jldesmear Documentation

Release 2015.0623.1

Pete R Jemian

Sep 27, 2017

Contents

Desmear 1-D SAXS or SANS data according to the method of JA Lake as implemented by Pete Jemian.

version 2015.0623.1

date Sep 27, 2017

Reference Citation

Use this reference to cite this code in a publication.

PhD Thesis, 1990, Northwestern University, Evanston, IL, USA, Pete R. Jemian. http://jemian.org/pjthesis.pdf

Contents:

CHAPTER 1

Contents

Overview

Desmear 1-D SAXS or SANS data according to the method of JA Lake as implemented by Pete Jemian.

This program applies the iterative desmearing technique of Lake to small-angle scattering data. The way that the program works is that the user selects a file of data *(x,y,dy)* to be desmeared. If a file was not chosen, the program will end. Otherwise the user is then asked to specify the slit-length (in the units of the x-axis); the *x* at which to begin fitting the last data points to a power-law of *x*, the output file name, and the number of iterations to be run. Then the data file is opened, the data is read, and the data file is closed. The program begins iterating and shows an indicator of progress on the screen in text format.

It is important to only provide *smeared* data (data that has not been desmeared, even partially) to this program as you will see. The iterative desmearing technique should be made to iterate with the original, smeared data and subsequent trial solutions of desmeared data.

The integration technique used by this program to smear the data is the trapezoid-rule where the step-size is chosen by the spacing of the data points themselves. A linear interpolation of the data is performed. To avoid truncation effects, a power-law extrapolation of the intensity is made for all values beyond the range of available data. This region is also integrated by the trapezoid rule. The integration covers the region from `l = 0` up to `l = lo`. (see routine `jldesmear.api.smear`). This technique allows the slit-length weighting function to be changed without regard to the limits of integration coded into this program.

Input data format

Input data will be provided in an ASCII TEXT file as three columns *(Q, I, dI)* separated by white space. Units must be compatible. (*I* and *dI* must have same units)

Q scattering vector (any units)

I measured SAS intensity

dI estimated uncertainties of *I* (usually standard deviation). Note that *dI* **MUST** be provided and **MUST** not be zero.

Command-line program for Jemian/Lake desmearing

jldesmear.jl_api.traditional is the main program to run desmearing. It provides the same command-line interface as its FORTRAN and C predecessors. The main command-line interface is started with a Python command such as:

```
import jldesmear.api
jldesmear.jl_api.traditional.command_line_interface()
```

traditional user interface documentation

Iterative desmearing technique of Lake to small-angle scattering data

Credits

author Pete R. Jemian

organization Late-Nite(tm) Software

note lake.py was derived from lake.c in 2009

note lake.c was derived from the FORTRAN program Lake.FOR

note Lake.FOR 25 May 1991

see P.R.Jemian,; Ph.D. thesis, Northwestern University (1990).

see J.A. Lake; ACTA CRYST 23 (1967) 191-194.

Overview

This program applies the iterative desmearing technique of Lake to small-angle scattering data. The way that the program works is that the user selects a file of data *(x,y,dy)* to be desmeared. If a file was not chosen, the program will end. Otherwise the user is then asked to specify the slit-length (in the units of the x-axis); the *x* at which to begin fitting the last data points to a power-law of *x*, the output file name, and the number of iterations to be run. Then the data file is opened, the data is read, and the data file is closed. The program begins iterating and shows an indicator of progress on the screen in text format.

caution It is important to only provide *smeared* data (data that has not been desmeared, even partially) to this program as you will see. The iterative desmearing technique should be made to iterate with the original, smeared data and subsequent trial solutions of desmeared data.

The integration technique used by this program to smear the data is the trapezoid-rule where the step-size is chosen by the spacing of the data points themselves. A linear interpolation of the data is performed. To avoid truncation effects, a power-law extrapolation of the intensity is made for all values beyond the range of available data. This region is also integrated by the trapezoid rule. The integration covers the region from `l = 0` up to `l = lo`. (see module `smear`). This technique allows the slit-length weighting function to be changed without regard to the limits of integration coded into this program.

Other deconvolution methods

These authors have presented desmearing/deconvolution methods that were considered or reviewed in the development of this work.

- O. Glatter. ACTA CRYST 7 (1974) 147-153.

- W.E. Blass & G.W.Halsey. (1981) "Deconvolution of Absorption Spectra." New York City: Academic Press
- P.A. Jansson. (1984) "Deconvolution with Applications in Spectroscopy." New York City: Academic Press.
- G.W.Halsey & W.E. Blass. (1984) "Deconvolution Examples" in "Deconvolution with Applications in Spectroscopy." Ed. P.A. Jansson. (see above)

Source Code Documentation

`jldesmear.jl_api.traditional.`**`GetInf`**(*params*)
Get information about the desmearing parameters. This is designed to be independent of wavelength or radiation-type (i.e. neutrons, X rays, etc.)

Parameters **`params`** (*obj*) – Desmearing parameters object

Returns params or None

`jldesmear.jl_api.traditional.`**`callback`**(*dsm*)
this function is called after every desmearing iteration from `desmear.Desmearing.traditional()`

Parameters **`dsm`** (*obj*) – Desmearing object

Returns should desmearing stop?

Return type bool

`jldesmear.jl_api.traditional.`**`command_line_interface`**()
SAS data desmearing, by Pete R. Jemian Based on the iterative technique of PR Jemian and JA Lake. P.R.Jemian,; Ph.D. thesis, Northwestern University (1990). J.A. Lake; ACTA CRYST 23 (1967) 191-194.

```
$Id$
Desmear using the same command line interface as the FORTRAN & C predecessors.
```

`jldesmear.jl_api.traditional.`**`no_plotting_callback`**(*dsm*)
this function is called after every desmearing iteration from `desmear.Desmearing.traditional()`

Parameters **`dsm`** (*obj*) – Desmearing object

Returns should desmearing stop?

Return type bool

`jldesmear.jl_api.traditional.`**`plot_results`**(*q*, *E*, *C*)
plot the results of the desmearing

Parameters

- **`q`** (*numpy.ndarray*) – magnitude of scattering vector
- **`E`** (*numpy.ndarray*) – experimental (smeared) data
- **`C`** (*numpy.ndarray*) – corrected (desmeared) data

Graphical User Interface for Lake/Jemian desmearing

Several classes are defined in the source code. This class is used to start the GUI: `jldesmear.jl_api.gui.DesmearingGui`. The main GUI program is started with a Python command such as:

```
from jldesmear.jl_api import gui
gui.main()
```

`gui` graphical user interface documentation

Example using `test1.smr` data set

Input Commands

Start the program from the `data` directory in the source tree. We'll use UNIX shell redirection to get everything in a text file:

```
cd src/jldesmear/data
python ../api/traditional.py < test1.inp > test1.out
```

The program will print a header:

```
<<<     SAS data desmearing, by Pete R. Jemian
<<<     Based on the iterative technique of JA Lake and PR Jemian.
<<<     P.R.Jemian,; Ph.D. thesis, Northwestern University (1990).
<<<     J.A. Lake; ACTA CRYST 23 (1967) 191-194.
<<<
<<<     $Id$
<<<     desmear using the same FORTRAN & C command line interface
<<<
```

Then, the program will ask some questions about the input data. Here, the test data is `test1.smr`:

```
<<< What is the input data file name? <''=Quit> <> ==>
>>> test1.smr
```

Name the (new) file name to write the results. If it exists, it will be overwritten without further comment. Here, we choose the name `test1.out`:

```
<<< What is the output data file name? <> ==>
>>> test1.out
```

