SIOC 221A: HW5

Author: Victoria Boatwright

I certify that this represents my own work and that I have not worked with classmates or other individuals to complete this assignment.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import netCDF4
import datetime as dt
import matplotlib.dates as mdates
import pandas as pd
import xarray as xr
import scipy
from scipy import fft

plt.rcParams.update({'font.size': 16})
```

```
C:\Users\vboat\conda3\lib\site-packages\scipy\__init__.py:138: UserWarning: A NumPy v
ersion >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version
1.24.4)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required
for this version of "
```

In [2]:
```
# download wind records

links = ['https://dods.ndbc.noaa.gov/thredds/dodsC/oceansites/DATA/T8S110W/OS_T8S110W
         'https://dods.ndbc.noaa.gov/thredds/dodsC/oceansites/DATA/T8S110W/OS_T8S110W
         'https://dods.ndbc.noaa.gov/thredds/dodsC/oceansites/DATA/T8S110W/OS_T8S110W

url = links[0]
# read current file:
nc = netCDF4.Dataset(url)
print(nc)
print(nc['UWND'])
```

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF3_CLASSIC data model, file format DAP2):
    time_coverage_start: 2015-04-26T01:00:00Z
    time_coverage_end: 2016-03-21T15:00:00Z
    featureType: timeSeries
    format_version: 1.3
    platform_code: T8S110W
    platform_type: TAO Refresh
    site_code: T8S110W
    network: TAO
    wmo_platform_code: 32319
    update_interval: void
    title: TAO Delayed/Mixed Mode Data
    summary: Delayed/Mixed mode data from NDBC Tropical Atmosphere Ocean (TAO) Array
    source: moored surface buoy
```

```
    keywords: EARTH SCIENCE > ATMOSPHERE > ATMOSPHERIC WINDS > SURFACE WINDS, WIND SP
EED/WIND DIRECTION; EARTH SCIENCE > OCEANS > OCEAN WINDS > SURFACE WINDS; EARTH SCIEN
CE > ATMOSPHERE > ATMOSPHERIC TEMPERATURE > AIR TEMPERATURE; EARTH SCIENCE > ATMOSPHE
RE > ATMOSPHERIC WATER VAPOR > HUMIDITY; EARTH SCIENCE > CLIMATE INDICATORS > ATMOSPH
ERIC/OCEAN INDICATORS > TELECONNECTIONS > EL NINO SOUTHERN OSCILLATION (ENSO), ENSO;
    keywords_vocabulary: GCMD Science Keywords
    data_mode: D
    Conventions: CF-1.6, OceanSITES 1.3
    netcdf_version: NetCDF-4 classic
    naming_authority: NOAA/NDBC
    id: TAO_T8S110W_DM134A-20150425_D_WIND_10min
    cdm_data_type: Station
    area: Tropical Pacific Ocean
    geospatial_lat_units: degrees_north
    geospatial_lon_units: degrees_east
    geospatial_lat_min: -8.0
    geospatial_lat_max: -8.0
    geospatial_lon_min: -110.0
    geospatial_lon_max: -110.0
    geospatial_vertical_positive: up
    geospatial_vertical_units: meters
    geospatial_vertical_min: 0
    geospatial_vertical_max: 4
    citation: These data were collected and made freely available by National Data Bu
oy Center (NDBC).
    program: NOAA/NDBC/TAO
    references: http://tao.ndbc.noaa.gov
    principal_investigator: NOAA/National Data Buoy Center (NDBC)
    institution: NOAA/National Data Buoy Center (NDBC), U.S.
    institution_references: http://www.ndbc.noaa.gov
    data_assembly_center: NDBC
    publisher_name: NDBC Webmaster
    publisher_email: webmaster.ndbc@noaa.gov
    date_created: 2017-05-16T17:57:26Z
    date_modified: 2017-05-16T17:57:26Z
    processing_level: Data manually reviewed
    history: 2017-05-16T17:57:26Z data and metadata assembled, JZ.
    DODS.strlen: 1
    DODS.dimName: HEIGHT
    DODS_EXTRA.Unlimited_Dimension: TIME
    dimensions(sizes): TIME(47605), HEIGHT(1), LATITUDE(1), LONGITUDE(1), maxStrlen64
(64)
    variables(dimensions): float64 TIME(TIME), float32 HEIGHT(HEIGHT), float32 LATITU
DE(LATITUDE), float32 LONGITUDE(LONGITUDE), |S1 WDIR_DM(TIME, maxStrlen64), |S1 WSPD_
DM(TIME, maxStrlen64), float32 WDIR(TIME, HEIGHT), int8 WDIR_QC(TIME, HEIGHT), float3
2 WSPD(TIME, HEIGHT), int8 WSPD_QC(TIME, HEIGHT), float32 UWND(TIME, HEIGHT), float32
VWND(TIME, HEIGHT)
    groups:
<class 'netCDF4._netCDF4.Variable'>
float32 UWND(TIME, HEIGHT)
    standard_name: eastward_wind
    long_name: Wind Speed U Component
    units: meters/second
    _FillValue: -999.0
    valid_min: 0.0
    valid_max: 35.0
    accuracy: 0.3 meters/second or 3%
    resolution: 0.2
    sensor_name: Wind Propeller [R.M. Young/05103]
    sensor_serial_number: 137964
    ancillary_variables: WDIR_QC WDIR_DM WSPD_QC WSPD_DM
    cell_methods: TIME: mean (interval: 2 minutes) HEIGHT:point LATITUDE:point LONGIT
```

```
UDE:point
    coordinates: TIME HEIGHT LATITUDE LONGITUDE
unlimited dimensions: TIME
current shape = (47605, 1)
```

#1: Make a preliminary assessment of the data. First, plot time series of the total wind speed ("WSPD"), zonal wind ("UWND"), and meridional wind ("VWND").

What is the time interval between data points? What is the time gap between the data files? How are gaps in the data handled?

In [3]:

```
time = []
ht = []
lon = []
lat = []

wind = []
u = []
v = []

# based on units of 'days since 1950-01-01'
s0 = dt.datetime(1950,1,1);
start = []; end = []

for nn,url in enumerate(links):
    nc = netCDF4.Dataset(url)

    ttime = nc['TIME'][:];
    datetimes = [s0+dt.timedelta(days=float(tt)) for tt in ttime]
    print(f'Dataset {nn} starts on {datetimes[0]} and ends on {datetimes[-1]}')
    start = np.append(start,datetimes[0])
    end = np.append(end,datetimes[-1])

    hht = nc['HEIGHT'][:]
    llon = nc['LONGITUDE'][:]
    llat = nc['LATITUDE'][:]

    wwind = nc['WSPD'][:];
    uu = nc['UWND'][:]
    vv = nc['VWND'][:]

    time = np.append(time,ttime)
    ht = np.append(ht,hht)
    lon = np.append(lon,llon); lat = np.append(lat,llat)

    wind = np.append(wind,wwind)
    u = np.append(u,uu); v = np.append(v,vv)



print(nc['WSPD'].units)
print(nc['TIME'].units)
```

```
Dataset 0 starts on 2015-04-26 01:00:00 and ends on 2016-03-21 15:00:00
Dataset 1 starts on 2016-03-22 00:30:00 and ends on 2017-06-05 21:00:00
```

```
Dataset 2 starts on 2017-06-06 19:10:00 and ends on 2018-04-27 16:00:00
meters/second
davs since 1950-01-01T00:00:00Z
```

In [4]:

```
# convert to datetimes based on units of 'days since 1950-01-01'
s0 = dt.datetime(1950,1,1)
dates = [s0+dt.timedelta(days=float(tt)) for tt in time]
```

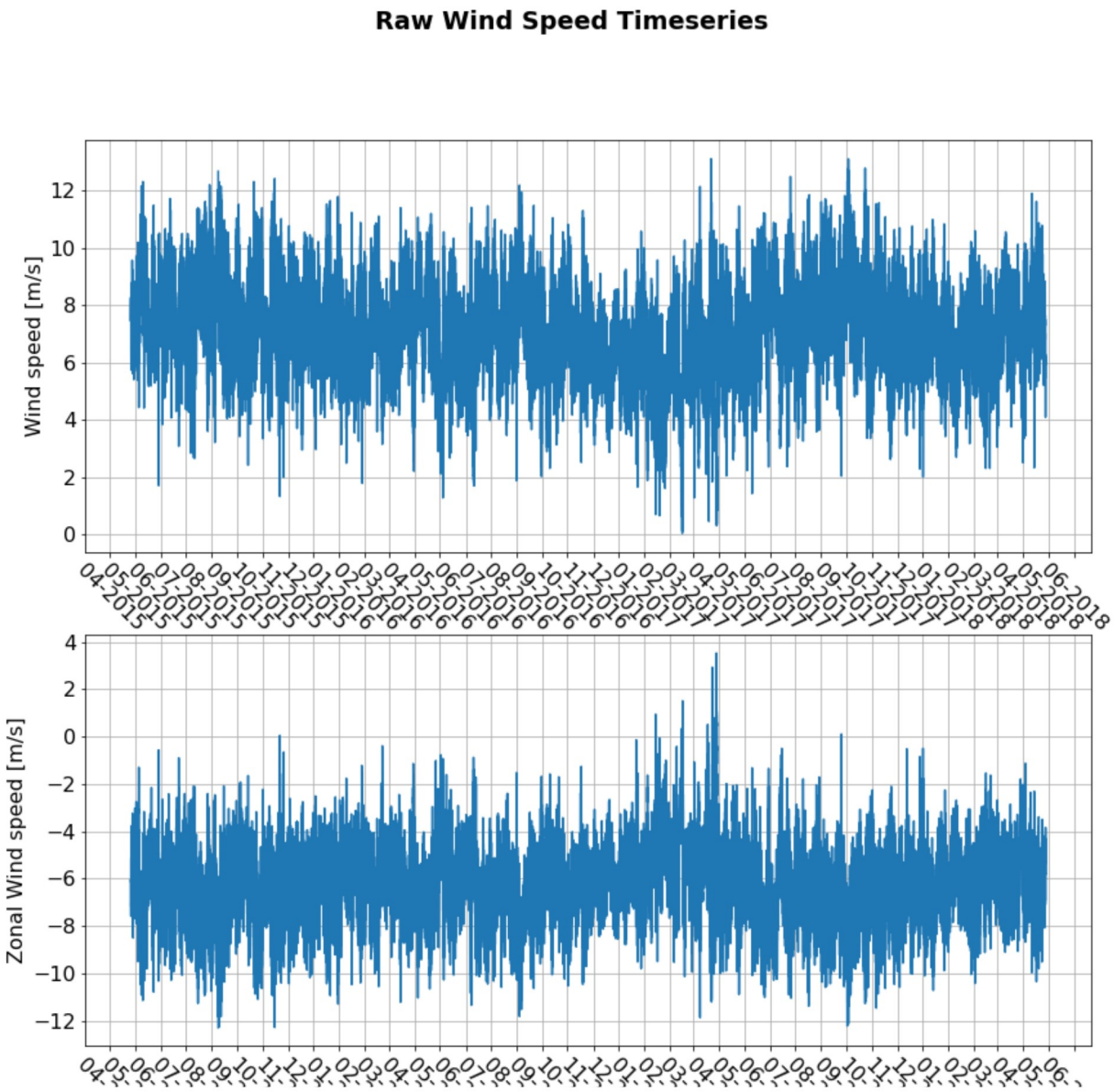
In [5]:

```
# doesn't physically make sense for wind speed to be less than zero - therefore, <0 v
# (also - fillValues for missing values are -999.0, so this method will also reset th

wind[np.nonzero(wind<0)] = np.nan

# u and v are velocities - therefore, just set np.nan if == fillValue == -999.0
u[np.nonzero(u<=-999.0)] = np.nan
v[np.nonzero(v<=-999.0)] = np.nan
```

In [6]:

```
x = [wind,u,v]
labels = ['Wind speed [m/s]','Zonal Wind speed [m/s]','Meridional Wind speed [m/s]']


fig,axes = plt.subplots(3,1,figsize=(14,20))
fig.suptitle('Raw Wind Speed Timeseries',y=0.95,fontweight='bold')
for nn,ax in enumerate(axes):
    print(x[nn])
    ax.grid()
    ax.plot(dates,x[nn])
    ax.set(ylabel=labels[nn])
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
    ax.xaxis.set_major_locator(mdates.MonthLocator())
    ax.tick_params(axis='x', labelrotation = -45)

plt.show()
```

```
[7.46999979 7.75       8.14000034 ... 6.01000023 5.59000015 6.01999998]
[-6.11999989 -6.57000017 -6.98000002 ... -5.0999999  -4.78999996
 -4.80999994]
[4.28000021 4.11000013 4.19000006 ... 3.19000006 2.88000011 3.63000011]
```



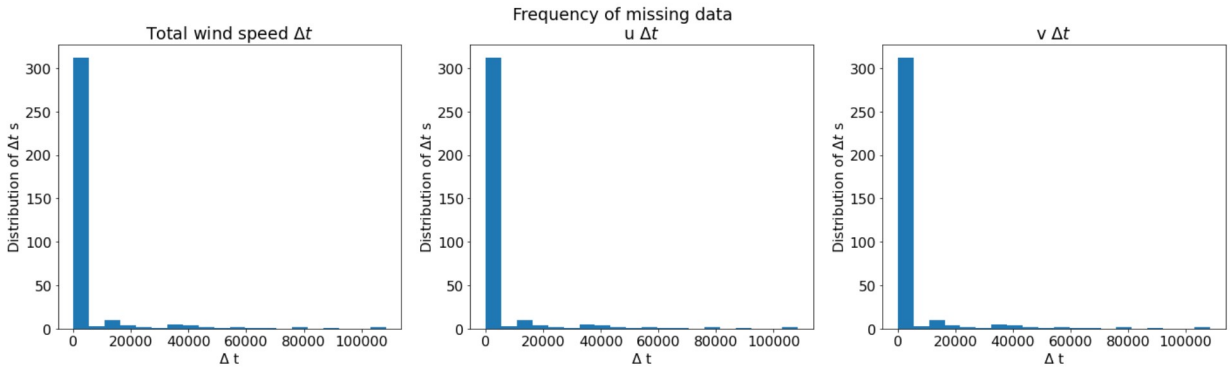
In [7]:

```
deltat = np.diff(time) # time in days
plt.hist(deltat*24*60,100,range=(8,12))
plt.xlim([0,20])
print(f'Time interval of sampling: {deltat*24*60} minutes')
```

```
Time interval of sampling: [10. 10. 10. ... 10. 10. 10.] minutes
```

In [8]:

```
# find the gaps

wgaps = np.isnan(wind)
ugaps = np.isnan(u)
vgaps = np.isnan(v)

# time vectors in minutes
wtime = time[wgaps]*24*60
utime = time[ugaps]*24*60
vtime = time[vgaps]*24*60

# how often is there missing data

wdt = np.diff(wtime)
udt = np.diff(utime)
vdt = np.diff(vtime)


titles = [r'Total wind speed $\Delta t$', r'u $\Delta t$', r'v $\Delta t$']

x = [wdt,udt,vdt]

fig,axes = plt.subplots(1,3,figsize=(24,6))
plt.suptitle('Frequency of missing data')
for vv,ax in enumerate(axes):
    ax.hist(x[vv],20)
    ax.set_title(titles[vv])
    ax.set_ylabel(r'Distribution of $\Delta t$ s')
    ax.set_xlabel(r'$ \Delta$ t ')
```



In [9]:

```
for nn,tt in enumerate(start):
    print(f'Dataset {nn} starts on {tt} and ends on {datetimes[-1]}')


print('Therefore, gap between data files is about 9 hours and 22 hours (respectively

print('The timestep between data samples is 10 minutes')

print('The missing data is stored with a FillValue of -999.0 in the NETCDF file.')
```

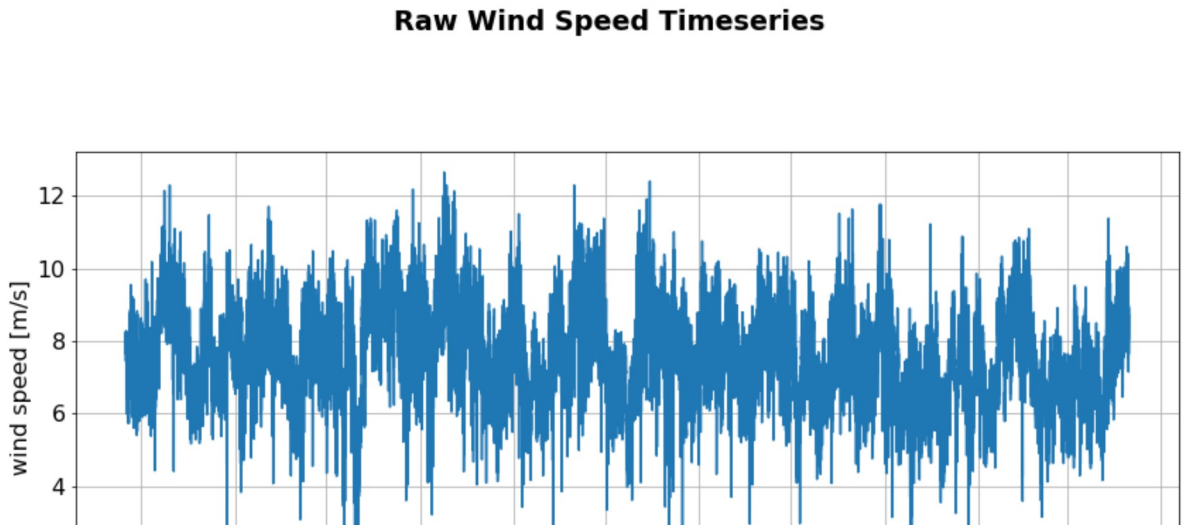
```
Dataset 0 starts on 2015-04-26 01:00:00 and ends on 2018-04-27 16:00:00
Dataset 1 starts on 2016-03-22 00:30:00 and ends on 2018-04-27 16:00:00
Dataset 2 starts on 2017-06-06 19:10:00 and ends on 2018-04-27 16:00:00
```

