Notes for Using "Imfit"

Peter Erwin
MPE and USM
erwin@sigmaxi.net

February 24, 2013

Contents

1 What Is It? **3**

2 Getting and Installing `Imfit` **4**
2.1 Pre-Compiled Binaries 4
2.2 Building `Imfit` from Source: Outline 4
2.3 Building `Imfit` from Source: Details 5
2.3.1 Building with SCons 5
2.3.2 Building with Make 5
2.3.3 Tests 6
2.3.4 Telling the Compiler Where to Find Header Files and Libraries . . 6
2.3.5 Options: Compiling Without OpenMP Support 6
2.3.6 Options: Compiling Without FFT Multithreading 7

3 Trying It Out **7**

4 Using `Imfit` **8**
4.1 Command-line Flags and Options 8

5 The Configuration File **10**
5.1 Parameter Names, Specifications, and Values 12
5.2 Parameter Limits 13
5.3 Optional Image-Description Parameters 14

6 Standard Image Functions **14**

7 Images **16**
7.1 Specifying Image Subsections, Compressed Images, etc. 16

8 Extras for Fitting Images **17**
8.1 Masks . . . 17
8.2 Noise, Variance, or Weight Maps . . . 17
8.3 PSF Convolution . . . 18

9 Minimization Options: Levenberg-Marquardt, Differential Evolution, Nelder-Mead Simplex **18**
9.1 Controlling the Tolerance for Minimization . . . 20

10 Outputs **20**
10.1 Main Outputs . . . 20
10.2 Uncertainties on Parameter Values: L-M Estimates vs. Bootstrap Resampling . . . 21

11 Makeimage **22**
11.1 Using `Makeimage` . . . 22
11.2 Configuration Files for `Makeimage` . . . 23
11.3 Generating Single-Function Output Images . . . 23
11.4 Using `Makeimage` to Estimate Fluxes and Magnitudes . . . 23

12 Rolling Your Own Functions **24**
12.1 Basic Requirements . . . 24
12.2 A Simple Example . . . 25
12.2.1 Create and Edit the Header File . . . 25
12.2.2 Create and Edit the Class File . . . 26
12.2.3 Edit add_functions.cpp . . . 27
12.2.4 Edit the SConstruct File . . . 28

13 Acknowledging Use of `Imfit` **28**

A Standard Functions in Detail **29**
A.1 2D Functions . . . 29
A.1.1 FlatSky . . . 29
A.1.2 Gaussian . . . 29
A.1.3 Moffat . . . 30
A.1.4 Exponential . . . 30
A.1.5 Exponential_GenEllipse . . . 30
A.1.6 Sersic . . . 31
A.1.7 Sersic_GenEllipse . . . 31
A.1.8 Core-Sersic . . . 32
A.1.9 BrokenExponential . . . 32
A.1.10 GaussianRing . . . 33
A.1.11 GaussianRing2Side . . . 33
A.1.12 EdgeOnDisk . . . 33
A.1.13 EdgeOnRing . . . 34
A.1.14 EdgeOnRing2Side . . . 34
A.2 3D Functions . . . 35

A.2.1 ExponentialDisk3D 35
A.2.2 GaussianRing3D 36

B Acknowledgments **36**
B.1 Data Sources 37
B.2 Specific Software Acknowledgments 37
B.2.1 Minpack 37

1 What Is It?

`Imfit` is a program for fitting astronomical images — specifically, for fitting images of galaxies, though it could certainly be used for fitting other sources. The user specifies a set of one or more 2D surface-brightness functions (e.g., elliptical exponential, elliptical Sérsic, circular Gaussian) which will be added together in order to generate a model image; this model image will then be matched to the input image by adjusting the 2D function parameters via nonlinear minimization of the total χ^2 (or of the total Cash statistic in the alternate case of pure Poisson statistics).

The 2D functions can be grouped into arbitrary sets sharing a common (x, y) position on the image plane; this allows galaxies with off-center components or multiple galaxies to be fit simultaneously. Parameters for the individual functions can be held fixed or restricted to user-specified ranges. The model image can (optionally) be convolved with a point spread function (PSF) image to better match the input image; the PSF image can be any square, centered image the user supplies – e.g., an analytic 2D Gaussian or Moffat, a *Hubble Space Telescope* PSF generated by the TinyTim program[1] [Krist, 1995], or an actual stellar image.

A key part of `imfit` is a modular, object-oriented design that allows easy addition of new, user-specified 2D image functions. This is accomplished by writing C++ code for a new image-function class (this can be done by copying and modifying an existing pair of `.h/.cpp` files for one of the pre-supplied image functions), making small modifications to two additional files to include references to the new function, and re-compiling the program.

An additional auxiliary program called `makeimage`, built from the same codebase, exists for generating artificial galaxy images (using the same input/output parameter-file format as `imfit`).

`Imfit` is an open-source project; the source code is freely available under the GNU Public License (GPL).

System Requirements: `Imfit` has been built and tested on Intel-based MacOS X (Snow Leopard and Lion) and Linux (Ubuntu) systems. It uses standard C++ and should work on any Unix-style system with a modern C++ compiler and the Standard Template Library (e.g., GCC v4 or higher[2]). It relies on two external, open-source libraries: version 3 of the CFITSIO library[3] for FITS image I/O and version 3 of the FFTW

[1] `http://www.stsci.edu/hst/observatory/focus/TinyTim`
[2] GCC v4.2 or higher is necessary to take advantage of OpenMP-related speedups.
[3] `http://heasarc.nasa.gov/fitsio/`

(Fastest Fourier Transform in the West) library[4] for PSF convolution. Some optional components require the GNU Scientific Library (GSL),[5] and the NLopt library[6], but the program can also be built without these.

`Imfit` also includes modified versions of Craig Markwardt's `mpfit` code (an enhanced version of the MINPACK-1 Levernberg-Marquardt least-squares fitting code) and the Differential Evolution fitting code of Rainer Storn and Kenneth Price (more specifically, a C++ wrapper written by Lester E. Godwin).

2 Getting and Installing `Imfit`

2.1 Pre-Compiled Binaries

Pre-built binaries for Intel-based MacOS X and Linux systems, along with the source code, are available at `http://www.mpe.mpg.de/~erwin/code/imfit/`. The pre-compiled binaries included statically linked versions of the CFITSIO, FFTW, GSL, and NLopt libraries, so you do not need to have those installed.

2.2 Building `Imfit` from Source: Outline

1. Install the CFITSIO library (version 3.0 or higher).
2. Install the FFTW library (version 3.0 or higher) — note that if you have a multi-core CPU (or multiple CPUs sharing main memory), you should install the threaded version of FFTW as well, since this speeds up PSF convolution.
3. (Optional) Install the NLopt library — this is only necessary if you wish to use the Nelder-Mead minimization algorithm. `Imfit` can easily be built without this, if for some reason you don't have access to the NLopt library.
4. (Optional) Install the GNU Scientific Library (GSL) — this is only necessary if you wish to use image functions that rely on GSL. Currently, the only such functions are the EdgeOnDisk (`func_edge-on-disk.cpp`) component, which uses a modified Bessell function, and the sample 3D line-of-sight integration functions (e.g., ExponentialDisk3D, `func_expdisk3d.cpp`). `Imfit` can easily be built without these components, if for some reason you don't have access to the GSL.
5. Install SCons (if needed; see below).
6. Build `imfit` and `makeimage`.
7. (Optional) Run test scripts `do_imfit_tests` and `do_makeimage_tests`.

[4] `http://www.fftw.org/`
[5] `http://www.gnu.org/s/gsl/`
[6] `http://ab-initio.mit.edu/wiki/index.php/NLopt`

2.3 Building `Imfit` from Source: Details

Assuming that CFITSIO and FFTW (and optionally NLopt and GSL) have already been installed on your system, unpack the source-code tarball (imfit-x.x-source.tar.gz) in some convenient location.

2.3.1 Building with SCons

By default, `imfit` uses SCons for the build process. SCons is a Python-based build system that is somewhat easier to use and more flexible than the traditional `make` system; it can be downloaded from `http://www.scons.org/`.

If things are simple, you should be able to build `imfit` and the companion program `makeimage` with the following commands:

```
$ scons imfit
$ scons makeimage
```

This will produce two binary executable files: `imfit` and `makeimage`. Copy these to some convenient place on your path.

If you do not have GSL installed, you will get compilation errors; use the following commands instead:

```
$ scons --no-gsl imfit
$ scons --no-gsl makeimage
```

Similarly, you will get compilation errors if you do not have the NLopt library installed; this can be dealt with by using:

```
$ scons --no-nlopt imfit
$ scons --no-nlopt makeimage
```

Various other compilation options may be useful; these are explained in the next subsections (note that all the SCons options can be combined on the command line).

2.3.2 Building with Make

As an alternative to using SCons, a preliminary version of a configure script and Makefile are included; you can use them to build `imfit` and `makeimage` via

```
$ ./configure
$ make all
```

The configure script will check for the presence of the various required and optional libraries in standard locations; if it cannot find the GSL or NLopt libraries, then compilation will be done without them (see below for specifying the path to the libraries manually).

As in the case of using SCons, this will produce two binary executable files: `imfit` and `makeimage`. Copy these to some convenient place on your path.