The slit length is the term *l_o* and has the same units as *X*:

```
<<< What is the slit length (x-axis units)? <1.0> ==>
>>> .08
```

To complete the smearing integral at highest *X*, it is necessary to extrapolate beyond the range of measured data. Choose the functional form that best represents the data at highest *X*. Fit coefficients will be evaluated for each desmearing iteration over the range `X_start <= X <= X_max`:

```
<<< Extrapolation forms to avoid truncation-error.
<<<    constant = flat background, I(q) = B
<<<    linear = linear, I(q) = b + q * m
<<<    powerlaw = power law, I(q) = b * q^m
<<<    Porod = Porod law, I(q) = Cp + Bkg / q^4
<<<
```

Choose the *linear* form (although *constant* would work with this data as well):

```
<<< Which form? <constant> ==>
>>> linear
```

This is `X-start` as noted above: `.08`:

```
<<< What X to begin evaluating extrapolation (x-axis units)? <1.0> ==>
>>> .08
```

Accept the solution after 20 iterations this time:

```
<<< How many iteration(s)? (10000 = infinite) <10000> ==>
>>> 20
```

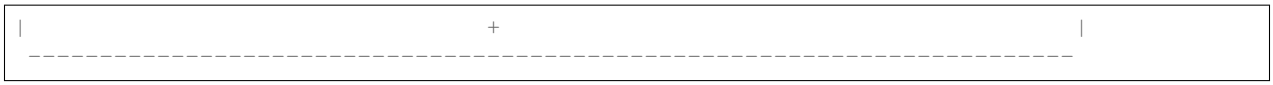
This question is largely historical. The `fast` method is **always** the best choice. The others were implementations of either Jansson or Halsey & Blass. They converge more slowly by far. That said, you are free to re-determine this for yourself. Press the [return] key to accept the default suggestion:

```
<<<  Weighting methods for iterative corrections:
<<<  Correction = weight * (MeasuredI - SmearedI)
<<<    constant: weight = 1.0
<<<    fast: weight = CorrectedI / SmearedI
<<<    ChiSqr: weight = 2*SQRT(ChiSqr(0) / ChiSqr(i))
<<<
<<< Which method? <fast> ==>
>>>
```

Program output to console

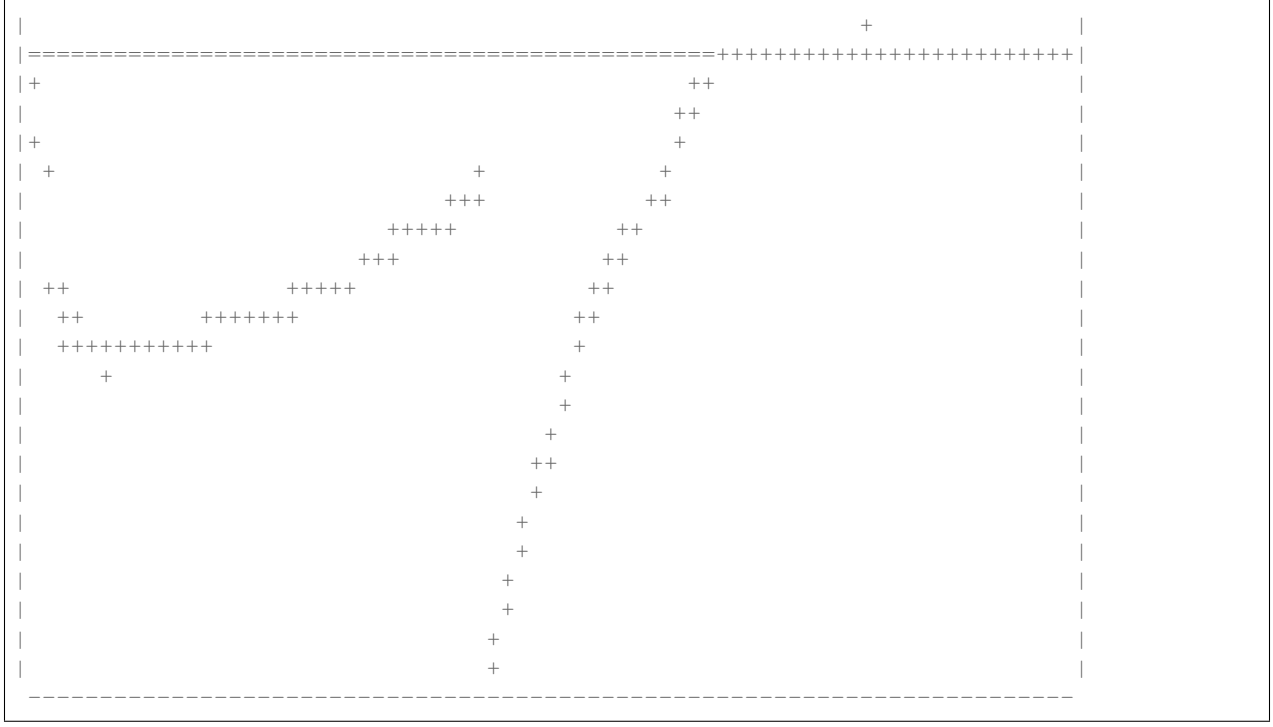
Now the program starts the work of desmearing. The first step shows an awful chi-square statistic. This will improve with subsequent iterations. The plot is standardized residual vs. data point number. There are `==========` bars indicated at `+1` and `-1`; these merge together on the first plot.:

```
Input file: test1.smr
-/|\ ...
standardized residuals, ChiSqr = 1.29823e+07, iteration=0
x: min=1   step=3.45833   max=250
y: min=-545.836   step=24.8717   max=1.34169
 ------------------------------------------------------------------------
|                                                                  +     |
|==================================================++++++++++++++++++++++|
|+                                                  ++                   |
|                                                  ++                    |
|                                                 ++                     |
|+                                 +         ++                          |
|++                                +        ++                           |
|                                 +        ++                            |
|                                +++       ++                            |
|                               +++         +                            |
|                             ++            +                            |
|                            ++            ++                            |
|                          +               +                             |
| ++                   +++                 ++                            |
|  ++            +++++                     +                             |
|   ++++++++++++                           +                             |
|                                         +                              |
|                                         +                              |
|                                         +                              |
|                                        +                               |
|                                        +                               |
|                                        +                               |
```



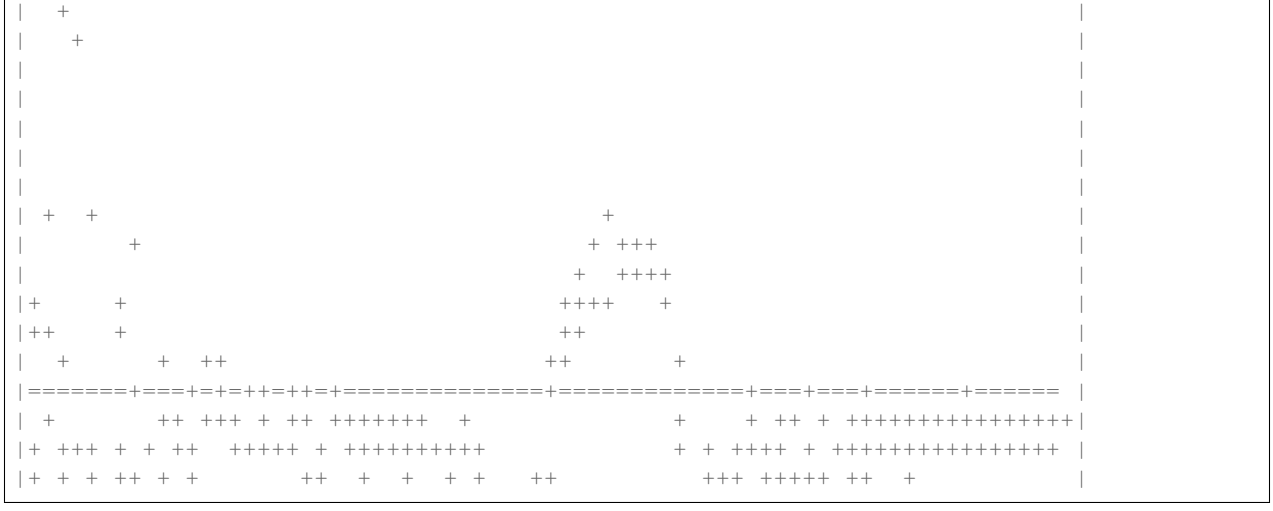
After the next iteration, the chi-squared statistic has improved by an order of magnitude but the plot still does not different:

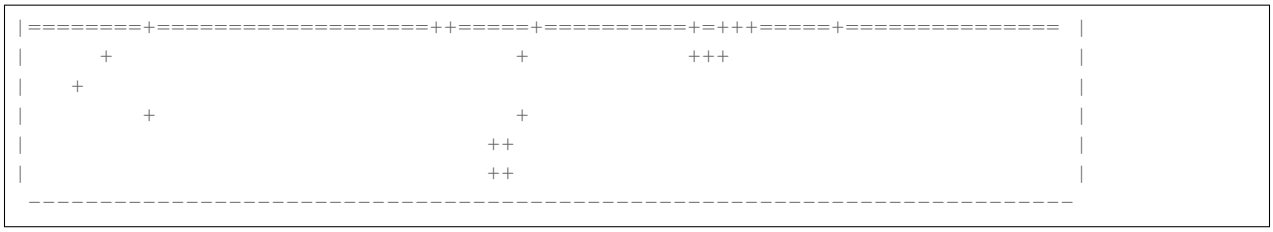
```
standardized residuals, ChiSqr = 1.36804e+06, iteration=1
x: min=1    step=3.45833    max=250
y: min=-206.354    step=9.44611    max=1.46073
```



Skipping forward a few iterations, we see some real progress:

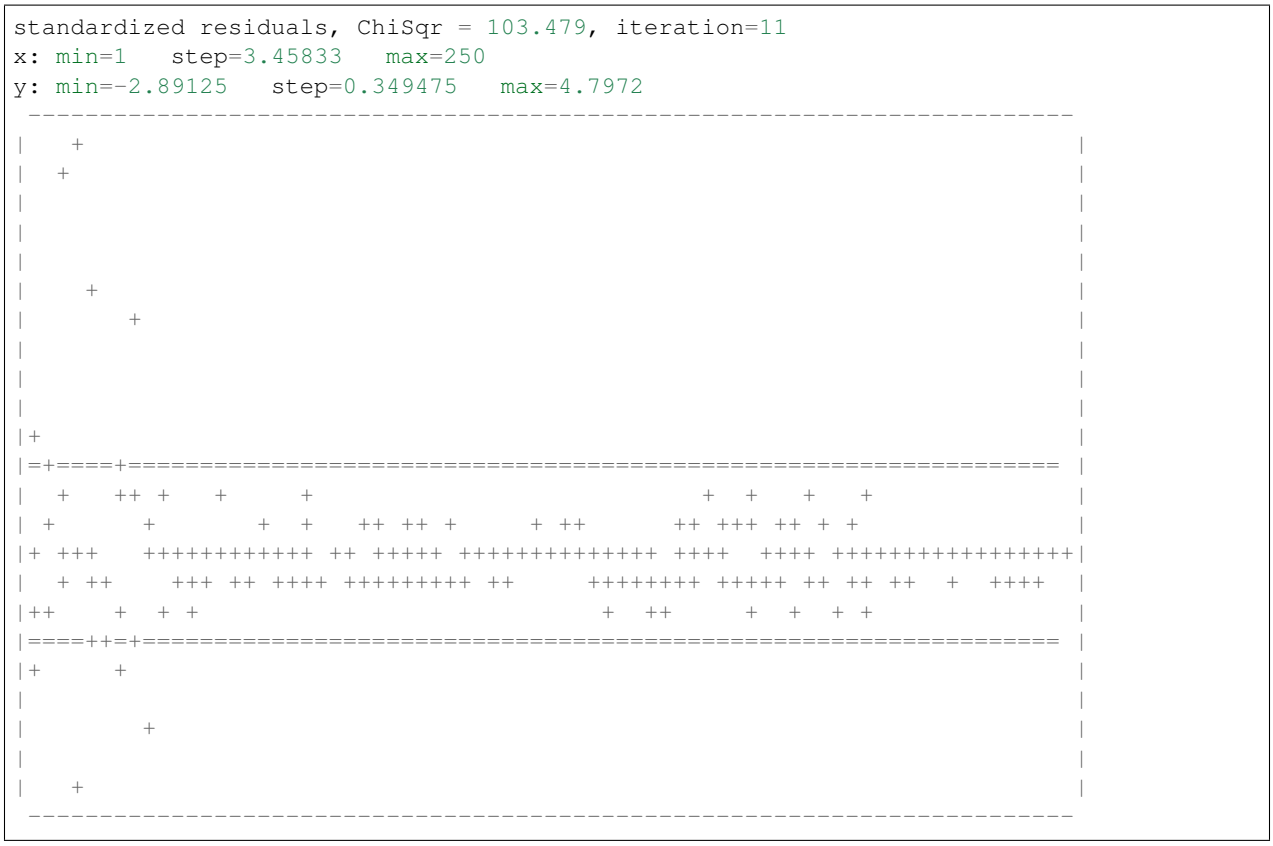
```
standardized residuals, ChiSqr = 566.385, iteration=5
x: min=1    step=3.45833    max=250
y: min=-3.97891    step=0.499962    max=7.02024
```





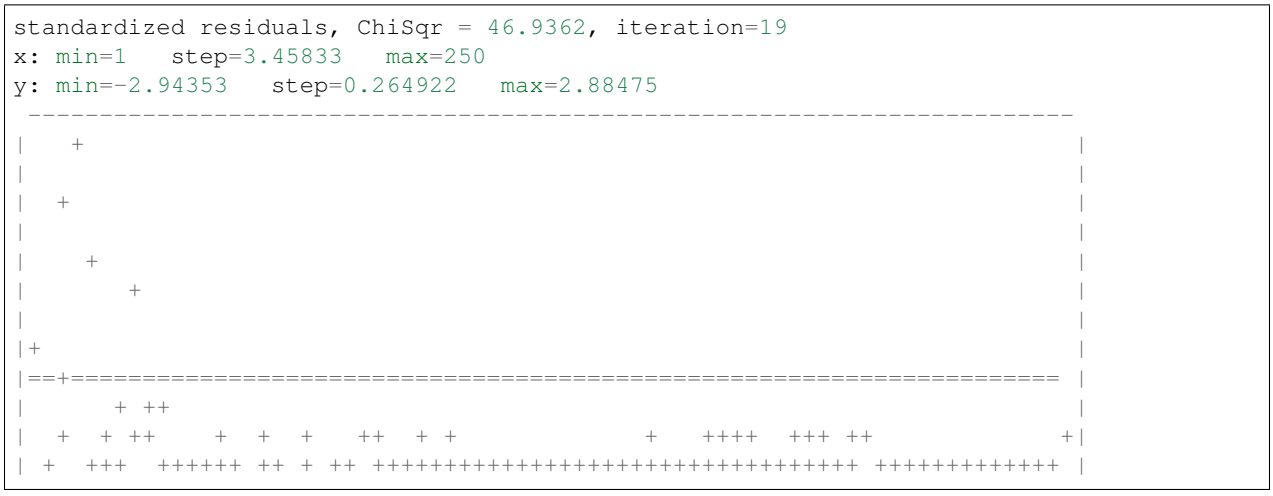
After about 10 iterations or so, it seems convergence has been achieved. The chi-squared statistic has dropped and the plot looks more randomly-arranged about 0.:

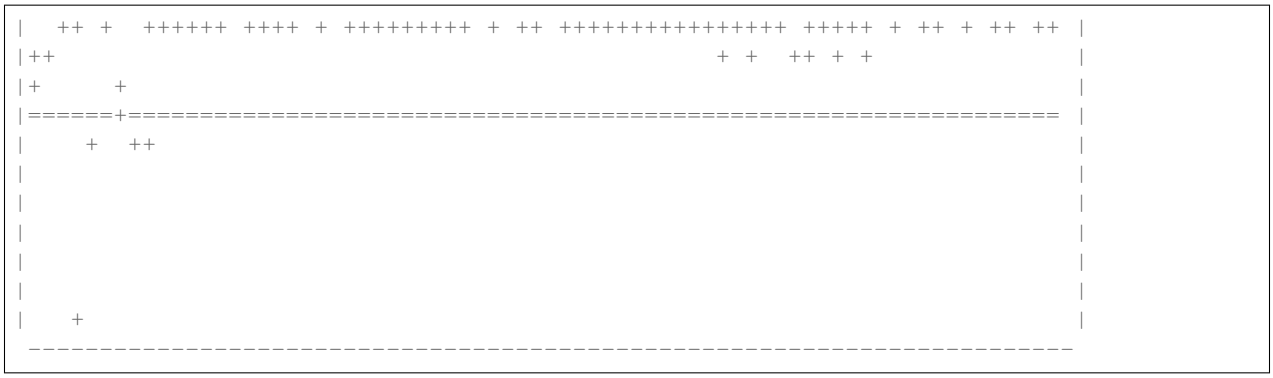
```
standardized residuals, ChiSqr = 103.479, iteration=11
x: min=1   step=3.45833   max=250
y: min=-2.89125   step=0.349475   max=4.7972
```



Finally, after 20 iterations (numbered 0 .. 19):

```
standardized residuals, ChiSqr = 46.9362, iteration=19
x: min=1   step=3.45833   max=250
y: min=-2.94353   step=0.264922   max=2.88475
```





The result is accepted and the data are saved to the output file:

Data Files

Command Input File (`test1.inp`)

```
test1.smr
test1.dsm
0.08
linear
0.08
20
```

```
fast
```

Input Data File (`test1.smr`)

Too big for the documentation. `test1.smr`

Output Data File (`test1.dsm`)

Too big for the documentation. `test1.dsm`

Complete Program Output (`test1.out`)

Too big for the documentation. `test1.out`

API: application programmer interface

Statistics Registers

Implement a set of statistics registers in the style of a pocket calculator.

The available routines are:

```
def  Clear():                       clear the stats registers
def  Show():                        print the contents of the stats registers
def  Add(x, y):                     add an X,Y pair
def  Subtract(x, y):                remove an X,Y pair
def  AddWeighted(x, y, z):          add an X,Y pair with weight Z
def  SubtractWeighted(x, y, z):     remove an X,Y pair with weight Z
def  Mean():                        arithmetic mean of X & Y
def  StdDev():                      standard deviation on X & Y
def  StdErr():                      standard error on X & Y
def  LinearRegression():            linear regression
def  LinearRegressionVariance():    est. errors of slope & intercept
def  LinearRegressionCorrelation(): the regression coefficient
def  CorrelationCoefficient():      relation of errors in slope & intercept
```

see http://stattrek.com/AP-Statistics-1/Regression.aspx?Tutorial=Stat

pocket calculator Statistical Registers, Pete Jemian, 2003-Apr-18

Source code documentation

class jldesmear.jl_api.StatsReg.**StatsRegClass**
pocket calculator Statistical Registers class

Add(*x*, *y*)
add an X,Y pair to the statistics registers

Parameters

- **x** (*float*) – value to accumulate

- **y** (*float*) – value to accumulate

AddWeighted(*x*, *y*, *z*)
add a weighted X,Y+/Z trio to the statistics registers

Parameters

- **x** (*float*) – value to accumulate
- **y** (*float*) – value to accumulate
- **z** (*float*) – variance (weight = `1/z^2`) of y

Clear()
clear the statistics registers: $N = w = \sum x = \sum x^2 = \sum y = \sum y^2 = \sum xy = 0$

CorrelationCoefficient()
relation of errors in slope and intercept

Returns correlation coefficient

Return type float

LinearEval(*x*)
Evaluate a linear fit at the given value: $y = a + bx$

Parameters **x** (*float*) – independent value, *x*

Returns y

Return type float

LinearRegression()
For (*x,y*) data pairs added to the registers, fit and find (*a,b*) that satisfy:

$$y = a + bx$$

Returns (a, b) for fit of y=a+b*x

Return type (float, float)

LinearRegressionCorrelation()
the regression coefficient

Returns (corr_a, corr_b) – is this correct?

Return type (float, float)

See http://stattrek.com/AP-Statistics-1/Correlation.aspx?Tutorial=Stat Look at "Product-moment correlation coefficient"

LinearRegressionVariance()
est. errors of slope & intercept

Returns (var_a, var_b) – is this correct?

Return type (float, float)

Mean()
arithmetic mean of X & Y

$$(1/N)\sum_{i}^{N} x_i$$

Returns mean X and Y values

Return type float

Show()
contents of the statistics registers

StdDev()
standard deviation on X & Y

Returns standard deviation of mean X and Y values

Return type (float, float)

StdErr()
standard error on X & Y

Returns standard error of mean X and Y values

Return type (float, float)

Subtract(*x*, *y*)
remove an X,Y pair from the statistics registers

Parameters

- **x** (*float*) – value to remove
- **y** (*float*) – value to remove

SubtractWeighted(*x*, *y*, *z*)
remove a weighted X,Y+/-Z trio from the statistics registers

Parameters

- **x** (*float*) – value to remove
- **y** (*float*) – value to remove
- **z** (*float*) – variance (weight = `1/z^2`) of y

determinant()
compute and return the determinant of the square matrices:

```
|  sum_w   sum_x      |          |  sum_w   sum_y      |
|  sum_x   sum_(x^2)  |          |  sum_y   sum_(y^2)  |
```

Returns determinants of x and y summation matrices

Return type (float, float)

Desmearing

Desmear the 1-D SAS data *(q, I, dI)* by method of Lake & Jemian. Desmear SAS data

To desmear, apply the method of Jemian/Lake to 1-D SAS data *(q, I, dI)*.

Source Code Documentation

class jldesmear.jl_api.desmear.**Desmearing**(*q*, *I*, *dI*, *params*)
desmear the 1-D SAS data *(q, I, dI)* by method of Jemian/Lake

$$I_0 \approx \lim_{i \to \infty} I_{i+1} = I_i \times \left(\tilde{I}_0 \div \tilde{I}_i \right)$$

To start Lake's method, assume that the 0-th approximation of the corrected intensity is the measured intensity.

