

Reading Fast Data and Timeseries analysis

This notebook focuses on decomposing the wind and temperature dataset to understand the various scales of motion observed in the dataset. We were not able to see the textbook co-spectral gap, but we do see some minor differences across time periods and we can possibly relate that to plume metrics through the morning hours of November 3rd.

```
In [29]: import xarray as xr
import pandas as pd
import numpy as np
import math
from matplotlib import pyplot as plt
import matplotlib.dates as mdates
from datetime import datetime
```

```
In [32]: # data files are large, make sure they load completely
file_path = 'data/isfs/isfs_geo_hr.savant_20181103_04.nc'#'data/isfs/isfs_geo_hr.savant_201811
from netCDF4 import Dataset

ds = Dataset(file_path)
ds
```

```
Out[32]: <class 'netCDF4.Dataset'>
root group (NETCDF3_CLASSIC data model, file format NETCDF3):
    history: Created: 2021-05-12 21:27:49 +0000

        NIDAS_version: v1.2-1464
        calibration_file_path: /h/eol/isfs/isfs/projects/SAVANT/ISFS/cal_files/QC/${SITE}:/h/eol/isfs/isfs/projects/SAVANT/ISFS/cal_files/QC:/h/eol/isfs/isfs/projects/SAVANT/ISFS/cal_files/QC/${SITE}:/h/eol/isfs/isfs/cal_files:/h/eol/isfs/isfs/projects/SAVANT/ISFS/cal_files/QC/$DSM
        dataset: hr_qc_geo_tiltcor
        dataset_description: High rate winds in geographic coordinates
        project_config: /h/eol/isfs/isfs/projects/SAVANT/ISFS/config/savant.xml;
        wind3d_horiz_coordinates: geographic
        file_length_seconds: 14400
        wind3d_horiz_rotation: 1
        wind3d_tilt_correction: 1
        dimensions(sizes): time(14400), sample(20), sample_10(10)
        variables(dimensions): int32 base_time(), float64 time(time), float32 P_1_5m_lconv(time,
sample), float32 P_1_5m_rel(time, sample), float32 P_20m_lconv(time, sample), float32 P_20m_r
el(time, sample), float32 Pirga_1_5m_init(time, sample), float32 Pirga_1_5m_lconv(time, sampl
e), float32 Pirga_1_5m_rel(time, sample), float32 Pirga_1_5m_uconv(time, sample), float32 Pir
ga_20m_lconv(time, sample), float32 Pirga_20m_rel(time, sample), float32 Pirga_6m_init(time,
sample), float32 Pirga_6m_lconv(time, sample), float32 Pirga_6m_rel(time, sample), float32 Pi
rga_6m_uconv(time, sample), float32 Tirga_1_5m_init(time, sample), float32 Tirga_1_5m_lconv(t
ime, sample), float32 Tirga_1_5m_rel(time, sample), float32 Tirga_1_5m_uconv(time, sample), f
loat32 Tirga_20m_lconv(time, sample), float32 Tirga_20m_rel(time, sample), float32 Tirga_6m_i
nit(time, sample), float32 Tirga_6m_lconv(time, sample), float32 Tirga_6m_rel(time, sample),
float32 Tirga_6m_uconv(time, sample), float32 co2_1_5m_init(time, sample), float32 co2_1_5m_l
conv(time, sample), float32 co2_1_5m_rel(time, sample), float32 co2_1_5m_uconv(time, sample),
float32 co2_20m_lconv(time, sample), float32 co2_20m_rel(time, sample), float32 co2_6m_init(t
ime, sample), float32 co2_6m_lconv(time, sample), float32 co2_6m_rel(time, sample), float32 c
o2_6m_uconv(time, sample), float32 diagbits_1_5m_init(time, sample), float32 diagbits_1_5m_lc
onv(time, sample), float32 diagbits_1_5m_rel(time, sample), float32 diagbits_1_5m_uconv(time,
sample), float32 diagbits_10m_init(time, sample), float32 diagbits_10m_lconv(time, sample), f
loat32 diagbits_10m_rel(time, sample), float32 diagbits_10m_uconv(time, sample), float32 diag
bits_15m_lconv(time, sample), float32 diagbits_15m_rel(time, sample), float32 diagbits_20m_lc
onv(time, sample), float32 diagbits_20m_rel(time, sample), float32 diagbits_3m_init(time, sam
ple), float32 diagbits_3m_lconv(time, sample), float32 diagbits_3m_rel(time, sample), float32
diagbits_3m_uconv(time, sample), float32 diagbits_4_5m_init(time, sample), float32 diagbits_4_
5m_lconv(time, sample), float32 diagbits_4_5m_rel(time, sample), float32 diagbits_4_5m_uconv(
time, sample), float32 diagbits_6m_init(time, sample), float32 diagbits_6m_lconv(time, sampl
e), float32 diagbits_6m_rel(time, sample), float32 diagbits_6m_uconv(time, sample), float32 d
iagbits_8_5m_lconv(time, sample), float32 diagbits_8_5m_rel(time, sample), float32 diagbits_a
1_1_5m_lconv(time, sample), float32 diagbits_a1_1_5m_rel(time, sample), float32 diagbits_a1_1_
5m_uconv(time, sample), float32 diagbits_a2_1_5m_lconv(time, sample), float32 diagbits_a2_1_
5m_rel(time, sample), float32 diagbits_a2_1_5m_uconv(time, sample), float32 h2o_1_5m_init(ti
me, sample), float32 h2o_1_5m_lconv(time, sample), float32 h2o_1_5m_rel(time, sample), float32
h2o_1_5m_uconv(time, sample), float32 h2o_20m_lconv(time, sample), float32 h2o_20m_rel(time,
sample), float32 h2o_6m_init(time, sample), float32 h2o_6m_lconv(time, sample), float32 h2o_6
m_rel(time, sample), float32 h2o_6m_uconv(time, sample), float32 irgadiag_1_5m_init(time, sam
ple), float32 irgadiag_1_5m_lconv(time, sample), float32 irgadiag_1_5m_rel(time, sample), flo
at32 irgadiag_1_5m_uconv(time, sample), float32 irgadiag_20m_lconv(time, sample), float32 irg
adiag_20m_rel(time, sample), float32 irgadiag_6m_init(time, sample), float32 irgadiag_6m_lcon
v(time, sample), float32 irgadiag_6m_rel(time, sample), float32 irgadiag_6m_uconv(time, sampl
e), float32 ldiag_1_5m_init(time, sample), float32 ldiag_1_5m_lconv(time, sample), float32 ld
iag_1_5m_rel(time, sample), float32 ldiag_1_5m_uconv(time, sample), float32 ldiag_10m_init(ti
me, sample), float32 ldiag_10m_lconv(time, sample), float32 ldiag_10m_rel(time, sample), flo
at32 ldiag_10m_uconv(time, sample), float32 ldiag_15m_lconv(time, sample), float32 ldiag_15m_r
el(time, sample), float32 ldiag_20m_lconv(time, sample), float32 ldiag_20m_rel(time, sample),
float32 ldiag_3m_init(time, sample), float32 ldiag_3m_lconv(time, sample), float32 ldiag_3m_r
el(time, sample), float32 ldiag_3m_uconv(time, sample), float32 ldiag_4_5m_init(time, sampl
e), float32 ldiag_4_5m_lconv(time, sample), float32 ldiag_4_5m_rel(time, sample), float32 ldi
ag_4_5m_uconv(time, sample), float32 ldiag_6m_init(time, sample), float32 ldiag_6m_lconv(tim
```

```

e, sample), float32 ldiag_6m_rel(time, sample), float32 ldiag_6m_uconv(time, sample), float32
ldiag_8_5m_lconv(time, sample), float32 ldiag_8_5m_rel(time, sample), float32 ldiag_a1_1_5m_l
conv(time, sample), float32 ldiag_a1_1_5m_rel(time, sample), float32 ldiag_a1_1_5m_uconv(tim
e, sample), float32 ldiag_a2_1_5m_lconv(time, sample), float32 ldiag_a2_1_5m_rel(time, sampl
e), float32 ldiag_a2_1_5m_uconv(time, sample), float32 tc_1_5m_init(time, sample), float32 tc
_1_5m_lconv(time, sample), float32 tc_1_5m_rel(time, sample), float32 tc_1_5m_uconv(time, sam
ple), float32 tc_10m_init(time, sample), float32 tc_10m_lconv(time, sample), float32 tc_10m_r
el(time, sample), float32 tc_10m_uconv(time, sample), float32 tc_15m_lconv(time, sample), flo
at32 tc_15m_rel(time, sample), float32 tc_20m_lconv(time, sample), float32 tc_20m_rel(time, s
ample), float32 tc_3m_init(time, sample), float32 tc_3m_lconv(time, sample), float32 tc_3m_re
l(time, sample), float32 tc_3m_uconv(time, sample), float32 tc_4_5m_init(time, sample), float
32 tc_4_5m_lconv(time, sample), float32 tc_4_5m_rel(time, sample), float32 tc_4_5m_uconv(tim
e, sample), float32 tc_6m_init(time, sample), float32 tc_6m_lconv(time, sample), float32 tc_6
m_rel(time, sample), float32 tc_6m_uconv(time, sample), float32 tc_8_5m_lconv(time, sample),
float32 tc_8_5m_rel(time, sample), float32 tc_a1_1_5m_lconv(time, sample), float32 tc_a1_1_5m
_rel(time, sample), float32 tc_a1_1_5m_uconv(time, sample), float32 tc_a2_1_5m_lconv(time, sa
mple), float32 tc_a2_1_5m_rel(time, sample), float32 tc_a2_1_5m_uconv(time, sample), float32
u_1_5m_init(time, sample), float32 u_1_5m_lconv(time, sample), float32 u_1_5m_rel(time, sampl
e), float32 u_1_5m_uconv(time, sample), float32 u_10m_init(time, sample), float32 u_10m_lconv
(time, sample), float32 u_10m_rel(time, sample), float32 u_10m_uconv(time, sample), float32 u
_15m_lconv(time, sample), float32 u_15m_rel(time, sample), float32 u_20m_lconv(time, sample),
float32 u_20m_rel(time, sample), float32 u_3m_init(time, sample), float32 u_3m_lconv(time, sa
mple), float32 u_3m_rel(time, sample), float32 u_3m_uconv(time, sample), float32 u_4_5m_init
(time, sample), float32 u_4_5m_lconv(time, sample), float32 u_4_5m_rel(time, sample), float32
u_4_5m_uconv(time, sample), float32 u_6m_init(time, sample), float32 u_6m_lconv(time, sampl
e), float32 u_6m_rel(time, sample), float32 u_6m_uconv(time, sample), float32 u_8_5m_lconv(ti
me, sample), float32 u_8_5m_rel(time, sample), float32 u_a1_1_5m_lconv(time, sample), float32
u_a1_1_5m_rel(time, sample), float32 u_a1_1_5m_uconv(time, sample), float32 u_a2_1_5m_lconv(t
ime, sample), float32 u_a2_1_5m_rel(time, sample), float32 u_a2_1_5m_uconv(time, sample), flo
at32 v_1_5m_init(time, sample), float32 v_1_5m_lconv(time, sample), float32 v_1_5m_rel(time,
sample), float32 v_1_5m_uconv(time, sample), float32 v_10m_init(time, sample), float32 v_10m_
lconv(time, sample), float32 v_10m_rel(time, sample), float32 v_10m_uconv(time, sample), flo
at32 v_15m_lconv(time, sample), float32 v_15m_rel(time, sample), float32 v_20m_lconv(time, sam
ple), float32 v_20m_rel(time, sample), float32 v_3m_init(time, sample), float32 v_3m_lconv(ti
me, sample), float32 v_3m_rel(time, sample), float32 v_3m_uconv(time, sample), float32 v_4_5m_
init(time, sample), float32 v_4_5m_lconv(time, sample), float32 v_4_5m_rel(time, sample), fl
oat32 v_4_5m_uconv(time, sample), float32 v_6m_init(time, sample), float32 v_6m_lconv(time, s
ample), float32 v_6m_rel(time, sample), float32 v_6m_uconv(time, sample), float32 v_8_5m_lco
nv(time, sample), float32 v_8_5m_rel(time, sample), float32 v_a1_1_5m_lconv(time, sample), flo
at32 v_a1_1_5m_rel(time, sample), float32 v_a1_1_5m_uconv(time, sample), float32 v_a2_1_5m_lc
onv(time, sample), float32 v_a2_1_5m_rel(time, sample), float32 v_a2_1_5m_uconv(time, sampl
e), float32 w_1_5m_init(time, sample), float32 w_1_5m_lconv(time, sample), float32 w_1_5m_rel
(time, sample), float32 w_1_5m_uconv(time, sample), float32 w_10m_init(time, sample), float32
w_10m_lconv(time, sample), float32 w_10m_rel(time, sample), float32 w_10m_uconv(time, sampl
e), float32 w_15m_lconv(time, sample), float32 w_15m_rel(time, sample), float32 w_20m_lconv(t
ime, sample), float32 w_20m_rel(time, sample), float32 w_3m_init(time, sample), float32 w_3m_
lconv(time, sample), float32 w_3m_rel(time, sample), float32 w_3m_uconv(time, sample), float3
2 w_4_5m_init(time, sample), float32 w_4_5m_lconv(time, sample), float32 w_4_5m_rel(time, sam
ple), float32 w_4_5m_uconv(time, sample), float32 w_6m_init(time, sample), float32 w_6m_lconv
(time, sample), float32 w_6m_rel(time, sample), float32 w_6m_uconv(time, sample), float32 w_8
_5m_lconv(time, sample), float32 w_8_5m_rel(time, sample), float32 w_a1_1_5m_lconv(time, samp
le), float32 w_a1_1_5m_rel(time, sample), float32 w_a1_1_5m_uconv(time, sample), float32 w_a2
_1_5m_lconv(time, sample), float32 w_a2_1_5m_rel(time, sample), float32 w_a2_1_5m_uconv(time,
sample), float32 Status_0_2m_init(time, sample_10), float32 Status_0_2m_lconv(time, sample_1
0), float32 Status_0_2m_rel(time, sample_10), float32 Status_0_2m_uconv(time, sample_10), flo
at32 Tc_0_2m_init(time, sample_10), float32 Tc_0_2m_lconv(time, sample_10), float32 Tc_0_2m_r
el(time, sample_10), float32 Tc_0_2m_uconv(time, sample_10), float32 U_0_2m_init(time, sampl
e_10), float32 U_0_2m_lconv(time, sample_10), float32 U_0_2m_rel(time, sample_10), float32 U_0
_2m_uconv(time, sample_10), float32 V_0_2m_init(time, sample_10), float32 V_0_2m_lconv(time,
sample_10), float32 V_0_2m_rel(time, sample_10), float32 V_0_2m_uconv(time, sample_10), flo
at32 P_1_5m_p1(time), float32 P_1_5m_p2(time), float32 P_1_5m_p3(time), float32 P_1_5m_p4(tim
e), float32 P_1_5m_p5(time), float32 P_1_5m_p6(time), float32 RH_0_2m_init(time), float32 RH_

