Investigation of Cusp Irregularities sounding rockets

Measurements using the m-NLP instrument has been preformed on the ICI-2 sounding rocket launched from Svaldbard in 2008. You will work on the collected m-NLP data from this rocket and plot the measure electron density as a function of time.

High-latitude F-region

- Increase in electron density give rise to great disturbances in radio-wave propagation conditions (reflection, absorption, and scatter) – impact GNSS
- These disturbances are small-scale irregularities in the ionosphere (< km targets)
- Resolution of current state of the art HF raders are usualy too poor to study fluctuations at decameter scale.

Research about small-scale irregularities in the high-latitude F-region has historically not progressed as fast as in equatorial regions. A probable reason is the geometrical impossibility for high-latitude ground based rader to obtain an appropriate backscatter angle at F-region using VHF and UHF systems. While F-region ground-based radar are possible at HF frequencies, their resolution is usually too poor to investigate the electron density fluctuations at decameter scales.

Therefore, Moen et al. 2002 pointed out an ultimate need for high-resolution in-situ measurements in order to be able to characterize these small-scale irregulariteis. This gave birth to the sounding rocket program Investigation of Cups Irregularties (ICI) by the University of Oslo. To day, the following rockets have been launched:

- ICI-1 (2003)
- ICI-2 (2008)
- ICI-3 (2011)
- ICI-4 (2015)
- ICI-5 (2019)
- ICI-5-bis: Planned 2024/2025

Source: Andres Spicher, Spectral Analysis of electron density fluctuations by the ICI-2 sounding rocket, Master's thesis, University of Oslo, Mai 2013. http://urn.nb.no/URN:NBN:no-38642

Case study ICI-2

Measurements using the m-NLP instrument has been preformed on the ICI-2 sounding rocket launched from Ny-Ålesund, Svaldbard (78.9° N, 11.9° E geographic coordinates) at 1035 Universal Time (UT) on the 5th of December 2008.

This notebooks demonstrate how to get from raw data to the electron density.

Method

The applied method is described in [Bekkeng 2010] and [Jacobsen 2010].

K. S. Jacobsen et al., "A new Langmuir probe concept for rapid sampling of space plasma electron density", Measurement Science and Technology, Volume 21, No. 8. http://iopscience.iop.org/article/10.1088/0957-0233/21/8/085902/meta

T. A. Bekkeng et al., "Design of a multi-needle Langmuir probe system", Measurements Science and Technology, Volume 21, No. 8 http://iopscience.iop.org /article/10.1088/0957-0233/21/8/085903/meta

Analysis steps

1. Prepare and plot the raw data (4 curves)
2. Find the relevant spin components
3. Filter out the spins components and plot the filtered currents
4. Calculate the electron density and plot the density curve

Electron density $N_e = \sqrt{K\frac{\Delta I^2}{\Delta V}}$

$K = \frac{m_e}{2q(q2rl)^2}$

m-NLP configuraiton

The m-NLP instrument flew four probes biased at 2.5 V, 4 V, 5.5 V, and 10 V.

The 16-bit ADC sampling rate was 5787 Hz, and the resistor value in the current to voltage converter was 4.7 MΩ. The dynamic range of the ADC is +10 V to -10 V, or 20 V.

The data file contains 5 columns:

- ADC sample number
- ADC count value for probe 1
- ADC count value for probe 2
- ADC count value for probe 3
- ADC count value for probe 4

```python
# Import all relevant libraries
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
```

```python
from scipy import signal
import scipy.constants as sc
```

