StarFISH

(For Inferring Star-formation Histories)

User Manual

version 1.1, May 2004

Jason Harris and Dennis Zaritsky
jharris@as.arizona.edu
dzaritsky@as.arizona.edu

Contents

Chapter 1

Introduction

1.1 Overview

The **StarFISH** package is a suite of FORTRAN programs designed to determine the best-fit star formation history (SFH) of a stellar population. The package constructs a library of synthetic color-magnitude diagrams (CMDs) based on theoretical isochrones and data-derived determinations of the interstellar extinction, photometric errors, and distance modulus. These synthetic CMDs are combined linearly and statistically compared to observed photometry. When the best-fitting model is found, the amplitude coefficients on each synthetic CMD describe the star formation history of the observed population.

If you have any comments, corrections, suggestions, or complaints, please email:
jharris@as.arizona.edu

To use the package, you need:

- **Multicolor photometry** of a stellar population.
- A (preferably empirical) **photometric error model**, typically derived from artificial star tests.
- A **library of isochrones** that covers relevant ranges in age and metallicity.
- *a priori* determination of the **interstellar extinction** and **distance modulus**.
- Estimates (or guesses) of the **binary fraction** and **IMF slope**.

The package consists of three main programs, and five support programs:

• `mklib`: constructs a library of isochrone photometry from raw isochrones.

• `synth`: constructs synthetic CMDs from the isochrones, applying the effects of interstellar extinction, binarity, IMF, and photometric errors as appropriate for the data.

• `sfh`: performs a maximum-likelihood comparison of composite model CMDs and input data CMDs. The best-fit star formation history is output.

- `interp`: interpolates between adjacent isochrones, to improve resolution in age and/or metallicity. Requires knowledge of equivalent evolutionary points (EEPs) along each isochrone.

- `testpop`: constructs artificial stellar populations from the isochrones. `testpop` incorporates the photometric errors, extinction, etc. exactly as is done in `synth`. These populations can be input to `sfh` for testing purposes. (`testpop` is in the subdirectory of the same name)

- `repop`: same as `testpop`, except used to construct an artificial population from the SFH amplitudes found by `sfh`. This is useful for model/data comparisons. (`repop` is in the subdirectory of the same name)

- `testchi`: allows one to examine the parameter space of the fit by hand. SF amplitudes and other parameters can be interactively adjusted, and the resulting fit recomputed on the fly. (`testchi` is in the subdirectory of the same name)

- `mkimages.sh`: creates CMD images from the best-fit model. It also creates CMD maps of each region's contribution to the fitting statistic.

1.2 Getting Started

To begin, you need to obtain the program code from the Magellanic Clouds Photometric Survey website:
http://ngala.as.arizona.edu/mcsurvey/

As of this writing, the current version is 1.1. Note that there is a full version and a compact version. The full version includes a sample pre-generated synthetic CMD library for testing purposes. If you download the compact version, you can still generate the sample synthCMD library, but it will take some time.

The following command will unpack the directory tree of the StarFISH distribution. It will create a `SFH-1.1` subdirectory in the current directory.

```
% tar zxvf SFH-1.1.tar.gz
```
(or `tar zxvf SFH-1.1compact.tar.gz`)

or (on Solaris):

```
% gunzip SFH-1.1.tar.gz
% tar xvf SFH-1.1.tar
```

The package contents are shown in **Table 1**.

Table 1: Package Contents

File/Directory Name	Description
`manual/`	The StarFISH manual (this document)
`libcode/`	source code and Makefile for the `mklib` program
`synthcode/`	source code and Makefile for the `synth` program
`sfhcode/`	source code and Makefile for the `sfh` program
`commoncode/`	source code used by multiple programs
`interpcode/`	source code and Makefile for the `geteep` and `interp` programs
`input/`	input data files for `mklib`, `synth` and `sfh`
`/lib.dat`	sample `mklib` parameter file
`/synth.dat`	sample `synth` parameter file
`/sfh.dat`	sample `sfh` parameter file
`/iso.dat`	sample isochrone description files
`/iso.lock`	describes how isochrones are to be locked together
`/test.cmds`	sample synthetic CMD description file
`/mask.none`	file indicating CMD regions to ignore in the fit (default: null mask)
`/test.hold`	file indicating which SFH amplitudes should be held fixed (default: none held)
`/av/*.av`	sample extinction data files
`/crowd/*.crowd`	sample crowding table from artificial star tests
`/dtime.dat`	file describing age interval covered by each isochrone
`/geteep.dat`	input file for `geteep`
`/interp.dat`	input file for `interp`
`/testchi.dat`	input file for `testchi`
`testpop/`	files related to the `testpop` support program
`testchicode/`	files related to the `testchi` support program
`repop/`	files related to the `repop` support program
`grid/`	files related to the `grid` support scripts
`iso/`	raw isochrone photometry from Girardi et al. (2002, 2000) and Bertelli et al. (1994)
`testlib/`[1]	sample isochrone library produced by `mklib`
`test.syn/`[1]	sample synthetic CMDs based on artificial star tests from an image of NGC 1978.
`data/`	input data photometry goes here (NGC 1978 photometry included)
`output/`	`sfh` results and the `sfh.sm` plotting script