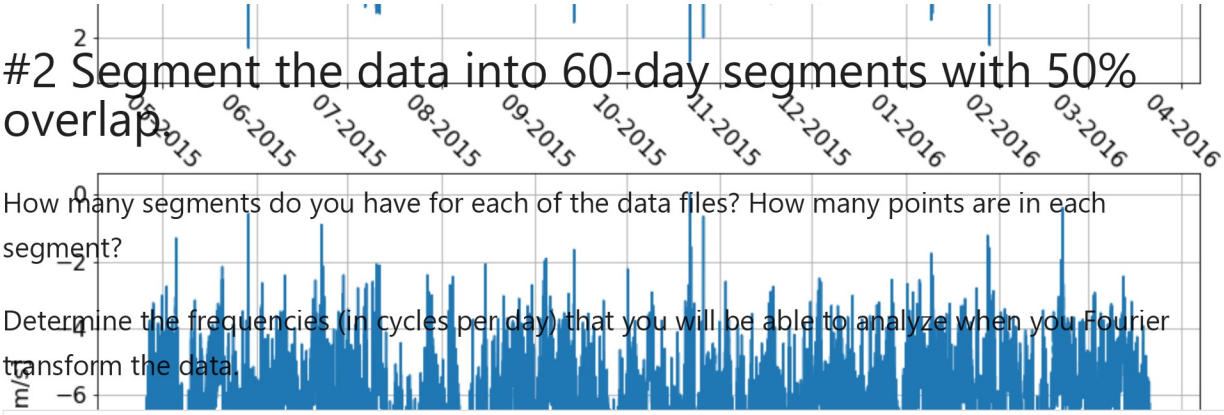
```
Therefore, gap between data files is about 9 hours and 22 hours (respectively between
Datset 1&2 and 2&3)
The timestep between data samples is 10 minutes
```

In [10]:

```
# since we know there are gaps, let's download again and interpolate as we go:

# reset variables

time = []
ht = []
lon = []
lat = []

wind = []
u = []
v = []

# don't need to loop because according to assignment, we want to look at each data f
'''
for nn,url in enumerate(links):
    nc = netCDF4.Dataset(url)

    ttime = nc['TIME'][:];
    hht = nc['HEIGHT'][:]
    llon = nc['LONGITUDE'][:]
    llat = nc['LATITUDE'][:]

    wwind = nc['WSPD'][:]; wwind[wwind<0] = np.nan # make nan when not physical
    wwind = np.squeeze(wwind.data) # make into a 1D array (before it was a (47605,1)
    # make into a xarray so we can use interpolate_na function
    dw = xr.DataArray(data=wwind.data,dims="time", coords={"time": ttime})
    dw = dw.interpolate_na(dim='time',method="cubic", fill_value="extrapolate")
    dw = dw.values

    uu = nc['UWND'][:]; uu[uu<=-999] = np.nan # make nan when fillValue
    uu = np.squeeze(uu.data) # make into a 1D array (before it was a (47605,1) shape
    du = xr.DataArray(data=uu.data,dims="time", coords={"time": ttime})
    du = du.interpolate_na(dim='time',method="cubic", fill_value="extrapolate")
    du = du.values

    vv = nc['VWND'][:]; vv[vv<=-999] = np.nan # make nan when fillValue
    vv = np.squeeze(vv.data) # make into a 1D array (before it was a (47605,1) shape
    dv = xr.DataArray(data=vv.data,dims="time", coords={"time": ttime})
    dv = dv.interpolate_na(dim='time',method="cubic", fill_value="extrapolate")
    dv = dv.values

    time = np.append(time,ttime)
    ht = np.append(ht,hht)
    lon = np.append(lon,llon); lat = np.append(lat,llat)

    # append with interpolated values now
    wind = np.append(wind,dw)
    u = np.append(u,du); v = np.append(v,dv)

'''

def download_winds(url=links[0],plotting=0):
    s0 = dt.datetime(1950,1,1);

    nc = netCDF4.Dataset(url)
    time = nc['TIME'][:]; dates = [s0+dt.timedelta(days=float(tt)) for tt in time]
    ht = nc['HEIGHT'][:]
```

```
    lon = nc['LONGITUDE'][:]; lat = nc['LATITUDE'][:]

    wind = nc['WSPD'][:]; wind[wind<0] = np.nan # make nan when not physical
    wind = np.squeeze(wind.data) # make into a 1D array (before it was a (47605,1) sh
    # make into a xarray so we can use interpolate_na function
    dw = xr.DataArray(data=wind.data,dims="time", coords={"time": time})
    dw = dw.interpolate_na(dim='time',method="linear")
    wind = dw.values

    u = nc['UWND'][:]; u[u<=-998] = np.nan # make nan when fillValue
    u = np.squeeze(u.data) # make into a 1D array (before it was a (47605,1) shaped a
    du = xr.DataArray(data=u.data,dims="time", coords={"time": time})
    du = du.interpolate_na(dim='time',method="linear")
    u = du.values

    v = nc['VWND'][:]; v[v<=-998] = np.nan # make nan when fillValue
    v = np.squeeze(v.data) # make into a 1D array (before it was a (47605,1) shaped a
    dv = xr.DataArray(data=v.data,dims="time", coords={"time": time})
    dv = dv.interpolate_na(dim='time',method="linear")
    v = dv.values

    if plotting == 1:
        plots = [wind,u,v]; labels = ['wind speed [m/s]','u [m/s]','v [m/s]']; titles
        fig,axes = plt.subplots(3,1,figsize=(14,20))
        fig.suptitle('Raw Wind Speed Timeseries',y=0.95,fontweight='bold')
        for nn,ax in enumerate(axes):
            ax.grid()
            ax.plot(dates,plots[nn])
            ax.set(ylabel=labels[nn]); # ax.set(title=titles[nn])
            ax.xaxis.set_major_formatter(mdates.DateFormatter('%m-%Y'))
            ax.xaxis.set_major_locator(mdates.MonthLocator())
            ax.tick_params(axis='x', labelrotation = -45)
        plt.show()

    return time, wind, u, v


# for 2015 dataset:
link15 = links[0]
[time15,wind15,u15,v15] = download_winds(link15,plotting=1)
```





#2 Segment the data into 60-day segments with 50% overlap

How many segments do you have for each of the data files? How many points are in each segment?

Determine the frequencies (in cycles per day) that you will be able to analyze when you Fourier transform the data.

In [11]:

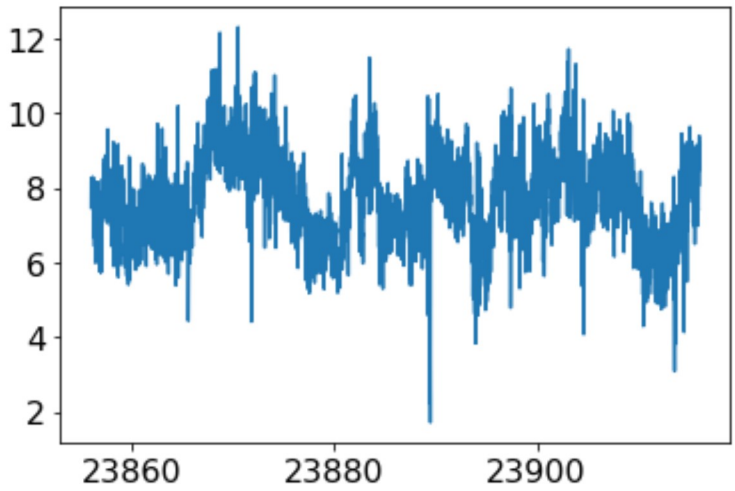
```
# windowing data

# change this depending on which file you are using
time = time15; wind = wind15; u = u15; v = v15;

tseg = dt.timedelta(days=float(60))
# or
tseg = 60 # time array is in day units

# number of 60 day intervals encompassed in full dataset
intervals = int(np.floor((time[-1]-time[0])/tseg))
# with 50% overlaps, it will be doubled but 1 less (because start 50% and end 50%)
cycles = (intervals*2)-1

num_samples = int(60/10 *60*24) # number of samples if taken every 10 min in 60 days


# can make arrays the correct size because we know how many 10 min samples will be i
# each array = (segment num, data within segment) == (18,8640)
wsegs = np.zeros((cycles,num_samples))
usegs = np.zeros((cycles,num_samples))
vsegs = np.zeros((cycles,num_samples))
time_segs = np.zeros((cycles,num_samples))


ind = 0
jump = int(num_samples/2);
for nn in np.arange(0,cycles):
    # looping through each segment in which we will save data
    wsegs[nn,:] = wind[ind:ind+num_samples]
    usegs[nn,:] = u[ind:+ind+num_samples]
    vsegs[nn,:] = v[ind:+ind+num_samples]
    t_arr = time[ind:ind+num_samples]
    time_segs[nn,:] = t_arr.data
    # date_segs[nn,:] = [s0+dt.timedelta(days=float(tt)) for tt in t_arr]


    # now add to the ind (start index) and jump (number of samples in the segment = 8
    ind = ind + jump
    print(ind)
```

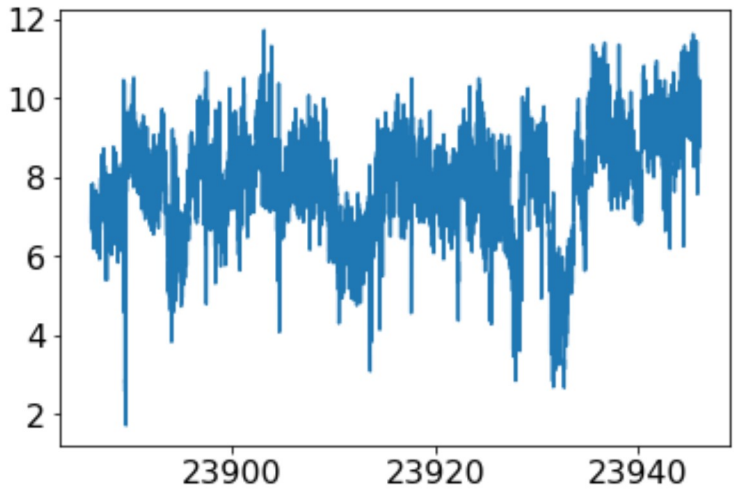
```
4320
8640
```

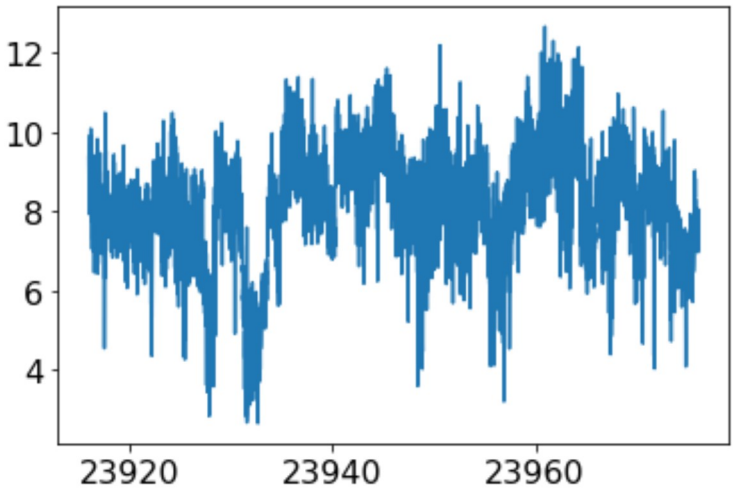
```
12960
17280
21600
25920
30240
34560
```

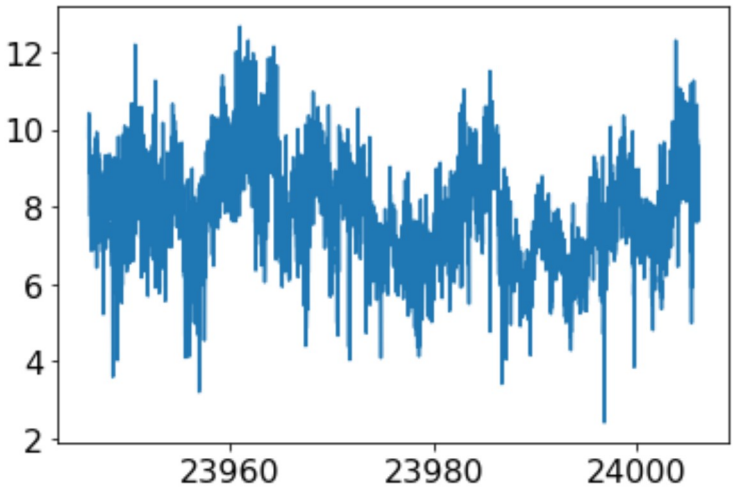
In [12]:

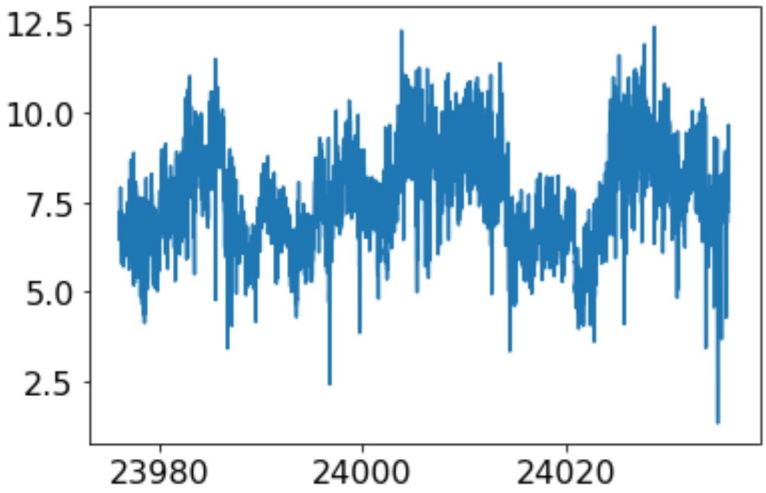
```
# plotting to make sure all went well & as intended

for nn in np.arange(0,cycles):
    plt.plot(time_segs[nn,:],wsegs[nn,:])
    plt.show()
```

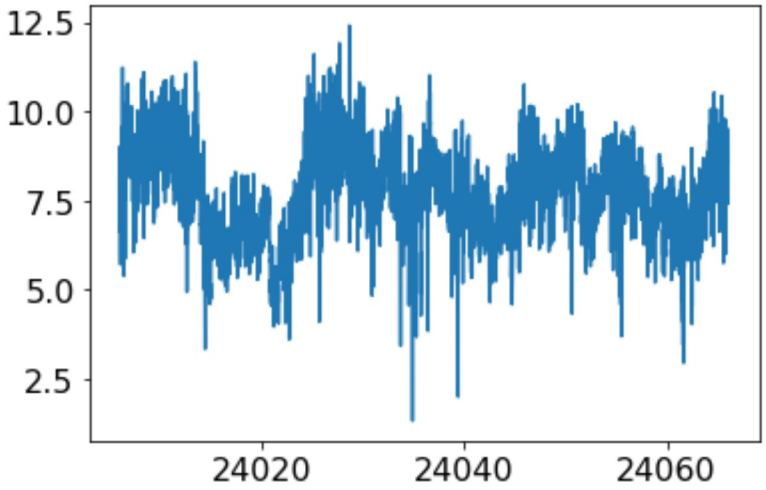


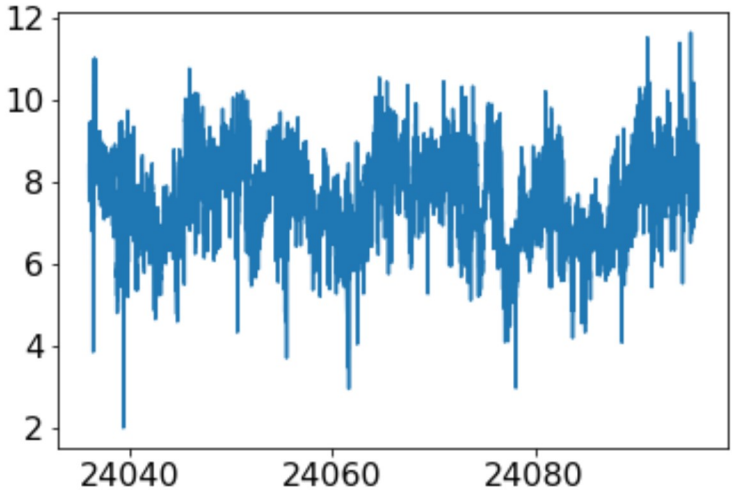


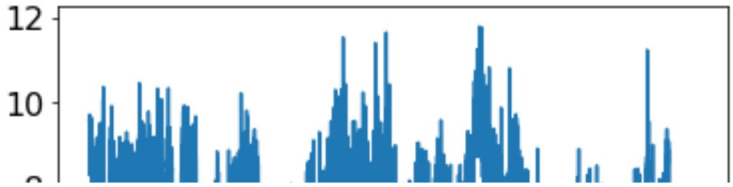

12
10
8
6
4
23920
23940
23960


12
10
8
6
4
2
23960
23980
24000


12.5
10.0
7.5
5.0
2.5
23980
24000
24020







In [13]:
```
print(f'For the entire 2015 data file, I have a total of {intervals} 60-day interval

print(f'In each segment, there are {num_samples} data points - each spaced 10 minute

print(f'When I Fourier transform, my Nyquist frequency will stay the same as the ori
print(f'But my fundamental frequency will now be {1/tseg: 5.5f} cycles per day (1/60
```

```
For the entire 2015 data file, I have a total of 5 60-day intervals,
 and when I overlap them by 50%, I have 9 segments that I can analyze
In each segment, there are 8640 data points - each spaced 10 minutes (or  0.006953 da
ys) apart
When I Fourier transform, my Nyquist frequency will stay the same as the original dat
aset, at 1/(2*10 min)
But my fundamental frequency will now be  0.01667 cycles per day (1/60 day^-1)
```

This also means that when I do my Fourier transform, I'll have $v = 2 * M$ degrees of freedom, where I have M data segments. Therefore, with 9 segments, I have 18 degrees of freedom.

