[17]:
```python
import math
import struct

import numpy as np
from scipy import signal, optimize
from matplotlib import pyplot as plt

import rocof_test_data
```

[18]:
```python
import matplotlib
from IPython.display import set_matplotlib_formats
#%matplotlib widget
%matplotlib inline
set_matplotlib_formats('png', 'pdf')
font = {'family' : 'normal',
        'weight' : 'normal',
        'size'   : 10}
matplotlib.rc('font', **font)
```

[19]:
```python
fs = 1000 # Hz
ff = 50 # Hz
duration = 60 # seconds
# test_data = rocof_test_data.sample_waveform(rocof_test_data.
 ↪test_close_interharmonics_and_flicker(),
#                                             duration=20,
#                                             sampling_rate=fs,
#                                             frequency=ff)[0]
# test_data = rocof_test_data.sample_waveform(rocof_test_data.
 ↪gen_noise(fmin=10, amplitude=1),
#                                             duration=20,
#                                             sampling_rate=fs,
#                                             frequency=ff)[0]

test_data = []
test_labels = [ fun.__name__.replace('test_', '') for fun in rocof_test_data.
 ↪all_tests ]
for gen in rocof_test_data.all_tests:
    test_data.append(rocof_test_data.sample_waveform(gen(),
                                            duration=duration,
                                            sampling_rate=fs,
                                            frequency=ff)[0])
# d = 10 # seconds
# test_data = np.sin(2*np.pi * ff * np.linspace(0, d, int(d*fs)))
```

[20]:
```
spr_fmt = f'{fs}Hz' if fs<1000 else f'{fs/1e3:f}'.rstrip('.0') + 'kHz'
for label, data in zip(test_labels, test_data):
    with open(f'rocof_test_data/rocof_test_{label}_{spr_fmt}.bin', 'wb') as f:
        for sample in data:
            f.write(struct.pack('<f', sample))
```

[21]:
```
analysis_periods = 10
window_len = 256 # fs * analysis_periods/ff
nfft_factor = 1
sigma = window_len/8 # samples
quantization_bits = 14

ffts = []
for item in test_data:
    f, t, Zxx = signal.stft((item * (2**(quantization_bits-1) - 1)).round().
↪astype(np.int16).astype(float),
                fs = fs,
                window=('gaussian', sigma),
                nperseg = window_len,
                nfft = window_len * nfft_factor)
                #boundary = 'zeros')
    ffts.append((f, t, Zxx))
```

