SAXS Documentation

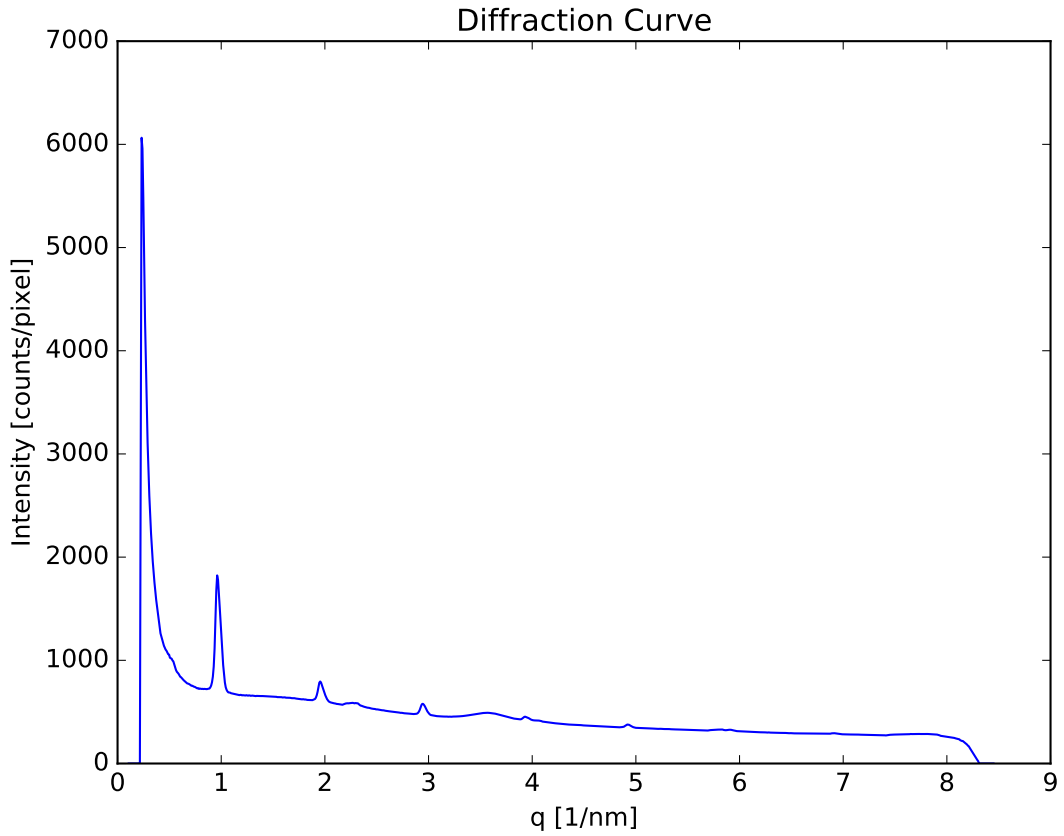
Release py3.5 V3

Christian Meisenbichler, Max Burian

Apr 15, 2019

CONTENTS

The SAXS Python package implements analysis tools for Small Angle X-Ray Scattering (SAXS) data. The first and most important one is to efficiently integrate 2d sensor data to an angle dependent diffraction curve.



The SAXS module consists of a Python library and 3 command line tools: *The Saxsdog*, *Plotchi* and *The Converter* and the *The Saxsdog Network* that integrates all of it and provides a GUI, *The Saxs Leash*.

CHAPTER

ONE

INSTALL

The SAXS Package distributes as a Python package. So in order to use it, you need a Python system installed. The program was originally written in Python 2.7 and has been updated to Python 3.5. **In the future, the AustoSAXS beamline will only support the Python 3.5 version!**

SAXSDog has been developed and tested in a specific Python environment that requires precise control of the used libraries. *We hence suggest to create a dedicated environment as explained further below*.

For software "end-users" we suggest to use the step-by-step instructions below. For implementation in a beamline-network, we recommend contacting the authors as some adjustments in the core-code might have to be necessary, depending on the available hardware infrastructure.

1.1 Sources

The source-code of SAXSDog is available from Github. Originally, SAXSdog was implemented in 2014 for Python 2.7. *This version will not be supported/updated by the AustroSAXS beamline!* The original repository can nevertheless be found at:

```
https://github.com/ChristianMeisenbichler/SAXS
```

After several program-extensions and optimizations, the code was updated to be compatible with Python 3.5.5 (last version supporting Qt4). **The latest version of SAXSdog can be obtained from**:

```
https://github.com/maxburian/SAXS_py3
```

1.2 Step-by-Step Instructions

The following is a step-by-step instruction to obtain and run SAXSDog on your machine. The instructions are given for Windows - MAC and Linux users might have to adjust them slightly. The SAXSdog code is based on the Anaconda framework, which is cross-platform compatible. In case of problems, please contact the authors.

1.2.1 1) Get Anaconda

Download and install the latest Anaconda Package for Python 3.7 from: https://www.anaconda.com/distribution/#download-section

1.2.2 2) Get GIT - version control

Download and install the latest GIT distribution from https://git-scm.com

1.2.3 3) Start Anaconda

Start your Anacoda console, which can be found under the name *Anaconda promp* or *Anaconda Powershell Pronpt*.

1.2.4 4) Find your installation path

Use the command line in the Anaconda console to navigate to the desired SAXSdog installation directory. If you are not sure, we suggest the following commands:

```
$ cd ~
$ mkdir GIT
$ cd GIT
```

This will create the folder "GIT" in your user-directory and set the current path to that folder.

1.2.5 5) Get SAXSdog

Use GIT to download a "clone-copy" of the latest SAXSdog version, by typing

```
$ git clone https://github.com/maxburian/SAXS_py3.git
```

Once you have downloaded the repository, move to correpsonding folder

```
$ cd SAXS_py3
```

1.2.6 6) Create the Python 3.5 Environment

SAXSdog is currently supported on Python 3.5 only and it requires very specific packet versions to work flawless. In order to create the corresponding environment and install all packages, type:

```
$ conda env create -f environment.yml
```

This can now take a few minutes as all packages have to be downloaded and installed.

In case you want to integrate SAXSdog in your existing environment, use the detailed list of the required dependencies further bellow.

1.2.7 7) Activate the Python Environment

You must now activate the Python environment such that you work with Python 3.5.5.:

```
$ conda activate py3p5_qt4
```

1.2.8 8) Install SAXSdog

You are now ready to install SAXSdog using the following command:

```
$ python setup.py install
```

1.2.9 9) Done!

You have now installed SAXSdog and the software is ready to be used. The installer has placed icons in the "Start Menu" as well as on your "Desktop". You can use either one to start "SAXSLeash": the graphical user interface to control your image integration.

1.2.10 (Optional) Create your Default Network Configuration

SAXSDog is a network-based program. If you want to use it in a feeder-based environment or such that it operates on a remote server, you will have to setup your network configuration. For more information, please read *The Saxsdog Network* and use

```
% saxsnetconf
```

This will generate a default configuration file with a random secret. The file must then be saved in `$Home/.saxdognetwork`.

1.3 Dependencies

In case you want to create your own environment, we provide a list of the required packages and versions for which SAXSdog has been tested.

Install using `$ conda install <module>=<version>=<build>`

```
- python=3.5.5=h0c2934d_2
- pyqt=4.11.4=py35_7

- bitarray=0.8.1=py35hfa6e2cd_1
- comtypes=1.1.4=py35_0
- jsonschema=2.6.0=py35h27d56d3_0
- matplotlib=1.5.1=np111py35_0
- numpy=1.11.3=py35h4a99626_4
- numpy-base=1.14.3=py35h5c71026_0
- pandas=0.23.0=py35h830ac7b_0
- pillow=3.4.2=py35_0
- pyqt=4.11.4=py35_7
- pytables=3.4.3=py35he6f6034_1
- pywin32=223=py35hfa6e2cd_1
- pyzmq=17.0.0=py35hfa6e2cd_1
- scipy=1.1.0=py35h672f292_0
- sphinx=1.7.9=py35_0
- sphinx_rtd_theme=0.4.3=py_0
- sphinxcontrib=1.0=py35_1
- sphinxcontrib-websupport=1.0.1=py35ha3690eb_1
- xlwt=1.3.0=py35hd04410a_0
```

Install using `$ pip install <module>==<version>`

```
- sphinxcontrib-programoutput==0.13
- sphinxcontrib-programscreenshot==0.0.0
- watchdog==0.9.0
- prettyplotlib==0.1.7
- py2exe==0.9.2.2
```

CHAPTER

TWO

THE TOOLS

2.1 The Saxsdog

The saxsdog is a script that converts directories with images to curves. It can use multiple threads and watch the file system for changes.

For help on the usage type:

```
$ saxsdog --help
Usage: saxsdog [options] directory/to/watch

Options:
  -h, --help            Show this help message and exit.
  -c FILE, --calibration=FILE
                        Path to calibration file (JSON).
  -t THREADS, --threads=THREADS
                        Number of concurrent threads.
  -m, --plotmonitor     Show a live updating plot window.
  -w, --watch           Watch directory for changes, using file system events
                        recursively for all sub directories.
  -r, --resume          Skip files that are already converted.
  -o OUTDIR, --out=OUTDIR
                        Specify output directory.
  -R RELPATH, --relpath=RELPATH
                        Specify output directory as relative path to image
                        file. Default: '../work'
  -s, --svg             Write plot to svg file.
  -p, --png             Write png of original.
  -P, --profile         Make a time Profile and print it.
  -S, --silent          Less output.
  -n, --nowalk          Don't scan for files already there, only watch file
                        system if -w flag is given.
  -D, --walkdirinthreads
                        Search all directories in parallel process.
  -V, --servermode      Servermode.
```

The calibration file must be a valid *Calibration file Reference*

2.2 The Converter

The converter extracts information from the calibration.txt generated by fit2d and adds them to a SAXS.calibration configuration file. (*Calibration file Reference*)

```
$ saxsconverter  --help
Usage:
To extract data from Fit2d output:
  saxsconverter [options] calibration.txt ouput.saxsconf
Or to convert fom older saxsconf:
```

(continues on next page)

(continued from previous page)

```
  saxsconverter [options] cal.saxsconf ouput.saxsconf


Options:
  -h, --help            show this help message and exit
  -t FILE, --template=FILE
                        Path to calibration file which serves as template.
```

If there is a target file and it is a valid *Calibration file Reference*, the parsed values are added or updated in place.

2.3 Plotchi

The tool "plotchi" plots a list of ".chi"-files:

```
$ plotchi  --help
Usage: plotchi [options] CHIFILE [List of more ".chi" files]

Options:
  -h, --help            show this help message and exit
  -o FILE, --out=FILE   Write the plot to FILE. The format is derived from the
                        suffix, e.g. '.svg','.pdf'.
  -c, --compare         Compare datasets to first one.
  -l, --log             Use log scale.
  -n, --no-legend       Hide legend.
  -t TITLE, --title=TITLE
                        Give plot title.
  -s N, --skip=N        Skip first N points.
  -k N, --clip=N        Clip last N points.
  -x TYPE, --xaxsistype=TYPE
                        Select type of X axis scale, might be
                        [linear|log|symlog]
  -y TYPE, --yaxsistype=TYPE
                        Select type of Y axis scale, might be
                        [linear|log|symlog]
```

2.4 Saxsdmerge

This tool adresses a very specific problem only applicable under our beamlines needs.

```
$ saxsdmerge   --help
Usage: saxsdmerge [options] iMPicture/dir peakinteg.log datalogger.log

Options:
  -h, --help            show this help message and exit
  -t SEC, --timeoffset=SEC
                        Time offset between logging time and time in
                        imagedata.
  -1, --syncfirst       Sync time by taking the time difference between first
                        shutter action and first image.
  -z, --synczero        Sync time by taking the time difference between first
                        shutter action and the zero.tif image.
  -o FILE, --outfile=FILE
                        Write merged dataset to this file. Format is derived
                        from the extesion.(.csv|.json|.hdf)
  -b, --batch           Batch mode (no plot).
  -c, --includechi      Include radial intensity data (.chi) in hdf.
  -f, --includetif      Include  all image data in hdf.
  -C FILE, --conf=FILE  Use config in  FILE to merge the data (ignore other
                        options)
```

What it does is the following: It merges the two log files by interpolating the missing data in the joined table of the the 1 sec interval data logger and the shutter log. The interpolation method takes the closest previous entry. The two logfiles are presumed to have the same clock. This data is then merged with the data extracted from the image headers and dedector logs and is written to a output file as a Table. As the image time stamps are from a different clock which might have a significant offset, the `-l` and `-t` options allow for dealing with that. In order to check if the time synchronization is reasonable, the tool displays a graph with the shutter times and the exposure times from the images.

One of the output file format options is hdf, the hirarchical data format, which ia a way to create portable binary data files. If you specify hdf, the radial average function "`.chi`" and even the image data can be written to the hdf file. The hdf export is done with pandas hdf exporter which in turn uses PyTables. The "`.chi`" files reside under the "Graphs" Label, the log data under the "LogData" label and the images if present under the "Images" Label. It can be restored to a Pandas pannel or data frame with:

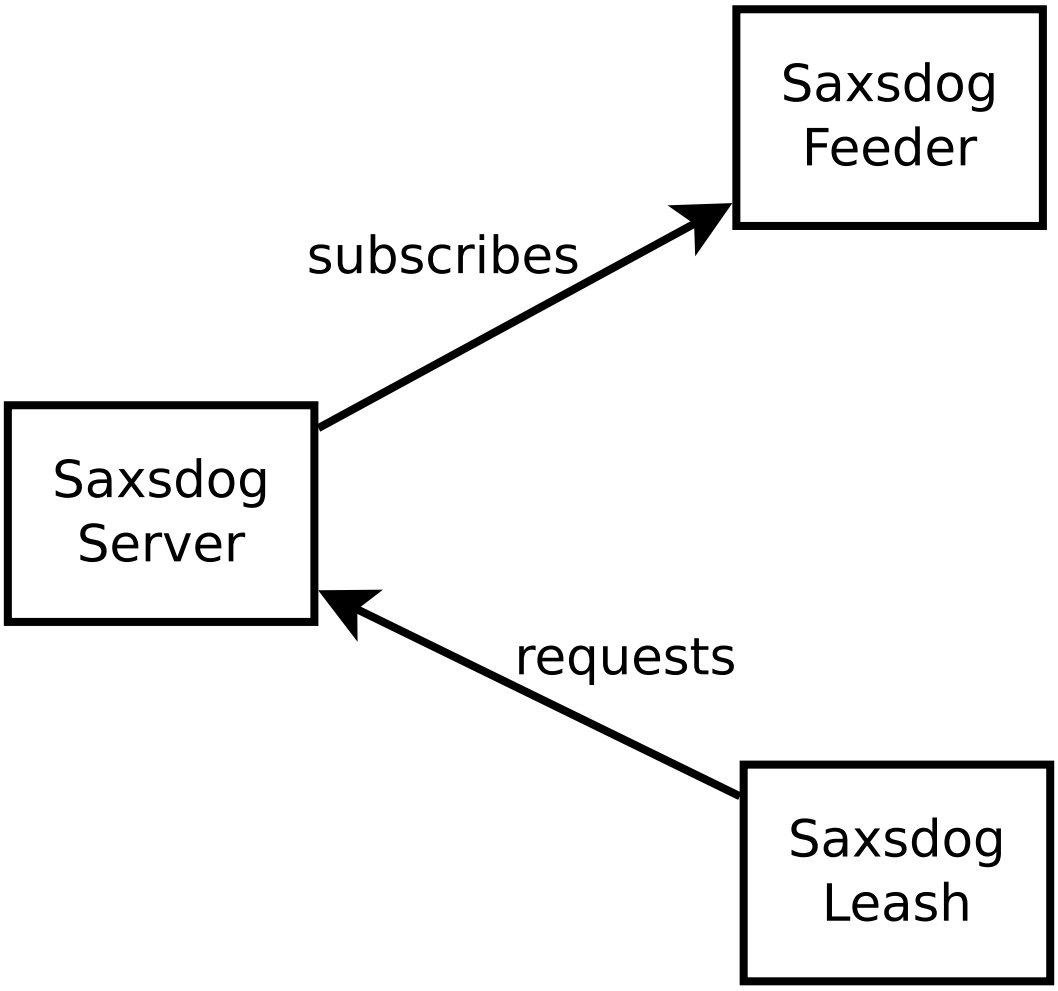
```
import pandas as pd
store=pd.HDFStore("/home/chm/saxs/merged.hdf")
Curves=store["Curves"] # gives Pandas panel with all the curves
Data=store["Data"] # gives Pandas data frame of merged log and image parameters
```

CHAPTER

THREE

THE SAXSDOG NETWORK

The network may consist of 3 different services. The “Saxsdog Server” does the image processing. The “Saxs Feeder” publishes new file events and the “Saxs Leash” controls an configures the server. The feeder is optional.