[1]Not present in the compact version of the program

1.3 Compiling the Principal Code

Each principal program (`mklib`, `synth`, and `sfh`) has its own Makefile, in its source code directory. To compile these programs, simply cd to the correct directory and type `make`:

```
% cd /path/to/SFH-1.1                          (if not already in the SFH directory)
% cd libcode
% make
% cd ../synthcode
% make
% cd ../sfhcode
% make
```

Each program will be copied into the `SFH-1.1` directory, which is where you should run them. You are now ready to experiment with the code.

1.4 A Quick Run

We will use all of the default parameter values to go from a freshly downloaded and compiled distribution to a successful first run of `sfh`. Steps labeled with an asterisk (*) may be skipped if you downloaded the full version of StarFISH, because this version already contains the pregenerated isochrone library and synthetic CMD library.

1. `% cd /path/to/SFH-1.1` (if not already in the SFH directory)

2. (*) Prepare isochrones: (see Section 2.1)
   ```
   % cd iso
   % tar zxvf girardi.tar.gz                  (unpack the isochrones)
   % gawk -f parse_iso.awk isocz*.dat         (parse isochrones)
   ```

3. (*) Generate isochrone library: (see Chapter 2)
   ```
   % cd ..
   % mkdir testlib          (create target directory before running mklib!)
   % ./mklib < input/lib.dat
   ```

4. (*) Generate Synthetic CMD library: (see Chapter 3)
   ```
   % mkdir test.syn         (create target directory before running synth!)
   % ./synth < input/synth.dat
   ```

5. Perform SFH Solution: (see Chapter 4)
   ```
   % ./sfh < input/sfh.dat
   ```

6. Plot solution:
   ```
   % cd output
   % sm                          (start supermongo plotting program)
   sm> macro read sfh.sm         (load sm script)
   sm> sfh c1978 c1978           (display SFH solution)
   ```

1.5 What's New in Version 1.1

There have been several improvements made since 1.0; the most important changes are listed here. See the files `ChangeLog` and `NEWSTUFF` for more details.

- Fixed random-number generator. We found a periodicity in the previous one.
- More robust handling of input files. The programs are now much less picky about the format of input files (*note that input files from previous versions of StarFISH are not compatible with this version*).
- Input files can now contain comment lines. Any line in an input file that begins with `#` or `*` will be ignored by the parser.
- Filenames in the input files can now be up to 40 characters long (previous maximum was 24 characters)
- Option to use Dolphin's "Poisson" fit statistic instead of the chi-squared statistic.
- The photometric error model used in `synth`, `testpop`, and `repop` now properly includes covariance between the color and magnitude errors.
- Option to use an analytic error model instead of an AST-derived crowding table in `synth`, `testpop`, and `repop`.
- `sfh` writes the current best solution to the logfile more frequently.
- New empirical isochrone interpolation code (`geteep` and `interp`)

Chapter 2

mklib

mklib is used for processing raw isochrone photometry into a format usable by synth. mklib applies a distance modulus, interpolates along the isochrone to a uniformly high resolution, and computes occupation probabilities (OP) for each isochrone point.

The package currently ships with the Padua isochrones, most recently published by Girardi et al. (2002). We found that the main sequence in these isochrones is too coarsely sampled for our purposes, resulting in artificially ”lumpy” synthetic CMDs. Therefore, mklib includes a photometric interpolation routine for points fainter than the main sequence turn-off (MSTO; defined for each isochrone in the isofile). If your isochrones do not require this interpolation, simply set the MSTO magnitude for each isochrone to be fainter than any isochrone point.