#3 Compute and plot spectra using 3 different

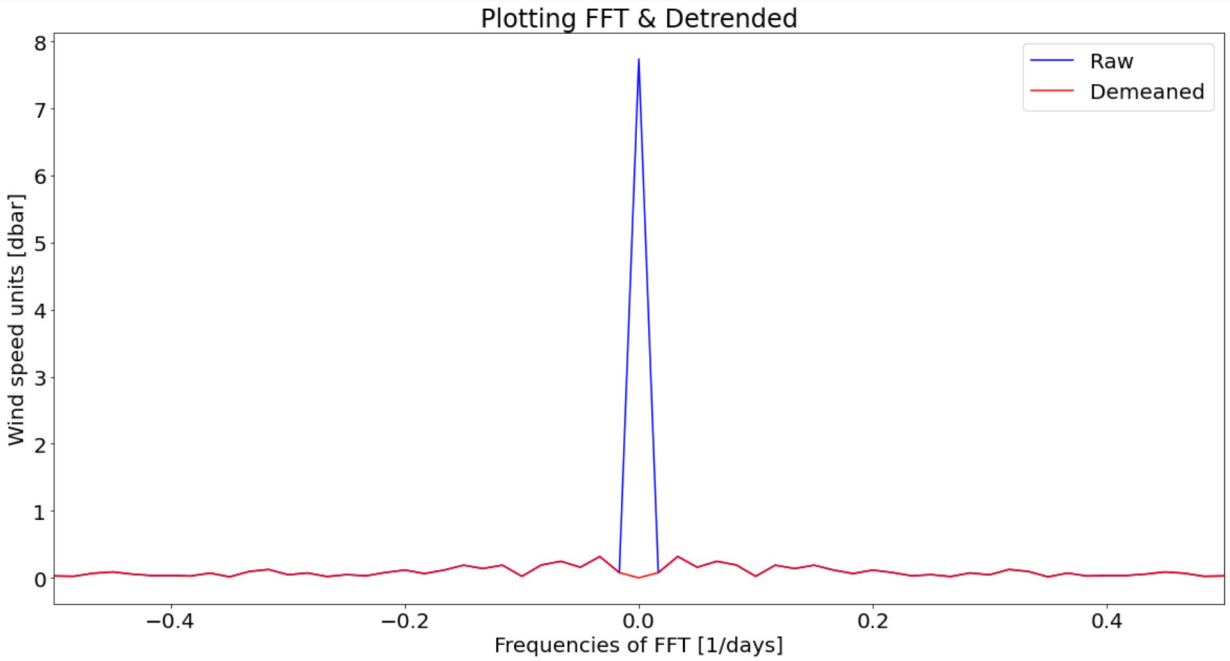
approaches, for the 2015 wind speed record only.

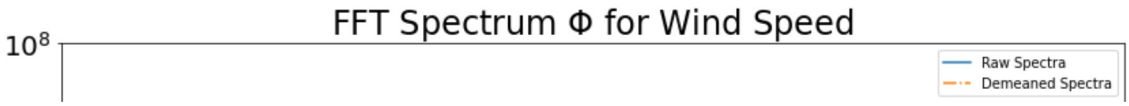
Compute the spectrum from the raw data, from the detrended data, and from the detrended data with a Hanning window applied. How does detrending and windowing alter the spectrum in this case?

In [16]:

```
# raw data for 1 segment

N = len(wsegs[nn,:])  # should be num_samples
step = np.nanmean(deltat) # should be 10 minutes (in days)
Nyq = 1/(2*step)
df = 1/tseg # cycles per day

# fourier
fft = scipy.fft.fft(wsegs[0,:])
freq = scipy.fft.fftfreq(N,step)
freq = scipy.fft.fftshift(freq)
fftplot = scipy.fft.fftshift(fft)


# fourier with very basic demean only
dtend = wsegs[0,:] - np.nanmean(wsegs[0,:])
dtfft = scipy.fft.fft(dtend)
dtp = scipy.fft.fftshift(dtfft)


# find means & amplitudes from FFT
from scipy import signal

norm_fft = 1.0/N * np.abs(fftplot)
norm_dt = 1.0/N * np.abs(dtp)

#  peaks, _ = scipy.signal.find_peaks(norm_fft,height=0.09) # threshold=(2,5))


fig,ax = plt.subplots(figsize=(20,10))
ax.plot(freq, norm_fft,color='blue',label='Raw')
ax.plot(freq, norm_dt,color='red',label='Demeaned')
ax.set_xlim([-1/2, 1/2]) # make a window that encompasses signals with period of 2 d
ax.set(ylabel='Wind speed units [dbar]',xlabel='Frequencies of FFT [1/days]',title='F
ax.legend()
plt.show()


# compare spectra too

even_idx = np.arange(0,N,2)
amp = abs(fft[even_idx])**2 # even N
ampdt = abs(dtfft[even_idx])**2 # even N

# find corresponding frequencies
period = step*N # this is the entire period (should == tseg)
df = 1/period # should equal fundamental_f
fN = Nyq # from before: Nyq = 1/(2*step)
freqs = np.arange(0,fN,df) # frequency vector, in [1/day], goes from 0 to Nyquist


# spectrum:
amp = amp/(N^2) # correct normalization
amp = amp*2 # account for discarded redundant complex FFT coefficients
amp = amp/df

# detrended spectrum:
ampdt = ((ampdt/(N^2))*2)/df # correct normalization, account for discarded redundan
```

```
plt.rcParams.update({'font.size': 20})
# plot
fig,ax = plt.subplots(figsize=(14,10))
ax.plot(freqs,amp,label='Raw Spectra')
ax.plot(freqs,ampdt,label='Demeaned Spectra',linestyle='-.')
ax.set_xscale('log'); ax.set_yscale('log')
ax.set_ylim([10e-3, 10e7])
ax.set(xlabel=r'FFT Frequencies [cycles per day]',
       ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',
       title=r'FFT Spectrum $\Phi$ for Wind Speed')
ax.legend(fontsize=10)
plt.show()
```





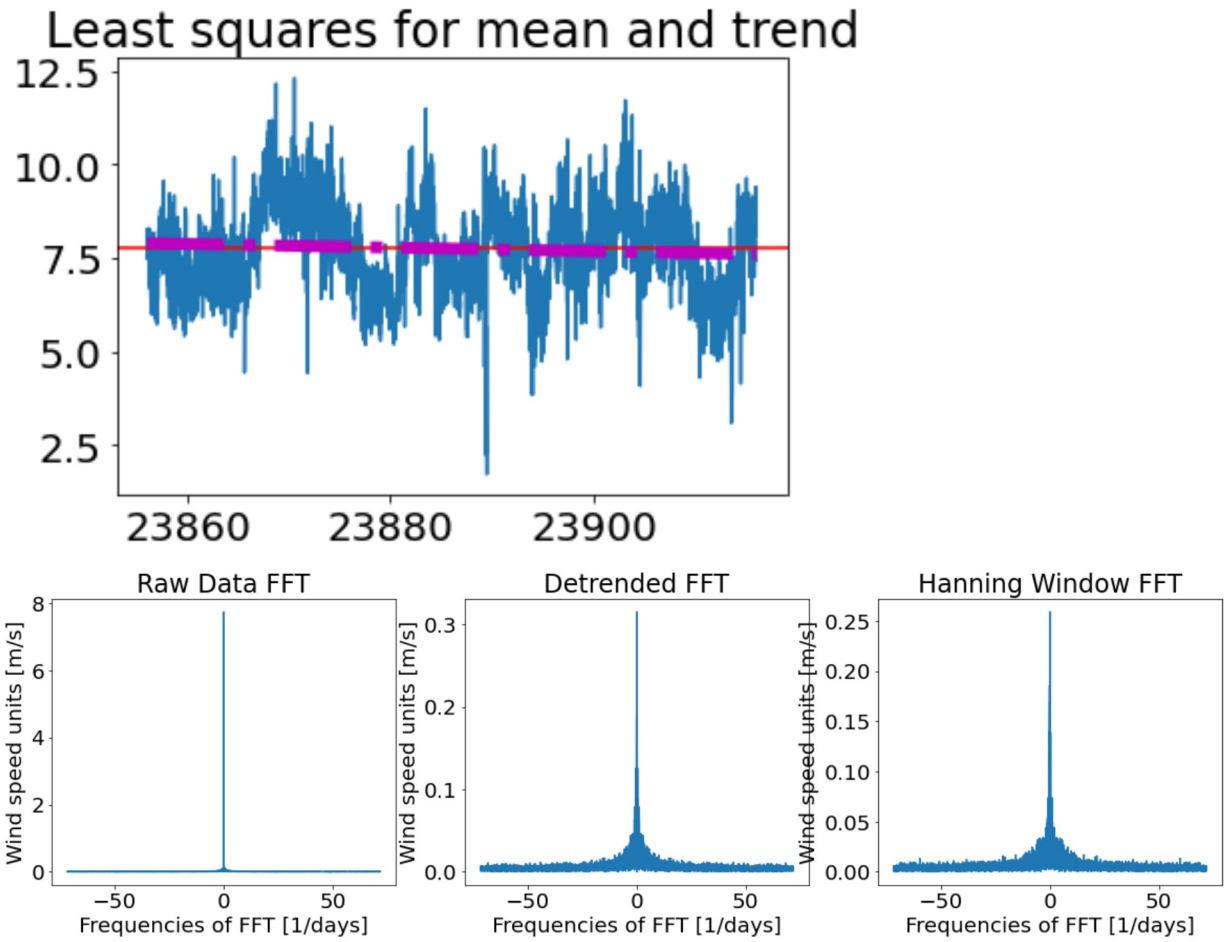
Should look pretty much the same - just removed mean for the demeaned.

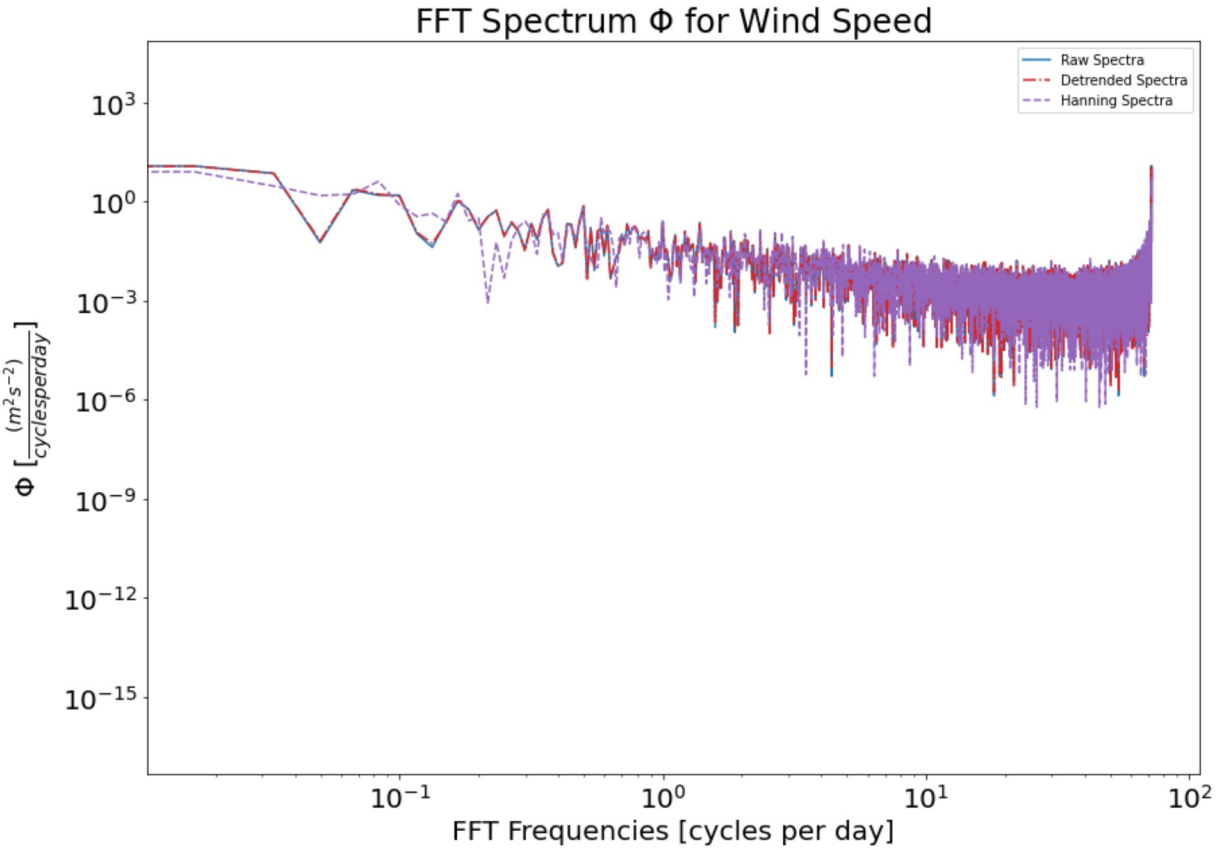
In [17]:

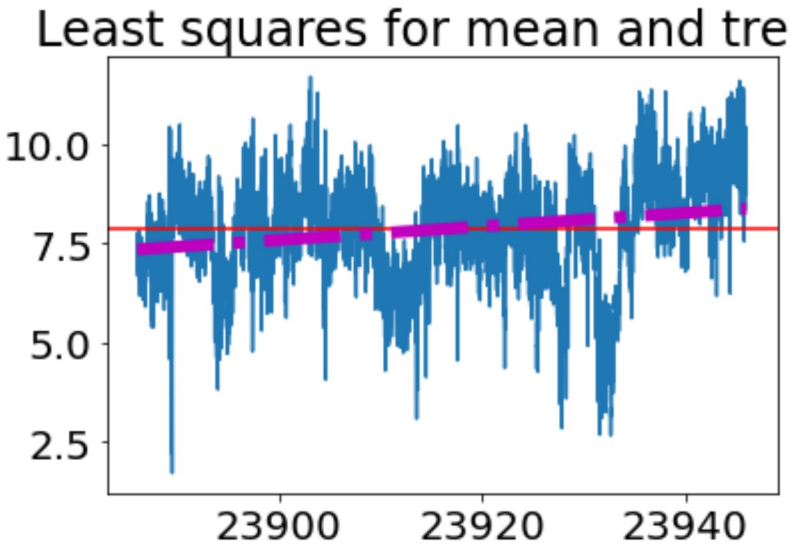
```
# we're going to do a test workflow with across 2015 windspeed data, and then make it

# raw data for all segments and averaged

'''
# how to window:
# 1: you must demean your data-otherwise, the window will shift energy from the mean
If you're working in segments, you should demean (and detrend) each segment before yo
# 2:  for a segment with N points, multiply by a window that is N points wide.
# 3: Since the window attenuates the impact of the edge of each segment,
you can use segments that overlap (typically by 50%). This will give you (almost) twi
so instead of ν degrees some larger number.
4. Now Fourier transform, scale appropriately
(e.g. by √8/3 for a Hanning window, to account for energy attenuation) and compute ar

'''

# all segment lengths should be the same
N = len(wsegs[0,:])  # == num_samples
step = np.nanmean(deltat) # step size
Nyq = 1/(2*step) # Nyquist frequency
period = step*N # this is the entire period (in days)
df = 1/period # fundamental frequency
segs = len(wsegs[:,0])

# start workflow: analyze each segment individually

amp_raw = np.zeros((segs,int(N/2)))
amp_dt =  np.zeros((segs,int(N/2)))
amp_han =  np.zeros((segs,int(N/2)))

for nn in np.arange(0,segs):
    segment = wsegs[nn,:]
    segtime = time_segs[nn,:]

    # demean & detrend

    # calculate mean and linear trend
    AA = np.array([np.ones(N), segtime]).T
    x = np.dot(np.linalg.inv(np.dot(AA.T, AA)), np.dot(AA.T, segment))
    mean = x[0]; trend = x[1]
    mymean = np.nanmean(segment)


    # plot to make sure it looks okay
    plt.plot(segtime,segment)
    plt.axhline(y=mymean,color='red');
    plt.plot(time_segs[nn,:],mean+segtime*trend,color='m',linewidth=6,linestyle='-.')
    plt.title('Least squares for mean and trend')
    plt.show()

    # do the demean & detrend !
    dmseg = segment-mean
    dtseg = dmseg-(trend*segtime)

    # compute each segment spectrum

    # fourier
```

