
powerxrd

Release 2.1

Andrew Garcia, Ph.D.

May 05, 2023

CONTENTS

1	The Open-Source Python Package That'll Make You Want to Say 'Origin Who?'	1
1.1	Contributors Wanted: Develop Rietveld Refinement for PowerXRD's Vertical XRD Analysis Integration	1
1.2	Colab Notebook	2
2	Contents	3
2.1	Usage	3
2.2	API Reference	10
3	Rietveld Refinement	15
3.1	Rietveld refinement	15
	Python Module Index	17
	Index	19

THE OPEN-SOURCE PYTHON PACKAGE THAT'LL MAKE YOU WANT TO SAY 'ORIGIN WHO?'

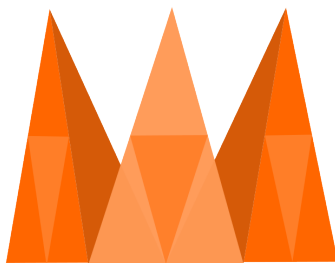


Fig. 1: Check out [powerxrd's open-source on GitHub](#)

A Python package designed to manage data from powder XRD experiments. The sole open-source Github project that is known to be developing a Rietveld refining method. In a nutshell, powerxrd is an open-source Python package for XRD data analysis. While Origin may still be preferred for more complex XRD analysis, the average Python user may find powerxrd to be more user-friendly than Origin for this type of analysis.

Check out the [Usage](#) section for further information, including how to [Install](#) the project.

1.1 Contributors Wanted: Develop Rietveld Refinement for PowerXRD's Vertical XRD Analysis Integration

We're seeking open-source contributors to help us develop a complete and efficient Rietveld refinement method for PowerXRD. This ambitious project has the potential to replace more complex refinement software, such as MAUD and Profex, by integrating all XRD processing steps from data acquisition to crystal analysis and plotting. While this will be a significant undertaking, we believe the potential benefits are substantial.

If you are interested in getting involved in this exciting project, we would love to have your contributions. There are several ways you can contribute, including familiarizing yourself with Rietveld refinement by checking the [Rietveld refinement](#) section, helping develop the Rietveld class in the [source file](#), contributing to the enhancement issue on our [Issue page](#), or sponsoring our project on the [main Github repository page](#).

```
class powerxrd.Rietveld(x_exp=[], y_exp=[])
```

refine()

Performs Rietveld refinement on the experimental data.

This function uses the fixed parameters specified by the user to perform a Rietveld refinement on the experimental data. It then generates a report of the fit results and plots the data with the initial and best fits. Data is then saved in a format to be loaded to the Chart class for additional plot processing.

Example Usage: To refine the data from 'my_data.xy' file:

```
import powerxrd as xrd

x, y = xrd.Data('my_data.xy').importfile()      # Import data from file
model = xrd.Rietveld(x, y)                      # Create Rietveld model
.
.
.
model.refine()                                  # Perform Rietveld refinement
```

1.2 Colab Notebook

If you prefer learning using a Jupyter notebook style, take a look at our Colab Notebook.



Note: This project is under active development.

CONTENTS

2.1 Usage

2.1.1 Installation

It is recommended you use powerxrd through a virtual environment. You may follow the below simple protocol to create the virtual environment, run it, and install the package there:

```
$ virtualenv venv
$ source venv/bin/activate
(.venv) $ pip install powerxrd
```

To exit the virtual environment, simply type `deactivate`. To access it at any other time again, enter with the above source command.

2.1.2 Data Preparation

First, we need to acquire all the synthetic data available on Github, which can be found at [this address](https://raw.githubusercontent.com/andrewrgarcia/powerxrd/main/synthetic-data/sample). Although it's possible to manually download each file by opening the raw text and saving it onto your system, this method can be time-consuming. A more efficient approach would be to use an automated `wget` routine in your command line, as shown below.

```
!wget https://raw.githubusercontent.com/andrewrgarcia/powerxrd/main/synthetic-data/sample
↪{0..1}.xy
!wget https://raw.githubusercontent.com/andrewrgarcia/powerxrd/main/synthetic-data/sample
↪{0..4}.csv
```

It's important to note that these files are not real experimental XRD data, but artificially-generated XRD patterns produced by a function that creates peaks with random heights and random locations.

Basic functionality of powerxrd is shown through the below examples.

2.1.3 Background subtraction

Single Plot with Emission Lines

The below example processes the .xy file into a 2-D np.array (x and y arrays) defined as data with the importfile() method from xrd.Data and runs the plot operation of it.