2.3.3 Tests

Finally, there are two shell scripts – `do_imfit_tests` and `do_makeimage_tests` – which can be run to do some very simple sanity checks (e.g., do the programs fit some simple images correctly, are common config-file errors caught, etc.). They make use of files and data in the `tests/` subdirectory. Possible differences in output at the level of the least significant digit may occur; these are not a problem. (For the full set of tests to run, you should have Python version 2.6 or 2.7 installed, along with the numpy[7] and pyfits[8] Python libraries. If these are not available, then the parts of the tests which compare output images with reference versions will simply be skipped.)

2.3.4 Telling the Compiler Where to Find Header Files and Libraries

By default, the SConstruct file (the equivalent of a Makefile for SCons) tells SCons to look for header files in `/usr/local/include` and library files in `/usr/local/lib`. If you have the FFTW, CFITSIO, and (optionally) NLopt and GSL headers and libraries installed somewhere else, you can tell SCons about this by using the `--header-path` and `--lib-path` options:

```
$ scons --header-path=/some/path ...
$ scons --lib-path=/some/other/path ...
```

(note that "..." is meant to stand for the rest of the compilation command, whatever that may be).

Multiple paths can be specified if they are separated by colons, e.g.

```
$ scons --lib-path=/some/path:/some/other/path ...
```

If you are using `configure + make` instead of SCons, you can specify the paths to the relevant header and library files via:

./configure CPPFLAGS=-I/the/location/include LDFLAGS=-L/the/location/lib

2.3.5 Options: Compiling Without OpenMP Support

By default, `imfit` and `makeimage` are compiled to take advantage of OpenMP compiler support, which speeds up image computation by splitting it up across multiple CPUs (and multiple cores within multi-core CPUs). Currently, the code uses OpenMP 2.5 options, which means that if you are using the GCC compiler, you need version 4.2 or higher. If your compiler does not support OpenMP – or you want, for whatever reason, a version that does not include OpenMP support, you can disable it by compiling with the following commands:

```
$ scons --no-openmp imfit
$ scons --no-openmp makeimage
```

[7] `http://numpy.scipy.org/`

[8] `http://www.stsci.edu/institute/software_hardware/pyfits`

If you are using the `configure + make` approach instead, you can use the following when running `configure`:

```
$ ./configure --disable-openmp
```

2.3.6 Options: Compiling Without FFT Multithreading

By default, `imfit` and `makeimage` are compiled to take advantage of multi-core CPUs (and other shared-memory multiple-processor systems) when performing PSF convolutions by using the multithreaded version of the FFTW library. If you do not have (or cannot build) the multithreaded FFTW library, you can remove multithreaded FFT computation by compiling with the following commands:

```
$ scons --no-threading imfit
$ scons --no-threading makeimage
```

3 Trying It Out

In the `examples/` directory are some sample galaxy images, masks, PSF images, and configuration files.

To give `imfit` a quick spin (and check that it's working on your system), change to the `examples/` directory and execute the following on the command line (assuming that `imfit` is now in your path):

```
$ imfit ic3478rss_256.fits -c config_sersic_ic3478_256.dat --sky=130.14
```

This converges to a fit in a few seconds or less (e.g., about 0.5 seconds on a 2011 MacBook Pro with a 2.3 GHz Core i7 processor). In addition to being printed to the screen, the final fit is saved in a file called `bestfit_parameters_imfit.dat`.

The preceding command told `imfit` to fit using every pixel in the image and to estimate the noise assuming an original (previously subtracted) sky level of 130.14, an A/D gain of 1.0, and zero read noise (the latter two are default values). A better approach would be to include a mask (telling `imfit` to ignore, e.g., pixels occupied by bright stars) and to specify more accurate values of the gain and read noise:

```
$ imfit ic3478rss_256.fits -c config_sersic_ic3478_256.dat --mask ic3478rss_256_mask.fits
--gain=4.725 --readnoise=4.3 --sky=130.14
```

If you want to see what the best-fitting model looks like, you can use the companion program `makeimage` on the output file:

```
$ makeimage bestfit_parameters_imfit.dat --refimage ic3478rss_256.fits
```

This will generate and save the model image in a file called `modelimage.fits`. (`Imfit` itself can save the best-fitting model image at the end of the fitting process if the `--save-model` option is used.)

You can also fit the image using PSF convolution, by adding the "`--psf`" option and a valid FITS image for the PSF; the `examples/` directory contains a Moffat PSF image which matches stars in the original image fairly well:

imfit ic3478rss_256.fits -c config_sersic_ic3478_256.dat --mask ic3478rss_256_mask.fits --gain=4.725 --readnoise=4.3 --sky=130.14 --psf psf_moffat_51.fits

The PSF image was generated using `makeimage` and the configuration file `makeimage_config_moffat_psf_51_for_ic3478rss.dat`:

makeimage --ncols=51 --nrows=51 -o psf_moffat_51.fits makeimage_config_moffat_psf_51_for_ic3478rss.dat

4 Using `Imfit`

Basic use of `imfit` from the command line looks like this:

`$ imfit` -c *config-file* *input-image* [options]

where *config-file* is the name of the configuration file which describes the model (the combination of 2D functions, initial values for parameters, and possible limits on parameter values) and *input-image* is the FITS image we want to fit with the model.

The "options" are a set of command-line flags and options (use "`imfit -h`" or "`imfit --help`" to see the complete list). Options must be followed by an appropriate value (e.g., a filename, an integer, a floating-point number); this can be separated from the option by a space, or they can be connected with an equals sign. In other words, both of the following are valid:

```
imfit --gain 2.5
imfit --gain=2.5
```

Note that `imfit` does not follow the full GNU standard for command-line options and flags (as implemented by, e.g., the GNU `getopt` library): you cannot merge multiple one-character flags into a single item (if "`-a`" and "`-b`" are flags, "`-a -b`" will work, but "`-ab`" will *not*), and you cannot merge a one-character option and its target ("`-cfoo.dat`" is *not* a valid substitute for "`-c foo.dat`").

4.1 Command-line Flags and Options

Some notable and useful command-line flags and options include:

- `-c, --config` *config-file* — the only *required* command-line option, which tells `imfit` the name of the configuration file. (Actually, if you don't supply this option, `imfit` will look for a file called "imfit_config.dat", but it's best to explicitly specify your own configuration files.)

- `--psf` *psf-image* — specifies a FITS image to be convolved with the model image.

- `--mask` *mask-image* — specifies a FITS image which marks bad pixels to be ignored in the fitting process. By default, zero values in the mask indicate *good* pixels, and positive values indicate bad pixels.

- `--mask-zero-is-bad` — indicates that zero values (actually, any value < 1.0) in the mask correspond to *bad* pixels, with values ≥ 1.0 being good pixels.

- `--noise` *noisemap-image* — specifies a pre-existing noise or error FITS image to use in the χ^2 fitting process (by default, pixel values in the noise map are assumed to be Gaussian sigma values).

- `--errors-are-variances` — indicates that pixel values in the noise map are variances (sigma^2) instead of sigmas.

- `--errors-are-weights` — indicates that pixel values in the noise map should be interpreted as weights, not as sigmas or variances. (None of these three options are usable with Cash statistic minimization.)

- `--sky` *sky-level* — specifies an original constant sky background level (in counts/pixel) that was subtracted from the image; used for internal computation of the noise map (for χ^2 minimization) or for correcting the Cash statistic computation. (If the pixel units are counts/sec, then the sky level should also be in those units, and you should use the "`--exptime`" option to specify the original exposure time.)

- `--gain` *value* — specifies the A/D gain (in electrons/ADU) of the input image; used for internal computation of the noise map for χ^2 minimization.

- `--readnoise` *value* — specifies the read noise (in electrons) of the input image; used for internal computation of the noise map for χ^2 minimization.

- `--exptime` *value* — specifies the exposure time of the image; this should **only** be used *if* the image has been divided by the exposure time (i.e., if the pixel units are counts/sec).

- `--ncombined` *value* — if values in the input image are the result of averaging (or computing the median of) two or more original images, then this option should be used to specify the number of original images; used for internal computation of the noise map for χ^2 minimization. If multiple images were *added* together with no rescaling, then do not use this option.

- `--save-params` *output-filename* — specifies that parameters for best-fitting model should be saved using the specified filename (default is for these to be saved in a file named `bestfit_parameters_imfit.dat`).

- `--save-model` *output-filename* — the best-fitting model image will be saved using the specified filename.

- `--save-residual` *output-filename* — the residual image (input image $-$ best-fitting model image) will be saved using the specified filename.

- `--nm` — use Nelder-Mead simplex instead of Levenberg-Marquardt as the minimization technique (WARNING: slower)
- `--de` — use Differential Evolution instead of Levenberg-Marquardt as the minimization technique (WARNING: much slower!)
- `--cashstat` — use Cash statistic instead of χ^2 as the fit statistic for minimization; for use in the case of Poisson statistics and zero read noise. Cannot be used with the (default) Levenberg-Marquardt minimization technique.
- `--ftol` *FTOL-value* — specify tolerance for fractional improvements in the fit statistic (χ^2 or Cash statistic) value; if further iterations do not reduce the fit statistic by more than this, the minimization is considered a success and halted (default value = 10^{-8})
- `--bootstrap` *n-iterations*— Do *n-iterations* rounds of bootstrap resampling after the fit, to estimate parameter errors.
- `--chisquare-only` — Evaluate the χ^2 value for the initial input model as a fit to the input image, *without* doing any minimization to find a better solution.
- `--max-threads` *n-threads* — specifies the maximum number of CPU cores to use during computation (the default is to use *all* available CPU cores); has no effect if `imfit` was compiled without OpenMP or FFTW multithreading support.
- `--list-functions` — list all the functions `imfit` can use.
- `--list-parameters` — list all the individual parameters (in correct order) for each of the functions that `imfit` can use.