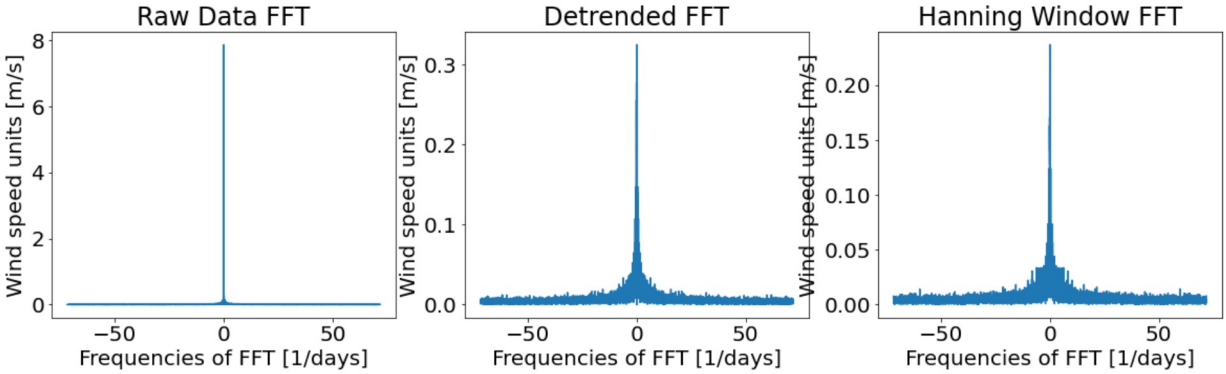
```
# raw
fft = scipy.fft.fft(segment);
freq = scipy.fft.fftfreq(N,step)
freq = scipy.fft.fftshift(freq)
fftplot = scipy.fft.fftshift(fft)
norm_fft = 1.0/N * np.abs(fftplot)

# detrended
dtfft = scipy.fft.fft(dtseg);
dtp = scipy.fft.fftshift(dtfft)
norm_dt = 1.0/N * np.abs(dtp)

# hanning
# make window:
hanwin = np.cos(np.pi*segtime / period)**2
# multiply the segment by window & normalize by sqrt(8/3)
han = dtseg*hanwin
han = han*np.sqrt(8/3)
hanfft = scipy.fft.fft(han);
htp = scipy.fft.fftshift(hanfft)
norm_han = 1.0/N * np.abs(htp)


# make figures on which you will put each of the calculated FFTs and spectra
fig1,(ax1,ax2,ax3) = plt.subplots(1,3,figsize=(20,5))

ax1.plot(freq, norm_fft); ax1.set(ylabel='Wind speed units [m/s]',xlabel='Freque
ax2.plot(freq, norm_dt); ax2.set(ylabel='Wind speed units [m/s]',xlabel='Frequen
ax3.plot(freq, norm_han); ax3.set(ylabel='Wind speed units [m/s]',xlabel='Freque

even_idx = np.arange(0,N,2)
amp = abs(fft[even_idx])**2 # even N
ampdt = abs(dtfft[even_idx])**2
amphan = abs(hanfft[even_idx])**2

# corresponding frequencies based on data length
freqs = np.arange(0,Nyq,df) # frequency vector, in [1/day], goes from 0 to Nyquis

# raw spectrum:
amp = amp/(N**2) # correct normalization
amp = amp*2 # account for discarded redundant complex FFT coefficients
amp = amp/df

# detrended spectrum
ampdt = ((ampdt/(N**2))*2)/df

# hanning window spectrum
# normalize by sqrt(8/3) - did this directly to the hanning window!
amphan = ((amphan/(N**2))*2)/df

plt.rcParams.update({'font.size': 20})
# plot
fig,ax = plt.subplots(figsize=(14,10))
ax.plot(freqs,amp,label='Raw Spectra',color='tab:blue')
ax.plot(freqs,ampdt,label='Detrended Spectra',color='tab:red',linestyle='-.')
ax.plot(freqs,amphan,label='Hanning Spectra',color='tab:purple',linestyle='--')

ax.set_xscale('log'); ax.set_yscale('log')
```

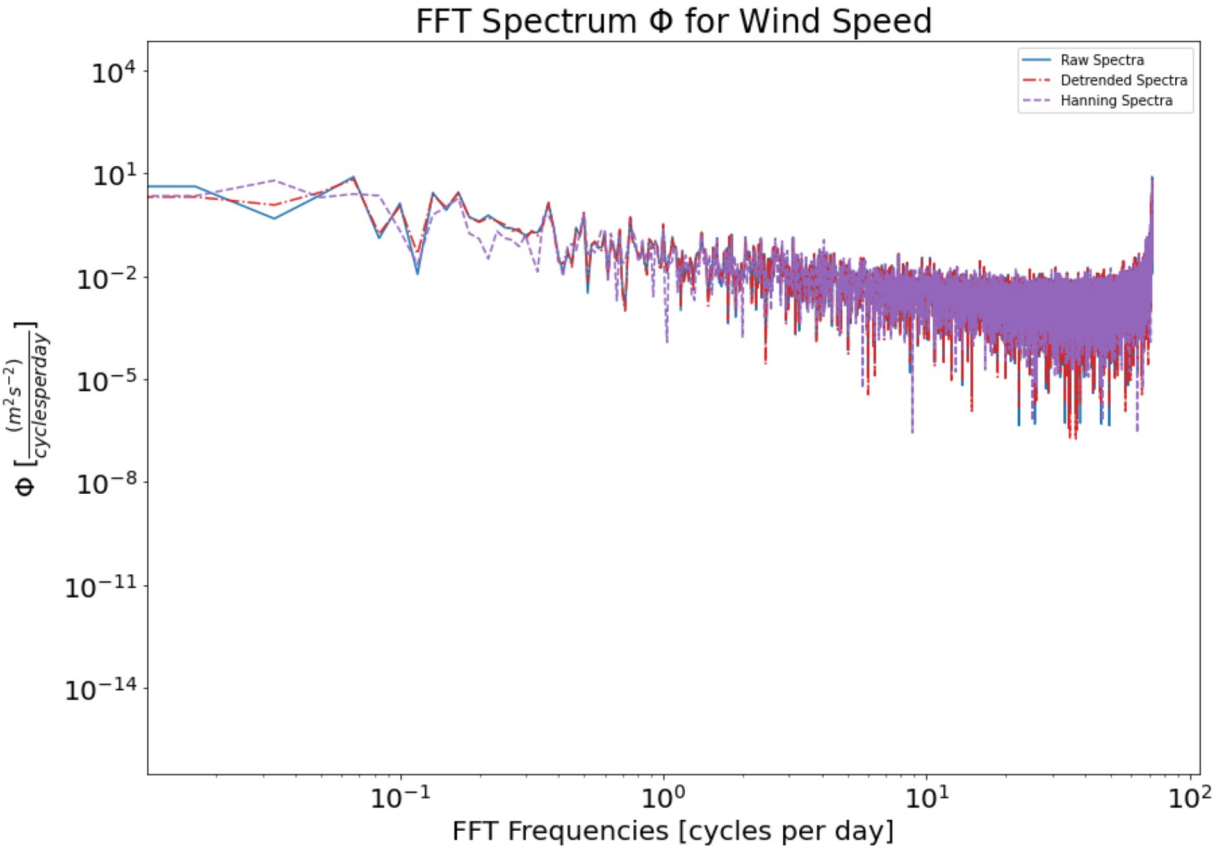
```
    ax.set(xlabel=r'FFT Frequencies [cycles per day]',
       ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',
       title=r'FFT Spectrum $\Phi$ for Wind Speed')
    ax.legend(fontsize=10)
    plt.show()


    amp_raw[nn,:] = amp
    amp_dt[nn,:] = ampdt
    amp_han[nn,:] = amphan
```

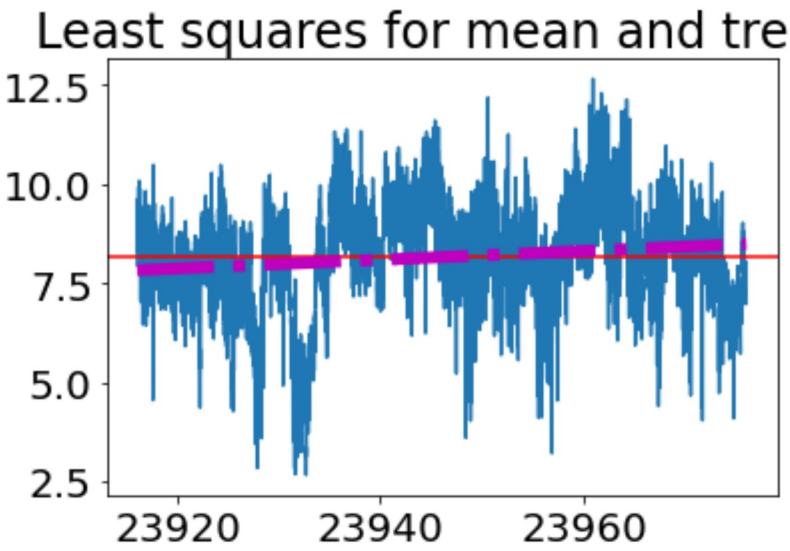


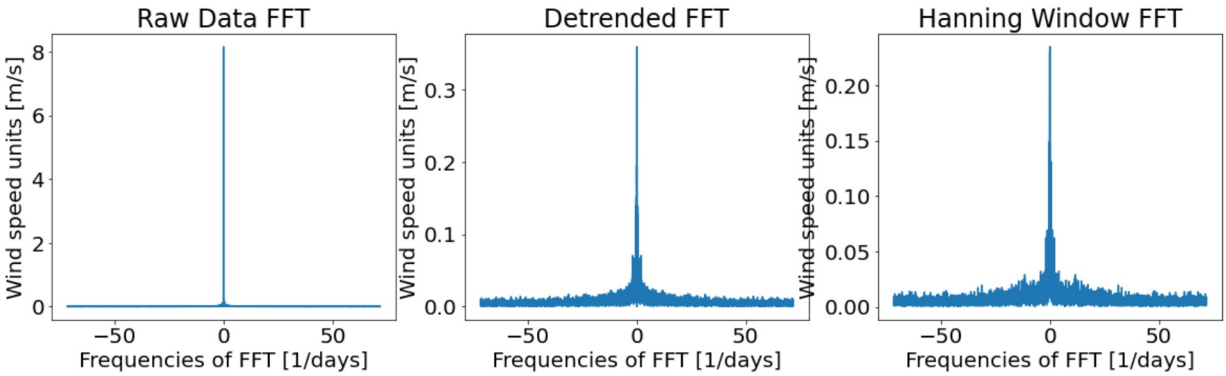

FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^{3}
10^{0}
10^{-3}
10^{-6}
10^{-9}
10^{-12}
10^{-15}
$\Phi\ [\frac{(m^2 s^{-2})}{cyclesperday}]$
10^{-1}
10^{0}
10^{1}
10^{2}
FFT Frequencies [cycles per day]

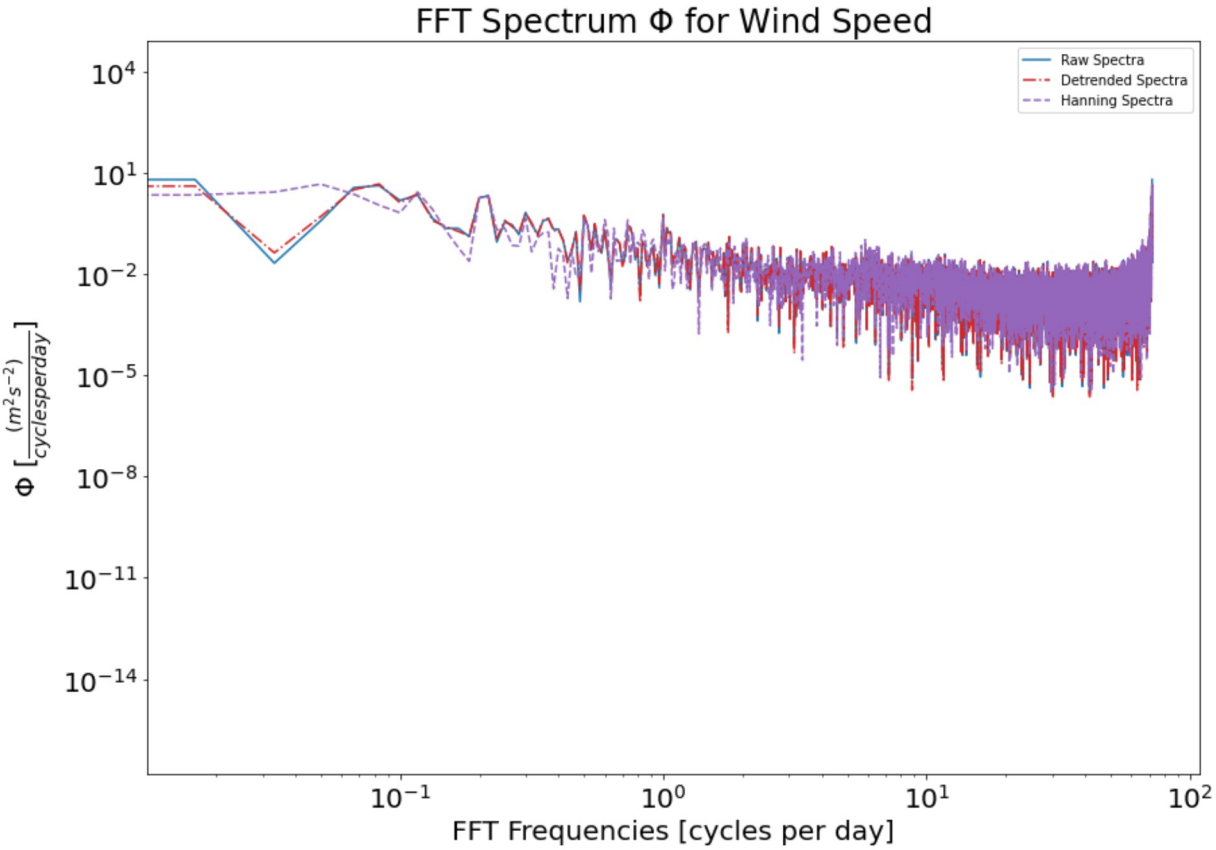

Least squares for mean and trend
10.0
7.5
5.0
2.5
23900
23920
23940

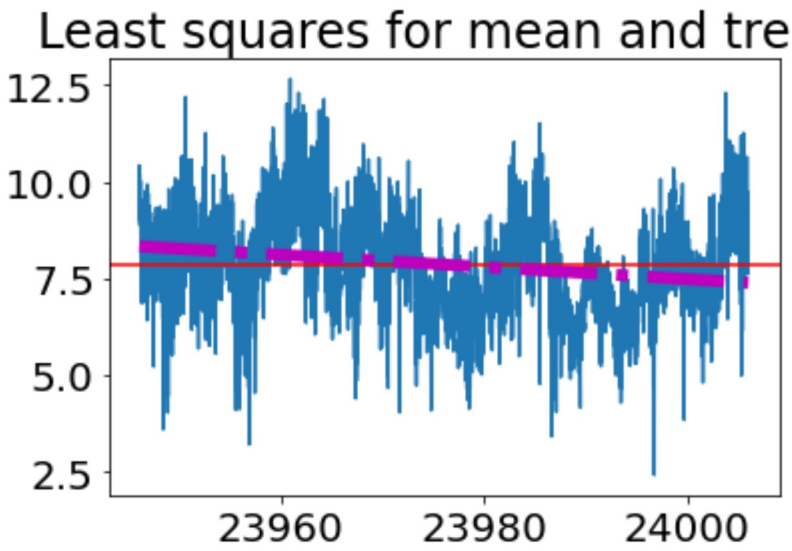

Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50

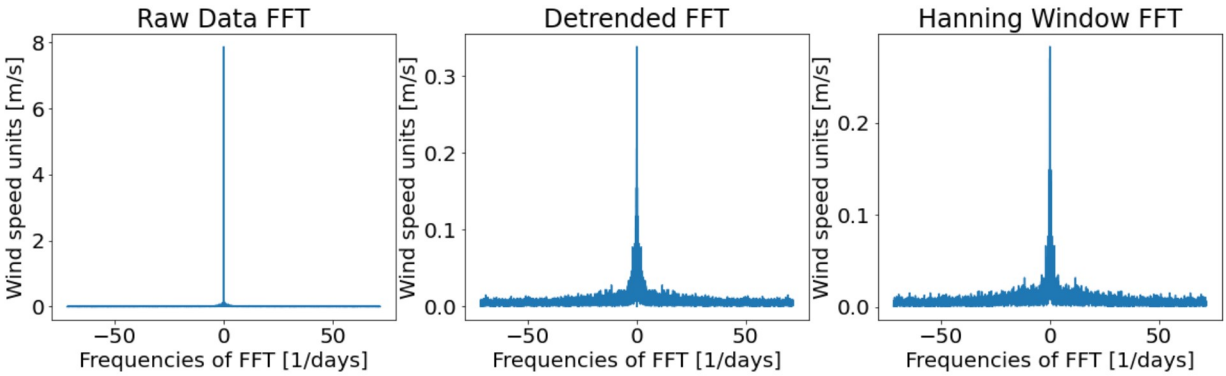

FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^{4}
10^{1}
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
$\Phi\ [\frac{(m^2 s^{-2})}{cyclesperday}]$
10^{-1}
10^{0}
10^{1}
10^{2}
FFT Frequencies [cycles per day]