3.1 The SAXSNetwork configuration

The Saxsdog server and the Saxsleash have a common configuration file, which tells them how to connect with each other and which also includes a shared secret for authentication. If you want two computers to connect via the Saxsleash you need to have a copy of the file on each of them.

To create such a configuration, use the command:

```
$ saxsnetconf
```

It will ask for the Feeder URL and for the Saxsdog Server URL. Then it will generate a random secret and save the file in file in `$Home/.saxdognetwork`. You will have to copy the file to the other computers you need to allow to connect to your network. The secret must be the same on all of them.

```
[{
   "Server":"tcp://hostname:port",
   "Feeder":"tcp://hostname:port",
   "Secret":"Some large random string."
   "Name":"Pilatus1M"
}]
```

The authentication is done by hashing the request and the secret including a time stamp. The time stamp is checked if it lies within 900 seconds of the servers time.

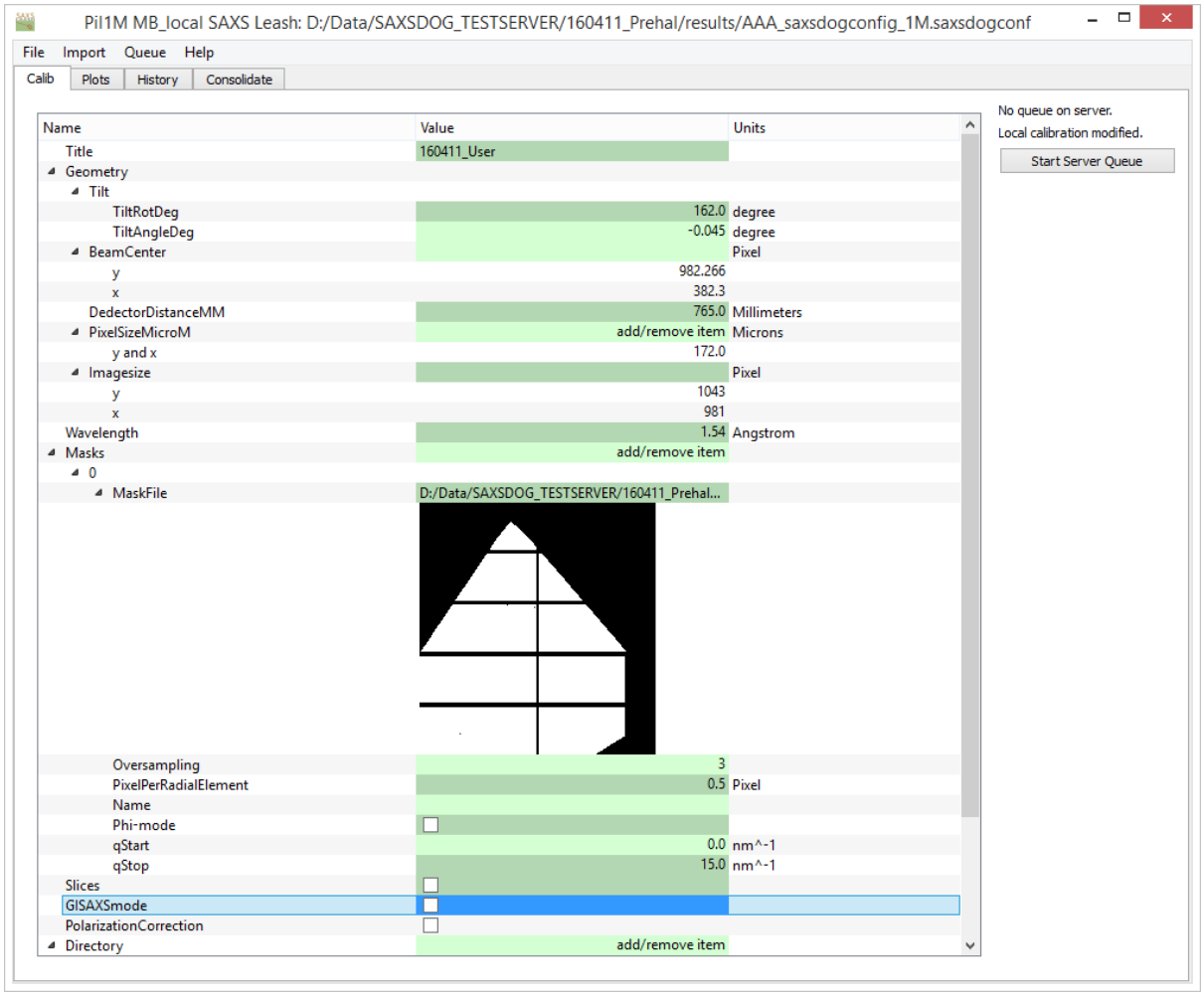
3.2 The Saxsdog Server

The Saxdog Server is the program that is started on the processing computer (node). It may subscribe to a "new file" event service.

```
$ saxsdogserver --help
Usage: saxsdogserver [options] basedir

Options:
  -h, --help            show this help message and exit
  -p port, --port=port  Port to offer command service. Default is 7777.
  -t THREADS, --threads=THREADS
                        Number of concurrent processes.
  -f tcp://hostname:port, --feeder=tcp://hostname:port
                        Specify the URL of the new file event service (Saxsdog
                        Feeder)
  -w, --watch           Watch directory for changes, using file system events
                        recursively for all sub directories.
  -R RELPATH, --relpath=RELPATH
                        Specify output directory as relative path to image
                        file. Default: '../work'
  -o OUTDIR, --out=OUTDIR
                        Specify output directory
  -d, --daemon          Start server  as daemon
```

3.3 The Saxs Leash



The Leash Program is a GUI to load calibrations into the Saxdog Server and monitor the processing of the data. It provides a calibration editor, as mask preview and basic data import from *The Converter*.

The main window has 3 tab cards. The first is for setting up the server, the second to review the currently processed data and the third for basic statistics. The command to launch it is.

```
leash
```

3.4 Saxs Leash Commandline

The "Saxs Leash" command line client can issue the commands for the Saxsdog Server.

```
$ saxsleash --help
Usage: saxsleash␣
↪close|abort|new|get|plot|plotdata|readdir|stat|listdir|fileslist|mergedata|getmergedata|mergesta
↪[options] [arguments]

Options:
  -h, --help            show this help message and exit
  -S tcp://HOSTNAME:PORT, --server=tcp://HOSTNAME:PORT
                        URL of "Saxsdog Server"
  -s N, --skip=N        plot: Skip first N points.
  -k N, --clip=N        plot: Clip last N points.
  -x TYPE, --xaxsistype=TYPE
```

(continues on next page)

(continued from previous page)

```
                        plot: Select type of X axis scale, might be
                        [linear|log|symlog]
  -y TYPE, --yaxsistype=TYPE
                        plot: Select type of Y axis scale, might be
                        [linear|log|symlog]
  -N SERVERNO, --serverno=SERVERNO
                        select server from config list by index default:0
```

Most of the command line options are about the `plot` command, but in order to visualize the processed data, one has to send the commands to setup a calibration.

3.4.1 New

```
$ saxsleash new cal.json data/AAA_integ.msk data/
```

The `new` command loads a calibration and starts the queue to receive new files. It requires 3 arguments:

1. Calibration file. as in *The Dedector Calibration File*,
2. mask file,
3. directory where the image files are or are going to be.

If there is a queue running, this command will abort the other one and replace it. One server can have only one queue at a time.

3.4.2 Plot

```
$ saxsleash plot
```

The `plot` command will grab the next image and show a plot of the result in a window. This command will be repeated until the user interrupts it with `Ctrl-C`.

3.4.3 Close

```
$ saxsleash close
```

Closes the queue. Which means, the server will process what is left in the queue but ignore all new files.

3.4.4 Abort

```
$ saxsleash abort
```

The `abort` command will close the queue and stop all data processing processes. It will only wait for each process to finish the picture they started before. The remaining pictures in the queue are ignored.

3.4.5 Stat

```
$ saxsleash stat
```

Return basic statistics data about the processes.

3.4.6 Read Dir

```
$ saxsleash readdir
```

This command will put all the images in the configured directory into the queue. This is useful to reprocess pictures.

3.4.7 List Directory

```
  $ saxsleash listdir Serverdirectory

Returns a list of files and directories on the server. Mostly used for the Leash␣
↪GUI dialoges.
```

3.4.8 Files List

```
$ saxsleash fileslist
```

Returns a list of processed files.

3.4.9 Merge Data

```
$  saxsleash mergedata mergeconf.json
```

Sends command to merge data. Takes a merge configuration file as argument.

3.5 The Saxsdog Network Protocol

3.5.1 The Saxsdog Server

The saxdog server can watch for files ystem events for himself or subscribe to a zmq service, the Saxsdog Feeder, that publishes new file names. The server can process the new images according to one calibration. The server may only have one calibration at a time, it is not designed to be used by multiple users at the same time.

3.5.2 The Saxsdog Feeder

The "Saxsdog Feeder" service offers file events for subscription. It should not do any buffering or pre-selection, just send a new message when any new file was copied and is ready for processing. Also when a file is overwritten: Send a message. It should however, only send this event, when the file is completely written to the file system.

New file events are composed of the following message:

```
{
   "command":"new file",
   "argument":"/Path/to/file/"
}
```

The service must be a ZeroMQ `zmq.PUP` socket. This code is a simulation of the messages:

```
import zmq
import random
import sys
import time
import os
import json
from optparse import OptionParser

def startfeeder():
    """
    Simulator for new file anounciation service. For development and testing.
    """
    parser = OptionParser()
    usage = "usage: %prog [options]  "
    parser = OptionParser(usage)
    parser.add_option("-p", "--port", dest="port",
                      help="Port to offer file changes service", metavar="port",␣
↪default="")
```

(continues on next page)

(continued from previous page)

```
    parser.add_option("-d", "--dir", dest="dir",
                      help="Directory to monitor", metavar="dir", default=".")
    parser.add_option("-s", "--sdir", dest="sdir",
                      help="server dir, (prefix to filepaths)", metavar="dir",␣
↪default=".")
    (options, args) = parser.parse_args(args=None, values=None)



    context = zmq.Context()
    socket = context.socket(zmq.PUB)
    if options.port=="":
        conf=json.load(open(os.path.expanduser("~"+os.sep+".saxsdognetwork")))
        port=conf['Feeder'].split(':')[-1]
    else:
        port=options.port
    print("conecting:", "tcp://*:%s" % port)
    socket.bind("tcp://*:%s" % port)

    fileslist=[]
    if len(args)>0 and options.dir==".":
        dirtosearch =args[0]
    else:
        dirtosearch =options.dir
    for path, subdirs, files in os.walk(dirtosearch):
                for name in files:
                    if name.endswith('tif'):
                        fileslist.append( os.path.join(path, name))
    messageobj={"command":"new file","argument":""}
    while True:
       for file in fileslist:
            print(file)
            messageobj['argument']=file
            message=json.dumps(messageobj)
            socket.send(message)
            time.sleep(7)


if __name__ == '__main__':
    startfeeder()
```

3.5.3 The Saxsdog Leash

The Saxsdog Leash is a user-facing control interface. There, the user should enter new calibrations and specify the data directories connected to it. During the processing, it shows a graph of one of the current images.

It may send the following commands:

Close

Request:

```
{
  "command":"close queue",
  "time":1404979588.715198,
  "sign":"Signature generated for request"
}
```

Answer:

```
{
"result":"queue closed",
"data":{
   "stat": {
      "time interval": 0.8776118755340576,
      "queue length": 0,
      "frames per sec": 10.25510279760422,
       "images processed": 235, "pics": 9
      }
   }
}
```

Abort

Request:

```
{
   "command":"abort queue",
   "time":1404979588.715198,
   "sign":"Signature generated for request"
}
```

Answer:

```
{
   "result":"queue stopped emptied and closed",
   "data":{
      "stat": {
         "time interval": 0.8776118755340576,
         "queue length": 0,
         "frames per sec": 10.25510279760422,
         "images processed": 235,
         "pics": 9
      }
   }
}
```

New

Request:

```
{
   "command":"new queue",
   "argument":{
      "directory":["path","to","data"],
      "calibration":{},
      "maskbin":""
   },
"time":1404979588.715198,
"sign":"Signature generated for request"
}
```

Answer:

```
{ "result":"new queue",
  "data":{
         }

}
```

Plot

Request:

```
{   "command":"send plot",
    "time":1404979588.715198,
    "sign":"Signature generated for request"
}
```

Answer:

```
{
 "result":"plot data",
 "data":{
    "filename":"/name/.tiv" ,
    "stat": {
           "time interval": 0.8776118755340576,
           "queue length": 0,
           "frames per sec": 10.25510279760422,
           "images processed": 235,
           "pics": 9
           },
    "array":[[0],[0],[0]]
    }
}
```

Readdir

This puts all existing files in the queue directory into the queue again.

Request:

```
{
    "command":"readdir",
    "time":1404979588.715198,
    "sign":"Signature generated for request"

}
```

Answer:

```
{
    "result":"directory refilled queue",
    "data":{
       "stat": {
          "time interval": 0.8776118755340576,
          "queue length": 0,
          "frames per sec": 10.25510279760422,
          "images processed": 235, "pics": 9
       }
    }
}
```

Stat

Get basic processing statistics.

Request:

```
{   "command":"stat","argument":{},
    "time":1404979588.715198,
    "sign":"Signature generated for request"}
```

Answer:

```
{
"data": {
   "stat": {
      "time interval": 711.6886098384857,
      "queue length": 0,
      "frames per sec": 9.972057866165134,
      "images processed": 7332,
      "pics": 7097
      }
    },
"result": "stat"
}
```

Error

In case of error in the Saxsdog server it will return an error message:

```
{
"result":"Error",
"data":{"Error":"Error message"}
}
```

3.6 The Protocol Schemas

3.6.1 Leash Request Schema

This describes how the requests to the server must be composed.