5 The Configuration File

`Imfit` always requires a configuration file, which specifies the model which will be fit to the input image, initial values for model parameters, any limits on parameter values (optional for fitting with the Levenberg-Marquardt solver, but required for fitting with the Differential Evolution solver), and possibly additional information (e.g, gain and read noise for the input image).

The configuration file should be a plain text file. Blank lines and lines beginning with "#" are ignored; in fact, anything on the same line after a "#" is ignored, which allows for comments at the end of lines.

A model for an image is specified by one or more **function blocks**, each of which is a group of one or more 2D image functions sharing a common (x, y) spatial position. Each function-specification consists of a line beginning with "FUNCTION" and containing

the function name, followed by one or more lines with specifications for that function's parameters.

More formally, the format for a configuration file is:

1. Optional specifications of general parameters and settings (e.g., the input image's A/D gain and read noise)
2. One or more function blocks, each of which contains:
   (a) X-position parameter-specification line
   (b) Y-position parameter-specification line
   (c) One or more function + parameters specifications, each of which contains:
      i. `FUNCTION` + function-name line
      ii. one or more parameter-specification lines

This probably sounds more complicated than it is in practice. Here is a very bare-bones example of a configuration file:

```
X0   150.1
Y0   149.5
FUNCTION Exponential
PA   95.0
ell   0.45
I_0   90.0
h   15.0
```

This describes a model consisting of a single elliptical exponential function, with initial values for the x and y position on the image, the position angle (PA), the ellipticity (ell), the central intensity (I_0) in counts/pixel, and the exponential scale length in pixels (h). None of the parameters have limits on their possible values.

Here is the same file, with some additional annotations and with limits on some of the parameters (comments are colored red for clarity):

```
# This line is a comment

X0    150.1   148,152
Y0    149.5   148,152 # a note
FUNCTION Exponential   # here is a comment
PA   95.0   0,180    # limits on the position angle
ell   0.45   0,1     # ellipticity should always be 0--1
I_0   90.0   fixed     # keep central intensity fixed
h   15.0
```

Here we can see the use of comments (lines or parts of lines beginning with "#") and the use of parameter limits in the form of "lower,upper": the X0 and Y0 parameters are required to remain ≥ 148 and ≤ 152, the position angle is limited to 0–180, the ellipticity must stay ≥ 0 and ≤ 1, and the central intensity I_0 is held fixed at its initial value.

Finally, here is a more elaborate example, specifying a model that has two function blocks, with the first block having two individual functions (so this could be a model for, e.g., simultaneously fitting two galaxies in the same image, one as Sérsic + exponential, the other with just an exponential):

```
# This line is a comment

GAIN 2.7 # A/D gain for image in e/ADU
READNOISE 4.5 # image read-noise in electrons

# This is the first function block:  Sersic + exponential
X0    150.1    148,152
Y0    149.5    148,152
FUNCTION Sersic # A Sersic function
PA    95.0     0,180
ell    0.05    0,1
n     2.5      0.5,4.0    # Sersic index
I_e    20.0     # intensity at the half-light radius
r_e    5.0     # half-light radius in pixels
FUNCTION Exponential
PA    95.0     0,180
ell    0.45    0,1
I_0    90.0    fixed
h     15.0

# This is the second function block:  just a single exponential
X0    225.0    224,226
Y0    181.7    180,183
FUNCTION Exponential
PA    22.0     0,180
ell    0.25    0,1
I_0    10.0
h      20.0
```

5.1 Parameter Names, Specifications, and Values

The X0/Y0 position lines at the start of each function block and the individual parameter lines for each function all share a common format:

parameter-name *initial-parameter-value* *optional-limits*

The separation between the individual pieces must consist of one or more spaces and/or tabs. The final piece specifying the limits is optional (except that fitting in Differential Evolution mode *requires* that there be limits for each parameter).

Parameter Names: The X0/Y0 positional parameters for each function block must be labeled "X0" and "Y0". Names for the parameters of individual functions can be anything the user desires; only the order matters. Thus, the position-angle parameter could be labeled "PA", "PosAngle", "angle", or any non-space-containing string — though it's a good idea to have it be something relevant and understandable.

Important Note: *Do not change the order of the parameters for a particular function!* Because the strings giving the parameter names can be anything at all, `imfit` actually ignores them and simply assumes that all parameters are in the correct order for each function.

Note that any output which `imfit` generates will use the default parameter names defined in the individual function code (use "`--list-parameters`" to see what these are for each function).

Values for Positional Parameter (X0, Y0): The positional parameters for each function block are pixel values – X0 for the column number and Y0 for the row number. `Imfit` uses the IRAF pixel-numbering convention: the center of first pixel in the image (the lower left pixel in a standard display) is at $(1.0, 1.0)$, with the lower-left corner of that pixel having the coordinates $(0.5, 0.5)$.

General Parameter Values for Functions: The meaning of the individual parameter values for the various 2D image functions is set by the functions themselves, but in general:

- position angles are measured in degrees counter-clockwise from the image's vertical ($+y$) axis (i.e., degrees E of N if the image has standard astronomical orientation);
- ellipticity $= 1 - b/a$, where a and b are the semi-major and semi-minor axes of an ellipse;
- intensities are in counts/pixel;
- lengths are in pixels.

If you write your own functions, you are encouraged to stick to these conventions.

5.2 Parameter Limits

Individual parameters can be limited in two ways:

1. Held fixed;
2. Bounded between lower and upper limits.

To hold a parameter fixed, use the string "fixed" after the initial-value specification. E.g.:

```
X0   442.85   fixed
```

To specify lower and upper limits for a parameter, include them as a comma-separated pair following the initial-value specification. E.g.:

```
X0   442.85   441.0,443.5
```

5.3 Optional Image-Description Parameters

The configuration file can, optionally, contain one or more specifications of parameters describing the whole image, which take the place of certain command-line options for computing the internal noise map. The specifications should be placed at the beginning of the configuration file, *before* the first function block is described. The format is the same as for other parameters in the configuration file: the name of the parameter, followed by one or more spaces and/or tabs, followed by a numerical value. E.g.,

```
GAIN 2.7
READNOISE 4.5
```

The currently available image-description parameters are (see Section 4.1 for more details about the corresponding command-line options):

- GAIN – same as command-line option `--gain` (A/D gain in electrons/ADU)
- READNOISE – same as command-line option `--readnoise` (read noise in electrons)
- EXPTIME – same as command-line option `--exptime`
- NCOMBINED – same as command-line option `--ncombined`
- ORIGINAL_SKY – same as command-line option `--sky` (original background level that was subtracted from the image)

In situations where a configuration file contains one of these specifications and the corresponding command-line option is also used, *the command-line option always overrides whatever value is in the configuration file*.

6 Standard Image Functions

`Imfit` comes with the following 2D image functions, each of which can be used as many times as desired. (As mentioned above, `imfit` is designed so that constructing and using new functions is a relatively simple process.) Most of these functions use a specified radial intensity profile (e.g., Gaussian, exponential, Sérsic) with elliptical isophote shapes. Note that elliptical functions can always be made circular by setting the "ellipticity" parameter to 0.0 and specifying that it be held fixed. See Appendix A for more complete discussions of all functions, including their parameters.

- FlatSky — a uniform sky background.
- Gaussian — an elliptical 2D Gaussian function.
- Moffat — an elliptical 2D Moffat function.
- Exponential — an elliptical 2D exponential function.
- Exponential_GenEllipse — an elliptical 2D exponential function using generalized ellipses ("boxy" to "disky" shapes) for the isophote shapes.
- Sersic — an elliptical 2D Sérsic function.
- Sersic_GenEllipse — an elliptical 2D Sérsic function using generalized ellipses ("boxy" to "disky" shapes) for the isophotes.
- Core-Sersic — an elliptical 2D Core-Sérsic function [Graham et al., 2003, Trujillo et al., 2004].
- BrokenExponential — similar to Exponential, but with *two* exponential radial zones (with different scalelengths) joined by a transition region at $R_{\rm break}$ of variable sharpness.
- GaussianRing — an elliptical ring with a radial profile consisting of a Gaussian centered at $r = R_{\rm ring}$.
- GaussianRing2Side — like GaussianRing, but with a radial profile consisting of an asymmetric Gaussian (different values of σ for $r < R_{\rm ring}$ and $r > R_{\rm ring}$).
- EdgeOnDisk — the analytical form for a perfectly edge-on exponential disk, using the Bessel-function solution of van der Kruit & Searle [1981] for the radial profile and the generalized sech function of van der Kruit [1988] for the vertical profile. Note that this function requires that the GNU Scientific Library (GSL) be installed; if the GSL is not installed, `imfit` should be compiled without this function (see Section 2.3).
- EdgeOnRing — a simplistic model for an edge-on ring, using a Gaussian for the radial profile and another Gaussian (with different σ) for the vertical profile.
- EdgeOnRing2Side — like EdgeOnRing, but using an asymmetric Gaussian for the radial profile (see description of GaussianRing2Side).

