daily

August 19, 2025

Dark-Fiber Charakterisierung für entanglement polarisierter QKD

Aufbereitung und Analyse der Messdaten nach Tageszyklen

Autor: Laura Komma
Datensatz: Erfurt - Sundhausen

1 Aufbereitung der Messdaten

1.1 Ressourcen und Literatur

- Messdaten
- Messgerät
- Plotting-Tool "py_pol"

1.2 Verwendete Libraries / Softwaretools

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import numpy as np
import io
import datetime
from IPython.display import Image

from scipy.stats import norm
from scipy.signal import butter, filtfilt

from py_pol.jones_vector import Jones_vector, degrees
from py_pol.stokes import Stokes
```

1.3 Eigene Hilfsfunktionen

[3]:
```
def format_time(x, pos):
    h = int(x) // 3600
    m = (int(x) % 3600) // 60
    return f"{h:02d}:{m:02d}"
```

[4]:
```
def lowpass(data, box = 100):
    box = box
    LP_filter = np.full(int(box), 1/box)
    lps = np.convolve(ydata, LP_filter)
    lps = lps[ int((box-1) / 2) :len(lps) - int((box - 1) / 2)]
    return lps

def highpass(data, box = 100):
    lps = lowpass(data, box)
    hps = data - lps[:-1]
    return hps
```

[6]:
```
def butter_filter(data, cutoff, fs=4, order=4, btype='low'):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype=btype, analog=False)
    y = filtfilt(b, a, data)
    return y
```

[7]:
```
def convert_angel(az, el):
    az = np.array(az) * degrees
    el = np.array(el) * degrees

    x = np.cos(2*el) * np.cos(2*az)
    y = np.cos(2*el) * np.sin(2*az)
    z = np.sin(2*el)
    return x, y, z
```

[8]:
```
def calculate_freq(x, y, z):
    bins = 50
    coords = np.vstack((x, y, z)).T
    hist, edges = np.histogramdd(coords, bins=bins)

    # Indices for each coordinate pair (with clipping)
    bin_indices = [
        np.clip(np.digitize(x, edges[0]) - 1, 0, bins - 1),
        np.clip(np.digitize(y, edges[1]) - 1, 0, bins - 1),
        np.clip(np.digitize(z, edges[2]) - 1, 0, bins - 1)
    ]
    freq = hist[bin_indices[0], bin_indices[1], bin_indices[2]]
```

```
    return freq
```

[9]:
```
def plot_poincare(x, y, z, freq, elev=15, azim=45, title="Not given"):
    fig = plt.figure(figsize=(6, 6))
    ax = fig.add_subplot(111, projection='3d')

    # Poincaré-Sphere
    u = np.linspace(0, 2 * np.pi, 50)
    v = np.linspace(0, np.pi, 50)
    X = np.outer(np.cos(u), np.sin(v))
    Y = np.outer(np.sin(u), np.sin(v))
    Z = np.outer(np.ones(np.size(u)), np.cos(v))

    ax.plot_surface(X, Y, Z, color='lightgray', alpha=0.1, edgecolor='k',␣
 ↪linewidth=0.5)

    # Axes
    ax.plot([0, 1.6], [0, 0], [0, 0], color='red')     # +S1
    ax.plot([0, -1.6], [0, 0], [0, 0], color='red')    # -S1
    ax.plot([0, 0], [0, 1.6], [0, 0], color='green')   # +S2
    ax.plot([0, 0], [0, -1.6], [0, 0], color='green')  # -S2
    ax.plot([0, 0], [0, 0], [0, 1.4], color='blue')    # +S3
    ax.plot([0, 0], [0, 0], [0, -1.4], color='blue')   # -S3

    # Labeling Axes
    ax.text( 2.0, 0, 0,  'S1',  color='red', fontsize=10)
    ax.text(-1.8, 0, 0, '-S1',  color='red', fontsize=10)
    ax.text( 0, 1.8, 0,  'S2',  color='green', fontsize=10)
    ax.text( 0, -2.0, 0, '-S2',  color='green', fontsize=10)
    ax.text( 0, 0, 1.6, 'S3',   color='blue', fontsize=10)
    ax.text( 0, 0, -1.6, '-S3',  color='blue', fontsize=10)

    # Measurement
    sc = ax.scatter(x, y, z, c=freq, cmap='hot', s=15)
    plt.colorbar(sc, label='density')

    # Scaling
    ax.set_box_aspect([1, 1, 1])
    ax.set_xticks([]); ax.set_yticks([]); ax.set_zticks([])

    # Aligning Poincaré-Sphere
    ax.view_init(elev=elev, azim=azim)

    ax.set_title(title)

    plt.show()
```

1.4 Datenaufbereitungsschritte

[10]:
```
filename ='20_02_to_26_02_Sundhausen to FZE port 2.csv'
skip = 8
sep = ";"
```

[18]:
```
columns = ['Time[date hh:mm:ss] ', ' Elapsed Time [hh:mm:ss:ms]', ' Normalized␣
↪s 1 ', ' Normalized s 2 ', ' Normalized s 3 ', ' S 0 [mW]', ' S 1 [mW]', ' S␣
↪2 [mW]', ' S 3 [mW]', ' Azimuth[°] ', ' Ellipticity[°] ', ' DOP[%] ', '␣
↪DOCP[%] ', ' DOLP[%] ', ' Power[mW] ', ' Pol Power[mW] ', ' Unpol Power[mW]␣
↪', ' Power[dBm] ', ' Pol Power[dBm] ', ' Unpol Power[dBm] ', '␣
↪Power-Split-Ratio ', ' Phase Difference[°] ', ' Warning']
for c in range(len(columns)):
    print(c, ': ', columns[c])

colors = ["tab:blue", "tab:orange", "tab:green", "tab:red", "tab:purple", "tab:
↪brown", "tab:pink", "tab:gray"]
```