Support functions

```python
#Functions

# Function to filter out a frequency band
# data: Array of data
# lowcut: Lowest frequency in the bandstop filter
# highcut: higest frequency in the bandstop filter
# fs: Sample frequency of data, we use 5787
# order: order of filter, we use a second order filter
def butter_bandstop_filter(data, lowcut, highcut, fs, order):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    i, u = signal.butter(order, [low, high], btype='bandstop')
    y = signal.lfilter(i, u, data)
    return y


# function to convert raw ADC values to current
# ADC is 2^16 = 65536 bits
# ADC dynamic range is 20 V (+-10 V), midrange at ~ 2^16 / 2
# LSB resolution:  20V / (2^16 * 4.7e6) => 6.49e-11 A
# V = IR  => I = V/R
def calc_current(x):
    midpoint = 2**15
    return (x-midpoint)*adc_dynamic_range/(adc_bits*resistor)

def calc_Ne(coef_a):
    res = np.sqrt(coef_a* ( sc.m_e/(2*sc.e*(e*2*rp*lp)**2)))
    return res
```

```python
# Raw data
#fData = "../data/ici2_mnlp_raw.txt"
fData = "https://folk.universitetetioslo.no/ketilroe/agf223/ici2_mnlp_raw.txt"
```

```python
# constants

# probe dimensions
rp=0.255e-3;
lp=2.5e-2;
Ae=2*math.pi*rp*lp;
```

```
# Voltages
# 2.5, 4, 5.5, 10
Vbias = [2.5, 4, 5.5, 10]

# sampling rate
sampling_rate = 5787;

# adc to current convertion constants
adc_bits = math.pow(2,16)
resistor = 4.7e6
adc_dynamic_range = 20 # +-10V
```

1. Prepare and plot the raw data

- Read the raw data in to a Pandas dataframe
- Convert column with time information to seconds by dividing by the sampling rate.
- Add four new columns converting the raw adc counts to currents.
- Plot the 4 currents as a function of time. (Plot only from 100 to 400 seconds)

```
# 1. Add your solution here
```

```
#. Prepare and plot the raw data - solution


#Read raw data into Pandas DataFrame and add column header names
df_mnlp= pd.read_csv(fData,sep="\s+",header=None,names = ["time","adc1", "adc2", "adc3", "ac

# Convert time to seconds
# Column one is sample counter, dividing by sample_rate gives time in seconds
df_mnlp["time"] /= sampling_rate


# Convert probe columns to current
df_mnlp[['i1','i2','i3','i4']] = df_mnlp[['adc1','adc2','adc3','adc4']].apply(calc_current)
```

```
df_mnlp.head()
```

```
       time     adc1     adc2     adc3     adc4            i1            i2  \
0  0.000000  32259.0  32250.0  32245.0  32298.0 -3.304989e-08 -3.363427e-08
1  0.000173  32256.0  32250.0  32243.0  32298.0 -3.324468e-08 -3.363427e-08
2  0.000691  32259.0  32249.0  32243.0  32297.0 -3.304989e-08 -3.369920e-08
3  0.000864  32260.0  32251.0  32243.0  32297.0 -3.298496e-08 -3.356934e-08
```

```
4  0.001037  32260.0  32249.0  32243.0  32300.0 -3.298496e-08 -3.369920e-08

             i3            i4
0 -3.395892e-08 -3.051758e-08
1 -3.408878e-08 -3.051758e-08
2 -3.408878e-08 -3.058251e-08
3 -3.408878e-08 -3.058251e-08
4 -3.408878e-08 -3.038772e-08
```

```
# Plot all raw data

# Reduce data to a subset for faster processing
t_low = 100
t_high = 450

#Returns a table with false or true for each row according to if condition is met or not.
idx_filter = ( df_mnlp['time'] >= t_low) & ( df_mnlp['time'] < t_high)

colors =['red','blue','green','orange']
y_columns = ['i1','i2','i3','i4']
ax = df_mnlp[idx_filter].plot(x='time', y=y_columns,kind='line',color=colors)#,linestyle=['
ax.set_xlabel("Time [s]");
ax.set_ylabel("Current [A]");
ax.set_title("Current plots for m-NLP Probes on ICI2");

# We see the the data cleary contains a lot of noise.
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
# Zooming in we can identify a regular oscillation in the data. This is due to the spin of
t_low = 428
t_high = 435

#Returns a table with false or true for each row according to if condition is met or not.
time_filter = ( df_mnlp['time'] >= t_low) & ( df_mnlp['time'] < t_high)

#df_mnlp_subset = df_mnlp[time_filter]

ax = df_mnlp[time_filter].plot(x='time', y=y_columns,kind='line',color=colors)#,linestyle=[
ax.set_xlabel("Time [s]");
ax.set_ylabel("Current [A]");
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

2. Find the relevant spin components

- Use the numpy FFT function to perform a frequency analysis on probe 4

```python
# Add your solution here:
```

```python
# Find spin frequency and harmonics
# Use the numpy FFT function to perform a frequency analysis on probe 4

# Create idx filter for a short time period
idx_filter = ( df_mnlp['time'] >= 100) & ( df_mnlp['time'] < 120)

# Extract data from probe 4
data = df_mnlp[idx_filter].i4

# Generate fft
fft = np.fft.fft(data)

# Return amplitude spectrum
fft_abs = np.abs(fft)

# return an array giving the frequencies of corresponding elements in the output
freqs = np.fft.fftfreq(len(data))*sampling_rate

#Plot
fig, ax = plt.subplots()
ax.plot(freqs,fft_abs,linestyle="--",marker=".")