In addition, two experimental "3D" functions are available. With these, the intensity value for each pixel comes from line-of-sight integration through a 3D luminosity-density model, generating a projected 2D model image given input specifications of the orientation and inclination to the line of sight. Both of these require the GSL for compilation from source (see Section 2.3).

- ExponentialDisk3D — uses a 3D luminosity-density model of an axisymmetric exponential disk (with different radial and vertical scale lengths), observed at an arbitrary inclination, to generate a projected surface-brightness image.

- GaussianRing3D — uses a 3D luminosity-density model of an elliptical ring with Gaussian radial and exponential vertical profiles.

A list of the currently available functions can always be obtained by running `imfit` with the "`--list-functions`" option:

```
$ imfit --list-functions
```

The complete list of function parameters for each function (suitable for copying and pasting into a configuration file) can always be obtained by running `imfit` with the "`--list-parameters`" option:

```
$ imfit --list-paremeters
```

7 Images

`Imfit` is designed to fit 2D astronomical images in FITS format, where pixel values are some form of linear surface-brightness (or surface density) measurement. The default internal error calculations (see Section 8.2, below) assume that pixel values are integrated counts (e.g., ADUs), which can be converted to detected photons using the A/D gain (provided by "`--gain`" option, or by the GAIN keyword in a configuration file). However, since `imfit` can also accept a user-supplied noise/error image in FITS format, you can use any linear pixel values as long as the corresponding noise image is appropriately scaled to match.

If your image is in counts/second, you can either multiply it by the exposure time to recover the integrated counts, or include the actual exposure time via the "`--exptime`" option (or the EXPTIME keyword in a configuration file).

If the image is an *average* of N input images of the same exposure time, you can either multipy the image by N *or* use the "`--ncombined`" option to tell `imfit` how to adjust the error estimations. The latter option is slightly better, because `imfit` will also scale the read noise accordingly.

`Imfit` does *not* assume the presence of any particular header keywords in the FITS file.

7.1 Specifying Image Subsections, Compressed Images, etc.

In many cases, you may want to fit an object which is much smaller than the whole image. You can always make a smaller cutout image and fit that, but it may be convenient to specify the image subsection directly. You can do this using a subset of the image-section syntax of CFITSIO (which will be familiar to you if you've ever worked with image sections in IRAF). An example:

```
ic3478rss_256.fits[45:150,200:310]
```

This will fit columns 45–150 and rows 200–310 of the image (column and row numbering starts at 1). Pixel coordinates in the configuration (and output) files refer to locations within the *full* image.

The only kind of image section specification that's allowed is a simple [x1:x2,y1:y2] format, though you can specify all of a particular dimension using an asterisk (e.g., [*,y1:y2] to specify the full range of x values). More complicated expressions which might extract part of a 3D datacube are not (currently) possible. However, you *can* specify a particular extension (header-data unit) in a multi-extension FITS file, e.g.:

ic3478rss.fits[2]

ic3478rss.fits[2][45:150,200:310]

Obviously, if you are also using a mask image (and/or a noise image), you should specify the same subsection in those images!

You can also use fit (or generate) images which have been compressed with gzip or Unix compress – e.g., `ic3478rss_256.fits.gz`. Images, masks, etc., can even be read via `http://` or `ftp://` URLs which point directly to accessible FITS files – e.g., `http://someplace.net/images/somefile.fits`; you cannot *save* files to URLs, however.

8 Extras for Fitting Images

8.1 Masks

A mask image can be supplied to `imfit` by using the command-line option `--mask`. The mask image should be an *integer*-valued FITS file with the same dimensions as the image being fitted (IRAF `.pl` mask files are not recognized, but these can be converted to FITS format within IRAF). The default is to treat zero-valued pixels in the mask image as *good* and pixels with values > 0 as *bad* (i.e., to be excluded from the fit); however, you can specify that zero-valued pixels are *bad* with the command-line flag `--mask-zero-is-bad`.

8.2 Noise, Variance, or Weight Maps

By default, `imfit` uses χ^2 as the statistic for minimization. As part of this, `imfit` normally calculates an internal weight map, using the input pixel intensities, the A/D gain, any previously subtracted background level, and the read noise to estimate Gaussian errors σ_i for each pixel i. The error-based weight map is $w_i = 1/\sigma_i^2$, with the dispersion defined as

$$\sigma_i^2 \;=\; (I_{d,i} + I_{\rm sky})/g_{\rm eff} \;+\; N_{\rm c}\,\sigma_{\rm rdn}^2/g_{\rm eff}^2\,, \tag{1}$$

where $I_{d,i}$ is the data intensity in counts/pixel, $I_{\rm sky}$ is the original subtracted sky background (if any), $\sigma_{\rm rdn}$ is the read noise, $N_{\rm c}$ is the number of separate images averaged to form the data image, and $g_{\rm eff}$ is the "effective gain" (the product of the A/D gain, $N_{\rm c}$, and optionally the exposure time, if the image units are counts/pixel). These weights are then used in the χ^2 calculation, summing over all N pixels:

$$\chi^2 \;=\; \sum_{i=0}^{N} w_i\,(I_{m,i} - I_{d,i})^2, \tag{2}$$

where $I_{m,i}$ and $I_{d,i}$ are the model and data intensities in counts/pixel, respectively. (Masking is handled by setting $w_i = 0$ for masked pixels.)

If you have a pre-existing error map as a FITS image, you can tell `imfit` to use that instead, via the `--noise` command-line option. By default, the pixel values in this image are assumed to be errors σ_i in units of ADU/pixel. If the values are *variances* (σ_i^2), you can specify this with the `--errors-are-variances` flag. You can also tell `imfit` that the pixel values in the noise map are actual *weights* w_i via the `--errors-are-weights` flag, if that happens to be the case. (If a mask image is supplied, the weights of masked pixels will still be set to 0, regardless of their individual values in the weight image.)

Note that `imfit` does *not* try to obtain information (such as the A/D gain or read noise) from the FITS header of an image. This is primarily because there is little consistency in header names across the wide range of astronomical images, so it is difficult pick one name, or even a small set, and assume that it will be present in a given image's header; this is even more true if an image is the result of a simulation.

If the user has specified the Cash statistic C instead of χ^2, the minimization uses

$$C \;=\; \sum_{i=0}^{N} w_i \, (I'_{m,i} \;-\; I'_{d,i} \ln I'_{m,i}), \tag{3}$$

where $I'_{m,i}$ and $I'_{d,i}$ are the model and data intensities in counts/pixel, multiplied by the effective gain $g_{\rm eff}$ as defined above. In this case, all weights are automatically $= 1$, except for masked pixels, which are still set $= 0$.

8.3 PSF Convolution

To simulate the effects of seeing and other telescope resolution effects, model images can be convolved with a PSF (point-spread function) image. This uses an input FITS file which contains the point spread function. The actual convolution uses Fast Fourier Transforms of the interally-generated model image and the PSF image to compute the output convolved model image.

PSF images should be square, ideally with width = an odd number of pixels, and the PSF should be centered in the central pixel. (An off-center PSF can certainly be used, but the resulting convolved model images will be shifted.) The PSF does *not* need to be normalized, as `imfit` will automatically normalize the PSF image internally.

Although `imfit` uses a multi-threaded version version of the FFTW library, which is itself quite fast, adding PSF convolution to the image-fitting process *does* slow things down considerably.

9 Minimization Options: Levenberg-Marquardt, Differential Evolution, Nelder-Mead Simplex

The default method for χ^2 minimization used by `imfit` is the Levenberg-Marquardt algorithm, based on the classic MINPACK-1 implementation with enhancements by Craig Markwardt.[9] This is very fast and robust, and is the most extensively tested algorithm

[9] Original C version available at `http://www.physics.wisc.edu/~craigm/idl/cmpfit.html`

in `imfit`, but requires an initial guess for the parameter values and can sometimes become trapped in local minima in the χ^2 landscape. In addition, it is *not* appropriate if one is using the Cash statistic, since the latter can potentially have both per-pixel and total values < 0, which the Levenberg-Marquardt algorithm cannot handle.

An alternative algorithm, available via the `--nm` flag, is a version of the Nelder-Mead simplex, as implemented by the NLopt library.[10] This is significantly slower than Levenberg-Marquardt minimization ($\sim$ 10 times slower for fits with one or two components), but significantly faster than Differential Evolution (below). Like the Levenberg-Marquardt method, it does require an initial guess for the parameter values, but is considered less likely to become trapped in local minima in the fit-statistic landscape. Unlike the L-M algorithm, it can be used for both χ^2 and Cash-statistic minimization. If you compile `imfit` from source and want to use this algorithm, you need to have the NLopt library installed on your system.

The second alternate algorithm is available via the `--de` flag. This performs the fit-statistic (χ^2 or Cash statistic) minimization using Differential Evolution (DE) [Storn & Price, 1997], a genetic-algorithms approach.[11] It has the drawback of being $\sim$ an order of magnitude slower than the Nelder-Mead simplex method, and *much* slower ($\sim$ two orders of magnitude) than Levenberg-Marquardt minimization. For example, fitting a single Sérsic function to the 256×256 image in the `examples/` subdirectory takes $\sim$ 60 times as long when using Differential Evolution as it does when using L-M minimization. It does has the advantage of being the least likely (at least in principle) of being trapped in local minima in the fit-statistic landscape; it also does *not* require an initial guess for the parameter values.

