data_proc.py

Data Processing Tools

Binning and Moving Average Smoothing

ElecSus Library Documentation

Abstract

This document provides comprehensive documentation for `data_proc.py`, which implements data processing utilities for experimental spectroscopy data. The module provides binning for noise reduction and moving average smoothing, essential for preparing experimental data for fitting and analysis.

Contents

1 Theoretical Foundation

1.1 Data Binning

Axiom 1 (Noise Reduction by Averaging). *For N independent measurements with variance σ^2, the mean has variance:*

$$Var(\bar{x}) = \frac{\sigma^2}{N} \tag{1}$$

Binning adjacent points reduces noise by $\sqrt{N_{bin}}$.

Definition 1 (Bin Average). *For a bin of length b centered at point i:*

$$\bar{y}_i = \frac{1}{b} \sum_{j=i-(b-1)/2}^{i+(b-1)/2} y_j \tag{2}$$

Theorem 1 (Standard Error). *The uncertainty in the binned value is:*

$$\sigma_{\bar{y}} = \frac{std(y_j \text{ in bin})}{\sqrt{b}} \approx \frac{\sigma}{\sqrt{b}} \tag{3}$$

for stationary noise.

1.2 Moving Average Smoothing

Definition 2 (Simple Moving Average). *The moving average with window size $2n+1$:*

$$\tilde{y}_i = \frac{1}{2n+1} \sum_{j=i-n}^{i+n} y_j \tag{4}$$

Definition 3 (Triangular Weighted Average). *A weighted moving average with triangular kernel:*

$$\tilde{y}_i = \frac{\sum_{j=-n}^{n} w_j \cdot y_{i+j}}{\sum_{j=-n}^{n} w_j} \tag{5}$$

where $w_j = n + 1 - |j|$ forms a triangle from 1 to $n+1$ and back.

Theorem 2 (Triangular Filter Properties). *The triangular (tent) filter:*

- *Provides smoother results than boxcar (simple average)*
- *Equivalent to convolving two boxcar filters*
- *Preserves integral of the signal*
- *Reduces high-frequency noise while preserving edges better than Gaussian*

2 Line-by-Line Code Analysis

2.1 Module Imports

```python
import numpy as np
```

NumPy for array operations.

2.2 Binning Function: bin_data

```python
def bin_data(x,y,blength):
    """ Takes 2 arrays x and y and bins them into groups of blength. """
    if blength % 2 == 0:
        blength -= 1
    nobins = int(len(x)/blength)
    xmid = (blength-1)/2
    xbinmax = nobins*blength - xmid
```

Force odd bin length for symmetric binning. Calculate number of complete bins.

```python
    a=0
    binned = np.zeros((nobins,3))
    xout,yout,yerrout = np.array([]), np.array([]), np.array([])
```

Initialize output arrays for x, y, and error.

```python
    for i in range(int(xmid),int(xbinmax),int(blength)):
        xmin = i-int(xmid)
        xmax = i+int(xmid)
        xout = np.append(xout,sum(x[xmin:xmax+1])/blength)
        yout = np.append(yout,sum(y[xmin:xmax+1])/blength)
        yerrout = np.append(yerrout,np.std(y[xmin:xmax+1]))
    return xout,yout,yerrout
```

$$\bar{x}_k = \frac{1}{b} \sum_{j \in \text{bin } k} x_j, \quad \bar{y}_k = \frac{1}{b} \sum_{j \in \text{bin } k} y_j \tag{6}$$

$$\sigma_k = \text{std}(y_j : j \in \text{bin } k) \tag{7}$$

Compute mean x, mean y, and standard deviation within each bin.

2.3 Smoothing Function: smooth_data

```python
def smooth_data(data,degree,dropVals=False):
    """performs moving triangle smoothing with a variable degree."""
    triangle = np.array(list(range(degree))+[degree]+list(range(degree)
        )[::-1])+1
```

Create triangular kernel: for degree=3, triangle = [1,2,3,4,3,2,1].

```python
    smoothed = []
    for i in range(degree,len(data)-degree*2):
        point = data[i:i+len(triangle)]*triangle
        smoothed.append(sum(point)/sum(triangle))
```

$$\tilde{y}_i = \frac{\sum_j w_j \cdot y_{i+j}}{\sum_j w_j} \tag{8}$$

Apply weighted average at each point.

```python
    if dropVals: return smoothed
    smoothed = [smoothed[0]]*(degree+degree/2)+smoothed
```

If not dropping values, pad the beginning to maintain array length.

```python
    j = len(data)-len(smoothed)
    if j%2==1:
        for i in range(0,(j-1)/2):
            smoothed.append(data[-1-(j-1)/2+i])
            smoothed.insert(0,data[(j-1)/2-i])
        smoothed.append(data[-1])
    else:
        for i in range(0,j/2):
            smoothed.append(data[-1-i])
            smoothed.insert(0,data[i])
    return np.array(smoothed)
```

Pad end of array to match original length, using edge values.

3 Numerical Examples

3.1 Binning Example

Given 100 data points binned with $b = 5$:

- Output: 20 binned points
- Noise reduction: factor of $\sqrt{5} \approx 2.2$
- Resolution reduction: factor of 5

3.2 Smoothing Example

For degree $= 3$:

$$\text{weights} = [1, 2, 3, 4, 3, 2, 1], \quad \sum w = 16 \tag{9}$$

$$\tilde{y}_i = \frac{y_{i-3} + 2y_{i-2} + 3y_{i-1} + 4y_i + 3y_{i+1} + 2y_{i+2} + y_{i+3}}{16} \tag{10}$$

4 Trade-offs and Considerations

4.1 Binning

Advantage	Disadvantage
Reduces noise by $\sqrt{b}$	Reduces resolution by factor b
Provides error estimate	May miss narrow features
Reduces data size	Information loss

Table 1: Binning trade-offs

4.2 Smoothing

5 Applications

5.1 Pre-processing for Fitting

Binning reduces the number of data points, speeding up fitting while maintaining signal quality:

$$\chi^2 = \sum_{i=1}^{N/b} \frac{(\bar{y}_i - f(\bar{x}_i))^2}{\sigma_i^2} \tag{11}$$

Advantage	Disadvantage
Preserves data length	Broadens sharp features
Continuous output	Edge effects
Adjustable degree	Correlation between points

Table 2: Smoothing trade-offs

5.2 Noise Estimation

The standard deviation from binning provides a noise estimate:

$$\sigma_{noise} \approx \mathrm{mean}(\sigma_k) \cdot \sqrt{b} \tag{12}$$

6 Summary

The `data_proc.py` module provides:

1. `bin_data(x, y, blength)`: Bin data with error estimation
   - Returns: (x_binned, y_binned, y_error)
   - Bin length forced to odd for symmetry
2. `smooth_data(data, degree, dropVals=False)`: Triangular smoothing
   - Triangular kernel for smooth filtering
   - Optional edge padding to preserve length

Usage:

```python
from data_proc import bin_data, smooth_data

# Bin noisy data
x_bin, y_bin, y_err = bin_data(x_raw, y_raw, 5)

# Smooth data
y_smooth = smooth_data(y_raw, degree=3)
```