PyDynamic tutorials Documentation

Sascha Eichstädt, Björn Ludwig

Sep 10, 2020

GETTING STARTED

1 *PyDynamic tutorials* 3

2 Basic measurement data pre-processing 7

3 Preparation of calibration data 11

4 Interpolation and extrapolation of calibration data 15

5 Calculation of impulse response of hydrophone 21

6 Deconvolution in the frequency domain 23

7 Regularized deconvolution 27

8 `PyDynamic.uncertainty.interpolation` 35

9 Indices and tables 39

PyDynamic_tutorials is a collection of tutorials based on Jupyter notebooks which are designed to simplify the handling of PyDynamic. The tutorials range from introductory examples to more in-depth use cases in the context of our work at PTB.

For the *PyDynamic_tutorials* homepage go to GitHub.

PyDynamic_tutorials are written in Python 3.

CHAPTER

ONE

PYDYNAMIC TUTORIALS

This is a collection of tutorials and examples to document, explain and illustrate the possibilities offered by the parent library *PyDynamic*. We will add more and more examples over time, especially those that are currently included in itself.

1.1 Getting started

To get going with the tutorials you can either start directly in your browser or get a local copy and experiment offline on your machine.

1.1.1 Quick start in current browser session

To start working in the notebooks directly in the browser, click .

1.1.2 Get a local copy to work offline

To get started on your local machine, follow these simple steps:

1. Clone the repository, if you haven't already
2. Set up a virtual environment for PyDynamic_tutorials
3. Install the dependencies
4. Start the Jupyter Notebook server
5. Go to localhost:8888 with your favourite browser
6. Browse the various examples in the repository, alter and execute the code right in your browser

1. Clone the repository

```
$ git clone https://github.com/PTB-PSt1/PyDynamic_tutorials.git
Cloning into 'PyDynamic_tutorials'...
[...]
Receiving objects: 100% (3/3), done.
$
```

2. Set up a virtual environment

On your command line/powershell execute:

```
$ python3 -m venv PyDynamic_tutorial_venv
$
```

This will create a subfolder *PyDynamic_tutorial_venv* and prepare a fully self-contained Python environment, which we can activate in the next step and install further Python packages without polluting or disturbing your system environment.

3. Install the dependencies

First we activate the previously created environment before we then install the required dependencies in two steps, because we are utilizing `pip-tools` to ensure you get a working copy of our environments.

```
$ source PyDynamic_tutorial_venv/bin/activate
(PyDynamic_tutorial_venv) $ pip install --upgrade pip pip-tools
Collecting pip
[...]
Successfully installed click-7.1.2 pip-20.1.1 pip-tools-5.2.0 setuptools-47.1.1 six-1.
↪15.0
$ pip-sync requirements/requirement.txt
Collecting attrs==19.3.0
[...]
Installing collected packages: attrs, [...]
webencodings-0.5.1
$
```

4. Start the notebook server

Now from the environment we created previously, start up the Jupyter Notebook server.

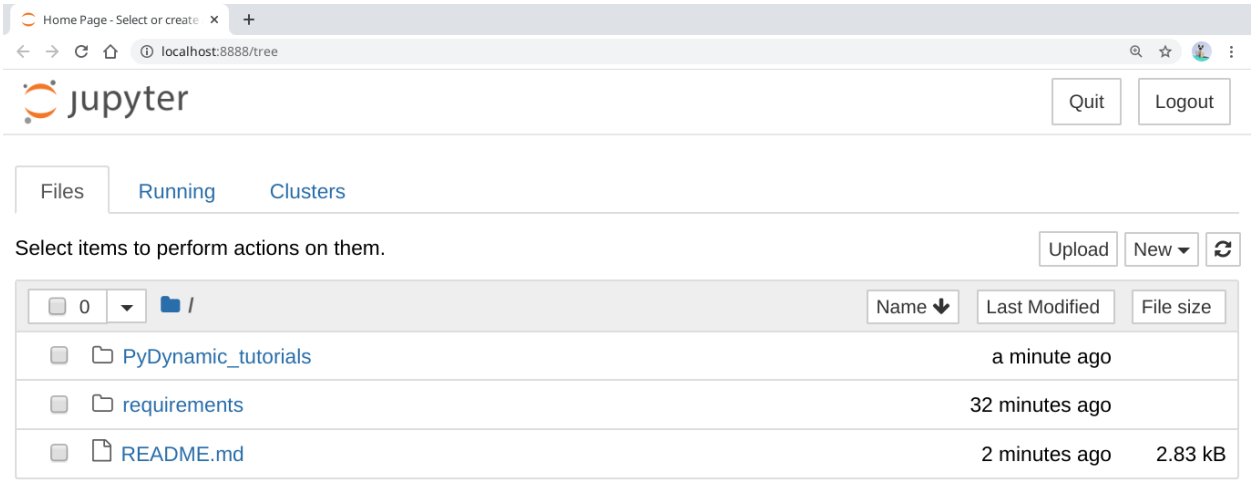
```
$ jupyter notebook
[I 13:01:24.790 NotebookApp] Serving notebooks from local directory: ~/code/PyDynamic_
↪tutorials
[I 13:01:24.790 NotebookApp] The Jupyter Notebook is running at:
[I 13:01:24.790 NotebookApp] http://localhost:8888/?
↪token=f368c552e042d48d46ff4c8a094448d0e7681790b0719215
```

5. Go to localhost:8888

Usually a browser window will have opened automatically at this point. Otherwise just open one yourself and navigate to the printed URL in the console, in our case `http://localhost:8888/?token=f368c552e042d48d46ff4c8a094448d0e7681790b0719215`.

6. Browse the various examples

You should see something like the following:



After a click on *PyDynamic_tutorials* the source code can be edited and executed directly in the browser.

1.2 Documentation

The detailed documentation of *PyDynamic*'s source code is available on pydynamic.readthedocs.io. The notebooks in this repository are built into these pages along with additional material.

CHAPTER

TWO

BASIC MEASUREMENT DATA PRE-PROCESSING

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
from meas_data_preprocessing import *
```

```
/Users/sascha/opt/anaconda3/lib/python3.7/site-packages/PyDynamic/identification/fit_
↪filter.py:34: DeprecationWarning: The module *identification* will be combined with␣
↪the module *deconvolution* and renamed to *model_estimation* in the next major␣
↪release 2.0.0. From then on you should only use the new module *model_estimation*␣
↪instead.
  DeprecationWarning)
```

2.1 Read data for a selected measurement scenario

[3]:
```
infos, measurement_data = read_data(meas_scenario = 13)
```

```
The file MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT was read and it contains 2500␣
↪data points.
The time increment is 2e-09 s
```

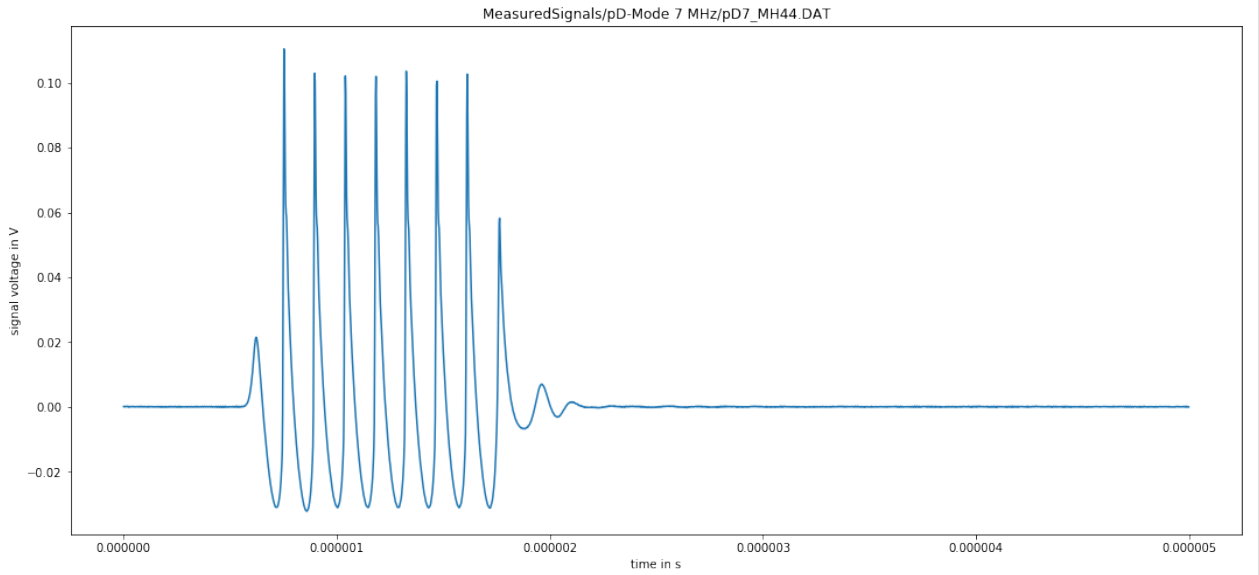
[4]:
```
# metadata for chosen measurement scenario
for key in infos.keys():
    print("%20s: %s" %(key,infos[key]))
```

```
                   i: 13
      hydrophonname: GAMPT MH44
    measurementtype: Pulse-Doppler-Mode 7 MHz
    measurementfile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT
          noisefile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT
        hydfilename: HydrophonCalibrationData/MW_MH44ReIm.csv
```

[5]:
```
# available measurement data
for key in measurement_data.keys():
    print("%10s: %s"%(key, type(measurement_data[key])))
```