```
0 :  Time[date hh:mm:ss]
1 :   Elapsed Time [hh:mm:ss:ms]
2 :   Normalized s 1
3 :   Normalized s 2
4 :   Normalized s 3
5 :   S 0 [mW]
6 :   S 1 [mW]
7 :   S 2 [mW]
8 :   S 3 [mW]
9 :   Azimuth[°]
10 :   Ellipticity[°]
11 :   DOP[%]
12 :   DOCP[%]
13 :   DOLP[%]
14 :   Power[mW]
15 :   Pol Power[mW]
16 :   Unpol Power[mW]
17 :   Power[dBm]
18 :   Pol Power[dBm]
19 :   Unpol Power[dBm]
20 :   Power-Split-Ratio
21 :   Phase Difference[°]
22 :   Warning
```

2 Polarisationsgrad (DOP [%])

[12]:
```
degree = pd.read_csv(filename,skiprows=skip, sep=sep, usecols=[columns[0],␣
↪columns[11]])
```

[13]:
```
degree[columns[0]] = pd.to_datetime(degree[columns[0]])
degree.set_index(columns[0], inplace=True)
```

2.1 Tägliche Aufteilung der Messdaten

[14]:
```
degree_daily = degree.groupby(degree.index.date)
degree_daily_list = list(degree_daily)
```

[15]:
```
for date, df_day in degree_daily_list:
    print(date, '\n')
    print(df_day.describe())
    print('')
```

```
2025-02-20

             DOP[%]
count  148479.000000
mean      100.027175
std         5.031556
min        72.420000
25%        97.480000
50%        99.770000
75%       102.400000
max       141.730000

2025-02-21

             DOP[%]
count  338320.000000
mean       99.089434
std         5.499147
min        69.580000
25%        96.310000
50%        98.760000
75%       101.600000
max       136.480000

2025-02-22

             DOP[%]
count  338429.000000
mean      100.040453
std         6.156741
min        66.290000
25%        96.840000
50%        99.810000
75%       103.230000
max       138.270000
```

2025-02-23

```
              DOP[%]
count  338655.000000
mean       100.177322
std          5.167570
min         70.560000
25%         97.500000
50%         99.760000
75%        102.520000
max        132.930000
```

2025-02-24

```
              DOP[%]
count  338847.000000
mean       100.474675
std          7.045158
min         58.560000
25%         97.210000
50%         99.900000
75%        103.300000
max        173.700000
```

2025-02-25

```
              DOP[%]
count  338305.000000
mean        99.809349
std          9.251338
min         59.380000
25%         94.600000
50%         99.050000
75%        104.200000
max        171.450000
```

2025-02-26

```
              DOP[%]
count  338653.000000
mean       100.865323
std          4.981738
min         74.090000
25%         98.440000
50%        100.560000
75%        103.420000
max        137.890000
```

```
2025-02-27

              DOP[%]
count  189966.000000
mean       100.276237
std          5.353409
min         74.220000
25%         96.800000
50%        100.460000
75%        103.370000
max        133.630000
```

[19]:
```python
plt.figure(figsize = (16,9))
for date, df_day in degree_daily_list:
    values = df_day
    plt.plot(values, label=str(date))
plt.grid()
plt.legend(loc = 'best')
plt.title(f'Course of {columns[11]}\n during full measurement')
plt.xlabel('time')
plt.ylabel('percentage')
plt.show()
```

[20]:
```
fig, axs = plt.subplots(len(degree_daily_list), 1, figsize=(16, 16),␣
↪sharex=False)

for i in range(len(degree_daily_list)):
    ax = axs[i] if len(degree_daily_list) > 1 else axs
    df_day = degree_daily_list[i][1]

    ax.plot(df_day.index, df_day, label=columns[11], color=colors[i])
    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'{columns[11]} on {degree_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('percentage')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

[21]:
```
fig, ax = plt.subplots(figsize = (16,9))

for date, df_day in degree_daily_list:
    times = (df_day.index - df_day.index.normalize()).total_seconds()
    values = df_day
    plt.plot(times, values, label=str(date))


ax.xaxis.set_major_locator(ticker.MultipleLocator(3600 * 2))
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_time))
ax.set_xlim(0, 24 * 3600)

plt.grid()
```

```
plt.legend(loc = 'best')
plt.title(f'Daily course of {columns[11]}')
plt.xlabel('time')
plt.ylabel('percentage')
plt.show()
```

2.2 Stundenweise Aufteilung (Beispiel: zweiter Tag)

[30]:
```
# exemplary day
day = 1 #second day
```

[31]:
```
degree_one_day = degree_daily_list[day][1]
hourly = degree_one_day.groupby(degree_one_day.index.hour)
degree_hourly = list(hourly)
```

[32]:
```
for hour, df_hour in degree_hourly:
    print(f'{hour}:00\n')
    print(df_hour.describe())
    print('')
```

```
0:00

            DOP[%]
count  14084.000000
mean     103.795743
std        2.797892
```

```
min        94.740000
25%       101.860000
50%       103.830000
75%       105.810000
max       114.490000

1:00

            DOP[%]
count  14086.00000
mean      98.75668
std        2.86260
min       86.98000
25%       97.16000
50%       99.13000
75%      100.27000
max      113.86000

2:00

             DOP[%]
count  14084.000000
mean      96.927555
std        4.610012
min       79.010000
25%       93.640000
50%       96.780000
75%      100.030000
max      116.620000

3:00

             DOP[%]
count  14086.000000
mean      97.751751
std        3.811546
min       82.810000
25%       95.330000
50%       98.030000
75%      100.320000
max      111.660000

4:00

             DOP[%]
count  14086.000000
mean      97.392722
std        1.681316
```

```
min        88.100000
25%        96.682500
50%        97.340000
75%        98.010000
max       106.230000
```

5:00

```
            DOP[%]
count  14087.000000
mean      96.566923
std        2.133682
min       89.030000
25%       95.370000
50%       96.980000
75%       98.000000
max      103.750000
```

6:00

```
            DOP[%]
count  14104.000000
mean      97.518590
std        1.587344
min       90.900000
25%       96.630000
50%       97.740000
75%       98.410000
max      102.810000
```