```

```
0_2m_lconv(time), float32 RH_0_2m_rel(time), float32 RH_0_2m_uconv(time), float32 RH_1_5m_init(time), float32 RH_1_5m_lconv(time), float32 RH_1_5m_rel(time), float32 RH_15m_lconv(time), float32 RH_15m_rel(time), float32 RH_20m_lconv(time), float32 RH_20m_rel(time), float32 RH_4_5m_init(time), float32 RH_4_5m_lconv(time), float32 RH_4_5m_rel(time), float32 RH_4_5m_uconv(time), float32 RH_8_5m_lconv(time), float32 RH_8_5m_rel(time), float32 RH_a1_1_5m_lconv(time), float32 RH_a1_1_5m_rel(time), float32 RH_a1_1_5m_uconv(time), float32 RH_a2_1_5m_lconv(time), float32 RH_a2_1_5m_rel(time), float32 RH_a2_1_5m_uconv(time), float32 T_0_2m_init(time), float32 T_0_2m_lconv(time), float32 T_0_2m_rel(time), float32 T_0_2m_uconv(time), float32 T_1_5m_init(time), float32 T_1_5m_lconv(time), float32 T_1_5m_rel(time), float32 T_1_5m_uconv(time), float32 T_10m_init(time), float32 T_10m_uconv(time), float32 T_15m_lconv(time), float32 T_15m_rel(time), float32 T_20m_lconv(time), float32 T_20m_rel(time), float32 T_4_5m_init(time), float32 T_4_5m_lconv(time), float32 T_4_5m_rel(time), float32 T_8_5m_uconv(time), float32 T_8_5m_rel(time), float32 T_a1_1_5m_lconv(time), float32 T_a1_1_5m_rel(time), float32 T_a1_1_5m_uconv(time), float32 T_a2_1_5m_lconv(time), float32 T_a2_1_5m_rel(time), float32 T_a2_1_5m_uconv(time)
```