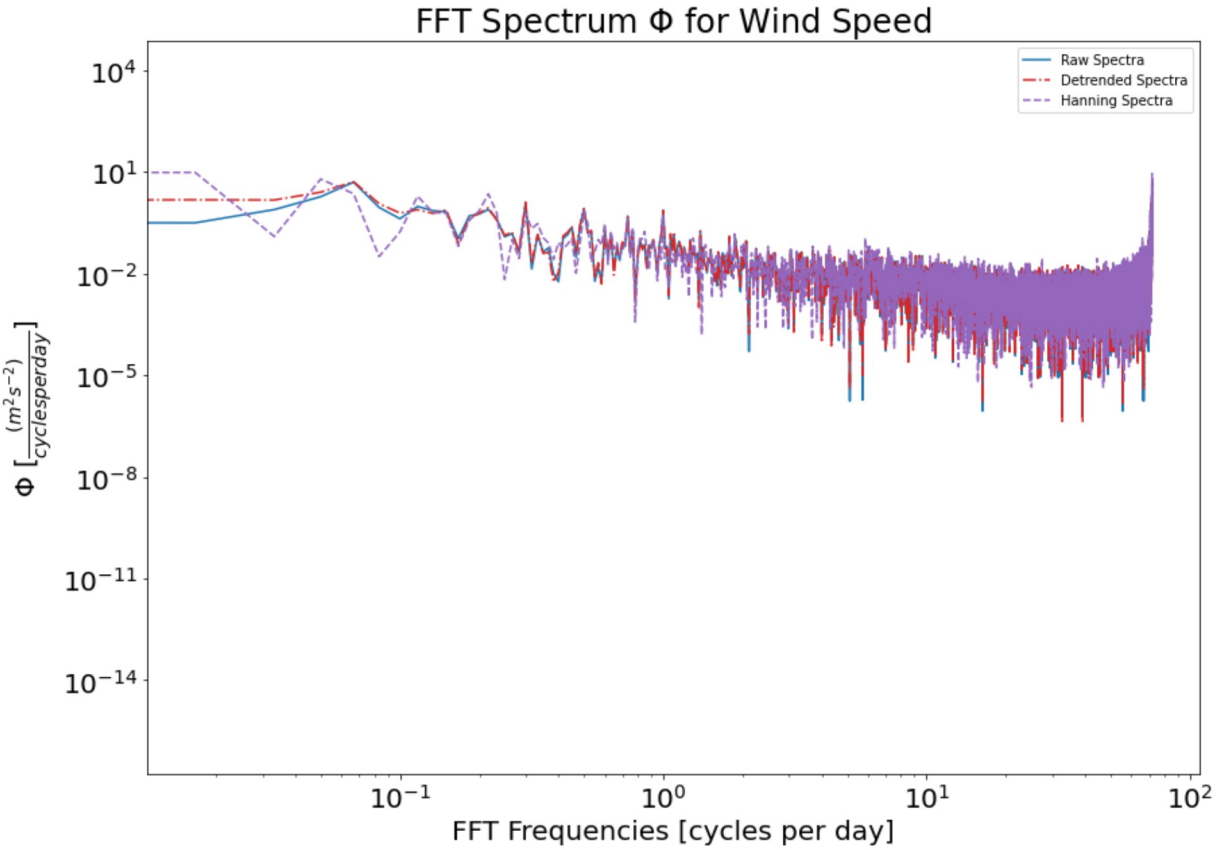

Least squares for mean and trend
12.5
10.0
7.5
5.0
2.5
23920
23940
23960

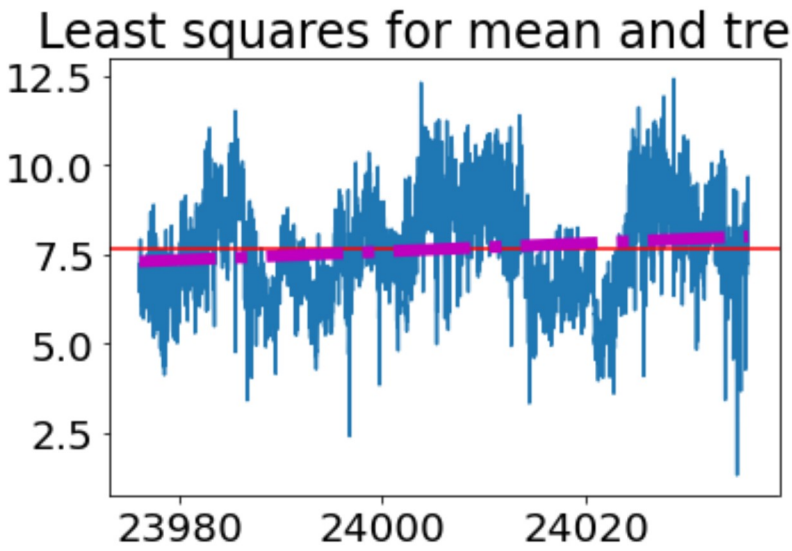

Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50

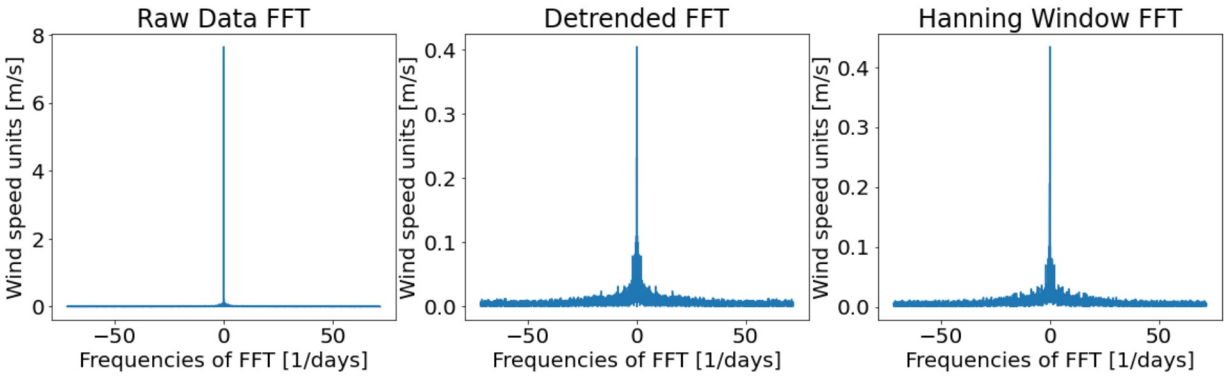

FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^{4}
10^{1}
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
$\Phi\ [\frac{(m^2 s^{-2})}{cyclesperday}]$
10^{-1}
10^{0}
10^{1}
10^{2}
FFT Frequencies [cycles per day]

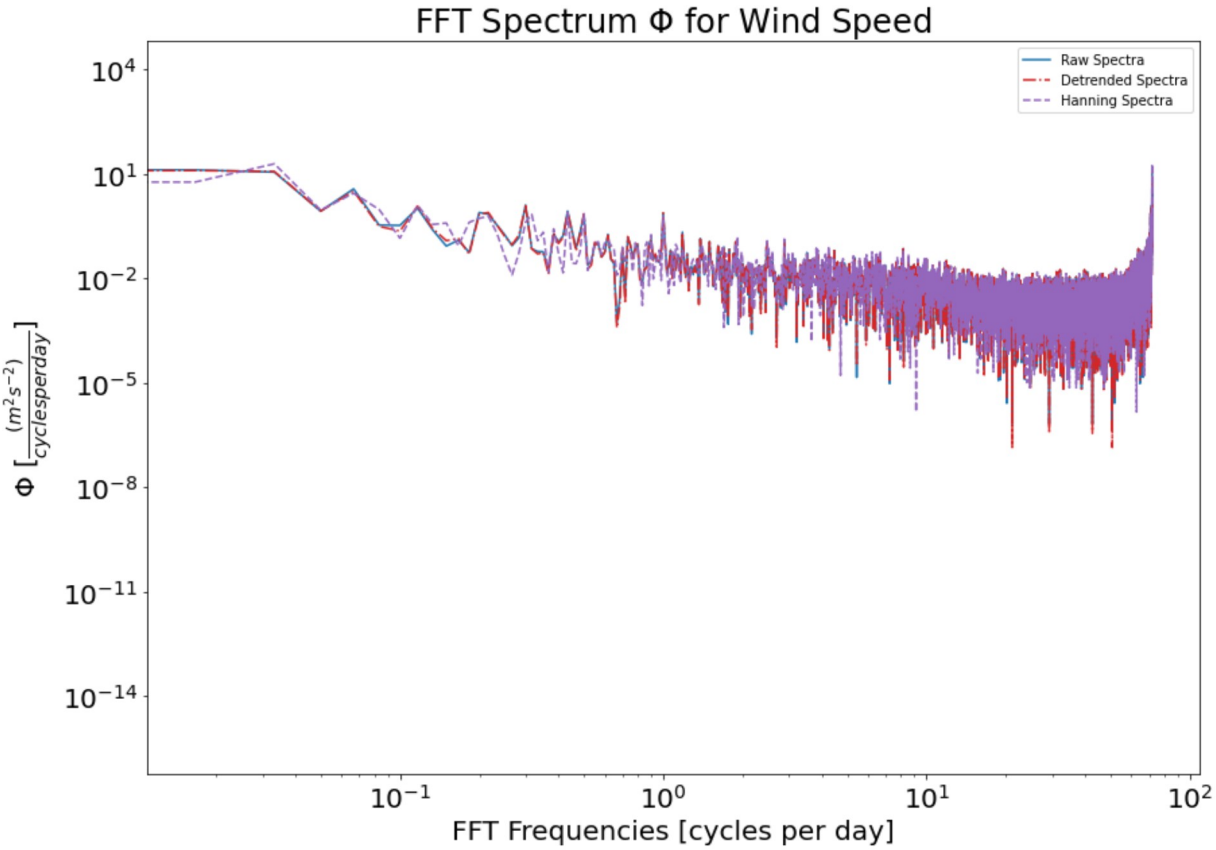

Least squares for mean and trend
12.5
10.0
7.5
5.0
2.5
23960
23980
24000

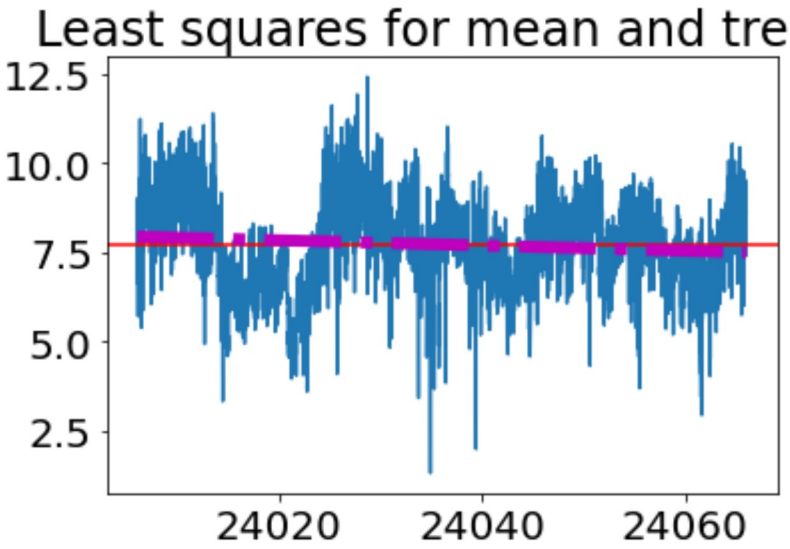

Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50

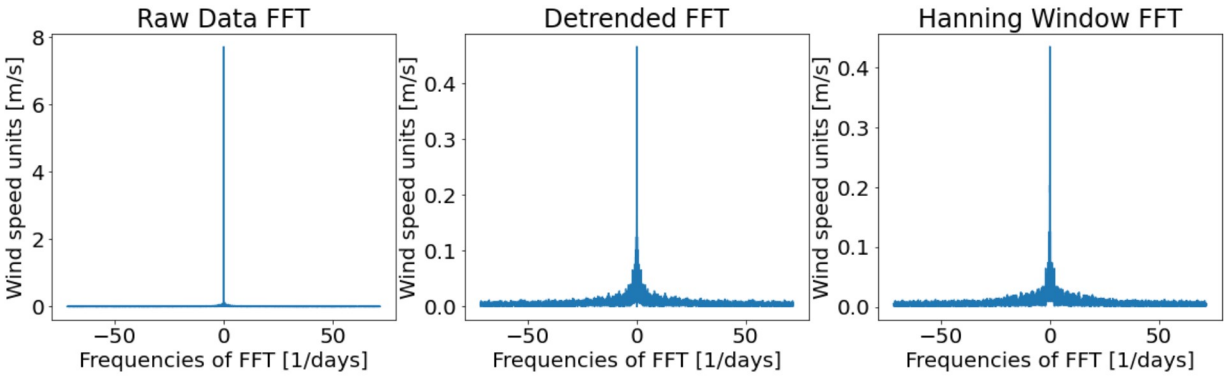

FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^4
10^1
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
$\Phi\ [\frac{(m^2s^{-2})}{cyclesperday}]$
10^{-1}
10^0
10^1
10^2
FFT Frequencies [cycles per day]

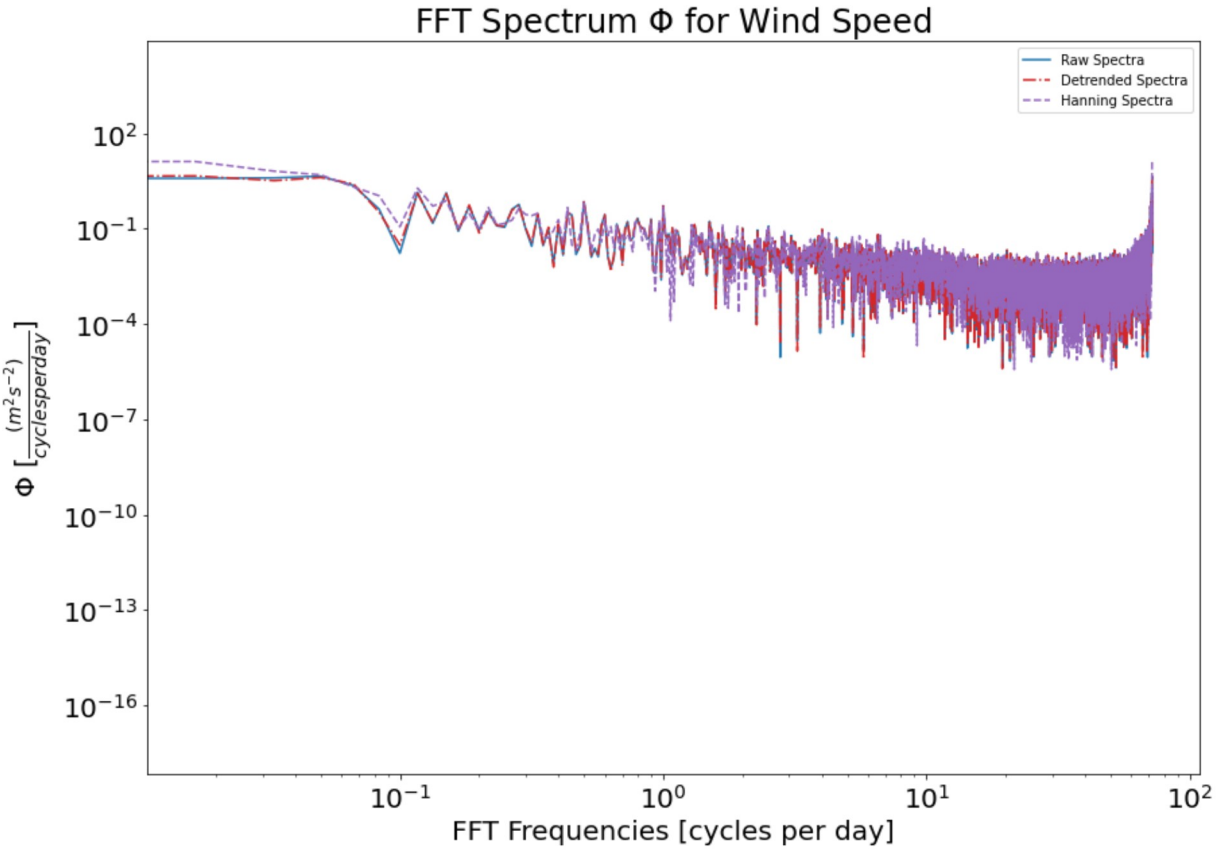

Least squares for mean and trend
12.5
10.0
7.5
5.0
2.5
23980
24000
24020

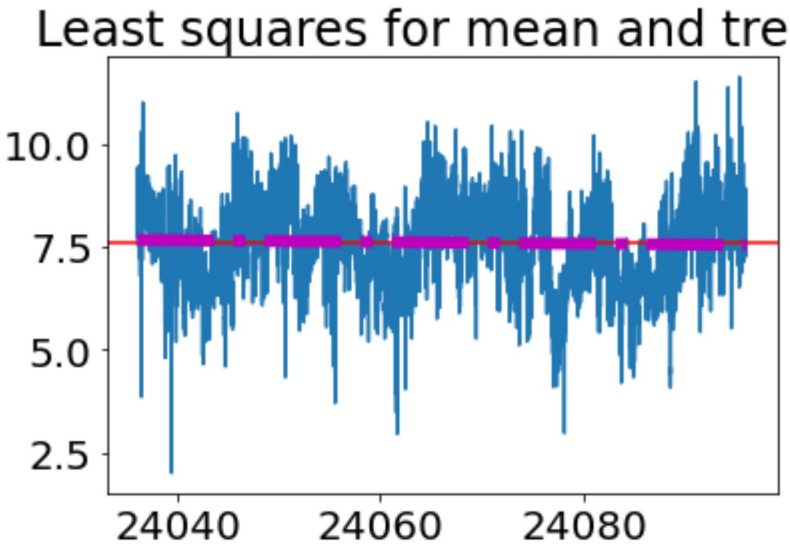

Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50

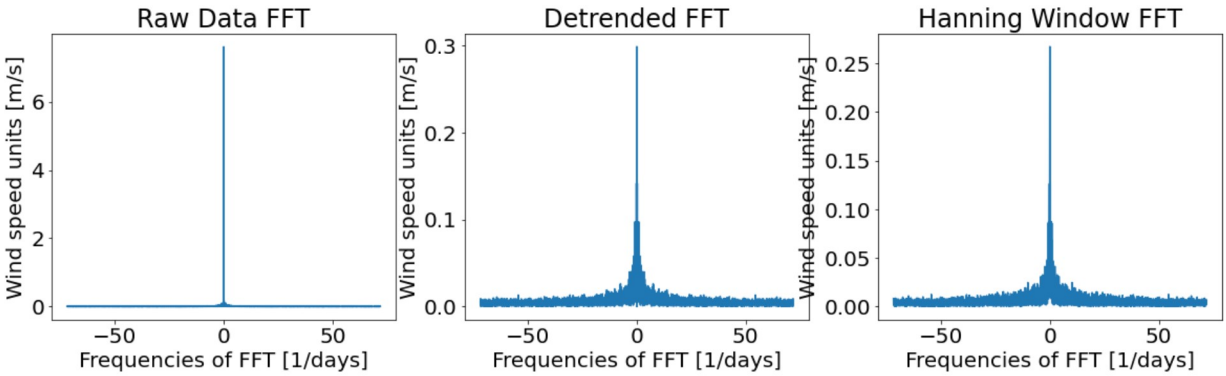

FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^{4}
10^{1}
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
$\Phi\ [\frac{(m^2 s^{-2})}{cyclesperday}]$
10^{-1}
10^{0}
10^{1}
10^{2}
FFT Frequencies [cycles per day]

