COMMUNICATIONS PROJECT

Doppler Shift Estimation

Homework Results

Authors:

Saqib FARAZ 458317
Ankita Kannan IYER 457620
Harsh GODHANI 461986
Richard GRÜNERT 289427
Kanaiyalal THUMMAR 435534

15.6.2023

Homework Tasks

```
HW8)

due June 16th 23:59 LT

Estimation of radial velocities / Doppler shift

1) Estimate the Doppler shift for all the 5 given RAW data files by using

a) moments method, e.g. adopting script 24a_read_rawdata_windMom
b) apply a fitting routine to estimate the essential parameters by assuming a
gaussian distribution

parameters to compare for both methods:
- amplitude
- Doppler shift
- spectral width

As usual provide meaningful plots, a bit of discussion and conclusions.
```

1 Introduction

For this, the the spectral representation of the received data is first computed using the FFT. Under the assumption that the absolute value of the received spectrum shows a gaussian shape, parameters of this shape (amplitude, mean, standard deviation) are estimated using two different methods. One is the Method of Moments and the other a curve fitting approach. The means of these distributions is then taken to be the doppler shifts.

1.1 Method of Moments

The Method of Moments (MoM) is a mathematical technique for evaluating and describing random processes such as noise or signal waveforms. It entails calculating multiple statistical moments of the random process in order to get insight into its characteristics such as form, distribution and various other aspects. The MoM approach allows us to compare and study various random processes based on their moments. The moments of a random process are calculated by taking weighted averages of the signal levels at distinct time instants. The instant order determines the weighting components. The moment order specifies the degree to which the signal values are raised during the averaging process.

1.2 Curve Fitting

To fit the doppler spectrum to a normal/gaussian/bell curve shape, first the noise level is calculated as the mean value of the averaged spectrum of the first five range gates and then subtracted from all spectra. Then, using the python function `scipy.optimize.curve_fit` with a gaussian model, the parameters amplitude, mean (doppler shift) and standard deviation are determined.

2 Results

For each data file (0...4), the absolute value of the combined receiver spectrum versus doppler frequency and range is shown as a colorplot for each of the two methods.

Also, for each data file, the spectrum of range gate number 35 is shown (combined receivers) together with both of the fittings from `scipy.optimize.curve_fit` and the method of moments.

A lot of these plots have missing parameter values along the range for those ranges where no fit could be determined due some issue with NaN values in the width calculation using the method of moments.

There are some with outlier values of the doppler shift. An analysis of the amplitude/power values at these points shows that the amplitude / SNR is too low for them to be considered as a correct result. In any case, one has to look at the combination of each of the graphs to make sense of them and judge whether a result is suitable for further considerations.

Comparing both methods, one finds that usually both follow a similar trajectory. Juding from the single-range-gate figures, the curve fitting approach seems to be more accurately estimating the doppler shift and amplitude. But the attractiveness of the moments-method might lie in the simplistic calculation approach which might yield better performance.