groups:

```
In [33]: print(f"dataset has {ds.dimensions['time'].size/60/60} hours of data")
print(f"sampling at {ds.dimensions['sample'].size} per second")
```

```
dataset has 4.0 hours of data
sampling at 20 per second
```

```
In [34]: # checking out variables. I see long and short name and its shape of 144000 rows at 20 samples
ds.variables['tc_10m_uconv']
```

```
Out[34]: <class 'netCDF4.Variable'>
float32 tc_10m_uconv(time, sample)
    _FillValue: 1e+37
    long_name: Virtual air temperature from speed of sound, CSAT3
    short_name: tc.10m.uconv
    units: degC
unlimited dimensions: time
current shape = (14400, 20)
filling on
```

```
In [35]: ds.variables['v_20m_rel']#[:][0]
```

```
Out[35]: <class 'netCDF4.Variable'>
float32 v_20m_rel(time, sample)
    _FillValue: 1e+37
    long_name: Wind V component, CSAT3
    short_name: v.20m.rel
    units: m/s
unlimited dimensions: time
current shape = (14400, 20)
filling on
```

```
In [36]: # Looking at time dimensions of the data. There are time stamps, but there are also 20 samples
ds.dimensions
```

```
Out[36]: {'time': '<class \'netCDF4.Dimension\'>' (unlimited): name = 'time', size = 14400,
'sample': '<class \'netCDF4.Dimension\'>': name = 'sample', size = 20,
'sample_10': '<class \'netCDF4.Dimension\'>': name = 'sample_10', size = 10}
```

```
In [38]: # Break the data into u and v components and combine to form windspeed.
v_samples_2d=ds.variables['v_10m_uconv'][:]
u_samples_2d=ds.variables['u_10m_uconv'][:]
tc_samples_2d=ds.variables['tc_10m_rel'][:]

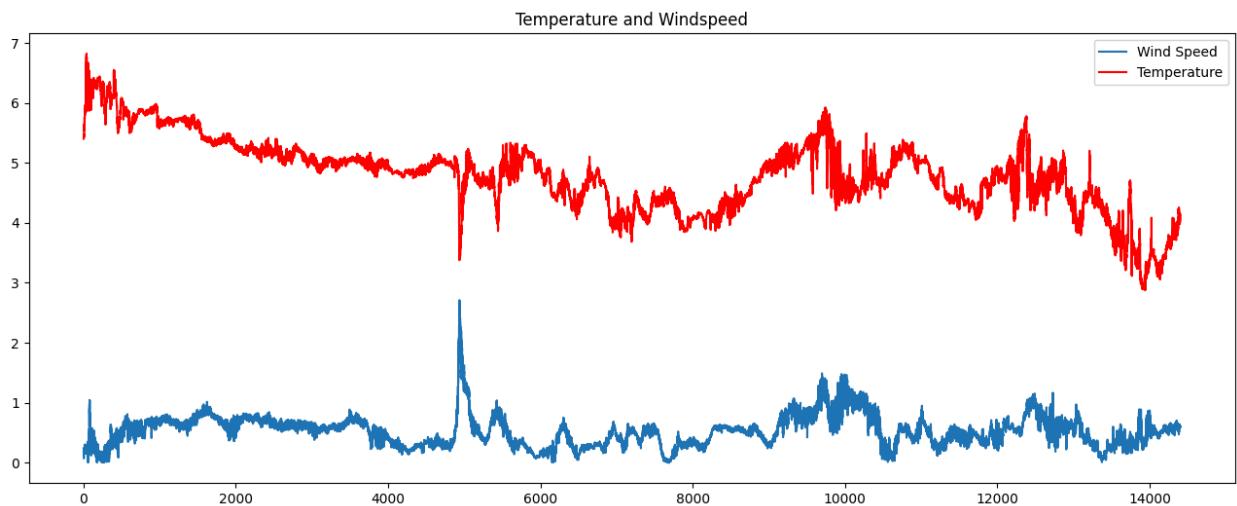
v_time_series_1d = v_samples_2d.flatten()
u_time_series_1d = u_samples_2d.flatten()
```

```
tc_time_series_1d = tc_samples_2d.flatten()
U = np.hypot(u_time_series_1d, v_time_series_1d)
```

```
In [39]: time_1s = ds.variables['time'][:]
num_samples = v_samples_2d.shape[0] * v_samples_2d.shape[1]
start_time = time_1s[0]
end_time = time_1s[0] + 14400 #time_1s[-1] +1.0
time_20hz = np.arange(start_time, end_time, 0.05)
len(time_20hz)
time_1s
```

```
Out[39]: masked_array(data=[5.00000e-01, 1.50000e+00, 2.50000e+00, ...,
                           1.43975e+04, 1.43985e+04, 1.43995e+04],
                         mask=False,
                         fill_value=1e+20)
```

```
In [42]: plt.figure(figsize=(16, 6))
plt.plot(time_20hz, U, label='Wind Speed')
plt.plot(time_20hz, tc_time_series_1d, label='Temperature', color='red')
plt.title('Temperature and Windspeed')
plt.legend()
plt.show()
```



```
In [43]: # finding wind fields
wind_fields = [var for var in ds.variables.keys() if hasattr(ds.variables[var], 'long_name') and
# wind_fields # not necessary
```

```
In [44]: # finding temp fields for heat flux
temperature_fields = [var for var in ds.variables.keys() if hasattr(ds.variables[var], 'long_name') and
# temperature_fields
```

```
In [45]: # Define the start and stop datetimes
start_time = np.datetime64('2025-01-01T00:00:00')
stop_time = np.datetime64('2025-01-01T00:00:01') # One second later for demonstration

# Define the step size as 0.05 seconds
step = np.timedelta64(50, 'ms') # 50 milliseconds = 0.05 seconds

# Create the array of datetimes
datetime_array = np.arange(start_time, stop_time, step)
```

```
In [48]: ds.variables['time'].units[0:] # we can see the series starts at 04:00 UTC.
```

```
Out[48]: 'seconds since 2018-11-03 04:00:00 00:00'
```

```
In [15]: # create a function for reading these fast files...
```

```
def read_fastdata(file):
    """
    Read high-frequency data from a netCDF file into a pandas dataframe.
    This is developed specifically for the SAVANT wind data I have available.
    This data is collected at 4-hour increments, with a sampling rate of 20 Hz (20 times per s
    I am interested in wind and temperature data to decompose the heat flux.
    """

    with Dataset(file, 'r') as ds:
        time_units = ds.variables['time'].units[14:]
        time_1s = ds.variables['time'][:]
        num_intervals = ds.dimensions['time'].size    # should be 14400, for 4 hours in seconds
        samples_per_interval = ds.dimensions['sample'].size    # should be 20 for 20 hertz or 2
        total_samples = num_intervals * samples_per_interval

        delta_t = ds.variables['time'].__dict__['interval(sec)']/samples_per_interval

        start_time = time_1s[0]
        end_time = start_time + 14400

        time_20hz = np.arange(start_time, end_time, delta_t)
        print(len(time_20hz))

        # extract wind data
        data_dict = {}
        wind_fields = [var for var in ds.variables.keys() if hasattr(ds.variables[var], 'long_
                           and 'wind' in ds.variables[var].long_name.lower())]
        for field in wind_fields:
            if ds.variables[field].shape==(14400, 20):
                samples_2d=ds.variables[field][:]
                time_series_1d = samples_2d.flatten().data
                data_dict[field]= time_series_1d
            else:
                print(f'{field} not used. Incorrect length.')
        temperature_fields = [var for var in ds.variables.keys() if hasattr(ds.variables[var], 'long_
                           and 'temp' in ds.variables[var].long_name.lower())]
        for field in temperature_fields:
            if ds.variables[field].shape==(14400, 20):
                samples_2d=ds.variables[field][:]
                time_series_1d = samples_2d.flatten().data
                data_dict[field]= time_series_1d
            else:
                print(f'{field} not used. Incorrect length.')

        df = pd.DataFrame(data_dict, index=time_20hz)

        try:
            df.index = pd.to_datetime(df.index, unit='s', origin=time_units)
        except ValueError as e:
            print(f"Warning: Could not set datetime index due to error: {e}")
            # If conversion fails, keep the index as seconds (float)

        df.index.name = 'time'

    return df

new_data = read_fastdata(file_path)
```