The mklib input file (named input/lib.dat by default) contains the parameters listed in Table 2 (the lines must be present in the listed order).

Table 2: mklib input file parameters

isofile	[string]	description of each isochrone: the input filename, the output filename, the age, and the MSTO magnitude.
faint	[real]	the faint magnitude limit for the output isochrone library. Should be several mag fainter than the data's faint limit.
dmag	[real]	photometric distance between adjacent interpolated points.
dmod	[real]	distance modulus
gamma	[real]	logarithmic IMF slope (Salpeter = -1.35)
nmag	[int]	number of magnitude values per isochrone point
mag0	[int]	which magnitude to check against **faint** and **msto**
iverb	[int]	verbosity flag (0=silent, 1=screen messages, 2=extra output files)

To use `mklib`:

1. **Create an isofile that describes your isochrones.** This file will also be used by `synth`. Each line in the file should contain the following space-delimited fields (the column format does not matter):

   `log(age)` [real number], `input raw isochrone file` [up to 40 chars], `output isochrone file` [up to 40 chars], `msto_mag` [real number]

   `msto_mag` is the absolute magnitude of each isochrone's MSTO point. If you do not know the MSTOs for your isochrones, you'll need to identify them. Use the fact that at the MSTO, the occupation probability changes dramatically.

2. **Preprocess the isochrone photometry files**. Each isochrone must be in its own file with a name matching the isochrone's entry in the `input raw isochrone file` list (column 2 of the isofile). The columns required for the isochrone files are:

   `mass`, `mag1`, `mag2`, `mag3`, ..., `magN`

   where N is the equal to the `nmag` parameter in `lib.dat`. Column format does not matter. The magnitudes should be expressed as absolute magnitudes; `mklib` will apply the distance modulus specified in `lib.dat` and output apparent magnitudes. For the default Padua isochrones, the input isochrone files are generated by the `parse_iso.awk` script. See the next section for details on the default isochrone-processing. Make sure the `mag0` parameter indicates the pass-band which should be used for comparing to the `faint` and `msto_mag` values.

3. **Create the target output directory, if necessary**. The output directory is specfied as part of the output filenames in the `isofile`.

4. **Run `mklib`:**
   ```
   % ./mklib < input/lib.dat
   ```

   Depending on the verbosity flag, `mklib` may provide screen messages regarding which isochrone it is working on. The output directory will contain the processed isochrones, with the following columns:

   `mag1`, `mag2`, `mag3`, ..., `magN`, `OP`, `mass`

   where the `magN` are the apparent magnitudes in each band, `OP` is the relative occupation probability associated with each point (determined from the IMF and from the relative duration of each evolutionary stage), and `mass` is the **initial** mass of each isochrone point, not including any mass-loss that may have occured since the star was formed.

2.1 Pre-processing the Padua Isochrones

The iso/ directory contains files related to the Padua isochrones, which we ship with StarFISH for convenience. You are free to use any other isochrone set that you wish, as long as it can be placed in the format that mklib requires (separate files for each isochrone, with the column format specified in step 2 above).

The Padua isochrones are disributed as a tar archive, which we include in StarFISH as the file iso/girardi.tar.gz. The archive contents are listed in Table 3.

Table 3: Padua isochrone package contents

isocz0[001,004,01,04,08,19,30].dat	Girardi et al. isochrones
isocsummz0[001,004,01,04,08,19,30].dat	Tables of Equivalent Evolutionary Points (EEPs)
read.me	Girardi's readme file

Each of the isochrone files isocz0*.dat contains isochrones of all ages for a given metallicity. StarFISH needs a separate file for each isochrone, so we provide the AWK script parse_iso.awk to divide each raw isochrone collection into individual files. We also provide the script parse_summ.awk to divide the EEP table collections into individual EEP tables for each isochrone.

If you downloaded the full version of StarFISH, then you do not need to run the parsing scripts; you already have the processed isochrone files, named iso/zNNNN_tt.tt (where NNNN is a 4-digit code for the metallicity, such as 0080; and tt.tt is the log of the age, such as 10.00). If you do not have these files, generate them like this:

```
% cd iso
% tar zxvf girardi.tar.gz
% gawk -f parse_iso.awk isocz*.dat
% gawk -f parse_summ.awk isocsummz*.dat
```

If you are satisfied with the age and metallicity resolution provided by the default Padua isochrones, then you are all done here. If you want to try to interpolate between isochrones to increase age or metallicity resolution, refer to the chapter discussing the interp support program.