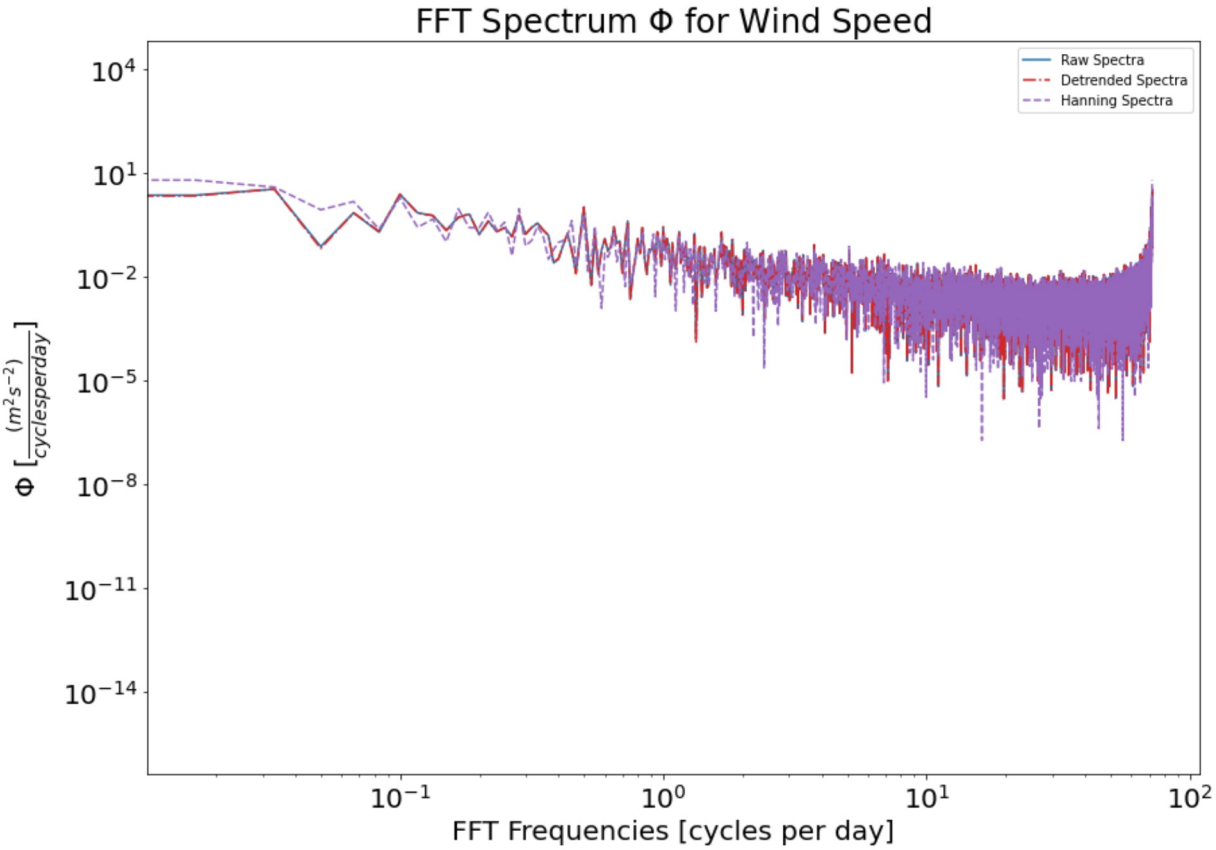

Least squares for mean and trend
12.5
10.0
7.5
5.0
2.5
24020
24040
24060

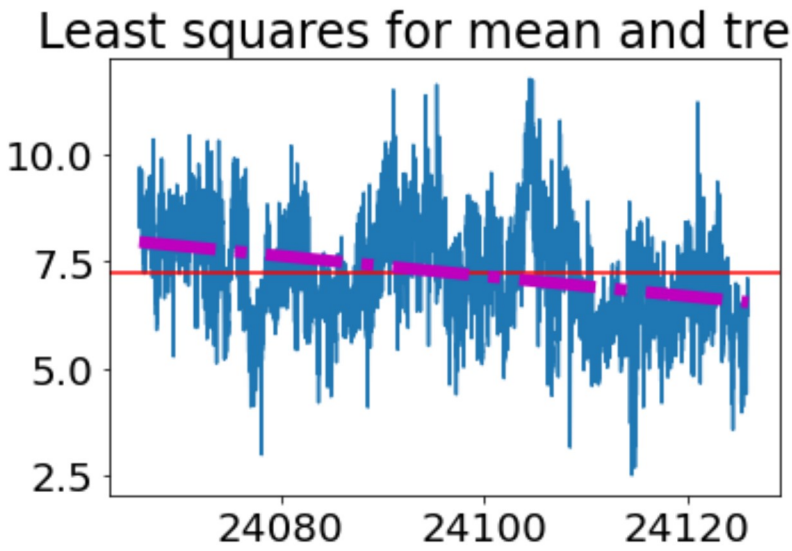

Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50

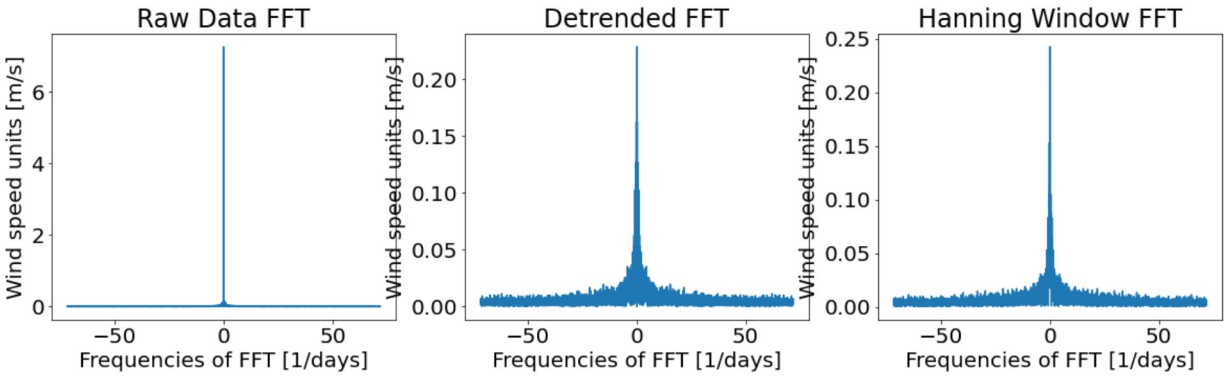

FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^{2}
10^{-1}
10^{-4}
10^{-7}
10^{-10}
10^{-13}
10^{-16}
$\Phi\ [\frac{(m^2 s^{-2})}{cyclesperday}]$
10^{-1}
10^{0}
10^{1}
10^{2}
FFT Frequencies [cycles per day]

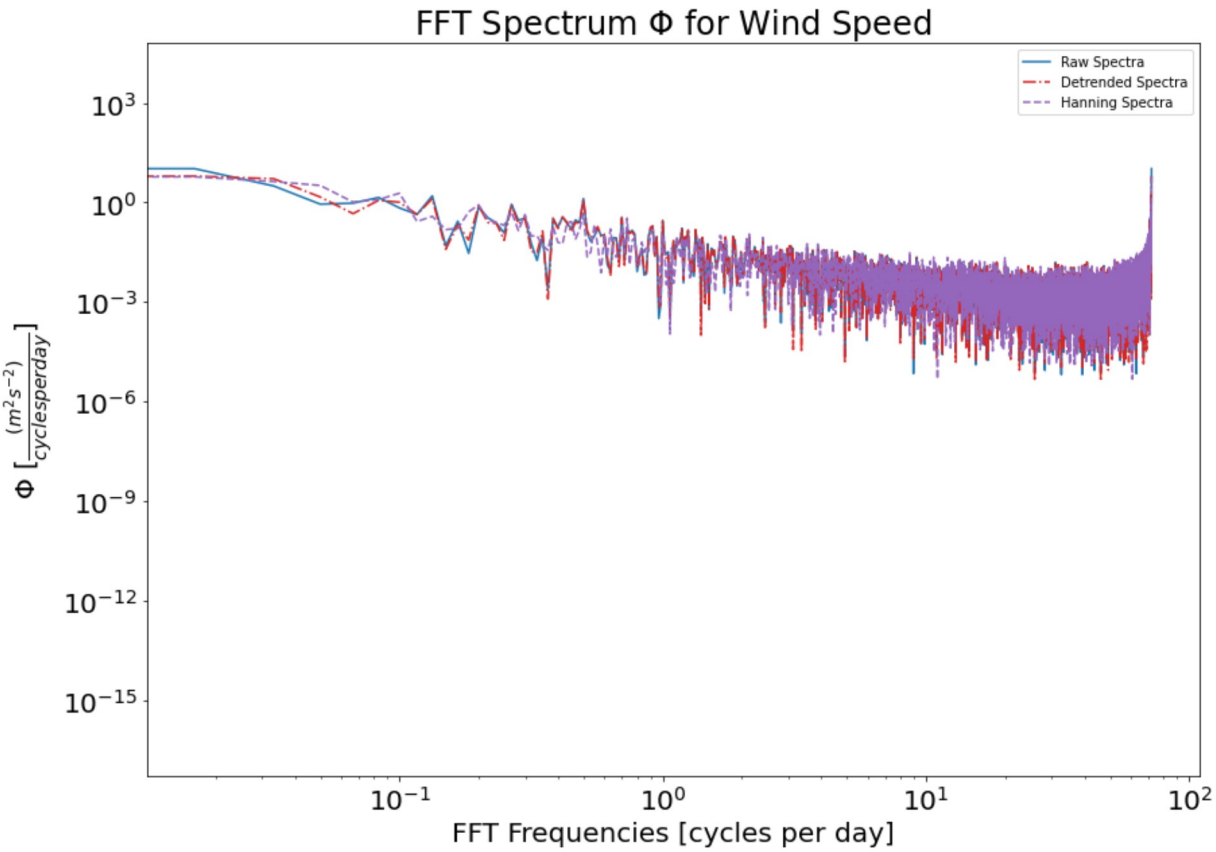

Least squares for mean and trend
10.0
7.5
5.0
2.5
24040
24060
24080

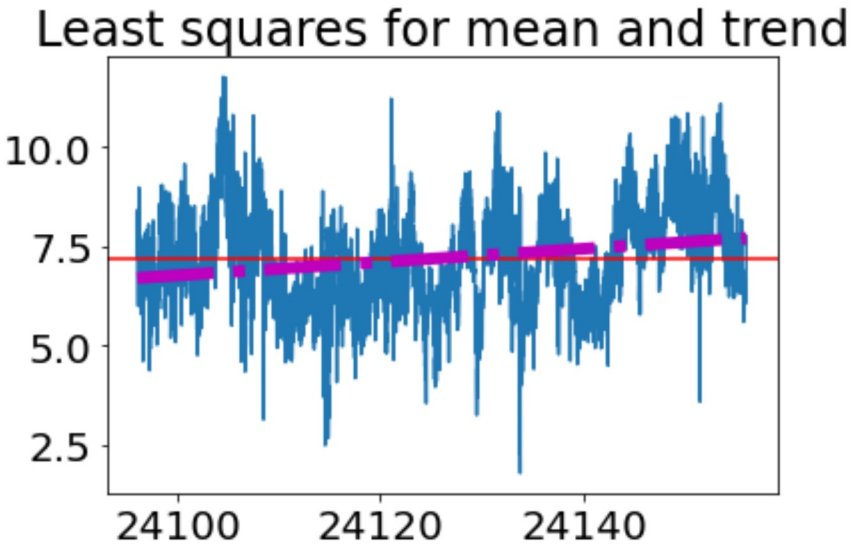

Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50


FFT Spectrum Φ for Wind Speed
Raw Spectra
Detrended Spectra
Hanning Spectra
10^4
10^1
10^{-2}
10^{-5}
10^{-8}
10^{-11}
10^{-14}
$\Phi\ [\frac{(m^2s^{-2})}{cyclesperday}]$
10^{-1}
10^0
10^1
10^2
FFT Frequencies [cycles per day]


Least squares for mean and trend
10.0
7.5
5.0
2.5
24080
24100
24120


Raw Data FFT
Detrended FFT
Hanning Window FFT
Wind speed units [m/s]
Frequencies of FFT [1/days]
−50
0
50





All looks good and as expected!

The raw data FFT has a peak at the mean, which overpowers the other frequencies.

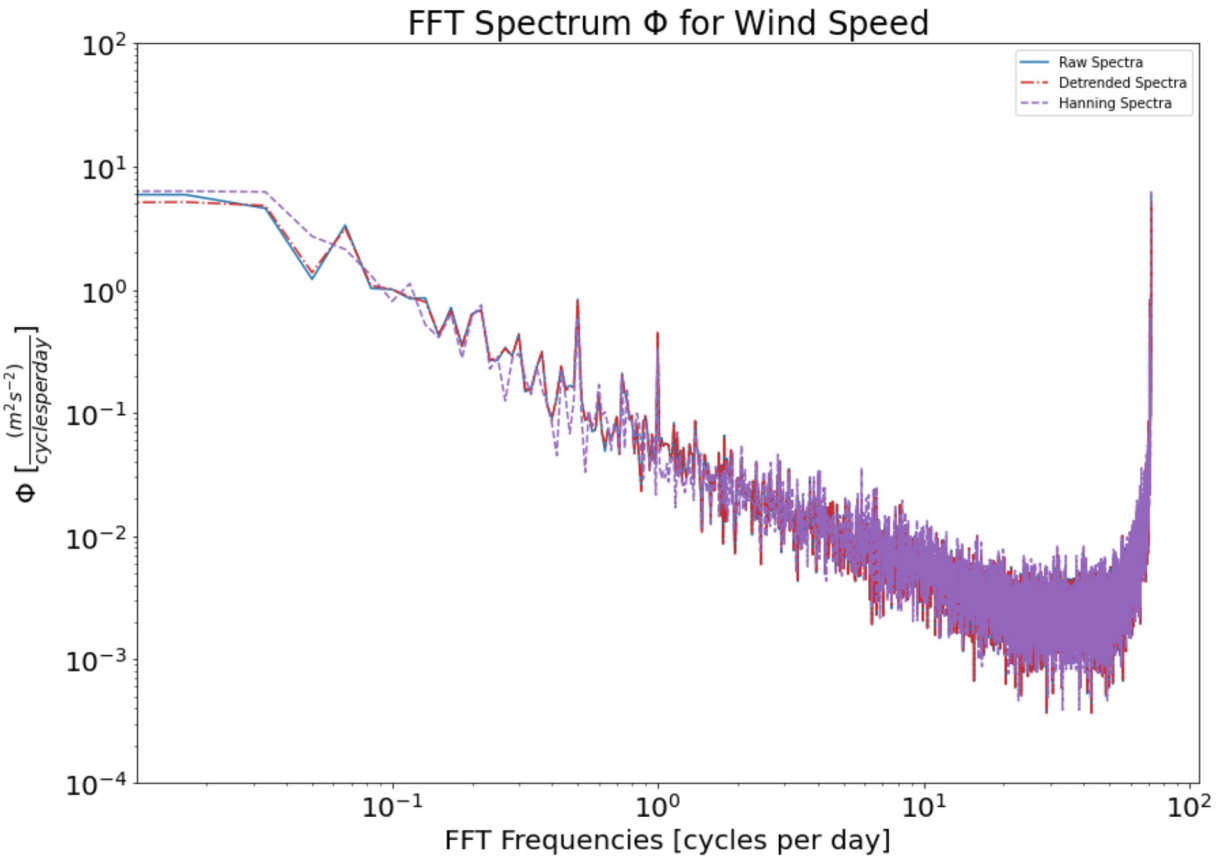
The demeaned & detrended FFT doesn't have such a sharp peak at the mean (as the raw data at least) and has slightly less energy in the spectrum than the raw data.