2.1 File 1

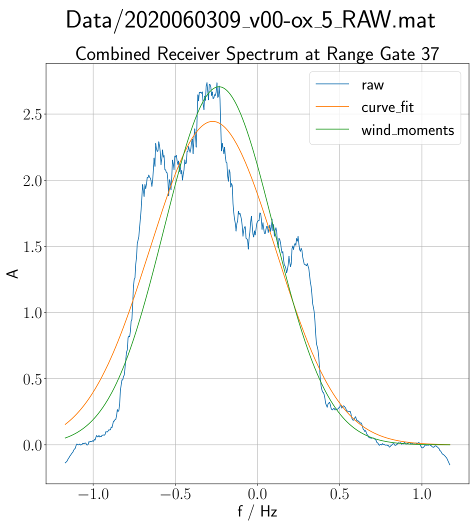

Figure 1

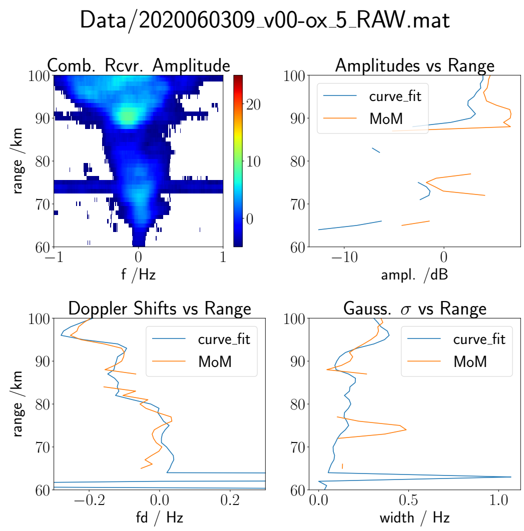

Figure 2

2.2 File 2

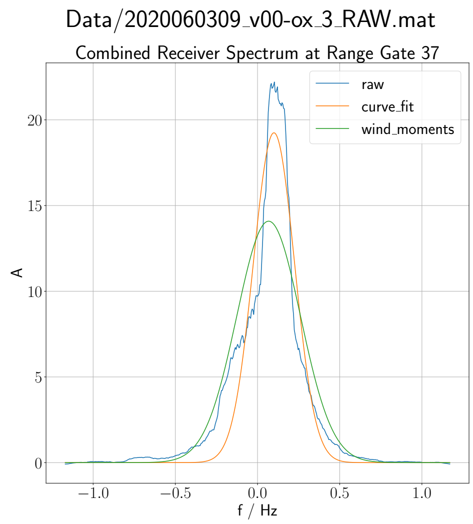

Figure 3

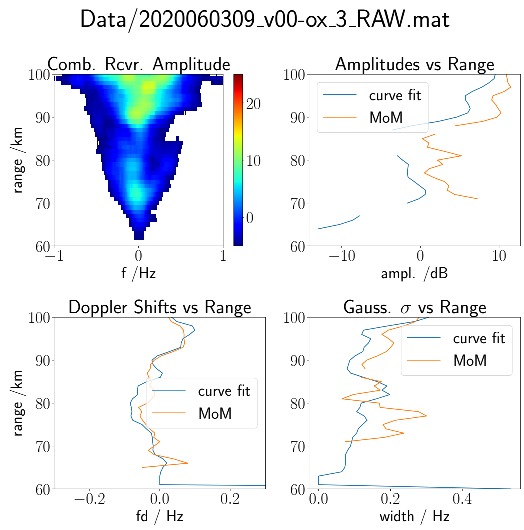

Figure 4

2.3 File 3

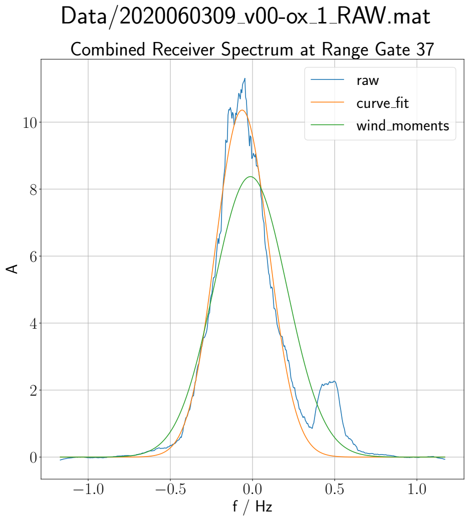

Figure 5

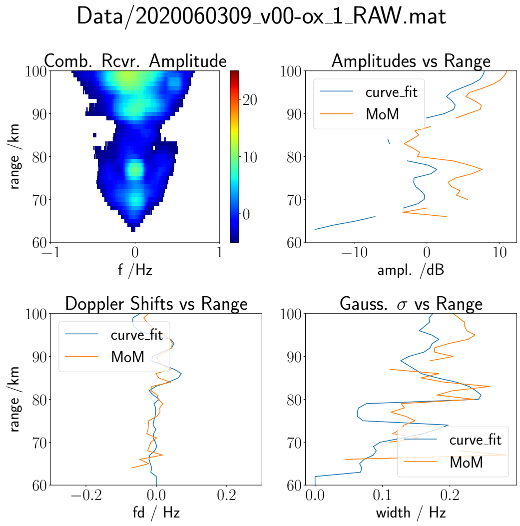

Figure 6

2.4 File 4