The '*' signifies a required Field.

Schema for requests from Saxs Leash to Saxs Server

Type object

Contains *command**, *argument*, *sign*, *time*

Required True

JSON Path

- *#*

Example JSON:

```
{"command": "close"}
```

command

Type string

values [close, abort, new, get, plot, plotdata, readdir, stat, listdir, fileslist, mergedata, getmergedata]

Required True

JSON Path

- *#* ['*command*']

Example JSON:

```
{"command": "close"}
```

argument

Type object

Contains *calibration*, *mergeconf*, *data*, *directory*

Required False

JSON Path

- *#* [‘*argument*’]

Example JSON:

```
{"argument": {}}
```

calibration

Calibration data according to *The Dedector Calibration File*

Type object

Contains see *The Dedector Calibration File*

Required False

JSON Path

- *#* [‘*argument*’][‘*calibration*’]

Example JSON:

```
{"calibration": {}}
```

mergeconf

Datamerger Configuratioin

Type object

Contains see *Leash Datamerge and saxsdmerge Schema*

Required False

JSON Path

- *#* [‘*argument*’][‘*mergeconf*’]

Example JSON:

```
{"mergeconf": {}}
```

data

type object

Required False

JSON Path

- *#* [‘*argument*’][‘*data*’]

Example JSON:

```
{"data": {}}
```

directory

type object

Required False

JSON Path

- *#* [‘*argument*’][‘*directory*’]

Example JSON:

```
{"directory": {}}
```

sign

Signature of request

Type string

Required False

JSON Path

- *#* [‘*sign*’]

Example JSON:

```
{"sign": ""}
```

time

time in seconds (pythons time.time())

Type number

Required False

JSON Path

- *#* [‘*time*’]

Example JSON:

```
{"time": 0}
```

3.6.2 Leash Response Schema

This is the Schema of the data the server sends back as a response.

The ‘*’ signifies a required Field.

Schema for requests from Saxs Leash to Saxs Server

Type object

Contains *result**, *data**

Required True

JSON Path

- *#*

Example JSON:

```
{"data": {},"result": ""}
```

result

Type string

Required True

JSON Path

- *#* [‘*result*’]

Example JSON:

```
{"result": ""}
```

data

Type object

Contains *cal*, *Error*, *syncplot*, *directory*, *attachments*, *threads*, *dircontent*, *history*, *fileslist*, *stat*, *filename*, *IntegralParameters*, *graphs*

Required True

JSON Path

- *#* [‘*data*’]

Example JSON:

```
{"data": {}}
```

cal

Type object

Required False

JSON Path

- *#* [‘*data*’][‘*cal*’]

Example JSON:

```
{"cal": null}
```

Error

type object

Required False

JSON Path

- *#* [‘*data*’][‘*Error*’]

Example JSON:

```
{"Error": {}}
```

syncplot

type object

Required False

JSON Path

- *#* [‘*data*’][‘*syncplot*’]

Example JSON:

```
{"syncplot": {}}
```

directory

Directory this queue is going to use. New files in other directories are going to be ignored.

Type array() items:

Required False

Default [u'.', u'', u'']

JSON Path

- *#* ['*data*']['*directory*']

Example JSON:

```
{"directory": [".","",""]}
```

attachments

type object

Required False

JSON Path

- *#* ['*data*']['*attachments*']

Example JSON:

```
{"attachments": {}}
```

threads

Type integer

Required False

JSON Path

- *#* ['*data*']['*threads*']

Example JSON:

```
{"threads": 0}
```

dircontent

type object

Required False

JSON Path

- *#* ['*data*']['*dircontent*']

Example JSON:

```
{"dircontent": {}}
```

history

type object

Required False

JSON Path

- *#* [‘*data*’][‘*history*’]

Example JSON:

```
{"history": {}}
```

fileslist

type object

Required False

JSON Path

- *#* [‘*data*’][‘*fileslist*’]

Example JSON:

```
{"fileslist": {}}
```

stat

type object

Contains *queue length*, *images processed*, *time*, *start time*, *mergecount*

Required False

JSON Path

- *#* [‘*data*’][‘*stat*’]

Example JSON:

```
{"stat": {}}
```

queue length

Type integer

Required False

JSON Path

- *#* [‘*data*’][‘*stat*’][‘*queue length*’]

Example JSON:

```
{"queue length": 0}
```

images processed

Type integer

Required False

JSON Path

- *#* [‘*data*’][‘*stat*’][‘*images processed*’]

Example JSON:

```
{"images processed": 0}
```

time

Type number

Required False

JSON Path

- *#* [‘*data*’][‘*stat*’][‘*time*’]

Example JSON:

```
{"time": 0}
```

start time

Type number

Required False

JSON Path

- *#* [‘*data*’][‘*stat*’][‘*start time*’]

Example JSON:

```
{"start time": 0}
```

mergecount

Type number

Required False

JSON Path

- *#* [‘*data*’][‘*stat*’][‘*mergecount*’]

Example JSON:

```
{"mergecount": 0}
```

filename

Type string

Required False

JSON Path

- *#* [‘*data*’][‘*filename*’]

Example JSON:

```
{"filename": ""}
```

IntegralParameters

type object

Required False

JSON Path

- *#* [‘*data*’][‘*IntegralParameters*’]

Example JSON:

```
{"IntegralParameters": {}}
```

graphs

Type array() items: {*kind*, *conf*, *columnLabels*, *array*}

Required False

JSON Path

- *#* [‘*data*’][‘*graphs*’]

Example JSON:

```
{"graphs": []}
```

kind

Type string

values [Radial, Slice]

Required False

JSON Path

- *#* [‘*data*’][‘*graphs*’][0][‘*kind*’]

Example JSON:

```
{"kind": "Radial"}
```

conf

Type object

Required False

JSON Path

- *#* [‘*data*’][‘*graphs*’][0][‘*conf*’]

Example JSON:

```
{"conf": null}
```

columnLabels

Type array() items: string

Required False

JSON Path

- *#* [‘*data*’][‘*graphs*’][0][‘*columnLabels*’]

Example JSON:

```
{"columnLabels": []}
```

array

Type array() items:

Required False

JSON Path

- *#* [‘*data*’][‘*graphs*’][0][‘*array*’]

Example JSON:

```
{"array": []}
```

3.6.3 Leash Datamerge and saxsdmerge Schema

The data merge tool can merge (join) log files with data from the images and create consolidated data files for archiving an further processing.

The ‘*’ signifies a required Field.

Type object

Contains *TimeOffset**, *LogDataTables*, *OutputFormats**, *OutputFileBaseName**, *HDFOptions**

Required False

JSON Path

- *#*

Example JSON:

```
{
  "TimeOffset": "0",
  "HDFOptions": {
    "IncludeCHI": false,
    "IncludeTIF": false
  },
  "OutputFormats": {
    "exel": false,
    "hdf": false,
    "json": false,
    "csv": false
  },
  "OutputFileBaseName": "../results/merged"
}
```

LogDataTables

Define log files to consolidate with image data. If more then one defined, they will be joined and missing values will be interpolated.

Type array() items: {*TimeEpoch*, *TimeOffset*, *FirstImageCorrelation*, *Name*, *Files*}

Required False

JSON Path

- *#* [‘*LogDataTables*’]

Example JSON:

```
{"LogDataTables": []}
```

TimeEpoch

Time epoch

Type string

values [Mac, Unix]

Required True

Default Mac

JSON Path

- *#* [‘*LogDataTables*’][0][‘*TimeEpoch*’]

Example JSON:

```
{"TimeEpoch": "Mac"}
```

TimeOffset

If offset is not found otherwise, use this as offset.

Type number in Seconds

Required True

Default 0

JSON Path

- *#* [‘*LogDataTables*’][0][‘*TimeOffset*’]

Example JSON:

```
{"TimeOffset": "0"}
```

FirstImageCorrelation

Find offset for all log data by correlating first image with first entry of this table.

Type boolean

Required True

Default False

JSON Path

- *#* [‘*LogDataTables*’][0][‘*FirstImageCorrelation*’]

Example JSON:

```
{"FirstImageCorrelation": false}
```

Name

Name field to be used as prefix in the joined collumn names.

Type string

Required True

Default log

JSON Path

- *#* [‘*LogDataTables*’][0][‘*Name*’]

Example JSON:

```
{"Name": "log"}
```

Files

One log table may be one file, or a list of files to be concatenated.

Type array() items: {*RemotePath*, *LocalPath*}

Required False

JSON Path

- *#* [‘*LogDataTables*’][0][‘*Files*’]

Example JSON:

```
{"Files": []}
```

RemotePath

Path of logfile on server if used in server mode.

Type array() items: string

Required False

JSON Path

- *#* [‘*LogDataTables*’][0][‘*Files*’][0][‘*RemotePath*’]

Example JSON:

```
{"RemotePath": []}
```

LocalPath

Path of logfile on client. Overrides *RemotePath*.

Type string

Required False

JSON Path

- *#* [‘*LogDataTables*’][0][‘*Files*’][0][‘*LocalPath*’]

Example JSON:

```
{"LocalPath": ""}
```

OutputFormats

List of outputformats to write the consolidated log or the consolidated ‘hdf’ file.

Type object

Contains *csv**, *hdf**, *exel**, *json**

Required True

JSON Path

- *#* [‘*OutputFormats*’]

Example JSON:

```
{
  "OutputFormats": {
    "exel": false,
    "hdf": false,
    "json": false,
    "csv": false
  }
}
```

csv

Type boolean

Required True

Default False

JSON Path

- *#* [‘*OutputFormats*’][‘*csv*’]

Example JSON:

```
{"csv": false}
```

hdf

Type boolean

Required True

Default False

JSON Path

- *#* [‘*OutputFormats*’][‘*hdf*’]

Example JSON:

```
{"hdf": false}
```

exel

Type boolean

Required True

Default False

JSON Path

- *#* [‘*OutputFormats*’][‘*exel*’]

Example JSON:

```
{"exel": false}
```

json

Type boolean

Required True

Default False

JSON Path

- *#* [‘*OutputFormats*’][‘*json*’]

Example JSON:

```
{"json": false}
```

OutputFileBaseName

Type string

Required True

Default ../results/merged

JSON Path

- *#* [‘*OutputFileBaseName*’]

Example JSON:

```
{"OutputFileBaseName": "../results/merged"}
```

HDFOptions

Options only relevant to hdf export.

Type object

Contains *IncludeCHI**, *IncludeTIF**

Required True

JSON Path

- *#* [‘*HDFOptions*’]

Example JSON:

```
{"HDFOptions": {"IncludeCHI": false,"IncludeTIF": false}}
```

IncludeCHI

Whether to include the .chi files as strings.

Type boolean

Required True

Default False

JSON Path

- *#* [‘*HDFOptions*’][‘*IncludeCHI*’]

Example JSON:

```
{"IncludeCHI": false}
```

IncludeTIF

Whether to include the images as integer array.

Type boolean

Required True

Default False

JSON Path

- *#* [‘*HDFOptions*’][‘*IncludeTIF*’]

Example JSON:

```
{"IncludeTIF": false}
```

CHAPTER

FOUR

THE DEDECTOR CALIBRATION FILE

The *SAXS.calibration* class and the saxsdog tool accepts a input file with the calibration data. This input file is written as a JSON file. This is a common syntax to express structured data as text. You might want to read a bit about it before moving on.

4.1 Calibration file Reference

The '*' signifies a required Field.

The SAXS configuration file specifies the parameters of a SAXS sensor calibration. It is written in the JSON format which governs the general syntax.

Type object

Contains *Title*, *Geometry**, *Masks**, *Slices*, *Wavelength**, *PolarizationCorrection*, *Directory**, *Threads*, *Live-Filelisting*, *OverwriteFiles*, *GISAXSmode*

Required True

JSON Path

- *#*

Example JSON:

```
{
  "Title": "Example Calibration",
  "Geometry": {
    "Tilt": {
      "TiltRotDeg": 0,
      "TiltAngleDeg": 0
    },
    "Imagesize": [
      1000,
      900
    ],
    "BeamCenter": [
      800.0,
      400.0
    ],
    "PixelSizeMicroM": [
      100.0
    ],
    "DedectorDistanceMM": 1000.0
  },
  "Wavelength": 1.54,
  "Directory": [],
  "Live-Filelisting": true,
  "OverwriteFiles": true,
  "Threads": 2,
  "Masks": [],
```

(continues on next page)

(continued from previous page)

```
  "Slices": [],
  "GISAXSmode": false
}
```

4.1.1 Title

Type string

Required False

JSON Path

- *#* [‘*Title*’]

Example JSON:

```
{"Title": ""}
```

4.1.2 Geometry

Type object

Contains *Tilt**, *BeamCenter**, *DedectorDistanceMM**, *PixelSizeMicroM**, *Imagesize**

Required True

JSON Path

- *#* [‘*Geometry*’]

Example JSON:

```
{
  "Title": "Example Calibration"
  "Geometry": {
    "Tilt": {
      "TiltRotDeg": 0,
      "TiltAngleDeg": 0
    },
    "Imagesize": [
      1000,
      900
    ],
    "BeamCenter": [
      800.0,
      400.0
    ],
    "PixelSizeMicroM": [
      100.0
    ]
  }
}
```

4.1.3 Tilt

The sensor, usually is not perfectly perpenticular to the ray direction. The tilt angle can be specified by giving the following paramters.

Type object

Contains *TiltRotDeg**, *TiltAngleDeg**

Required True

JSON Path

- # [‘*Geometry*’][‘*Tilt*’]

Example JSON:

```
{"Tilt": {"TiltRotDeg": 0,"TiltAngleDeg": 0}}
```

4.1.4 TiltRotDeg

This gives the angel of the tilt direction.

Type number in degree

Required True

Default 0

JSON Path

- # [‘*Geometry*’][‘*Tilt*’][‘*TiltRotDeg*’]

Example JSON:

```
{"TiltRotDeg": 0}
```

4.1.5 TiltAngleDeg

This gives the angle between the ray direction and the normal to the sensor plane.

Type number in degree

Required True

Default 0

JSON Path

- # [‘*Geometry*’][‘*Tilt*’][‘*TiltAngleDeg*’]

Example JSON:

```
{"TiltAngleDeg": 0}
```

4.1.6 BeamCenter

Gives the beam center in pixel coorinates.

Type array(2) items: number

Required True

Default [800.0, 400.0]

JSON Path

- # [‘*Geometry*’][‘*BeamCenter*’]

Example JSON:

```
{"BeamCenter": [800.0,400.0]}
```

4.1.7 DedectorDistanceMM

Distance between diffraction center and sensor.

Type number in Millimeters

Required True

Default 1000.0

JSON Path

- *#* [‘*Geometry*’][‘*DedectorDistanceMM*’]

Example JSON:

```
{"DedectorDistanceMM": 1000.0}
```

4.1.8 PixelSizeMicroM

The pixel size on the sensor.

Type array(2) items: number

Required True

Default [100.0]

JSON Path

- *#* [‘*Geometry*’][‘*PixelSizeMicroM*’]

Example JSON:

```
{"PixelSizeMicroM": [100.0]}
```

4.1.9 Imagesize

Size of sensor image in pixel.