```
      name: <class 'str'>
   voltage: <class 'numpy.ndarray'>
      time: <class 'numpy.ndarray'>
```

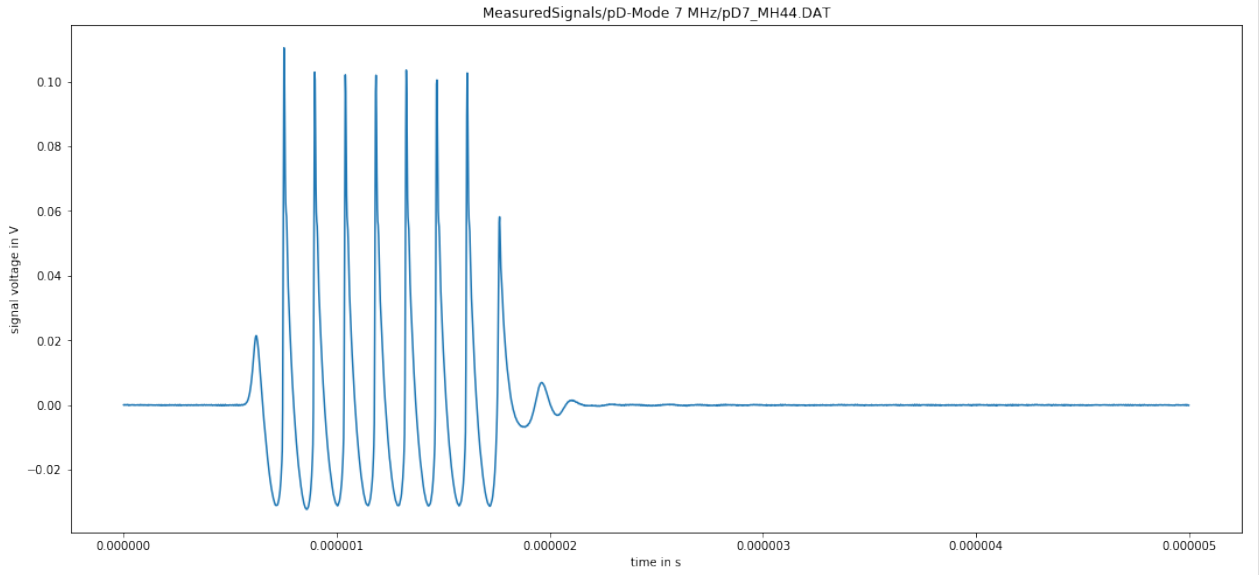
[6]:
```
figure(figsize=(18,8))
plot(measurement_data["time"], measurement_data["voltage"])
xlabel("time in s"); ylabel("signal voltage in V")
title(measurement_data["name"]);
```



2.2 Remove DC component

[7]:
```
measurement_data = remove_DC_component(measurement_data)
```

[8]:
```
figure(figsize=(18,8))
plot(measurement_data["time"], measurement_data["voltage"])
xlabel("time in s"); ylabel("signal voltage in V")
title(measurement_data["name"]);
```



2.3 Calculate measurement uncertainty from noise data

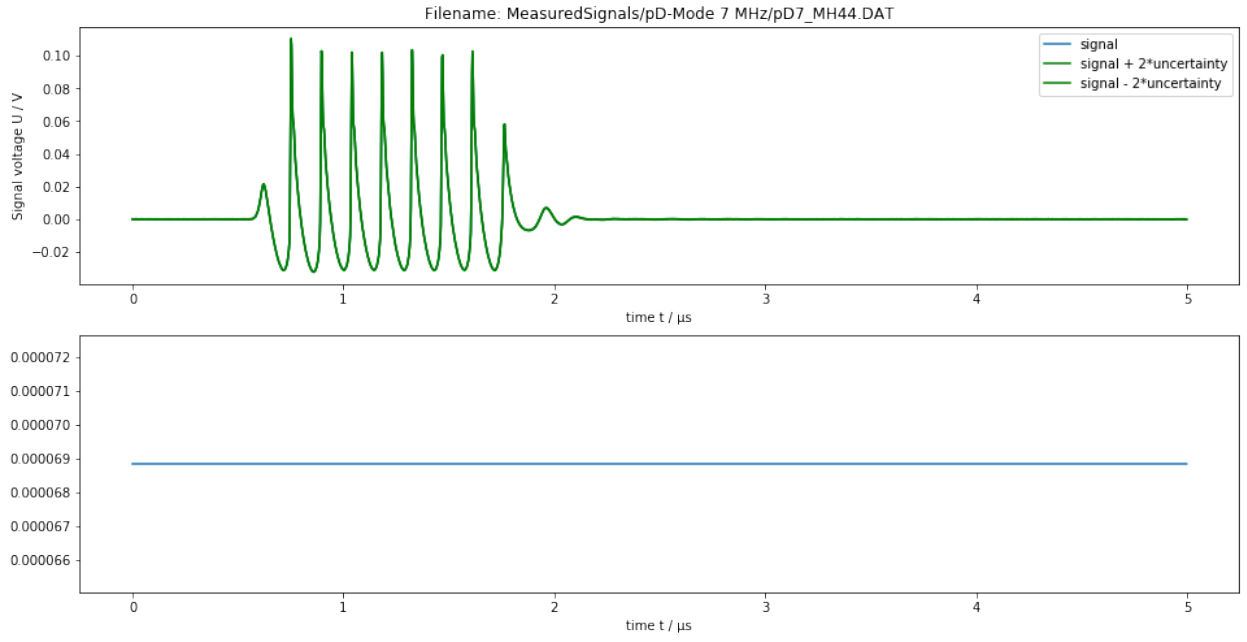
[9]:
```
measurement_data = uncertainty_from_noisefile(infos, measurement_data, do_plot=False)
```

```
The file "MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT" was read and it contains 2500␣
↪data points
```

[10]:
```
# available measurement data
for key in measurement_data.keys():
    print("%12s: %s"%(key, type(measurement_data[key])))
```

```
        name: <class 'str'>
     voltage: <class 'numpy.ndarray'>
        time: <class 'numpy.ndarray'>
 uncertainty: <class 'numpy.ndarray'>
```

[11]:
```
figure(figsize=(16,8))
subplot(211)
plot(measurement_data["time"] / 1e-6, measurement_data["voltage"])
plot(measurement_data["time"] / 1e-6, measurement_data["voltage"] + 2 * measurement_
↪data["uncertainty"], "g")
plot(measurement_data["time"] / 1e-6, measurement_data["voltage"] - 2 * measurement_
↪data["uncertainty"], "g")
legend(["signal", "signal + 2*uncertainty", "signal - 2*uncertainty"])
xlabel("time t / µs")
ylabel("Signal voltage U / V")
title("Filename: {}".format(measurement_data["name"]))

subplot(212)
plot(measurement_data["time"] / 1e-6, measurement_data["uncertainty"])
xlabel("time t / µs");
```

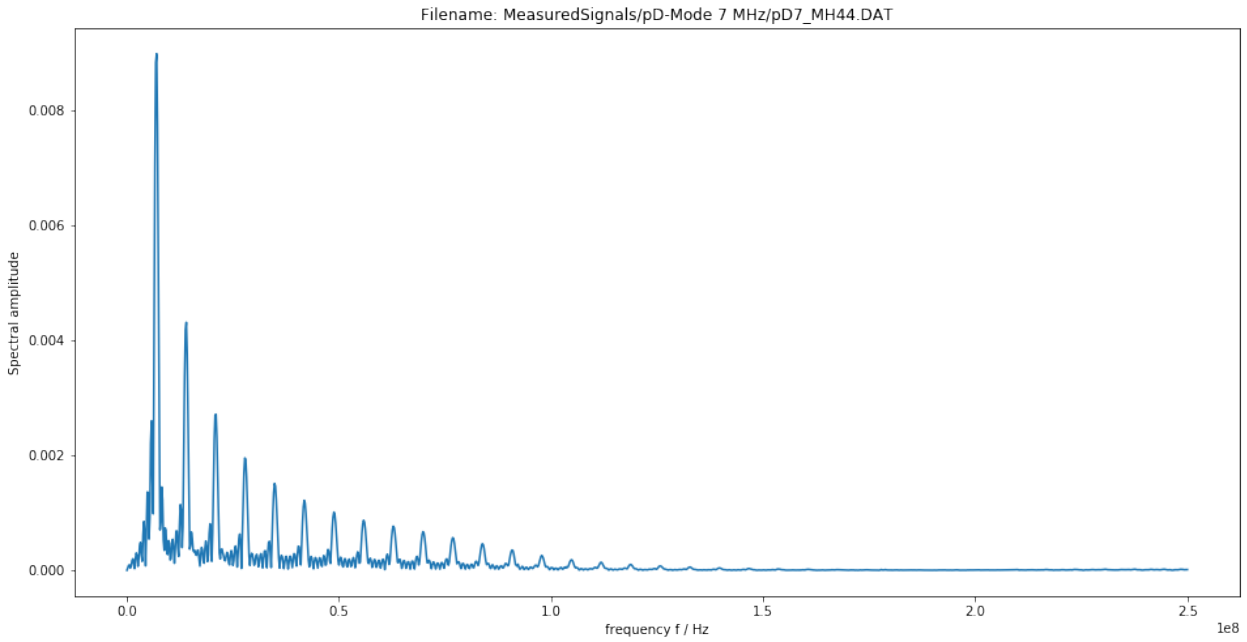


2.4 Calculate spectrum of measured data

[13]:
```
measurement_data = calculate_spectrum(measurement_data, do_plot = False)
```

```
        name: <class 'str'>
     voltage: <class 'numpy.ndarray'>
        time: <class 'numpy.ndarray'>
 uncertainty: <class 'numpy.ndarray'>
   frequency: <class 'numpy.ndarray'>
    spectrum: <class 'numpy.ndarray'>
     varspec: <class 'numpy.ndarray'>
```

[14]:
```
# available measurement data
for key in measurement_data.keys():
    print("%12s: %s"%(key, type(measurement_data[key])))
```



[ ]:
```
figure(figsize=(16,8))
plot(realpart(measurement_data["frequency"]), amplitude(measurement_data["spectrum"]))
xlabel("frequency f / Hz")
ylabel("Spectral amplitude")
title("Filename: {}".format(measurement_data["name"]));
```

CHAPTER

THREE

PREPARATION OF CALIBRATION DATA

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
from hydrophone_data_preprocessing import *
```

```
/Users/sascha/opt/anaconda3/lib/python3.7/site-packages/PyDynamic/identification/fit_
↪filter.py:34: DeprecationWarning: The module *identification* will be combined with␣
↪the module *deconvolution* and renamed to *model_estimation* in the next major␣
↪release 2.0.0. From then on you should only use the new module *model_estimation*␣
↪instead.
  DeprecationWarning)
```

3.1 Read calibration data for selected measurement scenario

[3]:
```
infos, hyd_data = read_calib_data(meas_scenario = 13, do_plot = False)
```

[4]:
```
# metadata for chosen measurement scenario
for key in infos.keys():
    print("%20s: %s" %(key,infos[key]))
```

```
                   i: 13
        hydrophonname: GAMPT MH44
     measurementtype: Pulse-Doppler-Mode 7 MHz
     measurementfile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT
           noisefile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT
         hydfilename: HydrophonCalibrationData/MW_MH44ReIm.csv
```

[5]:
```
# available measurement data
for key in hyd_data.keys():
    print("%10s: %s"%(key, type(hyd_data[key])))
```