7:00

```
            DOP[%]
count  14084.000000
mean      98.426234
std        4.452399
min       76.740000
25%       96.720000
50%       99.300000
75%      100.600000
max      120.560000
```

8:00

```
            DOP[%]
count  14095.000000
mean      99.689960
std        4.768432
```

```
min        79.690000
25%        97.670000
50%        99.380000
75%       101.090000
max       126.790000

9:00

             DOP[%]
count  14092.000000
mean      99.361822
std        4.830754
min       73.950000
25%       96.350000
50%       99.165000
75%      102.212500
max      123.180000

10:00

             DOP[%]
count  14092.000000
mean      99.019258
std        4.817737
min       80.800000
25%       96.670000
50%       99.140000
75%      101.310000
max      122.400000

11:00

             DOP[%]
count  14084.000000
mean      98.880787
std        5.318721
min       82.420000
25%       95.920000
50%       98.620000
75%      101.380000
max      121.390000

12:00

             DOP[%]
count  14093.000000
mean      97.368826
std        4.629314
```

```
min        80.000000
25%        94.860000
50%        97.270000
75%        99.610000
max       119.430000

13:00

             DOP[%]
count  14090.000000
mean      99.356842
std        3.765109
min       84.040000
25%       97.270000
50%       99.010000
75%      101.250000
max      118.230000

14:00

             DOP[%]
count  14087.000000
mean      99.872612
std        5.064116
min       80.840000
25%       96.975000
50%       99.890000
75%      102.850000
max      121.530000

15:00

             DOP[%]
count  14086.000000
mean     100.050959
std        5.504417
min       78.310000
25%       96.560000
50%      100.430000
75%      103.520000
max      120.970000

16:00

             DOP[%]
count  14112.000000
mean     100.712666
std        6.420720
```

```
min        74.700000
25%        96.640000
50%       100.230000
75%       104.772500
max       132.170000

17:00

             DOP[%]
count  14121.000000
mean     100.626842
std        4.375413
min       80.260000
25%       98.230000
50%      100.590000
75%      103.360000
max      119.320000

18:00

             DOP[%]
count  14175.000000
mean      98.739725
std        6.259073
min       74.380000
25%       95.170000
50%       98.730000
75%      101.640000
max      130.140000

19:00

             DOP[%]
count  14086.000000
mean     101.390371
std       10.464255
min       69.580000
25%       93.910000
50%      102.170000
75%      108.890000
max      136.480000

20:00

             DOP[%]
count  14120.000000
mean     101.194218
std        7.852199
```

```
min        75.030000
25%        97.160000
50%       100.060000
75%       105.940000
max       133.490000

21:00

             DOP[%]
count  14085.000000
mean      98.145146
std        9.601264
min       72.990000
25%       90.550000
50%       97.360000
75%      105.160000
max      130.910000

22:00

             DOP[%]
count  14086.000000
mean      97.547782
std        4.816707
min       79.500000
25%       95.032500
50%       97.445000
75%       99.240000
max      122.410000

23:00

             DOP[%]
count  14115.000000
mean      99.045413
std        2.235870
min       89.180000
25%       98.130000
50%       99.000000
75%       99.850000
max      109.160000
```

[33]:
```
daily_quaters = [degree_hourly[:6], degree_hourly[6:12], degree_hourly[12:18],␣
 ↪degree_hourly[18:]]
for quater in daily_quaters:
    fig, axs = plt.subplots(len(quater), 1, figsize=(16, 16), sharex=False)
```

```
    for i in range(len(quater)):
        ax = axs[i] if len(quater) > 1 else axs
        df_day = quater[i][1]

        ax.plot(df_day.index, df_day, label=columns[11], color=colors[day],␣
↪alpha=0.5)
        ax.grid()
        ax.legend(loc='best')
        ax.set_title(f'{columns[11]} on {degree_daily_list[day][0]} at␣
↪{quater[i][0]}:00')
        ax.set_xlabel('time')
        ax.set_ylabel('percentage')

        ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

    plt.tight_layout()
    plt.show()
```

DOP[%] on 2025-02-21 at 0:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 1:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 2:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 3:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 4:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 5:00
DOP[%]
percentage
time

DOP[%] on 2025-02-21 at 6:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 7:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 8:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 9:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 10:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 11:00
DOP[%]
percentage
time

DOP[%] on 2025-02-21 at 12:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 13:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 14:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 15:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 16:00
DOP[%]
percentage
time
DOP[%] on 2025-02-21 at 17:00
DOP[%]
percentage
time

2.3 Ausreißerbereinigung

[34]:
```
new_degree = degree[columns[11]]
new_degree = new_degree.drop(new_degree[(new_degree > 140) | (new_degree < 0)].
 ↪index)
new_degree.describe()
```

[34]:
```
count    2.368581e+06
mean     1.000677e+02
std      6.286610e+00
min      5.856000e+01
25%      9.693000e+01
```

```
50%      9.973000e+01
75%      1.030000e+02
max      1.400000e+02
Name:  DOP[%] , dtype: float64
```

[35]:
```
new_degree_daily = new_degree.groupby(new_degree.index.date)
new_degree_daily_list = list(new_degree_daily)
```

[36]:
```
plt.figure(figsize = (16,9))
for date, df_day in new_degree_daily_list:
    values = df_day
    plt.plot(values, label=str(date), alpha = 0.5)
plt.grid()
plt.legend(loc = 'best')
plt.title('Daily course of DOP [%]\nwithout peaks')
plt.xlabel('time')
plt.ylabel('percentage')
plt.show()
```

[37]:
```
fig, ax = plt.subplots(figsize = (16,9))

for date, df_day in new_degree_daily_list:
    times = (df_day.index - df_day.index.normalize()).total_seconds()
    values = df_day
    plt.plot(times, values, label=str(date) , alpha=0.5)
```

```
ax.xaxis.set_major_locator(ticker.MultipleLocator(3600 * 2))
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_time))
ax.set_xlim(0, 24 * 3600)

plt.grid()
plt.legend(loc = 'best')
plt.title('Daily course of DOP [%]\nwithout peaks')
plt.xlabel('time')
plt.ylabel('percentage')
plt.show()
```