Parameters

- **q** (*numpy.ndarray*) – magnitude of scattering vector
- **I** (*numpy.ndarray*) – SAS data I(q) +/- dI(q)
- **dI** (*numpy.ndarray*) – estimated uncertainties of I(q)
- **params** (*obj*) – Info object with desmearing parameters

---

Note: This equation shows the iterative feedback based on the *fast* method (as described by Lake). Alternative feedback methods are available (see *SetLakeWeighting()*). It is suggested to **always** use the fast method.

SetExtrap(*extrapolation_object=None*)

Parameters **extrapolation_object** (*obj*) – class used for extrapolation function

SetLakeWeighting(*LakeWeighting='fast'*)

Parameters **LakeWeighting** (*str*) – one of *constant*, *ChiSqr*, or *fast*

Constant weight = 1.0

ChiSqr weight = CorrectedI / SmearedI

Fast weight = 2*SQRT(ChiSqr(0) / ChiSqr(i))

SetQuiet(*suppress_output=True*)
if True, then no printed output from this routine

first_step()
the first step

calculate the standardized residuals ($z =$ `self.z`)

$$z = (\hat{y} - y)\sigma$$

where `y = S`, `yHat = I`, and `sigma = dI`

calculate the chi-squared statistic ($\chi^2 =$ `self.ChiSqr`)

$$\chi^2 = \sum z^2$$

iterate_and_callback()
Compute one iteration of the Lake algorithm and then call the supplied callback method. Use this method to run a desmearing operation in another thread.

iteration()
Compute one iteration of the Lake algorithm.

No need to call the callback routine, the caller can take care of that directly.

traditional()
the traditional LAKE code algorithm

This method is called from the class constructor. If this method is called directly, it has the effect of clearing any desmearing progress and resetting back to start. This technique is used here if the list of ChiSqr results is not empty.

Extrapolations at highest *q*

Extrapolation: Constant

Extrapolate as: I(q) = B

class jldesmear.jl_api.extrap_constant.**Extrapolation**
I(q) = B

calc(*q*)

$$I(q) = B$$

Parameters **q** (*float*) – magnitude of scattering vector

Returns value of extrapolation function at q

Return type float

fit_result(*reg*)
Determine the results of the fit and store them as the set of coefficients in the self.coefficients dictionary. Called from fit().

Note *must* override in subclass otherwise *fit_result()* will throw an exception

Parameters **reg** (*StatsRegClass object*) – statistics registers (created in fit())

Extrapolation: Linear

Extrapolate as: I(q) = B + m * q

class jldesmear.jl_api.extrap_linear.**Extrapolation**
I(q) = B + m*q

calc(*q*)

$$I(q) = B + mq$$

Parameters **q** (*float*) – magnitude of scattering vector

Returns value of extrapolation function at q

Return type float

fit_result(*reg*)
Determine the results of the fit and store them as the set of coefficients in the self.coefficients dictionary. Called from fit().

Note *must* override in subclass otherwise *fit_result()* will throw an exception

Parameters **reg** (*StatsRegClass object*) – statistics registers (created in fit())

Extrapolation: Porod Law

Extrapolate as: I(q) = B + Cp / q^4

class jldesmear.jl_api.extrap_Porod.**Extrapolation**
I(q) = B + Cp / q^4

calc(*q*)

$$I(q) = B + C_p/q^4$$

Parameters **q** (*float*) – magnitude of scattering vector

Returns value of extrapolation function at q

Return type float

fit_add(*reg*, *x*, *y*, *z*)
Add a data point to the statistics registers. Called from `fit_loop()`.

Note *might* override in subclass

Parameters

- **reg** (*obj*) – statistics registers (created in `fit()`), instance of StatsRegClass
- **x** (*float*) – independent axis
- **y** (*float*) – dependent axis
- **z** (*float*) – estimated uncertainty of y

fit_result(*reg*)
Determine the results of the fit and store them as the set of coefficients in the self.coefficients dictionary. Called from `fit()`.

Note *must* override in subclass otherwise *fit_result()* will throw an exception

Parameters **reg** (*obj*) – statistics registers (created in `fit()`), instance of StatsRegClass

Extrapolation: Power Law

Extrapolate as: I(q) = A * q^p

class jldesmear.jl_api.extrap_powerlaw.**Extrapolation**
I(q) = A * q^p

calc(*q*)

$$I(q) = A\, q^p$$

Parameters **q** (*float*) – magnitude of scattering vector

Returns value of extrapolation function at q

Return type float

fit_add(*reg*, *x*, *y*, *z*)
Add a data point to the statistics registers. Called from `fit_loop()`.

Note *might* override in subclass

Parameters

- **reg** (*StatsRegClass object*) – statistics registers (created in `fit()`)
- **x** (*float*) – independent axis
- **y** (*float*) – dependent axis
- **z** (*float*) – estimated uncertainty of y

fit_result(*reg*)
Determine the results of the fit and store them as the set of coefficients in the self.coefficients dictionary. Called from `fit()`.

Note *must* override in subclass otherwise *`fit_result()`* will throw an exception

Parameters **reg** (*`StatsRegClass object`*) – statistics registers (created in `fit()`)

superclass of functions for extrapolation of SAS data past available range

class `jldesmear.jl_api.extrapolation.`**Extrapolation**
superclass of functions for extrapolation of SAS data past available range

The general case to (forward) slit-smear small-angle scattering involves integration at q values past any measurable range.

$$\int_{-\infty}^{\infty} P_l(q_l) I(q, q_l) dq_l.$$

Due to symmetry, the integral is usually folded around zero,thus becoming

$$2\int_{0}^{\infty} P_l(q_l) I(q, q_l) dq_l.$$

Even when the upper limit is reduced due to finite slit dimension (the so-called slit-length, l_0),

$$2l_0^{-1}\int_{0}^{0} I(\sqrt{q^2 + q_l^2}) dq_l,$$

it is still necessary to evaluate $I(\sqrt{q^2 + q_l^2})$ beyond the last measured data point, just to evaluate the integral.

An extrapolation function is used to describe the $I(q)$ beyond the measured data. In the most trivial case, zero would be returned. Since this simplification is known to introduce truncation errors, a model form for the last few available data points is assumed. Fitting coefficients are determined from the final data points (in the method *`fit()`*) and are used subsequently to generate the extrapolation at a specific q value (in the method *`calc()`*).

Examples:

See the subclasses for examples implementations of extrapolations.

- `extrap_constant`
- `extrap_linear`
- `extrap_powerlaw`
- `extrap_Porod`

Example Linear Extrapolation:

Here is an example linear extrapolation class:

```
import extrapolation

class Extrapolation(extrapolation.Extrapolation):
    name = 'linear'         # unique identifier for users

    def __init__(self):     # initialize whatever is needed internally
        self.coefficients = {'B': 0, 'm': 0}
```

```
    def __str__(self):
        form = "linear: I(q) = " + str(self.coefficients['B'])
        form += " + q*(" + str(self.coefficients['m']) + ")"
        return form

    def calc(self, q):    # evaluate at given q
        return self.coefficients['B'] + self.coefficients['m'] * q

    def fit_result(self, reg):    # evaluate fitting parameters with regression␣
↪object
        (constant, slope) = reg.LinearRegression()
        self.coefficients = dict(B=constant, m=slope)
```

Basics:

Create an Extrapolation class which is a subclass of `extrapolation.Extrapolation`.