The Hanning windowed spectra shifts energy to be normalized correctly with the windowing, which we can see in the lower FFT and the lower spectrum (the peak near the Nyquist frequency/highest frequencies is lower than the raw or the detrended).

Next, average across my segments for the "coarser" / smoothed spectrum:

In [18]:

```
amp_r = np.nanmean(amp_raw,axis=0)
amp_d = np.nanmean(amp_dt,axis=0)
amp_h = np.nanmean(amp_han,axis=0)

# plot
fig,ax = plt.subplots(figsize=(14,10))
ax.plot(freqs,amp_r,label='Raw Spectra',color='tab:blue')
ax.plot(freqs,amp_d,label='Detrended Spectra',color='tab:red',linestyle='-.')
ax.plot(freqs,amp_h,label='Hanning Spectra',color='tab:purple',linestyle='--')
ax.set_xscale('log'); ax.set_yscale('log')
ax.set_ylim([10e-5, 10e1])
ax.set(xlabel=r'FFT Frequencies [cycles per day]',
       ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',
       title=r'FFT Spectrum $\Phi$ for Wind Speed')
ax.legend(fontsize=10)
plt.show()
```



In this case, we can see that detrending the data helped to remove the large peak at the mean and also decreased extremes in the power spectrum across the board (because in the raw spectrum, the mean is shifting energy into other spectra where it doesn't belong). The maximum spectral values are lower in the detrended data than in the raw data. The Hanning window especially helps to reallocate energy by redistributing the extra energy introduced from the side lobes introduced from a harsh-edged window.

In [22]:
```
# also: want to make sure we abidded by Parseval's theorem

# if I had more time, I would finish this! - but I don't :(

# would check whether that variance == the sum of the spectra
```

#4 Add uncertainty estimates to the 2015 wind speed spectra.

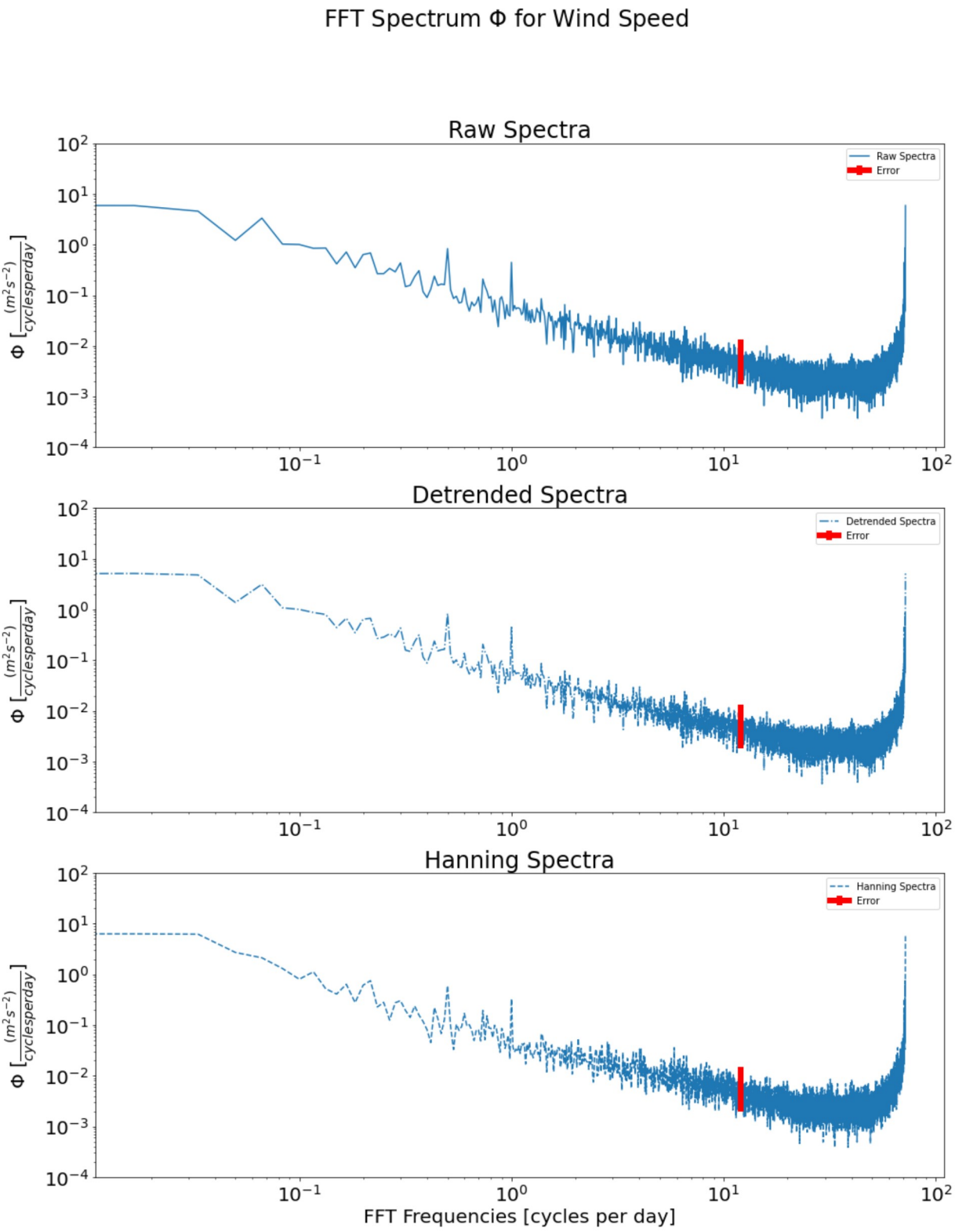
Indicate whether you think the uncertainty should differ for the 3 cases.

In [23]:
```
# calculate errors based on chi2 distribution

M = cycles # data segments --> 9
nu = 2*M # degrees of freedom

top = 1-0.05/2
bot = 0.05/2

err_low = nu/(scipy.stats.chi2.ppf(top, nu))
err_high = nu/(scipy.stats.chi2.ppf(bot, nu))
print(err_low,err_high)
```

```
0.5709504513509626 2.186922008537544
```

In [24]:

```
from scipy import stats

def ind_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return idx


# plot with errors
fig,axes = plt.subplots(3,1,figsize=(16,20))
plt.suptitle(r'FFT Spectrum $\Phi$ for Wind Speed')
amps = [amp_r,amp_d,amp_h]; labels=['Raw Spectra','Detrended Spectra','Hanning Spectr
for nn,ax in enumerate(axes):
    ax.plot(freqs,amps[nn],label=labels[nn],linestyle=styles[nn],zorder=1)
    # add error bars somewhere on plot (error will be the same at all frequencies)
    freq_value = 12  # plotting at frequency of a signal with 12 cycles per day
    vert = ind_nearest(freqs,freq_value)
    this_amp = amps[nn]; this_amp[vert]
    ax.errorbar(freq_value, this_amp[vert], yerr=[[err_low * this_amp[vert]], [err_hi
            color='red',linestyle='-',linewidth=6, label='Error',zorder=2)

    #ax.axvline(x=freqs[vert],ymin=err_low*amp_h[vert], ymax=err_high*amp_h[vert],col
    ax.set_xscale('log'); ax.set_yscale('log')
    ax.set(ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',
           title=labels[nn])
    ax.set_ylim([10e-5, 10e1])
    if nn == 2:
        ax.set(xlabel=r'FFT Frequencies [cycles per day]')
    ax.legend(fontsize=10,loc='upper right')
plt.show()
```



It looks like the error is roughly the same size as the spread in the spectra - that's good! That means that our χ^2 distribution holds.

Since we aren't directly changing the degrees of freedom (same number of segments in each spectral analysis), the magnitudes of the error between the different spectra should be the same - however, they should be transformed in the spectra space for whichever spectral curve they are applied to.

Make functions:

In [25]:

```
# making windowing function

def windowing(time=time15,data=wind15,tseg=60):
    # make sure time segments (tseg) is in units of days

    # number of tseg intervals encompassed in full dataset
    intervals = int(np.floor((time[-1]-time[0])/tseg))
    # with 50% overlaps, it will be doubled but 1 less (because start 50% and end 50%
    cycles = (intervals*2)-1

    num_samples = int(tseg/10 *60*24) # number of samples if taken every 10 min in 6(


    # can make arrays the correct size because we know how many 10 min samples will b
    # each array = (segment num, data within segment)
    data_segs = np.zeros((cycles,num_samples))
    time_segs = np.zeros((cycles,num_samples))

    ind = 0
    jump = int(num_samples/2);
    for nn in np.arange(0,cycles):
        # looping through each segment in which we will save data
        data_segs[nn,:] = data[ind:ind+num_samples]
        time_segs[nn,:] = time[ind:ind+num_samples]

        # now add to the ind (start index) and jump (number of samples in the segment
        ind = ind + jump

    return time_segs,data_segs
```

In [26]:

```
# making into a function !

def fft_analysis(data=wsegs,time=time_segs,demean=1,detrend=1,hanning=1):
    # specifications for inputs:
    # data_array should be size = (segments,data_in_segment); each data_in_segment co
    # time array = should be the same size as data_array, could adapt for just a 1D v
    # selecting whether to demean, detrend, hanning window, etc.

    # all segment lengths should be the same
    N = len(data[0,:])  # == num_samples
    deltat = np.diff(time)
    step = np.nanmean(deltat) # step size (in days)
    Nyq = 1/(2*step) # Nyquist frequency
    period = step*N # this is the entire period (in days)
    df = 1/period # fundamental frequency
    segs = len(data[:,0])
    M = segs

    # start workflow: analyze each segment individually

    # fft & spectra of interest to be averaged
    fft_oi = np.zeros((segs,N))
    amp_oi = np.zeros((segs,int(N/2)))


    for nn in np.arange(0,segs):
        segment = data[nn,:]
        segtime = time[nn,:]

        # demean & detrend

        # calculate mean and linear trend
        AA = np.array([np.ones(N), segtime]).T
        x = np.dot(np.linalg.inv(np.dot(AA.T, AA)), np.dot(AA.T, segment))
        mean = x[0]; trend = x[1]
        mymean = np.nanmean(segment)

        if demean==1:
            segment = segment-mean # demean by calculated mean

        if detrend==1:
            segment = segment-trend*segtime # detrend from calculated trend

        if hanning==1:
            hanwin = np.cos(np.pi*segtime / period)**2 # calculate hanning window
            segment = segment*hanwin*np.sqrt(8/3) # normalize hanning window by sqrt(

        # compute each segment spectrum
        # fourier
        fft = scipy.fft.fft(segment);
        freq = scipy.fft.fftfreq(N,step)
        freq = scipy.fft.fftshift(freq)
        fftplot = scipy.fft.fftshift(fft)
        norm_fft = 1.0/N * np.abs(fftplot)

        even_idx = np.arange(0,N,2)
        amp = abs(fft[even_idx])**2 # even N
```

```
        # corresponding frequencies based on data length
        freqs = np.arange(0,Nyq,df) # frequency vector, in [1/day], goes from 0 to Ny

        # do the steps for the spectrum:
        amp = amp/(N**2) # correct normalization
        amp = amp*2 # account for discarded redundant complex FFT coefficients
        amp = amp/df

        fft_oi[nn,:] = norm_fft
        amp_oi[nn,:] = amp

    avg_fft = np.nanmean(fft_oi,axis=0)
    tot_amp = np.nanmean(amp_oi,axis=0)

    nu = 2*M # degrees of freedom
    top = 1-0.05/2; bot = 0.05/2
    err_low = nu/(scipy.stats.chi2.ppf(top, nu)); err_high = nu/(scipy.stats.chi2.pp

    return freqs, avg_fft, tot_amp, err_low, err_high
```

#5 Compute the wind speed spectrum using data from all three data files.

Do this just using the detrended and Hanning-windowed records. Be sure to compute the uncertainty estimate. Overlay the multi-year spectrum over the 2015

In [28]:
```
link15 = links[0]; link16 = links[1]; link17 = links[2]


# downloading
[t15,w15,u15,v15] = download_winds(link15)
[t16,w16,u16,v16] = download_winds(link16)
[t17,w17,u17,v17] = download_winds(link17)

# windowing
[tsegs15,wsegs15] = windowing(t15,w15,tseg=60)
[tsegs16,wsegs16] = windowing(t16,w16,tseg=60)
[tsegs17,wsegs17] = windowing(t17,w17,tseg=60)
```

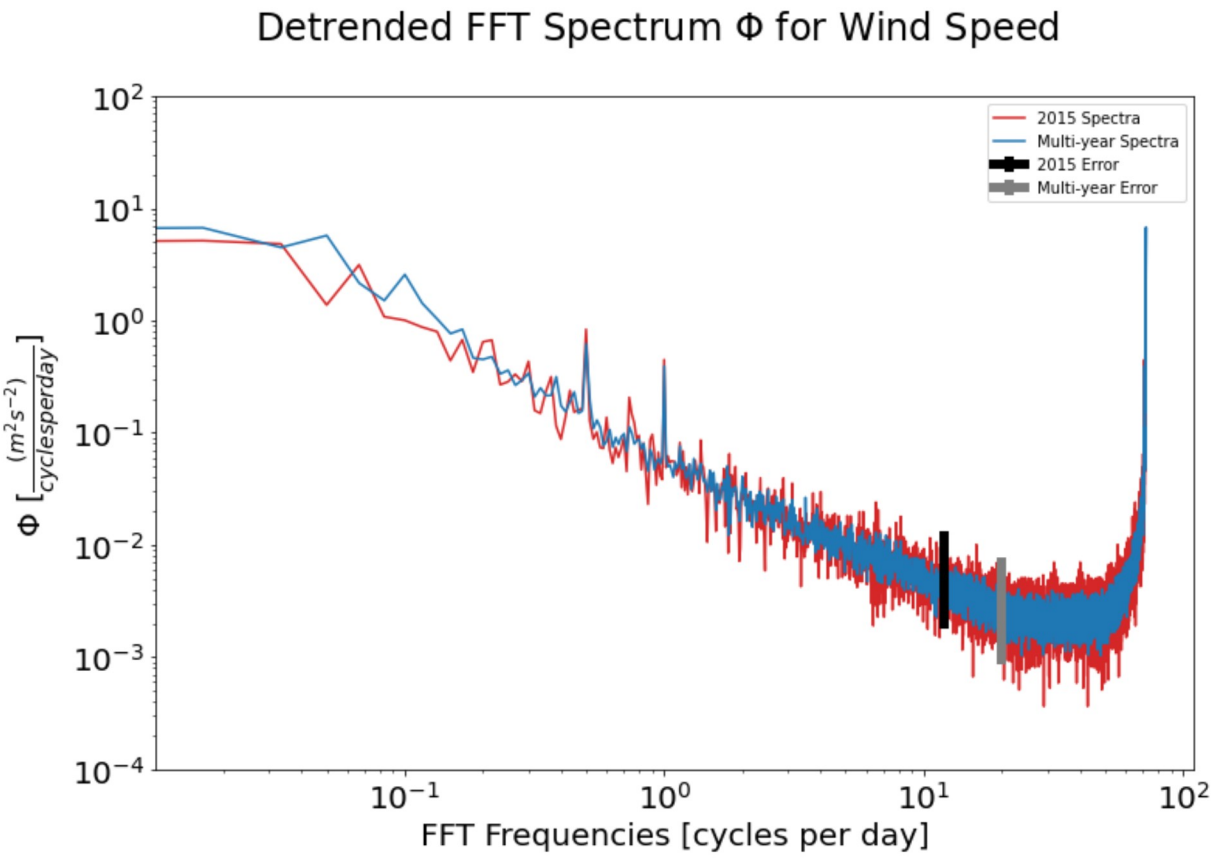
In [29]:
```
tsegs = np.concatenate([tsegs15,tsegs16,tsegs17])
wsegs = np.concatenate([wsegs15,wsegs16,wsegs17])
# check size to make sure they are stacking correctly:
print(tsegs.shape)
print(wsegs.shape)
```