new_data

```
288000
Status_0_2m_init not used. Incorrect length.
Status_0_2m_lconv not used. Incorrect length.
Status_0_2m_rel not used. Incorrect length.
Status_0_2m_uconv not used. Incorrect length.
Tc_0_2m_init not used. Incorrect length.
Tc_0_2m_lconv not used. Incorrect length.
Tc_0_2m_rel not used. Incorrect length.
Tc_0_2m_uconv not used. Incorrect length.
U_0_2m_init not used. Incorrect length.
U_0_2m_lconv not used. Incorrect length.
U_0_2m_rel not used. Incorrect length.
U_0_2m_uconv not used. Incorrect length.
V_0_2m_init not used. Incorrect length.
V_0_2m_lconv not used. Incorrect length.
V_0_2m_rel not used. Incorrect length.
V_0_2m_uconv not used. Incorrect length.
Tc_0_2m_init not used. Incorrect length.
Tc_0_2m_lconv not used. Incorrect length.
Tc_0_2m_rel not used. Incorrect length.
Tc_0_2m_uconv not used. Incorrect length.
T_0_2m_init not used. Incorrect length.
T_0_2m_lconv not used. Incorrect length.
T_0_2m_rel not used. Incorrect length.
T_0_2m_uconv not used. Incorrect length.
T_1_5m_init not used. Incorrect length.
T_1_5m_lconv not used. Incorrect length.
T_1_5m_rel not used. Incorrect length.
T_1_5m_uconv not used. Incorrect length.
T_10m_init not used. Incorrect length.
T_10m_uconv not used. Incorrect length.
T_15m_lconv not used. Incorrect length.
T_15m_rel not used. Incorrect length.
T_20m_lconv not used. Incorrect length.
T_20m_rel not used. Incorrect length.
T_4_5m_init not used. Incorrect length.
T_4_5m_lconv not used. Incorrect length.
T_4_5m_rel not used. Incorrect length.
T_4_5m_uconv not used. Incorrect length.
T_8_5m_lconv not used. Incorrect length.
T_8_5m_rel not used. Incorrect length.
T_a1_1_5m_lconv not used. Incorrect length.
T_a1_1_5m_rel not used. Incorrect length.
T_a1_1_5m_uconv not used. Incorrect length.
T_a2_1_5m_lconv not used. Incorrect length.
T_a2_1_5m_rel not used. Incorrect length.
T_a2_1_5m_uconv not used. Incorrect length.
```

Out[15]:

	u_1_5m_init	u_1_5m_lconv	u_1_5m_rel	u_1_5m_uconv	u_10m_init	u_10m_lconv	u_10m_uconv
time							
2018-11-03 00:00:00.500000000	-0.730278	-0.489497	-0.020739	-0.029100	0.006109	-0.161528	
2018-11-03 00:00:00.549999952	-0.737284	-0.489246	-0.038719	-0.023778	0.026885	-0.163021	
2018-11-03 00:00:00.599999905	-0.717173	-0.496192	-0.056892	0.012822	0.041164	-0.175584	
2018-11-03 00:00:00.650000095	-0.731094	-0.510473	-0.055331	-0.035599	0.022177	-0.179492	
2018-11-03 00:00:00.700000048	-0.770513	-0.501275	-0.064697	-0.125966	0.028782	-0.193024	
...
2018-11-03 04:00:00.250000000	-0.548560	-0.166679	-0.288865	-0.102193	-0.751853	-0.444064	
2018-11-03 04:00:00.299999952	-0.544533	-0.182729	-0.272737	-0.099198	-0.781932	-0.440990	
2018-11-03 04:00:00.349999905	-0.554240	-0.172331	-0.279760	-0.100894	-0.777633	-0.437579	
2018-11-03 04:00:00.400000095	-0.578812	-0.210540	-0.330690	-0.106968	-0.785173	-0.441777	
2018-11-03 04:00:00.450000048	-0.584420	-0.217807	-0.252970	-0.106915	-0.782349	-0.426571	

288000 rows × 138 columns



In [49]:

```
# the upper convergence tower is what I am interested in.
uconv_cols = [col for col in new_data.columns if 'uconv' in col.lower()]
uconv_cols
```

```
Out[49]: ['u_1_5m_uconv',
 'u_10m_uconv',
 'u_3m_uconv',
 'u_4_5m_uconv',
 'u_6m_uconv',
 'u_a1_1_5m_uconv',
 'u_a2_1_5m_uconv',
 'v_1_5m_uconv',
 'v_10m_uconv',
 'v_3m_uconv',
 'v_4_5m_uconv',
 'v_6m_uconv',
 'v_a1_1_5m_uconv',
 'v_a2_1_5m_uconv',
 'w_1_5m_uconv',
 'w_10m_uconv',
 'w_3m_uconv',
 'w_4_5m_uconv',
 'w_6m_uconv',
 'w_a1_1_5m_uconv',
 'w_a2_1_5m_uconv',
 'Tirga_1_5m_uconv',
 'Tirga_6m_uconv',
 'tc_1_5m_uconv',
 'tc_10m_uconv',
 'tc_3m_uconv',
 'tc_4_5m_uconv',
 'tc_6m_uconv',
 'tc_a1_1_5m_uconv',
 'tc_a2_1_5m_uconv']
```

```
In [50]: uconv_wind = new_data[uconv_cols]
uconv_wind
```