It then processes the contents of the data variable into the xrd.Chart class by unpackaging the x and y arrays in data with the *data operation and creates a chart object.

This object then runs the emission_lines method to find the location of the secondary radiation source for a peak between 10 and 20 2-theta degrees and runs the plot operation internally with the show=True kwarg. The chart object then runs the backsub method to subtract the background. The processed plot commands are then shown with plt.show.

```
#import packages
import powerxrd as xrd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

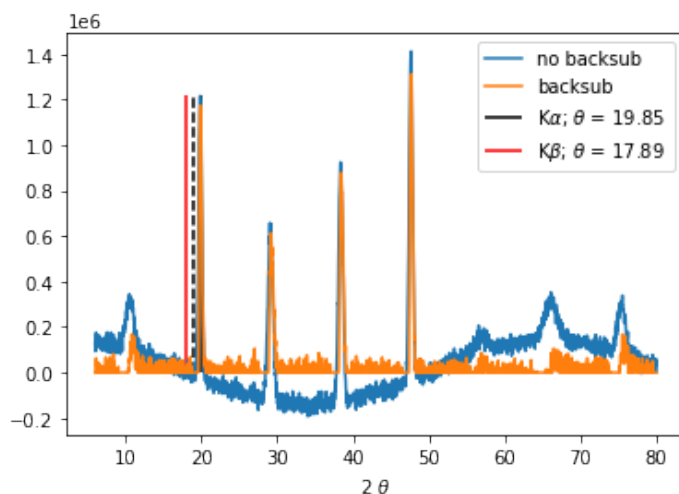
def test_backsub():

    data = xrd.Data('sample1.xy').importfile()
    plt.plot(*data,label='no backsub')

    chart = xrd.Chart(*data)
    chart.emission_lines(xrange_Ka=[10,20], show=True)
    plt.plot(*chart.backsub(),label='backsub')
    plt.xlabel('2  $\theta$ ')
    plt.legend()
    plt.show()

test_backsub()
```

```
>>> [Out]
local_max -- max x: 19.84993747394748 max y: 1215615.5744729957
```



Multiple Plots in One Chart

Multiple plots of background subtracted data can also be made and overlaid in a chart with matplotlib's `plt.subplot` method. Here `importfile('csv')` from `xrd.Data` is used to process comma-separated value (.csv) files.

```
def test_backsub_multiplt():

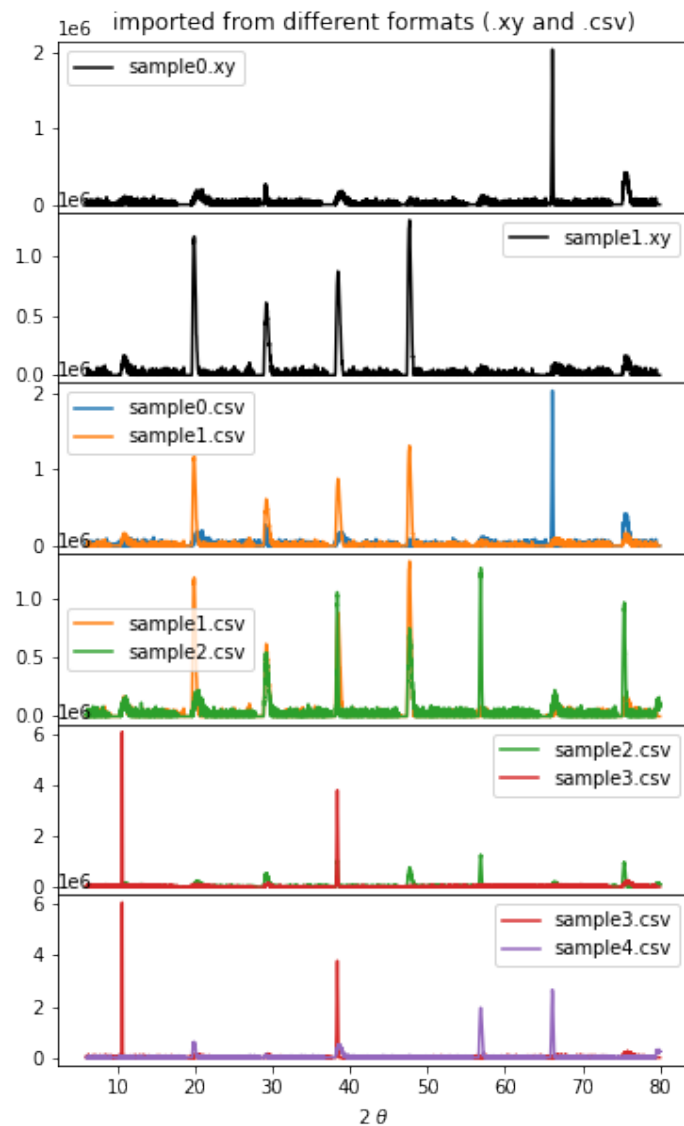
    fig, axs = plt.subplots(6, 1, figsize=(6,10), sharex=True)
    fig.subplots_adjust(hspace=0)

    # xrd.Data import tab-separated files (.xy) file with importfile()
    for i in range(2):
        data = xrd.Data('sample{}.xy'.format(i)).importfile()
        chart = xrd.Chart(*data)
        axs[i].plot(*chart.backsub(), color='k', label='sample{}.xy'.format(i))
        axs[i].legend()

    # xrd.Data can now also import csv file with .importfile('csv') option
    for j in range(2):
        for i in range(1,5):
            data = xrd.Data('sample{}.csv'.format(i+j-1)).importfile()
            chart = xrd.Chart(*data)
            axs[i+1].plot(*chart.backsub(), color='C'+str(i+j-1), label='sample{}.csv'.
↪format(i+j-1))
            axs[i+1].legend()

    plt.xlabel('2  $\theta$ ')
    # plt.suptitle('*all plots below are from synthetic data (i.e. not real XRD)')
    axs[0].set_title('imported from different formats (.xy and .csv)')
    plt.show()

test_backsub_multiplt()
```