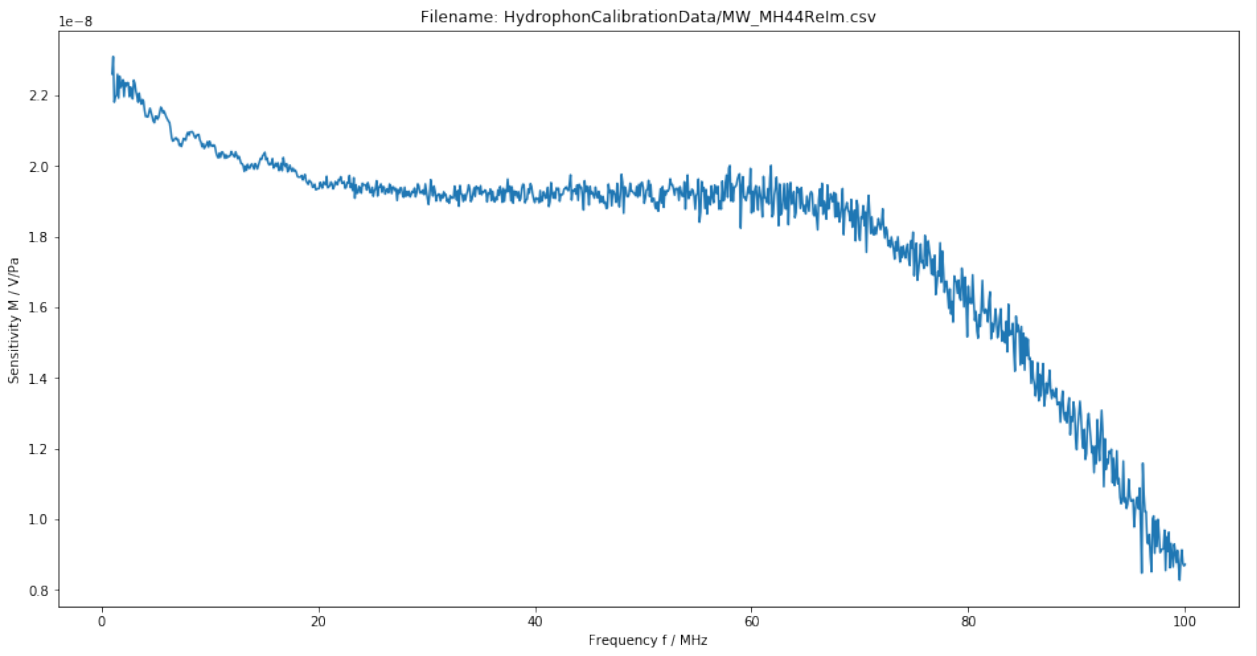
```
      name: <class 'str'>
 frequency: <class 'numpy.ndarray'>
      real: <class 'numpy.ndarray'>
      imag: <class 'numpy.ndarray'>
   varreal: <class 'numpy.ndarray'>
   varimag: <class 'numpy.ndarray'>
       cov: <class 'numpy.ndarray'>
```

3.1.1 Reduce frequency range

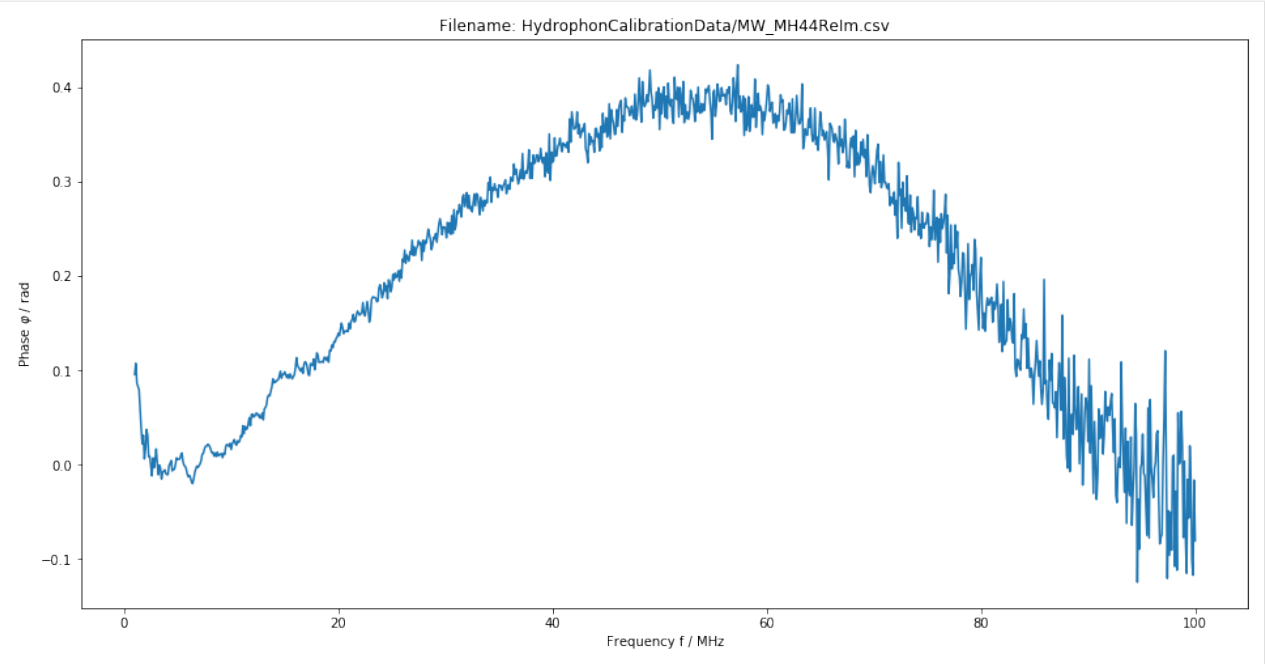
[6]:
```
hyd_data = reduce_freq_range(hyd_data, fmin = 1e6, fmax = 100e6)
```

3.1.2 Plot amplitude and phase data

[7]:
```
figure(figsize=(16,8))
plot(hyd_data["frequency"] / 1E6, np.sqrt(hyd_data["real"] ** 2 + hyd_data["imag"] **␣
↪2))
xlabel("Frequency f / MHz")
ylabel("Sensitivity M / V/Pa")
title("Filename: {}".format(hyd_data["name"]));
```



[8]:
```
figure(figsize=(16,8))
plot(hyd_data["frequency"] / 1E6, np.arctan2(hyd_data["imag"], hyd_data["real"]))
xlabel("Frequency f / MHz")
ylabel(r"Phase $\varphi$ / rad")
title("Filename: {}".format(hyd_data["name"]));
```


Filename: HydrophonCalibrationData/MW_MH44ReIm.csv
Phase φ / rad
0.4
0.3
0.2
0.1
0.0
−0.1
0
20
40
60
80
100
Frequency f / MHz

CHAPTER

FOUR

INTERPOLATION AND EXTRAPOLATION OF CALIBRATION DATA

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
from meas_data_preprocessing import *
from hydrophone_data_preprocessing import *
```

```
/Users/sascha/opt/anaconda3/lib/python3.7/site-packages/PyDynamic/identification/fit_
↪filter.py:34: DeprecationWarning: The module *identification* will be combined with␣
↪the module *deconvolution* and renamed to *model_estimation* in the next major␣
↪release 2.0.0. From then on you should only use the new module *model_estimation*␣
↪instead.
  DeprecationWarning)
```

[3]:
```
from PyDynamic.uncertainty.interpolation import interp1d_unc
```

4.1 Read measured data and calibration data from file

[4]:
```
meas_scenario = 13
infos, measurement_data = read_data(meas_scenario = meas_scenario)
_, hyd_data = read_calib_data(meas_scenario = meas_scenario, do_plot = False)
```

```
The file MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT was read and it contains 2500␣
↪data points.
The time increment is 2e-09 s
```

[5]:
```
# metadata for chosen measurement scenario
for key in infos.keys():
    print("%20s: %s" %(key,infos[key]))
```

```
                   i: 13
      hydrophonname: GAMPT MH44
    measurementtype: Pulse-Doppler-Mode 7 MHz
    measurementfile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT
          noisefile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT
        hydfilename: HydrophonCalibrationData/MW_MH44ReIm.csv
```

4.2 Perform basic pre-processing

[6]:
```
# remove DC component in measured data
measurement_data = remove_DC_component(measurement_data)
```

[7]:
```
# reduce frequency range of calibration data
hyd_data = reduce_freq_range(hyd_data, fmin = 1e6, fmax = 100e6)
```

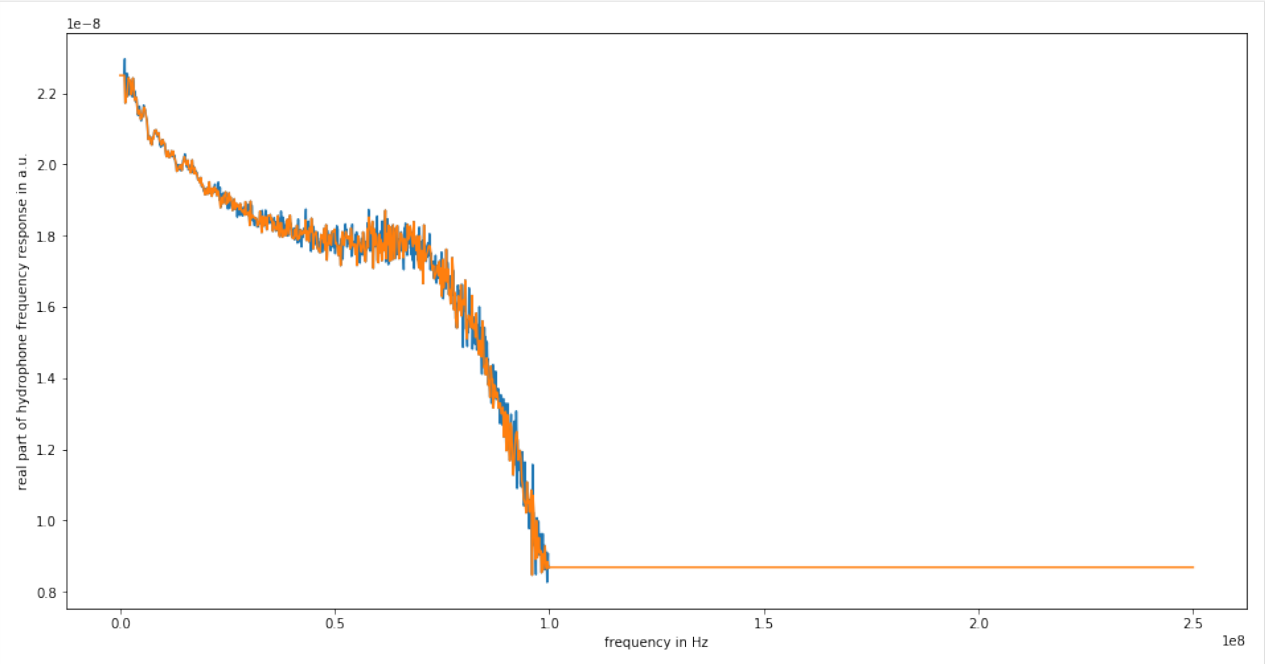
4.3 Align spectral data of calibration and measured data

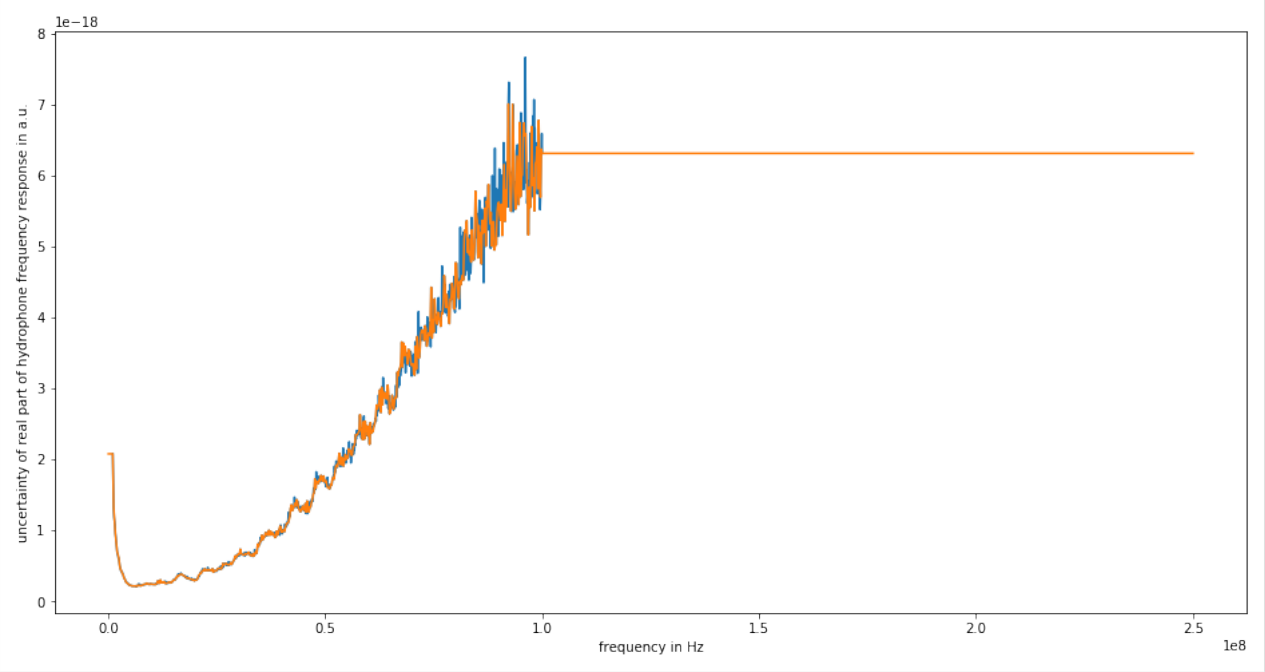
[8]:
```
measurement_data = uncertainty_from_noisefile(infos, measurement_data, do_plot=False,␣
↪verbose=False)
measurement_data = calculate_spectrum(measurement_data, do_plot = False)
fmeas = measurement_data["frequency"].round()
N = len(fmeas)//2
```

4.3.1 Interpolation of real part

[9]:
```
hyd_interp = dict([])
hyd_interp["frequency"], hyd_interp["real"], hyd_interp["varreal"], Creal = interp1d_
↪unc(
    fmeas[:N], hyd_data["frequency"], hyd_data["real"], hyd_data["varreal"],
    bounds_error=False, fill_value="extrapolate", fill_unc="extrapolate",␣
↪returnC=True)
```

[10]:
```
figure(figsize=(16,8))
plot(hyd_data["frequency"], hyd_data["real"])
plot(hyd_interp["frequency"], hyd_interp["real"])
xlabel("frequency in Hz")
ylabel("real part of hydrophone frequency response in a.u.")