Out[50]:

	time	u_1_5m_uconv	u_10m_uconv	u_3m_uconv	u_4_5m_uconv	u_6m_uconv	u_a1_1_5m
2018-11-03 00:00:00.500000000		-0.029100	0.138630	-0.179706	-0.208202	-0.476709	0
2018-11-03 00:00:00.549999952		-0.023778	0.120307	-0.195052	-0.236315	-0.460240	0
2018-11-03 00:00:00.599999905		0.012822	0.118316	-0.190777	-0.253072	-0.468736	0
2018-11-03 00:00:00.650000095		-0.035599	0.113922	-0.157592	-0.241704	-0.466041	-0
2018-11-03 00:00:00.700000048		-0.125966	0.105554	-0.158864	-0.208750	-0.460349	-0
...
2018-11-03 04:00:00.250000000		-0.102193	-0.431917	-0.178729	-0.264832	-0.198122	-0
2018-11-03 04:00:00.299999952		-0.099198	-0.437317	-0.184865	-0.264031	-0.199216	-0
2018-11-03 04:00:00.349999905		-0.100894	-0.434981	-0.190053	-0.258773	-0.200507	-0
2018-11-03 04:00:00.400000095		-0.106968	-0.432215	-0.191588	-0.257862	-0.201066	-0
2018-11-03 04:00:00.450000048		-0.106915	-0.435992	-0.196206	-0.251106	-0.201634	-0

288000 rows × 30 columns

In [51]:

```
def calc_wind(u, v):
    """
    Calculates wind speed (m/s) and direction (0-360 degrees) given
    u (zonal) and v (meridional) components.

    Params
    -----
    u (float): zonal wind component
    v (float): meridional wind component

    Returns
    -----
    speed in m/s
    direction in meteorological coordinates(0 = North)
    ...
    speed = np.sqrt(u**2 + v**2)
    direction = (np.rad2deg(np.arctan2(u, v)) + 360) %360
    return speed, direction

calc_wind(-4, 4)
```

Out[51]: (np.float64(5.656854249492381), np.float64(315.0))

```
In [52]: # for testing
df = uconv_wind.copy()

height_suffixes = set()
for col in df.columns:
    suffix = col[col.find('_'): ]
    print(col, suffix)
    height_suffixes.add(suffix)

for suffix in height_suffixes:
    u_col = 'u'+suffix
    v_col = 'v'+suffix

    if u_col in df.columns and v_col in df.columns:
        print(f'calculating {u_col} and {v_col}')
        df[f'spd{suffix}'], df[f'dir{suffix}'] = calc_wind(df[u_col], df[v_col])
```

```
u_1_5m_uconv _1_5m_uconv
u_10m_uconv _10m_uconv
u_3m_uconv _3m_uconv
u_4_5m_uconv _4_5m_uconv
u_6m_uconv _6m_uconv
u_a1_1_5m_uconv _a1_1_5m_uconv
u_a2_1_5m_uconv _a2_1_5m_uconv
v_1_5m_uconv _1_5m_uconv
v_10m_uconv _10m_uconv
v_3m_uconv _3m_uconv
v_4_5m_uconv _4_5m_uconv
v_6m_uconv _6m_uconv
v_a1_1_5m_uconv _a1_1_5m_uconv
v_a2_1_5m_uconv _a2_1_5m_uconv
w_1_5m_uconv _1_5m_uconv
w_10m_uconv _10m_uconv
w_3m_uconv _3m_uconv
w_4_5m_uconv _4_5m_uconv
w_6m_uconv _6m_uconv
w_a1_1_5m_uconv _a1_1_5m_uconv
w_a2_1_5m_uconv _a2_1_5m_uconv
Tirga_1_5m_uconv _1_5m_uconv
Tirga_6m_uconv _6m_uconv
tc_1_5m_uconv _1_5m_uconv
tc_10m_uconv _10m_uconv
tc_3m_uconv _3m_uconv
tc_4_5m_uconv _4_5m_uconv
tc_6m_uconv _6m_uconv
tc_a1_1_5m_uconv _a1_1_5m_uconv
tc_a2_1_5m_uconv _a2_1_5m_uconv
calculating u_6m_uconv and v_6m_uconv
calculating u_a2_1_5m_uconv and v_a2_1_5m_uconv
calculating u_1_5m_uconv and v_1_5m_uconv
calculating u_4_5m_uconv and v_4_5m_uconv
calculating u_3m_uconv and v_3m_uconv
calculating u_10m_uconv and v_10m_uconv
calculating u_a1_1_5m_uconv and v_a1_1_5m_uconv
```

```
In [53]: df.head()
```

Out[53]:

	time	u_1_5m_uconv	u_10m_uconv	u_3m_uconv	u_4_5m_uconv	u_6m_uconv	u_a1_1_5m
2018-11-03 00:00:00.500000000		-0.029100	0.138630	-0.179706	-0.208202	-0.476709	0
2018-11-03 00:00:00.549999952		-0.023778	0.120307	-0.195052	-0.236315	-0.460240	0
2018-11-03 00:00:00.599999905		0.012822	0.118316	-0.190777	-0.253072	-0.468736	0
2018-11-03 00:00:00.650000095		-0.035599	0.113922	-0.157592	-0.241704	-0.466041	-0
2018-11-03 00:00:00.700000048		-0.125966	0.105554	-0.158864	-0.208750	-0.460349	-0

5 rows x 44 columns



Curious about spectral analysis

In class we worked on periodograms and it was pretty neat. The atmosphere on very small scales is not as wave-like so we can test some others later in this notebook.

In [54]:

```
from scipy import signal
from scipy.interpolate import interp1d
```

When looking at the Power Spectrum, the low frequency (large scale motions contain most of the energy.)

In [59]:

```
fs = 20 # Hz
frequencies, power = signal.periodogram(df['spd_10m_uconv'], fs) #(sig, fs)

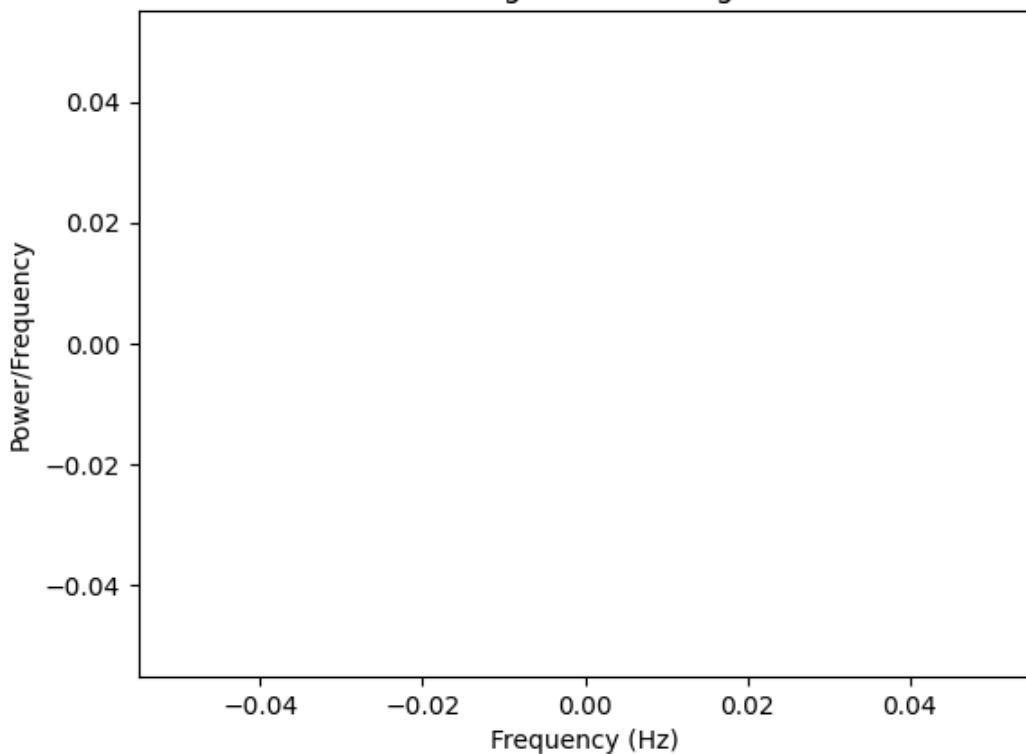
# print (frequencies, power)

#Plot the power (y) in each frequency (x)
plt.plot(frequencies, power)

# plt.xlim(0,0.01)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power/Frequency')
plt.title('Periodogram of the Signal')
plt.show()
```

```
/home/jupyter-bmarosites/.local/lib/python3.12/site-packages/scipy/signal/_signaltools.py:3927:
RuntimeWarning: invalid value encountered in subtract
    ret = data - np.mean(data, axis, keepdims=True)
/home/jupyter-bmarosites/.local/lib/python3.12/site-packages/scipy/signal/_spectral_py.py:2194:
RuntimeWarning: invalid value encountered in multiply
    result = win * result
```