Type array(2) items: integer

Required True

Default [1000, 900]

JSON Path

- *#* [‘*Geometry*’][‘*Imagesize*’]

Example JSON:

```
{"Imagesize": [1000,900]}
```

4.1.10 Masks

Type array() items: {*MaskFile*, *Oversampling*, *PixelPerRadialElement*, *Name*, *qStart*, *qStop*, *Phimode*}

Required True

JSON Path

- *#* [‘*Masks*’]

Example JSON:

```
{"Masks": []}
```

4.1.11 MaskFile

Path of Maskfile

Type string

Required True

Default AAA_integ.msk

JSON Path

- *#* [‘*Masks*’][0][‘*MaskFile*’]

Example JSON:

```
{"MaskFile": "AAA_integ.msk"}
```

4.1.12 Oversampling

Oversampling factor for radial integration. The higher, the longer the setup but the higher the accuracy. More then 3 is probably overkill.

Type integer

Required True

Default 3

JSON Path

- *#* [‘*Masks*’][0][‘*Oversampling*’]

Example JSON:

```
{"Oversampling": 3}
```

4.1.13 PixelPerRadialElement

Expresses the width of a radial step in terms of pixels. ‘1’ means $\delta R \approx 1$ *PixelSizeMicroM*.

Type number in Pixel

Required True

Default 1

JSON Path

- *#* [‘*Masks*’][0][‘*PixelPerRadialElement*’]

Example JSON:

```
{"PixelPerRadialElement": 1}
```

4.1.14 Name

Name for mask configuration.

Type string

Required False

JSON Path

- *#* [‘*Masks*’][0][‘*Name*’]

Example JSON:

```
{"Name": ""}
```

4.1.15 qStart

Starting q-value for integral parameters.

Type number in nm^-1

Required True

Default 0

JSON Path

- # ['*Masks*'][0]['*qStart*']

Example JSON:

```
{"qStart": 0}
```

4.1.16 qStop

Ending q-value for integral parameters.

Type number in nm^-1

Required True

Default 0

JSON Path

- # ['*Masks*'][0]['*qStop*']

Example JSON:

```
{"qStop": 0}
```

4.1.17 Phi-mode

To perform an integration as a function of angle in the give radial regime defined by qStart and qStop .

Type Boolean

Required False

Default False

JSON Path

- # ['*Masks*'][0]['*Phi-mode*']

Example JSON:

```
{"Phi-mode": False}
```

4.1.18 Slices

Slices are designed to analyse GISAXS data.It allows you to specify slices along the sensor axis and get intensity along q_x, q_z directions.

Type array() items: {*Direction*, *Plane*, *CutPosition*, *CutMargin*, *IncidentAngle*, *MaskRef* }

Required False

JSON Path

- # ['*Slices*']

Example JSON:

```
{"Slices": []}
```

4.1.19 Direction

'x' or 'y' direction in sensor pixel coordinates.

Type string

values [x, y]

Required True

Default x

JSON Path

- *#* ['*Slices*'][0]['*Direction*']

Example JSON:

```
{"Direction": "x"}
```

4.1.20 Plane

Whether the direction is in plane with scattering surface or vertical to it.

Type string

values [InPlane, Vertical]

Required True

Default InPlane

JSON Path

- *#* ['*Slices*'][0]['*Plane*']

Example JSON:

```
{"Plane": "InPlane"}
```

4.1.21 CutPosition

Cut position in pixel coordinates in the other coodinate then specified in 'Direction'. Origin is top left Corner.

Type integer in Pixel

Required True

Default 0

JSON Path

- *#* ['*Slices*'][0]['*CutPosition*']

Example JSON:

```
{"CutPosition": "0"}
```

4.1.22 CutMargin

Number of pixels left and right from cut to include into the average .

Type integer in Pixel

Required True

Default 1

JSON Path

- *#* ['*Slices*'][0]['*CutMargin*']

Example JSON:

```
{"CutMargin": 1}
```

4.1.23 IncidentAngle

Angle of incidence in GISAXS setup.

Type number in degree

Required True

Default 0

JSON Path

- *#* [‘*Slices*’][0][‘*IncidentAngle*’]

Example JSON:

```
{"IncidentAngle": 0}
```

4.1.24 MaskRef

Chose which mask to use for the sclice. ‘-1’ means don’t use mask

Type integer

Required True

Default 0

JSON Path

- *#* [‘*Slices*’][0][‘*MaskRef*’]

Example JSON:

```
{"MaskRef": 0}
```

4.1.25 Wavelength

Refined wavelength.

Type number in Angstrom

Required True

Default 1.54

JSON Path

- *#* [‘*Wavelength*’]

Example JSON:

```
{"Wavelength": 1.54}
```

4.1.26 PolarizationCorrection

The scattering direction id dependend on the light polarization. This may be acconted for with the polarization correction.

Type object

Contains *Fraction**, *Angle**

Required False

Default OrderedDict([(u'Fraction', 0.95), (u'Angle', 0)])

JSON Path

- # ['*PolarizationCorrection*']

Example JSON:

```
{"PolarizationCorrection": {"Angle": 0.0,"Fraction": 0.95}}
```

4.1.27 Fraction

Fraction of light polarized in the given (*Angle*) direction.

Type number

Required True

Default 0.95

JSON Path

- # ['*PolarizationCorrection*']['*Fraction*']

Example JSON:

```
{"Fraction": 0.95}
```

4.1.28 Angle

Angle of the polarization plane.

Type number in degree

Required True

Default 0.0

JSON Path

- # ['*PolarizationCorrection*']['*Angle*']

Example JSON:

```
{"Angle": 0.0}
```

4.1.29 Directory

Directory to take into acount for processing images. Given as a list of subdirectories.

Type array() items: string

Required True

JSON Path

- # ['*Directory*']

Example JSON:

```
{"Directory": []}
```

4.1.30 Threads

Number of threads that will simultaniously perform the integration.

Type integer

Required False

Default 2

JSON Path

- # [‘*Threads*’]

Example JSON:

```
{"Threads": 2}
```

4.1.31 Live-Filelisting

Writes all processed file names (.chi) into a log-file in ./results/

Type Boolean

Required False

Default True

JSON Path

- # [‘*Live-Filelisting*’]

Example JSON:

```
{"Live-Filelisting": True}
```

4.1.32 GISAXSmode

If true, no radial integration will be performed and only slices will be integrated.

Type Boolean

Required False

Default False

JSON Path

- # [‘*GISAXSmode*’]

Example JSON:

```
{"GISAXSmode": False}
```

4.1.33 OverwriteFiles

Select if already integrated files should be overwritten.

Type Boolean

Required False

Default True

JSON Path

- # [‘*OverwriteFiles*’]

Example JSON:

```
{"OverwriteFiles": False}
```

CHAPTER

FIVE

THE TECHNOLOGY

5.1 Integration as Matrix-Vector Multiplication

Every SAXS image $\mathbf{p}$ is a list of pixels that have an intensity value. This 2d array might as well be addressed as a vector with all the pixels addressable with one index $\mathbf{p}_i$.

The integration over pixels that are within a certain radial interval is in any case a weighted sum of some of the pixels.

This weighted sum is a scalar product with another vector containing the weight factors. As only the pixels in a radius interval are counted, most of these factors are 0.

$$r = \mathbf{c} \cdot \mathbf{p}$$

As we intend to do all the radial intervals at once, we write it as a matrix vector product.

$$\mathbf{r} = \mathbf{X} \cdot \mathbf{p}$$

The columns are the weight factors for the i^{th} radial element. Rearranged in the order of the image, this looks like the ring element relevant for the radial Point.

This matrix would be quite big as it has the dimensions len($\mathbf{r}$)*len($\mathbf{p}$). Fortunately most of the entries are 0 and we can use a sparse matrix representation which uses only about ~len($\mathbf{p}$) of memory, as every pixel is counted only once, or, as we will see, about once.

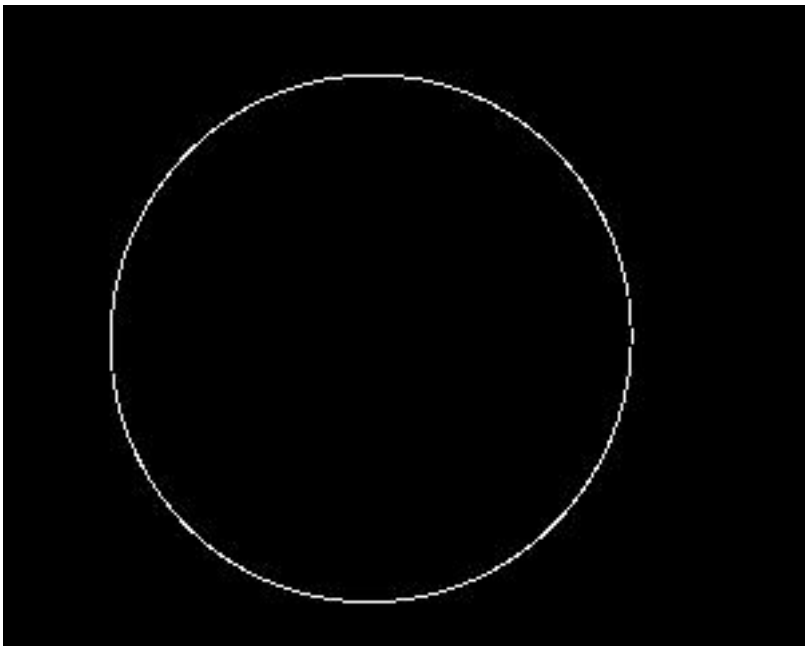


Fig. 1: The vector $\mathbf{c}$ displayed as image.

Above shows the data of such a matrix column.

5.2 Over sampling

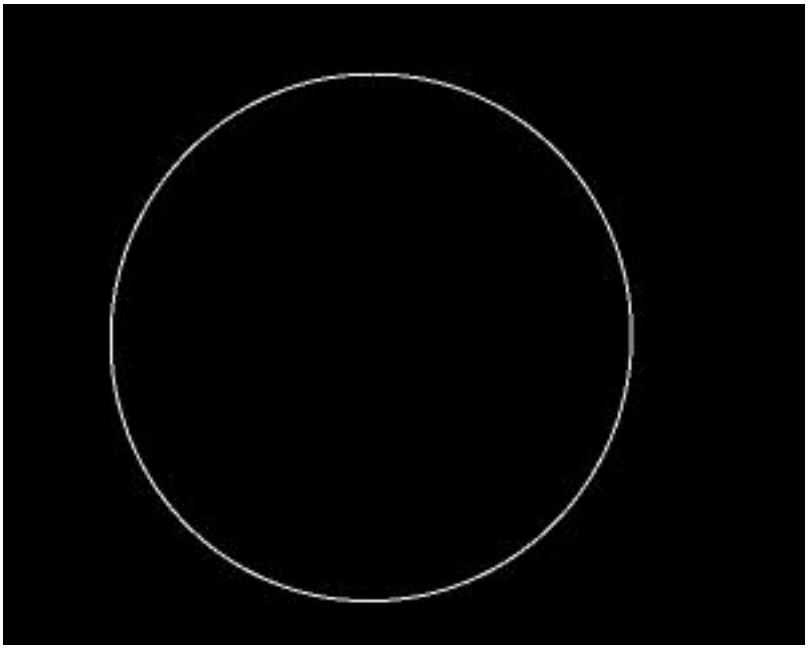


Fig. 2: Ring with anti aliasing / over sampling.

A pixel might lie on the border of two radial intervals, making it unclear to which one it should be added. By only choosing the nearest one, one may get artifacts in the resulting curve especially when only few pixels contribute. So, how could we calculate to which fraction a pixel should account to one radial interval?

The idea here is to use an algorithm comparable to anti aliasing in computer graphics. We will divide a much larger picture into the radial intervals and down sample it to the real pixels. Which results in nicely balanced factors for the border pixels that add up nicely over joining intervals such that the intensity is conserved. If one looks closer at the image above , one sees that the ring has soft edges. Quite as it would have through anti aliasing.

5.3 The Geometry

The plane of the sensor is not perfectly normal to the beam. So in order to calculate witch pixel is on witch cone in the diffracted light, we need to express the geometry somehow.

Every pixel has the polar coordinates r, ϕ with the projected diffraction center in the origin. For each pixel (P) the triangle S,C,P (Sample, Center, Pixel, θ, β, γ) can be fully expressed with the law of cosines.

l is the distance the light travels from the diffraction center to the sensor.

$$l^2 = d^2 + r^2 - 2dr\cos(\pi/2 + \alpha)$$

r is the radial coordinate of the pixel P.

$$r^2 = l^2 + d^2 - 2ld\cos(\theta)$$

from these two formulas the diffraction angle (here) θ can be computed.

$$\theta = \arccos(-r^2 - l^2 - d^2 2ld)$$

α comes from the following relation in figure *Angle between two planes.*

The angle between the sensor plane and the normal plane to the ray is given by τ. The slope s derived from τ is

$$s = \sin(\tau)$$

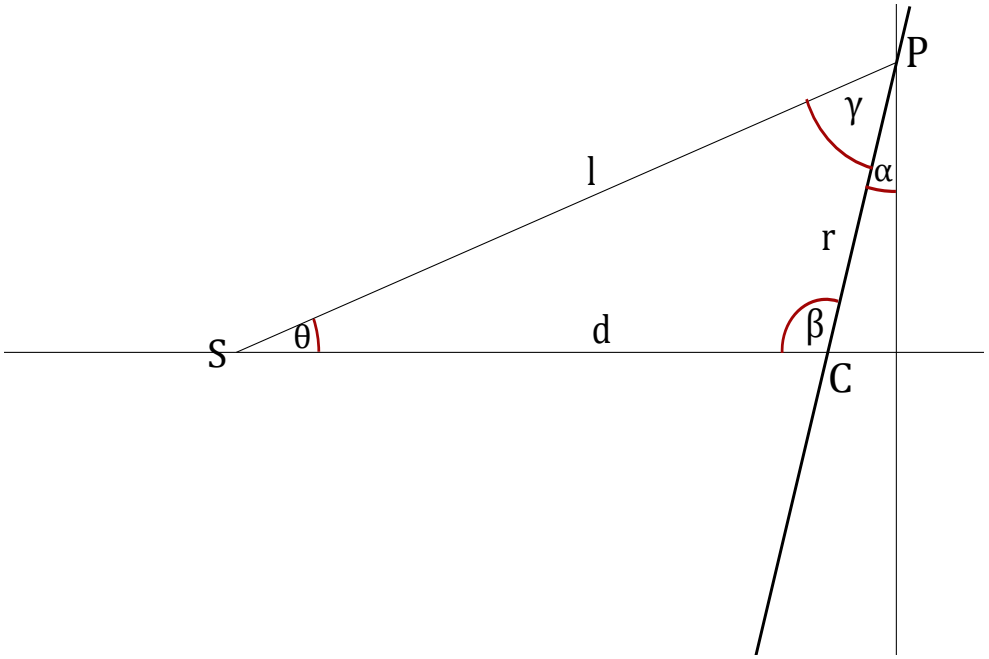


Fig. 3: The SCP triangle.