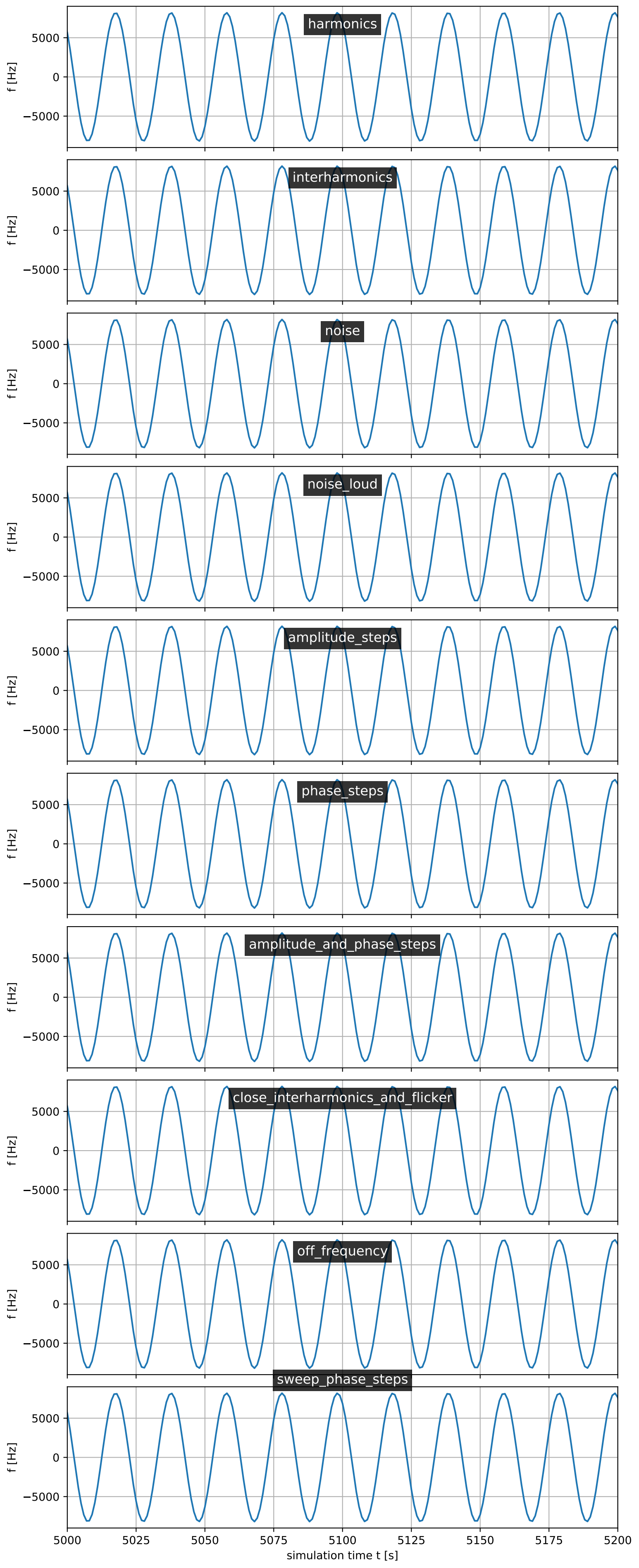
[22]:
```
Zxx.shape
```

[22]:
```
(129, 470)
```

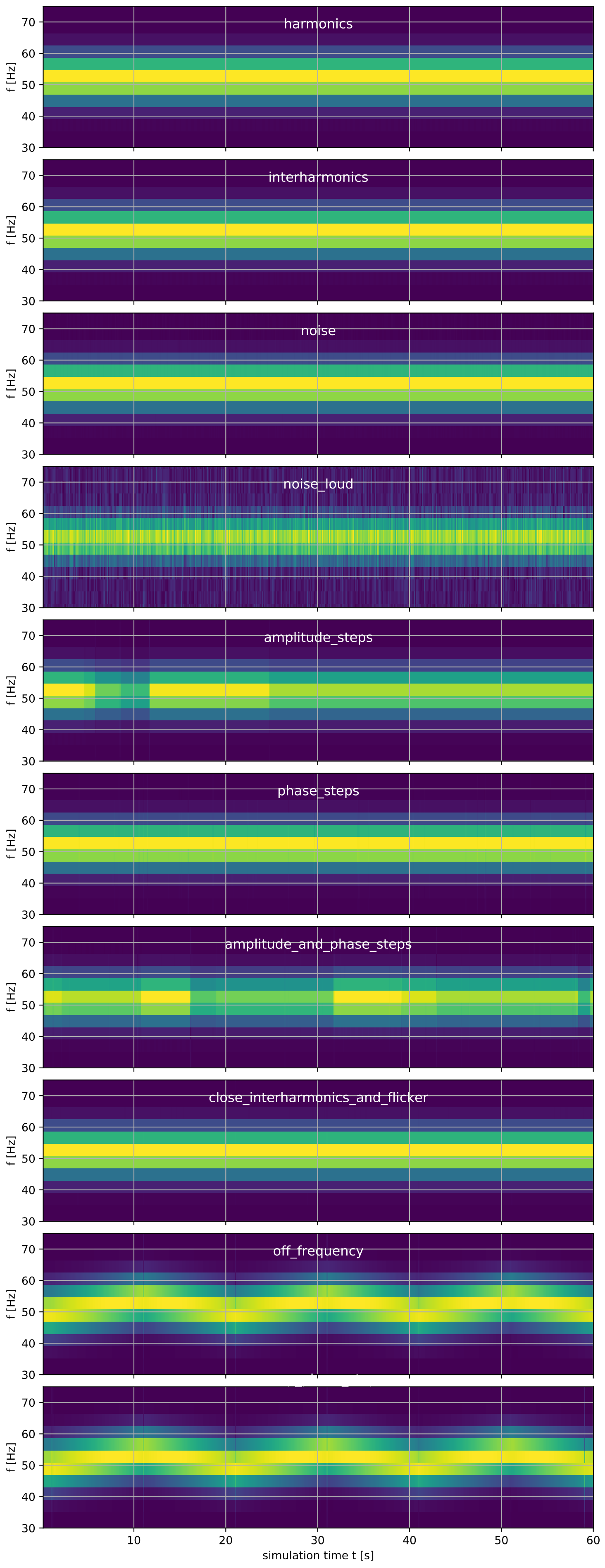
[23]:
```
1000/256
```

[23]:
```
3.90625
```

[24]:
```
fig, ax = plt.subplots(len(test_data), figsize=(8, 20), sharex=True)
fig.tight_layout(pad=2, h_pad=0.1)

for fft, ax, label in zip(test_data, ax.flatten(), test_labels):
    ax.plot((item * (2**(quantization_bits-1) - 1)).round())

    ax.set_title(label, pad=-20, color='white', bbox=dict(boxstyle="square",␣
↪ec=(0,0,0,0), fc=(0,0,0,0.8)))
    ax.grid()
    ax.set_ylabel('f [Hz]')
ax.set_xlabel('simulation time t [s]')
ax.set_xlim([5000, 5200])
None
```


harmonics
interharmonics
noise
noise_loud
amplitude_steps
phase_steps
amplitude_and_phase_steps
close_interharmonics_and_flicker
off_frequency
sweep_phase_steps
f [Hz]
5000
0
−5000
5000
5025
5050
5075
5100
5125
5150
5175
5200
simulation time t [s]

[25]:
```
fig, ax = plt.subplots(len(test_data), figsize=(8, 20), sharex=True)
fig.tight_layout(pad=2, h_pad=0.1)

for fft, ax, label in zip(ffts, ax.flatten(), test_labels):
    f, t, Zxx = fft
    ax.pcolormesh(t[1:], f[:250], np.abs(Zxx[:250,1:]))
    ax.set_title(label, pad=-20, color='white')
    ax.grid()
    ax.set_ylabel('f [Hz]')
    ax.set_ylim([30, 75]) # Hz
ax.set_xlabel('simulation time t [s]')
None
```