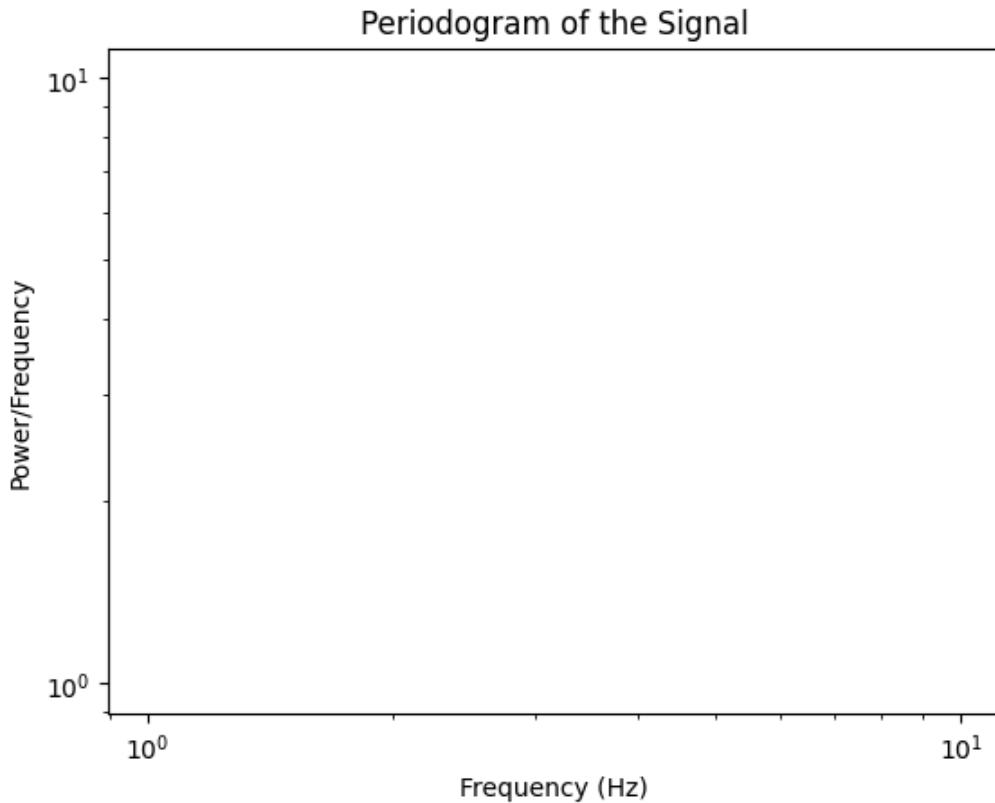
Periodogram of the Signal



```
In [56]: fs = 20 # Hz
frequencies, power = signal.periodogram(df['spd_10m_uconv'], fs) #(sig, fs)

plt.loglog(frequencies, power)
# plt.xlim(0,20)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power/Frequency')
plt.title('Periodogram of the Signal')

plt.show()
```



Now I can test a different method

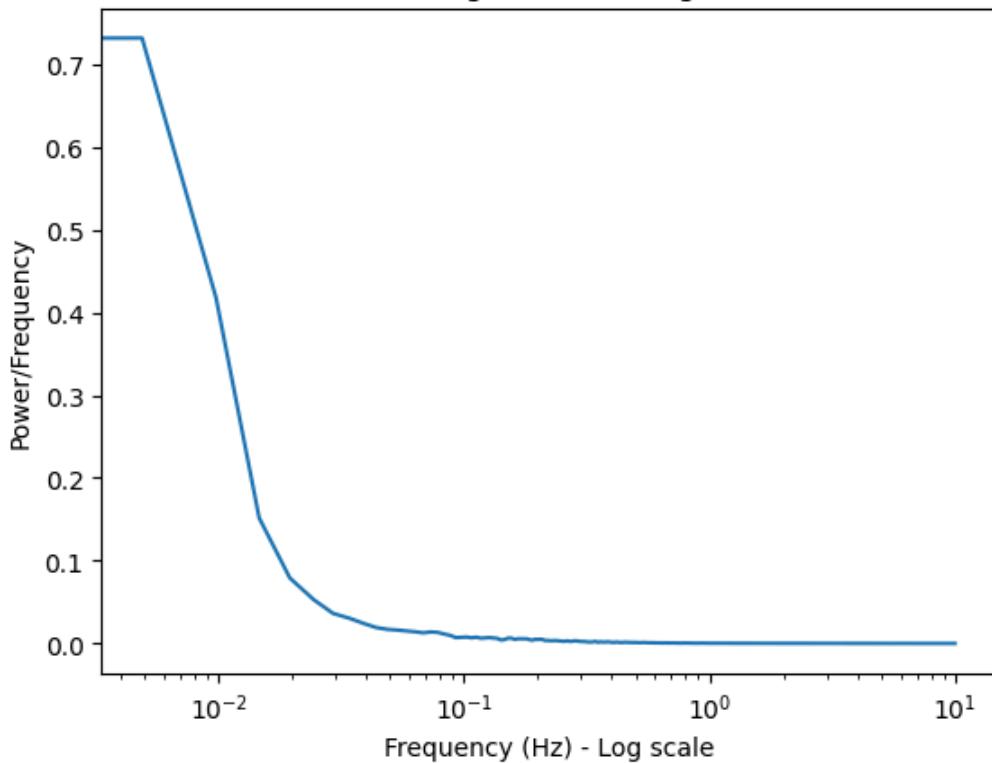
```
In [60]: # Try Welch
nperseg = 2**12
print(nperseg/20/60)

# Calculate the Power Spectral Density (PSD)
f, S_v = signal.welch(
    df['spd_3m_uconv'],
    fs=fs,
    nperseg=nperseg,
    scaling='density',
    detrend='linear' # Recommended for atmospheric data to remove mean/trend
)
```

3.4133333333333336

```
In [61]: #Plot the power (y) in each frequency (x)
plt.plot(f, S_v)
plt.xscale('log')
# plt.xlim(1,10)
plt.xlabel('Frequency (Hz) - Log scale')
plt.ylabel('Power/Frequency')
plt.title('Periodogram of the Signal')
plt.show()
```