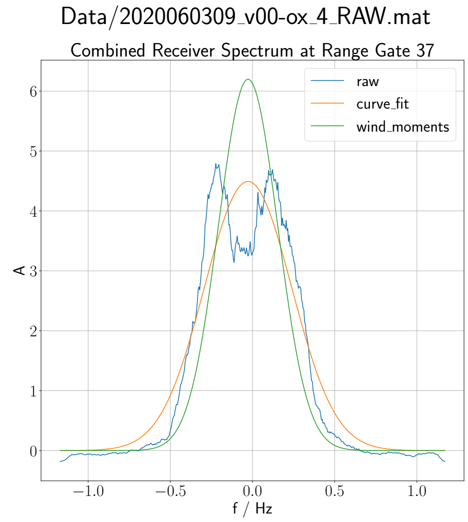

Figure 7

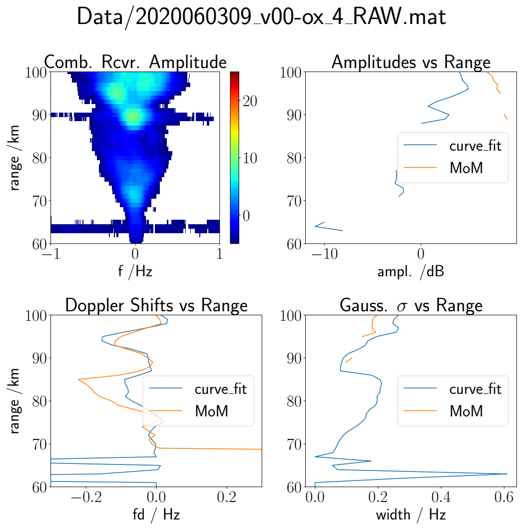

Figure 8

2.5 File 5

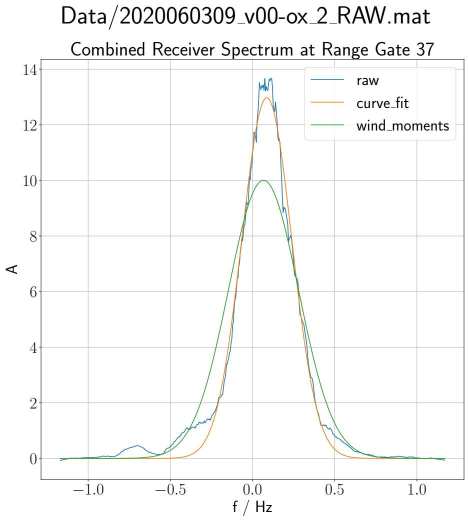

Figure 9

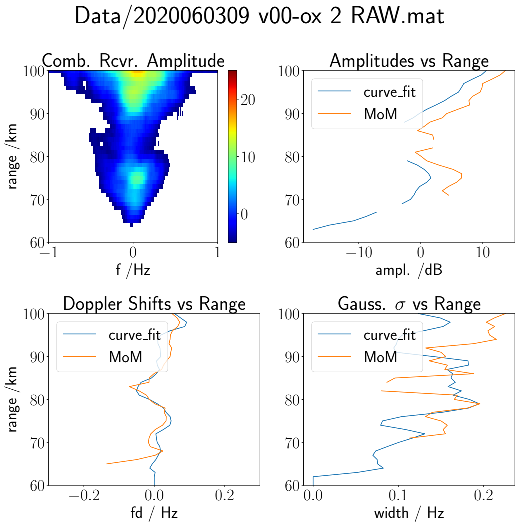

Figure 10

References

Software Used

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.7. Python Software Foundation. 2019. URL: https://www.python.org/.

