Small-Angle Scattering kit for Interpretation and Analysis (SASkia)

Almut Stribeck

July 2018

1 Introduction

1.1 Preamble and motivation

When I started to write the program, I had only little knowledge of Python. Therefore it is expected that the coding is not on an expert level. SASkia is a computer program with a focus on the analysis of **small-angle scattering** curves, although several methods[1] are already dealing with multidimensional data. The code and parts of it are free to use in other programs. **No guarantee, no liability!** Like many computer programs in science, SASkia is continuously extended according to the needs of the developer. The program is made available to trained scientists. Instead of a user manual this handout and some examples are given to help the user to learn **how to use the program**. Feedback is appreciated.

1.2 SASkia's habitat

SASkia is written in **Python**. The design goal is to use only standard Python packages except for the command interpreter[2]. The work has been done under **Linux** (Kubuntu and CentOS) using Python 2.7 and Python 3.6. A **Microsoft Windows 10** environment has been tested with Anaconda2 (Python 2.7), Anaconda3 (Python 3.6) and several previous Python 3 versions. In doing so several implementation errors had been encountered. The last[3] of the critical errors in Python for Windows was only corrected in version Python 3.7.0. **Therefore, under Microsoft Windows it is recommended to run SASkia under Python 3.7.0**[4]. As long as Anaconda does not support this version, a plain native installation of Python 3.7.0 does the job.

[1] Only some of these methods are supplied. They do not reside in the directory `saskia` or one of its subdirectories, but in a parallel directory "`devel`".

[2] The command interpreter is based on the package cmd2

[3] http://bugs.Python.org/issue31546 PyOS_InputHook is not called when waiting for input()

[4] The fundamental functions of SASkia (commands for the evaluation of curves; a "graphic" command which only opens one single x-y-plot window at a time) should not be affected by the remnant implementation error. Affected are scripts which permit to have several graphical windows open at the same time.

1.3 Its data formats

The native data formats of SASkia are human readable[5]. Some more data formats are supported from the beginning[6]. I can offer implementation help for further formats, if a description of the format and sample files are provided.

1.4 Functionality of SASkia

Commands, Largo scripts, and Python scripts. Aiming at the processing of big data SASkia is controlled by commands. Commands provide *basic* operations on curves. Many tasks of scientific relevance require to carry out a *sequence of commands*. Such sequences are provided by LARGO scripts. LARGO scripts are text files which can easily be adapted to special conditions of the data[7]. Tasks which require even deeper knowledge of a special experimental setup are served by Python scripts. Processing of 2D scattering patterns is also provided by Python scripts – at this stage of program development.

LARGO[8]-scripts concatenate a sequence of commands and/or other scripts. They define a "super command". The call of a LARGO script can pass arguments to the contained commands and scripts. LARGO scripts may be nested. A typical LARGO script `corrfun_lam.lrg` computes, e.g., the correlation function from the isotropic scattering curve supplied as a text file, displays it and keeps it in operand 1 (OP1) of saskia. Call:

```
saskia> @corrfun_lam nmbsy08_0001.dat
```

More versatile than LARGO scripts are Python scripts. They can also be called from the command line. Python scripts require programming skills. They can access the native data of SASkia. They can call commands. They cannot call LARGO scripts. They can implant their own data structures in SASkia. A typical Python script reads all raw data related to an experimental run together with its environmental data and writes for each frame a curve file which contains the curve together with environmental attributes. Call:

```
saskia> @@mergecAT,Y023_up_000 sy02_up saxs
```

`mergecAT`: merge curves written at "A"lba in a run related to "T"emperature variation. The first argument "Y023_up_000" is the run name. The second argument "sy02_up" is the file prefix for the output files. The third argument "saxs" tells the script to use the data of the SAXS detector. By this last feature the script considers the fact, that during the respective beam time the scattering has been monitored by two detectors, namely a SAXS detector and a WAXS detector.

[5] Curves are stored in simple text files with comments and attributes, 2D patterns are stored in JSON text files.

[6] EDF files written at ALBA and ESRF; TIFF files written by MAR detectors; 2D patterns written by the author's PV-WAVE procedures ("`.ima`"); curves in data files written at ALBA.

[7] "Adapting" means fixing some parameters of contained commands and keeping other parameters suppliable.