figure(figsize=(16,8))
plot(hyd_data["frequency"], hyd_data["varreal"])
plot(hyd_interp["frequency"], hyd_interp["varreal"])
xlabel("frequency in Hz")
ylabel("uncertainty of real part of hydrophone frequency response in a.u.");
```

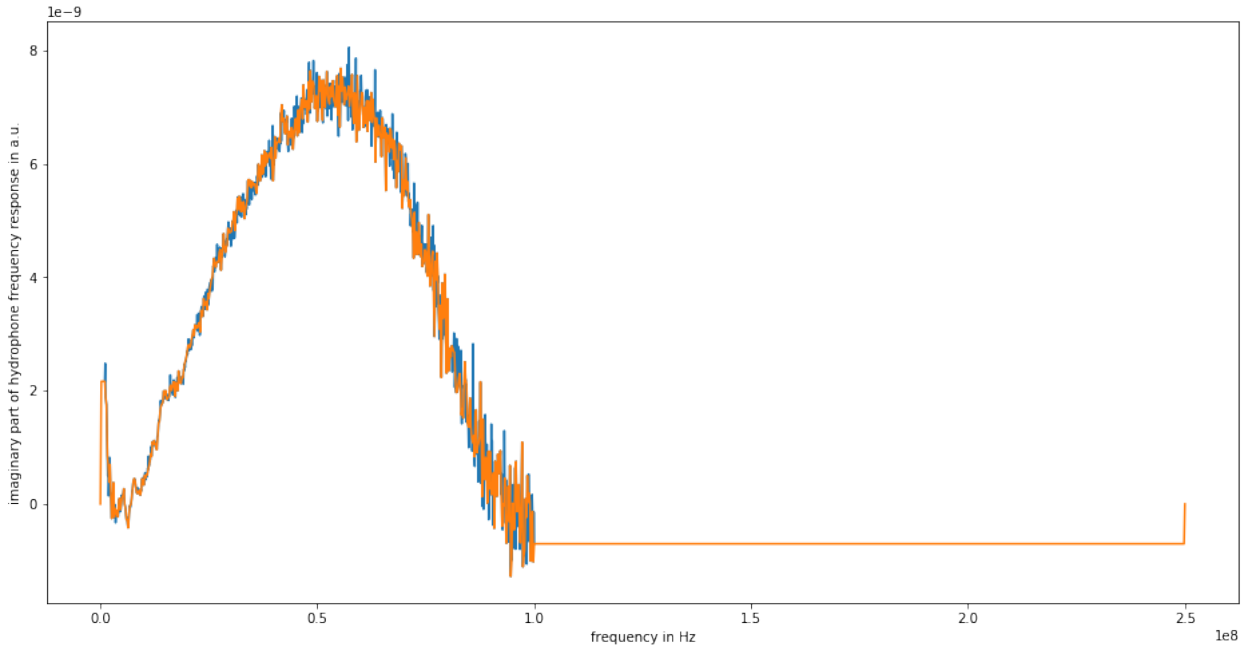

1e−8
real part of hydrophone frequency response in a.u.
frequency in Hz
1e8

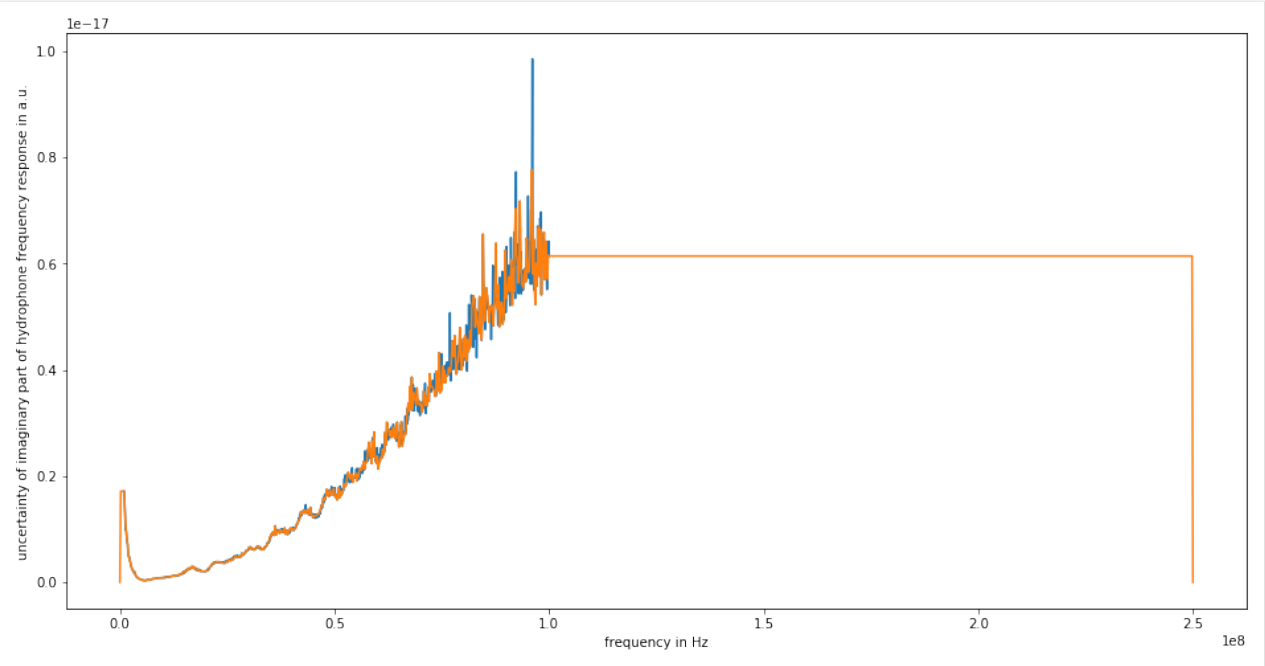

1e−18
uncertainty of real part of hydrophone frequency response in a.u.
frequency in Hz
1e8

4.3.2 Interpolation of imaginary part

[11]:
```
hyd_interp["frequency"], hyd_interp["imag"], hyd_interp["varimag"], Cimag = interp1d_
↪unc(
    fmeas[:N], hyd_data["frequency"], hyd_data["imag"], hyd_data["varimag"],
    bounds_error=False, fill_value="extrapolate", fill_unc="extrapolate",␣
↪returnC=True)
# adjustment of end points
hyd_interp["imag"][0] = 0   # Must be 0 by definition
hyd_interp["imag"][-1] = 0
hyd_interp["varimag"][0] = 0   # Must be 0 by definition
hyd_interp["varimag"][-1] = 0
```

[12]:
```
figure(figsize=(16,8))
plot(hyd_data["frequency"], hyd_data["imag"])
plot(hyd_interp["frequency"], hyd_interp["imag"])
xlabel("frequency in Hz")
ylabel("imaginary part of hydrophone frequency response in a.u.")