2.4 Tägliche Normalverteilung mit Histogramm

[38]:
```
ydata1 = np.array(new_degree_daily_list[day][1])
mu1 = np.mean(ydata1)
sigma1 = np.std(ydata1)
x1 = np.linspace(ydata1.min(),ydata1.max(),len(ydata1))
y1 = norm.pdf(x1, mu1, sigma1)

plt.plot(x1, y1, label = 'Normal Distribution', color=colors[day])
plt.hist(ydata1, bins = 50, density = True, color=colors[day], alpha = 0.3,␣
 ↪label = "HIST")
```

```
plt.title(f'{new_degree_daily_list[day][0]}\n(mu={mu1:.2f}, std={sigma1:.2f})')
plt.legend(loc = 'best')
plt.xlabel(columns[11])
plt.ylabel('density')
plt.grid()
plt.show()
```

[39]:
```
fig, axs = plt.subplots(len(new_degree_daily_list), 1, figsize=(8, 16),␣
↪sharex=False)

for i in range(len(new_degree_daily_list)):
    ax = axs[i] if len(new_degree_daily_list) > 1 else axs
    ydata1 = np.array(new_degree_daily_list[i][1])
    mu1 = np.mean(ydata1)
    sigma1 = np.std(ydata1)
    x1 = np.linspace(ydata1.min(),ydata1.max(),len(ydata1))
    y1 = norm.pdf(x1, mu1, sigma1)
```

```
    ax.plot(x1, y1, label = 'Normal Distribution', color=colors[i])
    ax.hist(ydata1, bins = 50, density = True, color=colors[i], alpha = 0.3,␣
 ↪label = "HIST")

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'{degree_daily_list[i][0]}\n(mu={mu1:.2f}, std={sigma1:.2f})')
    ax.set_xlabel(columns[11])
    ax.set_ylabel('density')

plt.tight_layout()
plt.show()
```

2025-02-20
(mu=100.03, std=5.03)
2025-02-21
(mu=99.09, std=5.50)
2025-02-22
(mu=100.04, std=6.16)
2025-02-23
(mu=100.18, std=5.17)
2025-02-24
(mu=100.38, std=6.72)
2025-02-25
(mu=99.75, std=9.11)
2025-02-26
(mu=100.87, std=4.98)
2025-02-27
(mu=100.28, std=5.35)
Normal Distribution
HIST
density
DOP[%]

[40]:
```
for i in range(len(new_degree_daily_list)):
    ydata1 = np.array(new_degree_daily_list[i][1])
    mu1 = np.mean(ydata1)
    sigma1 = np.std(ydata1)
    x1 = np.linspace(ydata1.min(),ydata1.max(),len(ydata1))
    y1 = norm.pdf(x1, mu1, sigma1)

    plt.plot(x1, y1, label = new_degree_daily_list[i][0])

plt.title(f'Normal Distribution')
plt.legend(loc = 'best')
plt.xlabel(columns[11])
plt.ylabel('density')
plt.grid()
plt.show()
```

2.5 Filterung mittels Tief- und Hochpass

[41]:
```
ydata = np.array(new_degree_daily_list[day][1])

lps = lowpass(ydata, 100)
plt.figure(figsize = (16,9))
plt.plot(ydata, label = 'Data', color = colors[day], alpha=0.3)
plt.plot(lps, label = 'LPF', color = colors[day])
plt.title(f'Applied Low Pass Filter\n of DOP [%] on␣
 ↪{new_degree_daily_list[day][0]}')
plt.grid()
plt.legend(loc = 'best')
plt.show()
```

[42]:
```
ydata = np.array(new_degree_daily_list[day][1])

lps = lowpass(ydata, 10000)
plt.figure(figsize = (16,9))
plt.plot(ydata, label = 'Data', color = colors[day], alpha=0.3)
plt.plot(lps, label = 'LPF', color = colors[day])
plt.title(f'Applied Low Pass Filter\n of {columns[11]} on␣
 ↪{new_degree_daily_list[day][0]}')
plt.grid()
plt.legend(loc = 'best')
plt.show()
```

[43]:
```
cutoff = 1.0
btype='low'

ydata = np.array(new_degree_daily_list[day][1])
lps = butter_filter(ydata, cutoff = cutoff, btype=btype)
plt.figure(figsize = (16,9))
plt.plot(ydata, label = 'Data', color = colors[day], alpha=0.3)
plt.plot(lps, label = 'LPF', color = colors[day])
plt.title(f'Applied Low Pass Filter\n of {columns[11]} on␣
↪{new_degree_daily_list[day][0]}')
plt.grid()
plt.legend(loc = 'best')
plt.show()
```

[44]:
```
fig, ax = plt.subplots(figsize = (16,9))

for date, df_day in new_degree_daily_list:
    times = (df_day.index - df_day.index.normalize()).total_seconds()
    ydata = np.array(df_day)
    lps = lowpass(ydata, 100)
    plt.plot(times, lps[:-1], label=str(date) , alpha=0.5)


ax.xaxis.set_major_locator(ticker.MultipleLocator(3600 * 2))
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_time))
ax.set_xlim(0, 24 * 3600)

plt.grid()
plt.legend(loc = 'best')
plt.title(f'Applied Low Pass Filter\nof {columns[11]}')
plt.xlabel('time')
plt.ylabel('percentage')
plt.show()
```

[45]:
```
ydata = np.array(new_degree_daily_list[day][1])

hps = highpass(ydata, 100)
plt.figure(figsize = (16,9))
plt.plot(ydata, label = 'Data', color=colors[day], alpha=0.3)
plt.plot(hps, label = 'HPF', color=colors[day])
plt.title(f'Applied High Pass Filter\n of {columns[11]} on␣
↪{new_degree_daily_list[day][0]}\n')
plt.grid()
plt.legend(loc = 'best')
plt.show()
```