The Differential Evolution algorithm does, however, *require* lower and upper limits for *all* parameters in the configuration file (see Section 5.2); this is because DE generates parameter-value "genomes" by random uniform sampling from within the ranges specified by the parameter limits. The format of the configuration file still requires that initial-guess values be present for all parameters as well, but these are actually ignored by the DE algorithm. (This is to ensure that the same configuration file can be used with all minimization routines.)

TBD. [more details of DE implementation]

Note that the N-M simplex and DE algorithms do *not* produce uncertainty estimates for the best-fitting parameter values, in contrast to the Levenberg-Marquardt approach. However, the L-M error estimates are themselves only reliable if the minimum in the χ^2 landscape is symmetric and parabolic, and if the errors for the input image are truly Gaussian and well-determined. See Section 10.2 for an alternative (and probably more accurate) way of estimating the parameter uncertainties.

The fact that the minimization algorithms are relatively decoupled from the rest of the code means that future versions of `imfit` could include other minimization techniques, or that an ambitious user could add such techniques on their own.

[10] `http://ab-initio.mit.edu/wiki/index.php/NLopt`
[11] `http://www.icsi.berkeley.edu/~storn/code.html`

9.1 Controlling the Tolerance for Minimization

All three minimization algorithms have stop conditions based on fractional changes in the fit-statistic (χ^2 or Cash statistic) value: if further iterations do not improve the current value by more than FTOL $\times$ the fit statistic, then the algorithm declares success and terminates. (In the case of DE, the test condition is actually no further improvement after 30 generations.) The default value of FTOL is 10^{-8}, which seems to do a reasonable job for typical images (in fact, it's probably overkill). If you want to experiment with different values of FTOL, you can do this via the command-line option `--ftol`.

There are also built-in stopping conditions based on maximum number of iterations for the L-M algorithm (1000), maximum number of generations for DE (600), or maximum number of function evaluations (i.e., computations of model images) for the Nelder-Mead simplex algorithm (10000 times the number of free parameters).

TBD: info about extra stopping conditions for the L-M and N-M simplex algorithms

10 Outputs

10.1 Main Outputs

Assuming that the fitting process converges, `imfit` will print a summary of the results, including the final, best-fitting parameter values. The output parameter list is in the same format as the configuration file, except that if the L-M algorithm is used, its error estimates are listed after each parameter value.[12] These error estimates are separated from the parameter values by "#"; this means that you can copy and paste the parameter list into a text file and use that file as an input configuration file for `imfit` or `makeimage`.

The best-fitting parameters will also be written to an output text file (default name = `bestfit_parameters_imfit.dat`; use `--save-params` to specify a different name), *without* the error estimates. The output file will also include a copy of the original command used to start `imfit` and the date and time it was generated; these are commented out so that the file can be subsequently used as an `imfit` or `makeimage` configuration file without modification.

The final value of the fit statistic (χ^2 or Cash statistic) is also printed; if the fit statistic was χ^2, then the reduced χ^2 (which accounts for the total number of unmasked pixels and non-fixed parameter values)[13] is also printed. Finally, two alternate measures of the fit are also printed: the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). The latter two are included on a provisional basis; they are, in principle, useful for comparing different models fit to the same data.

By default, no model or residual (image $-$ model) images are saved, but the command-line options `--save-model` and `save-residual` can be used to specify output names for those images, and corresponding FITS files will be saved to disk.

TBD.

[12] No in-line error estimates are produced if the Nelder-Mead simplex or Differential Evolution algorithms were used for the minimization.

[13] The reduced χ^2 value should be interpreted with caution; it is valid in absolute terms only if the noise has been correctly estimated *and* if all differences between the model and the data are solely due to noise, which is rarely true for galaxies and other astronomical objects.

10.2 Uncertainties on Parameter Values: L-M Estimates vs. Bootstrap Resampling

The (default) Levenberg-Marquardt minimization algorithm used by `imfit` automatically generates a set of (symmetric) uncertainty estimates for each free parameter at the conclusion of the fitting process; as noted above, these are printed to the terminal as part of the summary output.

These values come from the covariance matrix derived from the final Jacobian matrix corresponding to the best-fit solution, and should be viewed with caution: for example, they assume that the χ^2 landscape in the vicinity of the best-fit solution is parabolic XXX. In practice, they should probably be seen as *lower limits* on the uncertainty.

The other fitting algorithms used by `imfit` do not compute gradients in the fit landscape, and so they do not produce automatic uncertainty estimates. Although one could certainly take the solution of a χ^2 fit done with the N-M simplex or DE algorithms and use it as input to a L-M run of `imfit`, thus generating L-M–based uncertainties, this is not possible when using the Cash statistic.

As an alternative to the somewhat dubious L-M uncertainty estimates – and as a method for estimating uncertainties in the case of Cash-statistic minimization – `imfit` offers the option of bootstrap resampling. This is done with the "`--bootstrap`" command-line option, which takes the *number* of iterations as its corresponding value; e.g.

```
imfit someimage.fits -c config.dat [. . .] --bootstrap 200
```

Each iteration of bootstrap resampling generates a new data image by sampling pixel values (with replacement) from the original data image, and then re-runs the fit to generate a new set of parameter values.[14] After n iterations, the combined set of bootstrapped parameter values can be used as a distribution for estimating confidence intervals; `imfit` finds and outputs the 68.3% confidence intervals and the standard deviation for each parameter. While this approach is likely to produce more plausible uncertainties for the best-fit parameters – and can be used with both χ^2 and Cash-statistic minimization and as an adjunct to any of the three minimization algorithms – it *is* slow, as one is essentially repeating (a somewhat faster version of) the fitting process n times. Ideally, one should do at least 200 iterations – 1000 or more is preferable – to get reasonably consistent confidence intervals. Since this can take *much* longer than the original fitting process, it is probably not a good idea to use bootstrap resampling when one is engaged in exploratory fitting, but to instead postpone it until one is reasonably certain one has the final fit. To keep things as fast as possible, `imfit` automatically chooses L-M minimization for the bootstrap process – unless the Cash statistic is being used, in which case the N-M simplex method is used.

(Future versions of `imfit` may include the option of dumping all bootstrapped parameter values to a text file, to allow more detailed inspection of the distributions.)

TBD.

[14] To speed things up, the original best-fit parameters are used as starting values for the new fit.

11 Makeimage

`Imfit` has a companion program called `makeimage`, which will generate model images using the same functions (and parameter files) as `imfit`. In fact (as noted above), the output "best-fitting parameters" file generated by `imfit` can be used as input to `makeimage`, as can an `imfit` configuration file.

`Makeimage` *does* require an output image size. This can be specified via command-line flags ("`--ncols`" and "`--nrows`"), via specifications in the configuration file (see below), or by supplying a reference FITS image ("`--refimage` *image-filename*"); in the latter case, the output image will have the same dimensions as the reference image.

`Makeimage` can also be run in a special mode to estimate the magnitudes and fractional luminosities of different components in a model.

11.1 Using `Makeimage`

Basic use of `makeimage` from the command line looks like this:

`$ makeimage` [options] *config-file*

where *config-file* is the name of the `imfit`-style configuration file which describes the model.

As for `imfit`, the "options" are a set of command-line flags and options (use "`makeimage -h`" or "`makeimage --help`" to see the complete list). Options must be followed by an appropriate value (e.g., a filename, an integer, a floating-point number); this can be separated from the option by a space, or they can be connected with an equals sign.

Some notable and useful command-line flags and options include:

- `-o, --output` *filename* — filename for the output model image (default = "modelimage.fits").
- `--refimage` *filename* — existing reference image to use for determining output image dimensions.
- `--ncols` *N_columns* — number of columns in output image
- `--nrows` *N_rows* — number of rows in output image
- `--psf` *psf-image* — specifies a FITS image to be convolved with the model image.
- `--nosave` — do *not* save the model image (useful for testing purposes, or when using `makeimage` to estimate fluxes)
- `--list-functions` — list all the functions `makeimage` can use
- `--list-parameters` — list all the individual parameters (in correct order) for each functions that `makeimage` can use

11.2 Configuration Files for `Makeimage`

The configuration file for `makeimage` has essentially the same format as that for `imfit`; any parameter limits that might be present are ignored.

Optional general parameters like GAIN and READNOISE are ignored, but the following optional general parameters are available:

- NCOLS — number of columns for the output image (x-size)
- NROWS — number of rows for the output image (y-size)

11.3 Generating Single-Function Output Images

`Makeimage` can also output individual images for each function in the configuration file. For example, if the configuration file specifies a model with one Sérsic function and two exponential functions, `makeimage` can generate three separate FITS files, in addition to the (standard) sum of all three functions. This is done with the `--output-functions` option:

`--output-functions` *root-name*

where *root-name* is a string that all output single-function filenames will start with. The single-function filenames will be sequentially numbered (starting with 1) according to the order of functions in the configuration file, and the name of each function will added to the end; the resulting filenames will have this format:

*root-name*N_*function-name*`.fits`

Using the example specified above (a model with one Sérsic and two exponential functions), one could execute the following command

```
$ makeimage config-file --output-functions mod
```

and the result would be three FITS files, named `mod1_Sersic.fits`, `mod2_Exponential.fits`, and `mod3_Exponential.fits` (in addition to `model.fits`, which is the sum of all three functions).