figure(figsize=(16,8))
plot(hyd_data["frequency"], hyd_data["varimag"])
plot(hyd_interp["frequency"], hyd_interp["varimag"])
xlabel("frequency in Hz")
ylabel("uncertainty of imaginary part of hydrophone frequency response in a.u.");
```



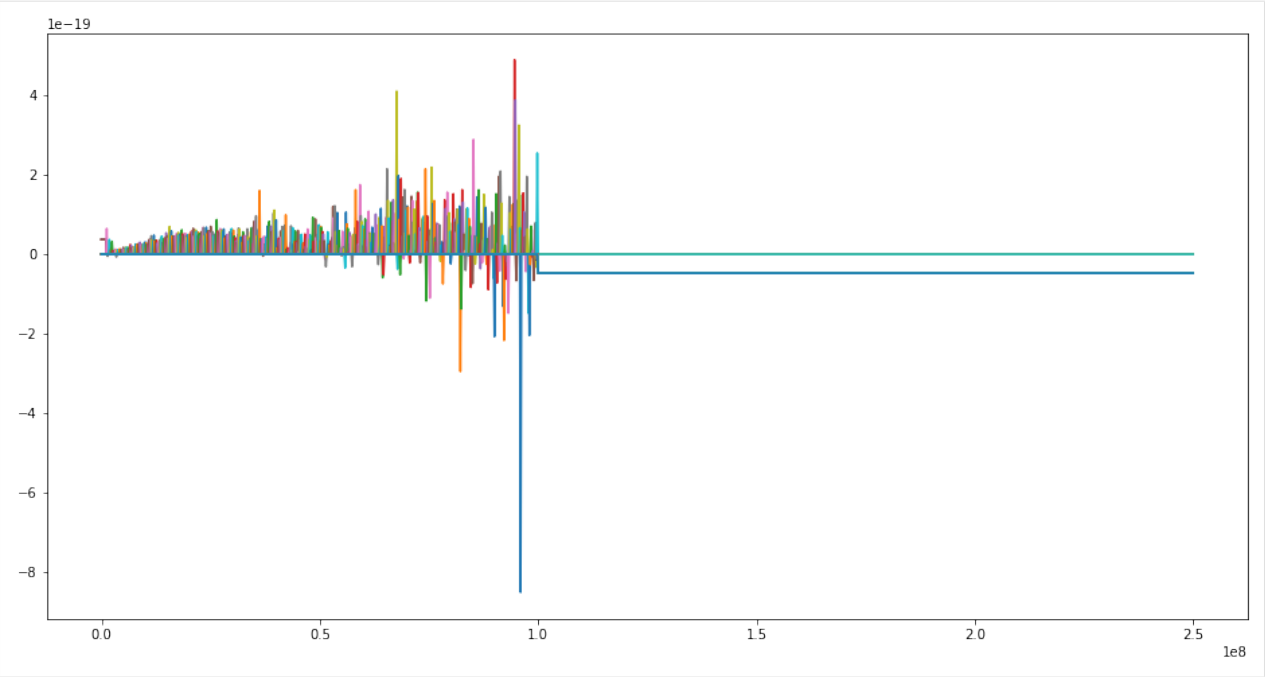


4.3.3 Calculation of mixed uncertainties at new frequencies

$$U_{r_{interp},i_{interp}} = C_r U_{r,i} C_i^T$$

[13]:
```
hyd_interp["cov"] = (Creal.dot(diag(hyd_data["cov"]))).dot(Cimag.T)
```

[14]:
```
figure(figsize=(16,8))
plot(hyd_interp["frequency"], hyd_interp["cov"]);
```


1e−19
4
2
0
−2
−4
−6
−8
0.0
0.5
1.0
1.5
2.0
2.5
1e8

CHAPTER

FIVE

CALCULATION OF IMPULSE RESPONSE OF HYDROPHONE

[1]:
```
%pylab inline
```
```
Populating the interactive namespace from numpy and matplotlib
```

[8]:
```
from meas_data_preprocessing import *
from hydrophone_data_preprocessing import *
from PyDynamic.uncertainty.propagate_DFT import GUM_iDFT
```

5.1 Load calibration data

[3]:
```
meas_scenario = 13
infos, measurement_data = read_data(meas_scenario = meas_scenario)
_, hyd_data = read_calib_data(meas_scenario = meas_scenario, do_plot = False)
```
```
The file MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT was read and it contains 2500␣
↪data points.
The time increment is 2e-09 s
```

5.2 Align calibration data with measurement data

[4]:
```
# reduce frequency range of calibration data
hyd_data = reduce_freq_range(hyd_data, fmin = 1e6, fmax = 100e6)
```

[5]:
```
measurement_data = uncertainty_from_noisefile(infos, measurement_data, do_plot=False,␣
↪verbose=False)
measurement_data = calculate_spectrum(measurement_data, do_plot = False)
fmeas = measurement_data["frequency"].round()
```

[6]:
```
hyd_interp = interpolate_hyd(hyd_data, fmeas)
```

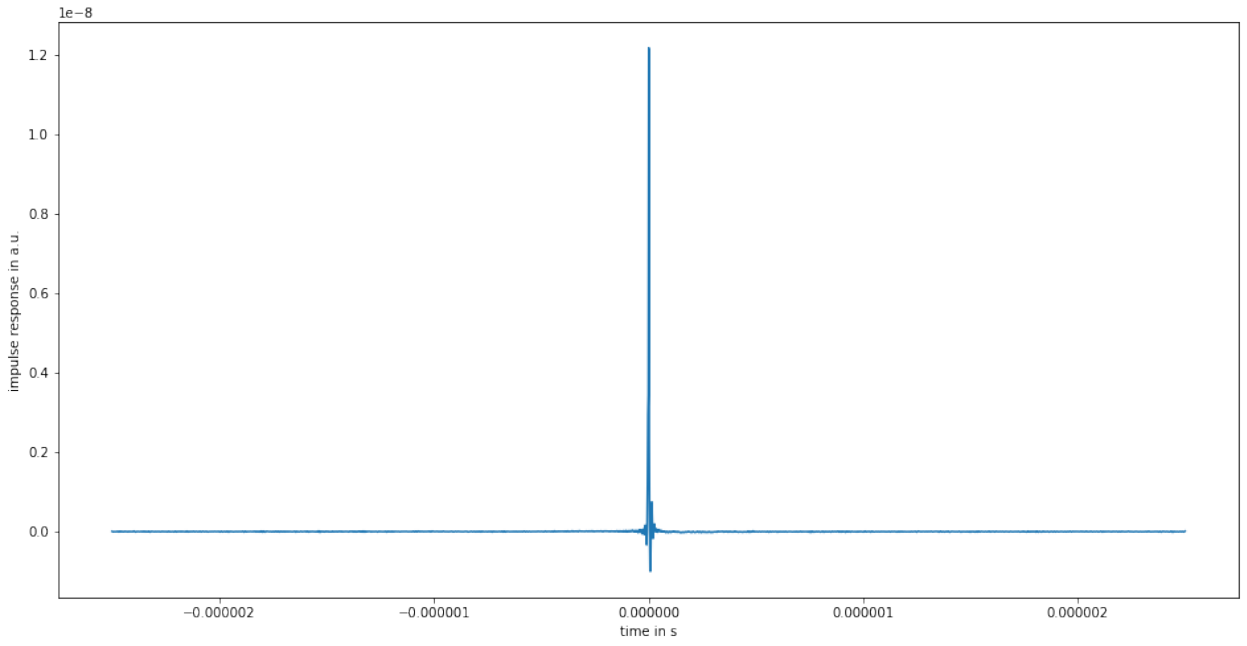
5.3 Transform to time domain to calculate impulse response

[14]:
```
H_RI = np.r_[hyd_interp["real"],hyd_interp["imag"]]
U_HRI = np.r_[
    np.c_[np.diag(hyd_interp["varreal"]), hyd_interp["cov"]],
    np.c_[hyd_interp["cov"], np.diag(hyd_interp["varimag"])]]
```

[16]:
```
# application of inverse Fourier transform
imp, Uimp = GUM_iDFT(H_RI, U_HRI)
```

[17]:
```
# centralisation of impulse response
dt = 1/(hyd_interp["frequency"][1] - hyd_interp["frequency"][0])
c_time = linspace(-dt/2,dt/2,np.size(imp))
c_imp = np.fft.fftshift(imp)
```

[18]:
```
figure(figsize=(16,8))
plot(c_time, c_imp)
xlabel("time in s")
ylabel("impulse response in a.u.");
```



CHAPTER

SIX

DECONVOLUTION IN THE FREQUENCY DOMAIN

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[15]:
```
from meas_data_preprocessing import *
from hydrophone_data_preprocessing import *
from PyDynamic.uncertainty.propagate_DFT import DFT_deconv, GUM_iDFT
```

6.1 Load calibration data

[3]:
```
meas_scenario = 13
infos, measurement_data = read_data(meas_scenario = meas_scenario)
_, hyd_data = read_calib_data(meas_scenario = meas_scenario, do_plot = False)
```

```
The file MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT was read and it contains 2500␣
↪data points.
The time increment is 2e-09 s
```

[4]:
```
# metadata for chosen measurement scenario
for key in infos.keys():
    print("%20s: %s" %(key,infos[key]))
```

```
                   i: 13
      hydrophonname: GAMPT MH44
    measurementtype: Pulse-Doppler-Mode 7 MHz
    measurementfile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT
          noisefile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT
        hydfilename: HydrophonCalibrationData/MW_MH44ReIm.csv
```

6.2 Pre-process measurement data

[5]:
```
# remove DC component
measurement_data = remove_DC_component(measurement_data)
```

[6]:
```
# Calculate measurement uncertainty from noise data
measurement_data = uncertainty_from_noisefile(infos, measurement_data, do_plot=False)
```

```
The file "MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT" was read and it contains 2500␣
↪data points
```

[7]:
```
# calculate spectrum
measurement_data = calculate_spectrum(measurement_data, do_plot = False)
```

[8]:
```
# available measurement data
for key in measurement_data.keys():
    print("%12s: %s"%(key, type(measurement_data[key])))
```

