Ruby-Helix

Masahide Kikkawa

June 18, 2018
4 Script References.
4.1 3D-fourier-synthesis.rb ............................................. 51
4.2 auto_unbent ........................................................... 51
4.3 bG-lG-3D-renumber-rot-lg.rb ................................. 51
   4.3.1 Read a big G file ......................................... 52
4.4 Apply Fourier-Bessel transform to the big G and obtain a small g. .... 52
4.5 correlation .......................................................... 52
4.6 correlationTube ...................................................... 52
4.7 check_installation .................................................. 52
4.8 correlation_matrix .................................................. 53
4.9 ccd2ctf .............................................................. 53
4.10 cut_out .............................................................. 53
4.11 fft ................................................................. 54
4.12 filter_radius ....................................................... 54
   4.12.1 What it is? .................................................. 54
   4.12.2 How it works? .............................................. 54
   4.12.3 How to use? ................................................ 54
4.13 header .............................................................. 55
4.14 helix_boxer ........................................................ 55
4.15 hlxfit5 and hlxfit6 ................................................ 55

3.7 Unbending (or straightening) ....................................... 26
3.7.1 "unbent" on Linux ............................................... 26
3.7.2 behind the scene: auto_unbent ................................ 29
3.8 ruby-helix-gui ...................................................... 32
3.9 Bootstrapping with ruby scripts .................................. 33
3.10 CUI analysis – scenario 3 – semi-automated analysis .......... 34
   3.10.1 Parameter file 1.a0000total.rb ............................ 34
   3.10.2 Parameter file 2: a0000.rb ................................. 36
   3.10.3 Refinement strategy ......................................... 37
3.11 Averaging ............................................................ 38
   3.11.1 What should be done? ..................................... 38
   3.11.2 How? .......................................................... 38
   3.11.3 hlxavgctf ...................................................... 38
   3.11.4 Select good dataset ....................................... 39
   3.11.5 CTF correction scheme .................................... 39
3.12 Asymmetric Helical Reconstruction. ............................. 41
   3.12.1 Judging the seam side and the seam-free side ........... 42
3.13 Determining effective resolution using Fourier Shell Correlation ... 42
3.14 Visualizing 3D maps ................................................. 43
   3.14.1 Generate MRC and XPLOR-format density maps ......... 43
   3.14.2 UCSF Chimera ............................................... 43
   3.14.3 O, PyMol, and VMD ......................................... 43
   3.14.4 Convert the MRC map for O .............................. 44
   3.14.5 Convert the MRC map for PyMol ........................ 44
   3.14.6 Reading an XPLOR format file and visualize isomesh in PyMol .. 45
   3.14.7 Convert the MRC map for VMD .......................... 46
   3.14.8 With AVS ...................................................... 46
3.15 Docking atomic models using colores ........................... 46
3.16 Obsolete procedures ............................................... 47
   3.16.1 Unbend on Mac OS X ...................................... 47
   3.16.2 Unbending with bshow ..................................... 48

4 Script References. 51
   4.1 3D-fourier-synthesis.rb ............................................. 51
   4.2 auto_unbent ........................................................... 51
   4.3 bG-lG-3D-renumber-rot-lg.rb ................................. 51
      4.3.1 Read a big G file ......................................... 52
   4.4 Apply Fourier-Bessel transform to the big G and obtain a small g. .... 52
   4.5 correlation .......................................................... 52
   4.6 correlationTube ...................................................... 52
   4.7 check_installation .................................................. 52
   4.8 correlation_matrix .................................................. 53
   4.9 ccd2ctf .............................................................. 53
   4.10 cut_out .............................................................. 53
   4.11 fft ................................................................. 54
   4.12 filter_radius ....................................................... 54
      4.12.1 What it is? .................................................. 54
      4.12.2 How it works? .............................................. 54
      4.12.3 How to use? ................................................ 54
   4.13 header .............................................................. 55
   4.14 helix_boxer ........................................................ 55
   4.15 hlxfit5 and hlxfit6 ................................................ 55

4
5 Method References.

5.1 NArray class methods. .................................................. 67
  5.1.1 NArray initialization methods. ................................... 67
  5.1.2 Conversion methods. ................................................. 67
  5.1.3 NArray information .................................................. 68
  5.1.4 Slicing the array .................................................... 68
  5.1.5 Replacing Elements ................................................. 69
  5.1.6 Filling values ....................................................... 69
  5.1.7 Arithmetic operators .............................................. 69
  5.1.8 Bitwise operators ................................................... 69
  5.1.9 Comparisons ........................................................ 69
  5.1.10 Statistics .......................................................... 70
  5.1.11 Manipulating Array ............................................... 70
  5.1.12 Type conversion ................................................... 70
  5.1.13 Iteration .......................................................... 70
  5.1.14 Complex compound number ...................................... 70
  5.1.15 Arithmetic functions as NMath module ............................ 73

5.2 NMatrix ................................................................. 73
  5.2.1 NMatrix, standard matrix generators .............................. 74

5.3 NVector ................................................................. 74

5.4 NMatrixLU ............................................................... 74

5.5 FFTW3 module ............................................................ 74

5.6 CUFFT module ............................................................ 75

5.7 EM related NArray methods. .......................................... 76
  5.7.1 Input and output .................................................... 76
  5.7.2 Image manipulations. ............................................... 77
  5.7.3 FFT-image related. .................................................. 79
  5.7.4 Header related ....................................................... 80

5.8 LayerLine ............................................................... 80
  5.8.1 Accessor methods ................................................... 81
  5.8.2 Manipulation ........................................................ 81

5.9 BigG ..................................................................... 81
5.9.1 Input and output ............................................................... 81
5.9.2 Manipulating big G files .................................................. 82
5.9.3 Fourier-Bessel conversions. ............................................. 82
5.9.4 Accessor methods .......................................................... 82
5.9.5 Basic Helices. .............................................................. 83
5.9.6 Methods for analyzing the data ....................................... 85
5.10 SplineArray ................................................................. 85
5.11 SerialG ..................................................................... 87
  5.11.1 Accessor methods ..................................................... 88
  5.11.2 Operators .............................................................. 88
5.12 Frealign .................................................................... 88
5.13 PDB (protein data bank) .................................................. 88
5.14 Minimizer2D, a two-dimensional minimizer ......................... 89
5.15 Minimizer3D, a three-dimensional minimizer ....................... 89
5.16 Histogram ................................................................. 90
Chapter 1

Overview.

Ruby-Helix is an integrated system for helical reconstruction. The programs used in this system are mostly script written in Ruby, an object oriented scripting language, like Python. There are several reasons why I chose this language for integrating helical image analysis programs.

1. Very powerful numerical N-dimensional array library, NArray, is available. Based on this library, most of the script is purely written in Ruby without using C, without loosing the computational power.

2. Easy to migrate from PERL. As the name suggest, Ruby is somewhat similar to PERL or even C. So it is easy to migrate from PERL or C.

1.1 Biological application of Ruby-Helix

Ruby-Helix has been used mainly for high resolution studies of kinesin-microtubule complexes. KIF1A-microtubule complex was solved at 10Å resolution [1] and mitotic kinesin-microtubule complex was solved at 9Å resolution [2].

However its application is not limitted to kinesins. Dynein-microtubule complexes were solved at 15Å resolution [3, 4]. Since writing scripts is easy using Ruby-Helix, a graduate student (Toshiyuki Oda) also use it to combine helical and single particle analysis to study outer dynein arm-microtubule complex [5].

1.2 About Ruby Programming Language

Here is a quote from ruby-lang1.

Ruby is the interpreted scripting language for quick and easy object-oriented programming. It has many features to process text files and to do system management tasks (as in Perl). It is simple, straight-forward, extensible, and portable.

If you are familiar with Perl or C, after reading “the book” Programming Ruby2 for a few hours, you will feel comfortable with the language. Need a reference? Please buy “The Ruby Programming Language” written by David Flanagan [6]. This is better than “The Programming Ruby: The Pragmatic Programmers’ Guide [7]”, which is also an excellent book.

---

1 http://www.ruby-lang.org
2 http://www.rubycentral.org/book/
1.3. NARRAY

NArray is a ruby array library for fast and easy calculation. It has following properties.

- Fast and easy calculation for large numerical array.
- Accepting Elements: 8,16,32 bit integer, single/double float/complex, Ruby Object.
- Easy extraction/substitution of array subset, using assignment with number, range, array index.
- Operators: +, -, *, /, %, **, etc.
- FFTW interface.
- NImage: Image viewer class.
- Ruby/PGPLOT: Graphics library interface (separately distributed) X-Y Graph, Histogram, Contour map, Image map, etc.

In this sense, NArray is similar to Python/NumPy\(^3\), Perl/PDL\(^4\), Yorick\(^5\), or IDL.

Though most of the functions are usable, NArray is far from complete. It is still in experimental stage. In the future, specification may be changed. If you find any bug, please report it to the author.

1.4 Ruby-Helix: tested Platform

NArray was tested in the following platforms.

- ruby 1.8.6 and 1.8.7
- gcc version 4.0.1 or later.
- Mac OS X Leopard 10.5.3, Snow Leopard (10.6), and Lion (10.7)
- Fedora Core from 8 to 16

1.5 License

This program is free software. You can distribute/modify this program under the same terms as Ruby itself. NO WARRANTY.

1.5.1 Cephes Mathematical Libraries

Ruby-helix also uses Cephes Mathematical Libraries for non-integer order Bessel functions, and include the source code. Here is the readme from Cephes Mathematical Libraries.

Some software in this archive may be from the book "Methods and Programs for Mathematical Functions" (Prentice-Hall or Simon & Schuster International, 1989) or from the Cephes Mathematical Library, a commercial product. In either event, it is copyrighted by the author. What you see here may be used freely but it comes with no support or guarantee.

The two known misprints in the book are repaired here in the source listings for the gamma function and the incomplete beta integral.

Stephen L. Moshier moshier@na-net.orl.gov

\(^3\)http://www.scipy.org
\(^4\)http://pdl.perl.org/
\(^5\)http://www.maumae.net/yorick/doc/index.php
1.5.2 Ruby/ProgressBar

Ruby-Helix uses Ruby/ProgressBar to estimate remaining time. Ruby/ProgressBar is a free software written by Satoru Takabayashi http://raa.ruby-lang.org/project/progressbar/.

1.6 Authors

- Masahiro TANAKA
- Masahide Kikkawa

If you have any question about ruby-helix, please send e-mails to Masa Kikkawa.

1.7 Reference

If you publish papers using Ruby-Helix, please refer our paper:
Metlagel, Z., Y. S. Kikkawa, M. Kikkawa
“Ruby-Helix : an implementation of helical image processing based on object-oriented scripting language”
Chapter 2

Installation.

2.1 Installing required libraries

Ruby-Helix depends on several other libraries. Therefore installation will be a little bit cumbersome process. The library dependencies are summarized in Fig. 2.1.

For each OS, we describe our recommended way to install required libraries. After finishing the following sub section, please go to next section 2.2.

Note about 64bit environment You can compile Ruby-Helix in 64 bit OS, such as Linux x86_64 environment or Mac OS X 10.6 Snow Leopard. However, you may not be able to use very large array that has more than $2^{32}$ elements.

2.1.1 Fedora 10 and higher

Fedora 10 and higher comes with pre-compiled libraries. By using `yum`, you can install required libraries.

```
# yum install ruby-devel libtiff-devel fftw-devel gsl-devel
```

If you want to use GUI unbent, you also need to install following libraries

```
# yum install ruby-libglade2 glade2
```

2.1.2 Mac OS X 10.5, 10.6, 10.7, and 10.8

We recommend using `macports`\textsuperscript{1} to install libraries.

After installing macport, do the following:

```
sudo port install gsl tiff fftw-3 fftw-3-single ruby rb-libglade2
```

macports installs ruby 1.8 as /opt/local/bin/ruby1.8, which is recommended for ruby-helix. So, in the following, please use ruby1.8 instead of system's /usr/bin/ruby.

2.1.3 Manual installation

If you like a hard way, please go to the last section of this chapter.

\textsuperscript{1}http://www.macports.org
Figure 2.1: Library dependencies
2.2 Installing ruby-helix

Compilation and installation of ruby-helix follows the standard ruby package procedure:

```sh
$ tar zxvf ruby-helix.tar.gz
$ cd ruby-helix
$ ruby extconf.rb
```

Please take a look at the log. Depending on the library installed on your computer, you will see a log like the following:

```
checking for u_int8_t... yes
checking for int16_t... yes
checking for int32_t... yes
checking for u_int32_t... yes
checking for main() in -lm... yes
checking for asinh() in math.h... yes
checking for jn() in math.h... yes
checking for jnf() in math.h... yes
checking for main() in -lsicblas... yes
checking for main() in -lgsl... yes
checking for main() in -ltiff... yes
checking for fftw3.h... yes
checking for main() in -lfftw3... yes
checking for main() in -lfftw3f... yes
Have single float FFTW3 library
checking for #include <cuda.h>
... yes
checking for main() in -lcufft... yes
checking for gsl/gsl_specfunc.h... yes
checking for main() in -lgsl... yes
checking for main() in -lgomp... yes
checking for omp.h... yes
creating numpy_config.h
creating Makefile
```

You may see that some of the libraries are missing. If any libraries are missing, go back to the previous section.

If it is OK, go ahead makeing and installing the libraries.

```sh
$ make
$ su
# make site-install (or, make install)
```

On Mac OS X, you may use sudo as follows:

```sh
$ make
$ sudo make site-install
```

2.2.1 Ruby/GSL

Ruby/GSL\(^2\), GSL library implementation by Yoshiki Tsunesada. Version 1.7 does not work. Ruby-Helix is currently tested with rb-gsl 1.14.5. Anyway, this is usually trickiest part of the installation. If you have any problem, please send e-mail to me.

\(^2\)http://rb-gsl.rubyforge.org/
2.3 Other software required for helical image analysis.

For helical image analysis, several programs are not yet ported to Ruby-Helix. Nigel Unwin (MRC, Cambridge) kindly allowed us to distribute the programs required for Ruby-Helix.

- Nigel's MRC helical package:
2.4 CHECK INSTALLATION

- hftout
- hlx
- srch (Please rename it to srch0 or make a symbolic link)

- xplot\(^3\): The program is used to plot data during helical image analysis.
- fftrans (This is included in the MRC image2000 library)
- ctffind\(^4\)

When you build above commands, the commands may have “.exe” extension. If it is the case, please make a copy or symbolic link that does not have “.exe” extension.

In addition, please make a symbolic link from srch0 to srch. This is just because historical reason and we will fix it in the future.

2.3.1 Compiling MRC programs

Compilation of MRC programs requires a Fortran compiler. Free fortrans such as f77, g77, or gfortran are sometimes incompatible with MRC programs. We recommend using PGI Fortran\(^5\) or Intel Fortran[http://software.intel.com/en-us/intel-compilers/](http://software.intel.com/en-us/intel-compilers/).

2.4 Check installation

To identify lack of libraries you need, you can use a script check_installation in ruby-helix/app directory. If your installation is complete, you should see like this

```
$ ruby check_installation
Start checking installation
Checking Ruby/GSL installation to see whether compiled with NArray
Ruby/GSL is OK
Finished with no problems!
```

2.5 Set your PATH (you are almost done)

Please add ruby-helix/app directory to your path.

If you are done up to here, you can open your image file (MRC format) using unbent and start unbending. See the following chapter.

2.6 Setting up irb (optional)

irb is a Ruby module that let you enter Ruby programs interactively and see the result immediately. When used with Ruby-Helix, it is quite useful, because you can modify the image interactively. To facilitate this interactive shell, please prepare ~/.irbrc like follows.

```
require 'irb/completion'
require 'mrc.rb'
require 'LayerLine.rb'
```

\(^3\)[http://www.xplot.org/]
\(^4\)[http://emlab.rose2.brandeis.edu/software]
\(^5\)[http://www.pgroup.com/]
\(^6\).
2.7 Other optional software for helical image analysis.

Ruby-Helix mainly provides most of the CUI based analysis, however, several other software is useful for entire analysis. Here is the list of software you may find useful.

- **AVS5**: Optional, but it is quite useful for three-dimensional visualization. We made a “big G to 3D” module, which can generate a three-dimensional map directly from a big G, without generating a little g file.
- **Bsoft**:

2.8 Document compilation

Following is only for Ruby-Helix developers. Users, please ignore this.

The document of Ruby-Helix is written in **LaTeX**.

There is a script to make ruby-helix documentations. In our environment, we use “.make” command. To make the script work, you need following software.

- **BioRuby**.
- **latex2html**

if you encounter an error similar to shown below:

```
Unable to read an entire line —— bufsiz...e =200000.
Please increase bufsiz...e in texmf.cnf.
```

In this case, please edit `texmf.cnf` and increase `buf_size` to 200000 (or else).

2.9 Installing Bsoft on Mac OS X

Please take a look at Bsoft website[^10]. Especially `bshow` is useful.

```
    cd /usr/local/
    tar xvf bsoft.tar
    cd bsoft
    ./bmake
```

Set the library path etc. Here are some environmental variable for `bsoft`.

```
declare -x BPARAM="/usr/local/bsoft/parameters/"
declare -x BSOFT="/usr/local/bsoft"
declare -x LD_LIBRARY_PATH="/home/app/EMAN/lib:/usr/local/bsoft/lib"
```

[^7]: [www.avs.com](http://www.avs.com)
2.10 Manual installation

**LibTiff** Most Unix-flavored systems, such as RedHat and Fedora, include libtiff library. Then you do not need to install extra libraries.

If you use Mac OS X (10.4, 10.5, and 10.6), please download recent version of LibTIFF\(^{11}\) library, compile and install it.