11.4 Using `Makeimage` to Estimate Fluxes and Magnitudes

Given a configuration file, you can use `makeimage` to estimate the total fluxes and magnitudes of different model components. For some components – e.g., the purely elliptical versions of the Gaussian, Exponential, and Sersic functions – there are analytical expressions which could be used. But since `imfit` and `makeimage` are designed to use arbitrary functions, including ones which do not have analytical expressions for total flux, `makeimage` estimates the flux for each component by internally constructing a large model image for each component function in the configuration file, with the component centered within this image, and then summing the pixel values of that image. The output includes a list of total and relative fluxes for each component in the model image (and their magnitudes, if a zero point is supplied).

`$ makeimage` –print-fluxes *config-file*

Useful command-line flags and options:

- `--estimation-size` *N_columns_and_rows* — size of the (square) image to construct (the default size is 5000 pixels on a side)
- `--zero-point` *value* — zero point for converting total counts to magnitudes:

$$m = Z - 2.5 \log_{10}(\text{counts}) \tag{4}$$

This enables you to compute things like bulge/total ratios – but it's up to you to determine which component(s) should be considered "bulge", "disk", etc.

When run in this mode, `makeimage` will still produce an output image file – unless you also specify the `--nosave` option.

12 Rolling Your Own Functions

12.1 Basic Requirements

A new image function should be implemented in C++ as a subclass of the FunctionObject base class (`function_object.h`, `function_object.cpp`). At a minimum, it should provide its own implementation of the following public methods, which are defined as virtual methods in the base class:

- The class constructor — in most cases the code for this can be copied from any of the existing FunctionObject subclasses, unless some special extra initialization is needed.
- Setup() — this is used by the calling program to supply the current set of function parameters (including the (x_0, y_0) pixel values for the center) prior to determining intensity values for individual pixels. This is a convenient place to do any general calculations which don't depend on the exact pixel (x, y) values.
- GetValue() — this is used by the calling program to obtain the surface brightness for a given pixel location (x, y). In existing FunctionObject classes, this method often calls other (private) methods to handle details of the calculation.
- GetClassShortName() – this is a class function which is used to obtain the short version of the class name as a string.

The new class should also redefine the following internal class constants:

- `N_PARAMS` — the number of input parameters (*excluding* the central pixel coordinates);
- `PARAM_LABELS` — an array of string labels for the input parameters;
- `FUNCTION_NAME` — a short string describing the function;

- `className` — a string (no spaces allowed) giving the official name of the function.

The `add_functions.cpp` file should then be updated by:

1. including the header file for the new class;
2. adding 2 lines to the PopulateFactoryMap() function to add the ability to create an instance of the new class.

Finally, the name of the C++ implementation file for the new class should be added to the `Sconstruct` file to ensure it gets included in the compilation.

Existing examples of FunctionObject subclasses can be found in the "`function_objects`" subdirectory of the source-code distribution, and are the best place to look in order to get a better sense of how to implement new FunctionObject subclasses.

12.2 A Simple Example

To illustrate, we'll make a new version of the Moffat function (which already exists, so this is purely for pedagogical purposes) by copying and modifying the code for the Gaussian function.

We need to make three sets of changes:

- Change the class name from "Gaussian" to our new name ("NewMoffat");
- Change the relevant code which computes the function;
- Rename, add, or delete variables to accomodate the new algorithm.

12.2.1 Create and Edit the Header File

Change directory to the directory with the `imfit` source code, and then `cd` to the "`function_objects`" subdirectory. Copy the file `func_gaussian.h` and rename it to `func_new-moffat.h`. Edit this file and change the following lines:

```
#define CLASS_SHORT_NAME  "Gaussian"
```

(replace `"Gaussian"` with `"NewMoffat"`)

```
class Gaussian : public FunctionObject
```

(replace `Gaussian` with `NewMoffat`)

```
    Gaussian( );
```

(replace `Gaussian` with `NewMoffat`)

And finally edit the list of class data members, changing this:

```
  private:
    double  x0, y0, PA, ell, I_0, sigma;   // parameters
    double  q, PA_rad, cosPA, sinPA;   // other useful (shape-related) quantities
```

to this:

```
private:
  double  x0, y0, PA, ell, I_0, fwhm, beta;   // parameters
  double  alpha;
  double  q, PA_rad, cosPA, sinPA;   // other useful (shape-related) quantities
```

12.2.2 Create and Edit the Class File

Copy the file `func_gaussian.cpp` and rename it to `func_new-moffat.cpp`.

Initial changes, including parameter number and names:

Edit this file and change the following lines (changed text indicated in red):

```
#include "func_new-moffat.h"
```

const int N_PARAMS = 5;

const char PARAM_LABELS[][20] = {"PA", "ell", "I_0", "fwhm", "beta"};

const char FUNCTION_NAME[] = "Moffat function";

Change references to class name:

Change all class references from "Gaussian" to "NewMoffat" (e.g., `Gaussian::Setup` becomes `NewMoffat::Setup`).

Changes to Setup method:

In the Setup method, you need to change how the input is converted into parameters, and do any useful pre-computations. So the initial processing of the "params" input changes from this:

```
PA = params[0 + offsetIndex];
ell = params[1 + offsetIndex];
I_0 = params[2 + offsetIndex];
sigma = params[3 + offsetIndex];
```

to this:

```
PA = params[0 + offsetIndex];
ell = params[1 + offsetIndex];
I_0 = params[2 + offsetIndex];
fwhm = params[3 + offsetIndex];
beta = params[4 + offsetIndex];
```

and at the end we replace this:

```
  twosigma_squared = 2.0 * sigma*sigma;
```

with this:

```
  // compute alpha:
  double  exponent = pow(2.0, 1.0/beta);
  alpha = 0.5*fwhm/sqrt(exponent - 1.0);
```

Changes to CalculateIntensity method:

Although it is the public method GetValue() which is called by other parts of the program, we don't actually need to change the current version of that method in this simple example. The code in the original Gaussian version of GetValue() just converts pixel positions to a scaled radius value, given input values for the center, ellipticity, and position angle, and then calls the private method CalculateIntensity() to determine the intensity as a function of the radius. Since we're still assuming a perfectly elliptical shape, we can keep the existing code. It probably doesn't make sense to change the CalculateSubsamples method, either, so we can leave that alone.

Instead, we actually implement the details of the new function's algorithm in CalculateIntensity(). Replace the original version of this method with the following:

```
double NewMoffat::CalculateIntensity( double r )
{
  double  scaledR, denominator;

  scaledR = r / alpha;
  denominator = pow((1.0 + scaledR*scaledR), beta);
  return (I_0 / denominator);
}
```

At this point, most of the work is done. We only need to update `add_functions.cpp` so it knows about the new function and update the SConstruct file so that the new function is included in the compilation.

12.2.3 Edit add_functions.cpp

We need to do two simple things here:

1. Include the header file for our new function. Add the following line near the top of the file, where the other header files are included:
   `#include "func_new-moffat.h"`

2. Add code to generate an instance of our new class as part of the function-factory map. Inside the function PopulateFactoryMap, add the following lines:

   ```
   NewMoffat::GetClassShortName(classFuncName);
   input_factory_map[classFuncName] = new funcobj_factory<NewMoffat>();
   ```

12.2.4 Edit the SConstruct File

In the SConstruct file, locate the place where the variable "functionobject_obj_string" is defined (currently somewhere near line 280, though this may change in the future). This is a string containing a compact list of all the filenames containing function-object code. Insert our new function's name ("func_new-moffat") into the list.

That's it! You should now be able to recompile `imfit` and `makeimage` (see Section 2.3) to use the new function. (Assuming there aren't any bugs in your new code....)

13 Acknowledging Use of `Imfit`

A paper describing `imfit` is currently in preparation; until it is available, you can reference the current URL (`http://www.mpe.mpg.de/~erwin/code/imfit/`) if `imfit` has been useful in your research.

A Standard Functions in Detail

Unless otherwise noted, all "intensity" parameters (`I_sky`, `I_0`, `I_e`, etc.) are in units of counts per pixel, and all lengths are in pixels.

A sample function specification (giving the parameters in their proper order), as you would use in a configuration file, is listed for each function description.

"Elliptical" functions are defined to have an intensity which is constant on concentric, similar ellipses (with specified ellipticity and major-axis position angle); the intensity profile is defined as a function of the semi-major axis a.

A.1 2D Functions

The main set of image functions provided create 2D intensity distributions directly. These include most of the usual suspects used in 2D image fitting: constant background, Gaussian, exponential, Sérsic, etc.

Common parameters:

- `PA` = position angle (e.g., of the major axis), measured in degrees CCW from the image +y axis. This is equivalent to standard astronomical position angles *if* your image has standard astronomical orientation (N up, E to the left).
- `ell` = ellipticity ($1 - b/a$, where a and b are semi-major and semi-minor axes of the ellipse, respectively).