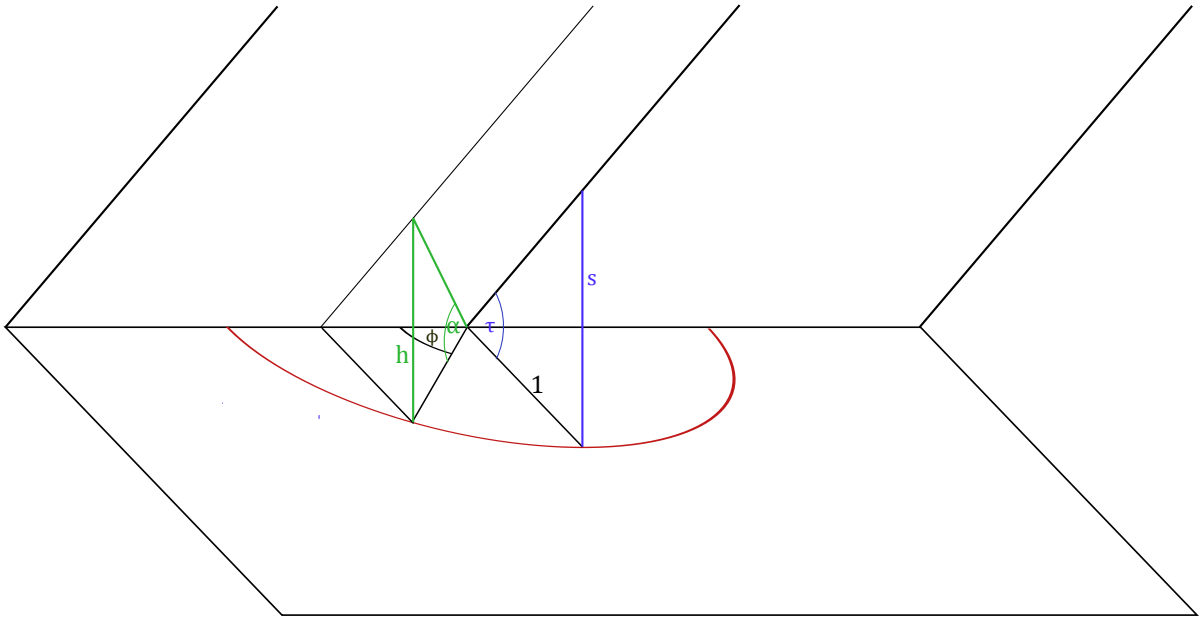


Fig. 4: Angle between two planes.

On the (red) unit circle in the plane of the sensor the distance to the plane normal to the ray is expressed as

$$h = \sin(\phi)s$$

The angle α is therefore:

$$\alpha = \arcsin(\sin(\tau)\sin(\phi))$$

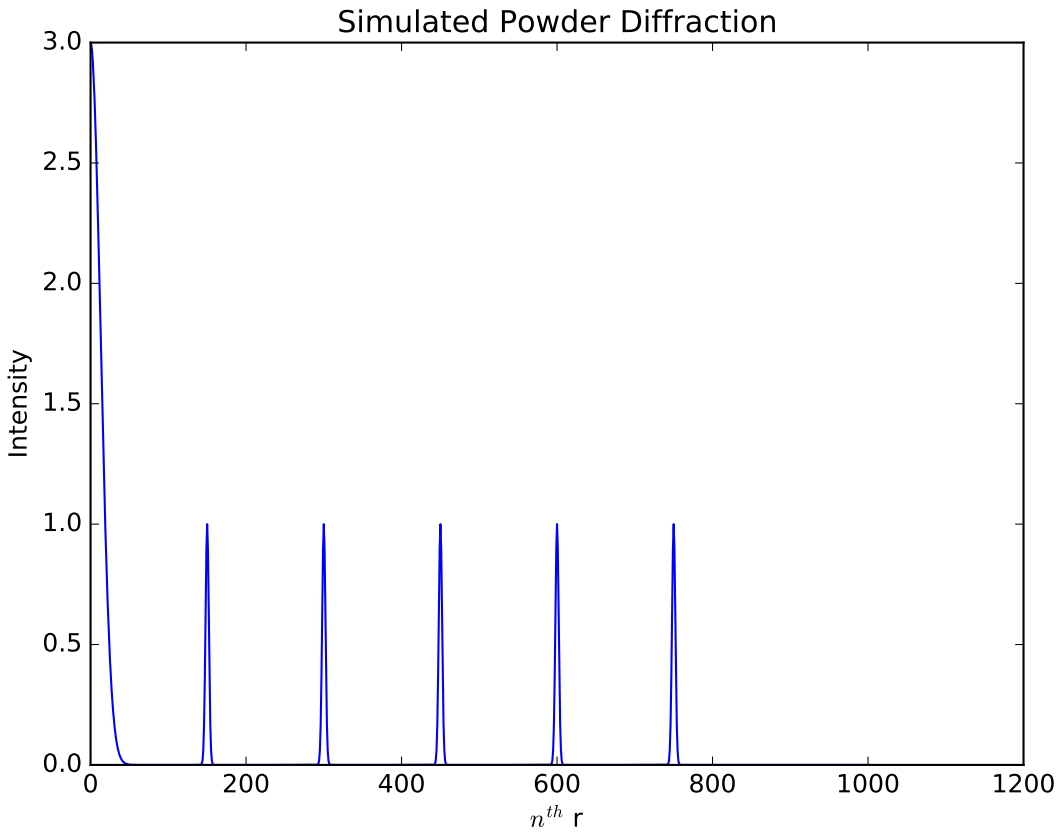
in Python code this is *SAXS.calc_theta()*:

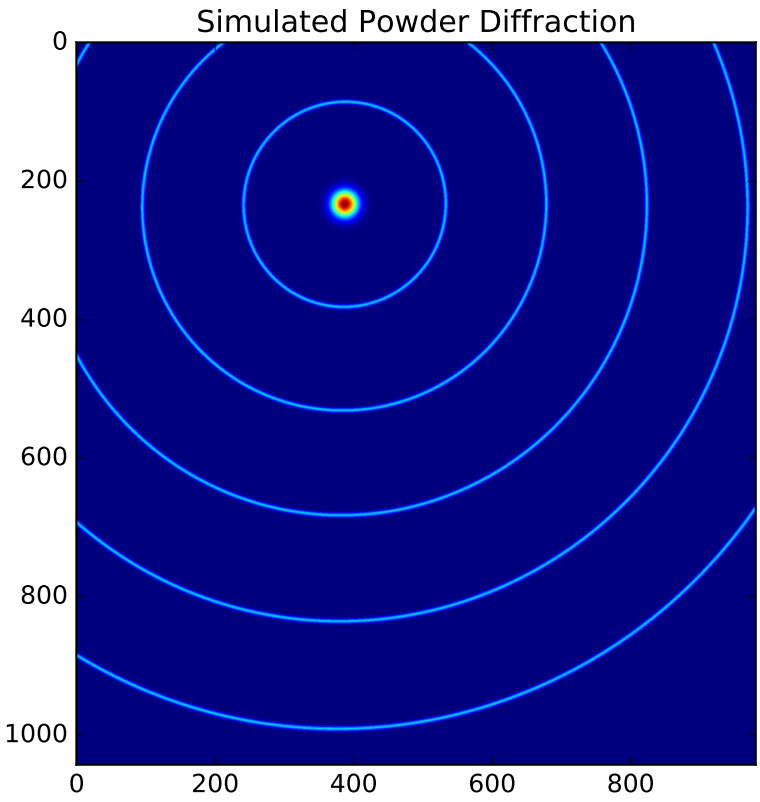
```
def calc_theta(r,theta,d,tilt,tiltdir):
    alpha=np.arcsin(np.sin(tilt)*np.sin(theta+tiltdir))
    lsquared=d**2 +r**2 -2*d*r*np.cos(np.pi/2+alpha)
    return np.arccos(-(r**2-lsquared-d**2)/(2*np.sqrt(lsquared)*d))
```

This angle θ then is calculated for every sub pixel in the sensor. This number then can be rescaled and rounded to the nearest integer in order to get unique integer labels for all the pixels. This labels are the index of the radial interval.

5.3.1 Tilt Angle Correction Test

To check if the tilt angle correction is working, lets create some fake calibration data, with the following peaks in the diffraction curve:





This was done with this configuration file:

```
{
    "Title": "Example Calibration",
    "Geometry": {
        "Tilt": {
            "TiltAngleDeg": -10,
            "TiltRotDeg": 73.569
        },
        "DedectorDistanceMM": 1031.657,
        "BeamCenter": [
            808.37,
            387.772
        ],
        "Imagesize": [
            1043,
            981
        ],
        "PixelSizeMicroM": [
            172.0,
            172.0
        ]
    },
    "Directory": [
        "."
    ],
    "Masks": [
        {
            "PixelPerRadialElement": 1,
            "MaskFile": "emptymask.tif",
            "Oversampling": 2,
```

(continues on next page)

(continued from previous page)

```
            "Name": "",
            "Phi-mode": false,
            "qStart": 0.0,
            "qStop": 5.0
        }],
    "Slices": [],
    "GISAXSmode": false,
    "Wavelength": 1.54,
    "Live-Filelisting": true,
    "OverwriteFiles": true,
    "Threads": 2
}
```

Which amounts to a large tilt, lets see what Fit2d makes of it

```
INFO: SOLUTION 2
INFO: Best fit beam centre (X/Y mm) =   66.78356      138.9544
INFO: Best fit beam centre (X/Y pixels) =   388.2765      807.8745
INFO: Cone  1 best fit 2 theta angle (degrees) =   1.392646
INFO: Cone  2 best fit 2 theta angle (degrees) =   2.780810
INFO: Cone  3 best fit 2 theta angle (degrees) =   4.168858
INFO: Cone  4 best fit 2 theta angle (degrees) =   5.556975
INFO: Cone  5 best fit 2 theta angle (degrees) =   6.944765
INFO: Best fit angle of tilt plane rotation (degrees) =   73.59509
INFO: Best fit angle of tilt (degrees) =  -10.00601
INFO: Estimated coordinate radial position error (mm) =  0.7136476E-02
INFO: Estimated coordinate radial position error (X pixels) =  0.4149114E-01
```

Seems OK.

5.4 Polarization Correction

The polarization correction is expected to be small at small angles, but it is deemed important.

$$I_{cor} = I_j \left[P(1 - (sin(\phi)sin(2\theta))^2)(1 - P)(1 - (cos(\phi)sin(2\theta))^2) \right]$$

where ϕ is the azimuthal angle on the detector surface (defined here clockwise, 0 at 12 o'clock) 2θ the scattering angle, and P the fraction of incident radiation polarized in the horizontal plane (azimuthal angle of 90°) The polarization correction is configured by two parameters in *PolarizationCorrection*. Its factors are included in the integration matrix (operator).

This input:

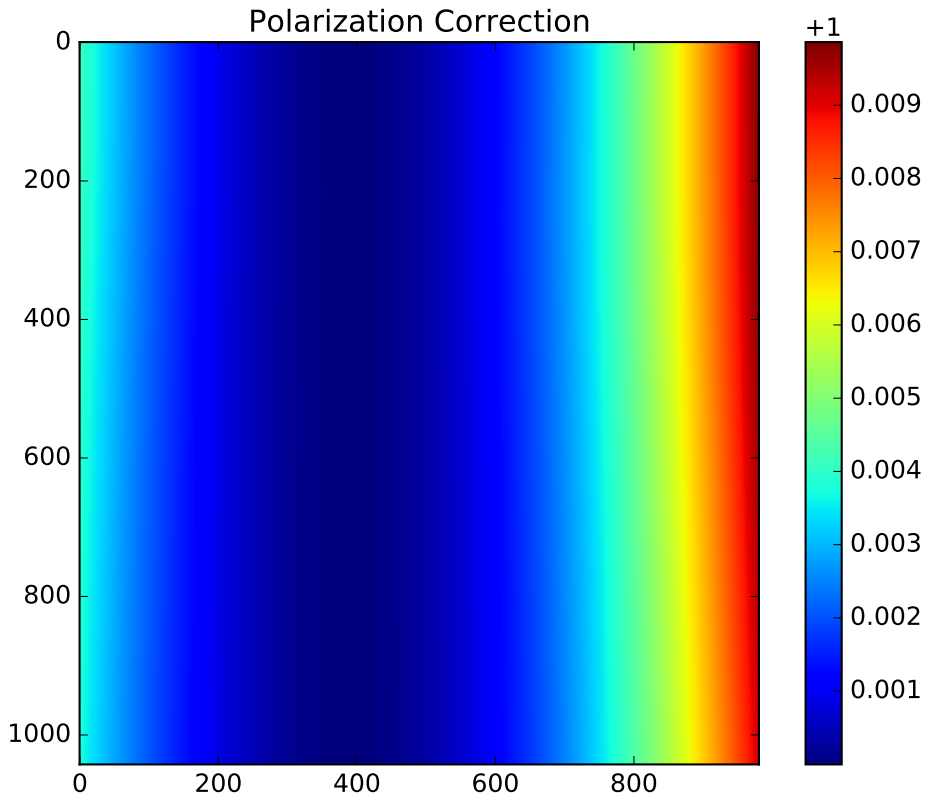
```
{
    "Title": "Example Calibration",
    "Geometry": {
        "Tilt": {
            "TiltAngleDeg": -10,
            "TiltRotDeg": 73.569
        },
        "DedectorDistanceMM": 1031.657,
        "BeamCenter": [
            808.37,
            387.772
        ],
        "Imagesize": [
            1043,
            981
        ],
        "PixelSizeMicroM": [
```

(continues on next page)

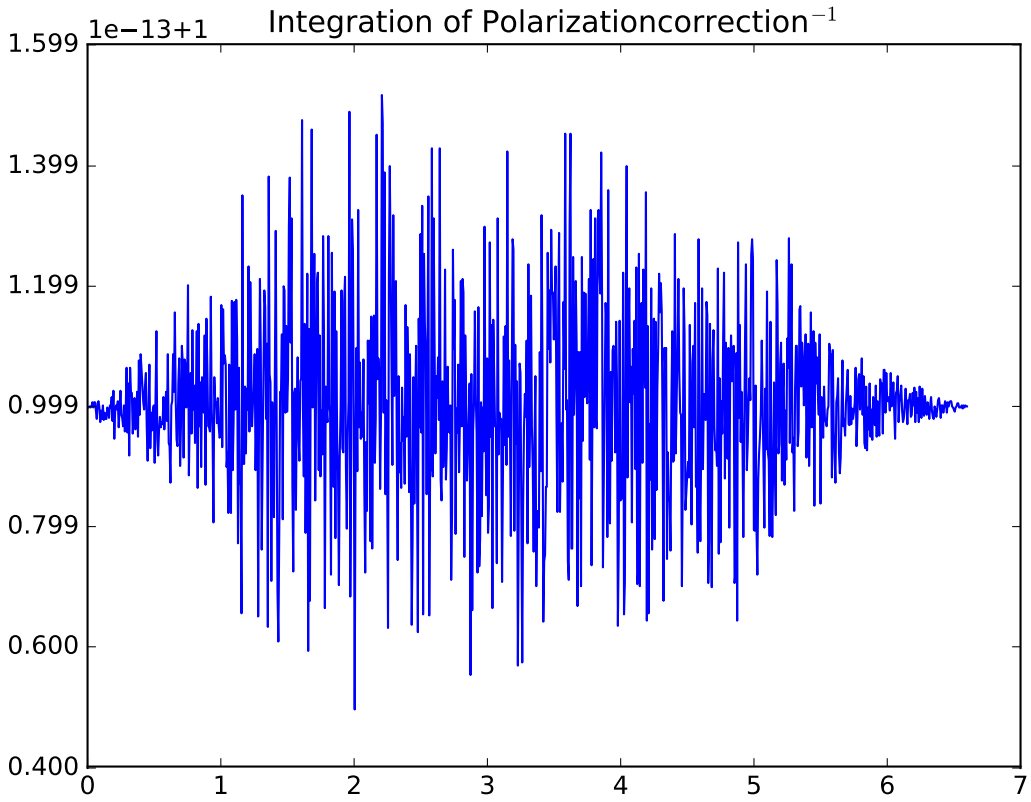
(continued from previous page)

```
            172.0,
            172.0
        ]
    },
    "Directory": [
        "."
    ],
    "PolarizationCorrection": {
        "Angle": 0,
        "Fraction": 1
    },
    "Masks": [
        {
            "PixelPerRadialElement": 1,
            "MaskFile": "emptymask.tif",
            "Oversampling": 2,
            "Name": "",
            "Phi-mode": false,
            "qStart": 0.0,
            "qStop": 5.0
        }],
    "Slices": [],
    "GISAXSmode": false,
    "Wavelength": 1.54,
    "Live-Filelisting": true,
    "OverwriteFiles": true,
    "Threads": 2
}
```