As of 2009, we switched from Fink to In this case, we install following libraries:

- gsl-devel
- libtiff-devel
- fftw-3
- fftw-3-single

However, we don’t use rb-gsl.

Until 2008, we have been installing libtiff through Fink\(^{12}\). Don’t forget to install developer include files for libtiff. However, it seems macports are more stable than Fink and we switched to macports.

**FFTW3** Ruby-Helix uses FFTW3\(^{13}\). Although many Linux distribution may contain FFTW3, installation from the source code is recommended, because having both single and double float FFTW3 saves the computation time and memory. See instruction for installation below.

- Ruby-Helix is tested with fftw-3.1.X and fftw-3.2.X

**GSL** GNU scientific library (GSL)\(^{14}\). Tested with gsl-1.11, 1.12, and 1.13.

**Ruby** Ruby-Helix requires version 1.8 of Ruby\(^{15}\). Ruby-Helix is not compatible with 1.9. For Linux, most of the distribution has a package for Ruby. Mac OS X 10.5 (Leopard) and 10.6 (Snow Leopard) come with fully functional Ruby, which is compatible with Ruby-Helix.

- Note that Mac OS X 10.4 (Tiger) also comes with Ruby, but it does not contain include files required for compiling ruby-helix.

2.11 Optional libraries

Most of ruby-helix scripts are non-GUI-based programs that do not require graphics libraries. Still, some scripts uses graphics libraries.

**RMagick** Some methods use RMagick. First install ImageMagick\(^{16}\). Most Linux distributions include RPM files. Then, install RMagick\(^{17}\).

2.12 Ruby

Ruby is installed in many operating systems, including RedHat Linux, Fedora, and Mac OS X.

If Ruby is not installed on your system, you will be able to find packages of Ruby. Please use ruby version 1.8. Ruby-Helix is not compatible with ruby-1.6.

\(^{11}\)http://www.libtiff.org/
\(^{12}\)http://fink.sourceforge.net/
\(^{13}\)http://www.fftw.org
\(^{14}\)http://www.gnu.org/software/gsl/
\(^{15}\)http://www.ruby-lang.org/
\(^{16}\)http://www.imagemagick.org/
\(^{17}\)http://rmagick.rubyforge.org/
2.12. RUBY

2.12.1 Ruby for Mac OS X 10.4 (Tiger)

Note about Mac OS X 10.4 (Tiger). If you are using Mac OS X 10.5 or higher, you can ignore the following section.

Ruby on Mac OS X (Tiger) comes with operating system can not compile Ruby/GSL. Please compile ruby by yourself or Ruby one-click installer\(^\text{18}\) (not well tested in our hand).

Here is the outline.

First install GNU readline library. This is to enable \texttt{irb} (interactive ruby). Please download readline-5.2 from the following link:

\texttt{ftp://ftp.cwru.edu/pub/bash/readline-5.2.tar.gz}

Then make and install the library.

\begin{verbatim}
$ ./configure --prefix=/usr/local --enable-shared
$ make
$ sudo make install
\end{verbatim}

Next, please download ruby-1.8.6 from the following link:

\texttt{http://www.ruby-lang.org}

My recommendation for the configuration option is:

\begin{verbatim}
$ ./configure --prefix=/usr --enable-thread --enable-shared \ 
   --with-readline-dir=/usr/local
$ make
$ sudo make install
\end{verbatim}

Sometimes, the compilation fails. In that case, please try to remove the --enable-thread option.

2.12.2 GSL

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License.

The library is required for non-integer order Bessel functions in Ruby-Helix. Please download the most recent version from your closest GNU mirror\(^\text{19}\) and install it as follows:

\begin{verbatim}
$ ./configure --prefix=/usr --enable-shared
$ make
# make install
\end{verbatim}

We tested Ruby-Helix with some packages such as \texttt{gsl-1.11-4.fc10.x86\_64} or \texttt{gsl-1.12-3.fc11.x86\_64}.

2.12.3 FFTW3

To activate the single-float transform, you have to install FFTW3 with the single-float compilation, in addition to the default double-float version. This can be done by configuring FFTW3 with the --enable-float option as follows: (in this case, optimized for Intel CPU(s) by adding --enable-fma option)

\begin{verbatim}
$ ./configure --prefix=/usr --enable-shared --enable-fma --enable-sse2
$ make
# make install
$ ./configure --prefix=/usr --enable-shared --enable-fma --enable-sse --enable-float
$ make
# make install
\end{verbatim}

\(^{18}\)\texttt{http://rubyosx.rubyforge.org/}
\(^{19}\)\texttt{http://www.gnu.org/prep/ftp.html}
CHAPTER 2. INSTALLATION

For Mac OS X (PowerPC), I would compile FFTW3 like:

```
$ ./configure --prefix=/usr --enable-altivec --enable-float --enable-fma --enable-shared
$ make
$ sudo make install
$ ./configure --prefix=/usr --enable-fma --enable-shared
$ make
$ sudo make install
```

For Mac OS X (Intel), I would compile FFTW3 like:

```
$ ./configure --prefix=/usr --enable-float --enable-fma --enable-shared --enable-sse
$ make -j3
$ sudo make install
$ ./configure --prefix=/usr --enable-fma --enable-shared --enable-sse2
$ make -j3
$ sudo make install
```

The single-float version will coexist with the double-float version. If you do not install the single-float version, FFT is always done with double precision, which should still work and is not bad idea if your computers have a lot of memory and CPU powers.
Chapter 3

Helical image analysis.

In this chapter, overall procedure for helical analysis is described using ruby-helix.

3.1 Brief history of helical reconstruction theory

The theory of helical image analysis was first described by A. Klug, F. H. C. Crick, and H. W. Wyckoff [9], which is also known as “KCW paper”. The helical reconstruction method was later applied to EM images to produce 3D structure of a tail of bacteriophage T4 [10]. As an extension of original helical reconstruction to helices with a seam, such as microtubule, Kikkawa developed a method called “asymmetric helical reconstruction” [11].

3.2 Scan EM films or transfer CCD images.

Scan films using Leafscan 45 (in Kikkawa’s lab) or your favorite scanner. Save the image as 16 bit gray scale tiff image format. Either Mac or PC format is fine. You can also save the image with compress option to cut the file size.

Tubes/filaments should be scanned so that their axis is parallel to the $y$-axis. Otherwise, please rotate the image using Photoshop or other image processing software.

3.3 Directory organization

We recommend to make a new directory for one project. In the directory you have following sub-directories.

2. Directories for image.
3. Directories for big G files.

The database file is a SQLite3 database file, which you can generate database (EMdatabase.sqlite) from template SQL by the following command:

```
$ cat /somewhere/ruby-helix/control_files/EMdatabase.sql | sqlite3 EMdatabase.sqlite
```

In the following procedure, we use this “EMdatabase.sqlite” to store some of the information.
3.3. DIRECTORY ORGANIZATION

CHAPTER 3. HELICAL IMAGE ANALYSIS.

Image Acquisition (cryo-EM or negative staining)
Digitization
Unbending
determine true pitch
cut out the image / FFT
indexing
layer line extraction
averaging
Fourier-Bessel conversion to little g
3D visualization

CTF determination

reference

determine true pitch

Figure 3.1: Helical Analysis Overview
CHAPTER 3. HELICAL IMAGE ANALYSIS

3.4 From the scanned tiff image

If you have a database file, you can use following command to set up image analysis directory.

```
setup_from_tiff *.tiff
```

This command does:
1. make a directory for each tiff file
2. convert the tiff file to mrc using tif2mrc command.
3. Use specified CTFFIND3 script
4. Read the ctffind3.log file and update sqlite database file

If you do this, skip to 3.7 Unbending.

3.5 Convert the scanned image into the MRC format.

As a first step for the image analysis, please convert the original tiff file to MRC format and print out a preview image.

1. convert the 16bit tiff image to MRC format using tif2mrc. This program convert the density according to the following equation.

\[
\rho_{\text{new}} = \log_{10}(\frac{\rho_{\text{scan}} - \min + 1}{\max - \min}) \cdot \frac{32768}{\text{Max OD}}
\]

(3.1)

Here, we use \(\min = 0.0\), \(\max = 65536.0\), \(\text{Max OD} = 4\). Depending on your scanner property, you can change this value by modifying your tif2mrc script.

2. (optional) make preview image using Photoshop.

3. (optional, but recommended) print out image compressed along the axis of the tube using tif2ps.

In the following helical image analysis, the axis of the tubes/filaments is assumed to be parallel to the y-axis. If your tubes/filaments are in horizontal, use the transpose script. This script does not use any interpolation.

3.6 Determine the defocus level using ctffind3.

First convert the image to MRC format. If the image has black or white region, it is better to crop only the region include the “image” using Photoshop. (Therefore, you need to work with tiff and then convert the cropped image to MRC format.) Carbon region is fine, and sometimes helps to determine the defocus level.

Then execute ctffind3\(^1\) [12].

```
> ctffind3

CTF DETERMINATION, V3.0 (21–Sep–2002)
```

First you will be asked the file name.

```
Input image file name
b1012.mrc
b1012.mrc
```

\(^1\)http://emlab.rose2.brandeis.edu/
3.6. DETERMINE THE DEFOCUS LEVEL USING CTFFIND3

CHAPTER 3. HELICAL IMAGE ANALYSIS.

<table>
<thead>
<tr>
<th>Electron Microscope</th>
<th>CS = spherical aberration</th>
<th>Acc Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>JEM-2010F, H pole piece</td>
<td>3.3 mm</td>
<td>200KV</td>
</tr>
<tr>
<td>JEM-2100F, high contrast</td>
<td>2.0 mm</td>
<td>200KV</td>
</tr>
<tr>
<td>JEM-2200SE, high contrast</td>
<td>2.2 mm</td>
<td>200KV</td>
</tr>
<tr>
<td>JEM-Z2100F</td>
<td>1.6 mm</td>
<td>200KV</td>
</tr>
<tr>
<td>JEM-3000SFF</td>
<td>1.5 mm</td>
<td>300KV</td>
</tr>
</tbody>
</table>

Table 3.1: CS values for JEOL microscopes.

<table>
<thead>
<tr>
<th>Brand</th>
<th>Type</th>
<th>Pixel Size</th>
<th>Post Column mag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tietz</td>
<td>F224HD</td>
<td>24 μm</td>
<td>1.63 (Kyoto)</td>
</tr>
</tbody>
</table>

Table 3.2: Size of CCD pixels.

Second, you will be asked about the diagnostic file name. This will be generated by ctffind3. You can view the image using bshow or other MRC readable programs. Since this program can not overwrite existing file, This output diagnostic file should not exist.

Output diagnostic file name
b1012.pow
b1012.pow

Third question is a little bit complicated. CS = spherical aberration. For specific values, see Table 3.1. HT is the acceleration voltage of your microscope. AmpCnst is the amplitude constant[13]. Usually this value range from 0.04-0.07. This value also depends on the microscope. XMAG is the (nominal) magnification of the image and DStep is the pixel size of the image when you scan the image. If you use our CCD (Tietz F224HD), the post column magnification is 1.63 and the pixel size is 24 micron, see Table 3.2.

<table>
<thead>
<tr>
<th>CS[mm], HT[kV], AmpCnst, XMAG, DStep[um]</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3, 200.0, 0.07, 40000.0, 10.0</td>
</tr>
</tbody>
</table>

Positive defocus values for underfocus

The final values are shown below. Usually keep these values. If the defocus maximum is larger than this value, it should be changed.

<table>
<thead>
<tr>
<th>Box, ResMin[A], ResMax[A], dFMin[A], dFMax[A], FStep</th>
</tr>
</thead>
<tbody>
<tr>
<td>128, 200.0, 8.0, 10000.0, 30000.0, 500.0</td>
</tr>
</tbody>
</table>

Then, the program start fitting the CTF to the amplitude.
CHAPTER 3. HELICAL IMAGE ANALYSIS

3.6. DETERMINE THE DEFOCUS LEVEL USING CTFFIND3.

<table>
<thead>
<tr>
<th>Number of columns, rows, sections</th>
<th>843 5981 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map mode</td>
<td>2</td>
</tr>
<tr>
<td>Start points on columns, rows, sections</td>
<td>0 0 0</td>
</tr>
<tr>
<td>Grid sampling on x, y, z</td>
<td>843 5981 1</td>
</tr>
<tr>
<td>Cell axes</td>
<td>0.0000E+00 0.0000E+00 0.0000E+00</td>
</tr>
<tr>
<td>Cell angles</td>
<td>0.000 0.000 0.000</td>
</tr>
<tr>
<td>Fast, medium, slow axes</td>
<td>X Y Z</td>
</tr>
<tr>
<td>Origin on x, y</td>
<td>0.00000E+00 0.36714E+42</td>
</tr>
<tr>
<td>Minimum density</td>
<td>0.11432</td>
</tr>
<tr>
<td>Maximum density</td>
<td>2.5493</td>
</tr>
<tr>
<td>Mean density</td>
<td>0.62362</td>
</tr>
<tr>
<td>Space group, # bytes symmetry</td>
<td>0 0</td>
</tr>
<tr>
<td>Number of titles</td>
<td>0</td>
</tr>
</tbody>
</table>

Titles:

READING IMAGE...
NX, NY= 843 5981

TILING IMAGE...
Total tiles and number used 276 272

FILTERING POWER SPECTRUM...

SEARCHING CTF PARAMETERS...

<table>
<thead>
<tr>
<th>DFMID1</th>
<th>DFMID2</th>
<th>ANGAST</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000.00</td>
<td>10000.00</td>
<td>0.00</td>
<td>-0.02572</td>
</tr>
<tr>
<td>10000.00</td>
<td>22500.00</td>
<td>0.00</td>
<td>-0.02570</td>
</tr>
<tr>
<td>10000.00</td>
<td>23000.00</td>
<td>0.00</td>
<td>-0.01771</td>
</tr>
<tr>
<td>10000.00</td>
<td>23500.00</td>
<td>0.00</td>
<td>-0.01101</td>
</tr>
<tr>
<td>10000.00</td>
<td>24000.00</td>
<td>0.00</td>
<td>-0.00644</td>
</tr>
</tbody>
</table>

....

| 25000.00 | 26000.00 | 0.00  | 0.13925 |
| 24500.00 | 26000.00 | 22.50 | 0.14243 |
| 24500.00 | 26500.00 | 22.50 | 0.14299 |
| 25000.00 | 26000.00 | 22.50 | 0.14349 |

REFINING CTF PARAMETERS...

<table>
<thead>
<tr>
<th>DFMID1</th>
<th>DFMID2</th>
<th>ANGAST</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>24778.33</td>
<td>26228.72</td>
<td>21.96</td>
<td>0.14424</td>
</tr>
</tbody>
</table>

Final Values

WRITING IMAGE...
NX, NY= 128 128
(Q)OPEN allocated # 2
User: mkikkawa Logical Name: b1012.pow
Status: NEW Filename: b1012.pow

Filename for NEW image file on unit 2: b1012.pow
NEW VALUES:
DMIN, DMAX, DMEAN
1.000 1.000 0.090

These values (DFMID1,DFMID2,ANGAST) are used in later step.

The Thone ring pattern can be examined by using a program bshow. Since bshow recognizes the file format based on the extension, you need to make a symbolic link from hoge.pow file to hoge.pow.mrc by the following command.

```
ln -s hoge.pow hoge.pow.mrc
```

3.7 Unbending (or straightening)

For unbending, you need a GUI interface. We mainly use “unbent” on Linux. Although we have been using Mac OS X version of “Unbend”, this “Unbend” program becomes obsolete.

3.7.1 "unbent" on Linux

Unbent

In Kikkawa lab, Linux version of “unbent” is mainly used for straightening tubes and filaments.

Please start by

```
unbent <mrc filename>
```

or if you have unbent the image before,

```
unbent <ub filename>
```

Figure 3.2: "unbent" program running on Fedora Linux
CHAPTER 3. HELICAL IMAGE ANALYSIS. 3.7. UNBENDING (OR STRAIGHTENING)

Typical work flow:

1. Adjust contrast (Contrast setting is stored in .ub file).
2. Place control points for unbent.
3. Click “auto center” button to see the unbent image as well as adjust the control points.
   Keyboard short cut: Control-C can be also used for the “auto center”.
4. Click auto center several times until no further improvement is made.
5. Check the diffraction pattern by using “Analysis” → “FFT”.
   You will see a window like shown in Figure 3.2.

Options

When starting unbent, you can specify parameters used in the unbending. These parameters can also be changed in the GUI.

- \( -w, \, --width \, VAL \) cut out width in pixel.
- \( -p, \, --pixel \, VAL \) pixel size in \( \text{Å} \).

Menu Functions

- File
  - New
  - Open
  - Read unbent.final
  - Refresh
  - save unbent
  - save UB

- Edit
  - Cut
  - Copy
  - Paste
  - Delete

- Image
- Scale
- Analysis
- Help
3.7. UNBENDING (OR STRAIGHTENING)  

CHAPTER 3. HELICAL IMAGE ANALYSIS.

Auto Unbent

Within this window, a mouse click functions as follows:

- Left click: add or delete a new control points.
- Shift - left click:
  - If upper half of the image is clicked, set start point of spline.
  - If lower half of the image is clicked, set end point of spline.
- Right click: show the coordinate of clicked point in the terminal.

Next, push “auto unbent” button to refine the positions of control points. This function is equivalent to auto_unbent.

If the delta is small enough (for example, less than 0.1 pixels), save the control points, usually named as “unbent.final”.

Adjusting control points

To straigten the filament, you can use “left click” on the “Auto Center” window.

If you find a region where the filament is not well straightened, left click the region (Figure 3.3). Then, in the main window, corresponding region is shown using a line, and its coordinate is also shown in the status line.

Using this like as a guide, replace or add control points.