Chapter 3

synth

`synth` constructs a synthetic CMD library from the isochrone library, incorporating the photometric effects of extinction, binarity, the IMF, and photometric errors. Each synthetic CMD is represented as a list of "pixel values", equal to the relative number of stars found at the corresponding subregion in the CMD. If the data contain more than two filters, then multiple CMDs can be created for each isochrone, so that all of the data can be utilized in the fit. For example, if your data include U, B, and V photometry, you might use `synth` to create two CMDs per isochrone: one with $U - B$ vs. B axes, and another with $B - V$ vs. V axes.

Some of the user-provided parameters (extinction, photometric errors, and binarity) are first encountered in the `synth` program. Others (the IMF and distance modulus) have already been used to create the isochrones output by `mklib`. Therefore, if you need to change the IMF or distance modulus you must run `mklib` first.

Depending on the age and metallicity resolution of your isochrone library, and on the quality of your photometric data, it is likely that some of the isochrones are photometrically degenerate. If `synth` is allowed to produce degenerate synthetic CMDs, then the SFH solution determined by `sfh` will be ambiguous, and will suffer from large correlated uncertainties among the SFH amplitudes. To avoid ambiguous SFH fits, `synth` can combine photometrically-degenerate isochrones together into the same synthetic CMD. This is referred to as **locking** the isochrones.

The synth input file (`input/synth.dat`) contains the parameters listed in Table 4. *Note that synth input files from previous versions of StarFISH will no longer work.*

Table 4: synth input file parameters

	Filenames	
`isofile`	[string]	description of the isochrones (can use the file from `lib.dat`)
`lockfile`	[string]	file describing the combination of degenerate isochrones
`hotfile`	[string]	list of extinction measurements of hot (young) stars
`coldfile`	[string]	list of extinction measurements of cool (old) stars
`crowd1`	[string]	crowding table from artificial star test
`crowd2`	[string]	output file for lookup table of binned delta-magnitudes
	CMDs	
`nmag`	[int]	same as nmag in `lib.dat`
`ncmd`	[int]	number of CMDs to construct per isochrone group
`mag0`	[int]	same as mag0 in `lib.dat`
`dpix`	[real]	size of CMD pixels (in magnitudes) *[NEW]*
	parameters for each of the CMDs	
`xeq`	[string]	filter equation for the x-dimension
`yeq`	[string]	filter equation for the y-dimension
`xmin`	[real]	smallest allowable x-value
`xmax`	[real]	largest allowable x-value
`ymin`	[real]	smallest allowable y-value
`ycmax`	[real]	largest allowable y-value, for crowding table stars
`ypmax`	[real]	largest allowable y-value, for data/model stars
`suffix`	[string]	filename suffix for the synthetic CMD output files
	Crowding	
`dbinx`	[real]	width of crowding bins, in magnitudes
`dbiny`	[real]	height of crowding bins, in magnitudes
`emin`	[real]	minimum delta-magnitude cutoff
`emax`	[real]	maximum delta-magnitude cutoff
`dbin`	[real]	binsize of delta-magnitude histograms
	Reddening parameters	
`red(i)`	[real]	extinction in mag(i) (relative to the filter in which extinction is measured)
	Run-time parameters	
`verb`	[int]	verbosity flag (higher N = more output)
`interp_err`	[flag][2]	1 = interpolate errors between bracketing crowding bins *[NEW]*
`fake_errs`	[flag][2]	1 = use analytic error model instead of crowding table *[NEW]*
`nscale`	[int]	number of model stars in each CMD
`seed`	[int]	random seed value
`mass1`	[real]	minimum mass (for OP normalization)
`mass2`	[real]	maximum mass (for OP normalization)
`gamma`	[real]	logarithmic IMF slope (Salpeter = -1.35)
`faint`	[real]	faint limit for generated model stars
`fbinary`	[real]	binary fraction

[2] A flag is an int whose value can either be 0 or 1.