[46]:
```
cutoff = 1.0
btype='high'

ydata = np.array(new_degree_daily_list[day][1])
lps = butter_filter(ydata, cutoff = cutoff, btype=btype)
plt.figure(figsize = (16,9))
plt.plot(ydata, label = 'Data', color = colors[day], alpha=0.3)
plt.plot(lps, label = 'LPF', color = colors[day])
plt.title(f'Applied Low Pass Filter\n of {columns[11]} on␣
↪{new_degree_daily_list[day][0]}')
plt.grid()
plt.legend(loc = 'best')
plt.show()
```

[47]:
```
fig, ax = plt.subplots(figsize = (16,9))

for date, df_day in new_degree_daily_list:
    times = (df_day.index - df_day.index.normalize()).total_seconds()
    ydata = np.array(df_day)
    hps = highpass(ydata, 100)
    plt.plot(times, hps, label=str(date) , alpha=0.5)


ax.xaxis.set_major_locator(ticker.MultipleLocator(3600 * 2))
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_time))
ax.set_xlim(0, 24 * 3600)

plt.grid()
plt.legend(loc = 'best')
plt.title(f'Applied High Pass Filter\nof {columns[11]}')
plt.xlabel('time')
plt.ylabel('percentage')
plt.show()
```

3 Winkelparameter (Azimut [°], Elliptizität [°])

[48]:
```
angle = pd.read_csv(filename,skiprows=skip, sep=sep, usecols=[columns[0],␣
↪columns[9], columns[10]])
```

[49]:
```
angle[columns[0]] = pd.to_datetime(angle[columns[0]])
angle.set_index(columns[0], inplace=True)
```

3.1 Tägliche Aufteilung der Messdaten

[50]:
```
angle_daily = angle.groupby(angle.index.date)
angle_daily_list = list(angle_daily)
```

[51]:
```
for date, df_day in angle_daily_list:
    print(date, '\n')
    print(df_day.describe())
    print('')
```

```
2025-02-20

         Azimuth[°]   Ellipticity[°]
count  148479.000000    148479.000000
mean       -2.877728        -1.448010
std        46.737623        19.713427
min       -90.000000       -44.790000
```

```
25%       -35.800000       -16.770000
50%        -4.210000        -2.760000
75%        30.190000        12.230000
max        90.000000        44.920000
```

2025-02-21

```
         Azimuth[°]   Ellipticity[°]
count  338320.000000    338320.000000
mean       13.072056         0.313493
std        45.974164        20.598256
min       -90.000000       -44.940000
25%       -15.320000       -16.080000
50%        19.500000        -0.920000
75%        46.960000        16.880000
max        90.000000        44.900000
```

2025-02-22

```
         Azimuth[°]   Ellipticity[°]
count  338429.000000    338429.000000
mean       -2.956018        -1.922901
std        48.328637        19.591658
min       -90.000000       -44.940000
25%       -46.530000       -17.030000
50%        -0.050000        -1.860000
75%        33.630000        12.850000
max        90.000000        44.880000
```

2025-02-23

```
         Azimuth[°]   Ellipticity[°]
count  338655.000000    338655.000000
mean       -5.964077         2.385165
std        54.157520        19.226503
min       -90.000000       -44.860000
25%       -54.040000       -12.650000
50%        -8.810000         1.890000
75%        38.490000        17.310000
max        90.000000        44.970000
```

2025-02-24

```
         Azimuth[°]   Ellipticity[°]
count  338847.000000    338847.000000
mean       -9.769522         5.084350
std        50.936056        19.320113
min       -90.000000       -44.650000
```

```
25%       -52.560000        -8.840000
50%       -17.920000         6.930000
75%        32.145000        19.320000
max        90.000000        44.930000

2025-02-25

         Azimuth[°]   Ellipticity[°]
count  338305.000000    338305.000000
mean        7.664524        -2.890380
std        44.152672        16.785454
min       -90.000000       -44.840000
25%       -17.180000       -14.430000
50%         7.330000        -5.480000
75%        42.100000         7.950000
max        90.000000        44.810000

2025-02-26

         Azimuth[°]   Ellipticity[°]
count  338653.000000    338653.000000
mean      -16.139451        -9.472128
std        54.435849        16.267431
min       -90.000000       -44.980000
25%       -61.920000       -21.800000
50%       -27.630000       -13.290000
75%        20.140000         2.100000
max        90.000000        44.910000

2025-02-27

         Azimuth[°]   Ellipticity[°]
count  189966.000000    189966.000000
mean       -2.210967         2.472772
std        48.599497        15.998868
min       -90.000000       -44.860000
25%       -45.140000        -7.740000
50%        -8.465000         2.470000
75%        45.420000        12.920000
max        90.000000        44.870000
```

[52]:
```
plt.figure(figsize = (16,9))
for date, df_day in angle_daily_list:
    values = df_day[columns[9]]
    plt.plot(values, label=str(date), alpha=0.5)
plt.grid()
```

```
plt.legend(loc = 'best')
plt.title(f'Course of {columns[9]} during full measurement')
plt.xlabel('time')
plt.ylabel('degree')
plt.show()
```

[53]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(16, 16),␣
↪sharex=False)

for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    df_day = angle_daily_list[i][1][columns[9]]

    ax.plot(df_day.index, df_day, label=columns[9], color=colors[i])
    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'{columns[9]} on {angle_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('degree')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

[54]:
```
plt.figure(figsize = (16,9))
for date, df_day in angle_daily_list:
    values = df_day[columns[10]]
    plt.plot(values, label=str(date), alpha=0.5)
plt.grid()
plt.legend(loc = 'best')
plt.title(f'Course of {columns[10]} during full measurement')
plt.xlabel('time')
plt.ylabel('degree')
plt.show()
```

[55]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(16, 16),␣
↪sharex=False)

for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    df_day = angle_daily_list[i][1][columns[10]]

    ax.plot(df_day.index, df_day, label=columns[10], color=colors[i])
    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'{columns[10]} on {angle_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('degree')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

3.2 Tägliche Normalverteilung mit Histogramm

[56]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(8, 16), sharex=False)

for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    ydata1 = np.array(angle_daily_list[i][1][columns[9]])
    mu1 = np.mean(ydata1)
    sigma1 = np.std(ydata1)
    x1 = np.linspace(ydata1.min(),ydata1.max(),len(ydata1))
    y1 = norm.pdf(x1, mu1, sigma1)
```