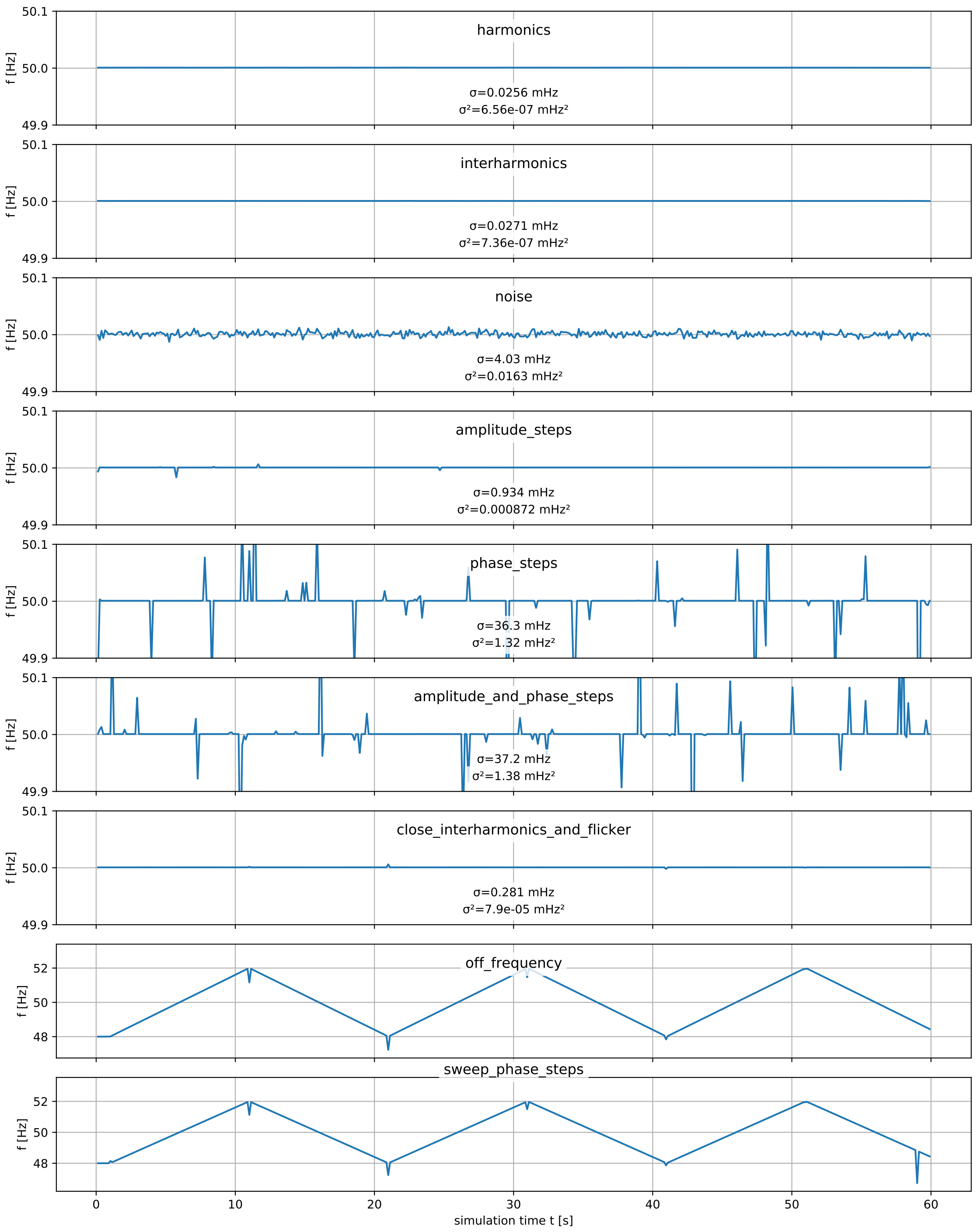

harmonics
interharmonics
noise
noise_loud
amplitude_steps
phase_steps
amplitude_and_phase_steps
close_interharmonics_and_flicker
off_frequency
f [Hz]
30
40
50
60
70
10
20
30
40
50
60
simulation time t [s]

```
[26]: f
```

```
[26]: array([  0.       ,   3.90625,   7.8125 ,  11.71875,  15.625  ,  19.53125,
              23.4375 ,  27.34375,  31.25   ,  35.15625,  39.0625 ,  42.96875,
              46.875  ,  50.78125,  54.6875 ,  58.59375,  62.5    ,  66.40625,
              70.3125 ,  74.21875,  78.125  ,  82.03125,  85.9375 ,  89.84375,
              93.75   ,  97.65625, 101.5625 , 105.46875, 109.375  , 113.28125,
             117.1875 , 121.09375, 125.     , 128.90625, 132.8125 , 136.71875,
             140.625  , 144.53125, 148.4375 , 152.34375, 156.25   , 160.15625,
             164.0625 , 167.96875, 171.875  , 175.78125, 179.6875 , 183.59375,
             187.5    , 191.40625, 195.3125 , 199.21875, 203.125  , 207.03125,
             210.9375 , 214.84375, 218.75   , 222.65625, 226.5625 , 230.46875,
             234.375  , 238.28125, 242.1875 , 246.09375, 250.     , 253.90625,
             257.8125 , 261.71875, 265.625  , 269.53125, 273.4375 , 277.34375,
             281.25   , 285.15625, 289.0625 , 292.96875, 296.875  , 300.78125,
             304.6875 , 308.59375, 312.5    , 316.40625, 320.3125 , 324.21875,
             328.125  , 332.03125, 335.9375 , 339.84375, 343.75   , 347.65625,
             351.5625 , 355.46875, 359.375  , 363.28125, 367.1875 , 371.09375,
             375.     , 378.90625, 382.8125 , 386.71875, 390.625  , 394.53125,
             398.4375 , 402.34375, 406.25   , 410.15625, 414.0625 , 417.96875,
             421.875  , 425.78125, 429.6875 , 433.59375, 437.5    , 441.40625,
             445.3125 , 449.21875, 453.125  , 457.03125, 460.9375 , 464.84375,
             468.75   , 472.65625, 476.5625 , 480.46875, 484.375  , 488.28125,
             492.1875 , 496.09375, 500.     ])
```

```
[35]: fig, axs = plt.subplots(len(test_data)-1, figsize=(12, 15), sharex=True)
      axs = axs.flatten()

      for fft, label in zip(ffts, test_labels):
          if label in ['noise_loud']: # custom test case, not part of upstream suite