2.1.4 Crystallite Size - Peak Calculations

Single Peak

The SchPeak method from the xrd.Chart class is used to calculate the Scherrer length of the peak in the 2-theta range xrange of 18 to 22. The show kwarg of this method is set to True to run the plot processing information to plot the highlighted peak (magenta) and the Gaussian fit of the peak (dashed cyan). SchPeak also outputs all the fitting information used to calculate the crystallite size with the Scherrer equation.

```
def test_sch():
    data = xrd.Data('sample1.xy').importfile()
    chart = xrd.Chart(*data)

    chart.backsub(tol=1.0, show=True)
```

(continues on next page)

(continued from previous page)

```

chart.SchPeak(xrange=[18,22], verbose=True, show=True)
plt.xlabel('2  $\theta$ ')
plt.title('backsub and Scherrer width calculation')
plt.show()

test_sch()

```

```

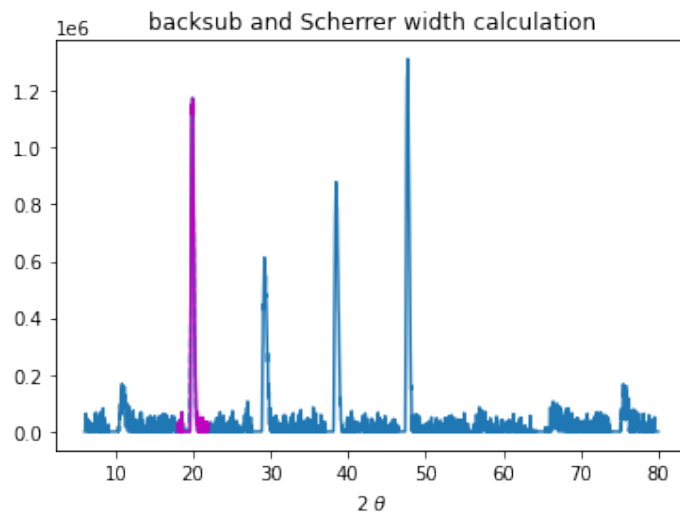
>>> [Out]
-Gaussian fit results-
y-shift 10071.343657500349
amplitude 498186.5044519722
mean 19.921493157135924
sigma 0.1692913723155234
covariance matrix
[[ 2.20553363e+07 -1.98537382e+07 -3.73304414e-08 -4.49772395e+00]
 [-1.98537382e+07  7.90011550e+07 -2.89541102e-09  1.78971558e+01]
 [-3.73304414e-08 -2.89541102e-09  9.41177383e-06 -8.26802823e-12]
 [-4.49772395e+00  1.78971558e+01 -8.26802823e-12  1.03289908e-05]]

```

SchPeak: Scherrer width calc. for peak in range of [18,22]

FWHM == $\sigma * 2 * \sqrt{2 * \ln(2)}$: 0.39865071697939203 degrees K (shape factor): 0.9 K-alpha: 0.15406 nm max
 2-theta: 19.91162984576907 degrees

SCHERRER WIDTH: 20.23261907915097 nm



Multiple Peaks (Automated)

The `allpeaks` method from the `xrd.Chart` class is used to automate the calculation for all peaks present [within a certain peak height tolerance] in the XRD spectrum. This method calls the `SchPeak` multiple times and finds peak maxima through a recursion algorithm which crops the ranges to find local maxima from left to right recursively.