[8] Load with ARguments and GO

Curve operands inside SASkia. The program saskia provides three standard[9] "operand arrays" (OP0, OP1, OP2) which can contain curves together with their attributes. The saskia commands work on these operands. `readcurve` reads a curve from a file into an operand, `writecurve` writes curves from an operand into a file.

2D scattering patterns inside SASkia. Presently there are no standard operands for 2D scattering patterns in saskia, but there is already a package `sapatt.py` which works with patterns (or even photos). For the display of patterns `sapiawin.py` is provided. It does not limit the number of patterns which may be displayed simultaneously. This interactive concept made patterns vulnerable to shortcomings of the implementations of Python for MS-Windows.

A cmd2 feature: Diving into SASkia at runtime. Whenever Python has the package `ipython` installed, SASkia has also the command `ipy`, which starts an interactive Python session in the console. Then the user can test interactively, what single Python instructions do. Scientists who have worked with PV-WAVE, IDL or matlab A have come to appreciate this feature.

A cmd2 feature: Executing operating system commands. An input line which starts with an exclamation mark "!" is interpreted as the call of a command to the operating system. Under Windows this is, in principle, a DOS command. Under Windows, if you want to rename the file rawdata.dat you would issue

```
saskia> !rename rawdata.dat baddata.dat
```

Under unix the same operation would require the phrase

```
saskia> !mv rawdata.dat baddata.dat
```

Read the documentation of cmd2 to find out more about built-in features before you write your own scripts.

2 Installation the simple way

2.1 MS-Windows computers

2.1.1 The SASkia user

Generate a **user account with a single word** (e.g. "joe", not "joe smith") for your work with Python. This is the first step to avoid file paths which includes blank characters (" "). **Do not make any directory names with blanks!** Otherwise the command processor of SASkia may run crazy.

[9] In the 3 standard operand arrays the number of points is limited to 4096 points (self._maxp in saskia.py may be changed to change the limit). This static definition is conservative and robust. It appears sufficiently suited for the processing of curves.

2.1.2 Install a mature Python version

In July 2018 Python 3.7 is not yet available for Anaconda. Therefore I describe the plain vanilla installation of Python 3.7 in the normal command terminal (`cmd`).

> Under MS-Windows **Python 3.7.0 is mandatory**. Earlier versions of Python for Windows carry implementation errors. Then the use of advanced functions of Python's standard plotting package matplotlib will blow up the program under Windows.

- From `Python.org` download `Python-3.7.0-amd64.exe`, if you run a 64-bit Windows. Open this installer. On the first window that opens be sure to activate the button **Add Python 3.7 to PATH**! Otherwise it will become difficult to start Python. Install Python 3.7.
- Open a console terminal[10] by typing into the task bar: `cmd`.

Check Python. In the console terminal (cmd) issue

```
C:\Users\MYSELF>python --version
Python 3.7.0

C:\Users\MYSELf>
```

O.K. – you have a working Python 3.7.0

2.1.3 Place and expand the distribution archive

The Python directory for program sources. Be sure to have a Python directory[11] on your data volume

```
C:\Users\MYSELF>D:
D:>md python
D:>cd python
D:\python>
```

Place the SASkia distribution. Place the SASkia distribution archive saskiayymm.zip (e.g. saskia1807.zip) in the directory `D:\python\`. Unzip the file ***to*** D:\python. These operations can be done by clicking and drawing the mouse in the graphical file manager[12] Unzipping generates the directory tree for saskia. Check by using the DOS `dir` command

[10] If you have Anaconda installed, the "Anaconda Prompt" window will not work, although it looks almost the same. Anaconda replaces the `%PATH%` variable.

[11] This directory is only for such program code that is of general importance

[12] As soon as you hook it in the GUI window, a message box "Tools for compressed folders" pops up. Click it. Before starting change the folder to which the archive shall be expanded to `D:\python`

```
D:\python>dir
```

you should see two directories

```
22.07.18  19:19  <DIR>  devel
22.07.18  19:19  <DIR>  saskia
22.07.18  18:51  2.4M   saskia1807.zip
22.07.18  19:19  86KB   saskia_handout1807.pdf
```

`saskia` and its subdirectories hold program and general scripts. `devel` is a directory for script development.

2.1.4 "Plan A": Supply required Python packages by batch script

Now some packages and an optimization are still missing. If you are brave and trust in a batch file of mine you can try to do the work using a provided batch file `make_envsaskia.cmd`.

```
D:\python>cd BatchFiles
D:\python\BatchFiles>make_envsaskia
```

If this does not work, you will have to install the required packages manually.

2.1.5 "Plan B": Supply required Python packages manually

- Upgrade Pythons Package installer "pip" by issuing:

  ```
  python -m pip install --upgrade pip .
  ```

- Install packages which are needed for saskia:

  ```
  pip install cmd2
  pip install pygame
  pip install json_tricks
  pip install ipython
  pip install scipy
  ```

If in the function test some packages turn out to be missing, then install them, too!

Optimization. Sure, you want to start saskia simply by writing "saskia" when you are in the project directory which contains your experimental data:

- Verify your search path

  ```
  echo %PATH%
  ```

In the search path there should be a directory like

```
C:\Users\MYSELF\AppData\Local\Programs\Python\Python37\
```

with `MYSELF` replaced by your actual user name. This directory can serve[13] as your "**bin directory**", as it is called by unix people. There you can put own Python-related programs and scripts (in particular the script `saskia.cmd`). If this directory does not exist in your search path, then choose another suitable one or extend the search path by your favorite "bin directory" and place `saskia.cmd` there. In the favorite case, the command

```
copy D:\python\saskia\BatchFiles\saskia.cmd &
%userprofile%\AppData\Local\Programs\Python\Python37\
```

(in a single line, please, and remove the "&") places the saskia.cmd in one of the standard directories in the search path on my Windows computer. Thereafter it is found by the operating system on call from everywhere.

2.1.6 Low-level functionality test

Check, if saskia can be started from its home directory within the "cmd" console terminal (The first start takes quite a long time, because the sources are compiled):

```
D:\python:>cd saskia
D:\python\saskia:>python saskia.py
Retrieving individual configuration from saskia.rc
------------------------------------------------
SASkia>
```

This is the saskia prompt which indicates a successful start of saskia. You leave saskia by the `quit` command. Try the `help` command.

2.1.7 Functionality test

From a console-terminal prompt started by "cmd" issue

```
C:\Users\MYSELF>saskia
Retrieving individual configuration from saskia.rc
------------------------------------------------
SASkia>
```

This should start saskia, as well.

2.1.8 Demonstration of SAXS evaluation reciprocal-to-real space

```
C:\Users\MYSELF\>D:
D:>cd saskia\examples
D:\saskia\examples>saskia @run_examples
```

[13] In fact, every directory in the search path is searched for binaries or batch files. My suggestion is, to keep similar things together, i.e. Python related stuff should be kept together with Python related stuff.

2.2 Linux computers

- If Python 2.7 is not already installed, then use your system package installer to install it. For (K)ubuntu this is "apt", for Centos "yum", Debian has "dpkg". You will know. You may, as well, decide to run SASkia under Python 3.6 or Python 3.7.
- Under Linux the Python directory is suggested to be a direct descentent of the user's home directory. I.e.: as a subdirectory of your home directory make a directory named "Python".

  ```
  ~> mkdir python
  ```

- Put the distribution file in and unzip it:

  ```
  ~> cd python
  ~/python> unzip saskia1807.zip
  ```

- This will generate the same saskia tree as in the case of the mentioned installation under Microsoft Windows. Nevertheless, here it is residing in your home directory.
- Install the required extra packages:

  ```
  ~> pip install cmd2
  ~> pip install pygame
  ~> pip install json_tricks
  ```

- Try to start saskia

  ```
  ~> cd python/saskia
  ~/python/saskia> python saskia.py
  ```

- If this works: Have a look at the **Section Optimization**. If missing packages are reported: Install them. If there is a more severe problem: The **Section Installation Problems** may help.
- For unix-based computers saskia works with many Python versions starting from Python 2.7, e.g. Python 3.6.

3 Optimization

Starting SASkia is simplified, if it can be called from everywhere by typing `saskia` .

> "Everywhere" typically is a directory which contains the data to be evaluated. Moreover, in such an experiment related directory the user can develop and keep local scripts. Local scripts are a proper way to document the specialities[14] of the applied evaluation route.
>
> E.g.: Starting from the program directory saskia enter the subdirectory `examples` and start "`saskia @run_examples`" from there.

[14] Such specialities are: the *s*-range of the valid curve points, the chosen methods of extrapolation and smoothing, the number of extra iterations of spatial frequency filtering ...

The simple start of saskia is achieved by putting a **batch script in the search path**. Respective scripts for Microsoft Windows (saskia.cmd) and Unix (saskia) are provided in the subdirectory `BatchFiles` of `saskia`. There you also find a description concerning the placement and activation of these scripts.

> For Unix-based operating systems place the bash script saskia in the directory ~/bin/ – the user's directory for exexutable files. Do not forget to make the bash script executable.
>
> Concerning Microsoft Windows, I have to confess that I am no expert. I have suggested to put the cmd-script in a directory, which has been put in the search path anyway at the time when Python 3.7 has been installed.

4 From reciprocal to real space: Correlation functions and chord distributions.

Probably the reader has come here, because he was searching for a program that offers to compute chord distributions $g(r)$ according to Méring and Tchoubar or interface distributions like $g_1(r)$ according to Ruland. Then go to the subdirectory[15] `examples` of `saskia`, start saskia and issue

```
saskia> @run_examples
```

This script will read some supplied scattering curves from isotropic data and compute the addressed functions and the respective correlation functions. Each result will be shown in a plot window for inspection. The next curve will be computed after the actual plot window is closed. The provided curves are only pre-evaluated[16]. So other pre-evaluated curves should be processable in a similar manner after minor adaption of some parameters.

A good start to dive into saskia may be to **inspect the scripts** in the directory `examples`. Own curves in text files can easily be read in. Have the script code available, issue command by command and check the result of each step by the "`graphic`" command. Probably arguments must be adapted (limits, estimated background, extrapolation method, scope and step-size of the resulting functions in real space). With noisy data smoothing may help (commands: `median` or `smplain`). If only the unstructured tail of the curve shall be smoothed, then smooth the whole curve, call `graphic` and there li(**n**)k your smoothed tail to the unsmoothed original — but doing so is not suitable with big data.

When a script is working for some frames of an experiment, we want to apply it to all the frames. In a pedestrian way we redirect[17] a file listing to a super-script text file. Thereafter suitable "find and replace" makes every line a call of our tuned script.

[15] In a standard installation on Microsoft-Windows this is `D:\python\saskia\examples` – on Linux this is `~/python/saskia/examples`.

[16] Pre-evaluation: Spike removal, background correction, normalization, conversion to units of s. If your data are in units of q, the units are converted by the supplied script `qts.lrg` (call: saskia> `@qts`)

[17] In Linux: `ls -1 nmbsy03_????.dat > doy03.lrg`. For Windows in a command prompt window: `dir /B nmbsy03_????.dat > doy03.lrg`. (Check content by: `type doy03.lrg`)

When the user gets annoyed by such manual work using a text editor, he may think of writing a Python script which assembles the lines of the "super-script" text file from arguments <head-text>, <data-file-name> and <tail-text>. After this script works nicely, he may find out that this task would make a good saskia command and incorporate it.

5 SASkia, its command language, and cmd2

5.1 SASkia and cmd2

SASkia is controlled by commands. The command language interpreter (CLI) is based on the excellent basic CLI package "**cmd2**". I have chosen not to rely on the newest version of `cmd2`, but to freeze in an older version that still has the feature of command abbreviation. This package is provided as "`savcmd2.py`". Nevertheless, "`cmd2`" should be available as well, because then the background packages **`pyparsing`**, **`pyperclip`** and **`six`** are automatically installed, and these packages are required by `savcmd2.py`. I admit, in doing so I am relying on that the developers of the background packages will not trim their interfaces.

The reason for this decision is not only that I like to have a command interpreter which understands deliberate abbreviations as long as they are unique, but also the fact that the developers of cmd2 have chosen to revoke a central feature on short-notice. Freezing the package prevents SASkia from being destabilized by a package that must be considered unstable.

5.2 Extensions of cmd2

In order to permit script nesting[18], cmd2 has been extended in several respects. The extension which is most important for the user is the way, how commands and their arguments are separated. It has been introduced to facilitate the most simple mapping paradigm of actual[19] arguments to formal[20] arguments, namely by their position on the command line.

Separators. On the user prompt line[21] multiple commands are separated by an ampersand ("&") character. Arguments are either separated by blanks (as is the standard with cmd2) or by commata (my extension). This means that

- arguments are passed to commands and scripts by their position
- arguments (e.g. strings, names, directories) must not contain blanks
- sequences of " `,`" or "`,` " act as repeated separators and should be avoided

[18] Script nesting means, that a script can also contain calls of other, "inner", scripts. When an inner script returns successfully, the outer script is continued. The inner script can be called with arguments, and these arguments may even been assembled in a new way by the outer script.

[19] Actual arguments are the data which the user supplies when he calls the command or the script

[20] Formal arguments are the arguments which the programmer supplies when he writes the command or the script. In scripts the formal arguments are designated by $1, $2, $3, . . . , $8, $9.

[21] Ony on the user prompt `saskia>` ! In LARGO scripts every line must only contain one single command.

- the comma separator permits to skip the explicit specification of arguments. Such skipping tells the command to use the default argument. E.g.: `mul,,2` will multiply the curve in OP1 by s^2, whereas `mul,12.566,,2` will multiply it by $4\pi s^2$.[22]

Passing arguments to LARGO scripts. Inside LARGO scripts arguments are represented by a placeholder sequence "`$?`" in which `?` must be one of the digits "`1`" to "`9`". Upon interpretation of a script line it is scanned for such placeholders. Only if the script has been called with a respective *actual* argument, the supplied argument string[23] replaces the placeholder. If not, the placeholder remains unchanged[24].

6 Installation problems? Information which may help

Preamble. This section is not edited any further. It has been written during the time when I was working on the optimization of Python versions and program code for my program to run at least under unix and MicroSoft Windows. Therefore, there are also aberrations described here. Be warned!

6.1 Installation for Python 2.7

Even in 2018 Python 2.7.12 is still the standard provided by Linux (Ubuntu and Kubuntu), and after own experiences with both 2.7 and 3.6 I agree that this is a good decision. Python 2.7 has everything needed, it is slim and fast.

If a Python 2.7 is provided by the operating system, then only a few packages must be installed in order to run SASkia.

For (K)ubuntu Linux there are only very few Python packages which are not automatically installed by the default distribution. They are installed by:

```
sudo apt-get install python-tk
sudo python -m pip install pygame
```

If some package should be missing in some other environment, then it may help to look below where the compilation of Python 3.6 and the manual installation of all packages is described.

6.2 The SASkia package

The SASkia package requires the files `sa*.py` to be in the program directory (sugggestion: `~/python/saskia`).

User-written scripts can be placed in a subdirectory `scripts`. Following the suggested example this would be `~/python/saskia/scripts`.

[22] For information about the available commands issue the `help` command. For information concerning the multiply command issue `help mul` or `help,mul`.

[23] This means that the command processor handles an argument merely as a sequence of characters. Conversion into numbers or other kind of data types is the task of commands and Python scripts.

[24] In general, not servicing the request for an actual argument will cause an error by the command or the Python script which is finally called.

More flexibility can be gained by adapting the file `saskia.rc` to the special environment of the user. Presently SASkia only reads the variable scriptpath from `saskia.rc`. The path may be specified in an operating-system independent notation syntax using dot-symbols instead of slashes (Linux, Mac) or backslashes (MS-Windows)

SASkia can be made callable from anywhere by providing a corresponding script (bash-script for Linux, cmd-file for Windows) in a directory which is in the search path. For Linux this script is conveniently placed in `~/bin/`

```
#! /bin/bash
# Script: ~/bin/saskia
# Make it executable by: "chmod 755 saskia"
# call SASkia from everywhere
python ~/python/saskia/saskia.py $*
```

NOTICE: FEATURES FROM ONGOING WORK For the evaluation of scattering patterns I have written several extra packages (`sapiawin.py`, `sapatt.py` ...), which are not called by SASkia directly. They are only called by Python scripts which interact with SASkia. Such scripts are called from SASkia following the syntax "@@scriptname".

2D patterns. For example, some of these Python scripts read a complete set of curves and convert it into two-dimensional data sets. Such data sets are "patterns" (`sapatt.py`), which are stored in complex data files. The formats of these data files are based on standard formats. Therefore the corresponding packages must be installed, if such scripts are called. Such packages are:

JSON My standard cross-platform representation of 2D data

json_tricks A more powerful data format, only readable by Python

pickle The standard Python-dinosaur format. Slow.

xdrlib Data format of IDL and PV-WAVE (to read old 2D files – `*.ima`)

By deleting respective parts of `sapatt.py` it is possible to eliminate the need to import some of these packages, if some of the functionality is not needed. JSON is my favorite for permanent storage of complex scattering data, because it is used by many other programs (well introduced) and human-readable (i.e. JSON is no binary file, but a text file).