              continue
          ax, *axs = axs

          f, f_t, Zxx = fft

          n_f, n_t = Zxx.shape
          f_min, f_max = 30, 70 # Hz
          bounds_f = slice(np.argmax(f > f_min), np.argmin(f < f_max))

          f_mean = np.zeros(Zxx.shape[1])
          for t in range(1, Zxx.shape[1] - 1):
              frame_f = f[bounds_f]
              frame_step = frame_f[1] - frame_f[0]
              time_step = f_t[1] - f_t[0]
              frame_Z = np.abs(Zxx[bounds_f, t])
```

```
        def gauss(x, *p):
            A, mu, sigma = p
            return A*np.exp(-(x-mu)**2/(2.*sigma**2))

        f_start = frame_f[np.argmax(frame_Z)]
        A_start = np.max(frame_Z)
        p0 = [A_start, f_start, 1.]
        try:
            coeff, var = optimize.curve_fit(gauss, frame_f, frame_Z, p0=p0)
            A, mu, sigma, *_ = coeff
            f_mean[t] = mu
        except RuntimeError:
            f_mean[t] = np.nan
    ax.plot(f_t[1:-1], f_mean[1:-1])

    ax.set_title(label, pad=-20, bbox=dict(fc='white', alpha=0.8, ec='none'))
    ax.set_ylabel('f [Hz]')
    ax.grid()
    if not label in ['off_frequency', 'sweep_phase_steps']:
        ax.set_ylim([49.90, 50.10])
        var = np.var(f_mean[1:-1])
        ax.text(0.5, 0.1, f' ²={var * 1e3:.3g} mHz²', transform=ax.transAxes,␣
 ↪ha='center', bbox=dict(fc='white', alpha=0.8, ec='none'))
        ax.text(0.5, 0.25, f' ={np.sqrt(var) * 1e3:.3g} mHz', transform=ax.
 ↪transAxes, ha='center', bbox=dict(fc='white', alpha=0.8, ec='none'))
    else:
        f_min, f_max = min(f_mean[1:-1]), max(f_mean[1:-1])
        delta = f_max - f_min
        ax.set_ylim(f_min - delta * 0.1, f_max + delta * 0.3)

ax.set_xlabel('simulation time t [s]')
fig.tight_layout(pad=2.2, h_pad=0, w_pad=1)
fig.savefig('fig_out/freq_meas_rocof_reference.pdf')
None
```


harmonics
σ=0.0256 mHz
σ²=6.56e-07 mHz²
interharmonics
σ=0.0271 mHz
σ²=7.36e-07 mHz²
noise
σ=4.03 mHz
σ²=0.0163 mHz²
amplitude_steps
σ=0.934 mHz
σ²=0.000872 mHz²
phase_steps
σ=36.3 mHz
σ²=1.32 mHz²
amplitude_and_phase_steps
σ=37.2 mHz
σ²=1.38 mHz²
close_interharmonics_and_flicker
σ=0.281 mHz
σ²=7.9e-05 mHz²
off_frequency
sweep_phase_steps
f [Hz]
50.1
50.0
49.9
52
50
48
0
10
20
30
40
50
60
simulation time t [s]