```
        name: <class 'str'>
     voltage: <class 'numpy.ndarray'>
        time: <class 'numpy.ndarray'>
 uncertainty: <class 'numpy.ndarray'>
   frequency: <class 'numpy.ndarray'>
    spectrum: <class 'numpy.ndarray'>
     varspec: <class 'numpy.ndarray'>
```

6.3 Pre-process calibration data

[9]:
```
# reduce frequency range of calibration data
hyd_data = reduce_freq_range(hyd_data, fmin = 1e6, fmax = 100e6)
```

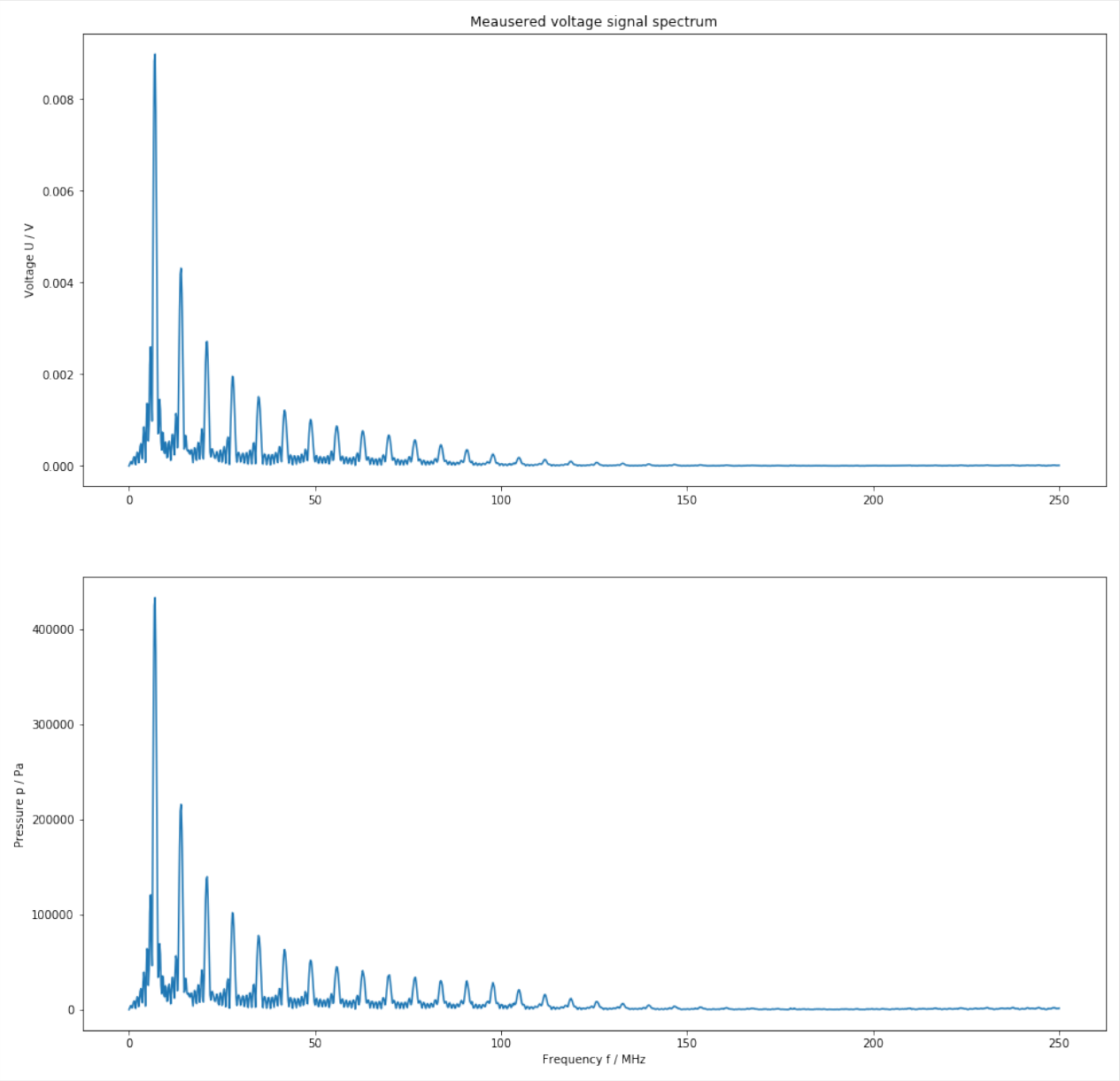
[10]:
```
# align spectrum of hydrophone frequency response with spectrum of measurement
fmeas = measurement_data["frequency"].round()
hyd_interp = interpolate_hyd(hyd_data, fmeas)
```

6.4 Deconvolution in the frequency domain

[11]:
```
# prepare matrix-vector notation for DFT_deconv
H_RI = np.r_[hyd_interp["real"],hyd_interp["imag"]]
U_HRI = np.r_[
    np.c_[np.diag(hyd_interp["varreal"]), hyd_interp["cov"]],
    np.c_[hyd_interp["cov"], np.diag(hyd_interp["varimag"])]]

# application of DFT_deconv
deconv = {"frequency": measurement_data["frequency"]}
deconv["P"], deconv["U_P"] = DFT_deconv(H_RI , measurement_data["spectrum"], U_HRI,␣
↪measurement_data["varspec"])
```

[12]:
```
f = measurement_data["frequency"]
N = len(f)//2
figure(figsize=(16,16))
subplot(2,1,1)
plot(f[:N]/1E6, amplitude(measurement_data["spectrum"]))
title("Meausered voltage signal spectrum")
ylabel("Voltage U / V")

subplot(2,1,2)
plot(f[:N]/1e6, amplitude(deconv["P"]))
xlabel("Frequency f / MHz")
ylabel("Pressure p / Pa");
```

[12]:
```
Text(0, 0.5, 'Pressure p / Pa')
```



6.5 Transformation to the time domain

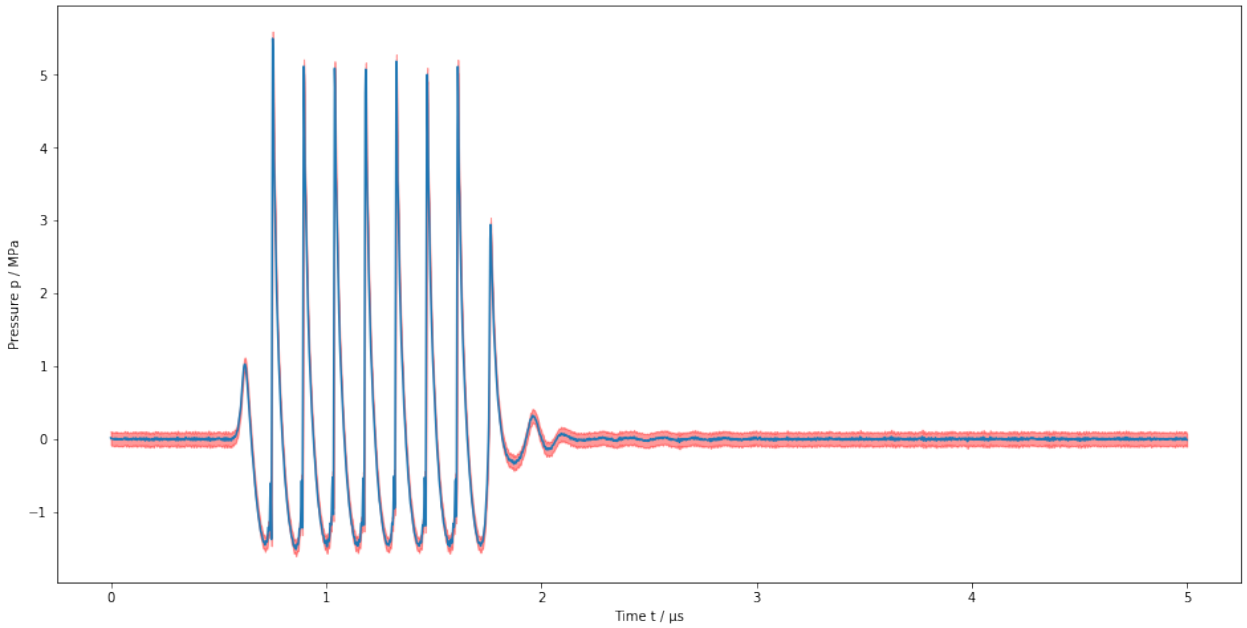
[13]:
```
deconvtime = {"t": measurement_data["time"]}
deconvtime["p"], deconvtime["Up"] = GUM_iDFT(deconv["P"], deconv["U_P"])

# correct for normalisation
deconvtime["p"] = deconvtime["p"]/2*np.size(deconvtime["t"])
deconvtime["Up"] = deconvtime["Up"]/4*np.size(deconvtime["t"])**2
```

[14]:
```
figure(figsize=(16,8))
plot(deconvtime["t"]/1E-6, deconvtime["p"]/1E6)
```

(continues on next page)

(continued from previous page)

```
fill_between(deconvtime["t"]/1E-6,
             deconvtime["p"]/1E6 - 2*np.sqrt(np.diag(deconvtime["Up"]))/1E6,
             deconvtime["p"]/1E6 + 2*np.sqrt(np.diag(deconvtime["Up"]))/1E6,
            color="red", alpha=0.4)
xlabel("Time t / µs")
ylabel("Pressure p / MPa");
```



CHAPTER

SEVEN

REGULARIZED DECONVOLUTION

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[12]:
```
from meas_data_preprocessing import *
from hydrophone_data_preprocessing import *
from PyDynamic.uncertainty.propagate_DFT import DFT_deconv, GUM_iDFT, DFT_multiply
from helper_methods import *
from regularization_bias import *
```

7.1 Pre-process measurement data

[3]:
```
meas_scenario = 13
infos, measurement_data = read_data(meas_scenario = meas_scenario)
_, hyd_data = read_calib_data(meas_scenario = meas_scenario, do_plot = False)

# metadata for chosen measurement scenario
for key in infos.keys():
    print("%20s: %s" %(key,infos[key]))
```

```
The file MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT was read and it contains 2500␣
↪data points.
The time increment is 2e-09 s
                   i: 13
        hydrophonname: GAMPT MH44
     measurementtype: Pulse-Doppler-Mode 7 MHz
     measurementfile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44.DAT
           noisefile: MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT
         hydfilename: HydrophonCalibrationData/MW_MH44ReIm.csv
```

[4]:
```
# remove DC component
measurement_data = remove_DC_component(measurement_data)

# Calculate measurement uncertainty from noise data
measurement_data = uncertainty_from_noisefile(infos, measurement_data, do_plot=False)