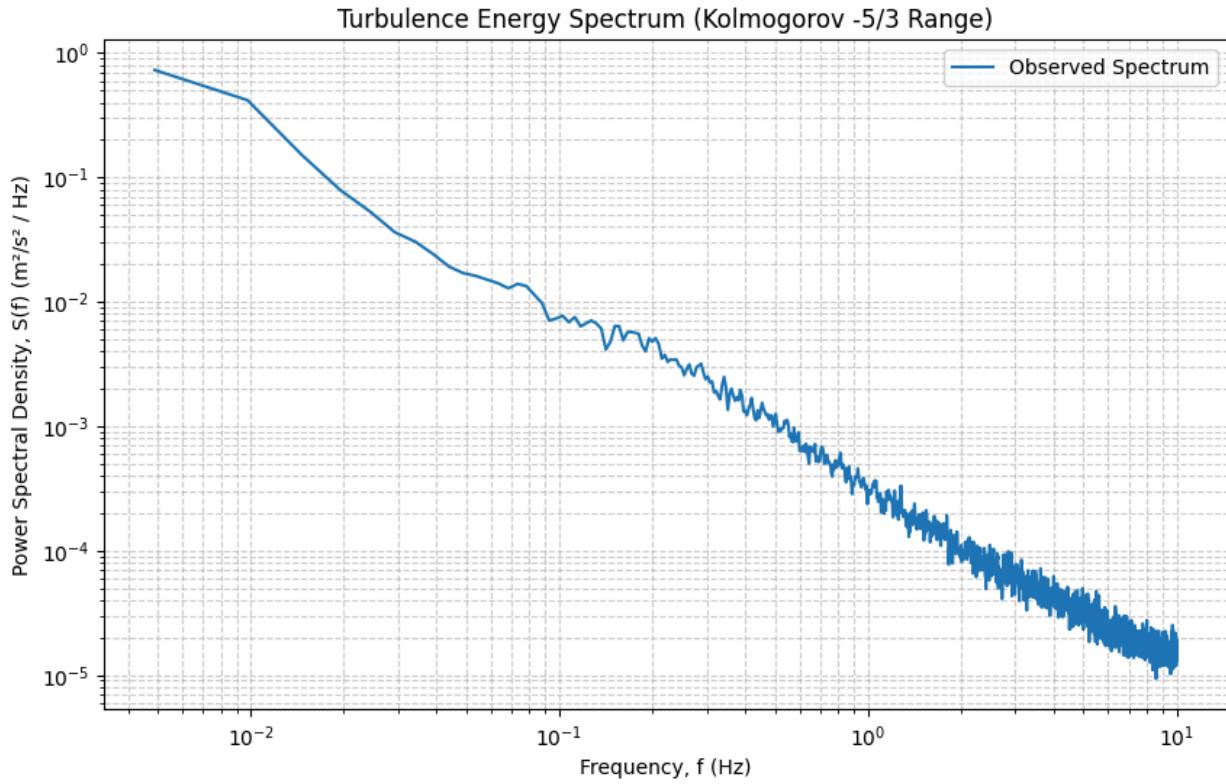
Periodogram of the Signal



Additional things to look at include the slope of this rate in decrease. I did not calculate it here, but the expectation from Kolmogorov (1954) there should be a slope of $-5/3$ in the range between the highest density, low frequency to the low energy density, high frequency turbulence.

```
In [62]: plt.figure(figsize=(10, 6))
plt.loglog(f[1:], S_v[1:], label='Observed Spectrum')

plt.xlabel('Frequency, f (Hz)')
plt.ylabel('Power Spectral Density, S(f) (m²/s² / Hz)')
plt.title('Turbulence Energy Spectrum (Kolmogorov -5/3 Range)')
plt.grid(True, which="both", ls="--", alpha=0.6)
plt.legend()
plt.show()
```



Now I want to look into the multiresolution decomposition MDR

```
In [64]: def multires(a, b, m):
    # Provided from Dr. Carmen Nappo
    d = np.zeros(int(m))
    for ims in np.arange(m, 0, -1):
        l = 2.0 ** ims
        nw = (2.0 ** m) / l
        sumab = 0.0

        for i in np.arange(0, int(nw)):
            k = (i) * int(l) + 1
            start_idx = int(k - 1)

            # Calculate block means (za and zb)
            za = 0.0
            zb = 0.0
            for j in np.arange(k, (k + l)):
                za = za + a[int(j - 1)]
                zb = zb + b[int(j - 1)]

            za = za / l
            zb = zb / l
            sumab = sumab + za * zb

    # Detrending/Filtering step: Remove the block average
    for j in np.arange(k, (int(i + 1) * l + 1)):
        if int(j-1) < len(a):
            a[int(j - 1)] = a[int(j - 1)] - za
            b[int(j - 1)] = b[int(j - 1)] - zb
```

```

    if (nw > 1):
        d[int(ims-1)] = (sumab / nw)

    return d

def mrd_9000(u, w, T, timestamp, sample_rate, total_period):
    delta_t = 1.0/sample_rate
    num_secs = total_period * 3600.
    n_float = num_secs * sample_rate

    m = np.floor(math.log10(n_float) / math.log10(2))
    n = 2.0 ** m
    n_int = int(n)

    num_segs = np.floor(len(u)/n)
    num_segs_int = int(num_segs)

    values = np.zeros([int(m), num_segs_int])
    averaging_periods = np.arange(0,m)
    averaging_periods = (2.0 ** averaging_periods) * delta_t

    for i in np.arange(0, num_segs_int):
        start_idx = int(i * n)
        end_idx = int((i * n) + n)

        w_final = w[start_idx:end_idx][:n_int]
        T_final = T[start_idx:end_idx][:n_int]

        mapped_w = w_final
        mapped_T = T_final

        values[:,i] = multires(mapped_w, mapped_T, m) #multires(mapped_w.x, mapped_T.x, m)

    output={'time_scale':averaging_periods,'results':values}
    print ('mrd calculations complete')
    return output

def getprimes(data,times,avg_time):
    total_length=times[-1]-times[0]

    # Check if times is a numpy Datetime array or Python List of datetime objects
    if isinstance(total_length, np.timedelta64):
        total_seconds = total_length / np.timedelta64(1, 's')
        avg_timedelta = np.timedelta64(avg_time, 's')
    else: # Assume datetime.timedelta object
        total_seconds = total_length.total_seconds()
        avg_timedelta = dt.timedelta(0, avg_time)

    output_length = int(total_seconds / avg_time) # <-- THE CRITICAL FIX

    prime_data=np.zeros(len(times))
    avg_start=times[0]

    for i in range(0, output_length): # <-- Now 'output_length' is an integer

        avg_end = avg_start + avg_timedelta # calculate the end time for each average period

        # Use np.searchsorted for fast index finding (instead of a slow loop)
        start_index = np.searchsorted(times, avg_start)
        stop_index = np.searchsorted(times, avg_end)

```

```

avg_start = avg_end # update to the start of the next period

segment = data[start_index:stop_index]

if len(segment) < 1:
    avg_value=float('nan')
else:
    avg_value=np.sum(segment)/len(segment)

# Only apply detrending if the average is not NaN
if not np.isnan(avg_value):
    prime_data[start_index:stop_index]=segment - avg_value

return prime_data

# Extract Data and Parameters # Change this to different tower heights 4_5m, 10m etc.
u = df['u_4_5m_uconv'].values
w = df['w_4_5m_uconv'].values
T = df['tc_4_5m_uconv'].values
timestamp = df.index.to_numpy()

# Parameters
AVG_TIME_S = 30
MRD_RATE_HZ = 20
MRD_PERIOD_HRS = 1

print(f"--- Running MRD: w'T' Flux Decomposition ---")
print(f"Data Rate: {MRD_RATE_HZ} Hz, Fluctuation Block: {AVG_TIME_S} seconds")

# 3. Calculate Primes (This step now works)
wprime=getprimes(w, timestamp, AVG_TIME_S)
Tprime=getprimes(T, timestamp, AVG_TIME_S)

# 4. Run MRD
mrd_results = mrd_9000(u, wprime, Tprime, timestamp, MRD_RATE_HZ, MRD_PERIOD_HRS)

# 5. Process Results
num_segments = mrd_results['results'].shape[1]
segment_cols = [f'Segment {i+1}' for i in range(num_segments)]

# Create the DataFrame
results_df = pd.DataFrame(mrd_results['results'], columns=segment_cols)

# Insert the 'time_scale' array as the first column
results_df.insert(0, 'Time_Scale_Seconds', mrd_results['time_scale'])

results_df['Mean_Covariance'] = results_df[segment_cols].mean(axis=1)

print("\n--- Summary of Mean MRD Results (w'T' Flux Contribution) ---")
print(results_df.to_string(index=False))