The basic methods to override are

- `__str__()` : string representation
- *`calc()`* : determines $I(q)$ from `q` and `self.coefficients` dictionary
- *`fit_result()`* : assigns fit coefficients to `self.coefficients` dictionary

By default, the base class Extrapolation uses the `jldesmear.api.StatsReg` module to accumulate data and evaluate fitted parameters. Override any or all of these methods to define your own handling:

- *`fit()`*
- *`fit_setup()`*
- *`fit_loop()`*
- *`fit_add()`*
- *`fit_result()`*
- *`calc()`*
- *`show()`*
- *`format_coefficient()`*

See the source code of `Extrapolation` for an example.

documentation from source code:

GetCoefficients()
return the function coefficients

SetCoefficients(*coefficients*)
define the function coefficients

Parameters **coefficients** (*dict*) – named terms used in evaluating the extrapolation

calc(*q*)
evaluate the extrapolation function at the given q

Note **must** override in subclass

Parameters **q** (*float*) – magnitude of scattering vector

Returns value of extrapolation function at q

Return type float

fit (q, I, dI)
fit the function coefficients to the data

Note *might* override in subclass

Parameters

- **q** (*float*) – magnitude of scattering vector
- **I** (*float*) – intensity or cross-section
- **dI** (*float*) – estimated uncertainty of intensity or cross-section

fit_add (*reg*, x, y, z)
Add a data point to the statistics registers. Called from *fit_loop()*.

Note *might* override in subclass

Parameters

- **reg** (*StatsRegClass object*) – statistics registers (created in *fit()*)
- **x** (*float*) – independent axis
- **y** (*float*) – dependent axis
- **z** (*float*) – estimated uncertainty of y

fit_loop (*reg*, x, y, z)
Add a dataset to the statistics registers for use in curve fitting. Called from *fit()*.

Note *might* override in subclass

Parameters

- **reg** (*StatsRegClass object*) – statistics registers (created in fit())
- **x** (*numpy.ndarray*) – independent axis
- **y** (*numpy.ndarray*) – dependent axis
- **z** (*numpy.ndarray*) – estimated uncertainties of y

fit_result (*reg*)
Determine the results of the fit and store them as the set of coefficients in the self.coefficients dictionary. Called from *fit()*.

Example:

```
def fit_result(self, reg):
    (constant, slope) = reg.LinearRegression()
    self.coefficients['B']  = constant
    self.coefficients['m']  = slope
```

Note *must* override in subclass otherwise *fit_result()* will raise an exception

Parameters **reg** (*StatsRegClass object*) – statistics registers (created in *fit()*)

fit_setup ()
Create a set of statistics registers to evaluate the coefficients of the curve fit. Called from *fit()*.

Note *might* override in subclass

Returns statistics registers

Return type StatsRegClass object

format_coefficient(*key*, *value*)
Format a specific coefficient. Called from *show()*.

Note *might* override in subclass

Parameters

- **key** (*str*) – name of coefficient (must exist in self.coefficients dictionary)
- **value** (*usually float*) – usually value of self.coefficients[key]

show()
print the function and fit coefficients

Note *might* override in subclass

jldesmear.jl_api.extrapolation.**discover_extrapolations**()
return a dictionary of the available extrapolations

Extrapolation functions must be in a file named `extrap_KEY.py` where `KEY` is the key name of the extrapolation function. The file is placed in the source code tree in the same directory as the module: `extrapolation`.

The `calc()` method should be capable of handling `q` as a `numpy.ndarray` or as a `float`.

The file must contain:

•*Extrapolation*: a subclass of `Extrapolation`

`fileio` documentation

superclass of modules supporting different file formats

class jldesmear.jl_api.fileio.**FileIO**
superclass of file format support

jldesmear.jl_api.fileio.**discover_support**()
return a dictionary of the available file formats

Support modules must be in a file in the **jl_api** directory package in the source tree and begin with the prefix `fileio_`.

`fileio_inp` documentation

fileio support for .inp file: traditional command-line input format

class jldesmear.jl_api.fileio_inp.**CommandInput**
command input file format

This file format was created to pipe the inputs directly to the interactive command-line FORTRAN program. There were two benefits:

1.desmearing parameters were documented in a file

2.the answer to each question was automatically provided

contents:

```
SMR_filename        (absolute path or relative to directory of .inp file)
DSM_filename
slitlength
extrapolation_method
sFinal
number_iterations
feedback_method
```

The file names (SMR and DSM) are given as either absolute or relative to the directory of the *.inp* file. The data are stored in three-column ASCII, with whitespace separators with the columns *Q I dI*. Individual data points may be commented out by placing a # character at the start of that line of text. This format is known to some as *QRS*.

The *slitlength* and *sFinal* are given as floating point numbers in the same units as q. It is expected that `sFinal < qMax` by at least a few data points.

Example `test1.inp` file:

```
test1.smr
test1.dsm
0.08
linear
0.08
20
fast
```

read (*filename*)
read desmearing parameters from a command input file

Parameters **filename** (*str*) – full path to the command input file

Returns instance of `jldesmear.api.info.Info`

read_SMR (*filename=None*)
Open a file with 3-column smeared SAS data

save (*filename*)
write desmearing parameters to a command input file

save_DSM (*filename*, *dsm*)
Save the desmeared data to a 3-column ASCII file

Desmearing parameters, the `Info` object

desmearing parameters:

```
infile = ""                     # input data file
outfile = ""                    # output data file
slitlength = 1.0                # slit length (l_o) as defined by Lake
sFinal = 1.0                    # fit extrapolation constants for q>=sFinal
NumItr = INFINITE_ITERATIONS    # number of desmearing iterations
extrapname = "constant"         # model final data as a constant
LakeWeighting = "fast"          # shows the fastest convergence most times
extrap = None                   # extrapolation function object
quiet = False                   # suppress output from desmearing operations
callback = None                 # function object to call after each desmearing␣
↪iteration
```

class `jldesmear.jl_api.info.`**`Info`**
parameters used by the desmearing methods

`moreIterationsOk`(*iteration_count*)

Returns is it OK to take more iterations?

Return type bool

Forward Smearing

Some instruments designed to measure small-angle scattering have intrinsic *slit-smearing* of the in their design. One such example is the Bonse-Hart design which uses single crystals to collimate the beam incident on the sample as well as to collimate the scattered beam beam that will reach the detector.

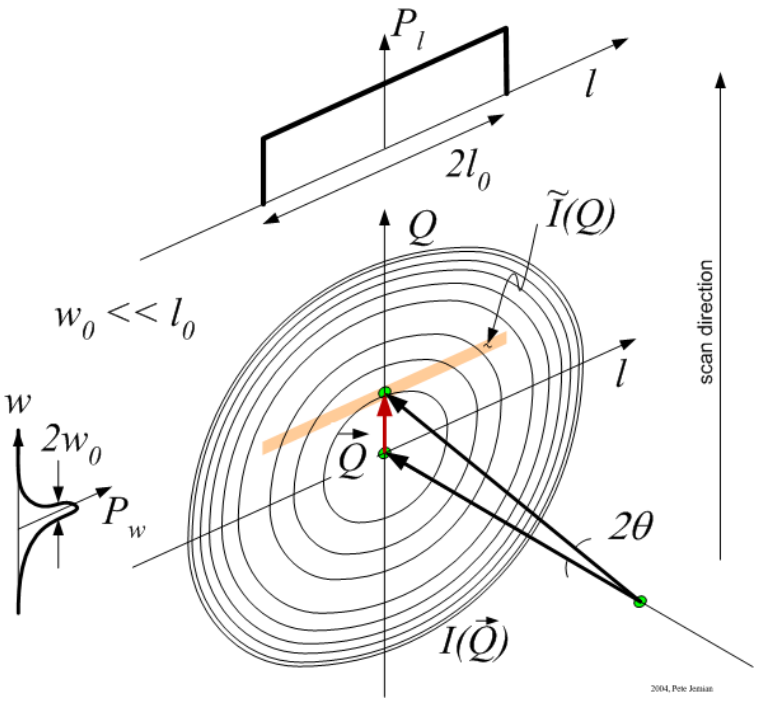


Fig. 1.1: Slit smearing geometry of the Bonse-Hart design.