A.1.1 FlatSky

A uniform background: $I(x, y) = I_{\rm sky}$ everywhere.

```
FUNCTION FlatSky
I_sky
```

A.1.2 Gaussian

This is an elliptical 2D Gaussian function, with the major-axis intensity profile given by

$$I(a) \;=\; I_0 \exp(-a^2/(2\sigma^2)). \tag{5}$$

```
FUNCTION Gaussian
PA
ell
I_0
sigma
```

A.1.3 Moffat

This is an elliptical 2D Moffat function, with the major-axis intensity profile given by

$$I(a) \;=\; \frac{I_0}{(1+(a/\alpha)^2)^{\beta}}, \tag{6}$$

where α is defined as

$$\alpha \;=\; \frac{\mathrm{FWHM}}{2\sqrt{2^{1/\beta}-1}}. \tag{7}$$

In practice, FWHM describes the overall width of the profile, while β describes that strength of the wings: lower values of β mean more intensity in the wings than is the case for a Gaussian (as $\beta \to \infty$, the Moffat profile approaches a Gaussian).

The Moffat function is often a good approximation to typical telescope PSFs (see, e.g., Trujillo et al. 2001), and `makeimage` can easily be used to generate Moffat PSF images.

```
FUNCTION Moffat
PA
ell
I_0
fwhm
beta
```

A.1.4 Exponential

This is an elliptical 2D exponential function, with the major-axis intensity profile given by

$$I(a) \;=\; I_0 \exp(-a/h), \tag{8}$$

where I_0 is the central surface brightness and h is the scale length.

```
FUNCTION Exponential
PA
ell
I_0
h
```

A.1.5 Exponential_GenEllipse

Similar to the Exponential function, but using generalized ellipses ("boxy" to "disky" shapes) instead of pure ellipses for the isophote shapes. Following Athanassoula et al. [1990], the shape of the elliptical isophotes is controlled by the `c0` parameter, such that a generalized ellipse with ellipticity $= 1 - b/a$ is described by

$$\left(\frac{|x|}{a}\right)^{c_0+2} + \left(\frac{|y|}{b}\right)^{c_0+2} \;=\; 1, \tag{9}$$

where $|x|$ and $|y|$ are distances from the ellipse center in the coordinate system aligned with the ellipse major axis (c_0 corresponds to $c - 2$ in the original formulation of Athanassoula et al). Thus, values of $c_0 < 0$ correspond to disky isophotes, while values > 0 describe boxy isophotes; $c_0 = 0$ corresponds to a perfect ellipse.

```
FUNCTION Exponential_GenEllipse
PA
ell
c0
I_0
h
```

A.1.6 Sersic

This is an elliptical 2D Sérsic function with the major-axis intensity profile given by

$$I(a) \;=\; I_e \, \exp\left\{-b_n \left[\left(\frac{a}{r_e}\right)^{1/n} - \,1\right]\right\}, \tag{10}$$

where I_e is the surface brightness at the effective (half-light) radius r_e and n is the Sérsic index controlling the shape of the intensity profile. The value of b_n is formally given by the solution to the transcendental equation

$$\Gamma(2n) \;=\; 2\gamma(2n, b_n), \tag{11}$$

where $\Gamma(a)$ is the gamma function and $\gamma(a, x)$ is the incomplete gamma function. However, in the current implementation b_n is calculated via the polynomial approximation of Ciotti & Bertin [1999] when $n > 0.36$ and the approximation of MacArthur, Courteau, & Holtzman [2003] when $n \leq 0.36$.

Note that the Sérsic function is equivalent to the de Vaucouleurs "$r^{1/4}$" profile when $n = 4$, to an exponential when $n = 1$, and to a Gaussian when $n = 0.5$.

```
FUNCTION Sersic
PA
ell
n
I_e
r_e
```

A.1.7 Sersic_GenEllipse

Similar to the Sersic function, but using generalized ellipses ("boxy" to "disky" shapes) instead of pure ellipses for the isophote shapes. See the discussion of the `Exponential_GenEllipse` function above for details of the isophote shapes.

```
FUNCTION Sersic_GenEllipse
PA
```

```
ell
c0
n
I_e
r_e
```

A.1.8 Core-Sersic

This generates an elliptical 2D function with the major-axis intensity profile given by the Core-Sérsic model [Graham et al., 2003, Trujillo et al., 2004]. This has a Sérsic profile (parameterized by n and r_e) for radii $>$ the break radius r_b and a single power law with index $-\gamma$ for radii $< r_b$. The transition between the two regimes is mediated by the parameter α: for low values of α, the transition is very gradual and smooth, while for high values of α the transition becomes very abrupt (a perfectly sharp transition can be approximated by setting α = some large number such as 100). The overall intensity scaling is set by I_b, the intensity at the break radius r_b.

```
FUNCTION Core-Sersic
PA
ell
n
I_b
r_e
r_b
alpha
gamma
```

A.1.9 BrokenExponential

Similar to Exponential, but with *two* exponential radial zones (with different scale-lengths) joined by a transition region at R_b of variable sharpness:

$$I(a) \;=\; S\, I_0\, e^{-\frac{a}{h_1}} \left[1 + e^{\alpha(a - R_b)}\right]^{\frac{1}{\alpha}\left(\frac{1}{h_1} - \frac{1}{h_2}\right)}, \tag{12}$$

where I_0 is the central intensity of the inner exponential, h_1 and h_2 are the inner and outer exponential scale lengths, R_b is the break radius, and α parameterizes the sharpness of the break. (See Erwin, Pohlen, & Beckman [2008].) Low values of α mean very smooth, gradual breaks, while high values correspond to abrupt transitions. S is a scaling factor, given by

$$S \;=\; \left(1 + e^{-\alpha R_b}\right)^{\frac{1}{\alpha}\left(\frac{1}{h_1} - \frac{1}{h_2}\right)}. \tag{13}$$

Note that the parameter α has units of length^{-1} (i.e., pixels^{-1}).

```
FUNCTION BrokenExponential
PA
ell
I_0
```

```
h1
h2
r_break
alpha
```

A.1.10 GaussianRing

An elliptical ring with a radial profile consisting of a Gaussian centered at $r = R_{\rm ring}$.

```
FUNCTION GaussianRing
PA
ell
A
R_ring
sigma_r
```

A.1.11 GaussianRing2Side

Similar to GaussianRing, but now using an asymmetric Gaussian (different values of σ for $r < R_{\rm ring}$ and $r > R_{\rm ring}$).

```
FUNCTION GaussianRing2Side
PA
ell
A
R_ring
sigma_r_in
sigma_r_out
```

A.1.12 EdgeOnDisk

This function provides the analytical form for a perfectly edge-on disk with a radial exponential profile, using the Bessel-function solution of van der Kruit & Searle [1981] for the radial profile and the generalized sech function of van der Kruit [1988] for the vertical profile.[15] The position angle parameter (PA) describes the angle of the disk major axis; there is no ellipticity parameter.

In a coordinate system aligned with the edge-on disk, r is the distance from the minor axis (parallel to the major axis) and z is the perpendicular direction, with $z = 0$ on the major axis. (The latter corresponds to height z from the galaxy midplane.) The intensity at (r, z) is given by

$$I(r,z) \;=\; \mu(0,0)\;(r/h)\;K_1(r/h)\;\;\mathrm{sech}^{2/n}(n\,z/(2\,z_0)) \tag{14}$$

[15] This model was used by Yoachim & Dalcanton [2006] for 2D modeling of thin and thick disks in edge-on galaxies, though typically with n fixed to values of 1 or 2.

where h is the exponential scale length in the disk plane, z_0 is the vertical scale height, and K_1 is the modified Bessel function. The central surface brightness $\mu(0,0)$ is given by

$$\mu(0,0) \; = \; 2\,h\,L_0, \tag{15}$$

where L_0 is the central luminosity *density* (see van der Kruit & Searle 1981). Note that L_0 is the actual parameter required by the function; $\mu(0,0)$ is calculated internally.

When $n = 1$, this becomes the familiar sech^2 model for the vertical distribution of a disk (with z_0 corresponding to $1/2$ of the z_0 in the original definition of van der Kruit & Searle [1981]). As $n \to \infty$, the vertical distribution approaches an exponential with $\exp(-z/z_0)$; in practice, the can be approximated by setting n equal to some fixed, large number.

Note that this particular function requires that the GNU Scientific Library (GSL) be installed; if the GSL is not installed, `imfit` should be compiled without this function. (The pre-compiled binary versions include the necessary code from the GSL.)

```
FUNCTION EdgeOnDisk
PA
L_0
h
n
z_0
```

A.1.13 EdgeOnRing

A simplistic model for an edge-on ring, using two offset components located at distance $\pm$`r` from the center of the function block. Each component (i.e., each side of the ring) is a symmetric Gaussian with size `sigma_r` for the radial profile and a symmetric Gaussian with size `sigma_z` for the vertical profile. (See GaussianRing3D for a similar component which does line-of-sight integration through a 3D luminosity-density model of a ring.)

```
FUNCTION EdgeOnRing
PA
I_0
r
sigma_r
sigma_z
```

A.1.14 EdgeOnRing2Side

Similar to EdgeOnRing, but now the radial profile for the two components is asymmetric: the inner ($|R| < R_{\mathrm{ring}}$) side of each component is a Gaussian with radial size `sigma_r_in`, while the outer side has radial size `sigma_r_out`.

```
FUNCTION EdgeOnRing2Side
PA
I_0
```

r
sigma_r_in
sigma_r_out
sigma_z

A.2 3D Functions

The following are experimental image functions which use line-of-sight integration through a 3D luminosity-density model to create a projected 2D image.