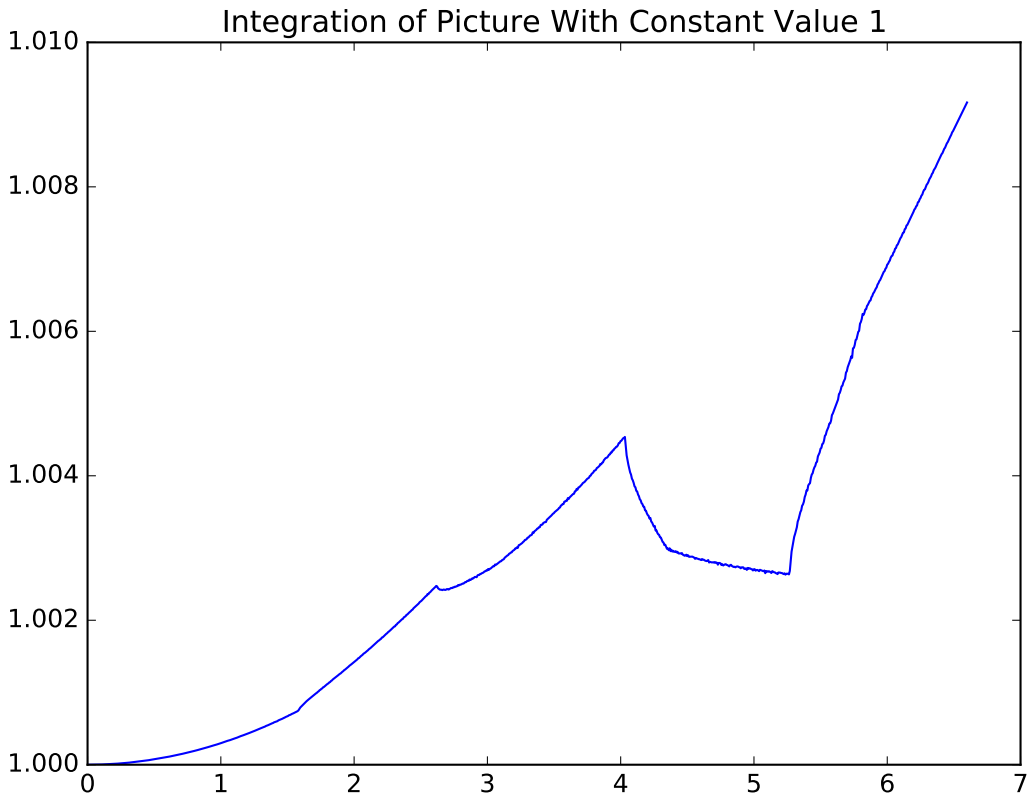
Gives:



If the correction factors are all correctly in the algorithm, the integration of an image containing $1/I_{corr}$ should give constant 1.0.



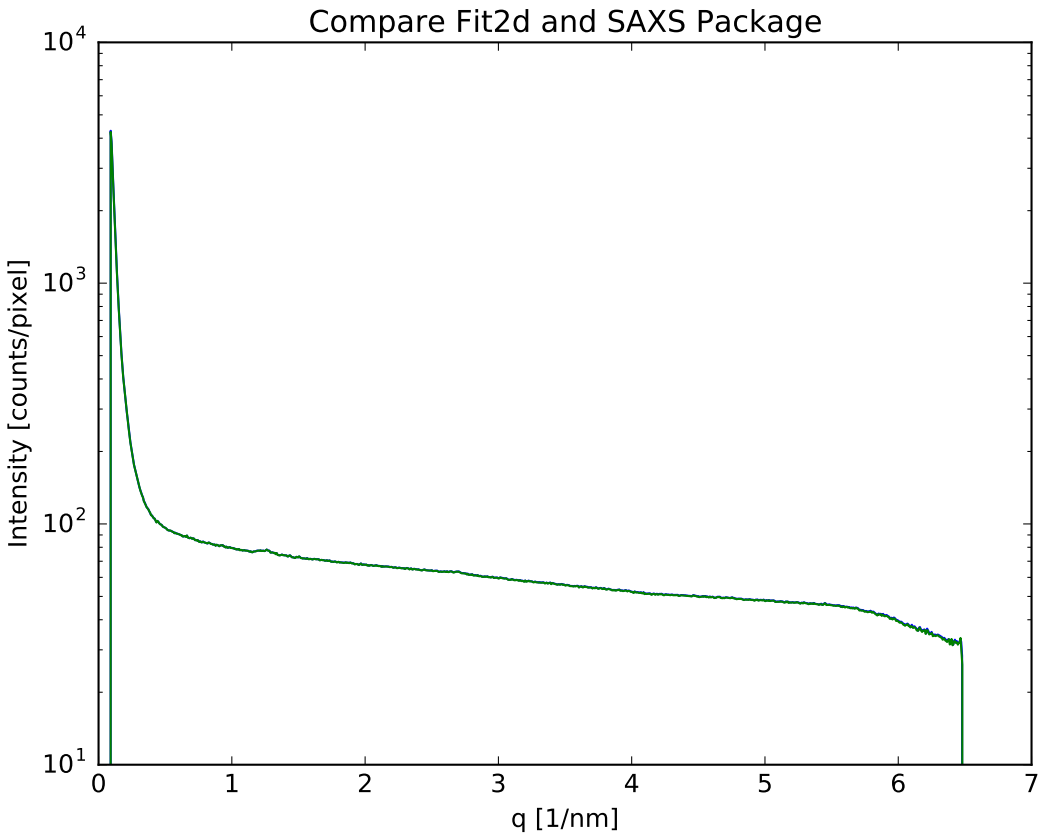
Just for checking: integrating a picture with only ones gives something different:



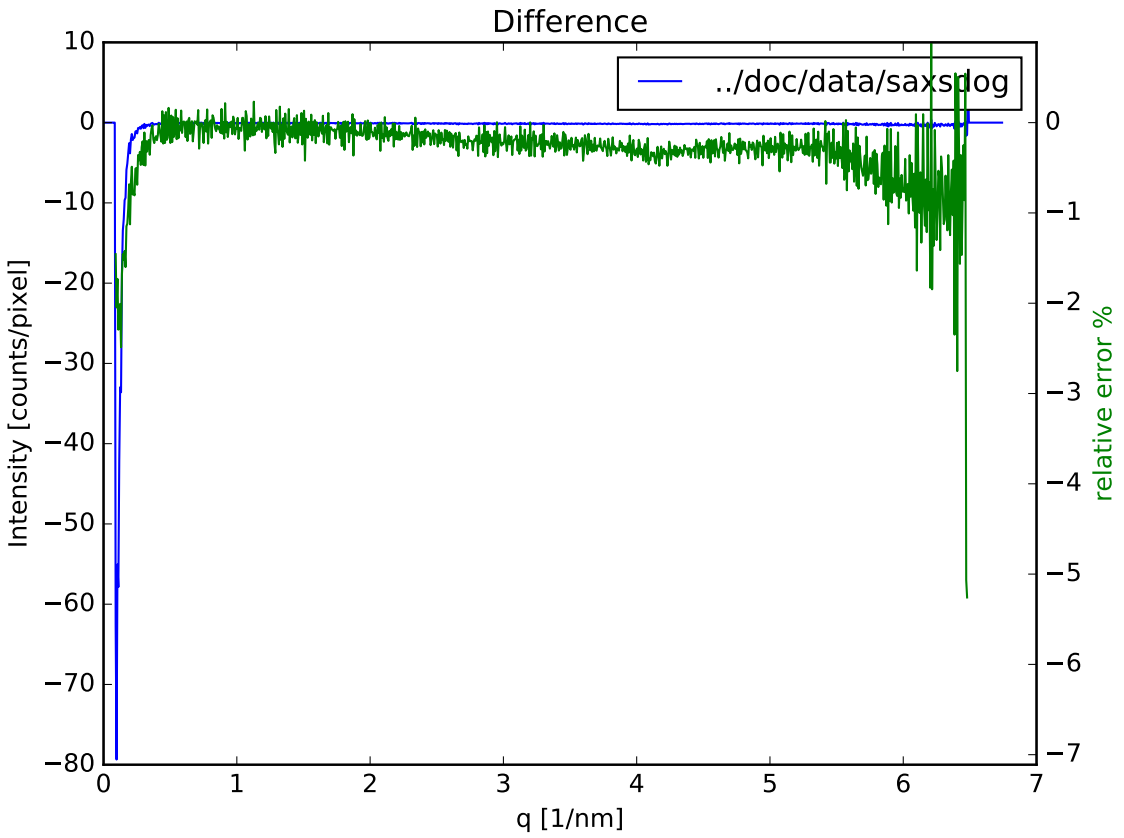
This are the wiggles that come from the polarization correction pattern

5.5 Compare With Fit2d

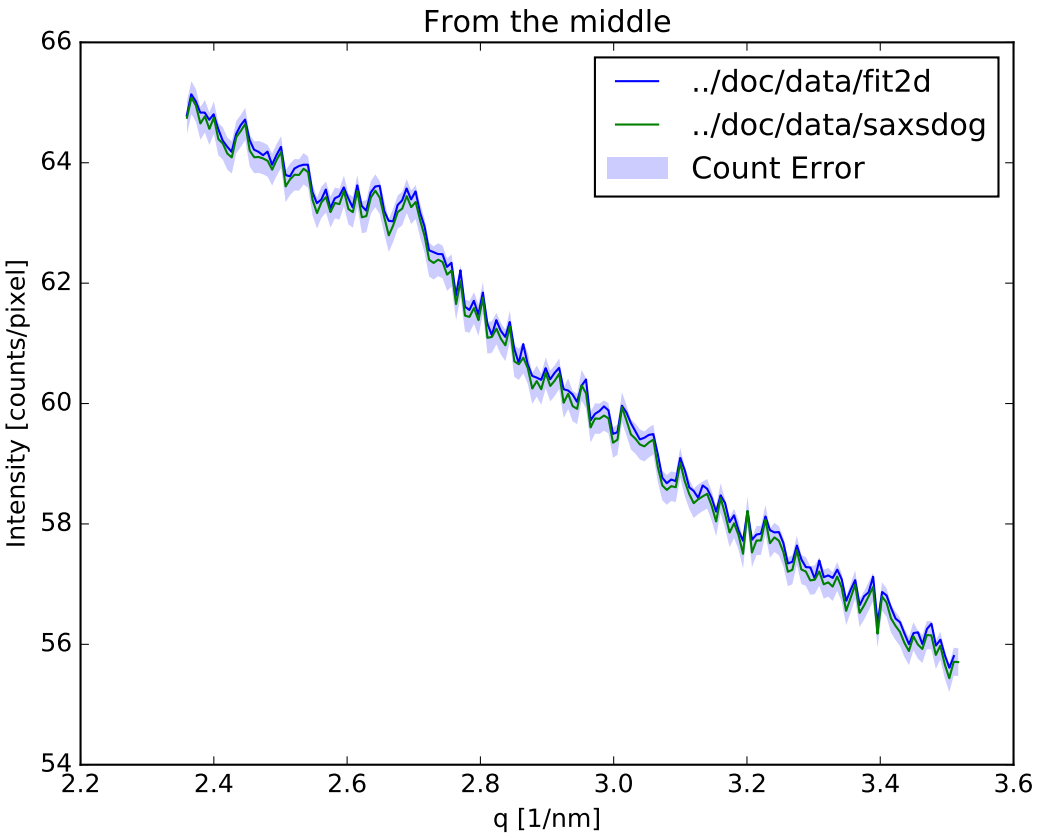
The program fit2d, which this package aims to partly replace, is the standard, so we better include a comparison plot here:

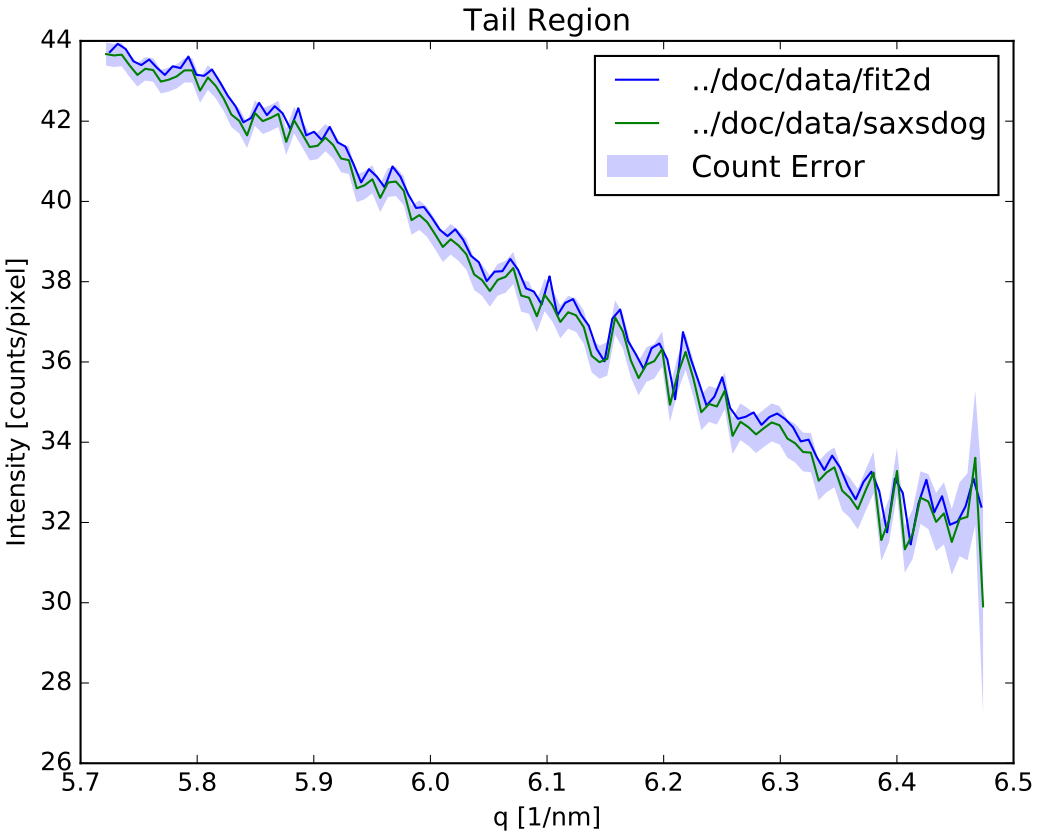


Okay, this doesn't show much but if we plot the difference:

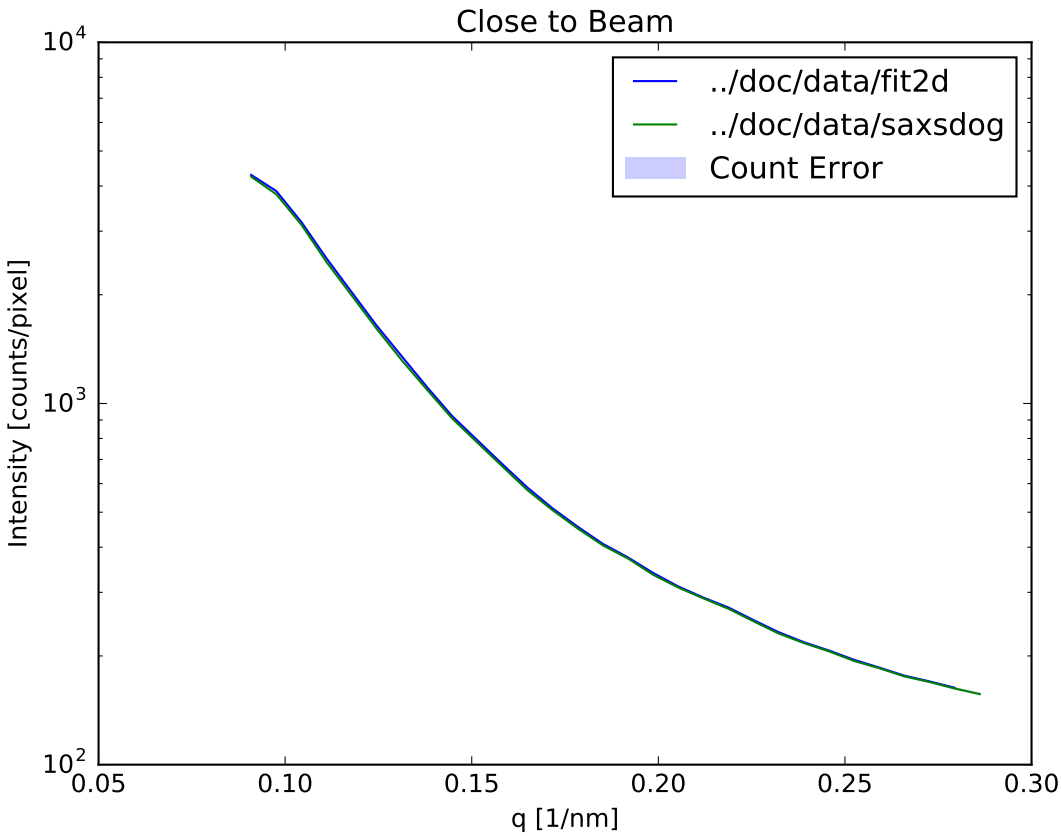


Still looks okay.


From the middle
../doc/data/fit2d
../doc/data/saxsdog
Count Error
Intensity [counts/pixel]
q [1/nm]
66
64
62
60
58
56
54
2.2
2.4
2.6
2.8
3.0
3.2
3.4
3.6


Tail Region
../doc/data/fit2d
../doc/data/saxsdog
Count Error
Intensity [counts/pixel]
q [1/nm]
44
42
40
38
36
34
32
30
28
26
5.7
5.8
5.9
6.0
6.1
6.2
6.3
6.4
6.5

In the tail region the blue halo (Poisson error) signifies that there are not enough counts to make good statistics.



5.6 Integrating a Constant Image With Masked Values

This test shows that nothing wrong happens at mask borders. For those we want to integrate an image that is one everywhere except for the masked regions

We use the following calibration without Polarization correction and mask:

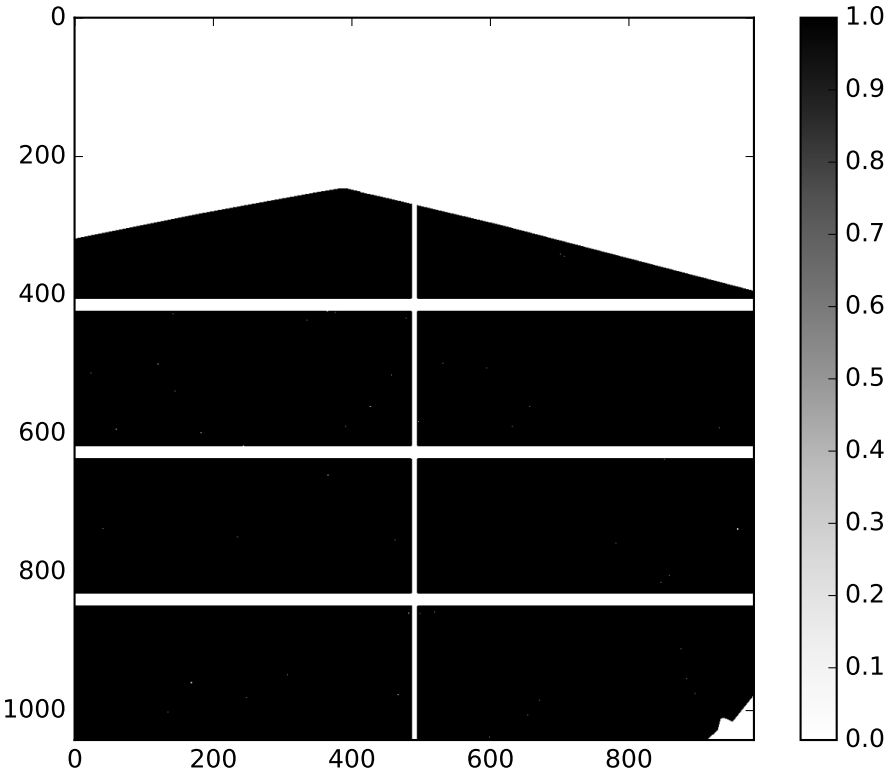
```
{
    "Title": "Example Calibration",
    "Geometry": {
        "Tilt": {
            "TiltAngleDeg": -0.56,
            "TiltRotDeg": 73.569
        },
        "DedectorDistanceMM": 1031.657,
        "BeamCenter": [
            808.37,
            387.772
        ],
        "Imagesize": [
            1043,
            981
        ],
        "PixelSizeMicroM": [
            172.0,
            172.0
        ]
    },
```

(continues on next page)

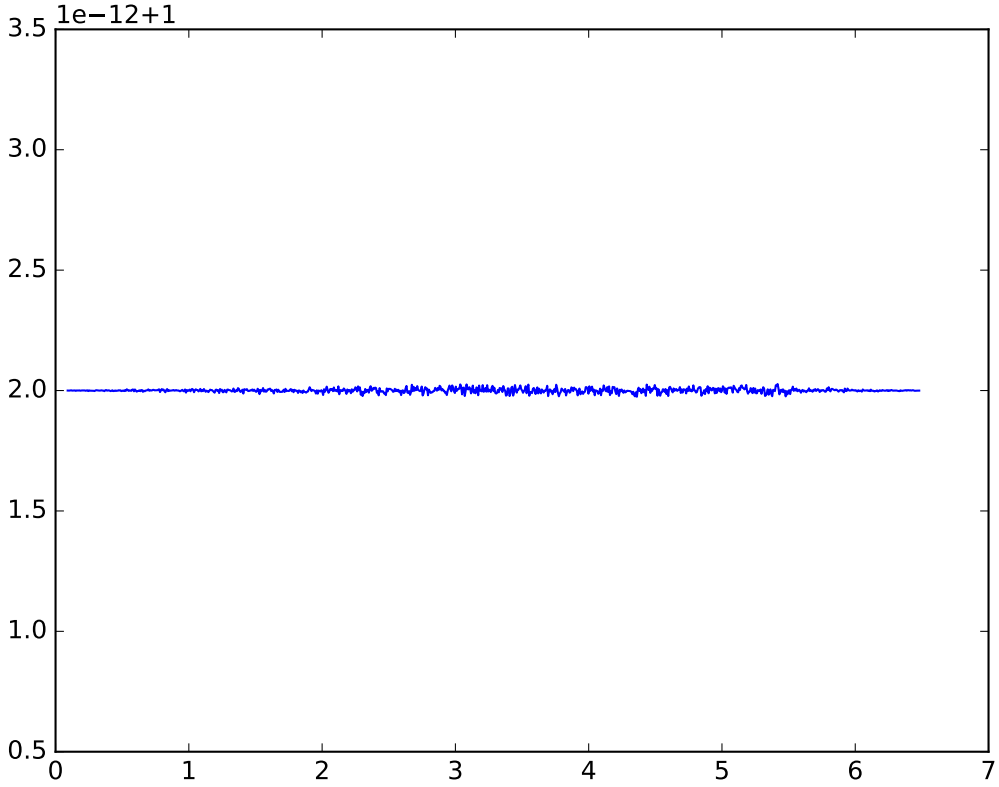
(continued from previous page)

```
    "Directory": [
        "."
    ],
    "Masks": [
        {
            "PixelPerRadialElement": 1,
            "MaskFile": "data/AAA_integ.msk",
            "Oversampling": 3,
            "Name": "",
            "Phi-mode": false,
            "qStart": 0.0,
            "qStop": 5.0
        }],
    "Slices": [],
    "GISAXSmode": false,
    "Wavelength": 1.54,
    "Live-Filelisting": true,
    "OverwriteFiles": true,
    "Threads": 2
}
```

The image we are going to integrate is exactly the array the *SAXS.openmask()* returns:



The result is constant 1 (where the intensity is not 0), save 2e-12.



5.7 Statistics

5.7.1 Poisson Statistics

The most important error is the statistical fluctuation that stems from the randomness of the scattering events. Counts of such events follow the Poison distribution. Such, the error (σ) is $\sqrt{n}$ for a count of n. The result of which is, that the relative error $\frac{\sqrt{n}}{n}$ rapidly gets small for larger counts.

Each Pixel in the SAXS sensor counts the number of events, and thus follows the Poisson statistics. The error of a sum of pixels is calculated as.

$$\sigma_{sum} = \sqrt{\sum_i \sigma_i^2}$$

which means here

$$\sigma_{sum} = \sqrt{\sum_i n_i}$$

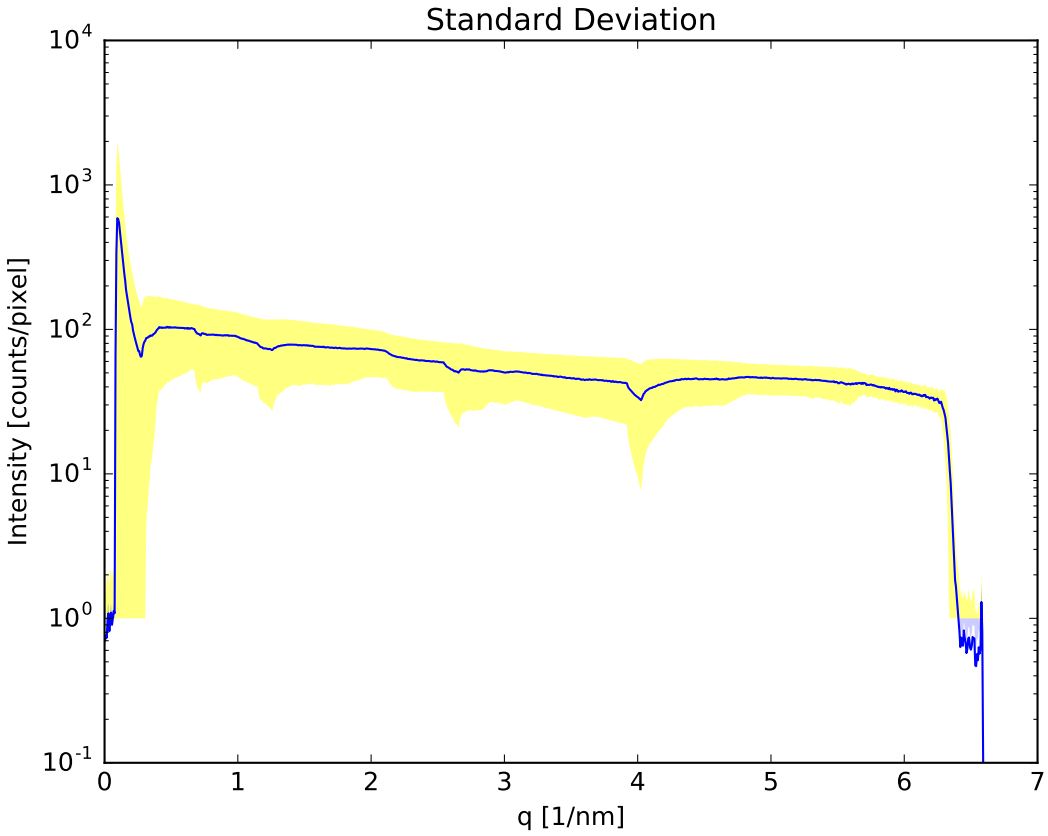
Rescaled over the number of pixels (P) in the sum this gives:

$$\sigma_{sum} = \frac{\sqrt{\sum_{i=1}^{P} n_i}}{P}$$

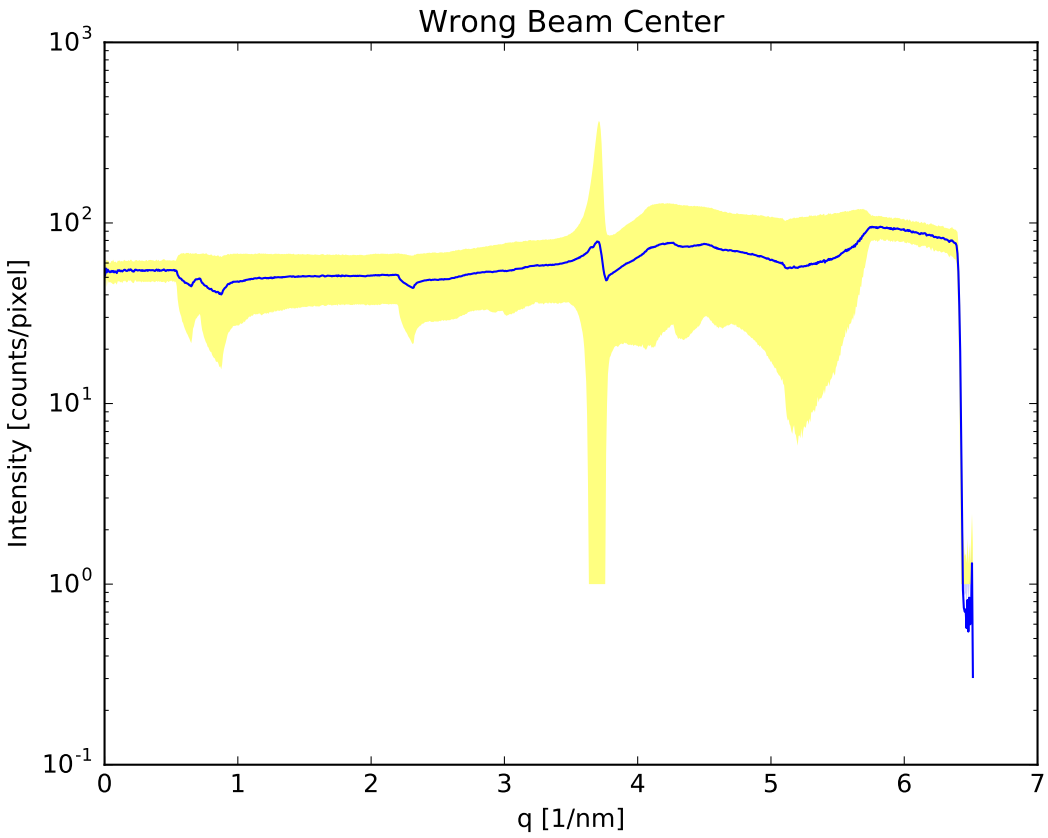
The `SAXS.calibration.plot()` method of the `SAXS.calibration` class will give you the Poisson error along with the standard deviation. So for regions, where the total number of counts is too small, you can see if there is a significant error. This might occur, if too few pixels are used for a data point or the intensity is just to small.