# Limit x-axis to find expected spin frequencies in the order of some revolutions per secon
ax.set_xlim(0,30)
#ax.set_ylim(5e-7,5e-3)
ax.set_yscale('log')
ax.set_title("Frequency spectrum of Probe 4 signal")
ax.set_ylabel("Amplitude [a.u.]")
ax.set_xlabel("Frequency [Hz]")


# Detect peaks without any filters
# Height: filter out peaks lower value
# Distance: Filter out peaks which are closer than X sample points
indexes = signal.find_peaks(fft_abs, height=0.00005,distance=50)

print("The following peaks are found:")

x_peaks = indexes[0]
y_peaks = indexes[1]['peak_heights']
print(freqs[x_peaks])
print(y_peaks)

# The rocket has a spin frequency of approximately 3.25 Hz.
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
The following peaks are found:
[ 0.65  3.25  6.55  9.8  -9.8  -6.55 -3.25 -0.65]
[1.80278070e-04 1.41473178e-03 4.17794968e-04 5.88045698e-05
 5.88045698e-05 4.17794968e-04 1.41473178e-03 1.80278070e-04]
```

3. Filter out the spin components and plot the filtered currents

```
# Filter out spin frequencies

data = df_mnlp

# Probe 1
p1_temp = butter_bandstop_filter(data.i1,3.0,3.5,sampling_rate,2)
p1_temp2 = butter_bandstop_filter(p1_temp,6.25,6.75,sampling_rate,2)
p1_filter = butter_bandstop_filter(p1_temp2,9.5,10.0,sampling_rate,2)

# Probe 2
p2_temp = butter_bandstop_filter(data.i2,3.0,3.5,sampling_rate,2)
p2_temp2 = butter_bandstop_filter(p2_temp,6.25,6.75,sampling_rate,2)
p2_filter = butter_bandstop_filter(p2_temp2,9.5,10.0,sampling_rate,2)


# Probe 3
p3_temp = butter_bandstop_filter(data.i3,3.0,3.5,sampling_rate,2)
p3_temp2 = butter_bandstop_filter(p3_temp,6.25,6.75,sampling_rate,2)
p3_filter = butter_bandstop_filter(p3_temp2,9.5,10.0,sampling_rate,2)

# Probe 4
p4_temp = butter_bandstop_filter(data.i4,3.0,3.5,sampling_rate,2)
p4_temp2 = butter_bandstop_filter(p4_temp,6.25,6.75,sampling_rate,2)
p4_filter = butter_bandstop_filter(p4_temp2,9.5,10.0,sampling_rate,2)



df_mnlp['i1_f'] = p1_filter
df_mnlp['i2_f'] = p2_filter
df_mnlp['i3_f'] = p3_filter
```

```
df_mnlp['i4_f'] = p4_filter


# Plot filtered data
colors =['red','blue','green','orange']
y_columns = ['p1','p2','p3','p4']

# Reduce data to a subset for faster processing
```

Plot the various steps of the spin filtering

```
# Plot the various steps of the spin filtering
# Reduce data to a subset for faster processing and to view the spin components
t_low = 427#280 #122
t_high = 435#330 #124
y_lim_low = 155
y_lim_high = 300
#Returns a table with false or true for each row according to if condition is met or not.
time_filter = ( df_mnlp['time'] >= t_low) & ( df_mnlp['time'] < t_high)

label1 = "Unfiltered data"
label2 = "Filterd with band-reject at 3.25 HZ"
label3 = "Filterd with band-reject at (3.25, 6.5) Hz"
label4 = "Filterd with band-reject at (3.25, 6.5, 9.75) Hz"


