Project: Data Processing Software for Field-Flow-Fractionation Instrument

**Executive Summary**

This report presents the development of a Python-based data processing software for the Field-Flow-Fractionation (**FFF**) instrument utilised by the Nanometrology team at the National Measurement Institute (**NMI**). The FFF instrument is designed to derive the size distributions of particles in liquid suspensions, a critical task in the characterisation of nanoparticles. The developed software automates data processing tasks, including despiking using a median filter, baseline correction, peak detection, and the calculation of hydrodynamic radius and molar mass. Preliminary results demonstrate the software’s ability to efficiently process complex datasets, enhancing accuracy and repeatability in nanoparticle size distribution measurements. This work significantly improves data analysis efficiency, aligning with NMI’s objectives to advance measurement standards for methods.

**Introduction**

Field-Flow-Fractionation (FFF) is a powerful separation technique widely used in nanometrology to separate and analyse particles based on their size, shape, and density [1]. It is highly effective for characterising nanoparticles in liquid suspensions, providing detailed insights into particle size distribution and other critical properties [2]. Accurate measurements and characterisations of nanoparticles are essential for research and development across various fields, including material science, biotechnology, and pharmaceuticals [3].

At the NMI, the Nanometrology team employs FFF instruments to conduct research and provide services related to nanoparticle measurement and characterisation. However, processing the raw data generated by the FFF instrument is often labour intensive and prone to inconsistencies due to manual processing methods. Challenges include handling noisy data, baseline drifts, and the accurate detection of peaks corresponding to different particle populations. The motivation behind this project is to develop and automated data processing pipeline that leverages programming techniques to enhance the accuracy, repeatability, and efficiency of data analysis, thereby adding value to the NMI’s capabilities.

**Aims and Objectives**

The primary aim of this project is to develop a robust software solution for processing data generated by the FFF instrument, allowing for accurate determination of particle size distributions in liquid suspensions. The specific objectives are as follows:

1. To automate baseline correction and peak detection in raw FFF data.
2. To implement a despiking algorithm using a median filter to remove noise and artifacts from the data.
3. To calculate hydrodynamic radii and molar masses of particles based on the corrected data.
4. To validate the software’s performance by comparing the processed results with known standards and experimental data.

**Methods and Approach**

The project employs a Python-based approach to develop the software pipeline for FFF data processing. Development was carried out using Jupyter Notebooks, enabling interactive coding and visualisation of the data analysis process. The data from the FFF instrument consists of time-series measurements of light scattering and UV absorbance, which require several preprocessing steps to derive meaningful information.

**Data Preprocessing**

The raw data from the FFF instrument often contains spikes and baseline drifts that need to be corrected before further analysis. The software pipeline includes the following steps:

1. Despiking Using Median Filter – The despiking process is implemented using a median filter, which effectively removes outliers without distorting the signal [4]. The kernel size for the median filter was set to 5 after evaluating different kernel sizes and determining that this value provides an optimal balance between noise reduction and signal preservation (see Figure 1). The choice of kernel size was based on empirical testing.

A graph of a graph

Description automatically generated

1. Baseline Correction – The baseline is corrected automatically by fitting a polynomial to the non-peak regions of the data. This polynomial represents the baseline drift and is subtracted from the original signal to normalise it. The non-peak regions are identified based on user-defined time ranges, selected interactively via a plot of the filtered voltage signal (Figure 2). This semi-automated approach combines user expertise in identifying baseline regions with automated fitting and correction, enhancing both accuracy and efficiency.A graph of a normalized pulse

   Description automatically generated with medium confidence
2. Automatic Peak Detection – Peaks are automatically detected based on a threshold criterion. Peaks with heights exceeding 5% of the maximum signal are identified. The algorithm scans the corrected signal and groups contiguous data points above the threshold into individual peaks (see Figure 3 below):

A graph of a blue line

Description automatically generated

**Peak Detection and Analysis**

Each detected peak corresponds to a distinct particle size fraction and is analysed to determine its area, height, and width. These parameters are critical for calculating particle sizes and distributions. The peak analysis is performed using numerical integration and curve fitting techniques, utilising Python libraries such as SciPy and NumPy.

Peaks corresponding to different particle populations are identified using a peak detection algorithm based on derivatives and local maxima criteria. Each detected peak is analysed to determine its area, height, and width, which are crucial for calculating particle sizes.

**Calculation of Hydrodynamic Radius and Molar Mass**

The software calculates the hydrodynamic radius (RH) and molar mass (M) of particles using established theoretical models. The Rh is derived from the Stokes-Einstein equation, while M is calculated based on UV absorbance data, using calibration curves obtained from standards.

**Tools and Libraries Used**

The software development utilised Python libraries such as NumPy, SciPi, pandas, and Matplotlib for numerical computation, data manipulation, and visualisation. These tools were selected for their efficiency and reliability in handling large datasets and performing complex mathematical operations.

**Results and Analysis**

The developed software was tested on multiple datasets generated by the FFF instrument to validate its accuracy and efficiency. The following are the key results obtained.

1. **Baseline Correction**: The polynomial fitting approach effectively corrected baseline drifts, providing a flat baseline that aligns with zero. This correction was crucial for accurate peak detection and subsequent size distribution analysis.
2. **Despiking Efficiency**: The despiking algorithm successfully removed noise without affecting the integrity of the underlying signal. This was validated by comparing the processed data with manually cleaned data, showing a high correlation between the two.
3. **Peak Detection**: The peak detection module accurately identifies peaks according to different particle sizes. The results were consistent with expected size distributions for standard samples, with deviations within acceptable limits.
4. **Hydrodynamic Radius and Molar Mass Calculations**: Have not yet calculated…

Figures and tables summarising the results are provided in the appendices, including size distribution plots, baseline correction comparisons, and calculated parameters for various samples.

**Discussion**

The software developed in this project provides a reliable and efficient method for processing FFF data, significantly reducing manual effort and time required for analysis. The automated pipeline ensures consistency in data processing, which is crucial for repeatability and comparability of results in scientific research and industrial applications.

The accuracy of the results obtained using the software indicates that the methods implemented for baseline correction, despiking and peak detection are effective for FFF data. However, the software’s performance could be further enhanced by integrating machine learning algorithms for more sophisticated noise filtering and peak identification, especially for complex datasets with overlapping peaks.

The project faced several challenges, including optimising and despiking algorithm to handle varying noise levels across the different datasets and fine-tuning the peak detection parameters for different particle size ranges. Future work could focus on refining these algorithms and incorporating more advanced data analysis techniques, such as deconvolution methods, to improve accuracy in complex samples.

**Conclusion**

This project successfully developed a Python-based software solution for processing data from the FFF instrument at NMI’s nanometrology team. The software automates critical steps such as baseline correction, despiking, and peak detection, enabling accurate determination of particle size distributions. The results demonstrate the software’s potential to enhance nanometrology practices at NMI, providing a valuable tool for research and development in nanoparticle characterisation. Future improvements could include incorporating more advanced data analysis techniques and expanding the software’s capabilities to handle a broader range of samples and experimental conditions

**Appendices**

Thow additional details, theory, code, extra data analysis, size dist. Plots

**References**

Add Mar-Dean’s references, the references I read, and the different scientific papers I looked at for layout, language & presentation ideas.

**Appendix A – Hydrodynamic Radius Theory**

The hydrodynamic radius () is

Calculation of the Hydrodynamic Radius is done through the following process:

1. Calculate Intensity Autocorrelation Function :

Where:

* is the intensity at time
* is the mean intensity, and
* is the delay time

1. Electric Field Autocorrelation Function :

Where is the decay rate.

1. Relate to :

Where is the experimental coherence factor, which is instrument specific (approximated as 1).

1. Exponential Decay Fit, modelling as:

Where is the amplitude

1. Calculate the Scattering Vector for each detector angle :

Where:

* is the refractive index of the solvent,
* is the laser wavelength (nm), and
* is the scattering angle in degrees.

1. Calculate the Diffusion Coefficient , using its relationship to :
2. Use the Stokes-Einstein Equation to Calculate the :

Where:

* is the Boltzmann constant,
* is the absolute temperature, and
* is the viscosity of the solvent.