```
    ax.plot(x1, y1, label = 'Normal Distribution', color=colors[i])
    ax.hist(ydata1, bins = 50, density = True, color=colors[i], alpha = 0.3,␣
 ↪label = "HIST")

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'{columns[9]} on {angle_daily_list[i][0]}\n(mu={mu1:.2f},␣
 ↪std={sigma1:.2f})')
    ax.set_xlabel(columns[9])
    ax.set_ylabel('density')

plt.tight_layout()
plt.show()
```

Azimuth[°] on 2025-02-20
(mu=-2.88, std=46.74)
Azimuth[°] on 2025-02-21
(mu=13.07, std=45.97)
Azimuth[°] on 2025-02-22
(mu=-2.96, std=48.33)
Azimuth[°] on 2025-02-23
(mu=-5.96, std=54.16)
Azimuth[°] on 2025-02-24
(mu=-9.77, std=50.94)
Azimuth[°] on 2025-02-25
(mu=7.66, std=44.15)
Azimuth[°] on 2025-02-26
(mu=-16.14, std=54.44)
Azimuth[°] on 2025-02-27
(mu=-2.21, std=48.60)
Normal Distribution
HIST
density
Azimuth[°]

[57]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(8, 16), sharex=False)

for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    ydata1 = np.array(angle_daily_list[i][1][columns[10]])
    mu1 = np.mean(ydata1)
    sigma1 = np.std(ydata1)
    x1 = np.linspace(ydata1.min(),ydata1.max(),len(ydata1))
    y1 = norm.pdf(x1, mu1, sigma1)

    ax.plot(x1, y1, label = 'Normal Distribution', color=colors[i])
    ax.hist(ydata1, bins = 50, density = True, color=colors[i], alpha = 0.3,␣
 ↪label = "HIST")

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'{columns[10]} on {angle_daily_list[i][0]}\n(mu={mu1:.2f},␣
 ↪std={sigma1:.2f})')
    ax.set_xlabel(columns[10])
    ax.set_ylabel('density')

plt.tight_layout()
plt.show()
```

Ellipticity[°] on 2025-02-20
(mu=-1.45, std=19.71)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-21
(mu=0.31, std=20.60)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-22
(mu=-1.92, std=19.59)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-23
(mu=2.39, std=19.23)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-24
(mu=5.08, std=19.32)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-25
(mu=-2.89, std=16.79)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-26
(mu=-9.47, std=16.27)
Normal Distribution
HIST
density
Ellipticity[°]
Ellipticity[°] on 2025-02-27
(mu=2.47, std=16.00)
Normal Distribution
HIST
density
Ellipticity[°]

3.3 Filterung mittels Tief- und Hochpass

[58]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(8, 16), sharex=False)

for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    df_day = angle_daily_list[i][1][columns[9]]
    ydata = np.array(df_day)
    lps = lowpass(ydata, 100)

    ax.plot(df_day.index, lps[:-1], label=columns[9], color=colors[i], alpha=0.
 ↪5)

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'Applied Low Pass Filter\n of {columns[9]}on␣
 ↪{angle_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('degree')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

Applied Low Pass Filter
of Azimuth[°] on 2025-02-20
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-21
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-22
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-23
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-24
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-25
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-26
Azimuth[°]
degree
time
Applied Low Pass Filter
of Azimuth[°] on 2025-02-27
Azimuth[°]
degree
time

[59]:
```
fig, ax = plt.subplots(figsize = (16,9))

for date, df_day in angle_daily_list:
    times = (df_day.index - df_day.index.normalize()).total_seconds()
    ydata = np.array(df_day[columns[9]])
    lps = lowpass(ydata, 100)
    plt.plot(times, lps[:-1], label=str(date) , alpha=0.5)


ax.xaxis.set_major_locator(ticker.MultipleLocator(3600 * 2))
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_time))
ax.set_xlim(0, 24 * 3600)

plt.grid()
plt.legend(loc = 'best')
plt.title(f'Applied Low Pass Filter\nof {columns[9]}')
plt.xlabel('time')
plt.ylabel('degree')
plt.show()
```

[60]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(8, 16), sharex=False)
```

```
for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    df_day = angle_daily_list[i][1][columns[10]]
    ydata = np.array(df_day)
    lps = lowpass(ydata, 100)

    ax.plot(df_day.index, lps[:-1], label=columns[10], color=colors[i], alpha=0.
 ↪5)

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'Applied Low Pass Filter\n of {columns[10]}on␣
 ↪{angle_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('degree')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

Applied Low Pass Filter
of Ellipticity[°] on 2025-02-20
Ellipticity[°]
degree
25
0
−25
13:00:00 14:00:00 15:00:00 16:00:00 17:00:00 18:00:00 19:00:00 20:00:00 21:00:00 22:00:00 23:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-21
Ellipticity[°]
degree
25
0
−25
00:00:00 03:00:00 06:00:00 09:00:00 12:00:00 15:00:00 18:00:00 21:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-22
Ellipticity[°]
degree
25
0
−25
00:00:00 03:00:00 06:00:00 09:00:00 12:00:00 15:00:00 18:00:00 21:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-23
Ellipticity[°]
degree
25
0
−25
00:00:00 03:00:00 06:00:00 09:00:00 12:00:00 15:00:00 18:00:00 21:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-24
Ellipticity[°]
degree
25
0
−25
00:00:00 03:00:00 06:00:00 09:00:00 12:00:00 15:00:00 18:00:00 21:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-25
Ellipticity[°]
degree
25
0
−25
00:00:00 03:00:00 06:00:00 09:00:00 12:00:00 15:00:00 18:00:00 21:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-26
Ellipticity[°]
degree
25
0
−25
00:00:00 03:00:00 06:00:00 09:00:00 12:00:00 15:00:00 18:00:00 21:00:00 00:00:00
time
Applied Low Pass Filter
of Ellipticity[°] on 2025-02-27
Ellipticity[°]
degree
25
0
−25
00:00:00 02:00:00 04:00:00 06:00:00 08:00:00 10:00:00 12:00:00 14:00:00
time

