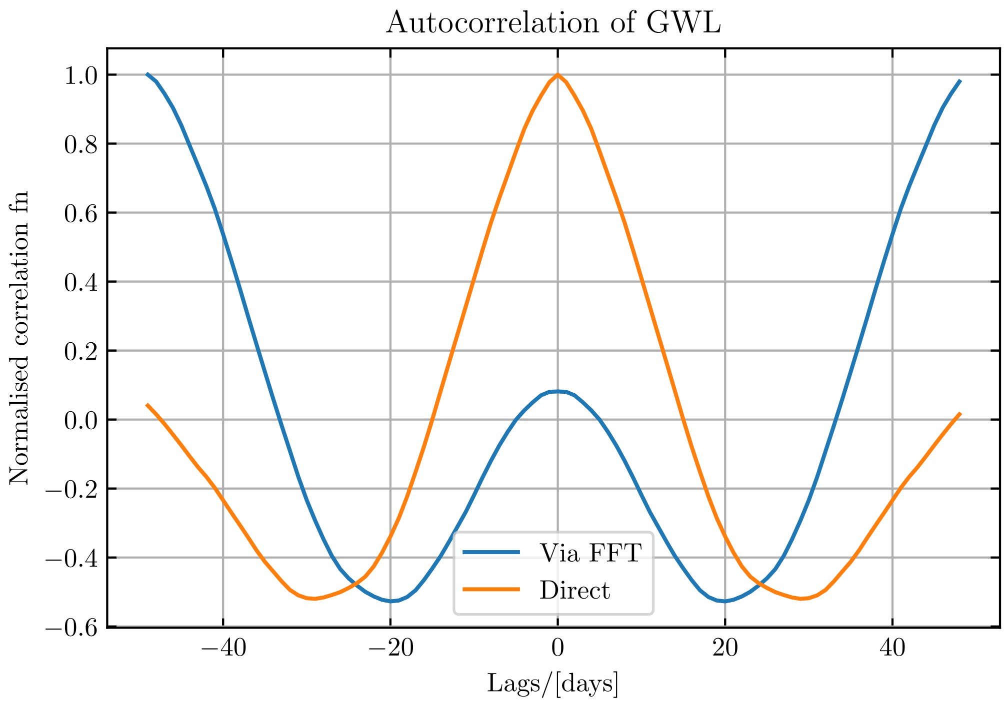
[13]:
```
Text(0.5, 1.0, 'PSD of GWL data')
```



Regardless of the method used to calculate them, the PSDs are essentially identical up to a multiplicative factor

2.5 Calculate the correlation fn from the PSDs – to be completed

[14]:
```
y_corr = ifft(ifftshift(psd))
plt.plot(lags, y_corr, label='Via FFT')
a_corr = scipy.signal.correlate(y_, y_, 'same')
plt.plot(lags, a_corr, label='Direct')
plt.legend()
plt.xlabel('Lags/[days]');
plt.ylabel('Normalised correlation fn');
plt.title('Autocorrelation of GWL')
plt.grid()
```



Autocorrelation of GWL
Normalised correlation fn
Lags/[days]
Via FFT
Direct
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−40
−20
0
20
40


```
[ ]:
```