plt.figure(figsize=(10,5))
plt.plot(df_mnlp[time_filter].time, df_mnlp.i4[time_filter]*1e9,label=label1,color='red',mar
#plt.plot(df_mnlp[time_filter].time, p4_temp[time_filter]*1e9,label=label2, color='blue',mar
#plt.plot(df_mnlp[time_filter].time, p4_temp2[time_filter]*1e9,label=label3, color='black')
#plt.plot(df_mnlp[time_filter].time, df_mnlp.i4_f[time_filter]*1e9,label=label4)
plt.legend(loc='upper right')
plt.title("Probe 4 current with various band-reject filters")
plt.ylabel("Current [nA]")
plt.xlabel("Time [s]")
plt.ylim(y_lim_low,y_lim_high)
plt.grid()


plt.figure(figsize=(10,5))
plt.plot(df_mnlp[time_filter].time, df_mnlp.i4[time_filter]*1e9,label=label1,color='red',mar
plt.plot(df_mnlp[time_filter].time, p4_temp[time_filter]*1e9,label=label2, color='blue',mark
#plt.plot(df_mnlp[time_filter].time, p4_temp2[time_filter]*1e9,label=label3, color='black')
#plt.plot(df_mnlp[time_filter].time, df_mnlp.i4_f[time_filter]*1e9,label=label4)
plt.legend(loc='upper right')
plt.title("Probe 4 current with various band-reject filters")
plt.ylabel("Current [nA]")
plt.xlabel("Time [s]")
plt.ylim(y_lim_low,y_lim_high)
plt.grid()


plt.figure(figsize=(10,5))
#plt.plot(df_mnlp[time_filter].time, df_mnlp.i4[time_filter]*1e9,label=label1,color='red',ma
#plt.plot(df_mnlp[time_filter].time, p4_temp[time_filter]*1e9,label=label2, color='blue',mar
plt.plot(df_mnlp[time_filter].time, p4_temp2[time_filter]*1e9,label=label3, color='black')
plt.plot(df_mnlp[time_filter].time, df_mnlp.i4_f[time_filter]*1e9,label=label4)
plt.legend(loc='upper right')
plt.title("Probe 4 current with various band-reject filters")
plt.ylabel("Current [nA]")
plt.xlabel("Time [s]")
plt.ylim(y_lim_low,y_lim_high)
plt.grid()

#ax = plt.plot(df_mnlp[time_filter].time,df_mnlp.p4[time_filter])
#ax = plt.plot(df_mnlp[time_filter].time,df_mnlp.p4[time_filter])
#ax = plt.plot(df_mnlp[time_filter].time,df_mnlp.p4[time_filter])
#ax = plt.plot(p4_temp[time_fixlter])
#ax = plt.plot(p4_temp2[time_filter])
#ax = plt.plot(p4_filter[time_filter])
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>
```

After removing the spin components the plot above reveals smaller structures in the trailing / gradient part of a enhanced density structure.

Plot the complete filtered data

```
colors =['red','blue','green','orange']
y_columns = ['i1_f','i2_f','i3_f','i4_f']

# Reduce data to a subset for faster processing
```

```
t_low = 100
t_high = 450

#Returns a table with false or true for each row according to if condition is met or not.
time_filter = ( df_mnlp['time'] >= t_low) & ( df_mnlp['time'] < t_high)

ax = df_mnlp[time_filter].plot(x='time', y=y_columns,kind='line',color=colors,figsize=(10,5)


ax.set_xlabel("Time [s]");
ax.set_ylabel("Current [A]");
plt.legend(loc='upper left')
plt.grid()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