Use xdisph to display the diffraction pattern

If xdisph is installed, you can also use it to display diffraction pattern of the unbent image.
3.7. unbending (or straightening)

3.7.2 behind the scene: auto_unbent

Unbent program has built-in “auto_unbent” function. So, if you go through the previous sub section, you don’t need to follow the procedure written here. However, if you want to know how the program works or need to shoot troubles, following subsection may help.

To refine the manually placed control points. Please copy auto_unbent script from ruby-helix/app directory to your local directory. You may need to edit the header as follows.

For a segment of length \( l \), usually on the order of the repeat length, and centered around each control point, auto_unbent calculates the 1D projection along the spline curve. The program then performs a cross-correlation \( CC_n(\delta x) \) of this 1D projection with its mirror image, and adjusts the spline curve towards the cross-correlation peak defined by:

\[
CC_n(\delta x) = \int \left( \int_{y_n-l/2}^{y_n+l/2} \rho(x+\delta x, y)dy \right) \left( \int_{y_n-l/2}^{y_n+l/2} \rho(-x-\delta x, y)dy \right) dx;
\]  

where \( \rho \) is the image of the helical polymer, whose axis is parallel to the \( y \)-axis and \((x_n, y_n)\) is the coordinate of the \( n \)-th control point.

The box size surrounding the control point is specified by two parameters:

```plaintext
# Specify the box size to find the center of the tube
X_width = 80  # half width in X direction
Y_length = 200  # half length along tube axis
MaxDelta = 10  # pixels, must be an integer

Usually X_width is 20-30% larger than the radius of the tube. Y_length depends on the radius of the tube and the contrast of the image. If the radius is large or the contrast of the image are large, you can make Y_length small.

Usually control points are placed within 10 pixels of the actual center, so specify that MaxDelta is 10.

# Number of control points that do not need refinement.
Omit_start = 1
Omit_end = 1

These two parameters specify the number of control points that do not need to be refined. Other parameters are not so important, so leave them as they are.

# For iterative unbending
MinDelta = 0.01
MaxRound = 10  # How many iterations?
```

Figure 3.4: Call xdisph to show the diffraction pattern
Figure 3.5: (A) Control panel of unbent program (top), the initial spline curve (red line), and the start of
the spline curve (blue line) superimposed on the raw cryo-EM image of KIF1A-microtubule complex. Only
control points set between the start and end points, as defined in the boxes at the left of the control panel,
are refined. (B) Comparison of the shift detected by cross-correlation between the 1D projection of a
segment and its mirrored projection. After three iterations of distortion, the shift decreased to less than one
pixel. (C) Images of the original in A, showing motor-microtubule complex over three iterations. The images
are shortened along the filament to enhance the moiré pattern of the microtubule and to demonstrate the
effect of unbending.

In figure 3.5, an example of unbending is shown. After editing the file, execute following command.

```plaintext
./auto_unbent d1343a.ub d1343a.ub.new
d1343a.ub
start = 150.000000 , end = 5700.000000
#<Spline:0x406a3bd4 @array=NArray.float(4,5577):
[ [ 0.0 , 151.0 , 171.0 , 11.89 ],
  [ 1.0 , 152.0 , 171.2 , 11.89 ],
  [ 2.0 , 152.9 , 171.4 , 11.89 ],
  [ 3.0 , 153.9 , 171.6 , 11.89 ],
  [ 4.0 , 154.9 , 171.8 , 11.89 ],
  [ 5.0 , 155.9 , 172.0 , 11.89 ],
  [ 6.0 , 156.8 , 172.2 , 11.89 ],
  [ 7.0 , 157.8 , 172.4 , 11.89 ],
  [ 8.0 , 158.8 , 172.6 , 11.89 ],
  [ 9.0 , 159.8 , 172.8 , 11.89 ],
```
opening d1343a.mrc file as MRC format

Number of columns, rows, sections .................. 798 5778 1
Map mode .............................................. 2
Start points on columns, rows, sections ............. 0 0 0
Grid sampling on x, y, z .............................. 798 5778 1
Cell axes .............................................. 1.000 1.000 1.000
Cell angles .......................................... 0.000 0.000 0.000
Fast, medium, slow axes ............................... X Y Z
Origin on x, y ........................................ 0.000 0.000
min, max, mean density ............................... -13291.691 -4641.763 -8437.582
Number of titles ..................................... 0

Titles :
Writing out MRC format file to: test.mrc ...
type code = 4
save as single precision float
Writing MRC finished
1 : 279.938494 692.000000 ->
background subtraction parameters
delta x = -0.003375
2 : 383.243608 1360.000000 ->
background subtraction parameters
delta x = 0.019633
3 : 447.272609 2296.000000 ->
background subtraction parameters
delta x = -0.017832
4 : 414.447279 3320.000000 ->
background subtraction parameters
delta x = 0.000075
5 : 335.173323 4316.000000 ->
background subtraction parameters
delta x = -0.006579
6 : 314.325953 5188.000000 ->
background subtraction parameters
delta x = -0.012281
1 : y = 692.000000, delta x = -0.003375
2 : y = 1360.000000, delta x = 0.019633
3 : y = 2296.000000, delta x = -0.017832
4 : y = 3320.000000, delta x = 0.000075
5 : y = 4316.000000, delta x = -0.006579
6 : y = 5188.000000, delta x = -0.012281
[nil, -0.00337492892358071, 0.0196328434324603, -0.0178318316499032, 7.54634939948971e-05, -0.00657948187809069, 0.0122808299962445]

If you find that the delta is large, please repeat the process after copying the new ub file to original as:

    cp d1343a.ub.new d1343a.ub

To make sure the program works fine, you can also take a look at shrink.mrc image using MRC viewing software such as bshow. This image is shrunk along the tube axis with factor specified by SHRINK.

The files specified by FFT_Image and FFT_fft can be used to inspect the diffraction pattern of the filament. The file specified by FFT_Image contains entire length specified by start and end point in .ub file.

For newer version of auto_unbent, you can also specify "unbent.final" and "unbent.result" files. Then you can omit the following two steps to convert the ub file to other format.

    ./auto_unbent d1343a.ub d1343a.ub.new unbent.final unbent.result
3.8 ruby-helix-gui

To start image analysis, ruby-helix-gui is useful.

```
ruby-helix-gui &
```

Alternatively, you can specify the parameter file

```
ruby-helix-gui foo.rb &
```

or an MRC image file:

```
ruby-helix-gui foo.mrc &
```

Then you will see the GUI front end for ruby-helix as shown in Fig 3.6.

Let’s start by filling the Image specifications (Fig. 3.7). Fill the file name of MRC image file. Please remove “.mrc” from the name.
3.9 Bootstrapping with Ruby scripts

Here, we start only from an image of helical specimen. No requirement for template big G files.

1. Generate unbent points file (usually unbent.final). This step is done by auto_unbent if you are using newer version of auto_unbent (later than June 2008). If necessary, the unbend file (.ub) should be converted to a "uniform" format using ub2unbent. Usually the conversion is done like:

```
ub2unbent foo.ub unbent.final
```

2. Generate spline points file. (spline.result) This step is done by auto_unbent if you are using newer version of auto_unbent (later than June 2008). Note that many operating system comes with spline command included in GNU Plotting Utilities. So, to specify the spline command comes from Ruby-Helix, you may need to use full path like here:

```
˜ mkikkawa/bin/spline <parameter_file> <start> <end> <output>
```

This program seems to have inconsistency with the Unbend, so if you notice it, please use new ruby and gsl-based program spline2. The usage is the same as spline.

3. Roughly find the truepitch. If you used auto_unbent, you can find the rough truepitch using correlation_matrix. Please copy the correlation_matrix script to your local working directory and modify the header. Usually modification is not necessary. There are two parameters you can play with:

```
aa = NArray.open_mrc("shrink.mrc")
Average_width = 10
```

4. Edit foo.rb - ImageFile - Start,End

5. cut_out foo Take a look at foo.hftout. Number of pages will be determined by HLXS and L01 (max k * L01 or max k * L10). So, if you want to see layer line please change these values. To identify the phase of layer lines, following graph will be useful.

![Graph of layer lines]

Then, if you find layer lines, specify the Bessel orders and height by N10,L10,N01,L01. By specifying the basic layer lines, all other layer lines will be automatically extracted.

6. scanLength This is optional. It may not necessary to refine it at this stage. However to understand the following refinement strategy, it is good to know that how the start postion of the cut out is refined. A program scanLength tries to extract layer line data using different cut out length.
3.10. CUI ANALYSIS – SCENARIO 3 – SEMI-AUTOMATED ANALYSIS

The best cut out length is judged by amplitude of the layer lines specified by two parameters, TargetLayerline and TargetRange. TargetLayerline specifies the height of the layer line and TargetRange specifies the range of big R to find maximum.

7. refine_omega_xshift foo Then edit HKR and re-run refine_omega_xshift foo. After this, please take a look at hlxs.log.

8. hlxflctf foo

hlxflctf extract data based on the control file foo.hlx. The control file specifies FFT file to be extracted and the layer lines to be extracted.

The output of the command is foo.far, foo.meq and foo.avg. These bigG files have 4th column, which is the absolute CTF value at each points. The data itself is not multiplied by CTF, but the signs are corrected. CTF value will be multiplied if you use hlxavgctf.rb command in the averaging step.

hlxflctf4 is an obsolete command. The command was replaced by hlxflctf.

3.10 CUI analysis – scenario 3 – semi-automated analysis

Edit ***total.rb. Sample is explained below.

3.10.1 Parameter file 1.a0000total.rb

First, specify the original image file.

```
# original image
ImageFile = "b1704a.mrc"
```

Then, specify the reference image. Actually, reference image is generated from a big G file specified by ReferenceAvg. In the case of tubes with polarity (eg. microtubule, actin), you need to pay attention to the polarity of the big G file. Then, give the reference image name by ReferenceImage. This image is generated according to the following parameters, Truepitch, RefYsize. Since RefYsize specifies the size of the reference image, depending on the sample, you need to change this value.

Since this reference generation takes a time, so if you want to use already generated reference file, please change Generate_ref_flag from true to nil.

```
#==============================================
# In order to find truepitch
#==============================================
ReferenceAvg = "/home/oda/data/C351/templates/6_60dr.avg"
ReferenceImage = "b1704a.ref" # to be generated
Truepitch = 1520.0 # angstrom
RefYsize = 128 # pixel
Generate_ref_flag = true # "true" to generate reference or "nil"
```

Following values are used to specify the start point and the end point of the spline curve. Based on the UnbentFile, SplineFile is generated.

```
# For spline curve generation
Spline_start = 20.0
Spline_end = 7315.0
UnbentFile = "unbent.final"
SplineFile = "spline.result"
```
CHAPTER 3. HELICAL IMAGE ANALYSIS – SCENARIO 3 – SEMI-AUTOMATED ANALYSIS

Following values specify how to cut out the image. This cut out is for finding the true pitch. Note that Start and End values are for \( x \)-coordinate. Therefore, it should be inside of the spline curve. If you are not sure, please take a look at SplineFile. The first column is the \( x \)-coordinate.

```
# How to cut out:==============================================================
Start = 800.0  #
End   = 5000.0  # End - Start should be even number
```

The pixel size should be given. Only \( x \)-direction will be fine.

```
PixelSizeX = 2.5  # angstrom
#PixelSizeY = 1.448  # optional. Calculated by (End-Start) * PixelSizeX / BoxSizeY
```

Specify the cut out width, which should be larger than the tube diameter. BoxSizeX and BoxSizeY should be \( 2^n \), and larger than CutoutWidth and End-Start.

```
CutoutWidth = 180  # pixels
BoxSizeX = 512
BoxSizeY = 8192
TubeRadius = 100  # pixels. note this is radius, not diameter
# if you are not sure about the radius, put larger value
```

Seems like there are unnecessary parameters. But leave it for backword compatibility.

```
Unbent_para_filename = UnbentFile
Spline_parameter_filename = SplineFile
```

Following parameters are mainly used for refinement of \( x \)-shift and \( \omega \)-tilt. First, specify the template parameter file with TemplateFile. Other parameters will be explained later.

```
#===========================================================
# For auto_refine
#===========================================================
TemplateFile = "b1704a.rb"
Title = "b1704a"

# To be determined by find_truepitch
StartEndArray = 
[ [1146.0, 1710], [1709.0, 2307.5], [2305.0, 2901], [2901.0, 3528.5], [3528.0, 4155]

# pair of start and end point

# Flags
CopyFromTemplate = true
CutOut = true  # for prepare
RefineStart = true
RefineOmegaXshift = true
Hixctf = nil
RefineUnbent = true
CleanUp = nil
```

In most cases, you can set the start and end of xdash according to the spline result. However if the image is full of noises, the amplitude of the waves of find_truepitch result can be too low to analyze. In that case the start and end of xdash will be set according to the cut out value of the previous procedure on AVS. It would be better if the length of the cut out xdash is longer than that of original cut out. Set the reference file according to the orientation of the MT.

to find the true pitch, run

```
find_truepitch XXXXtotal
```
3.10. CUI ANALYSIS – SCENARIO 3 – SEMI-AUTOMATED ANALYSTER 3. HELICAL IMAGE ANALYSIS.

If the peaks of waves are obvious, all you have to do is just pick up the coordinates of the peak. But in most cases it is not that easy. First find a clear peak in which the height of the green line is within ±3. Next search for adjacent peak 600±50 away from it. You can figure out peaks relatively easy if you tighten the criteria of the height of the green line to ±2 or ±1 when the result of `find_truepitch` is full of noises. If you cannot find even a single clear peak, the pattern of the green lines will help you. Keep it in mind that the pseudopeaks are often higher than the true peaks. When the wave is blunt and you cannot choose one spike as a peak, choose a peak, which is nearest to the middle of the two adjacent peaks.

Put the start and end points in to `****total.rb` file, like follows:

```ruby
# To be determined by find_truepitch
StartEndArray = [[1146.0, 1710], [1709.0, 2307.5], [2305.0, 2901], [2901.0, 3528.5], [3528.0, 4155]]
```

This value will be further refined in the following step.

3.10.2 Parameter file 2: a0000.rb

```ruby
# original image
ImageFile = "b1847a.mrc"

# How to cut out:=================================
Start = 1146.0 #
End = 1710 #

PixelSizeX = 2.5 # angstrom
#PixelSizeY = 1.448 # optional. Calculated by (End - Start) * PixelSizeX / BoxSizeY

CutoutWidth = 180 # pixels
BoxSizeX = 512
BoxSizeY = 1024
TubeRadius = 100 # pixels. note this is radius, not diameter
# if you are not sure about the radius, put larger value

UnbentFile = "unbent.final"
SplineFile = "spline.result"

# Basic helical parameters of the filament
N10 = -15
L10 = 1
S10 = 10

N01 = 2
L01 = (((End - Start) * PixelSizeX / 81.2 + 0.5).to_i) # This is optimized for microtubules. 21
S01 = 10

# standard helical parameters (used for hixfl: extract bigG from FFT)
StdN10 = -15
StdL10 = 1

StdN01 = 2
StdL01 = 19

TargetResolution = 7.0 # angstrom

# Title = "b1012a" #optional
```

# Tobe determined by find_truepitch
StartEndArray = [[1146.0, 1710], [1709.0, 2307.5], [2305.0, 2901], [2901.0, 3528.5], [3528.0, 4155]]

This value will be further refined in the following step.

3.10.2 Parameter file 2: a0000.rb
CHAPTER 3. HELICAL IMAGE ANALYSIS – SCENARIO 3 – SEMI-AUTOMATED ANALYSIS

# Defocus parameters
Cs = 3.3 # mm
Acc_voltage = 200 # KV
AContrast = 0.07 # amplitude contrast
Defocus1 = 13599.08
Defocus2 = 15645.48
AngleAst = 63.55

# Out-of-plane-tile
Omega = 9.3 # degree
XshiftBox = 0.0 # grid unit
Xshift = 0.0 # grid unit

# File for list of peak positions
PeakFile = "HKR"

# Orientation of the filament
Orientation = "R" # "F" for forward, "R" for reverse

# Layer lines used for hlxs [h,k] pairs.
HLXS = [[0,0],[1,0],[2,0],[-1,1],[0,1],[1,1],[-1,2],[0,2],[1,2],[0,3],[0,4]]
Bfactor = 35.0

3.10.3 Refinement strategy

Initial auto_refine ****total should be done with the following flags.

- If CopyFromTemplate is true, always the segment parameter files are copied from the template file.
- If CutOut is true, the image will be cut out from the original image. This parameters are usually on.
- If RefineStart is true, scanlength will be executed. Usually turned on only once during the refinement, because this process takes a lot of time. This function works only when two parameters, TargetLayerline and TargetRange are set properly. Often TargetLayerLine are written with equation containing L01. Therefore, if L01 is written with equation, the equation should work properly.

Therefore, for getting an initial structure, it is better to turn off this function and manually type-in L01 or L10 and then run scanLength manually.

- If RefineOmegaXshift is true, this will allow the refinement of x-shift and \( \omega \)-tile refinement. If the image is good, usually the program automatically find the right value and you can leave this value true. However, some in the case of noisy image, the x-shift and \( \omega \)-tilt may not converge. In this case, manually change the individual files and turn this switch off.

- If Hlxctf is true, the final far, nea and avg data will be extracted with a command hlxflctf. For high resolution data, this process takes a time, so please keep the flag off until the last moment.

- If RefineUnbent is true, according the x-shift value for each segment, UnbentFile will be updated with the command refine_unbent. RefineUnbent = true

# Flags
CopyFromTemplate = true
CutOut = true # for prepare
RefineStart = true
RefineOmegaXshift = true
Hlxctf = nil
RefineUnbent = true
CleanUp = nil
3.11. AVERAGING

After this run, please copy and paste the refined start positions to

```
# To be determined by find_truepitch
StartEndArray = [[1146.0, 1710], [1709.0, 2307.5], [2305.0, 2901], [2901.0, 3528.5], [3528.0, 4155]]
```

Typical second run will be like this:

```
# Flags
CopyFromTemplate = nil
CutOut = true # for prepare
RefineStart = nil
RefineOmegaXshift = true
Hlxctf = nil
RefineUnbent = true
CleanUp = nil
```

Since RefineStart takes a lot of time, turn this switch off.

Finally, when you are satisfied with the refined values, extract big G data. In fact, you can do the same process with hlxflctf. The output files of hlxflctf are foo.avg, foo.nea and foo.far.

Also by putting CleanUp as true, auto_refine will clean up bulky files, such as foo.fft, foo.img, corr.img and compress foo.hlxslog.

```ruby
# Flags
CopyFromTemplate = nil
CutOut = nil # for prepare
RefineStart = nil
RefineOmegaXshift = nil
Hlxctf = true
RefineUnbent = nil
CleanUp = true
```

3.11 Averaging

3.11.1 What should be done?

First, to average dataset, all the AVG files should be brought to same origin. The shift of the phase is defined by \((\varphi, \bar{z})\).

Second, the CTF effect should be corrected.

3.11.2 How?

To bring all the data to same origin, pick up a reference. Then, Put CTF multiplied AVG files into one directory. For this purpose, we currently two kind of programs are used.

3.11.3 hlxavgctf

Please copy the script (ruby-helix/app/hlxavg) to your local directory. Edit the header. Actually this script will automatically run hlxfit, if Reference_filename is given.

```ruby
#########################################
# set helical parameters
#########################################
Tg = BigG.new
Tg.n10 = -14
```
CHAPTER 3. HELICAL IMAGE ANALYSIS.

3.11. AVERAGING

\[
\begin{align*}
\text{tg.l10} &= 1 \\
\text{tg.n01} &= -1.5 \\
\text{tg.l01} &= 19 \\
\text{tg.repeat} &= 1520.0 & \text{# angstrom, used in make layer lines and hlxfit} \\
\text{tg.resolution} &= 12.0 & \text{# angstrom, used in make layer lines and hlxfit} \\
\text{tg.make.ll} &= (7.8125E\text{-}4, 200.0) & \text{# deltaR (1/angstrom), radius (angstrom)}
\end{align*}
\]

# parameters for hlxfit.
Reference_filename = "test.avg"
# If this is null, hlxfit will not be done.
# Please do it manually.

# Array of file names to be fit to Reference.
array = ["b1388a.nea", "b1388b.nea", "b1388c.nea", "b1388d.nea", "b1388d.far", "b1388e.far", "b1388f.far"]

Output_filename = "test1.avg"

Typically you will run the following command

```
./hlxavgctf *.nea *.far
```

This command execute hlxfit5, average big Gs whose phase residual smaller than Residual_Cut_off, and write out the averaged big G file to the file specified by Output_filename.

Note about the data in the averaged big G file:
The data columns (second and third columns) are:

\[
F_{\text{total}} = \sum F \times CTF
\]  \hspace{1cm} (3.3)

and the CTF column (fourth column) is:

\[
CTF_{\text{total}} = \sum CTF^2.
\]  \hspace{1cm} (3.4)

3.11.4 Select good dataset.

There are several ways to examine the quality of each big G file.

Polarity (specific to microtubule) Sometimes, polarity of the filament was not determined right. In this case, phase residual of (1, 0) layer line becomes bad.

To examine this,

3.11.5 CTF correction scheme

Please see Table 3.3.

hlxfit5 The program was developed for averaging filaments or tubes even with a seam, but can also be used for averaging filaments/tubes without a seam. (Please do not use hlxfit, because the program is obsolete).

First, fit a file to reference, like:

```
hlxfit5 b1388c.nea test.avg >>log
```
3.11. AVERAGING

CHAPTER 3. HELICAL IMAGE ANALYSIS.

<table>
<thead>
<tr>
<th>data</th>
<th>data and phase columns</th>
<th>CTF column</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRC FFT</td>
<td>$F$</td>
<td>$F \times \text{sign}_{CTF}$</td>
</tr>
<tr>
<td>nea, far, avg (big G files)</td>
<td>$\sum F \times CTF$</td>
<td>$\sum CTF^2$</td>
</tr>
<tr>
<td>after hexavg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>after ctfdivide</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Summary of CTF correction scheme.

Please take a look at the log.
First the program tries to determine basic helix from the given layer lines.

Basic $(n,l)$ vectors were not given, so guess from layer lines...

<table>
<thead>
<tr>
<th>$n_{10}$</th>
<th>$l_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-15.00</td>
<td>1.00</td>
</tr>
<tr>
<td>-1.50</td>
<td>28.00</td>
</tr>
</tbody>
</table>

Actually the algorithm for finding the basic helix is not extensively tested. Therefore, if you find a bug, please tell it to the author.

Then, in the next step, the program starts searching for the best $\varphi$ and $z$. The log will look like:

<table>
<thead>
<tr>
<th>Target precision $\varphi$: 0.001022, $z$: 0.000566</th>
</tr>
</thead>
<tbody>
<tr>
<td>max $n$ = 32.80 $l$ = 57.64</td>
</tr>
<tr>
<td>$y \times x$</td>
</tr>
<tr>
<td>===============================================</td>
</tr>
<tr>
<td>-0.27</td>
</tr>
<tr>
<td>-0.22</td>
</tr>
<tr>
<td>-0.16</td>
</tr>
<tr>
<td>-0.11</td>
</tr>
<tr>
<td>-0.05</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.11</td>
</tr>
<tr>
<td>0.16</td>
</tr>
<tr>
<td>0.22</td>
</tr>
<tr>
<td>0.27</td>
</tr>
</tbody>
</table>

As the refinement cycles go, more layer lines are included.
Although this program assumes the repeat length as $2\pi$ for fitting, you still need to specify the repeat length (in angstrom) to specify the range of layer lines used for fitting.

As a result, $\varphi$ and $z$ are “symmetric” in a sense. To specify the repeat length in angstrom,

**export** REPEAT=1520

or if you use tcsh

**setenv** REPEAT 1520

Also the range of search is automatically determined from the layer lines.

At the end of the run, you will see a table like this:

<table>
<thead>
<tr>
<th>minimum phi: 0.398473, $z$: -0.004360, residual: 79.306297 Applying phi,z shift: 0.398473,-0.004360</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase residual for each test layer line at global minimum</td>
</tr>
<tr>
<td>L N 1/Z weight points ref-amp-test residual</td>
</tr>
</tbody>
</table>
3.12 Asymmetric Helical Reconstruction.

Asymmetric helical reconstruction is a new helical analysis method that can analyze “helical” structures with a seam [11]. Mostly the analysis is the same.

<table>
<thead>
<tr>
<th>i</th>
<th>g.u.</th>
<th>% (degree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>16.99 44.85</td>
</tr>
<tr>
<td>1</td>
<td>-15.0</td>
<td>10.25 61.92</td>
</tr>
<tr>
<td>2</td>
<td>-30.0</td>
<td>3.33 56.63</td>
</tr>
<tr>
<td>3</td>
<td>-45.0</td>
<td>1.51 92.58</td>
</tr>
<tr>
<td>4</td>
<td>-60.0</td>
<td>0.72 83.24</td>
</tr>
<tr>
<td>5</td>
<td>58.5</td>
<td>0.72 104.11</td>
</tr>
<tr>
<td>6</td>
<td>43.5</td>
<td>1.98 91.47</td>
</tr>
<tr>
<td>7</td>
<td>28.5</td>
<td>3.21 108.64</td>
</tr>
<tr>
<td>8</td>
<td>13.5</td>
<td>5.10 99.08</td>
</tr>
<tr>
<td>9</td>
<td>-1.5</td>
<td>8.42 95.74</td>
</tr>
<tr>
<td>10</td>
<td>-16.5</td>
<td>4.52 97.20</td>
</tr>
<tr>
<td>11</td>
<td>-31.5</td>
<td>1.98 103.56</td>
</tr>
<tr>
<td>12</td>
<td>-46.5</td>
<td>1.26 65.66</td>
</tr>
<tr>
<td>13</td>
<td>-61.5</td>
<td>0.70 103.20</td>
</tr>
<tr>
<td>14</td>
<td>57.0</td>
<td>0.51 122.03</td>
</tr>
<tr>
<td>15</td>
<td>42.0</td>
<td>1.19 116.13</td>
</tr>
<tr>
<td>16</td>
<td>27.0</td>
<td>2.62 77.13</td>
</tr>
<tr>
<td>17</td>
<td>12.0</td>
<td>6.16 45.56</td>
</tr>
<tr>
<td>18</td>
<td>-3.0</td>
<td>12.01 58.85</td>
</tr>
<tr>
<td>19</td>
<td>-18.0</td>
<td>3.55 103.84</td>
</tr>
<tr>
<td>20</td>
<td>-33.0</td>
<td>1.86 95.51</td>
</tr>
<tr>
<td>21</td>
<td>-48.0</td>
<td>1.03 93.41</td>
</tr>
<tr>
<td>22</td>
<td>40.5</td>
<td>0.83 109.65</td>
</tr>
<tr>
<td>23</td>
<td>25.5</td>
<td>1.68 95.76</td>
</tr>
<tr>
<td>24</td>
<td>10.5</td>
<td>2.38 88.50</td>
</tr>
<tr>
<td>25</td>
<td>-4.5</td>
<td>2.38 100.20</td>
</tr>
<tr>
<td>26</td>
<td>-19.5</td>
<td>1.73 97.69</td>
</tr>
<tr>
<td>27</td>
<td>-34.5</td>
<td>1.08 96.73</td>
</tr>
<tr>
<td>28</td>
<td>9.0</td>
<td>0.32 161.53</td>
</tr>
</tbody>
</table>

residual = 79.306

This is the summary of layer lines. Make sure that the strong layer lines have smaller residual values. Usually it is smaller than 50, can be 60, but I do not like 70. But it also depends on the resolution range.

The final \( \varphi, z \) and phase residual (in degree) will be written to \( \text{foo.avg.rdl} \) file.

**ctdivide**  After averaging, the amplitude should be divided by CTF factor. This process is usually done by hlxavgctf.

If you want to do it manually, follow this:

```
ctdivide foo.avg >food.avg
```

This program calculates:

\[
\frac{\sum F \times CTF}{\sum CTF^2}.
\]

(3.5)

There is another program **ctf-half-divide**. This program is used to examine individual big G files (nea, far, avg). If the CTF is less than 0.5, the amplitude is divided by 0.5.
3.12.1 Judging the seam side and the seam-free side.

Microtubule has seam on one side, called seam side. To judge which near or far side is the seam side, use the `plot8n-lg` command. Please refer to corresponding section. Another useful command is `summarize`.

```
./summarize *.nea *.far
```

will give you a list of filename, phase residual, phase residual for 8 nm layer line and you can select specified files based on the values.

3.13 Determining effective resolution using Fourier Shell Correlation

To determine the effective resolution, calculate Fourier Shell Correlation, often abbreviated as FSC [14]. First, you need to divide the dataset into two groups. Copy `summarize` script from `ruby-helix/app/summarize` and then modify it. For FSC, change `DivideHalf` to true.

```
DivideHalf = true # or false
```

Execute it and copy the output into `hlxavgctf`. Generate two averaged big G files. For example, `example-Ad.avg` and `example-Bd.avg`.

Second, apply `limitR`. Copy the script from `ruby-helix/app/limitR` and edit the header. Then execute it:

```
./limitR example-Ad.avg example-Bd.avg
```

The program generate `solexample-Ad.avg` and `solexample-Bd.avg`. Third, execute `fsc`.

```
./fsc solexample-Ad.avg solexample-Bd.avg 3085.6
```

Here, 3085.6 is the repeat length in Angstrom. You will see a plot like in Fig. 3.8. Alternatively, you can also use

```
phaseResidualPlot solexample-Ad.avg solexample-Bd.avg 3085.6
```
CHAPTER 3. HELICAL IMAGE ANALYSIS

3.14. VISUALIZING 3D MAPS

Then, you will see a plot like in Fig. 3.9. This plot was used in Kikkawa 2000 [15] and Hirose 2006 [16].

3.14 Visualizing 3D maps

There are several ways to visualize 3D maps generated from a big G file. To choose a particular program, please see Fig 3.10.

3.14.1 Generate MRC and XPLOR-format density maps

First, you need to generate a 3D MRC density map using the 3D-fourier-synthesis.rb script (see section 4.1). The script reads the big G file specified in the script and generates a MRC-format density map and an XPLOR-format density map. The XPLOR-format density map can be read using PyMol.

3.14.2 UCSF Chimera

The easiest way to visualize the MRC map is to use UCSF Chimera\(^2\). Just read the MRC map and select a threshold value.

3.14.3 O, PyMol, and VMD

Alternatives to visualize a density map is to use O, PyMol, or VMD. These softwares are generally used for model building by X-ray crystallographer, but it is also used for docking atomic models into a density map.

\(^2\)http://www.cgl.ucsf.edu/chimera/
3.14. VISUALIZING 3D MAPS

Figure 3.10: Visualizing 3D density map

derived from cryo-EM as well. Choice of software depends on what you want to work on the map.
If you will dock atomic models manually, O is our first choice.
If you dock atomic models using Situs\(^3\), VMD or UCSF Chimera are convenient.
When you make figures for publications, PyMol might also a good choice.

3.14.4 Convert the MRC map for O

convert the MRC format file to BRIX map format file like following:
```
mrc2map -i XXX.mrc -o XXX.map
```
Execute the O and read the map like
```
map_file XXX.map
map_obj com
map_param 500 500 500 160 orange : :
map_dr
```

3.14.5 Convert the MRC map for PyMol

Please use mrc2xplor command (see 4.22 as well) to convert a MRC map into the XPLOR format, which

\(^3\)http://situs.biromachina.org/
\(^4\)http://pymol.sourceforge.net/
Below is an alternative conversion protocol:
To convert the MRC map file to XPLOR format, use mapman as follows:

MAPMAN > re
New map ? ( ) map
File name ? (not defined) best8d.map
Format ? (CCP4) BRICK
Reconstructing DSN6 ("BRICK") file
Little-endian machine; will swap header
Prod : ( 9.900E-01)
Plus : ( 0)
Nr of pages : ( 512)
Map size : ( 216000)
Reading map ...
Map read into memory
Cell axes (A) : ( 120.000 120.000 120.000)
Cell angles (d) : ( 90.000 90.000 90.000)
Grid axes (pts) : ( 60 60 60)
Origin (pts) : ( 0 0 0)
Extent (pts) : ( 60 60 60)
Spacegroup set to P1
Map read into memory – calculating statistics
Sum of density in map : ( 2.250E+07)
Requested dynamic range : 0.0000E+00 2.5657E+02
Value of Prod and Plus : 9.9390E-01 0
Actual dynamic range : 0.0000E+00 2.5657E+02
MAPMAN > wr
Which map ? (map)
Which file ? (best8d.map) best8d.2.cns
Format ? (CCP4) CNS
Writing ASCII X-PLOR map ... 
CPU total/user/sys : 1.3 1.2 0.0
MAPMAN > quit

3.14.6 Reading an XPLOR format file and visualize isomesh in PyMol

To read the electron density maps: (from PyMol manual)

load 2fofc.xplor.map1 # type inferred from the extension
load 2fofc.map.map1,1,xplor # type explicitly provided

Map objects are used to store the data and are represented by a wire-frame brick in space indicating the extent of the map. An arbitrary number of mesh or dots objects can be created from a given map using the “isomesh” and “isodot” commands.

isomesh msh1, map1,1.0 # display an isosurface-mesh at level 1.0 over
# the entire map object “map1”
isomesh msh2, map1,1.0,(chain A),3.0 # display isosurface-mesh at 1.0
# in a brick about chain A with a
# border of 3.0 Angstroms

See “help isomesh” or the reference section for additional information.

5http://xray.bmc.uu.se/usf/mapman_man.html
3.14.7 Convert the MRC map for VMD

VMD\(^6\) is also used for visualizing density maps and atomic models. The conversion can be done using two programs that come with Situs.
   First convert MRC map format using `conformat`.
   Second, use `volcube` to select an isosurface.
   For details, please take a look at Situs tutorial page.

3.14.8 With AVS

Please take a look at Fig 3.12.

**To generate a MRC density map in AVS** Using bigG to 3D, generate 3D map inside of AVS, confirm that the map looks good. (please use `3D.net` for example.)
   Convert the float data to byte image. (with a “field to byte” module) This is necessary because the BRICK format, which is readable to O, uses byte data to represent one data point.
   Save the file using `writeMRC`. Before pushing the “outswitch”, please remember to put \(x, y, z\) length in angstrom which are equal to the size of the cell. For example, if the grid size is 2.5Å and the size of the volume is 80 pixels, then the length is 200Å.

3.15 Docking atomic models using colores

To use colores

\(^6\)http://www.ks.uiuc.edu/Research/vmd/
Colores is used to dock atomic model into low or medium resolution cryo-EM density map. See the colores tutorial for details.

First, your 3D density map should be converted into the Situs format.

```
map2map i12.mrc i12.situs
```

Then select

3: MRC (= old CCP4) binary (limited support*)

option. You will be asked about the size of grid.

Perform actual fitting.

```
colores i12-masked.situs 1MKJ.pdb --res 15.0 --deg 20 --nprocs 4 --explor 5
```

In this case, four CPUs are used for docking. This will significantly improves the speed of docking. Examine the result using VMD or PyMol.

### 3.16 Obsolete procedures

#### 3.16.1 Unbend on Mac OS X

If you have Mac OS X 10.4 (Tiger) or 10.5 (Leopard), please use unbend. In Kikkawa’s lab, most people uses the program for unbending. This procedure assumes the tube axis is parallel to the vertical axis of the image. See Figure 3.13.

1. convert tiff file to MRC format for Unbend using a command tiff2mrc.

2. Drag and drop the MRC file to Unbend icon in the Dock. You can also simply double click the MRC file to open the image file. Adjust the contrast and brightness (in the Image menu), if necessary.

3. Click the “sort by y”.
3.16. OBSOLETE PROCEDURES

3.16.2 Unbending with bshow.

As we now have unbent for Linux, the following section is obsolete. However, if you can’t install ruby-gnome2, try following:

As not all laboratory uses Mac, we have also made a script to convert bshow markers to the ub file.

1. convert tiff file to MRC format using a command tiff2mrc.

2. open the MRC file using bshow (on unix) or bshowX (on Mac OS X) Bsoft\(^7\).

---

\(^7\)http://lsbr.niams.nih.gov/bsoft/

---

Figure 3.13: A screen shot of “unbend”.

4. Left click the center of the tubes. If you click on the boxed point, it will remove the box. If you are going to use tube_tracer, just click once.

5. Set start and end points.


Use a script called tube_tracer to automatically put control point along the microtubule. Open “temp.ub” file by Unbend.
CHAPTER 3. HELICAL IMAGE ANALYSIS.  3.16. OBSOLETE PROCEDURES

Figure 3.14: a screen shot of “bshow”.

```
bshow foo.mrc
Adjust the contrast, brightness, and scale.
3. Select Image -> Markers and linkers
4. Left click the center of the tubes and put markers (Fig. 3.14).
5. Save the markers as the STAR format.
6. convert the STAR file to .ub format.
   star2rb foo.star foo.ub
7. Edit the .ub file. Please change ClipEnd and ClipStart. Default ClipStart is 0 and ClipEnd is the size of the image.
8. Follow the procedure described below auto_unbent.

Note that the file generated by this star2ub is not compatible with Mac OS X Unbend. So please do not try to read the .ub file using Mac OS X Unbend.

49
Chapter 4

Script References.

4.1 3D-fourier-synthesis.rb

Copy the script to your working directory and edit the header. The header looks like:

`#========= big G and little G parameters ===========
BigG_file_name = "ADP16-6d.avg"
Resolution = 10.0
Little_g_Mesh_Size = 2.5 # Angstrom
Little_g_Array_Size = 120
Repeat_Length = 2560 # Angstrom
#========= Map parameters ===========
Map_X = 90
Map_Y = 120
Map_Z = 110
Map_Origin_X = -50 # Angstrom
Map_Origin_Y = 80
Map_Origin_Z = 58
Map_Mesh_Size = 1.0 # Angstrom
Map_file_name = "test.mrc"
`

So I assume the use of the script is self-explanatory. The calculation will take one to ten minutes dependind-
ing on the bigG file and the map size.

4.2 auto_unbent

4.3 bG-lG-3D-renumber-rot-lg.rb

This script is used to convert one type of helix to another. The script was mainly designed for microtubules, but it can be used for other helical structures.

The script consists of eight steps.

1. Read a big G file.
4.4. APPLY FOURIER-BESSEL TRANSFORM TO THE BIG G AND OBTAIN A SMALL G.

2. Apply Fourier-Bessel transform to the big G and obtain a small g.

3. Generate 3D density map from the small g.

4. Pick up an asymmetric unit.

5. Rotate and translate the asymmetric unit.

6. Generate a re-indexed small g.

7. Back Fourier-Bessel transform to obtain a bigG from the new small g.

8. Generate density map as a test.

4.3.1 Read a big G file

4.4 Apply Fourier-Bessel transform to the big G and obtain a small g.

4.5 correlation

4.6 correlationTube

Calc correlation between an unbent image and a test image. The test image can be smaller than the unbent image. The center of the test image is used as the origin. This script is based on fiind_truepitch.

Experimental program.

Required files Standard foo.rb file. Fields that is used is listed.

- PixelSizeX
- ImageFile
- SplineFile
- Start, End
- CutoutWidth

4.7 check_installation

To identify lack of libraries you need, you can use a script check_installation in ruby-helix/app directory. If your installation is complete, you should see like this

$ ruby check_installation
Start checking installation
Checking Ruby/GSL installation to see whether compiled with NArray
Ruby/GSL is OK
Finished with no problems!
4.8 correlation_matrix

See Figure 4.1.

Designed for finding truepitch of filaments or tubes. After auto_unbent was executed, the program generates a file test.mrc. This file can be used to find the rough truepitch length. Copy correlation_matrix from the template. Edit Average_width = 10 in the header. If you used SHRINK = 8, each pixel is considered as 8 pixels and in total 80 pixels will be averaged to represent one pixel in matrix.mrc. Run correlation_matrix

Then examine matrix.mrc by MRC viewer. It should look like Fig. 4.1. In this figure, black region has higher correlation value.

Using this figure, please identify the truepitch.

4.9 ccd2ctf

This command is used to obtain CTF curve from a CCD image. First, please convert your CCD image into a MRC format image. Copy the script to your local directory and modify the script.

./ccd2ctf image.mrc
c tf result.mrc | matrix2xplot | xplot

This should give you a ctf curve.

4.10 cut_out

Cut out an image along the spline curve specified in —.rb— parameter file. This is an initial step of the helical reconstruction. This script generates following files:
4.11 fft

This command was made to be compatible with MRC’s Image 2000 fftrans. Currently only forward transform is performed.

```plaintext
fft source.mrc destination.fft
```

4.12 filter_radius

4.12.1 What it is?

The program is designed to extract something bound to tube. When the structure of the tube is well understood, but not the bound stuff, this program will be useful. In principle, this program subtract known helical portion from a given image.

4.12.2 How it works?

The core of the filter is the following code.

```plaintext
def filter_radius!(pixel_size,dim,rmin,rmax,pixel_size_R,dim_R)
    self.fourier_bessel!(pixel_size,dim)
    printf("rmin = %f rmax = %f\n",rmin,rmax)
    self.limit_radius!(rmin,rmax)
    if( self.data != nil ) then
        self.fourier_bessel!(pixel_size_R, dim_R)
    end
end
```

In principle, the program extracts layer lines specified in Title.hlx from Title.fft file. Then, each near and far side layer lines are averaged and transformed into the real space (little g). In the real space, only the region specified by inner_radius and outer_radius is selected and back-transformed to reciprocal space.

Finally, this layer line data is subtracted from the FFT image.

So, the specified region is subtracted. It is also important that Bessel order for the layer line is correctly assigned for the known part.

4.12.3 How to use?

Analyze the tube part that you want to subtract as a helical object. The tube can include more than one true pitch, but should have integer number of true pitch, so that all the layer lines are at the integer height. In principle, what you need to prepare is Title.fft and Title.hlx. These files are usually generated by preparing and analyzing the tube based on a parameter file foo.rb. Then,

```plaintext
filter_radius3 Title inner_radius outer_radius output.fft
```
CHAPTER 4. SCRIPT REFERENCES.

4.13 header

Print header information of a MRC file.

4.14 helix_boxer

Designed for MSA analysis of the tubular crystal. The script is based on cut.out. Therefore, the control file is almost same as the control file for cut.out.

Important parameters in foo.rb file.

- CutoutWidth:
- BoxSizeX:
- BoxSizeY:
- L01:
- Title: specify the output file names.

See figure 4.2.

Output of the script is “Title” given in the foo.rb paremeter file + numbers. +m is a mirror images.

After generating a set of images, there are two ways to

4.15 hlxfit5 and hlxfit6

Ruby version of helix fitting. Fit two big G files, and calculate the best phi and z values. hlxfit6 calculates also rscale. To execute these scripts, environmental variables RESOLUTION and REPEAT should be set. Both RESOLUTION and REPEAT should be given in angstrom. REPEAT is only used for limiting the resolution, but not in z-shift. For z-shift, the repeat length is assumed to be 2π.

In the hlxfit6 script, there is also two more variables. Thus, it is recommended to copy the script to your own directory and change the values in the script.

4.16 hlxflctf

See section 8.
4.17 hlximage

Generate a 2D image from a big G file. Mainly used to make a reference for find_truepitch. The program requires a template image file, from which several parameters are read from its header.

Here is the list of arguments:

1. big G file
2. template image file
3. output file (to be generated)
4. true pitch length (angstrom)
5. start x
6. end x
7. start y
8. end y

Usage:

hlximage a2150a.avg a2150a.img result.img 2000 0 256 0 1024 0

reading bigG file : a2150a.avg ...
finished
opening a2150a.img file as MRC format
20th century format map
Number of columns, rows, sections .............. 256 1024 1
Map mode ........................................ 2
Start points on columns, rows, sections .......... 0 0 0
CHAPTER 4. SCRIPT REFERENCES.

4.18. LIMIT_RESOLUTION

Grid sampling on x, y, z ....................... 256 1024 1
Cell axes ...................................... 2.540 1.718 0.000
Cell angles .................................... 0.000 0.000 0.000
Fast, medium, slow axes ...................... X Y Z
Origin on x,y,z ................................ 127.500 0.000 0.000
min,max,mean density .......................... -20945.023 36040.430 1328.460
Number of titles ............................... 0

Titles:
0.000000 256.000000 0.000000 1024.000000
Fourier synthesis 2D started
n= -40 , l = 4  pitch = 500.000000
n= -30 , l = 3  pitch = 666.666667
n= -14 , l = 101 pitch = 19.801980
n= -23 , l = 77  pitch = 25.974026
n= -3 , l = 75  pitch = 26.666667
n= 27 , l = 72  pitch = 27.777778
n= -32 , l = 53  pitch = 37.735849
n= -2 , l = 50  pitch = 40.000000
n= 9 , l = 24  pitch = 83.333333
n= -10 , l = 1  pitch = 2000.000000
n= 19 , l = 23  pitch = 86.956522
n= 29 , l = 22  pitch = 90.090909
n= 16 , l = 98  pitch = 20.408163
n= -4 , l = 100 pitch = 20.000000
n= 17 , l = 73  pitch = 27.397260
n= 18 , l = 48  pitch = 41.666667
n= -21 , l = 27  pitch = 74.074074
n= -1 , l = 25  pitch = 80.000000
n= -20 , l = 2  pitch = 1000.000000
n= -12 , l = 51  pitch = 39.215686
n= 6 , l = 99  pitch = 20.202020
n= 7 , l = 74  pitch = 27.027027
n= -11 , l = 26  pitch = 76.923077
n= 28 , l = 47  pitch = 42.553191
n= 39 , l = 21  pitch = 95.238095
n= -22 , l = 52  pitch = 38.461538
n= -13 , l = 76  pitch = 26.315789
n= 8 , l = 49  pitch = 40.816327
n= -31 , l = 28  pitch = 71.428571
n= 0 , l = 0
Fourier synthesis 2D finished
Writing out MRC format file to : result-ruby.img ...
type code = 4
save as single precision float
Writing MRC finished

4.18 limit_resolution

Read a bigG file and limit the resolution of the bigG.

limit_resolution <original bigG file> <resolution in Å> <repeat length>
4.19  **line_filter_2D.rb**

Read an image file and apply line filter. Used to see the filtered image. The image should contain filament along y-axis. To apply filter, copy the file to your local directory and modify the headers.

```ruby
#==============================================
filter_lines = [0,49,98,148,197,296]
filter_width_y = 5
filter_width_x = 50
Input_file = "c00917.img"
FFT_ABS_file = "fft.mrc"
Output_file = "filtered.mrc"
#==============================================
```

`filter_lines` specifies the lines to be filtered. This number corresponds to the y-axis coordinated in the FFT file. To make sure, you are filtering the right file, it might better to check the `FFT_ABS_file` file.

**4.20  llplot**

Read a ruby control file and

**4.21  mrc2situs**

Convert a MRC type map to a Situs format map. Requires external map2map command. Voxel size and origin information are derived from the map itself.

```bash
mrc2situs source.mrc destination.situs
```

**4.22  mrc2xplor**

Convert a mrc type map to an Xplor ASCII format map. This script was designed for pymol and probably only works for single float or float.

```bash
mrc2xplor source.mrc destination.xplor
```

**4.23  mrcImageCTFCompensation**

This is an EOS program clone, to inverse CTF effect of MRC images.

```bash
mrcImageCTFCompensation -i test.mrc -o test-out1.mrc -kV 200 -Cs 1.5 -df 40000 -A 0.03 -m 1
./mrcImageCTFCompensation -i test.mrc -o test-out2.mrc --kV 200 --Cs 1.5 --df1 40000 --df2 40000 --am
```
4.24 **tif2mrc**

Convert 16 bit tiff image to MRC single float image. This program convert the density according to the following equation.

\[
\rho_{new} = \log_{10}(\frac{\rho_{scan} - \min + 1}{\max - \min}) \times \frac{32768}{\text{Max}_{OD}}
\]  

(4.1)

Here, we use \(\min = 0.0\), \(\max = 65536.0\), \(\text{Max}_{OD} = 4\).

The above equation is used to convert tiff file generated by Leafscan 45. Depending on your scanner property, you can change this value. This tif2mrc script try to detect scanner model based on the header information stored in the tiff file. Currently two models are auto-detected:

1. Scitex LeafScan 45
2. Nikon Coolscan 9000ED

To use the command type

```
tif2mrc source.tiff output.mrc 5
```

The last argument is the shrink factor, which can be omitted.

For other options, see below:

**Usage:**
tif2mrc [options]

- \(-t\), \(--transpose\) Transpose the image
- \(--nikon\) set the scanner to Nikon
- \(--leafscan\) set the scanner to LeafScan (default)
- \(--shrink\) Shrink image by the factor

4.25 **phaseshift**

Shift the phases of the big G file. The script was a perl script, but recently rewritten in Ruby. phaseshift2 is the same command as phaseshift now. Usage:

```
phaseshift original_avg_file phishift zshift ipole <rscale> new_avg_file
```

If rscale is ommitted, rscale = 1.0 is assumed. \(\phi\)-shift and \(z\)-shift should be given in radian. These numbers are found in .rdl files.

4.26 **phaseshift hlxfit4.com**

Shift the phases of the big G file. The script is written with Ruby, but has backward compatibility with hlxfit4.com. Please set REPEAT before you use this command Usage:

```
phaseshift_hlxfit4.com original_avg_file phishift zshift ipole <rscale> new_avg_file
```

If rscale is ommitted, rscale = 1.0 is assumed. \(\phi\)-shift is in **degree** and \(z\)-shift should be given in **angstrom**. These numbers are found in .rdl files after executing hlxfit4.com
4.27. PLOT8NM-LG

This program is used to distinguish between seam side and seem-free size.

plot8nm-lg <title>

The program requires three files. 1. title.rb, 2. title.nea and 3. title.far. Using title.rb file, it identifies 8nm layer lines in title.nea and title.far files. Those layer lines are converted to little g and plotted.

If the big G file is from seam-free side, the amplitude looks like in the plot shown in Fig. 4.3. Note that the plot has a sharp peak corresponding to the radius of protein. On the other hand, the plot of seam side, there are two peaks and also the amplitude in the inner radius region has high amplitude.

By using these criteria, usually you can tell which is the seam side and which is the seam-free side.

![Plot of Seam-Free and Seam Side](image)

**Figure 4.3: A result of plot8nm-lg**

4.28. plotHelicalLattice

This program is designed for better understanding of helical lattice. Requires xplot.

**usage:** plotHelicalLattice n10 n01 l10 l01

See a result in Figure 4.4.

4.29. plot-radial-density-profile

**usage:** plot-radial-density-profile d0000.avg

In general, the resolution of the big G file is lower than 20 Å. If the resolution is higher, it is difficult to interpret the results.

4.30. refine_omega_xshift

Using a program srch, compare near and far side layer lines and optimize $x$-shift and $\omega$-tilt (out-of-plane tilt).
External programs required for this script

- fftrans: part of MRC suite.
- hlxs: part of Nigel Unwin's program.
- hlxs2srch2: You should find it in app directory.
- makehlxs: A ruby script that convert a .rb file to a .hlxs file. You should find it in app directory.
- srch0: “srch-Old”. This is a fortran program provided by Nigel Unwin (MRC Cambridge, UK) [17]. We use ver 1.2. Please compile the srch command and rename it to srch0.

Required parameter files

- .rb file
- .img file
- “Peak file” (sometimes we call it “PQX” file. In fact, “HKX” would be better.)

Output files

- .hlxs
- .src
- .hlxs.log

First, please edit .rb file and set appropriate N10, N01, L10, L01. Run the script without the “Peak file”. Open .hlxs.log file.

According to the .hlxs.log file, please create a peak file. If your helical object is uniform (helical lattice is the same), you can use it for other samples. An example of peak file is shown below:

```
1 0 32
1 0 33
1 0 38
1 0 40
1 0 41
1 0 44
1 0 45
1 0 49
1 0 50
1 0 54
1 0 55
1 0 56
1 0 60
1 0 61
-1 1 24
```
The first and second columns are $h$ and $k$, and the third column is the $x$. From hlxs.log, you pick up several peaks and specify the peaks using these three values. It is always a good idea to include peaks near the equator for refining $x$-shift and peaks at higher $l$ for refining $\omega$-tilt.

Here is an example of hlxs.log.

In this case, the layer line's $(h,k)$ is $(1,0)$. Then look for peaks. In this case, there is a peak from $x = 20$ to 29. So, put numbers in the peak file like:

```
1 0 20
1 0 21
1 0 22
1 0 23
1 0 24
1 0 25
1 0 26
1 0 27
1 0 28
1 0 29
```

4.31 ub2unbent

See section 3.9.

4.32 tiffseries2ps

This command is designed to print out 2k x 2k CCD image. Please copy the command to your local directory, because you need to change TMP variable in the program, which you specify the working directory.
CHAPTER 4. SCRIPT REFERENCES.  

4.33 tube_tracer

This script automatically traces a tube starting from a point specified in Unbent application.

First open an MRC image file and put one control point at the center of the tube. Also please turn on the switch “sort by y” if the tube runs along y-axis.

![Figure 4.5: Unbend, after pointing center of a tube](image)

Save .ub file.

Run the following command:

```
./tube_tracer file.ub
```

You will be shown two plots. One is the x-y plot and the second one is θ-y plot. The plot should be smooth. If this is OK, open “temp.ub” file by Unbend to see all the control points are placed near the center of the tube.

Also, it is useful to examine “average.mrc”. This file should look like this:

For the second round, you can use this average image as a template. In this case, please rename average.mrc to template.mrc because average.mrc is overwritten.

If you already have a template microtubule image, you can use it as well.

Execute following command.

```
./tube_tracer file.ub template.mrc
```

Then open “temp.ub” file using Unbend.

4.33.1 parameters for tube_tracer

Current parameter is optimized for microtubules taken at x40,000 and scanned using 5 µm / pixel. You may need to play with the parameters in the header of tube_tracer.

To change the parameters, you need to understand the algorithm behind tube_tracer.

At the first point, which is specified in the .ub file, the program tries to find the orientation and the center of the tube.

To find the orientation of the tube, the program follows these steps:

1. Crop the image with a circle whose radius is WindowSize.
2. Project the image into different orientations.

3. The projection is windowed using Radius.

4. Score the projection according to:

   (a) If template images are not available, the score is based on the maximum absolute value in the projection.

   (b) If a template image is specified, the score is correlation value between the projection and the template projection.

Once, the orientation of the tube becomes available, the program tries to find the center of tube at the neighboring segment. See the following parameters for detail.
Figure 4.7: Parameters used in Tube_tracer
Chapter 5

Method References.

In this chapter, methods for each class are explained. Convention in this section is as follows.

If a method start from capital letter, such as NArray.byte, this means that it is a constructive method. If a method start from lower letter, such as narray.dim or self.dim, these are class methods, so please substitute the first word to your own instance object.

5.1 NArray class methods.

5.1.1 NArray initialization methods.

NArray uses following data types (Table 5.1.1). All above method initialize with 0 or nil. Note that argument(s) can not be array. For example, you can not use NArray.sfloat([1,2,3]) to make an array whose size is [1,2,3].

<table>
<thead>
<tr>
<th>method</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NArray.new(typecode, size, ...)</td>
<td>create new NArray. initialize with 0</td>
</tr>
<tr>
<td>NArray.byte(size,...)</td>
<td>1 byte unsigned integer</td>
</tr>
<tr>
<td>NArray.sint(size,...)</td>
<td>2 byte signed integer</td>
</tr>
<tr>
<td>NArray.int(size,...)</td>
<td>4 byte signed integer</td>
</tr>
<tr>
<td>NArray.sfloat(size,...)</td>
<td>single precision float</td>
</tr>
<tr>
<td>NArray.float(size,...)</td>
<td>double precision float</td>
</tr>
<tr>
<td>NArray.scomplex(size,...)</td>
<td>single precision complex</td>
</tr>
<tr>
<td>NArray.complex(size,...)</td>
<td>double precision complex</td>
</tr>
<tr>
<td>NArray.object(size,...)</td>
<td>Ruby object</td>
</tr>
</tbody>
</table>

Table 5.1: NArray data types

5.1.2 Conversion methods.

NArray.to_na(array) convert to NArray

NArray.to_na(string,type[,size,...])

NArray[...]

NArray[1,5,10.0] \#=> NArray.float(3):[1.0, 5.0, 10.0]
NArray[1..10] \#=> NArray.int(10):[1,2,3,4,5,6,7,8,9,10]
5.1. NArray CLASS METHODS.

5.1.3 NArray information
The dimension and index sizes of the instance array can be retrieved by the following methods

- `self.dim`: Return the dimension = the number of indices.
- `self.rank`: same as dim
- `self.shape`: Returns the array of sizes of each index.
- `self.total`: Returns the number of total elements.

5.1.4 Slicing the array
The power of NArray is in its flexible slicing. You may play with various ways of indeces to extract various arrays from the original array.

Components of the index can be one of the following: Integer, Range, Array, true. The index order is the FORTRAN type.

- `a[ 1, 2, -1 ]`: element slicing. If negative, counts backward from the end. Element-dimensions are contracted.
- `a[ 0..3, 4..1 ]`: extract in the range. If the former of the range is bigger, return elements in reversed order.
- `a[ [1,3,2,4] ]`: an array with the elements of the indices. If ‘a’ has multi-dimension and single index is specified in [], ‘a’ is treated as a single dimension array.
- `a[ 1, 2..3, [1,3,2,4], true ]`: Compound index. This returns three-dimensional array.
- `a[]`: Same as `a.dup`. Dupulicate the array.
- `a[ 0, true ]`: Sams as `a[0,0..-1]`. ‘true’ means all.
- `a[ false, 0 ]`: Same as `a[true,true,0]`, if `a` is a 3-d array, ‘false’ means ellipsis dimension.
- `a[ mask ]`: masking. “mask” is a byte NArray with its length equal to that of “a”. According to the value of each element of mask, the corresponding element in “a” is eliminated (when 0) or retained (when not 0).

Example:

```ruby
a=NArray.float(2,2).indgen!
p a[ a.lt 3 ]
--> [ 0.0, 1.0, 2.0 ]
```

(Here, `a.lt 3` gives a byte NArray) (This is also done by `a[ (a.lt 3).where ]`)

A two-or-more-dimensional array object with only one argument in ‘[]’, is treated as a flat one-dimensional array. e.g.: `a[3]` is same as `a[0,1]` if `a` is 3x3 array.

- `self.slice(...)`: Same as `self[...]` but keeps the rank of original array by not elimiting dimensions whose length became equal to 1 (which `self[]` does). This is not the case with the one-dimensional indexing and masking (same as []).
CHAPTER 5. METHOD REFERENCES.

5.1. NARRAY CLASS METHODS.

5.1.5 Replacing Elements

Same rule as slicing Filling values

- \( a[1, 2, 3] = 1 \)
- \( a[0..3, 1..4, 2..5] = 2 \)
- \( a[1,3,2,4], true \) = 3
- \( a[] = 4 \) Same as a.fill!(4)

Special cases:
- \( a[0..2] = b[1..5] \)
  - \( \rightarrow \) Error! due to different num of elements.
- \( a[1,2] = b[0..2,1..3] \)
  - Storing elements from index \([1,2]\) ( \( a[1,2] = b[0,1], a[2,2] = b[1,1], \ldots \)). In other words, \([1,2]\) corresponds to the corner of \( b \) array.
- \( a[0..2,0..3] = b[0..2,1] \)
  - Storing repetitively ( \( a[0,0] = b[0,1], \ldots, a[0,3] = b[0,1] \))

5.1.6 Filling values

- self.indgen!(\([\text{start}, \text{step}]\)) Generate index; Set values from 'start' with 'step' increment
- self.fill!(\(\text{value}\)) Fill elements with 'value'
- self.random!(\(\text{max}\)) Set random values between 0\(\leq \text{x}\leq\text{max}\)
- self.randomn Set Normally distributed random values with mean=0, dispersion=1 (Box-Muller)

5.1.7 Arithmetic operators

Arithmetic operations are performed element by element in NArray. Some operators are introduced by Kikkawa, such as \(+=, -=, *=, /=\). A new useful operator is \text{ternary}. By using this methods, you do not need to use \text{if} operation. For example, the methods can be used as follows:

\( a = \text{NArray.sfloat(5,5).randomn} \)
\( b = \text{NArray.sfloat(5,5).randomn} \)
\( c = (a > b).\text{ternary}(a,b) \)

This program returns the bigger value either in \( a \) or \( b \). For other operators, see Table 5.2.

5.1.8 Bitwise operators

These operators works only for integers. See Table 5.3

5.1.9 Comparisons

When the following comparisons are used, it is interpreted as element-wise comparison and results in BYTE-type NArray. Note that the returned value is not boolean (true or false) as usual comparisons of Ruby. See Table 5.4.
5.1. NARRAY CLASS METHODS.

### Method References

<table>
<thead>
<tr>
<th>Operator</th>
<th>Destructor Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>-self</td>
<td>self += other, self.add! other</td>
</tr>
<tr>
<td>self + other</td>
<td>self -= other, self.sbt! other</td>
</tr>
<tr>
<td>self - other</td>
<td>self *= other, self.mul! other</td>
</tr>
<tr>
<td>self / other</td>
<td>self /= other, self.div! other</td>
</tr>
<tr>
<td>self % other</td>
<td>self.mod! other</td>
</tr>
<tr>
<td>self ** other</td>
<td>Note: self is upcasted according to other. For example NArray.sfloat ** 0.5 is upcasted to float.</td>
</tr>
<tr>
<td>self.abs</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: NArray arithmetic operators

<table>
<thead>
<tr>
<th>~self</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>self &amp; other</td>
<td>OR</td>
</tr>
<tr>
<td>self</td>
<td>other</td>
</tr>
</tbody>
</table>

Table 5.3: NArray bitwise operators

5.1.10 Statistics

The argument of the statistics methods are the dimensions. If the dimensions are omitted, all dimensions are used. The argument can be range. See Table 5.5.

5.1.11 Manipulating Array

See Table 5.6.

Referencing can be used to change NMatrix to NArray.

- self.transpose( dim0, dim1, .. )
  - Transpose array. The dim0-th dimension goes to the 0-th dimension of new array. Negative number counts backward. transpose(-1,1..-2,0) is replacement between the first and the last.
  - self.mirror( dim )
    - Mirror array along the dimension specified by dim (x:0, y:1, z:2). Works when self’s dimension is either 1, 2, or 3.

5.1.12 Type conversion

See Table 5.7.

5.1.13 Iteration

Iteration works similar to that of ruby array. However, when you use NArray, these iteration methods are slow. So, please avoid using these iterations. See Table 5.8.

5.1.14 Complex compound number

See Table 5.9.
### CHAPTER 5. METHOD REFERENCES

#### 5.1. NARRAY CLASS METHODS

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.eq other</td>
<td>equal</td>
</tr>
<tr>
<td>self.ne other</td>
<td>not equal</td>
</tr>
<tr>
<td>self.gt other, self &gt; other</td>
<td>greater than</td>
</tr>
<tr>
<td>self.ge other, self &gt;= other</td>
<td>greater or equal</td>
</tr>
<tr>
<td>self.lt other, self &lt; other</td>
<td>less than</td>
</tr>
<tr>
<td>self.le other, self &lt;= other</td>
<td>less or equal</td>
</tr>
<tr>
<td>self.and other</td>
<td>element-wise condition.</td>
</tr>
<tr>
<td>self.or other</td>
<td>element-wise OR</td>
</tr>
<tr>
<td>self.xor other</td>
<td>element-wise XOR</td>
</tr>
<tr>
<td>self.not other</td>
<td>element-wise NOT</td>
</tr>
<tr>
<td>self.all?</td>
<td>true if all the elements are true.</td>
</tr>
<tr>
<td>self.any?</td>
<td>true if any element is true.</td>
</tr>
<tr>
<td>self.none?</td>
<td>true if none of the element is true.</td>
</tr>
<tr>
<td>self.where</td>
<td>Return NArray of indices where elements are true.</td>
</tr>
<tr>
<td>self.where2</td>
<td>Return Array including two NArrays of indices, where elements are true and false, respectively.</td>
</tr>
<tr>
<td>self.max_pos</td>
<td>returns maximum position as an array.</td>
</tr>
<tr>
<td>self.max_pos_cubic</td>
<td>Similar to max_pos, but use cubic interpolation to guess the position of maximum.</td>
</tr>
</tbody>
</table>

Table 5.4: Comparison operators.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.sum(dim,..)</td>
<td>Summation of elements in the specified dimensions.</td>
</tr>
<tr>
<td>self.mul_add(other,dim)</td>
<td>multiple and sum. faster (x*y).sum(dim)</td>
</tr>
<tr>
<td>self.mean(dim,..)</td>
<td>Mean of elements in specified dimensions.</td>
</tr>
<tr>
<td>self.stddev(dim,..)</td>
<td>Standard deviation of elements in the dimensions.</td>
</tr>
<tr>
<td>self.rms(dim,..)</td>
<td>Root mean square</td>
</tr>
<tr>
<td>self.rmsdev(dim,..)</td>
<td>Root mean square deviation</td>
</tr>
<tr>
<td>self.min(dim,..)</td>
<td>Minimum in the specified dimensions.</td>
</tr>
<tr>
<td>self.max(dim,..)</td>
<td>Maximum in the specified dimensions.</td>
</tr>
<tr>
<td>self.median(dim)</td>
<td>Median in 0..dim</td>
</tr>
</tbody>
</table>

Table 5.5: Statistic methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.sort(dim)</td>
<td>Sort in the 0..dim (All dimensions if omitted)</td>
</tr>
<tr>
<td>self.sort!(dim)</td>
<td>Sort, destructive</td>
</tr>
<tr>
<td>self.sort_index(dim)</td>
<td>Return index of Sort result.</td>
</tr>
<tr>
<td>self.refer</td>
<td>create NArray obj referring to another NArray</td>
</tr>
<tr>
<td>self.reshape(size,...)</td>
<td>same as self.refer.reshape!</td>
</tr>
<tr>
<td>self.newdim(dim,...)</td>
<td>same as self.refer.newdim!</td>
</tr>
<tr>
<td>self.mirror(dim)</td>
<td>return mirrored array.</td>
</tr>
</tbody>
</table>

Table 5.6: Array manipulation methods.

---

71
5.1. NARRAY CLASS METHODS.  

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.floor</td>
<td>Return integer NArray whose elements processed 'floor'</td>
</tr>
<tr>
<td>self.ceil</td>
<td>Return integer NArray whose elements processed 'ceil'</td>
</tr>
<tr>
<td>self.round</td>
<td>Return integer NArray whose elements processed 'round'</td>
</tr>
<tr>
<td>self.to_f</td>
<td>Convert NArray type to float</td>
</tr>
<tr>
<td>self.to_sf</td>
<td>Convert NArray type to single float</td>
</tr>
<tr>
<td>self.to_i</td>
<td>Convert NArray type to integer</td>
</tr>
<tr>
<td>self.to_a</td>
<td>Convert NArray type to Ruby-object</td>
</tr>
<tr>
<td>self.to_s</td>
<td>Convert NArray data to String as a binary data.</td>
</tr>
<tr>
<td>self.to_string</td>
<td>Convert NArray type to Ruby-object containing Strings as printed elements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.swap_byte</td>
<td>swap byte order</td>
</tr>
<tr>
<td>self.hton</td>
<td>convert to network byte order</td>
</tr>
<tr>
<td>self.ntoh</td>
<td>convert to VAX byte order</td>
</tr>
<tr>
<td>self.htov</td>
<td>convert from VAX</td>
</tr>
<tr>
<td>self.vtoh</td>
<td>convert from VAX</td>
</tr>
</tbody>
</table>

**Table 5.7: Type conversion methods.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.each</td>
<td>...</td>
</tr>
<tr>
<td>self.collect</td>
<td>...</td>
</tr>
<tr>
<td>self.collect!</td>
<td>...</td>
</tr>
</tbody>
</table>

**Table 5.8: Iteration methods.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>self.real</td>
<td></td>
</tr>
<tr>
<td>self.imag</td>
<td>set imaginary part</td>
</tr>
<tr>
<td>self.conj</td>
<td>set imaginary part</td>
</tr>
<tr>
<td>self.angle</td>
<td>set imaginary part</td>
</tr>
<tr>
<td>self.imag= other</td>
<td>multiply by imaginary unit</td>
</tr>
<tr>
<td>self.im</td>
<td>set imaginary part</td>
</tr>
</tbody>
</table>

**Table 5.9: Complex compound numbers.**
### 5.2 NMATRIX

NMATRIX is a subclass of NArray. First two dimensions are used as Matrix. Residual dimensions are treated as Multi-dimensional array. The order of Matrix dimensions is opposite from the notation of mathematics: \( a_{ij} \Rightarrow a_{[j,i]} \).

\(+,-\) enable if other is an NMATRIX.

\(*\) Matrix product if other is an NMATRIX or an NVECTOR. Scalar product if other is Numeric or NArray. For example,

\[
\text{NMATRIX}[[1,2],[3,4]] * [1,10] = \text{NMATRIX}[[1,20],[30,40]]
\]

\(/\) other Scalar division if other is Numeric or NArray. Solve Linear Equation with LU factorization, if other is square NMATRIX. \( a/b = b.lu.solve(a) \) This function is not 64bit compatible.

**transpose** transpose Matrix dimensions if argument omitted.

**diagonal(val)**

**diagonal!(val)** set val to diagonal elements. (set 1 if omitted).

**unit** set 1 to diagonal elements.

**NMATRIX.unit(dimension)** For a given dimension, returns integer type unit matrix.
5.3. NVECTOR

inverse  Inverse matrix.

lu  compute LU factorization. return NMatrixLU class object (see below).

5.2.1 NMatrix, standard matrix generators

- NMatrix.affine_shift(x, y, {z}) Returns 2D or 3D Affine transformation matrix. If z is not given, it returns 2D Affine matrix (3x3). If z is given, it returns 3D Affine matrix (4x4).
- NMatrix.2d_affine_rotation(phi)
- NMatrix.2d_affine_scale(scale)

5.3 NVector

NVector is a subclass of NArray. First dimension is used as Vector. Residual dimensions are treated as Multi-dimensional array.

- +,-: enable if other is an NVector.
- *: Matrix product if other is NMatrix. Inner product if other is NVector. Scalar product if other is Numeric or NArray.
- /: Scalar division if other is Numeric or NArray. Solve Linear Equation with LU factorization if other is square NMatrix. \( v/m == m.lu.solve(v) \)

5.4 NMatrixLU

NMatrixLU is created by NMatrix#lu method. Including LU (NMatrix) and pivot (NVector).

Example:

```ruby
a = NMatrix.sfloat(2,2)
a[0,0] = 1.0
a[0,1] = 0.0
a[1,0] = 1.0
a[1,1] = 1.0
b = a.lu
v = NVector.sfloat(2)
v[0] = 2.0
v[1] = 3.0
p(b.solve(v)) = NVector.sfloat(2): [-1.0, 3.0 ]
```

Solve with the result of LU factorization.

5.5 FFTW3 module

This module was originally written by Takeshi Horinouchi (GFD Dennou Club, 2003) and modified by Masahide Kikkawa.

The module support multi-dimensional complex FFT. It supports both double and single float transforms, if FFTW3 is compiled for both single and double float. Note that the values are not normalized as in FFTW.

- FFTW3.fftw( narray, dir, [,,dim,...] )
CHAPTER 5. METHOD REFERENCES

5.6 CUFFT MODULE

- FFTW3.fft( narray, dir, [dim, dim, ...] )
  Perform complex FFT. The 3rd, 4th, ... arguments are optional.

  narray (NArray or NArray-compatible Array) : array to be transformed. If real, coerced to complex
  before transformation. If narray is single-precision and the single-precision version of FFTW3 is
  installed (before installing this module), this method does a single-precision transform. Otherwise,
  a double-precision transform is used.

  dir (-1 or 1) : forward transform if -1; backward transform if 1.

  optional 3rd, 4th,... (Integer) : Specifies dimensions to apply FFT. For example, if 0, the first di-
  mension is transformed (1D FFT); If -1, the last dimension is used (1D FFT); If 0,2,4, the first,
  third, and fifth dimensions are transformed (3D FFT); If entirely omitted, ALL DIMENSIONS ARE
  SUBJECT TO FFT, so 3D FFT is done with a 3D array.

Return value: a complex NArray

Note: As in FFTW, return value is not normalized. Thus, a consecutive forward and backward trans-
form would multiply the size of data used for transform. You can normalize, for example, the forward
transform FFTW.fft(narray, -1, 0, 1) (FFT regarding the first (dim 0) & second (dim 1) dimen-
sions) by dividing with (narray.shape[0]*narray.shape[1]). Likewise, the result of FFTW.fft(narray,
-1) (FFT for all dimensions) can be normalized by narray.length.

- FFTW3.r2c(narray)
  Perform real to complex transform. Works for single float and double. Retern value depends on how
  you compile ruby-helix. If you compiled with single float-able FFTW3, it will return the same type. If
  not, always return double.

- FFTW3.c2r(narray)
  Perform complex to real transform.

- FFTW3.export_wisdom_to_file(filename)

- FFTW3.import_wisdom_from_file(filename)

Those two modules can save and restore FFTW's wisdom. Following is a quote from FFTW3's man-
ual:

FTTW implements a method for saving plans to disk and restoring them. In fact, what FFTW
does is more general than just saving and loading plans. The mechanism is called wisdom.
Here, we describe this feature at a high level. See FFTW Reference, for a less casual but
more complete discussion of how to use wisdom in FFTW.

5.6 CUFFT module

This module was developed to test CUFFT performance.

- CUFFT.fft(narray, direction)

  narray (NArray or NArray-compatible Array) : array to be transformed. If real, coerced to complex
  before transformation. If narray is single-precision and the single-precision version of FFTW3 is
  installed (before installing this module), this method does a single-precision transform. Other-
  wise, a double-precision transform is used.

  dir (-1 or 1) : forward transform if -1; backward transform if 1.

75
5.7. EM RELATED NARRAY METHODS.  
CHAPTER 5. METHOD REFERENCES. 

5.7 EM related NArray methods.

5.7.1 Input and output

Currently Ruby-Helix can read and write MRC format and tiff format.

- \texttt{NArray.open\_mrc(filename)}, \texttt{NArray.read\_mrc(filename)}: Read MRC file from the file specified by filename, e.g. if you have a file \texttt{test.mrc}, you can open it with \texttt{NArray.open\_mrc("test.mrc")}.

\textbf{Byte-order:} If the script tries to read a MRC file generated by a computer with different byte-order (e.g. Intel vs PowerPC), the script automatically swap the byte-order, based on a assumption that any dimensions of the array should not be larger than 300,000.

This method can read both 20th and 21st century format of MRC.

- \texttt{self.write\_mrc(filename)}: write various type of MRC byte, short-int, single float and double, which are written in the native format of the machine you use. The MRC format is 21st century new format.

- \texttt{self.write\_xplor(filename)}: write X-PLOR test type map to the file specified by the filename. Tested only for pymol and float type.

- \texttt{NArray.open\_uniform(filename)} \texttt{NArray.read\_uniform(filename)} This function is mainly used for read “uniform” ascii format. This format is mainly used for writing and reading spline curve. This format is described below:

\begin{verbatim}
dimensions
dim1 dim2 dim3
data...
\end{verbatim}

- \texttt{NArray.write\_uniform(filename)} This function is mainly used for write “uniform” ascii format. This format is mainly used for spline curve.

- \texttt{NArray.open\_tiff(filename)}:

- \texttt{NArray.read\_tiff(filename)}: open tiff format

- \texttt{NArray.write\_tiff(filename)}

- \texttt{NArray.byte(x,y).write\_tiff(filename)} or \texttt{NArray.byte(3,x,y).write\_tiff(filename)} In this case color.

- \texttt{NArray.xplot(title)} Title is optional. Plot the content of the NArray as one dimensional array.
If NArray contains \((N,2)\) or \((2,N)\) array, this methods will plot the pair as \((x,y)\).

- \texttt{NArray.write\_tiff2(filename)} NArray any types. Written in Ruby script, but fast. Automatically scale, and write gray scale image.

- \texttt{self.complex\_to\_color\_tiff(filename,max=nil)} Save complex value in colored tiff map. HUE is the phase of complex value and saturation is the amplitude.

- \texttt{narray.to\_ps(out, dot\_per\_inch, \_message)}: Print out the postscript image to \texttt{out} (an IO object). The image will be printed out with \texttt{dot\_per\_inch}. A message can also be printed out, optional.
5.7.2 Image manipulations.

- `narray.apodize(width)`: Gaussian apodization. \( \sigma = (width/3) \).
- `narray.background_subtract_along_x(width)`: Background subtraction along x. The background is fitted with parabollic function. Left and right \( width \) pixels are used as backgrounds.
- `narray.background_subtract_along_y(width)`: Background subtraction along y. The background is fitted with parabollic function. Left and right \( width \) pixels are used as backgrounds.
- `narray.clamp(min, max), narray.clamp!(min, max)`: Limit the minimum and maximum value of the array.
- `narray.correlation(ref)` Returns the correlation between self and ref. Self and ref need to be real array (sfloat or float). The two array also need to have the same dimension. Requires FFTW3 module. An example is shown in Figure 5.1.

![Image 1 Image 2 Image1.correlation(Image2)](image)

**Figure 5.1: narray.correlation(ref)**

- `narray.cut_out(start, end_p, length, width, spline, shift = 0.0)` Cut out along spline curve. The image will be extracted using bilinear interpolation.
  - Start point, end point in the “spline coordinate” should be specified.
  - The length and the width of the final image should also be specified. If (end-start) is different from the length, the image will be shrinked or stretched.
  - spline is an array of size (4,n), where the spline[0,i] is the index, spline[1,i] is \( y \) value, spline[2,i] is \( x \) value, and spline[3,i] is the angle of the spline curve.
  - “shift” is an optional float value.

Similar function `narray.unbend` was written, but not tested extensively. Therefore removed from the library. Please use `cut_out`.

- `narray.fit_sub(b)` (stands for fit and subtract) NArray object \( b \) is fitted to self by changing value \( c \) to minimize \( (self - b \cdot c)^2 \). Then return \( self - b \cdot c \).
- `narray.interpolate3d(x, y, z, background)`
- `narray.interpolate2d(x, y, background)`

Currently `interpolate2d` uses `na_interpolate_2d_c`, internally. This is because 2D version is twice as fast as n-dimensional version.
5.7. EM RELATED NARRAY METHODS.

- narray.interpolate1d(x, background)

These two methods extract data from narray image using coordinate specified by \( x, y, z \). Those \( x, y, z \) are float array of the same size. Dimensions of \( x, y, z \) need to be the same as the dimension of original image. If \( x, y, z \) are sfloat type, C-written version of interpolate is used, which is much faster and memory efficient.

background is optional. If not specified, background is set to 0.0.

Among these three methods, interpolate1d can accept scalar value, even if narray is not single float.

- narray.interpolate(xyz, background)

Interface to interpolate nd.c. This method include checking of the type and can be used also for scalar values.

Here is an example:

```plaintext
N = 10
a = NArray.sfloat(N).randomn!
x = NArray.sfloat(N).indgen!
b = a.interpolate([x-1], 0.0)
p b
b = a.interpolate([1])
p b
# 2D
a = NArray.sfloat(N,N).randomn!
x = NArray.sfloat(N).indgen!
y = NArray.sfloat(1,N).indgen!
xx = x + 0.0 * y
yy = x * 0.0 + y
b = a.interpolate([xx-1, yy], 0.0)
p b
b = a.interpolate([1,1])
p b
# 3D
a = NArray.sfloat(N,N,N).randomn!
x = NArray.sfloat(N).indgen!
y = NArray.sfloat(1,N).indgen!
z = NArray.sfloat(1,1,N).indgen!
xx = x + 0.0 * y + 0.0 * z
yy = 0.0 * x + y + 0.0 * z
zz = 0.0 * x + 0.0 * y + z
```

78


**CHAPTER 5. METHOD REFERENCES.**

**5.7. EM RELATED NARRAY METHODS.**

\[
b = a.\text{interpolate}([xx-1,yy-1,zz-1],0.0)
\]

- `narray.interpolate_nd_c(xyz,background)`
  
  C implementation of multidimensional bi-linear interpolation routine. This function when `narray` is an SFLOAT type array. `xyz` is an array of coordinate. For example, if `x` and `y` contain the coordinate, you may call this function as:

  \[
  \text{narray.interpolate_nd_c}([x,y],\text{background})
  \]

- `narray.fourier_synthesis_2d!(bigG,startx,endx,starty,endy,omega=0.0)`: Fourier synthesis in 2D. Here is a sample code

```ruby
require "LayerLine.rb"

g = BigG.read("AMPPNPnewd.avg")
g.repeat = 1520.0  # Set repeat length in angstrom
g.limits.resolution!(25.0)
a = NArray.sfloat(256,256)
a.dimensions = [2.5,2.5]
a.origin = [128,0]  # Origin is given by pixel, not angstrom

a.fourier_synthesis_2d!(g,50.206,0,128)
a.write_mrc("test.mrc")
```

- `narray.fourier_synthesis_3d!(littleG,centerx,centery,centerz,pixel_size)`: Yet to be described...

- `narray.mrc_fft`: Apply MRC type fft and returns the array. It executes an external command `fft` provided by MRC package.

- `narray.mrc_fftw`: Apply MRC type Fast Fourier transform using FFTW3. Approximately four times faster than `mrc_fft` because it does not involve writing and reading temporary file. The first dimension should be even.

- `narray.pad(x,y,padvalue=0,bevel=0)`: Pad the image to the final size of `(x,y)`. The original image is located at the center.

- `narray.put_pdb!(pdb)`: Put a molecule specified by a PDB object onto the image.

- `narray.shrink(x,y)`: Shrinks the image by factors given by `x` and `y`.

- `narray.ya_pad(direction,size,position=Pad_center,pad_value = 0,bevel=0)`: Yet another padding method.

**5.7.3 FFT-image related.**

Following methods work only for MRC type FFT image files (so called FT format).

- `narray.resolution`: Returns the resolution array based on the cell dimension.

- `narray.bfactor`: Works for FFT image. Returns the b-factor based on the cell dimension. What actually returned is \( \exp(br) \), where `b` is the b-factor and `r` is the resolution of each point.

79
5.8 LAYERLINE CHAPTER 5. METHOD REFERENCES.

- `narray.ctf(def1,def2,angle,hv=200,amp=0.07,cs=3.3)`: The pixel size of the array should be given in the header of the array. `def1` and `def2` is the defocus level in angstrom. Angle of the stigmatism should be given by angle. `hv` is the acceleration voltage in kV. `amp` is the amplitude contrast. `cs` is the spherical aberration.

- `narray.extract_layerline(height,rmin,rmax,n,l)`: Mainly used from verb `narray.hlxfl—.`

- `narray.hlxfl(controlFilename, [ctf])`: Extract layer lines according to the `controlFilename`. Optionally, `ctf` image is given. The `ctf` should contain the CTF value at each points. Returns three Big G objects, near, far and average layer lines in an array.

- `narray.limit_resolution!(res)`: Based on the header information, limit the resolution to specified by `res` (Å).

- `narray.phaseshift!(x_origin,y_origin)`: 

5.7.4 Header related

- `self.header`: return the header of MRC

- `self.add_label`: add label to the header

- `self.mode`: `self.mode=`

- `self.origin`: `self.origin=[x,y]`

- `self.spacegroup`

- `narray.dimensions`: `narray.dimensions=`

- `narray.cellsize`: `narray.cellsize=`: set the cell size. Currently set the dimension size, but in the future, to conform with MRC format, this will be changed.

- `narray.pixelsize`: `narray.pixelsize=`: set the pixel size in Angstrom. Currently set the dimension size, but in the future, to conform with MRC format, this will be changed.

- `narray.n_label`: returns the number of labels.

- `narray.reset_label!`: Remove all the labels.

- `narray.add_label!(arg)`: Add a new label.

5.8 LayerLine

This class is for each one layer line. When a set of layer lines is treated, following BigG class is used instead.