5.7.2 Standard Deviation

The standard deviation of the mean that is taken through the integration is not as such particularly useful to estimate the error of the resulting intensities because there are quite a few things that produce an angle dependence. In an optimal case, if the angle dependence can be corrected with the Polarization correction, the standard deviation of the integration might be very small. In an ordinary case the standard deviation gives you a measure of how spread the intensities within a radius interval are.



The standard deviation is bright yellow and the Poison error is blueisch



If the calibration is wrong you will for example see in the standard deviation. Like in this example. Here the beam center is wrong.

5.8 GISAXS Slices

In grasing incidend SAXS looking at radial integration doesnt make much sence. In GISAXS you rather want to look at horizontal or vertical sclices. That is horizontal or vertical with respect to the scattering surface.

The SAXSDog tools allow to specify *Slices* of pixels and allow to look at them at the q_y, q_z scale as they are used in GISAX analysis.

The `slice` class implements this functionality very much the same as the radial integration except that the labels of the pixels simply are the integer x coordinate in pixels when we want a slice in x direction and the y coordinate when we want the slice in y direction. Oversampling doesnt make any sense in this scenario but the rest is the same. The only subtile thing is to calulate the q_y and q_z scale. Because the dedector coorinatesystem may have the x or the y axis aligned with the scattering suface the *Slices* allows to chose wether we are in plane or verical to the scattering surface. If you want to get correct q_z values you must also specify the incident angle α_i, (*IncidentAngle*).

CHAPTER

SIX

SAXSDOG DEVELOPER DOCUMENTATION

6.1 The Leash

The Leash GUI is part of the SAXSdog Network and the most user facing software. Its design allows to extend it with functionality by touching only small well defined parts. One is the Schemas of the configuration files and the other is the `QItemDelegate`. The following sections give an overview of the important parts.

6.1.1 Using the JSON Schema to Extend Leash

This software relies a lot on structured configuration files that have to constantly be checked for validity. This is done by defining the grammar in JSON Schema. This is a language, in its self expressed in JSON, made to specify what values may occur where in the file. This does not only allow for automatically generating documentation, as it is used in this document many times, but you can also use it to generate an GUI that can edit this structured data files.

The tree view in the "Calib" tab of `Leash` is build by recursively, going to the schema and the data, building the model that can be displayed in the QtreeView widget.

So, in order to add new parameters to the view, the only thing you must do, is to add the description to the schema. If you use similar constructs as in the rest of the data, it will work just so.

The leash uses the scheme in `SAXS/schema.json` to build the "Calib" tab and `SAXS/DataConsolidationConf.json` to build the Consolidate tab.

Consider this excerpt of the `SAXS/schema.json`:

```
{
"type": "object",
"$schema": "http://json-schema.org/draft-03/schema",
"required": true,
"description": "The SAXS configuration file specifies the parameters of a SAXS␣
↪sensor calibration. It is written in the JSON format which governs the general␣
↪syntax.",
"additionalProperties": false,
"properties":
   {
      "Directory":
      {
         "description":"Directory to take into acount for processing images. Given␣
↪as a list of subdirectories.",
         "type": "array",
         "required": true,
         "minItems": 1,
         "items":
         {
            "type": "string",
            "default": ".",
            "appinfo":
            {
```

(continues on next page)

(continued from previous page)

```
                "editor": "RemoteDir"
            }
        }
    }
}
```

The `type` in the root says `object` which means it is a dictionary like data structure. The possible keys are declared in the `properties` dictionary. Inside the `properties` other nested types are declared in this case an `array` called `Directory`.

The `appinfo` section is not part of the schema specification it is rather a custom field to tell the application about possible special treatments. In this case we want the Leash use editor widgets to pick a remote directory. This editor widget will be provided by the `QItemDelegate`. Which shall be explained in the following section.

6.1.2 The QItemDelegate

The `QtGui.QTreeView()` class allows to set an item delegate. This happens for example in `calibeditor.py`

```
self.treeview.setItemDelegateForColumn(1,calibeditdelegate.calibEditDelegate( app␣
↪))
```

The `calibeditdelegate.calibEditDelegate` class in turn is a custom class derived from QtGui.QItemDelegate. This is implemented in `calibeditdelegate.py`.

The constructor is initializing the base class:

```
class calibEditDelegate(QtGui.QItemDelegate):
    def __init__(self,app,  parent=None):
        super(calibEditDelegate, self).__init__(parent)
        self.app=app
```

The part that is interesting is the reimplementations of the `createEditor`, `commitAndCloseEditor`, `setEditorData` and `setModelData` methods.

The `createEditor` method is called when the user double clicks an item content cell in the tree view. The default behavior is to make the text content editable but you can return any widget you like, depending on the context.

```
def createEditor(self, parent, option, index):
    """
    special method of QItemDelegate class
    """
    try:
        subschema=json.loads(unicode(index.model().data(index,role=im.SUBSCHEMA).
↪toString()))
    except ValueError:
        return None

    type= unicode(index.model().data(index,role=im.TYPE).toString())
    editablearray= unicode(index.model().data(index,role=im.ISEDITABLEARRAY).
↪toString())
    editortype=None
    if subschema.get("appinfo"):
        editortype= subschema.get("appinfo").get("editor")

    print type
    if "enum" in subschema:
        isenum="true"
        enum=subschema['enum']
```

(continues on next page)

(continued from previous page)

```
    else:
        isenum="false"

    if type == "integer":
        spinbox = QtGui.QSpinBox(parent)
        spinbox.setRange(-200000, 200000)
        spinbox.setSingleStep(1)
        spinbox.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignVCenter)
        return spinbox
    elif type == "number":
        spinbox = QtGui.QDoubleSpinBox(parent)
        spinbox.setRange(-200000, 200000)
        spinbox.setSingleStep(0.1)
        spinbox.setDecimals(4)
        spinbox.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignVCenter)
        return spinbox
    elif editablearray=="editablearray":
        arrayeditdialog=arrayediddialog(index,parent)
        return arrayeditdialog
    elif  type== "object" or type=="array" or type=="arrayitem" :
        return None
    elif isenum=="true":
        combobox = QtGui.QComboBox(parent)
        combobox.addItems(sorted( enum))
        return combobox
    elif editortype=="File":
        dirname= os.path.dirname(unicode(index.model().filename))
        filepicker=QtGui.QFileDialog(directory=dirname)
        filepicker.setMinimumSize(800,500)
        filepicker.setFileMode(filepicker.ExistingFile)
        return filepicker
    elif editortype=="RemoteDir":
        return RemoteDirPicker(self.app,parent,index)
    elif editortype=="RemoteFile":
        return RemoteDirPicker(self.app,parent,index,showfiles=True)
    else:
        return QtGui.QItemDelegate.createEditor(self, parent, option,
                                                index)
```

The `createEditor` is called with an index object. Which is a class that is used by the `QtGui.QStandardItemModel` class to represent the data in a form the tree view can display it.

In the implementation in `jsonschematreemodel.py` the items have the subschema describing themselves and their children stored in special data attributes. We can use this to chose which editor to present to the user, depending of the type and role of the item on hand. Integers get a `QtGui.QSpinBox`, Enumerations get `QtGui.QComboBox` to select one of the options.

In case the item has `File` in the appinfo/editor field,

```
{
   "MaskFile":
   {
      "description": "Path of Maskfile",
      "type": "string",
      "default": "AAA_integ.msk",
      "required": true,
      "appinfo":
      {
         "editor": "File",
         "display": "MaskFile"
      }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

double clicking the cell will give the user a file system dialog to select a local file. This time we did not get a small widget that fits into the cell, we got a separate dialog. This means it is possible to launch any kind of fancy dialog from here. Think "mask editor", "powder diffraction calibration" anything you like.

The `"display": "MaskFile"` field will cause another method to execute custom behavior. The `setModelData` method. In this case it will load the mask file and display the picture in another cell in the tree view.

6.2 The Image Queue

The `imagequeue` class manages how and when to integrate images. It is instantiated by the server when you load up a new calibration and start a new queue. Or, alternatively the `saxsdog` command line tool will also create an image queue. It takes as argument a list of integration recipes e.g. radial integration or slices. This recipes can be any Python object that knows how to do something with images as long they implement the `integratechi()` method with the same API as the others.

In the initialization process, it will create a queue object, which is a very powerful synchronized data structure which can even be accessed by subprocesses. If the server creates it it will also create a process to listen to the Feeder service and push the image paths into the queue of the `imagequeue` object.

For the work to begin the `imagequeue.start()` method needs to be called. This will create the worker subprocesses to consume the images from the queue.

```
for threadid in range(1,self.options.threads):
    print "start proc [",threadid,"]"
    worker=Process(target=funcworker, args=(self,threadid))
    worker.daemon=True
    self.pool.append(worker)
    worker.start()
```

The imagequeue will launch and manage as many workers as configured in the calibration. The workers are in an infinite loop where they wait until a new image arrives through the queue to decide whether they are configured to work on the directory the images are in. If so they will process the image and push a small report into the history queue. This report includes the time (for the histogram) and the files written.

If the `readdir` command is issued to the server, it will call the `imagequeue.fillqueuewithexistingfiles()` method which will fill the queue with all ".tif" files it finds in the configured directory.

6.3 Data Consolidation or Datamerger

After the images are integrated and the measurements are done there remain a few data consolidation tasks. The `datamerge` module provides this functionality. The *Saxsdmerge* commandline tool is one way to use it, the other is via the `Leash`. Its main goal is to merge the logfiles with the parameters logged in the images and dedector logs.

The result is a table where each images has one row and the collumns are all the available parameters fom continuous logs or logs that log not regularly but only if an image is requested.

This is where future versions may include a compleete HDF export of all relevant data.

CHAPTER

SEVEN

SAXS MODULE API

class SAXS.**calibration**(*config*, *mask=None*, *attachment=None*)

This class represents a calibration for SAXS diffraction. After initialization, the `integrate()` method can compute the radial intensity very fast.

Parameters

- **config** (*dict*) – the calibration json object as dictionary *Calibration file Reference*:
- **mask** (*dict*) – The mask of the list to use for this instance
- **attachment** (*dict*) – If mask is not to be read from filesystem it must be provided as attachment

integParameters(*Intensity*)

function to calculate the integral parameters between qStart and qStop returns I0, I1, I2

integrate(*image*)

Integrate a picture.

Parameters **image** (*numpy.array(dim=2)*) – Sensor image to integrate as 2d *NumPy* array

Returns Returns Angle and intensity vector as a tuple (angle,intensity)

integratechi(*image*, *path*, *picture*)

Integrate and save to file in "chi" format.

Parameters

- **image** (*np.array()*) – Image to integrate as numpy array
- **path** (*string*) – Path to save the file to

Returns Scattering curve data as numpy array

integrateerror(*image*)

Integrates an image and computes error estimates.

Parameters **image** (*np.array()*) – Image to integrate as numpy array

Returns The intensity the standard deviation and the Poisson statistics error in a numpy array.

plot(*image*, *outputfile=''*, *startplotat=0*, *fig=None*)

Plot integrated function for image in argument.

Parameters

- **image** (*numpy.array(dim=2)*) – Sensor image to integrate as 2d *NumPy* array
- **outputfile** (*string*) – File to write plot to. Might be any image format supported by matplotlib.
- **startplotat** (*integer*) – radial point from which to start the plot

polcorr(*Pfrac*, *rot*)
Polarization Correction

class SAXS.**slice**(*conf*, *sliceconf*, *attachments=[]*)
Initializes slice integration

Parameters

- **conf** (*dictionary*) – Detector calibration
- **sliceconf** (*dictionary*) – The part of detector calibration which deals with this slice.
- **attachments** (*dictionary*) – Mask data.

integratechi(*image*, *path*, *picture*)
Integrate and save to file in "chi" format.

Parameters

- **image** (*np.array()*) – Image to integrate as numpy array
- **path** (*string*) – Path to save the file to

Returns Scattering curve data as json structure

plot(*image*, *outputfile=''*, *startplotat=0*, *fig=None*)
dummy function in order to not trip up image queue

SAXS.**calc_theta**(*r*, *phi*, *d*, *tilt*, *tiltdir*)
Calculates the difraction angle from pixel coordinates. It does work when called with arrays. See *The Geometry*

Parameters

- **r** (*float*) – Distance to beamcenter.
- **phi** (*float*) – Angle[rad] from polar sensor plane coordinates.
- **d** (*float*) – distance to difraction center.
- **tilt** (*float*) – Angle[rad] of sensor plane tilt.
- **tiltdir** (*float*) – Angle[rad] of direction of tilt.

Returns theta

SAXS.**scalemat**(*Xsize*, *Ysize*, *ov*)
Computes a scaling projection for use in computing the pixel weights for integration

Parameters

- **Xsize** (*int*) – Picture size in X direction.
- **Ysize** (*int*) – Picture size in Ydirection.
- **ov** (*int*) – Number of oversampling ticks in x ynd y direction
- **corr** (*array*) – Polarizationn an other correction factors

Returns sparce matrix toing the scaling

SAXS.**openmask**(*mfile*, *attachment=None*)
Open the mask file especialy the *.msk file. Unfortunately there is no library module for msk files available also no documentation. So, for the msk file, we have a very brittle hack it works for our sensor. Nevermind any other resolution or size.

Parameters **config** (*object*) – Calibration config object.

Returns Mask as logical numpy array.

SAXS.**convert**()
This implements the functionality oft *The Converter*. It parses the commandline options and converts the Fit2d info file to the JSON data used by the SAXS.callibration class.

SAXS.**saxsdog**()
This implements the functionality of *The Saxsdog*

class SAXS.**imagequeue**(*Cals*, *options*, *directory*, *conf*)
This class keeps a queue of images which may be worked on in threads.

Parameters

- **Cal** (SAXS.calibration) – The SAXS Calibration to use for the processing
- **options** (*optparser*) – The object with the comandline options of the saxsdog
- **args** (*list*) – List of command line options

fillqueuewithexistingfiles()
Fill the queue with the list of images that is already there.

start()
Start threads and directory observer.

class SAXS.**Server**(*conf*, *serverid*, *stopflag=None*, *serverdir=None*)
class to manage a saxsdog server

authenticate(*data*)
check signature of request

commandhandler(*object*, *attachment*)

start()
start server loop

start_image_queue(*object*, *attachment*)
prepare new image queue start processing threads

SAXS.**initcommand**(*options*, *arg*, *conf*)
Interface for issuing leash commands

SAXS.**validateResponse**(*message*)
Validate response from saxsdog server against the schema.

Get this guide as PDF.