[61]:
```
fig, ax = plt.subplots(figsize = (16,9))

for date, df_day in angle_daily_list:
    times = (df_day.index - df_day.index.normalize()).total_seconds()
    ydata = np.array(df_day[columns[10]])
    lps = lowpass(ydata, 100)
    plt.plot(times, lps[:-1], label=str(date) , alpha=0.5)


ax.xaxis.set_major_locator(ticker.MultipleLocator(3600 * 2))
ax.xaxis.set_major_formatter(ticker.FuncFormatter(format_time))
ax.set_xlim(0, 24 * 3600)

plt.grid()
plt.legend(loc = 'best')
plt.title(f'Applied Low Pass Filter\nof {columns[10]}')
plt.xlabel('time')
plt.ylabel('degree')
plt.show()
```

[62]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(8, 16), sharex=False)
```

```
for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    df_day = angle_daily_list[i][1][columns[9]]
    ydata = np.array(df_day)
    hps = highpass(ydata, 100)

    ax.plot(df_day.index, hps, label=columns[9], color=colors[i], alpha=0.5)

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'Applied High Pass Filter\n of {columns[9]}on␣
 ↪{angle_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('degree')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

Applied High Pass Filter
of Azimuth[°] on 2025-02-20
Azimuth[°]
degree
100
0
−100
13:00:00
14:00:00
15:00:00
16:00:00
17:00:00
18:00:00
19:00:00
20:00:00
21:00:00
22:00:00
23:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-21
Azimuth[°]
degree
100
0
−100
00:00:00
03:00:00
06:00:00
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-22
Azimuth[°]
degree
100
0
−100
00:00:00
03:00:00
06:00:00
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-23
Azimuth[°]
degree
100
0
−100
00:00:00
03:00:00
06:00:00
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-24
Azimuth[°]
degree
100
0
−100
00:00:00
03:00:00
06:00:00
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-25
Azimuth[°]
degree
100
0
−100
00:00:00
03:00:00
06:00:00
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-26
Azimuth[°]
degree
100
0
−100
00:00:00
03:00:00
06:00:00
09:00:00
12:00:00
15:00:00
18:00:00
21:00:00
00:00:00
time

Applied High Pass Filter
of Azimuth[°] on 2025-02-27
Azimuth[°]
degree
100
0
−100
00:00:00
02:00:00
04:00:00
06:00:00
08:00:00
10:00:00
12:00:00
14:00:00
time

[63]:
```
fig, axs = plt.subplots(len(angle_daily_list), 1, figsize=(8, 16), sharex=False)

for i in range(len(angle_daily_list)):
    ax = axs[i] if len(angle_daily_list) > 1 else axs
    df_day = angle_daily_list[i][1][columns[10]]
    ydata = np.array(df_day)
    hps = highpass(ydata, 100)

    ax.plot(df_day.index, hps, label=columns[10], color=colors[i], alpha=0.5)

    ax.grid()
    ax.legend(loc='best')
    ax.set_title(f'Applied High Pass Filter\n of {columns[10]}on␣
 ↪{angle_daily_list[i][0]}')
    ax.set_xlabel('time')
    ax.set_ylabel('degree')

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

plt.tight_layout()
plt.show()
```

Applied High Pass Filter
of Ellipticity[°] on 2025-02-20
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-21
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-22
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-23
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-24
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-25
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-26
Ellipticity[°]
degree
time
Applied High Pass Filter
of Ellipticity[°] on 2025-02-27
Ellipticity[°]
degree
time

3.4 Darstellung auf der Poincaré-Kugel

3.4.1 Visualisierung mit dem Python-Tool „py_pol“

[64]:
```
one_second = datetime.timedelta(seconds=1)
one_minute = datetime.timedelta(minutes=1)
one_hour = datetime.timedelta(hours=1)
one_quater = datetime.timedelta(hours=6)
one_half = datetime.timedelta(hours=12)
one_day = datetime.timedelta(days=1)

start_time = datetime.datetime.strptime(str(angle.index[0]), '%Y-%m-%d %H:%M:%S.
 ↪%f')
end_time_second = start_time + one_second
end_time_minute = start_time + one_minute
end_time_hour = start_time + one_hour
end_time_quater = start_time + one_quater
end_time_half = start_time + one_half
end_time_day = start_time + one_day
```

[70]:
```
first_val = angle.iloc[0]

az = np.array(first_val[columns[9]]) * degrees
el = np.array(first_val[columns[10]]) * degrees

S = Stokes("Calculated Point")
S.general_azimuth_ellipticity(az, el)

fig = S.draw_poincare(kind="scatter")
#fig.show()
Image(filename="./img/first_val.png")
```

[70]:

[71]:
```
first_second = angle.loc[(angle.index >= start_time) & (angle.index <␣
↪end_time_second)]

az = np.array(first_second[columns[9]].values) * degrees
el = np.array(first_second[columns[10]].values) * degrees

S = Stokes("Calculated Points")
S.general_azimuth_ellipticity(az, el)

fig = S.draw_poincare(kind="scatter")
#fig.show()
Image(filename="./img/first_second.png")
```

[71]:

[72]:
```
first_minute = angle.loc[(angle.index >= start_time) & (angle.index <␣
↪end_time_minute)]

az = np.array(first_minute[columns[9]].values) * degrees
el = np.array(first_minute[columns[10]].values) * degrees

S = Stokes("Calculated Points")
S.general_azimuth_ellipticity(az, el)

fig = S.draw_poincare(kind="scatter")
#fig.show()
Image(filename="./img/first_minute.png")
```

[72]:

[73]:
```
first_hour = angle.loc[(angle.index >= start_time) & (angle.index <␣
↪end_time_hour)]

az = np.array(first_hour[columns[9]].values) * degrees
el = np.array(first_hour[columns[10]].values) * degrees

S = Stokes("Calculated Points")
S.general_azimuth_ellipticity(az, el)

fig = S.draw_poincare(kind="scatter")
#fig.show()
Image(filename="./img/first_hour.png")
```