The functions are defined so as to have a primary plane (e.g., the equatorial plane for a disk galaxy); the orientation of this plane is defined by the PA and inc parameters, which specify the angle of the line of nodes (in degrees CCW with respect to the image +y axis) and the inclination to the line of sight (also in degrees), respectively. Thus, PA $= 0$ will align the line of nodes vertically, while PA $= 90$ will make it horizontal (parallel to the image x-axis).[16] The inclination is defined in the usual astronomical sense: $i = 0$ for a face-on system and $i = 90$ for an edge-on system.

For the GaussianRing3D function, which is not axisymmetric, there is an additional "position angle" parameter PA_ring, which defines the position of the ring's major axis *in the primary plane* (i.e., prior to any projection) with respect to the primary plane's +x axis. (The logic behind this is that when the primary plane's line of nodes is *horizontal* – i.e., PA $= 90$ – the orientation of the ring's major axis follows the usual orientation conventions with respect to the image +y axis. You are, of course, free to change this if you write 3D components of your own, though I will probably continue to follow it in the future.)

These functions use integration routines from the GNU Scientific Library (GSL); if the GSL is not installed, imfit should be compiled without them. (The pre-compiled binary versions include the necessary code from the GSL.)

A.2.1 ExponentialDisk3D

This function implements a 3D luminosity density model for an axisymmetric disk with an exponential radial profile and a $\mathrm{sech}^{2/n}$ vertical profile (as for the EdgeOnDisk function), using line-of-sight integration to create the projected surface-brightness profile for arbitrary inclinations.

In a cylindrical coordinate system (r, z) aligned with the disk (where the disk midplane has $z = 0$), the luminosity density at radius r from the central axis and at height z from the midplane is given by

$$j(r,z) \;=\; J_0 \;\exp(-r/h)\,\mathrm{sech}^{2/n}(n\,z/(2\,z_0)) \tag{16}$$

where h is the exponential scale length in the disk plane, z_0 is the vertical scale height, and J_0 is the central luminosity density.

[16] The goal is to ensure that the orientation of the component's line of nodes follows the same conventions as for the 2D functions, so that an inclined ExponentialDisk3D function with PA $= 30$ will have the same orientation as an elliptical 2D Exponential function with PA $= 30$.

When $n = 1$, the vertical distribution is the familiar sech^2 model (with z_0 corresponding to 1/2 of the z_0 in the original definition of van der Kruit & Searle [1981]). As $n \to \infty$, the vertical distribution approaches an exponential with $\exp(-z/z_0)$; in practice, the can be approximated by setting n equal to some fixed, large number.

```
FUNCTION ExponentialDisk3D
PA
inc
J_0
h
n
z_0
```

Because this function performs numerical integration for each pixel value, it will be slower than the analytic EdgeOnDisk function (though the latter is correct only in the $i = 90^\circ$ case), and even slower than the standard Exponential function.

A.2.2 GaussianRing3D

Similar to ExponentialDisk3D, this function does line-of-sight integration through an elliptical ring. The ring is defined as having luminosity density with a radial Gaussian profile (centered at `a_ring` along ring's major axis, with in-plane width σ) and a vertical exponential profile (with scale height `h_z`). The ring can be envisioned as residing in an (invisible) plane which has a line of nodes at angle `PA` and inclination `inc` (as for the ExponendialDisk3D function, above); within this plane, the ring's major axis is at position angle `PA_ring` *relative to the perpendicular to the line of nodes*, and the ring has an ellipticity given by `ell`.

```
FUNCTION GaussianRing3D
PA
inc
PA_ring
ell
J_0
a_ring
sigma
h_z
```

B Acknowledgments

Major inspirations for `Imfit` include both GALFIT [Peng et al., 2002, 2010] and BUDDA [de Souza, Gadotti, & dos Anjos, 2004, Gadotti, 2008].

Thanks also to Michael Opitsch and Michael Williams for being (partly unwitting) beta testers and for their feedback, to Martin Kuemmel for suggestion an improvement (and finding a related bug), to Roberto Saglia for urging me to implement the Core-Sérsic function, and to Maximilian Fabricius for suggesting improvements to the documentation.

B.1 Data Sources

Sample FITS images for demonstration and testing use are taken from Data Release 7 [Abazajian et al., 2009] of the Sloan Digital Sky Survey [York et al., 2000]. Funding for the creation and distribution of the SDSS Archive has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Aeronautics and Space Administration, the National Science Foundation, the U.S. Department of Energy, the Japanese Monbukagakusho, and the Max Planck Society. The SDSS Web site is `http://www.sdss.org/`.

The SDSS is managed by the Astrophysical Research Consortium (ARC) for the Participating Institutions. The Participating Institutions are The University of Chicago, Fermilab, the Institute for Advanced Study, the Japan Participation Group, The Johns Hopkins University, the Korean Scientist Group, Los Alamos National Laboratory, the Max-Planck-Institute for Astronomy (MPIA), the Max-Planck-Institute for Astrophysics (MPA), New Mexico State University, University of Pittsburgh, University of Portsmouth, Princeton University, the United States Naval Observatory, and the University of Washington.

B.2 Specific Software Acknowledgments

B.2.1 Minpack

This product includes software developed by the University of Chicago, as Operator of the Argonne National Laboratory.

References

Abazajian, K. N.., et al. 2009, "The Seventh Data Release of the Sloan Digital Sky Survey", *Astrophys.J. Supplement* **182**: 182

Athanassoula, E., Morin, S., Wozniak, H., Puy, D., Pierce, M. J., Lombard, J., & Bosma, A. 1990, *Monthly Notices of the Royal Astronomical Society* **245**: 130.

Ciotti, L., & Bertin, G. 1999, "Analytical properties of the $R^{1/m}$ law", *Astron. Astrophys.* **352**: 447.

de Souza, R. E., Gadotti, D. A., & dos Anjos, S. 2004, "BUDDA: A New Two-dimensional Bulge/Disk Decomposition Code for Detailed Structural Analysis of Galaxies", *Astrophys.J. Supplement* **153**: 411.

Erwin, P., Pohlen, B., & Beckman, J. E. 2008, "The Outer Disks of Early-Type Galaxies. I. Surface-Brightness Profiles of Barred Galaxies", *Astron.J.* **135**: 20.

Gadotti, D. A. 2008, "Image decomposition of barred galaxies and AGN hosts", *Monthly Notices of the Royal Astronomical Society* **384**: 420.

Graham, A., Erwin, P., Trujillo, I., & Asensio Ramos, A. 2003 "A New Empirical Model for the Structural Analysis of Early-Type Galaxies, and A Critical Review of the Nuker Model", *Astron.J.* **125**: 2951

Krist, J. 1995, "Simulation of HST PSFs using Tiny Tim", in *Astronomical Data Analysis Software and Systems IV*, R.A. Shaw, H.E. Payne, and J.J.E. Hayes, eds., *ASP Conference Series* **77**: 349.

MacArthur, L. A., Courteau, S., & Holtzman, J. A. 2003, "Structure of Disk-dominated Galaxies. I. Bulge/Disk Parameters, Simulations, and Secular Evolution", *Astrophys.J.* **582**: 689.

Peng, C. Y., Ho, L. C., Impey, C. D., & Rix, H. W. 2002, "Detailed Structural Decomposition of Galaxy Images", *Astron.J.* **124**: 266

Peng, C. Y., Ho, L. C., Impey, C. D., & Rix, H. W. 2010, "Detailed Decomposition of Galaxy Images. II. Beyond Axisymmetric Models", *Astron.J.* **139**: 2097

Sérsic, J.-L. 1968, *Atlas de Galaxias Australes* (Cordoba: Obs. Astron.)

Storn, R. and Price, K. 1997, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces", *Journal of Global Optimization* **11**: 314

Trujillo, I., Aguerri, J. A. L., Cepa, J., & Gutiérrez, C. M. 2001, "The effects of seeing on Sérsic profiles – II. The Moffat PSF", *Monthly Notices of the Royal Astronomical Society* **328**: 977.

Trujillo, I., Erwin, P., Asensio Ramos, A., & Graham, A. 2004, "Evidence for a New Elliptical-Galaxy Paradigm: Sérsic and Core Galaxies", *Astron.J.* **127**: 1917

van der Kruit, P. C., & Searle, L. 1981, "Surface Photometry of Edge-on Spiral Galaxies: I. A Model for the Three-dimensional Distribution of Light in Galactic Disks", *Astron. Astrophys.* **95**: 105

van der Kruit, P. 1988, "The Three-dimensional Distribution of Light and Mass in Disks of Spiral Galaxies", *Astron. Astrophys.* **192**: 117

Yoachim, P., & Dalconton, J. J. 2006, "Structural Parameters of Thin and Thick Disks in Edge-On Disk Galaxies", *Astron.J.* **131**: 226

York, D. G., et al. 2000, "The Sloan Digital Sky Survey: Technical Summary", *Astron.J.* **120**: 1579