# calculate spectrum
measurement_data = calculate_spectrum(measurement_data, do_plot = False)

# available measurement data
```

(continues on next page)

(continued from previous page)

```
for key in measurement_data.keys():
    print("%12s: %s"%(key, type(measurement_data[key])))
```

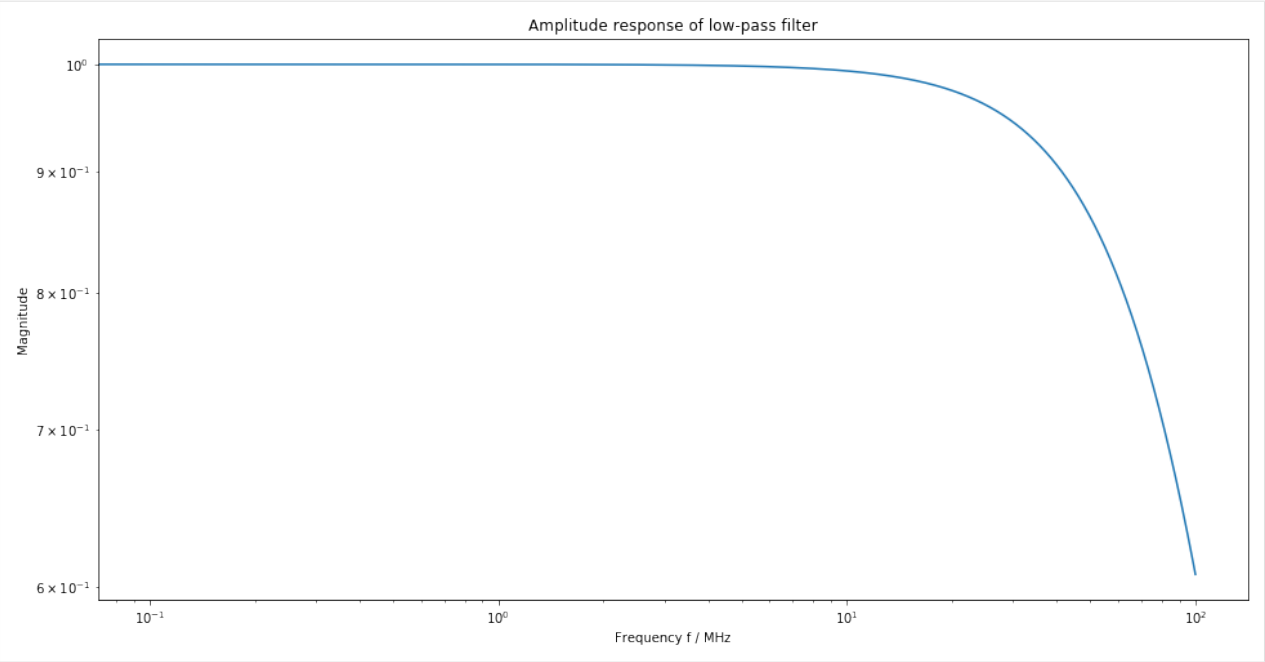
```
The file "MeasuredSignals/pD-Mode 7 MHz/pD7_MH44r.DAT" was read and it contains 2500␣
↪data points
        name: <class 'str'>
     voltage: <class 'numpy.ndarray'>
        time: <class 'numpy.ndarray'>
 uncertainty: <class 'numpy.ndarray'>
   frequency: <class 'numpy.ndarray'>
    spectrum: <class 'numpy.ndarray'>
     varspec: <class 'numpy.ndarray'>
```

7.2 Pre-process calibration data

[5]:
```
# reduce frequency range of calibration data
hyd_data = reduce_freq_range(hyd_data, fmin = 1e6, fmax = 100e6)

# align spectrum of hydrophone frequency response with spectrum of measurement
fmeas = measurement_data["frequency"].round()
hyd_interp = interpolate_hyd(hyd_data, fmeas)
```

7.3 Set low-pass filter to suppress noise

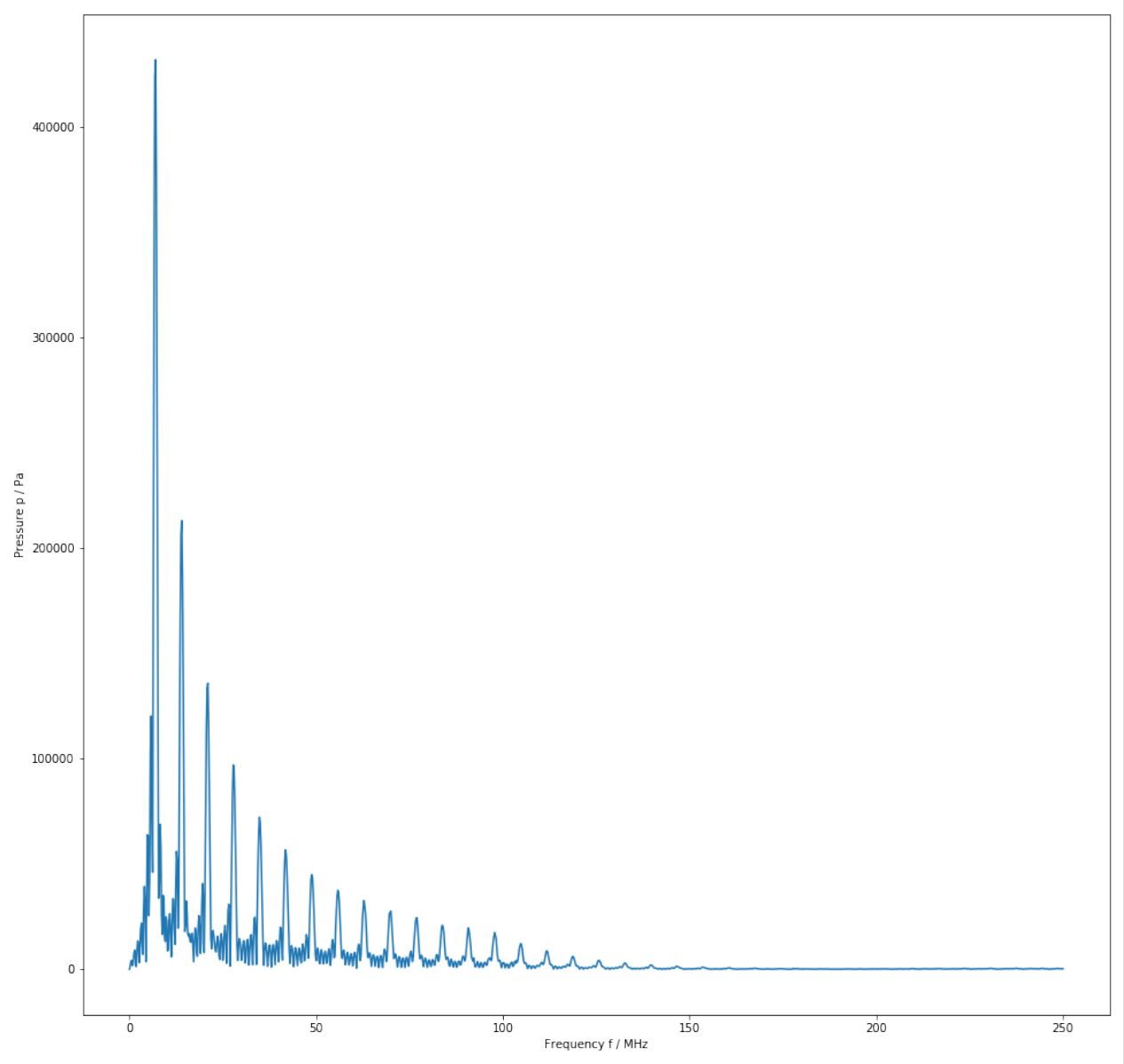
[6]:
```
fc = 80e6 #cut of frequency (Hz) #default is 80 MHz

H_lowpass = lambda f: 1/(1+1j*f/(fc*1.555))**2

fpl = linspace(0, 1e8, 1000)
figure(figsize=(16,8))
loglog(fpl*1e-6, np.abs(H_lowpass(fpl)))
plt.title("Amplitude response of low-pass filter")
plt.xlabel("Frequency f / MHz")
plt.ylabel("Magnitude");
```



7.4 Deconvolution and low-pass filtering

[7]:
```
# prepare matrix-vector notation for DFT_deconv
H_RI = np.r_[hyd_interp["real"],hyd_interp["imag"]]
U_HRI = np.r_[
    np.c_[np.diag(hyd_interp["varreal"]), hyd_interp["cov"]],
    np.c_[hyd_interp["cov"], np.diag(hyd_interp["varimag"])]]

# application of DFT_deconv
deconv = {"frequency": measurement_data["frequency"]}
deconv["P"], deconv["U_P"] = DFT_deconv(H_RI , measurement_data["spectrum"], U_HRI,␣
↪measurement_data["varspec"])

# application of low-pass filter
N = len(deconv["frequency"])//2
H1 = H_lowpass(deconv["frequency"][:N])
Hl_RI = np.r_[np.real(H1),np.imag(H1)]
deconv["P"], deconv["U_P"] = DFT_multiply(deconv["P"], Hl_RI, deconv["U_P"])
```

[8]:
```
f = measurement_data["frequency"]
N = len(f)//2
figure(figsize=(16,16))
plot(f[:N]/1e6, amplitude(deconv["P"]))
xlabel("Frequency f / MHz")
ylabel("Pressure p / Pa");
```



7.5 Estimation of regularization error

[16]:
```
# calculate the average working frequency
f_awf = calc_awf(measurement_data["frequency"], measurement_data["spectrum"])
```

```
searching for f2 in interval [6.8,20] MHz
determined f1: 6.68571 MHz
determined f2: 19.5559 MHz
resulting f_awf = 1.31208e+07
```

[20]:
```
def calculate_freq_points(fH, f, H, X, Fs, candidates, verbose = True):

    def get_candidates(f, S, number=40):
```

(continues on next page)

(continued from previous page)

```
        # Largest local maxima of np.abs(S)
        inds = dsp.argrelmax(np.abs(S))[0]
        inds2 = inds[np.argsort(np.abs(S[inds]))[::-1]]
        return f[inds2[:number]], inds2[:number], np.abs(S[inds2[:number]])

    def get_closest(freqs, f_localmax):
        # Closest local maximum to selected frequency
        cfreqs = freqs.copy()
        for k in range(len(freqs)):
            indf = np.argmin(np.abs(f_localmax - freqs[k]))
            cfreqs[k] = f_localmax[indf]
        return cfreqs

    Ts = 1/Fs
    f = np.fft.rfftfreq(Nf, Ts)