Source Code Documentation

Forward smearing

Smear *(q, I, dI)* data given to the routine `Smear()` using the slit-length weighting function `Plengt()`. The integration used below goes only over the slit length (does not include either slit width or wavelength broadening).

For now, `Plengt()` describes a rectangular slit and the integration extends up to the length of the slit. This could be changed if desired.

To complete the smearing for the last data points, extrapolation is necessary from the given data. The functional form may be only one of those provided (others could be added).

For q values in between given data points, interpolation is used. Log interpolation is tried first. If this fails due to a ValueError Exception, linear interpolation is used.

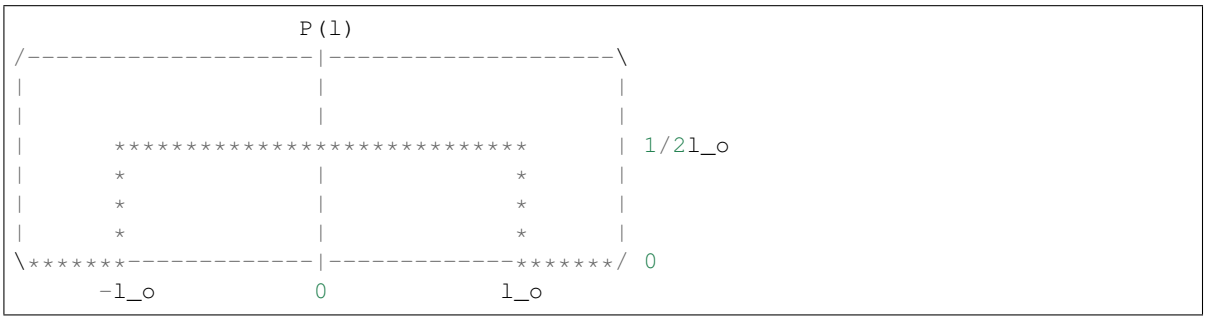
Source Code Documentation

jldesmear.jl_api.smear.**Plengt**(*l*, *slitlength*)

Slit-length weighting function, *P_l(l)*

$$\int_{-\infty}^{\infty} P_l(l)dl = 1$$

It is defined for a rectangular slit of length `2*slitlength` ($2l_o$) and probability `1/(2*slitlength)` ($1/2l_o$). It is zero elsewhere.:



Note integral(P(l) dl) = 1.0

Note If you change this to a different functional form ... It is not necessary to change the limit of the integration if the functional form here is changed. You may, however, need more parameters.

Parameters

- **l** (*float*) – lookup value
- **slitlength** (*float*) – slit length, *l_o,* as indicated above

Returns P_l(l)

Return type float

jldesmear.jl_api.smear.**Smear**(*q*, *C*, *dC*, *extrapname*, *sFinal*, *slitlength*, *quiet=False*, *weighted_transition=True*)

Smear the data of C(q) into S(q) using the slit-length weighting function `Plengt()` and an extrapolation of the data to avoid truncation errors. Assume that `Plengt()` goes to zero for `l > l_o` (the slit length).

Also assume that the slit length function is symmetrical about `l = zero`.

$$S(q) = 2\int_0^{l_o} P_l(l)\, C(\sqrt{q^2 + l^2})\, dl$$

This routine is written so that if `Plengt()` is changed (for example) to a Gaussian, that no further modification is necessary to the integration procedure. That is, this routine will integrate the data out to "slitlength" (`l_o`).

Parameters

- **q** (*numpy.ndarray*) – magnitude of scattering vector
- **C** (*numpy.ndarray*) – unsmeared data is C(q) +/- dC(q)
- **dC** (*numpy.ndarray*) – estimated uncertainties of C

- **extrapname** (*str*) – one of `constant | linear | powerlaw | Porod`
- **sFinal** (*float*) – fit extrapolation to I(q) for q >= sFinal
- **slitlength** (*float*) – l_o, same units as q
- **quiet** (*bool*) – if True, then no printed output from this routine
- **weighted_transition** (*bool*) – if True, make a weighted transition between sFinal <= q < qMax

Returns tuple of (S, extrap)

Return type (numpy.ndarray, object)

Variables **S** (*numpy.ndarray*) – smeared version of C

`jldesmear.jl_api.smear.`**get_Ic**(*qNow*, *sFinal*, *qMax*, *x*, *interp*, *extrap*, *weighted_transition=True*)
return the corrected intensity based on circular symmetry

`jldesmear.jl_api.smear.`**prepare_extrapolation**(*q*, *C*, *dC*, *extrapname*, *sFinal*)
Pick the extrapolation function for smearing

Parameters

- **q** (*numpy.ndarray*) – magnitude of scattering vector
- **C** (*numpy.ndarray*) – array (list) such that data is C(q) +/- dC(q)
- **dC** (*numpy.ndarray*) – estimated uncertainties of C
- **extrapname** (*str*) – one of `constant`, `linear`, `powerlaw`, or `Porod`
- **sFinal** (*float*) – fit extrapolation to I(q) for q >= sFinal

Returns function object of selected extrapolation

Return type object

Plotting on a console

Make charts on a text console using character graphics

Generate graphical output on a text console. These routines predate modern GUI environments. While the output may look rough, they work just about anywhere.

Here is how the code may be called:

```
>>> fn = toolbox.GetTest1DataFilename('.smr')
>>> x, y, dy = toolbox.GetDat(fn)
>>> print("Data plot: " + fn)
>>> Screen().xyplot(x, y)
```

Example, given *C(q)* and *S(q)*:

```
KratkyPlot = textplots.Screen()
title = "\nKratky plot, I * q^2 vs q: S=smeared"
q2C = q*q*(C - B)
q2S = q*q*(S - B)
KratkyPlot.SetTitle(title)
KratkyPlot.addtrace(lnq, q2S, "S")
KratkyPlot.printplot()
title = "\nKratky plot, I * q^2 vs q: C=input, S=smeared"
KratkyPlot.SetTitle(title)
```

```
KratkyPlot.addtrace(lnq, q2C, "C")
KratkyPlot.printplot()
```

with the right data, produces plots of $q^2C(q)$ and $q^2S(q)$:

```
Kratky plot, I * q^2 vs q: C=input, S=smeared
x: min=-7.898   step=0.0889226   max=-1.49558
y: min=-0.107876   step=0.199276   max=4.2762
 ------------------------------------------------------------------------------
|            C                                                                 |
|         CCCCCC                                                               |
|       CCC    CCC                                                             |
|      CC        CC                                                            |
|     CC          CC                                                           |
|    CC            C                                                           |
|    C              C                                                          |
|   C                C                                                         |
|  CC                CC                                                        |
|CC                   C                                                        |
|                      C                                                       |
|C                     CC                                                      |
|                       C                                                      |
|                        C                                                     |
|                         C                                                    |
|                          C                                                   |
|                          CC                                                  |
|                            C                                                 |
|                             CC                                               |
|                               CCC                                            |
|                                  CCCCCC                                CCCC  |
|SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS|
 ------------------------------------------------------------------------------
```