```

```

--- Running MRD: w'T' Flux Decomposition ---
Data Rate: 20 Hz, Fluctuation Block: 30 seconds
mrd calculations complete

```

```

--- Summary of Mean MRD Results (w'T' Flux Contribution) ---

```

Time_Scale_Seconds	Segment 1	Segment 2	Segment 3	Segment 4	Mean_Covariance
0.05	-7.149096e-06	-8.943997e-06	-8.595434e-06	-1.487877e-05	-9.891823e-06
0.10	-1.158141e-05	-1.551904e-05	-1.512073e-05	-3.047769e-05	-1.817472e-05
0.20	-1.438427e-05	-2.650303e-05	-2.773708e-05	-6.288831e-05	-3.287817e-05
0.40	-1.210001e-05	-4.676306e-05	-4.452271e-05	-1.131863e-04	-5.414303e-05
0.80	-3.875767e-06	-6.366763e-06	-8.341536e-05	-1.724082e-04	-6.651651e-05
1.60	-1.417416e-05	-3.566194e-05	-7.389919e-05	-1.751100e-04	-7.471133e-05
3.20	-2.044639e-05	-1.702641e-04	-1.821952e-04	-1.011496e-04	-1.185138e-04
6.40	-3.895272e-05	2.203355e-04	3.138486e-05	-5.317454e-05	3.989828e-05
12.80	3.402463e-05	-1.053103e-07	-4.577605e-05	1.654887e-05	1.173034e-06
25.60	-6.571452e-06	-1.846747e-05	4.732346e-06	-2.048272e-05	-1.019732e-05
51.20	-7.524736e-07	-8.117154e-06	-1.555423e-07	-1.899110e-06	-2.731070e-06
102.40	-7.195420e-08	-1.436933e-06	-2.990584e-06	-3.029009e-06	-1.882120e-06
204.80	-2.303580e-07	-2.341936e-06	-3.645551e-07	-3.549317e-07	-8.229452e-07
409.60	-9.338823e-09	-1.138384e-07	-2.138213e-07	-1.826137e-07	-1.299031e-07
819.20	-8.437912e-11	-5.534651e-08	-5.304740e-08	-9.067633e-08	-4.978865e-08
1638.40	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00

```

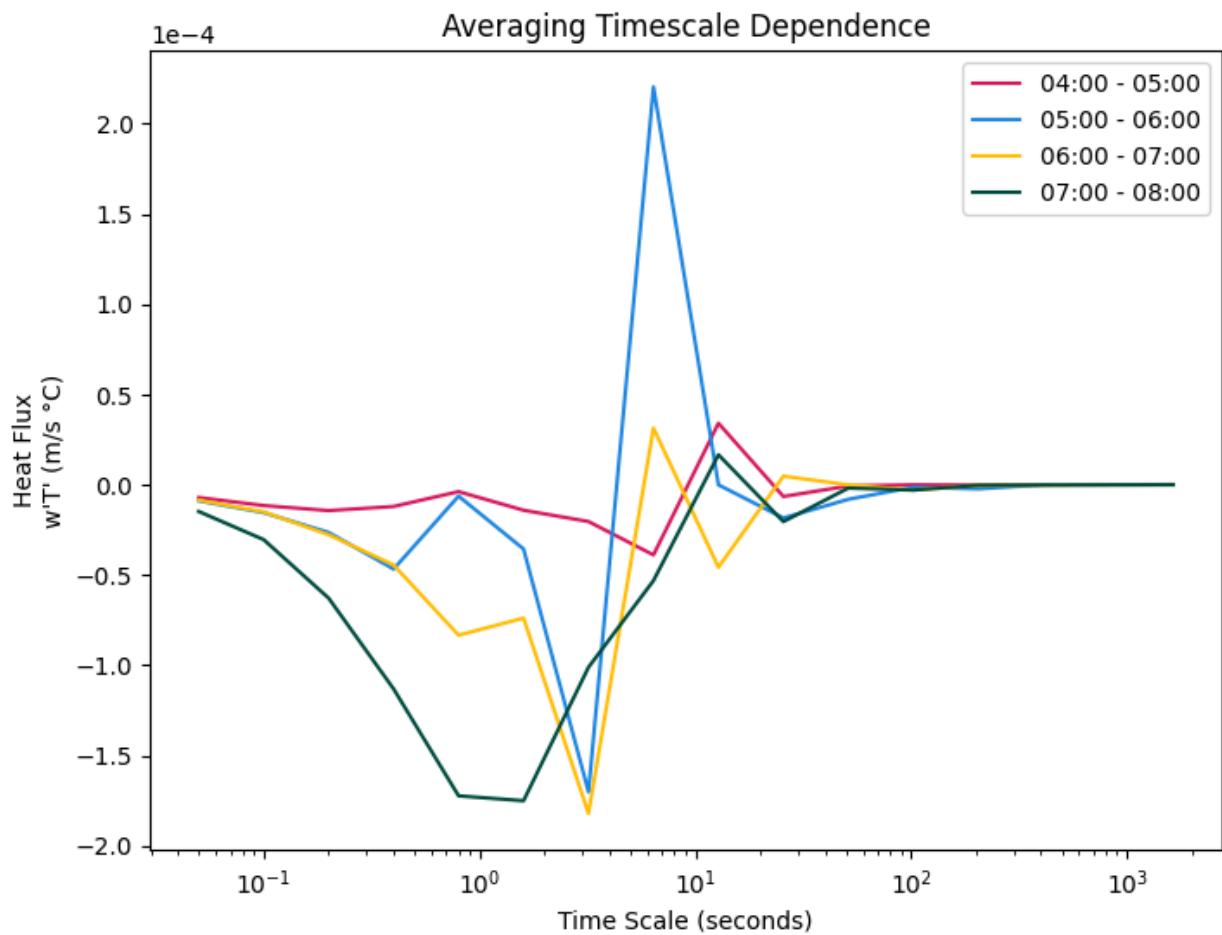
In [65]: friendly_colors = ['#D81B60', '#1E88E5', '#FFC107', '#004D40']
time_labels = ['04:00 - 05:00', '05:00 - 06:00', '06:00 - 07:00', '07:00 - 08:00']

plt.figure(figsize=(8,6))
columns = [col for col in results_df.columns if col.startswith('Segment')]
for i, segment in enumerate(columns):
    plt.plot(results_df['Time_Scale_Seconds'], results_df[segment], color=friendly_colors[i],
    plt.xscale('log')
    plt.ylabel("Heat Flux\nw'T' (m/s °C)")
    plt.xlabel("Time Scale (seconds)")
    plt.title('Averaging Timescale Dependence')
    plt.legend()

    plt.gca().ticklabel_format(axis='y', style='sci', scilimits=(0, 0))

plt.savefig('./figures/AveragingTimescale.png')

```



In []:

In []: