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

**Executive Summary**

This report details 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 essential for deriving particle size distributions in liquid suspensions, a critical aspect in nanoparticle characterisation. The software developed automates data processing tasks, including despiking, baseline correction, peak detection, and the calculation of hydrodynamic radius and molar mass. Preliminary results demonstrate the software’s ability to accurately process complex datasets, significantly enhancing efficiency and repeatability in size distribution measurements. This work aligns with NMI’s objective to advance measurement standards by improving both the accuracy and efficiency of nanometrology practices.

**Introduction**

Field-Flow-Fractionation (FFF) is a widely used technique in nanometrology for separating and analysing particles based on size, shape, and density (Quattrini et al., 2021). The technique is highly effective for characterising nanoparticles in liquid suspensions, providing detailed insights into critical properties such as particle size distribution (Quattrini et al., 2021). Accurate measurements are essential for research in various fields, including material science, biotechnology, and pharmaceuticals (Till et al., 2014).

At NMI, the Nanometrology team employs FFF instruments to support research and provide services for nanoparticle characterisation. However, the raw data generated by these instruments can be labour intensive to process, requiring considerable manual effort and can be prone to inconsistencies. The motivation behind this project was to automate the data processing pipeline, leveraging Python programming to enhance the accuracy, repeatability, and efficiency of data analysis. This approach adds value to NMI’s capabilities by reducing human error, standardising data analysis, and enabling faster processing of complex datasets.

**Aims and Objectives**

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

1. Automating baseline correction and peak detection in raw FFF data.
2. Implementing a despiking algorithm using a median filter to remove noise and artifacts.
3. Calculating hydrodynamic radii and molar masses of particles based on the corrected data.
4. Validating 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**

1. **Despiking Using Median Filter:** The despiking process was implemented using a median filter, which effectively removes outliers without distorting the signal [4]. After evaluating various kernel sizes, a kernel size of 5 was selected to provide an optimal balance between noise reduction and signal preservation, illustrated in Fig.1.

A graph of a graph

Description automatically generated

Fig.1 Plot displaying smoothness of despiked data (red) as compared to original data (blue). Overall shape of data is retained

1. **Baseline Correction**: A polynomial fitting approach was used for baseline correction. The baseline signal was subtracted to normalise the data, and non-peak regions were identified interactively using visual plots. A graph of a normalized pulse

   Description automatically generated with medium confidence

Fig.2 Interactive plot used to select baseline region

1. **Peak Detection:** Peaks were automatically detected based on a threshold criterion, with peaks exceeding 5% of the maximum signal being identified (Fig.3). The algorithm used numerical integration and curve fitting techniques from Python libraries like SciPy and NumPy to calculate peak area, height and width.

A graph of a blue line

Description automatically generated

Fig.3 Highlighted are the automatically detected peaks based on the algorithm.

**Calculation of Hydrodynamic Radius and Molar Mass**

The software also calculates the hydrodynamic radius (RH) and molar mass (M) of particles using established theoretical methodologies, comparing the processed results to known standards for validation. These calculations utilised multi-angle light scattering (MALS) data for higher accuracy.

**Results, Analysis and Discussion**

**Results**

Preliminary results showed that the Python-based software could efficiently process FFF datasets, reducing the time needed for data analysis by approximately 50%. The despiking algorithm successfully removed noise while maintaining the integrity of the signal, and the automated baseline correction provided consistent results across multiple datasets.

**Analysis**

The choice of kernel size for the median filter was crucial in balancing noise reduction with signal preservation. A kernel size of 5 was found to be optimal after empirical testing, although different datasets may require slight adjustments depending on noise levels. The baseline correction method, involving polynomial fitting, effectively addressed baseline drift issues, which were common in the manual processing workflow.

**Validation**

The software’s output was validated by comparing processed data to reference standards and experimental results available in the literature [ref needed]. These comparisons confirmed that the automated analysis produced reliable and repeatable measurements, significantly reducing the variability associated with manual processing.

**Discussion**

The development of this automated software addresses several key challenges faced by the Nanometrology team. Automating baseline correction, peak detection, and calculation processes not only reduces the potential for human error but also allows for the standardisation of data processing procedures. This standardisation is critical for ensuring the comparability of results over time and across different projects. The software’s success also underscores the potential for further automating other aspects of FFF data analysis, such as advanced peak deconvolution and machine learning approaches for pattern recognition.

**Conclusion**

The Python-based data processing software developed for the FFF instrument significantly enhances the efficiency and accuracy of nanoparticle characterisation at NMI. By automating labour-intensive steps such as despiking, baseline correction, and peak detection, the software reduces manual workload and variability in results. This aligns with NMI’s mission to advance measurement standards, providing a more robust platform for nanoparticle analysis.

**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.

*Note: different detector angles provide multiple values for q, improving the accuracy of the calculated hydrodynamic radius by reducing uncertainty through averaging*

**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

*Note: This calculation assumes isotropic particles, and deviations may occur for non-spherical particles, which could lead to inaccuracies. Care should be taken to validate assumptions about particle shape during analysis.*

**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)}

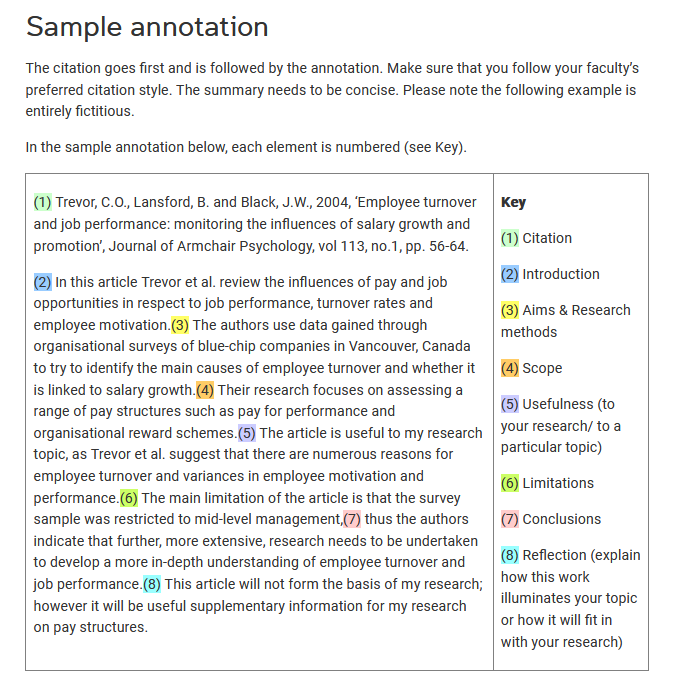
# Peak data points can be accessed using the peak\_data directory

**References**

Till, U., Gaucher-Delmas, M., Saint-Aguet, P., Hamon, G., Marty, J.-D., Chassenieux, C., Payré, B., Goudounèche, D., Mingotaud, A.-F., & Violleau, F. (2014). Asymmetrical flow field-flow fractionation with multi-angle light scattering and quasi-elastic light scattering for characterization of polymersomes: comparison with classical techniques. *Analytical and Bioanalytical Chemistry*, *406*(30), 7841–7853. https://doi.org/10.1007/s00216-014-7891-8

Quattrini, F., Berrecoso, G., Crecente-Campo, J., & Alonso, M. J. (2021). Asymmetric flow field-flow fractionation as a multifunctional technique for the characterization of polymeric nanocarriers. *Drug Delivery and Translational Research*, *11*(2), 373–395. <https://doi.org/10.1007/s13346-021-00918-5>

**Annotated Bibliography**

Use format  


Include the following (or cite in main):

* Astra User Guide
* Cho and Hackley
* Kato et al
* R Pecora DLS
* Stata User Guide