Source code documentation

class jldesmear.jl_api.textplots.**Screen**(*MaxRow=25*, *MaxCol=75*, *Symbol='O'*)
plotting canvas

SetTitle(*title*)
define a plot title

Parameters **title** (*str*) – the title of the plot

addtrace(*x*, *y*, *symbol='O'*)
add the (x,y) trace to the plot with the given symbol

Parameters

- **x** – array (list) of abcissae
- **y** – array (list) of ordinates
- **symbol** – plotting character

make_buffer(*rows=None*, *cols=None*)
prepare a screen buffer

paintbuffer()
plot the traces on the screen buffer data scaling functions are offset by +1 for plot frame

printplot()
plot to stdout

residualsplot(*z*, *title=None*)
convenience to plot *z* vs point number

Parameters **z** (*numpy.ndarray*) – ordinates (standardized residuals)

xyplot(*x*, *y*, *title=None*)
convenience to plot *y(x)*

Parameters

- **x** (*numpy.ndarray*) – abcissae
- **y** (*numpy.ndarray*) – ordinates

Utility Routines

General mathematical toolbox routines

The routines that follow are part of my general mathematical "toolbox". Some of them are taken (with reference) from book(s) but most, I have developed on my own. They are modular in construction so that they may be improved, as needed.

jldesmear.jl_api.toolbox.**AskDouble**(*question*, *answer*)
request a double from the command line

Parameters

- **question** (*str*) – string to pose
- **answer** (*double*) – default answer

Returns final answer

Return type double

jldesmear.jl_api.toolbox.**AskInt**(*question*, *answer*)
request an integer from the command line

Parameters

- **question** (*str*) – string to pose
- **answer** (*int*) – default answer

Returns final answer

Return type int

jldesmear.jl_api.toolbox.**AskQuestion**(*question*, *answer*)
request a string, float, or int from the command line

Parameters

- **question** (*str*) – string to pose
- **answer** (*string | float | int*) – default answer

Returns final answer

Return type str | float | int

jldesmear.jl_api.toolbox.**AskString**(*question*, *answer*)
request a string from the command line

Parameters

- **question** (*str*) – string to pose
- **answer** (*str*) – default answer

Returns final answer

Return type str

jldesmear.jl_api.toolbox.**AskYesOrNo**(*question*, *answer*)
one of two choices seems simple

Parameters

- **question** (*str*) – string to pose
- **answer** (*str*) – default answer

Returns `y | n`

Return type str

jldesmear.jl_api.toolbox.**GetDat**(*infile*)
read three-column data from a wss (white-space-separated) file

Data appear as Q I dI with one data point per line. A "#" may be used to comment out any line.

Parameters **infile** (*string*) – name of input data file

Returns x, y, dy

Return type (numpy.ndarray, numpy.ndarray, numpy.ndarray)

jldesmear.jl_api.toolbox.**GetTest1DataFilename**(*ext='.smr'*)
find the test1 data in the package

jldesmear.jl_api.toolbox.**Iswap**(*a*, *b*)

Returns (tuple) of (b, a)

jldesmear.jl_api.toolbox.**SavDat**(*outfile*, *x*, *y*, *dy*)
save three column ASCII data in tab-separated file

Parameters

- **outfile** (*str*) – name of output file
- **x** (*numpy.ndarray*) – column 1 data array
- **y** (*numpy.ndarray*) – column 2 data array
- **dy** (*numpy.ndarray*) – column 3 data array

jldesmear.jl_api.toolbox.**Spinner**(*i*, *quiet=False*)
Spins a stick to indicate program is still working. Call this routine frequently during long operations to show progress.

Parameters

- **i** (*int*) – selector (increment this in the calling routine)
- **quiet** (*bool*) – optional switch to turn off the spinner

jldesmear.jl_api.toolbox.**find_first_index**(*x*, *target*)
find i such that x[i] >= target and x[i-1] < target

Parameters

- **x** (*ndarray*) – array to search
- **target** (*float*) – value to bracket

Returns index of array x or None

Return type int

jldesmear.jl_api.toolbox.**isDataLine**(*line*)
test if a given line of text is not blank or commented out

Parameters **line** (*string*) – line of text from an input file (usually)

Returns True | False

Return type bool

jldesmear.jl_api.toolbox.**strtrim**(*txt*)
cut out any white space from the string (compatibility method for legacy code only)

Change History

2015.0623.1 publish documentation at http://jldesmear.readthedocs.org

2015.0623.0 removed scipy.interpolate requirement, added desmear graphic to documentation

2015.0530.1 removed support for PySide and traits, refactored Python imports

2014.03.14 cleaned up extrapolation plugin recognition

2014.03.13 refactored extrapolations to be easier to recognize and improved import

Older Development: lake-python (subversion repository trunk)

Changes:

2013-12 and previous noted blow[1]

- refactored `api.desmear` into a class: `api.desmear.Desmearing`
  - allows iterating one at a time
  - computes ChiSqr data after iteration
  - keeps record of all ChiSqr values
- added single iteration method to `api.desmear`
- added single and N desmearing iteration controls to GUI
- update plots in the GUI after each iteration by running desmear calculation in a separate thread
- provided ChiSqr v iteration plot (log-lin)
- [2000] auto-discover all extrapolation functions
- [2000] renamed packages and modules to reduce overuse of "lake"
- [2000] moved content off first page of documentation
- [2002] start to refactor all GUI code from Enthought Traits to PySide (or PyQt4)

[1] subversion changesets are noted in square brackets such as [2002] is change set 2002

- [2005] add package installation support
- [2006] release test data with package
- [2006] rebrand package as JLdesmear (Jemian/Lake desmearing code)
- [2009] start to use numpy.ndarray() instead of [float]

TODO:

1. add capability for GUI to write desmeared data to a file
2. read data from CanSAS XML
3. read data from HDF5/NeXus

Older Production

This documents tagged releases.

2011-08-25: lake-python-2011-08

Initial release:

- python code fully operable
- command line interface
  - uses same paradigm as original FORTRAN code
  - Q&A in a console session, then desmear
  - tested on Windows, Macintosh, and Linux
  - only uses standard Python libraries, no NumPy or SciPy
- All test data copied from legacy C and FORTRAN projects
- Documentation
  - as good or better than FORTRAN manual
  - could improve still with content from thesis
- graphical user interface
  - provisional, demo only
  - uses Enthought's Traits and Chaco
  - does not write desmeared data to a file

License

```
OPEN SOURCE LICENSE

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.  Software changes,
   modifications, or derivative works, should be noted with comments and
   the author and organization's name.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of Pete R. Jemian
   nor the names of its contributors may be used to endorse or promote
   products derived from this software without specific prior written
   permission.

4. The software and the end-user documentation included with the
   redistribution, if any, must include the following acknowledgment:

   "This product includes software produced by Pete R. Jemian."

*****************************************************************************

DISCLAIMER

THE SOFTWARE IS SUPPLIED "AS IS" WITHOUT WARRANTY OF ANY KIND.

Neither Pete R. Jemian, nor any of their employees, makes
any warranty, express or implied, or assumes any legal liability or
responsibility for the accuracy, completeness, or usefulness of any
information, data, apparatus, product, or process disclosed, or
represents that its use would not infringe privately owned rights.

*****************************************************************************
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Note: These instructions have been very limited but are beginning to get significantly better. Additional help with the fundamentals of desmearing and its application to small-angle scattering are available in the Theory chapter of my PhD thesis: http://jemian.org/pjthesis.pdf

Python Module Index

j

Index

G

I

J

L

M

N

P

Q

R

S

T

X