To use synth:

1. **Construct the lockfile.** This file describes how the isochrones are to be locked together into groups. Again, this is necessary if isochrones of similar age are photometrically degenerate, to avoid ambiguous SFH fits. The columns of the lockfile are as follows (*note that lockfiles from previous versions of StarFISH will no longer work*):

   group_id[int], isoname[up to 40 characters], synthfilestem [up to 40 characters]

   group_id identifies the locked group to which the isochrone belongs. The actual group_id values do not matter; when the value of one line is different from that of the previous line, a new synthetic CMD is started.

   isoname is the isochrone filename. It is only used to check against the filename in the isofile. If they don't match, an error is triggered; there must be a one-to-one correspondence of the isochrones in the isofile and the lockfile.

   synthfilestem is the filename stem for the output synthetic CMD pixel file. The output filenames are constructed by concatenating the CMD suffixes specified in synth.dat onto synthfilestem (these suffixes are typically the combination of magnitudes; for example a $U - B$ vs. B CMD might have a suffix of ".ub").

2. **Construct the extinction files (hotfile, coldfile).** The extinction files each contain a single column: extinction values, A_λ, measured in the magnitude whose reddening parameter in synth.dat is set to 1.00 (in the provided synth.dat, red(3) is set to 1.00; this is the V band, so the extinction files list A_V values). There are two files (hotfile and coldfile) to account for the different extinction properties of young and old stellar populations. If you do not require population-dependent extinction, simply point hotfile and coldfile to the same file. If you do not require extinction modeling at all, point hotfile and coldfile to the null extinction file, input/av/zero.av.

3. **Construct the crowding table (crowd1).** This is most likely constructed from artificial star tests. The crowding table contains:

   ra, dec, mag1, dmag1, mag2, dmag2, ..., magN, dmagN

   column format doesn't matter. The crowding table should contain all artificial stars injected into your data images, even if they were not detected in your photometric pipeline. In fact, these non-recovered "dropout" stars are extremely important, because synth uses them to determine the completeness rate as a function of photometric position in the CMDs. For bands in which an artificial star was detected, the listed mag value is its **recovered** magnitude, and dmag is (mag(recovered) - mag(input)). For bands in which an artificial star dropped out, mag is the star's **input** magnitude, and dmag is set to 9.99 in order to flag it as a dropout in that band.

If you decide not to perform artificial star tests on your images (although such tests are *highly* recommended), you can either generate a fake crowding table by hand, based on your estimates of the scatter and completeness rates in each band, or you can set the `fake_errs` parameter in `synth.dat` to 1, which will use a hueristic analytic error model instead of the empirical crowding table. In this case, you will need to modify the code in `synthcode/fakecrowd.f` to generate photometric errors and completeness rates appropriate for your data. The comments in that file will guide you in making these changes.

4. Decide how many CMDs you would like, and what their axes will be. You specify the CMD axes with the `xeq(icmd)` and `yeq(icmd)` string variables in `synth.dat`. These strings indicate the filter combination for each CMD axis. For example, you have VRI photometry, and you are using V=1, R=2, I=3. Your first CMD could be: $V - I$ vs. V, so `xeq(1)` = 1-3, `yeq(1)` = 1. Your other "CMD" could be a two-color diagram: $V - R$ vs. $V - I$, so `xeq(2)` = 1-2, `yeq(2)` = 1-3. The strings can be either the sum or the difference of two filter indices, or one index by itself. Do not put whitespace in the equations; each must be a single "word".

   You will also need to specify photometric limits for each CMD (parameters `xmin`, `xmax`, `ymin`, `ycmax`, and `ypmax`). Note that instead of one `ymax`, there are two variables: `ycmax` and `ypmax`. `ycmax` is the faint limit for the crowding bins, and `ypmax` is the faint limit for the CMD itself. `ycmax` should be at least 2 or 3 mag fainter than `ypmax`, because very faint stars can be blended and detected at much brighter magnitudes than their true magnitude.

   Also specify the size of CMD "pixels" in the output synthetic CMD files (`dpix`), in magnitudes.

   Finally, specify a suffix string that will be concatenated with `synthfilestem` to construct the output file name for each synthetic CMD. The suffices can be up to 8 characters long, and typically indicate something about the CMD axes. For example, a $B - V$ vs. V CMD might have a suffix "`.bv`".

5. Specify the size of the crowding bins in each direction (`dbinx` and `dbiny`). Artificial stars in the crowding table are binned in the CMDs so that the brightness- and color-dependence of the photometric errors can be accounted for. Independent Δmag histograms will be constructed from the artificial stars present in each bin.

   Specify the limits for the delta-mag histograms with `emin` and `emax`. Any Δmag value in the crowding table outside these limits is treated as a photometric dropout.

   The `dbin` parameter sets the width of the bins in the delta-mag histograms (*this parameter used to determine the size of the synthetic CMD pixels as well, but there is now a separate parameter for that: dpix*).