`allpeaks` takes 2 kwargs: The first one is `tols`, where `tols[0]` (default=0.2) is the threshold of the height required for a peak to be considered for the Scherrer calculation, and `tols[1]` (default = 0.8) is the “guessed” average half-width distance from the top of every peak to one of their tails.

```
def test_allpeaks():

    data = xrd.Data('sample1.csv').importfile()
    chart = xrd.Chart(*data)

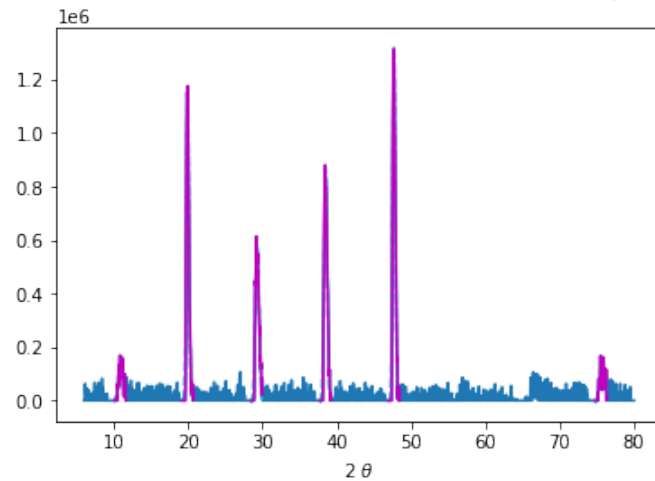
    chart.backsub(tol=1, show=True)
    chart.allpeaks(tols=(0.1, 0.8), verbose=False, show=True)
    plt.xlabel('2  $\theta$  /  $^{\circ}$ ')
    plt.suptitle('backsub & Automated Scherrer width calculation of all peaks*')
    plt.show()

test_allpeaks()
```

>>> [Out]

```
-----
↪ -
ALLPEAKS: Automated Scherrer width calculations with a recursive search of local maxima
--
local_max -- max x: 47.704043351396415 max y: 1311776.3933334802
--
--
local_max -- max x: 47.704043351396415 max y: 1311776.3933334802
local_max -- max x: 19.91162984576907 max y: 1173162.1712873918
local_max -- max x: 10.842851187995 max y: 168253.24331045512
local_max -- max x: 8.406002501042101 max y: 82154.38375744826
local_max -- max x: 12.292621925802418 max y: 74441.86925361764
local_max -- max x: 38.45018757815757 max y: 878296.1296163809
local_max -- max x: 29.19633180491872 max y: 612236.4986085768
local_max -- max x: 27.037098791162983 max y: 106100.29651133435
local_max -- max x: 35.488953730721136 max y: 85476.80399923288
local_max -- max x: 40.45518966235932 max y: 72528.3080639828
local_max -- max x: 75.52730304293456 max y: 167210.87548438687
local_max -- max x: 66.45852438516049 max y: 106019.89694947231
local_max -- max x: 76.85368903709879 max y: 78455.99312843043
--
SUMMARY (.csv format):
2-theta / deg, Intensity, Sch width / nm
10.842851187995, 168253.24331045512, 9.441628969524054
19.91162984576907, 1173162.1712873918, 20.135492415422068
29.19633180491872, 612236.4986085768, 14.141811480998149
38.45018757815757, 878296.1296163809, 16.68845964623238
47.704043351396415, 1311776.3933334802, 20.745821993861888
75.52730304293456, 167210.87548438687, 11.141240095107339
```