[73]:

[74]:
```
first_quater = angle.loc[(angle.index >= start_time) & (angle.index <␣
↪end_time_quater)]

az = np.array(first_quater[columns[9]].values) * degrees
el = np.array(first_quater[columns[10]].values) * degrees

S = Stokes("Calculated Points")
S.general_azimuth_ellipticity(az, el)

fig = S.draw_poincare(kind="scatter")
#fig.show()
Image(filename="./img/first_quater.png")
```

[74]:

3.4.2 Eigene Berechnung und Visualisierung der Daten

[76]:
```
normalized_stokes = pd.read_csv(filename,skiprows=skip, usecols=[columns[2],␣
↪columns[3], columns[4]], sep=sep, nrows=5)
x,y,z = convert_angel(first_second[columns[9]].values,␣
↪first_second[columns[10]].values)

data = pd.DataFrame({
    "S1": x,
    "Normalized S1": normalized_stokes[columns[2]],
    "S2": y,
    "Normalized S2": normalized_stokes[columns[3]],
```

```
    "S3": z,
    "Normalized S3": normalized_stokes[columns[4]]
})
print(data)
```

```
         S1  Normalized S1        S2  Normalized S2        S3  Normalized S3
0  0.796095           0.80  0.151863           0.15  0.585807           0.59
1  0.874517           0.87 -0.013433          -0.01  0.484810           0.48
2  0.879954           0.88  0.124923           0.12  0.458339           0.46
3  0.866366           0.87  0.090142           0.09  0.491208           0.49
4  0.828781           0.83 -0.088571          -0.09  0.552519           0.55
```

[77]:
```
print("Original normalized S1 equals calculated S1? ", data['Normalized S1'].
 ↪equals(round(data['S1'], 2)))
print("Original normalized S2 equals calculated S2? ", data['Normalized S2'].
 ↪equals(round(data['S2'], 2)))
print("Original normalized S3 equals calculated S3? ", data['Normalized S3'].
 ↪equals(round(data['S3'], 2)))
```

```
Original normalized S1 equals calculated S1?  True
Original normalized S2 equals calculated S2?  True
Original normalized S3 equals calculated S3?  True
```

[78]:
```
#%matplotlib notebook # interaktiver Plot-Mode
%matplotlib inline # statischer Plot-Mode
```

```
UsageError: unrecognized arguments: # statischer Plot-Mode
```

[79]:
```
title = 'Measurement points of the first second \non the Poincaré-Sphere'
x,y,z = convert_angel(first_second[columns[9]].values,␣
 ↪first_second[columns[10]].values)
freq = calculate_freq(x, y, z)
plot_poincare(x, y, z, freq, title=title)
```

[80]:
```
title = 'Measurement points of the first minute \non the Poincaré-Sphere'
x,y,z = convert_angel(first_minute[columns[9]].values,␣
 ↪first_minute[columns[10]].values)
freq = calculate_freq(x, y, z)
plot_poincare(x, y, z, freq, title=title)
```

[81]:
```
title = 'Measurement points of the first hour \non the Poincaré-Sphere'
x,y,z = convert_angel(first_hour[columns[9]].values, first_hour[columns[10]].
 ↪values)
freq = calculate_freq(x, y, z)
plot_poincare(x, y, z, freq, title=title)
```

[82]:
```
title = 'Measurement points of the first quarter day \non the Poincaré-Sphere'
x,y,z = convert_angel(first_quater[columns[9]].values,␣
 ↪first_quater[columns[10]].values)
freq = calculate_freq(x, y, z)
plot_poincare(x, y, z, freq, title=title)
```

[83]:
```
first_half = angle.loc[(angle.index >= start_time) & (angle.index <␣
↪end_time_half)]

title = 'Measurement points of the first half day \non the Poincaré-Sphere'
x,y,z = convert_angel(first_half[columns[9]].values, first_half[columns[10]].
↪values)
freq = calculate_freq(x, y, z)
plot_poincare(x, y, z, freq, title=title)
```

[84]:
```
angle_seconds_resample = angle.resample('s').mean().ffill()
angle_seconds_resample.head()
```

[84]:

Time[date hh:mm:ss]	Azimuth[°]	Ellipticity[°]
2025-02-20 13:28:29	3.000000	15.356667
2025-02-20 13:28:30	1.072500	16.080000
2025-02-20 13:28:31	-0.640000	15.827500
2025-02-20 13:28:32	-0.456667	15.610000
2025-02-20 13:28:33	3.132500	17.012500

[85]:
```
print(angle_seconds_resample.isnull().sum())
print(type(angle.index))
```

```
Azimuth[°]         0
Ellipticity[°]     0
dtype: int64
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

[86]:
```
title= 'Secondly resampled measurement points \nfor full measurement on the␣
↪Poincaré-Sphere'

x,y,z = convert_angel(angle_seconds_resample[columns[9]].values,␣
↪angle_seconds_resample[columns[10]].values)
freq = calculate_freq(x, y, z)
plot_poincare(x, y, z, freq, title=title)
```

[87]:
```
#elev=15, azim=45
title= 'Secondly resampled measurement points \nfor full measurement on the␣
↪Poincaré-Sphere'
plot_poincare(x, y, z, freq, elev=15, azim=135, title=title)
```

[88]:
```
#elev=15, azim=45
title= 'Secondly resampled measurement points \nfor full measurement on the␣
↪Poincaré-Sphere'
plot_poincare(x, y, z, freq, elev=15, azim=225, title=title)
```

[89]:
```
#elev=15, azim=45
title= 'Secondly resampled measurement points \nfor full measurement on the␣
↪Poincaré-Sphere'
plot_poincare(x, y, z, freq, elev=15, azim=315, title=title)
```

Secondly resampled measurement points
for full measurement on the Poincaré-Sphere
S3
-S3
S2
-S2
S1
400
350
300
250
200
150
100
50
density

```
[ ]:
```