Patterns: Display and manipulation. Patterns are displayed in **Python-i**ntera**c**tive **win**dows (`sapiawin.py`). If such a window is open, the corresponding data can be explored and manipulated interactively. Such interactive work is initiated by single key strokes followed by using the mouse to input a vector (`savecsel.py`). The vector can be interpreted as a line, a circle, a box or as the spine of a bar which must be “opened” by a second vector input. All these items except for a line define a region, and the region of interest (ROI) may be the inside or the outside of the region. A modifier key (Shift, Ctrl, Alt) defines the different modes. The exploration modes report the maximum intensity or the minimum intensity in the ROI together with position

information. The manipulation modes are used to declare invalid intensity by setting all pixels in the ROI to 0. ROI pixels can also be set to an intensity value of 1. In this way masks can be built. Such masks are useful to declare regions in all scattering patterns of an experiment which contain invalid data (primary beam stop, beam stop holder, shade of vacuum tube, ...).

Final remark. SASkia has been developed using Python 2.7 under Linux (Kubuntu) and Python 3.6 under Linux (CentOS, Redhat). Under Microsoft Windows the various Python implementations cause various problems. As long as no "sapiawin" 2D-patterns are displayed, the coding should be stable even under Windows. If such interactive windows shall be used, most of the Windows versions are buggy, although there are some version combinations of matplotlib and Pythons which have worked. The last severe lapsus was loss of the interactive hook (*http://bugs.python.org/issue31546 PyOS_InputHook is not called*), which has been **corrected** in June 2018 **with Python 3.7.0 for Windows**.

6.3 Installation of Python 3.6

If Python3.6 is not already present, it must be downloaded, compiled, and installed. I give some hints which may then be important.

6.3.1 matplotlib: Graphics backends

All graphics and interaction with graphics of SASkia are realized by the standard package `matplotlib`, which needs at least one "graphics backend". The standard graphics backend is `TkAgg`. **Only if Python must be compiled, the following information is essential**.

In order for the graphics backend to become available, the corresponding system libraries must be supplied. For instance, if the standard `TkAgg` backend must be made available under RedHat or CentOS, then **before compiling Python** as root do:

```
yum install tk
yum install tk-devel # (BOTH must be provided!)
```

or for the Qt4 backend:

```
yum install PyQt4
yum install PyQt4-devel#
```

or for pycairo:

```
yum install pycairo
yum install pycairo-devel
```

For ubuntu:

```
sudo apt-get install tk
sudo apt-get install tk-dev
```

6.3.2 Installing Python from sources

Download a Python source archive, unpack it, enter the generated subdirectory and as normal user do:

```
./configure --enable-optimizations
make
make test # (not necessary)
```

The make command compiles Python **and** all those **matplotlib backend interfaces** which have development headers.

As super user do:

```
make install
```

With the tk-packages pre-installed, we now have _tkinter, the interface for TkAgg.

6.4 Installing all packages

This section is in particular important, if Python is compiled from its sources. All of these packages belong to the standard libraries under (K)ubuntu, but some of them may miss in other Python distributions. All packages must be installed with administrative privilege (under Linux: as “root” or with “sudo”).

It may be necessary to install the package installer "pip" itself. Under Kubuntu (which runs Python 2.7) this is done by the instruction

```
sudo apt install python-pip
```

Only on a sudoer-Linux enter a root command shell:

```
sudo bash
```

On root-Linux systems and on MS-Windows in the Anaconda console, as long as there is only one Python version installed:

```
pip install cmd2
```

This automatically installs pyparsing, pyperclip, six

```
pip install ipython # otherwise SASkia has no "ipy"
pip install numpy # For curves and some numerics
pip install matplotlib # For the graphics to work
pip install pygame # For acoustic signals
pip install LaTeX # For Label typesetting aka LaTeX
pip install Pillow # Just for fun: The new image library.
```

If Python has been compiled by the user on a CentOS system and scipy is not available:

```
yum install lapack # For the Fast Fourier Transform
yum install lapack-devel
```

and for all systems which do not have scipy, but lapack and its development headers (lapack-devel) are already available

```
pip install scipy # There you go
```

If both Python 2.7 and Python 3 are installed on the same computer, all the pip commands must be prepended by the clause `python2 -m` (if preparing to use Python 2.7) or `python3 -m` (if preparing to use Python 3). If the packages are installed on a sudoer system by individual sudo commands, the sudo command should be called like `sudo -H` to specify that the packages are installed in the systemwide home directory to be accessible for all users of the computer.