backsub & Automated Scherrer width calculation of all peaks*



2.1.5 Noise Reduction through a Running Average

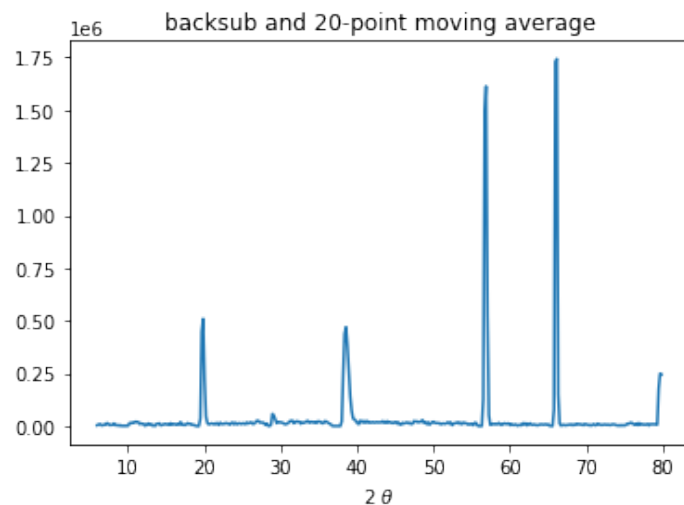
The `mav` method from the `xrd.Data` class outputs the x,y data made from a running “n” point average of the original data. Below, you’ll see `backsub` is used in combination with `mav` to render the plot. `mav` should be used with care as the operation may result in a substantial loss of resolution.

```
def test_mav():

    data = xrd.Data('sample4.csv').importfile()
    chart = xrd.Chart(*data)

    chart.backsub()
    n = 20
    plt.plot(*chart.mav(n))
    plt.xlabel('2  $\theta$ ')
    plt.title('backsub and {}-point moving average'.format(n))
    plt.show()

test_mav()
```



2.2 API Reference

2.2.1 Global Methods

class powerxrd.**braggs**(twotheta, lmda=1.54)

interplanar spacing “d_hkl” from Braggs law

class powerxrd.**scherrer**(K, lmda, beta, theta)

Scherrer equation

class powerxrd.**funcgauss**(x, y0, a, mean, sigma)

Gaussian equation

2.2.2 Local Methods to Data class

class powerxrd.**Data**(file)

__init__(file)

Data structure.

Parameters

file

[str] file name and/or path for XRD file in .xy format

2.2.3 Local Methods to Chart class

class powerxrd.**Chart**(*x*, *y*)

__init__(*x*, *y*)

Chart structure. Constructs x-y XRD data to manipulate and analyze.

Parameters

x

[np.array(float)] array with x-data 2-theta values

y

[np.array(float)] array with y-data peak intensity values

K

[float] dimensionless shape factor for Scherrer equation (default 0.9)

lambdaKa

[float] X-ray wavelength of alpha radiation

lambdaKi

[float] X-ray wavelength of “i” radiation (beta, gamma, other)

SchPeak(*xrange*=[12, 13], *verbose*=True, *show*=True)

Scherrer width calculation for peak within a specified range

Parameters

xrange

[[]](float)] range of x-axis (2-theta) where peak to be calculated is found

show: bool

show plot of XRD chart

XRD_int_ratio(*xR1*=[8.88, 9.6], *xR2*=[10.81, 11.52])

Calculate relative peak intensity (i.e. comparing one peak to another)

allpeaks(*tols*=(0.2, 0.8), *verbose*=False, *show*=True)

Driver code for allpeaks recursion : Automated Scherrer width calculation of all peaks

Parameters

tols

[(float, float)] tolerances for recursion tol[0]: Minimum peak height to be calculated as a percent of the chart's global maximum (default=0.2 [20% of global maximum]) tol[1]: Average distance from peak (top) to trough (bottom) of all peak (default=0.8)

show: bool

show plot of XRD chart

allpeaks_recur(*left*=0, *right*=1, *tols_*=(200000.0, 0.8), *schpeaks*=[], *verbose*=False, *show*=True)

recursion component function for main allpeaks function below

backsub(*tol=1, show=False*)

Background subtraction operation. This function is a running conditional statement which evaluates whether a small increase in the x-direction will increase the magnitude of the y variable beyond a certain tolerance.