```
LayerLine.new(fileHandle):
layerline.apply_r_scale:
layerline.apply_phi_z_shift(phi,z): phi: radian z: 2 Pi/ repeat_distance
layerline.write(fileHandle):
```
5.8.1 Accessor methods

- \texttt{layerline.n}:
- \texttt{layerline.l}:
- \texttt{layerline.n}:
- \texttt{layerline.l}:
- \texttt{layerline.r}:
- \texttt{layerline.data}:
- \texttt{layerline.size}:

5.8.2 Manipulation

These methods are usually called from BigG methods.

- \texttt{layerline.is_n_l_same}:
- \texttt{layerline.get_data_at_r(r)}:
- \texttt{layerline.get_ctf_at_r(r)}:
- \texttt{layerline.add(ll_add, weight)}:
- \texttt{layerline.reverse!}: Change the polarity of layer line data.
- \texttt{layerline.fourier_bessel!(pixel_size,dim)}: convert from bigG to little g vise versa. In this case, Bessel order can be float. The function is used for Asymmetric Helical Reconstruction [11].
- \texttt{layerline.fourier_bessel_matrix(pixel_size,dim)}: returns matrix used for Fourier Bessel conversion.

5.9 BigG

Originally designed for deal with big G file, but the class was extended to accomodate little g data.

5.9.1 Input and output

Currently ascii file BigG file can be read.

\begin{verbatim}
BigG.read(filename): read bigG from a file specified by the filename.
BigG.read_little_g(filename): read little g (Toyoshima and Yonekura format)
bigG.write(filename): write a bigG to a file.
bigG.write_ky(filename): write a bigG to a file in Yonekura and Toyoshima bigG format.
\end{verbatim}
5.9. BIGG

5.9.2 Manipulating big G files

There are several methods to manipulate big G files.

- bigG.apply_r_scale(rscal): Apply R scale factor.
- bigG.r_mask(rmin, rmax): Used for little g. Pick up only the region between rmin and rmax (including these values).
- bigG.apply_phi_z_shift(phi, z): Apply φ, z shift φ(φ): radian, z:
- bigG.hk_select(array): Return a bigG that contains layer lines specified by array. A sample of array is like [[0,0],[1,0]].
- bigG.reverse!: Reverse the orientation.
- bigG.add_layerline(LayerLine a): Add a new layer line file.
- bigG.multiply!(scale): The amplitude of the data is multiplied by scale
- bigG.renumbering(n10, l10, n01, l01, new_n10, new_l10, new_n01, new_l01, rmax10, rmax01): bigG.plot(n): plot n-th layer line with gnuplot.
- bigG.limit_resolution(resolution) limit the resolution. Before executing this method, repeat length should be set. give angstrom

5.9.3 Fourier-Bessel conversions.

- bigG.fourier_bessel!(pixelsize, dim): The Fourier Bessel conversion. The conversion works in either directions. (little g - big G, too) pixelsize is given in angstrom, when a big G is converted to a little g. When a little g is converted to a big G, the pixelsize is given in [1/A], dim is an integer value. (maximum radius should be pixelsize * dim ) This method works in both direction !!! (from little g to big G)

The new fourier Bessel transformation can transform float Bessel orders.

- bigG.littleG_to_bigG(littleG): The method convert a little g (littleG) according to the bigG given by bigG.

5.9.4 Accessor methods.

- bigG.resolution=(resolution): same as bigG.limit_resolution. Therefore, the method manipulate the data too.
- bigG.resolution: returns the current resolution
- bigG.limit_radius(rmax): mainly used for little G
- bigG.repeat=:
- bigG.truepitch=: set the truepitch in angstrom.
- bigG.truepitch: Returns the truepitch in angstrom.
In general, all the helical lattice can be described using basic helical lattice [18].

\[
\begin{pmatrix}
  n_{h,k} \\
  l_{h,k}
\end{pmatrix}
= h \begin{pmatrix}
  n_{1,0} \\
  l_{1,0}
\end{pmatrix}
+ k \begin{pmatrix}
  n_{0,1} \\
  l_{0,1}
\end{pmatrix}
\]  

(5.1)

Using the basic helices, now we can describe the unit cell as shown in Fig. 5.2.

Figure 5.2: A cylindrical section of a tubular crystal illustrating “helical” lattice lines and a unit cell. In this figure, \( \nu_{1,0} = 5 \), \( l_{1,0} = 1 \), \( \nu_{0,1} = 1.5 \), and \( l_{0,1} = 4 \). (A) Two unit cell vectors \( a \) and \( b \) define an asymmetric unit (gray rhombus). The vectors satisfy two conditions, \( \nu_{1,0}a - \nu_{0,1}b = (2\pi, 0) \) and \( -l_{1,0}a + l_{0,1}b = (0, c) \). (B) The relation between \((\phi, z)\)-coordinate and the unit cell vector-coordinate.

From this relation, the unit cell vectors have the relation shown with the basic helix vectors as follows. Here we assume \( a = (a_\phi, a_z) \) and \( b = (b_\phi, b_z) \).

\[
\begin{pmatrix}
a_\phi \\
a_z
\end{pmatrix}
\begin{pmatrix}
  n_{1,0} & -l_{1,0} \\
  -n_{0,1} & l_{0,1}
\end{pmatrix}
\begin{pmatrix}
a_\phi \\
a_z
\end{pmatrix}
= \begin{pmatrix}
  2\pi \\
  0
\end{pmatrix},
\]  

(5.2)

Actually this equation is interesting because, the determinant of

\[
\begin{pmatrix}
n_{1,0} & -l_{1,0} \\
n_{0,1} & l_{0,1}
\end{pmatrix}
\]  

(5.3)

is a number of unit cells in the repeat unit. If the value is negative, the size of the unit cell calculated by the following equation is also negative.

In addition, the determinant of

\[
\begin{pmatrix}
a_\phi \\
a_z
\end{pmatrix}
\begin{pmatrix}
a_\phi \\
a_z
\end{pmatrix}
\]  

(5.4)

is the size of the unit cell.

To facilitate these calculation, there are several methods.

- \texttt{bigG.nl_determine}: Guess the basic helix from the layer line data.
- \texttt{bigG.nl_check}: If basic helical vectors are not given, determine them using \texttt{nl_determine}.
5.9. **BIGG**

- bigG.n10:
- bigG.n01:
- bigG.l10:
- bigG.l01:
- bigG.n10=:
- bigG.n01=:
- bigG.l10=:
- bigG.l01=:

The following methods return real space basic vectors.

- bigG.phiz_matrix: returns the $\phi, z$ matrix. If repeat length is not given, assume repeat = $2\pi$. returned matrix is:

$$
\begin{pmatrix}
  a_{\phi} & b_{\phi} \\
  a_z & b_z
\end{pmatrix}
$$

(5.5)

- bigG.za: returns $z_a$
- bigG.zb: returns $z_b$
- bigG.phia: returns $\phi_a$
- bigG.phib: returns $\phi_b$
- bigG.phi_unit_cell: returns the size of unit cell in $\phi$ direction (radian). Calculate the edge-to-edge size. See figure 5.2.
- bigG.z_unit_cell: returns the size of unit cell in $z$ (radian or angstrom, depending on whether you gave repeat length). Calculate the edge-to-edge size. See figure 5.2.
- bigG.hk(n,l): returns [h,k] pair as an array.
- bigG.make_ll(delr,radius): after setting n10,l10,n01,l01,repeat,resolution make zero valued layer line.
  delr ... (1/Å) step size of bigG
  radius ... outer radius of the specimen (Å).

**Example** An example of the usage: Here is an example. In this case, we use a microtubule with 15 protofilaments and 2-start helix.

```
$ irb
irb(main):001:0> a = BigG.new
irb(main):002:0> a.n10= 15
irb(main):003:0> a.l10= 1
irb(main):004:0> a.n01= -2
irb(main):005:0> a.l01= 17
irb(main):006:0> a.repeat = 1360
irb(main):007:0> p a.phiz_matrix
```

84
CHAPTER 5. METHOD REFERENCES.  
5.10. SPLINEARRAY

NMatrix.sfloat(2,2):
NMatrix.sfloat(2,2):
[ [ 0.4156, 0.02445 ],
[ -10.58, 79.38 ] ]

irb(main):008:0> p a.nl_matrix
NMatrix.sfloat(2,2):
[ [ 15.0, -1.0 ],
[ 2.0, 17.0 ] ]

irb(main):009:0> exit

5.9.6 Methods for analyzing the data.

self.correlationPlot(bigG2) printout correlation plot for xplot

5.10 SplineArray

To generate and manipulate spline curve. This class depends on Ruby/GSL. There are two implementations for GSL for Ruby, but this class library depends on Tsunesada’s implementation. Please take a look at installation section.

This class uses x,y swapped axis shown in Figure 5.3.

- SplineArray.new(splinearray,startp,endp): The initialization method. Mainly for internal use.
- SplineArray.from_points(yarray,xarray,startp,endp): Generate spline curve using cubic spline curve. yarray: array of float, xarray: array of float, the dimension of yarray and xarray should be the same. startp, endp: start and end point in y, not in y'.
- SplineArray.read_unbent(filename): Read from “unbent” file, which contains uniform format of control points.
- splineArray.ydash_to_y(ydash):
- splineArray.ydash_to_x(ydash):
- splineArray.y_to_ydash(y):
- splineArray.y_to_x(y):
- splineArray.write(filename):
- splineArray.ydash:
  - y: return NArray that contains y coordinates.
  - x: return NArray that contains x coordinates.
  - theta: return NArray that contains \theta in degree.
  - size:
Figure 5.3: Axis for spline class
CHAPTER 5. METHOD REFERENCES.  

5.11 SerialG

- +: second argument (right argument) can be either SplineArray or Numeric.

If the second argument is an instance of SplineArray, the result is

\[
y_3(y'_3) = y_1(y'_1) - x_2(y_2) \sin \theta_1(y'_1), (y'_3 = y_2 = y'_1)
\]

\[
x_3(y'_3) = x_1(y'_1) + x_2(y_2) \cos \theta_1(y'_1), (y'_3 = y_2 = y'_1)
\]

where, \(x_1, y_1, y'_1, \theta_1\) are from the left SplineArray, \(x_2, y_2, y'_2, \theta_2\) are from the right SplineArray, and \(x_3, y_3, y'_3, \theta_3\) are the results.

- xplot(title = "", xplot = open("|xplot","w") ) Plot the spline curve using xplot.

5.11 SerialG

Usually a Big G stores the data as a set of LayerLine(s). However, sometimes it is difficult to apply some methods as a whole. Therefore we introduced SerialG (this include both big and little g’s). Also for BigG, serialization methods are implemented.

Serialized big G or little G. Mainly designed for easier manipulation of big G files.

To generate SerialG from BigG

\[
g = \text{BigG.read("filename.avg")}
\]

\[
sg = g\text{.serialize}
\]

Or using template:

\[
g = \text{BigG.read("filename.avg")}
\]

\[
sg = g\text{.serialize(template\_sg)}
\]

- \(sg\text{.repeat} = \): set or get repeat length in angstrom.
- \(sg\text{.x\_shift}!(x)\): apply x shift
- \(sg\text{.omega\_shift}!(x)\): apply omega shift
- \(sg\text{.reverse}!\): apply reverse
- \(sg\text{.change\_side}\): change near to far or far to near.
- \(sg\text{.phase\_residual}(sg2)\): return the phase residual value between two serial G.
- \(sg\text{.phase\_shift}!(phi,z)\): phi: radian z: radian — recently changed!!
- \(sg\text{.dup}\): duplicate
- \(sg\text{.bigG}\): convert to BigG data.
- \(sg\text{.resolutionPlot}(sg2,truepitch,xplot = nil)\): plot resolution (x) vs correlation (y) using xplot.
- \(sg\text{.limit\_nl}(n,l)\): limit \(\leq n\) and \(\leq l\) and returns a new sg
- \(\text{bigG.n}\): returns serialized n as an NArray object.
- \(\text{bigG.l}\): returns serialized l as an NArray array
- \(\text{bigG.r}\): returns serialized r array
- \(\text{bigG.data}\): returns serialized data array
- \(\text{bigG.ctf}\): returns serialized ctf array
5.12. FREALIGN

5.11.1 Accessor methods

- sg.r:
- sg.data:
- sg.ctf:
- sg.n:
- sg.l: get individual data

5.11.2 Operators

- +: Sum. Here the method assumes two serialBigG have same (n,l) pairs.
- *: Left hand is the serialBigG and right hand should be a scalar.
- ctf_multiply!: Inside, it does @data = @data * @ctf.
- ctf_square!: Inside, it does @ctf = @ctf * @ctf.

5.12 Frealign

This class is used to read, manipulate, and write FREALIGN [?] parameter files.

5.13 PDB (protein data bank)

This class is to handle PDB files.

- PDB.read(filename): read PDB file
- pdb.write(filename): write PDB file
- pdb.move(x,y=0,z=0): translate by (x,y,z) If x is a NVector object, y and z can be omitted.
- PDB.center: Retern the center of the gravity Currently each atom have same weight return value is a vector(3)
- PDB.rotate(matrix): take 3x3 NMatrix to rotate. Actually this is simply a multiply of the matrix to the coordinate.
- PDB.rotate_z(phi): rotate around z with phi
- PDB.dup: duplicate the content. only the coordinate (xyz), bfactor and occupancy are copied.
- NArray.put_pdb(pdb): project PDB subunit along the Z axis
5.14 Minimizer2D, a two-dimensional minimizer

- require "Minimizer.rb"
- obj = Minimizer2D.new: generate a new Minimize.
- obj.set_x(xmin,xmax,steps): set x range.
- obj.set_y(ymin,ymax,steps): set x range.
- obj.minimize{ |array| value }: given parameter array = x, y and return value so that the value is minimized. return the minimized values as [xmin, ymin, value].
- obj.update_range: update the search range. Search range r is updated to be 2r/n, where n is the number of steps.

Example code:

```ruby
require "Minimizer2D"

a = Minimizer2D.new
a.set_x(0,10,10)
a.set_y(0,10,10)

random_x = rand() * 5
random_y = rand() * 5

result = a.minimize{ |param|
  x = param[0]
  y = param[1]
  (x - random_x) * (x - random_x) + (y - random_y) * (y - random_y)
}
p random_x
p random_y
```

5.15 Minimizer3D, a three-dimensional minimizer

This class is implemented as a sub-class of Minimizer2D.

- obj.minimize3D{ |array| evaluate value }: given parameter array = x, y, z and return value so that the value is minimized. return the minimized values as [xmin, ymin, zmin, value]. obj.update_range:

Sample code:

```ruby
require "Minimizer2D"

a = Minimizer3D.new
a.set_x(0,10,10)
a.set_y(0,10,10)
a.set_z(0,10,10)

random_x = rand() * 5
random_y = rand() * 5
```
5.16. **Histogram**

```plaintext
random_y = rand() * 5
random_z = rand() * 5

result = a.minimize3D { |param|
    x = param[0]
    y = param[1]
    z = param[2]
    (x - random_x) * (x - random_x) +
    (y - random_y) * (y - random_y) +
    (z - random_z) * (z - random_z)
}

p random_x
p random_y
p random_z
p result
```
Bibliography


Index

3D-fourier-syntheiss.rb, 49
auto_unbent, 49

basic helices, 81
bigG.add_layerline, 80
bigG.apply_phi_z_shift, 80
bigG.apply_r_scale, 80
bigG.fourier_bessel, 80
bigG.hk_select, 80
bigG.limit_radius, 80
bigG.littleG_to_bigG, 80
bigG.multiply, 80
bigG.n, 85
bigG.nl_check, 81
bigG.nl_determine, 81
bigG.nl_i01, 82
bigG.nl_i01=, 82
bigG.nl_i10, 82
bigG.nl_i10=, 82
bigG.nl_n01, 82
bigG.nl_n01=, 82
bigG.nl_n10, 82
bigG.nl_n10=, 82
bigG.plot, 80
bigG.r_mask, 80
bigG.renumbering, 80
bigG.repeat=, 80
bigG.resolution, 80
bigG.resolution=, 80
bigG.reverse, 80
bigG.truepitch=, 80
bigG.write, 79
Bsoft, 14

ccd2ctf, 51
check_installation, 50
correlation_matrix, 51
correlationTube, 50
CUFFT, 73
CUFFT.fft, 73
cut_out, 51

fft, 52

FFTW3, 72
FFTW3.c2r, 73
FFTW3.export_wisdom_to_file, 73
FFTW3.fft, 73
FFTW3.fftw, 73
FFTW3.import_wisdom_from_file, 73
FFTW3.r2c, 73
filter_radius, 53

helix_boxer, 53
hlxflctf, 32
hlximage, 54

limit_resolution, 56
line_filter_2D, 56
llplot, 56

minimizer2d.minimize, 87
Minimizer2d.new, 87
minimizer2d.set_x, 87
minimizer2d.set_y, 87
minimizer2d.update_range, 87
minimizer3d.minimize3D, 87
mrc2situs, 56
mrc2xplor, 56

narra.phaseshift, 78
narray.add, 68
narray.add_label, 78
narray.apodize, 75
narray.background_subtract_along_x, 75
narray.background_subtract_along_y, 75
NArray.byte, 65
narray.ceil, 70
narray.cellsize, 78
NArray.complex, 65
narray.correlation(ref), 75
narray.ctf, 78
narray.dim, 66
narray.dimensions, 78
narray.div, 68
narray.fill, 67
narray.fit_sub(), 75
NArray.float, 65
INDEX

narray.floor, 70
narray.fourier_synthesis_2d, 77
narray.fourier_synthesis_3d, 77
narray.header, 78
narray.hlxfl, 78
narray.hton, 70
narray.htov, 70
narray.indgen, 67
NArray.int, 65
narray.interpolate(xyz,background), 76
narray.interpolate1d(x,background), 76
narray.interpolate2d(x,y,background), 75
narray.interpolate3d(x,y,z,background), 75
narray.interpolate
nd(xyz,background), 77
narray.mod, 68
narray.mode, 78
narray.mrc_fft, 77
narray.mrc_fftw, 77
narray.mul, 68
narray.n_label, 78
narray.ne, 69
narray.ntoh, 70
NArray.object, 65
NArray.open_mrc, 74
NArray.open_tiff, 74
NArray.open_uniform, 74
narray.origin, 78
narray.pixelsize, 78
NArray.put_pdb, 86
narray.put_pdb, 77
narray.randomn, 67
narray.rank, 66
NArray.read_mrc, 74
NArray.read_tiff, 74
NArray.read_uniform, 74
narray.reset_label, 78
narray.resolution, 77
narray.round, 70
narray.sbt, 68
NArray.scomplex, 65
NArray.sfloat, 65
narray.shape, 66
narray.shrink, 77
NArray.sint, 65
narray.spacegroup, 78
narray.sum, 69
narray.swap_byte, 70
narray.ternary, 67
narray.to_a, 70
narray.to_f, 70
narray.to_i, 70
NArray.to_na, 65
narray.to_ps, 74
narray.to_s, 70
narray.to_sf, 70
narray.to_string, 70
narray.total, 66
narray.unbend, 75
narray.vtoh, 70
narray.write_mrc, 74
NArray.write_tiff, 74
NArray.write_uniform, 74
narray.write_xplor, 74
narray.xplot(title), 74
narray.ya_pad, 77
NMArray.atan2, 71
NMArray.cos, 71
NMArray.exp, 71
NMArray.j0, 71
NMArray.j1, 71
NMArray.jn, 71
NMArray.jnu, 71
NMArray.log, 71
NMArray.log10, 71
NMArray.log2, 71
NMArray.sin, 71
NMArray.sqrt, 71
NMArray.tan, 71
NMArray.y0, 71
NMArray.y1, 71
NMArray.yn, 71
NMArray.ynu, 71
NMArray.matrix, 71
NMArray.matrix_slice, 71
NMArray.inverse, 72
NMArray.matrix, 72
NMArray.matrix, 71
NMArray.matrix, 71
PDB.center, 86
PDB.dup, 86
PDB.move(x,y,z), 86
PDB.read(filename), 86
PDB.rotate_z(matrix), 86
PDB.rotate_z(45), 86
PDB.write(filename), 86
phaseshift, 57
phaseshift_hlxfit4.com, 57
plot-radial-density-profile, 58
plot8nm-lg, 58
plotHelicalLattice, 58
refine_omega_xshift, 32, 58
scanOmega, 31
sg.bigG, 85
sg.change_side, 85
sg.dup, 85
sg.limit_nl(n,l), 85
sg.omega_shift
  (x), 85
sg.phase_residual(sg2), 85
sg.phase_shift
  (phi,z), 85
sg.repeat, 85
sg.repeat=, 85
sg.resolutionPlot(sg2.truepitch,xplot), 85
sg.reverse, 85
sg.x_shift
  (x), 85
spline, 31
spline2, 31
SplineArray.from_points, 83
SplineArray.new, 83
splineArray.y_to_x, 83
splineArray.y_to_ydash, 83
splineArray.ydash_to_x, 83
splineArray.ydash_to_y, 83
summarize, 40
tif2mrc, 21, 57
tiffseries2ps, 60
tube_tracer, 61
ub2unbent, 31