4. Calculate the electron density and plot the density curve

- The polyfit in numpy works best for data within a column.
- The four m-NLP data which should be fitted is however listed in separete columns.
- The trick is to tranpose the dataframe so that columns become rows and rows become columns. In this way, the fit can be made with a column, and the execution time will be significantly faster.

```
# Linear fitting to calcuate electron density

# Square the currents adding four new columns
df_mnlp[['i1_f_pow','i2_f_pow','i3_f_pow','i4_f_pow']] = df_mnlp[['i1_f','i2_f','i3_f','i4_f

# Reduce data to a relevent time period.
t_low = 100
t_high = 450

#Returns a table with false or true for each row according to if condition is met or not.
idx_filter = ( df_mnlp['time'] >= t_low) & ( df_mnlp['time'] < t_high)

# Extract relevant time period
df_reduced = df_mnlp[idx_filter]

# Transpose the columns containing the squared currents
temp = df_mnlp[['i1_f_pow','i2_f_pow','i3_f_pow','i4_f_pow']].T
```

```
# Fit a straight line to the four measured points for the four given bias voltages using the
fit_coef = np.polyfit(Vbias,temp,1)

#transpose the returned fit coefficients and add them to the dataframe
fit_coef = fit_coef.T
a = fit_coef[:,0]
b = fit_coef[:,1]
df_mnlp['a']= a
df_mnlp['b']= b


# Add a column with the electon density
df_mnlp['Ne'] = df_mnlp['a'].apply(calc_Ne)

df_mnlp.head()
```

```
       time     adc1     adc2     adc3     adc4            i1            i2  \
0  0.000000  32259.0  32250.0  32245.0  32298.0 -3.304989e-08 -3.363427e-08
1  0.000173  32256.0  32250.0  32243.0  32298.0 -3.324468e-08 -3.363427e-08
2  0.000691  32259.0  32249.0  32243.0  32297.0 -3.304989e-08 -3.369920e-08
3  0.000864  32260.0  32251.0  32243.0  32297.0 -3.298496e-08 -3.356934e-08
4  0.001037  32260.0  32249.0  32243.0  32300.0 -3.298496e-08 -3.369920e-08

             i3            i4          i1_f          i2_f          i3_f  \
0 -3.395892e-08 -3.051758e-08 -3.301185e-08 -3.359556e-08 -3.391984e-08
1 -3.408878e-08 -3.051758e-08 -3.313039e-08 -3.351818e-08 -3.397143e-08
2 -3.408878e-08 -3.058251e-08 -3.285943e-08 -3.350576e-08 -3.389310e-08
3 -3.408878e-08 -3.058251e-08 -3.271874e-08 -3.329872e-08 -3.381488e-08
4 -3.408878e-08 -3.038772e-08 -3.264315e-08 -3.335150e-08 -3.373676e-08

           i4_f      i1_f_pow      i2_f_pow      i3_f_pow      i4_f_pow  \
0 -3.048245e-08  1.089782e-15  1.128661e-15  1.150555e-15  9.291800e-16
1 -3.041225e-08  1.097623e-15  1.123468e-15  1.154058e-15  9.249049e-16
2 -3.040699e-08  1.079742e-15  1.122636e-15  1.148742e-15  9.245849e-16
3 -3.033681e-08  1.070516e-15  1.108805e-15  1.143446e-15  9.203220e-16
4 -3.007216e-08  1.065576e-15  1.112323e-15  1.138169e-15  9.043348e-16

              a             b  Ne
0 -2.479456e-17  1.210915e-15 NaN
1 -2.590471e-17  1.217489e-15 NaN
2 -2.420791e-17  1.202070e-15 NaN
3 -2.327954e-17  1.188810e-15 NaN
4 -2.526045e-17  1.194033e-15 NaN
```

```
# Plot the electron density
# Reduce data to a subset for faster processing
t_low = 428#138.66
```

```
t_high = 435 #138.8

#Returns a table with false or true for each row according to if condition is met or not.
time_filter = ( df_mnlp['time'] >= t_low) & ( df_mnlp['time'] < t_high)

# For comparison show data if a measurement was performed at a second rate rather then kHz.

every_nth_sample = int(sampling_rate)
print(every_nth_sample)

df_sec = df_mnlp.iloc[0::every_nth_sample]

ax = df_mnlp[time_filter].plot(x='time', y='Ne',kind='line',lw=1,color='red',figsize=(10,5)
df_sec[time_filter].plot(x='time', y='Ne',kind='line',lw=1,color='blue',figsize=(10,5), ax=

ax.set_xlabel("Time [s]");
ax.set_ylabel("Electron density [m^-3]");

plt.grid()

# We can cleary see how the smalle structures would not be visible without a high sampling
```

```
5787
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
/var/folders/67/zj89_nq11w568_yqwhxw_g3m0000gn/T/ipykernel_27369/1968513046.py:17: UserWarn
  df_sec[time_filter].plot(x='time', y='Ne',kind='line',lw=1,color='blue',figsize=(10,5), a
```

Compare the result with the plot in A. Spicher et al. "Direct evidence of double-slope power spectra in the high-latitude ionospheric plasma", Geophysical research letters, 2014.

https://doi.org/10.1002/2014GL059214