Parameters

tol

[float, optional] Tolerance for background subtraction. The function evaluates whether a small increase in the x-direction will increase the magnitude of the y variable beyond a certain tolerance value. This tolerance value is defined as a percentage of the y variable at each point. Peaks above this tolerance are considered and their background is removed. Default value is 1.

show

[bool, optional] Whether to show the plot of the XRD chart. Default value is False.

emission_lines(*xrange_Ka=[10, 20], show=True*)

Emission lines arising from different types of radiation i.e. K_beta radiation wavelength of K_beta == 0.139 nm

Parameters

show: bool

show plot of XRD chart

xrange_Ka

[[](float)] range of x-axis (2-theta) for K_alpha radiation

gaussfit(*verbose=True*)

Fit of a Gaussian curve (“bell curve”) to raw x-y data

local_max(*xrange=[12, 13]*)

Maximum finder in specified xrange

Parameters

xrange_Ka

[[](float)] range of x to find globalmax

mav(*n=1, show=False*)

Function for an “n” point moving average.

powerxrd

powerxrd

RIETVELD REFINEMENT

3.1 Rietveld refinement

Rietveld refinement is a method for refining crystal structures from X-ray and neutron powder diffraction data. The method was developed by Hugo Rietveld in 1969 and is widely used in materials science to determine the crystal structure of crystalline materials. The Rietveld method involves modeling the diffraction pattern of a crystal using a combination of known structural parameters and refined parameters that describe any deviations from the ideal structure. The refinement process involves adjusting these parameters to minimize the difference between the observed and calculated diffraction patterns. The refined crystal structure can then be used to gain insights into the properties and behavior of the material under investigation.

Below are some useful, hand-picked references for developing the Rietveld refinement tool in powerxrd:

3.1.1 Literature

Rietveld, H.M. (1969), A profile refinement method for nuclear and magnetic structures. *J. Appl. Cryst.*, 2: 65-71. <https://doi.org/10.1107/S0021889869006558>

FullProf : Rietveld, Profile Matching & Integrated Intensities Refinement of X-ray and/or Neutron Data (powder and/or single-crystal). Link: <https://www.ill.eu/sites/fullprof/>

Flores-Cano, D. A., Chino-Quispe, A. R., Rueda Vellasmin, R., Ocampo-Anticona, J. A., González, J. C., & Ramos-Guivar, J. A. (2021). Fifty years of Rietveld refinement: Methodology and guidelines in superconductors and functional magnetic nanoadsorbents. *Revista De Investigación De Física*, 24(3), 39–48. <https://doi.org/10.15381/rif.v24i3.21028>

Rietveld Refinement for Macromolecular Powder Diffraction Maria Spiliopoulou, Dimitris-Panagiotis Triandafillidis, Alexandros Valmas, Christos Kosinas, Andrew N. Fitch, Robert B. Von Dreele, and Irene Margiolaki *Crystal Growth & Design* 2020 20 (12), 8101-8123 DOI: [10.1021/acs.cgd.0c00939](https://doi.org/10.1021/acs.cgd.0c00939)

The Rietveld Refinement Method: Half of a Century Anniversary Tomče Runčevski and Craig M. Brown *Crystal Growth & Design* 2021 21 (9), 4821-4822 DOI: [10.1021/acs.cgd.1c00854](https://doi.org/10.1021/acs.cgd.1c00854)

Diffraction Line Profiles in the Rietveld Method Paolo Scardi *Crystal Growth & Design* 2020 20 (10), 6903-6916 DOI: [10.1021/acs.cgd.0c00956](https://doi.org/10.1021/acs.cgd.0c00956)

PYTHON MODULE INDEX

p

powerxrd, [13](#)

Symbols

`__init__()` (*powerxrd.Chart method*), 11
`__init__()` (*powerxrd.Data method*), 10

A

`allpeaks()` (*powerxrd.Chart method*), 11
`allpeaks_recur()` (*powerxrd.Chart method*), 11

B

`backsub()` (*powerxrd.Chart method*), 11
`braggs` (*class in powerxrd*), 10

C

`Chart` (*class in powerxrd*), 11

D

`Data` (*class in powerxrd*), 10

E

`emission_lines()` (*powerxrd.Chart method*), 12

F

`funcgauss` (*class in powerxrd*), 10

G

`gaussfit()` (*powerxrd.Chart method*), 12

L

`local_max()` (*powerxrd.Chart method*), 12

M

`mav()` (*powerxrd.Chart method*), 12
`module`
 `powerxrd`, 13

P

`powerxrd`
 `module`, 13

R

`refine()` (*powerxrd.Rietveld method*), 1

`Rietveld` (*class in powerxrd*), 1

S

`scherrer` (*class in powerxrd*), 10
`SchPeak()` (*powerxrd.Chart method*), 11

X

`XRD_int_ratio()` (*powerxrd.Chart method*), 11