[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Python Code

```
import scipy.io as spio
import os
import matplotlib.pyplot as plt
import numpy as np
import scipy


def savefig(name):
    plt.savefig("outputs/" + name)


# perform coherent integrations
def make_ci(t, y, ci):
    nptsn = int(np.floor(len(y) / ci))
    yn = np.empty(nptsn) + 1j * np.empty(nptsn)
    tn = np.empty(nptsn)
    for i in range(0, nptsn):
        yn[i] = np.mean(y[i * ci:i * ci + ci - 1])
        tn[i] = np.mean(t[i * ci:(i + 1) * ci])
    return tn, yn


# make FFT spectrum, frequency axis
def make_fft(t, y):
    dt = t[1] - t[0]  # dt -> temporal resolution ~ sample rate
    print(dt)
    f = np.fft.fftfreq(t.size, dt)  # frequency axis
    Y = np.fft.fft(y)  # FFT
    f = np.fft.fftshift(f)
    Y = np.fft.fftshift(Y) / (len(y))
    return f, Y


def wind_moments(f, spectr):

    spectr = np.abs(spectr)

    anzRG = np.size(spectr, 0)
    anzDP = np.size(spectr, 1)

    noise = np.nanmean(np.nanmean(spectr[:5, :], axis=0), axis=0)
    stds = np.std(spectr[:5, :])

    spectr = spectr - noise

    fd = np.zeros([noRG])
    width = np.zeros([noRG])
    amp = np.zeros([noRG])

    for RG in range(0, anzRG):
        # RG=15;
        spec = spectr[RG, :]
        # spec=spec-min(spec)
        # plt.figure; plt.plot(f,spec); plt.grid(1)

        sel = spec > stds * 3
        # (noise+stds*2);
        if sum(sel) > anzDP / 30:

            df = f[1] - f[0]
            # 0.-tes Moment
            m0 = df * sum(spec)
            # 1.-tes Moment
            m1 = df * sum(f * spec)
```

```
            # 2.-tes Moment
            m2 = df * sum(f ** 2 * spec)

            fd[RG] = m1 / m0
            width[RG] = np.sqrt((m2 / m0) - (m1 / m0) ** 2)
            # problem: if widths are NaNs, then amps will also be NaNs
            amp[RG] = m0 / (width[RG] * np.sqrt(2*np.pi))  # np.nansum(spec)

        else:
            fd[RG] = np.NaN
            width[RG] = np.NaN
            amp[RG] = np.nansum(spec) / (width[RG] * np.sqrt(2*np.pi))

    #    width=abs(width);

    return fd, width, amp


def smooth(y, box_pts):
    box = np.ones(box_pts) / box_pts
    y_smooth = np.convolve(y, box, mode="same")
    return y_smooth


# Gaussian general structure (does not include amplitude sigma term)
def gauss(x, *p):
    A, mu, sigma = p
    return A*np.exp(-(x-mu)**2/(2.*sigma**2))


# function for curve_fit
def gauss_for_fitting(x, *p):
    A, mu, sigma = p
    return A/(sigma * np.sqrt(2*np.pi))*np.exp(-(x-mu)**2/(2.*sigma**2))

# ----------------------------------------------------------------------------
# Load data and initialize

DataPath = "Data/"

Files = os.listdir(DataPath)

file_idx = 4
filename = str(Files[file_idx])
currentfile = str(DataPath) + filename

# importing MATLAB mat file   (containing radar raw data)
mat = spio.loadmat(currentfile, squeeze_me=True)

datenums = mat["datenums"]
ranges = mat["ranges"]
data = mat["data"]

# datenums ~ days since year 0
# here only the time is important for us -> hours, minutes, seconds
# => fraction / remainder of the integer

t = (datenums - np.floor(np.min(datenums))) * 24

# number of range gates , data points, receivers
noRG = np.size(data, 0)
noDP = np.size(data, 1)
noRx = np.size(data, 2)

RXs = np.linspace(1, noRx, noRx)

# ----------------------------------------------------------------------------
```

```
# Coherent Integrations

ci = 8

# CI window size
noDPn = int(np.floor(noDP / ci))

datan = np.zeros([noRG, noDPn, noRx]) + 1j * np.zeros([noRG, noDPn, noRx])

for rx in range(noRx):
    for rg in range(noRG):
        tn, datan[rg, :, rx] = make_ci(t, data[rg, :, rx], ci)

# update data size and t according to CI
data = datan
t = tn
noDP = np.size(data, 1)

# time vector in s
tsec = t * 60 * 60


# ----------------------------------------------------------------------------
# Spectra for all ranges and all receivers

Spectr = np.zeros([noRG, noDP, noRx]) + 1j * np.zeros([noRG, noDP, noRx])

for rx in range(noRx):