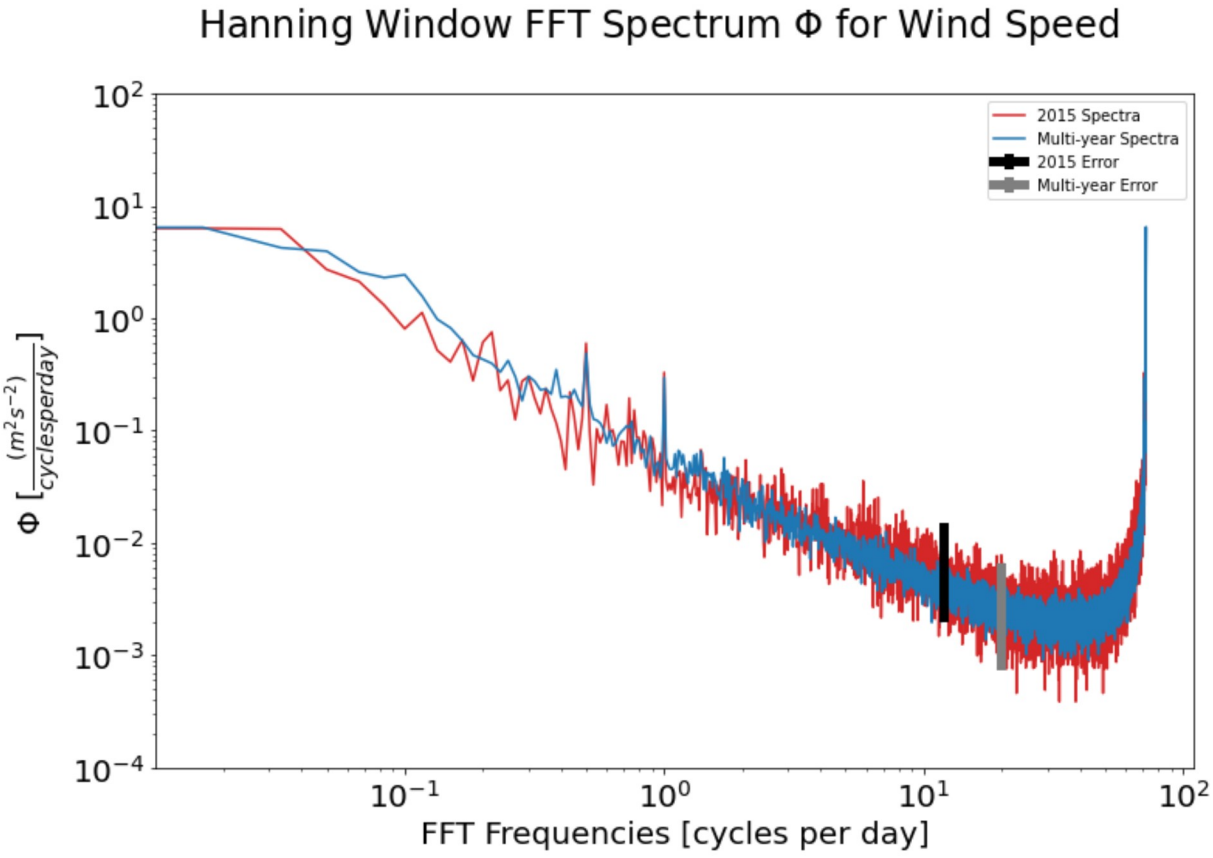
```
(31, 8640)
(31, 8640)
```

Looks good!

In [30]:

```
# finding spectra for multi-year spectra
[f,fft,amp,erl,erh] = fft_analysis(wsegs,tsegs,demean=1,detrend=1,hanning=0)
[fh,hfft,hamp,herl,herh] = fft_analysis(wsegs,tsegs,demean=1,detrend=1,hanning=1)

# plot detrended - multiyear v 2015
fig,ax = plt.subplots(1,1,figsize=(12,8))
plt.suptitle(r'Detrended FFT Spectrum $\Phi$ for Wind Speed')
ax.plot(freqs,amp_d,label='2015 Spectra',color='tab:red',zorder=1)
freq_value = 12  # plotting at frequency of a signal with 12 cycles per day
vert = ind_nearest(freqs,freq_value)
ax.errorbar(freq_value, amp_d[vert], yerr=[[err_low * amp_d[vert]], [err_high * amp_d
            color='black',linestyle='-',linewidth=6, label='2015 Error',zorder=3)

ax.plot(f,amp,label='Multi-year Spectra',color='tab:blue',zorder=2)
freq_value = 20 # plotting at frequency of a signal with 20 cpd - to get an offset fr
vert = ind_nearest(f,freq_value)
ax.errorbar(freq_value, amp[vert], yerr=[[erl * amp[vert]], [erh * amp[vert]]],
            color='tab:gray',linestyle='-',linewidth=6, alpha=1, label='Multi-year Er

ax.set_xscale('log'); ax.set_yscale('log')
ax.set(ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',xlabel=r'FFT Freq
ax.set_ylim([10e-5, 10e1])
ax.legend(fontsize=10,loc='upper right')
plt.show()


# plot hanning - multiyear v 2015
fig,ax = plt.subplots(1,1,figsize=(12,8))
plt.suptitle(r'Hanning Window FFT Spectrum $\Phi$ for Wind Speed')
ax.plot(freqs,amp_h,label='2015 Spectra',color='tab:red',zorder=1)
freq_value = 12  # plotting at frequency of a signal with 12 cycles per day
vert = ind_nearest(freqs,freq_value)
ax.errorbar(freq_value, amp_h[vert], yerr=[[err_low * amp_h[vert]], [err_high * amp_h
            color='black',linestyle='-',linewidth=6, label='2015 Error',zorder=3)

ax.plot(f,hamp,label='Multi-year Spectra',color='tab:blue',zorder=2)
freq_value = 20 # plotting at frequency of a signal with 20 cpd - to get an offset fr
vert = ind_nearest(f,freq_value)
# ax.axvline(freq_value,ymin=[herl * hamp[vert]],ymax=[herh * hamp[vert]],color='tab
ax.errorbar(freq_value, hamp[vert], yerr=[[herl * hamp[vert]], [herh * hamp[vert]]],
            color='tab:gray',linestyle='-',linewidth=6, alpha=1, label='Multi-year Er

ax.set_xscale('log'); ax.set_yscale('log')
ax.set(ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',xlabel=r'FFT Freq
ax.set_ylim([10e-5, 10e1])
ax.legend(fontsize=10,loc='upper right')
plt.show()
```





#6 Compare the spectra for wind speed, zonal wind, and meridional wind.

Use all 3 data files, and be sure to detrend and Hanning window.

Examine the peaks at the diurnal and semi-diurnal frequencies. How do they differ? Are the differences statistically significant? What do you hypothesize might account for differences?

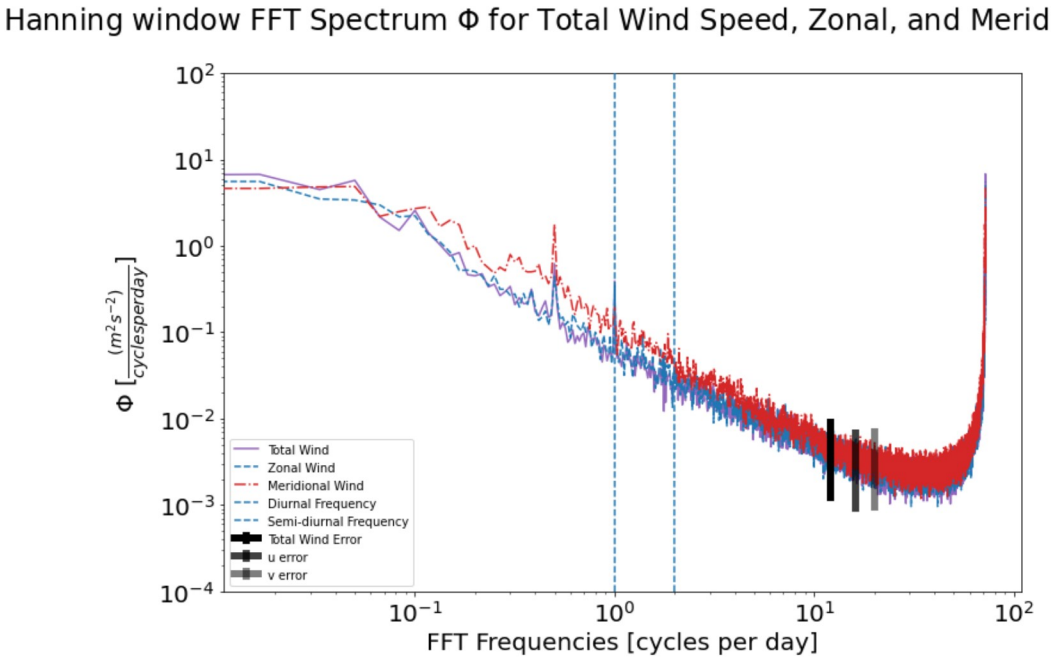
In [31]:

```
# already downloaded u and v for all years with previous function run

# windowing for u and v

# first u
[tsegs15,usegs15] = windowing(t15,u15,tseg=60)
[tsegs16,usegs16] = windowing(t16,u16,tseg=60)
[tsegs17,usegs17] = windowing(t17,u17,tseg=60)

# now v
[tsegs15,vsegs15] = windowing(t15,v15,tseg=60)
[tsegs16,vsegs16] = windowing(t16,v16,tseg=60)
[tsegs17,vsegs17] = windowing(t17,v17,tseg=60)

# concatenating datasets for u and v
tsegs = np.concatenate([tsegs15,tsegs16,tsegs17])
usegs = np.concatenate([usegs15,usegs16,usegs17])
vsegs = np.concatenate([vsegs15,vsegs16,vsegs17])

# check size to make sure they are stacking correctly:
print(tsegs.shape)
print(usegs.shape)
print(vsegs.shape)
```

```
(31, 8640)
(31, 8640)
(31, 8640)
```

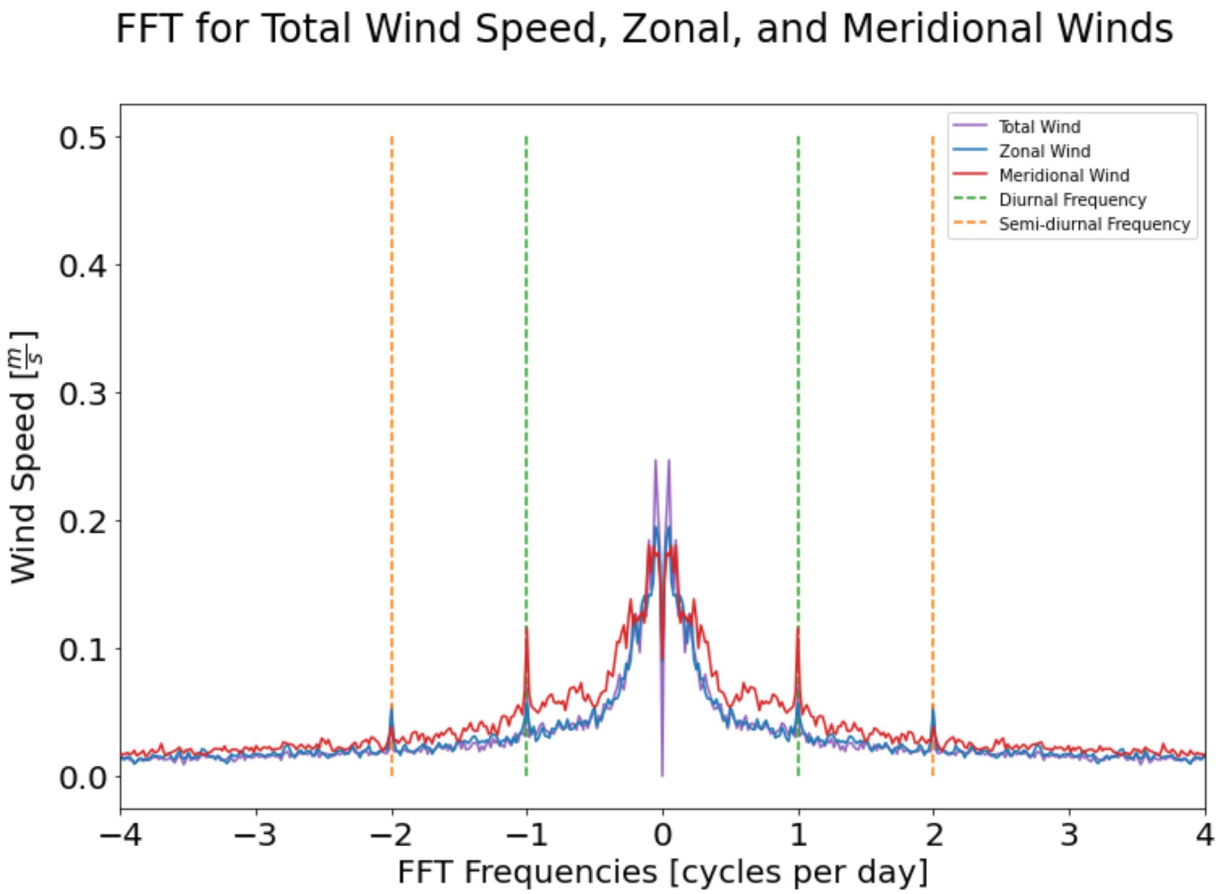
In [32]:

```
# finding spectra for multi-year u and v ( and w)
[uf,ufft,uamp,uerl,uerh] = fft_analysis(usegs,tsegs,demean=1,detrend=1,hanning=1)
[vf,vfft,vamp,verl,verh] = fft_analysis(vsegs,tsegs,demean=1,detrend=1,hanning=1)


# plot each spectra compared to each other!
# with their respective errors
fig,ax = plt.subplots(1,1,figsize=(12,8))
plt.suptitle(r'Hanning window FFT Spectrum $\Phi$ for Total Wind Speed, Zonal, and Me
ax.plot(f,amp,label='Total Wind',color='tab:purple',zorder=1)
freq_value = 12  # plotting at frequency of a signal with 12 cycles per day
vert = ind_nearest(f,freq_value)
ax.errorbar(freq_value, amp[vert], yerr=[[erl * amp[vert]], [erh * amp[vert]]],
            color='k',linestyle='-',linewidth=6, alpha=1, label='Total Wind Error',zo

ax.plot(uf,uamp,label='Zonal Wind',color='tab:blue',linestyle='--',zorder=2)
freq_value = 16  # plotting at frequency of a signal with 16 cycles per day
vert = ind_nearest(uf,freq_value)
ax.errorbar(freq_value, uamp[vert], yerr=[[uerl * uamp[vert]], [uerh * uamp[vert]]],
            color='k',linestyle='-',linewidth=6, alpha=0.75, label='u error',zorder=5

ax.plot(vf,vamp,label='Meridional Wind',color='tab:red',linestyle='-.',zorder=3)
freq_value = 20  # plotting at frequency of a signal with 20 cycles per day
vert = ind_nearest(vf,freq_value)
ax.errorbar(freq_value, vamp[vert], yerr=[[verl * vamp[vert]], [verh * vamp[vert]]],
            color='k',linestyle='-',linewidth=6, alpha=0.5, label='v error',zorder=6)

ax.axvline(x=1,label='Diurnal Frequency',linestyle='--',zorder=7) # once per day
ax.axvline(x=2,label='Semi-diurnal Frequency',linestyle='--',zorder=8) # twice per da
ax.set_xscale('log'); ax.set_yscale('log')
ax.set(ylabel=r'$\Phi$ [$ \frac{(m^2 s^{-2})}{ cycles per day } $]',xlabel=r'FFT Freq
ax.legend(fontsize=10,loc='lower left')
ax.set_ylim([10e-5, 10e1])
plt.show()
```



In [44]:

```
# compare the frequencies and peaks - therefore the FFT itself is easier to see for t

# plot each spectra compared to each other!
# with their respective errors
fig,ax = plt.subplots(1,1,figsize=(12,8))
plt.suptitle(r'FFT for Total Wind Speed, Zonal, and Meridional Winds')
ax.plot(freq,fft,label='Total Wind',color='tab:purple',zorder=1)
ax.plot(freq,ufft,label='Zonal Wind',color='tab:blue',zorder=2)
ax.plot(freq,vfft,label='Meridional Wind',color='tab:red',zorder=3)
ax.vlines(x=[-1,1],ymin=0,ymax=0.5,label='Diurnal Frequency',linestyle='--',color='ta
ax.vlines(x=[-2,2],ymin=0,ymax=0.5,label='Semi-diurnal Frequency',linestyle='--',colo
ax.set(ylabel=r'Wind Speed [$ \frac{m}{s} $]',xlabel=r'FFT Frequencies [cycles per da
ax.set_xlim([-4,4]) # signal with 4 cpd
ax.legend(fontsize=10,loc='upper right')
plt.show()
```



It looks like there is a bigger diurnal peak in the meridional winds, and a bigger semi-diurnal peak in the zonal winds. Depending on the location of the measurement station, we could make a guess about whether the east-west winds vs. north-south winds might be mediated by different process, eg. a daily heating of the land to induce a pressure differential that brings in north/south winds (yielding a bigger amplitude for the meridional peak) versus a east-west wind that could be induced two times per day from a land-sea temperature differential induced from upwelling from tides or a morning wind from the cold water and then afternoon wind once the water heats up more. I'm just making guesses right now, I don't know much about the atmosphere!

It's also interesting that wind speed doesn't have such obvious peaks at either of these peak points for u and v, even though wind speed would be the magnitude of the zonal and meridional components, $\sqrt{u^2 + v^2}$.

In [51]:
```
print(nc['LONGITUDE'][0])
print(nc['LATITUDE'][0])
print('seems like this is in the middle of the pacific ocean off the coast of South A
```

```
-110.0
-8.0
seems like this is in the middle of the pacific ocean off the coast of South America?
```

Pushed functions to git!