    Xh = X / H

    f_local = get_candidates(fH, np.abs(Xh))[0]
    frequencies = get_closest(candidates, f_local)

    return frequencies
```

[21]:
```
# calculate center frequency candidates as multiples of fawf
multiples = [1, 3, 8]
fvals = [mult*f_awf for mult in multiples]
```

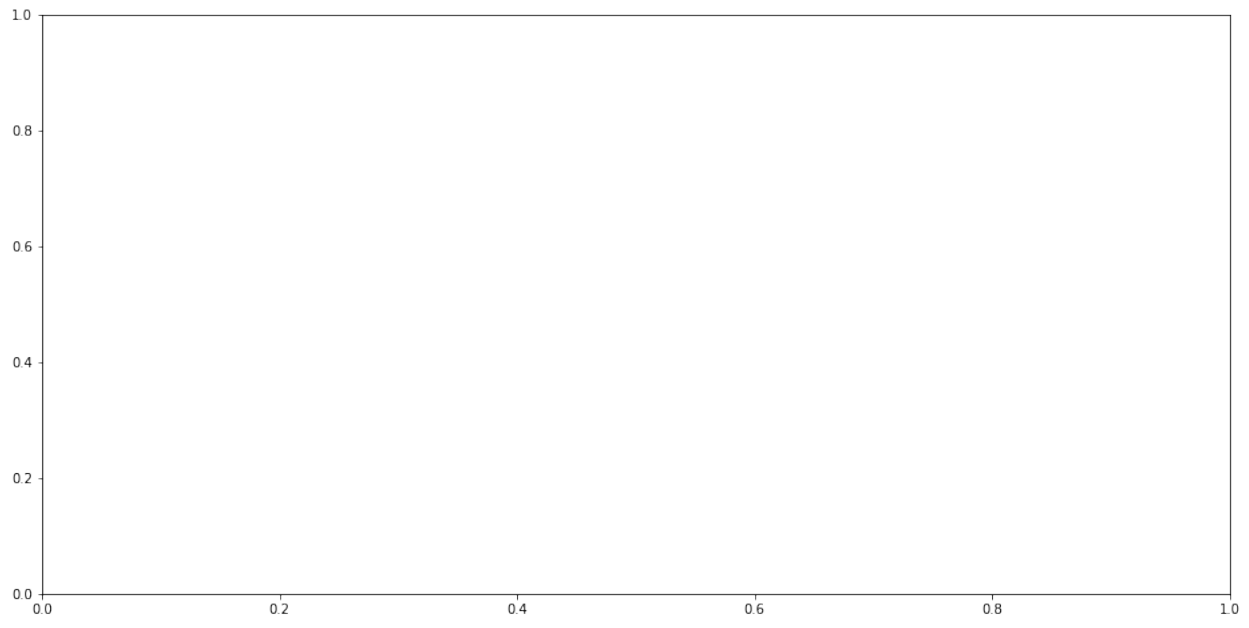
[27]:
```
H = amplitude(np.r_[hyd_interp["real"], hyd_interp["imag"]])
M = amplitude(np.r_[measurement_data["spectrum"]])
Ts= measurement_data["time"][1]-measurement_data["time"][0]
Fs= 1/Ts
N = len(measurement_data["frequency"])//2
center_frequencies = calculate_freq_points(measurement_data["frequency"][:N],
                                           H, N, Fs, fvals)

figure(figsize=(16,8))
plot(f, amplitude(measurement_data["spectrum"]))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-fc6c50d8d73b> in <module>
      8 f = np.fft.rfftfreq(N, Ts)
      9 figure(figsize=(16,8))
---> 10 plot(f, amplitude(measurement_data["spectrum"]))

~/opt/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py in plot(scalex,␣
↪scaley, data, *args, **kwargs)
   2794     return gca().plot(
   2795         *args, scalex=scalex, scaley=scaley, **({"data": data} if data
-> 2796         is not None else {}), **kwargs)
   2797
   2798

~/opt/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_axes.py in plot(self,␣
↪scalex, scaley, data, *args, **kwargs)
   1663         """
```

(continues on next page)

(continued from previous page)

```
   1664         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D._alias_map)
-> 1665         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1666         for line in lines:
   1667             self.add_line(line)

~/opt/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py in __call__(self,
↪ *args, **kwargs)
    223                 this += args[0],
    224                 args = args[1:]
--> 225             yield from self._plot_args(this, kwargs)
    226
    227     def get_next_color(self):

~/opt/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py in _plot_
↪args(self, tup, kwargs)
    389             x, y = index_of(tup[-1])
    390
--> 391         x, y = self._xy_from_xy(x, y)
    392
    393         if self.command == 'plot':

~/opt/anaconda3/lib/python3.7/site-packages/matplotlib/axes/_base.py in _xy_from_
↪xy(self, x, y)
    268         if x.shape[0] != y.shape[0]:
    269             raise ValueError("x and y must have same first dimension, but "
--> 270                              "have shapes {} and {}".format(x.shape, y.shape))
    271         if x.ndim > 2 or y.ndim > 2:
    272             raise ValueError("x and y can be no greater than 2-D, but have "

ValueError: x and y must have same first dimension, but have shapes (626,) and (1251,)
```



[23]:
```
center_frequencies
```

[23]:
```
[74599999.99999999, 74599999.99999999, 99399999.99999999]
```

[24]:
```
fvals
```

[24]:
```
[13120795.50035726, 39362386.50107178, 104966364.00285809]
```

CHAPTER

EIGHT

PYDYNAMIC.UNCERTAINTY.INTERPOLATION

In this notebook we illustrate the use of our method `interp1d_unc <https://pydynamic.readthedocs.io/en/latest/PyDynamic.uncertainty.html#PyDynamic.uncertainty.interpolation.interp1d_unc>`__, which is very much inspired by SciPy's interp1d method and therefore closely aligned with its signature and corresponding capabilities.

The examples proceed according to the following scheme:

- First we setup the appropriate execution and plotting environment.
- Afterwards we download an example data set of real sensor recordings from Zenodo .org, if not already done.
- We visualize the relevant part of the contained data and
- ... prepare a simple interpolation as a first step.

8.1 Setup the environment

[43]:
```
import csv
import json

import holoviews as hv
import numpy as np
import pandas as pd
from download import download
from PyDynamic.uncertainty import interp1d_unc
```

8.2 Download sample data

This step of course is only necessary once, but it checks on execution, if the expected file is present. Thus you can safely execute it without producing unnecessary network traffic or waiting times. We use a sample data set of real measured acceleration data from Zenodo .

[44]:
```
# Set URL and extract filename.
url =␣
↪"https://zenodo.org/record/3786587/files/Met4FOF_mpu9250_Z_Acc_10_hz_250_hz_6rep.dump"
filename = url.split("/")[-1]

path = download(url, filename, replace=False, verbose=True)
```

```
Replace is False and data exists, so doing nothing. Use replace=True to re-download␣
↪the data.
```

8.3 Unpack data and visualize sample set

We use a sample data set provided by the author of the repository github .com/Met4FoF/dataReceiver. The data itself is better described in the according README.md.

[45]:
```
# This is just a residual snippet for testing purposes. If you stumble across this,
# you can just ignore it.
f = open(path)
reader = csv.reader(f, delimiter=";")
fristrow = next(reader)
paramsdictjson = json.loads(fristrow[0])
line = next(reader)  # discard second line with headers
line = next(reader)
f.close()
```

[46]:
```
# Choose one of the data channels from 1 to 10.
i_sensor = 1
# Setup the according indexing string for the pandas dataframe.
pd_data_index = "".join(("Data_", str(i_sensor).zfill(2)))
# Choose the number of measurements extracted from the dataset.
n_rows = 20

# Read the csv-file.
measurements = pd.read_csv(path, sep=";", header=1, skiprows=0, nrows=n_rows)

# Show first five rows of sample data.
measurements.head()
```

[46]:

	id	sample_number	unix_time	unix_time_nsecs	time_uncertainty	Data_01	Data_02	Data_03	Data_04	Data_05	...	Data_07	Data_08	Data_09	Data_10	Data_11	Data_12	Data_13	Data_14	Data_15	Data_16
0	535035904	509730	1583255535	538929677	132	-0.464499	0.421401	-9.294765	-0.004261	0.012783	...	0.0	0.0	5545.864258	23.560877	0.014659	-0.006804	0.002154	0.0	0.0	0.0
1	535035904	509731	1583255535	539928884	132	-0.469287	0.402246	-9.304342	-0.003196	0.013849	...	0.0	0.0	5545.864258	23.551891	0.004898	0.002926	-0.007615	0.0	0.0	0.0
2	535035904	509732	1583255535	540928090	132	-0.498019	0.435767	-9.304342	-0.001065	0.014914	...	0.0	0.0	5545.864258	23.533920	0.009778	-0.001939	-0.002730	0.0	0.0	0.0
3	535035904	509733	1583255535	541927306	132	-0.507597	0.435767	-9.309131	-0.001065	0.013849	...	0.0	0.0	5545.864258	23.515949	0.000017	-0.016533	0.002154	0.0	0.0	0.0
4	535035904	509734	1583255535	542926513	132	-0.493231	0.402246	-9.261244	-0.002131	0.012783	...	0.0	0.0	5545.864258	23.530926	0.000017	-0.011668	0.002154	0.0	0.0	0.0

[5 rows x 21 columns]

[47]:
```
# Extract specified sensor measurements.
data_points_measured = measurements[pd_data_index].to_numpy()
# Extract time including nanoseconds.
t = pd.to_datetime(
    measurements["unix_time"] * 1e9 + measurements["unix_time_nsecs"]
).to_numpy(dtype=np.datetime64)
# Drop dataset to free memory.
del measurements
```

8.4 Setup plotting environment and labels

[48]:
```
hv.extension("matplotlib")

timestamp_labels = hv.Dimension(("time", "time of measurement"), unit="$s$")
measurement_labels = hv.Dimension(
    ("acceleration", paramsdictjson[str(i_sensor)]["PHYSICAL_QUANTITY"]),␣
↪unit="$m/s^2$"
)
```

Data type cannot be displayed: application/javascript, application/vnd.holoviews_load.v0+json

Data type cannot be displayed: application/vnd.holoviews_load.v0+json, application/javascript

8.5 Visualize the original data points

[49]:
```
curve_original = hv.Curve(
    (t, data_points_measured),
    timestamp_labels,
    measurement_labels,
    label="Original values",
)
curve_original
```

```
/home/ludwig10/code/envs/PyDynamic_tutorials-3.8/lib/python3.8/site-packages/numpy/
↪lib/type_check.py:277: DeprecationWarning: elementwise comparison failed; this will␣
↪raise an error in the future.
  return imag(x) == 0
```

[49]:
```
:Curve   [time]   (acceleration)
```

8.6 Setup interpolation timestamps and interpolate (with uncertainties set to 0)

[50]:
```
# Choose interval length for the interpolation in nanoseconds.
dt = 2000000

# Setup new vector of timestamps.
t_new = np.arange(np.min(t), np.max(t), dt)

# Since np.arange in overflow situations results in the last value not guaranteed to
# be smaller than t's maximum', we need to check for this and delete this
# unexpected value.
if t_new[-1] > np.max(t):
    t_new = t_new[:-1]

# Conduct actual interpolation.
data_points_interp = interp1d_unc(
    t_new=np.float64(t_new),
    t=np.float64(t),
    y=data_points_measured,
    uy=np.zeros_like(data_points_measured),
)[1]
```

8.7 Visualize the results

[51]:
```
curve_interp = hv.Curve(
    (t_new, data_points_interp),
    timestamp_labels,
    measurement_labels,
    label="Interpolated values",
)
curve_original + curve_interp
```

```
/home/ludwig10/code/envs/PyDynamic_tutorials-3.8/lib/python3.8/site-packages/numpy/
↪lib/type_check.py:277: DeprecationWarning: elementwise comparison failed; this will␣
↪raise an error in the future.
  return imag(x) == 0
/home/ludwig10/code/envs/PyDynamic_tutorials-3.8/lib/python3.8/site-packages/numpy/
↪lib/type_check.py:277: DeprecationWarning: elementwise comparison failed; this will␣
↪raise an error in the future.
  return imag(x) == 0
```

[51]:
```
:Layout
   .Curve.Original_values     :Curve   [time]   (acceleration)
   .Curve.Interpolated_values :Curve   [time]   (acceleration)
```

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search