    for rg in range(noRG):
        f, Spectr[rg, :, rx] = make_fft(tsec, data[rg, :, rx])

SpectrSm = np.zeros([noRG, noDP, noRx])

# smooth all spectra
for rx in range(noRx):
    for rg in range(noRG):
        SpectrSm[rg, :, rx] = smooth(abs(Spectr[rg, :, rx]), 31)


# combine spectra of all receivers by taking the median of them
SpectrComb = np.nanmedian(Spectr, axis=2)

SpectrCombSm = np.zeros([noRG, noDP])

# smoothed median spectrum
for rg in range(noRG):
    SpectrCombSm[rg, :] = smooth(abs(SpectrComb[rg, :]), 31)

# moments for the combined receiver
# (fd, width, amp) -> one for each range gate
fd, width, amp = wind_moments(f, SpectrCombSm)

print("fd: ", fd)
print("width: ", width)
print("amp: ", amp)

# individually for all Receiver

Fd = np.zeros([noRG, noRx])
Width = np.zeros([noRG, noRx])
Amp = np.zeros([noRG, noRx])

for rx in range(noRx):
    # rx=0
    Fd[:, rx], Width[:, rx], Amp[:, rx] = wind_moments(f, SpectrSm[:, :, rx])

spectr1 = np.abs(SpectrCombSm)
```

```python

anzRG = np.size(spectr1, 0)
anzDP = np.size(spectr1, 1)

# a=np.nanmedian(spectr[:5,:],axis=0)
# noise -> average spectra of first 5 range gates (incoherent int.)
#       -> mean of that
noise = np.nanmean(np.nanmean(spectr1[:5, :], axis=0), axis=0)
stds = np.std(spectr1[:5, :])

spectr1 = spectr1 - noise

# range gate select
RGi = 37

x0 = [1.0, 0.0, 1.0]
coeff2, var_matrix2 = scipy.optimize.curve_fit(gauss_for_fitting,
                                               f, spectr1[RGi, :],
                                               p0=x0)

fd_mom, width_mom, amp_mom = wind_moments(f, SpectrComb)
fd2 = fd_mom[RGi]
sig2 = width_mom[RGi]
amp2 = amp_mom[RGi]

# curve_fit for all range gates
amps_cf = np.zeros(noRG)
fd_cf = np.zeros(noRG)
sig_cf = np.zeros(noRG)
for rg in range(0, noRG):
    try:
        coeff, dump = scipy.optimize.curve_fit(gauss_for_fitting,
                                               f, spectr1[rg, :],
                                               p0=x0)
        amps_cf[rg], fd_cf[rg], sig_cf[rg] = coeff
        sig_cf[rg] = np.abs(sig_cf[rg])
        print(coeff)
    except RuntimeError:
        # awful error handling
        print("errorororor")


# ----------------------------------------------------------------------------
# Plotting

print("----")
print(sig_cf[noRG-1])
print("----")

plt.figure()
plt.suptitle(currentfile)
plt.title("Combined Receiver Spectrum at Range Gate " + str(RGi))
plt.xlabel("f / Hz")
plt.ylabel("A")
plt.plot(f, spectr1[RGi, :], label="raw")
plt.plot(f, gauss_for_fitting(f, *coeff2), label="curve_fit")
plt.plot(f, gauss(f, *(amp2, fd2, sig2)), label="wind_moments")
plt.legend()
plt.grid(True)
savefig("data_" + str(file_idx) + "_single_rg.pdf")
plt.tight_layout()

# ---------------------------------
# curve_fit

plt.figure()
plt.subplot(2, 2, 1)
```

```
ampl = 10 * np.log10(SpectrCombSm)
SNRsel = ampl < -5
ampl[SNRsel] = "nan"
plt.pcolor(f, ranges, ampl, cmap="jet")
plt.clim([-5, 25])
plt.xlim([-1, 1])
plt.xlabel("f /Hz")
plt.ylabel("range /km")
plt.colorbar()
plt.ylim([min(ranges), max(ranges)])
plt.suptitle(currentfile)
plt.title("Comb. Rcvr. Amplitude")

# parameters for MoM and curve_fit
ax = plt.subplot(2, 2, 2)
plt.plot(10 * np.log10(amps_cf), ranges, label="curve_fit")
plt.plot(10 * np.log10(amp_mom), ranges, label="MoM")
plt.xlabel("ampl. /dB")
plt.ylim([min(ranges), max(ranges)])
plt.title("Amplitudes vs Range")
ax.legend()

ax = plt.subplot(2, 2, 3)
plt.plot(fd_cf, ranges, label=("curve_fit"))
plt.plot(fd_mom, ranges, label=("MoM"))
plt.ylabel("range /km")
plt.xlabel("fd / Hz")
plt.title("Doppler Shifts vs Range")
plt.xlim([-0.3, 0.3])
plt.ylim([min(ranges), max(ranges)])
ax.legend()

ax = plt.subplot(2, 2, 4)
plt.plot(sig_cf, ranges, label="curve_fit")
plt.plot(width_mom, ranges, label="MoM")
plt.xlabel("width / Hz")
plt.title("Gauss. $\\sigma$ vs Range")
plt.ylim([min(ranges), max(ranges)])
ax.legend()
plt.tight_layout()
savefig("data_" + str(file_idx) + "_quad" + ".pdf")

plt.show()
```