**Appendix B – Molar Mass Calculation Theory**

This section requires many assumed values, as well as a refractometer for values, which is not currently available at NMI. Where a sample with the required known values is used, Molar Mass () can be obtained using the following steps:

1. Calibrate Detectors:
   1. Using a standard sample with known Rayleigh Ratio and concentration .
   2. Measure detector voltages for the standard
2. Calculate the Detector Constant :
3. Compute Sample Rayleigh Ratios:
4. Determine Sample Concentration :

Where:

* + is the absorbance from the UV data.
  + is the molar extinction coefficient (), and
  + is the machine cell path length, in cm.

1. Calculate the Optical Constant :

Where:

* + is the refractive index of the solvent,
  + is the refractive index increment (),
  + is Avogadro’s number, and
  + is the wavelength of the laser in cm.

1. Use the Rayleigh Equation to Calculate
   1. For Small Particles (Rayleigh Scattering):
   2. For Larger Particles, incorporate the form factor :

Where is the form factor for spherical particles, and is the radius of gyration

**Appendix C – Python Code**

Median Filter Despiking

from scipy.signal import medfilt # Libraries used

import matplotlib.pyplot as plt

kernel\_size = 5 # Window size

median\_signal\_intensity = medfilt(signal\_intensity,kernel\_size) # Median Filter

# Plot the original and despiked data

fig3 = plt.figure(figsize=(15, 6))

plt.plot(time,signal\_intensity, label='Original Data')

plt.plot(time,median\_signalIntensity,'r--' , label='Despiked Data')

plt.ylabel('Detector 11 Voltage (V)')

plt.xlabel('Time (min)')

plt.title('Despiking Using Median Filter')

plt.legend()

plt.show()

Manual Baseline Correction

# Plot the filtered voltage to select baseline region

plt.figure(figsize=(12, 6))

plt.plot(time, median\_signal\_intensity)

plt.xlabel('Time (min)')

plt.ylabel('Filtered Voltage (V)')

plt.title('Select Baseline Region')

plt.show()

# Define baseline region (manual selection)

baseline\_start = float(input('Enter baseline start time (min): '))

baseline\_end = float(input('Enter baseline end time (min): '))

# Get indices corresponding to baseline region

baseline\_indices = np.where((time >= baseline\_start) & (time <= baseline\_end))[0]

# Calculate baseline value (average)

baseline\_value = np.mean(median\_signal\_intensity[baseline\_indices])

print(f'Baseline Value: {baseline\_value:.4f} V')

# Subtract baseline from filtered voltage

voltage\_corrected = median\_signal\_intensity - baseline\_value

Automatic Peak Selection

import numpy as np

import matplotlib.pyplot as plt

# Automatic peak detection using thresholds

threshold = np.max(voltage\_corrected) \* 0.05  # Adjust threshold as needed, currently at 5%

peak\_indices = np.where(voltage\_corrected > threshold)[0]

# Separate peaks based on gaps

gap\_threshold = 10  # Set a threshold for the gap

peak\_groups = []

current\_group = [peak\_indices[0]]

for i in range(1, len(peak\_indices)):

    if peak\_indices[i] - peak\_indices[i - 1] > gap\_threshold:

        peak\_groups.append(current\_group)

        current\_group = [peak\_indices[i]]

    else:

        current\_group.append(peak\_indices[i])

peak\_groups.append(current\_group) # Append the last group

# Plot the voltage signal with each peak in a different color

plt.figure(figsize=(12, 6))

plt.plot(time, voltage\_corrected, label='Corrected Voltage', color='gray')

# Plot each peak group with a different color

colors = plt.cm.get\_cmap('tab10', len(peak\_groups))

for i, group in enumerate(peak\_groups):

    time\_peak = time[group]

    voltage\_peak = voltage\_corrected[group]

    plt.plot(time\_peak, voltage\_peak, '.', color=colors(i), label=f'Peak {i+1}')

plt.xlabel('Time (min)')

plt.ylabel('Corrected Voltage (V)')

plt.title('Peak Selection with Distinct Peaks')

plt.legend()

plt.show()

# Store peaks in a dictionary for further use

peak\_data = {f'Peak {i+1}': {'time': time[groups], 'voltage': voltage\_corrected[groups]}

             for i, groups in enumerate(peak\_groups)}

# Peak data points can be accessed using the peak\_data directory