6. Specify the reddening law to use, by setting the relative extinction for each band: `red(imag)`. The band for which you have measured extinction values in `hotfile` and `coldfile` should have `red(imag)` = 1.0. The rest of the values should be the extinction in that band, assuming the extinction in the reference band is 1.0. For

example, you have BV photometry, and measured A_V extinction values. $A_V = 1.0$. If we assume that $A_V/E(B\text{-}V) = 3.2 = A_V/(A_B - A_V)$, then $A_B = 1.3125$.

7. Specify miscellaneous parameters. `nscale` sets the number of model stars to include in each CMD. `nscale` should be much larger than the number of stars in your data CMDs, so that the Poisson noise in the models is much smaller than in the data. `iseed` is the random seed value; it can be any integer. `mass1`, `mass2`, and `gamma` are IMF parameters used for normalizing the pixel values. `synth` will only attempt to add model stars brighter than the `faint` parameter. This is an efficiency feature: without this cutoff, the code would waste a lot of time adding undetectable stars. Set `faint` to just fainter than where your completeness rate reaches 0%.

8. Once the parameter file is complete, and the other input files are ready, you can finally run `synth`. It will take quite a bit of time to complete, depending on computing power and the number of isochrones. I found it usually took several hours to process 150 isochrones on a 700 MHz linux box.

You can examine the synthetic CMDs as images, using either the IRAF `rtext` command, or the `mkgif.bat` and `pxl2ppm.awk` scripts I provide in the `commoncode` directory.

Chapter 4

sfh

sfh performs a “chi-squared-like” minimization comparison between data photometry and a linear combination of the synthetic CMDs. Each synthetic CMD is modulated by an **amplitude** that is proportional to the number of stars with the age and metallicity of the synthetic CMD’s parent isochrone(s). These amplitudes form the multidimensional parameter space of the minimization.

I say “chi-squared-like” because you may actually choose among three statistics to determine the quality of the fit: chi-squared, Lorentzian, or Poisson. The three parameters weight outliers slightly differently, but they are otherwise quite similar. Hereafter, I will simply refer to the fitting statistic as “chi-squared”.

The minimization analysis employs the downhill simplex “amoeba” algorithm as described in Numerical Recipes (Press *et al.*; hereafter NR). We have written our own implementation of this algorithm. It works by sampling a small region of the parameter space and determining the local chi-squared gradient. It then takes a small step in the down-gradient direction, and determines the local chi-squared gradient around this new location. The process iterates until a (nearly) zero gradient is found, signalling that a minimum has been found.

The amoeba algorithm in NR has some safeguards against finding local minima, but we found that it was still prone to getting stuck. The simplex by definition can only sample parameter space along directions parallel to the N axes of the parameter space; it is blind to off-axis directions. However, as it nears the minimum, lower chi-squared values will tend to be found when a change in one parameter is complemented by a change in one or more other parameters. In other words, downward gradients will tend to lie along off-axis directions.

To remedy this problem with the pure simplex method, we add a random-search loop, which is triggered whenever the simplex claims it has found a minimum. In the random-search loop, we select a random direction in the multidimensional parameter space, and take a small step along that direction. If the step yields a smaller chi-squared value, we continue stepping along that direction until we reach a step that no longer decreases chi-squared. This is repeated for a large number of random directions, and the parameter space location which yielded the lowest overall chi-squared value is recorded. The simplex is then restarted at this new parameter space location.

This iteration between the downhill simplex algorithm and the random-search loop is repeated until the random-search loop cannot find a direction which yields a lower chi-squared value. We then accept the last simplex location as the true minimum, and output the best-fit SFH amplitudes.

The `sfh` input file (`input/sfh.dat`) contains the following parameters:

	Filenames	
`datpre`	[string]	input data file prefix
`cmdfile`	[string]	Sample synthetic CMDs description file
`maskfile`	[string]	Allows one to ignore specified regions of the CMDs
`holdfile`	[string]	file w/ SFH amplitudes that should be held fixed
`outfile`	[string]	contains final best-fit SFH amplitudes
`logfile`	[string]	file w/ current best SFH (useful for monitoring, and for restarting an interrupted run)
`plgfile`	[string]	log of all tested parameter locations (this file will be very large; not used by default: `iplg`=0)
`chifile`	[string]	output file containing grid populations and chi-squared values
	Synthetic CMD parameters	
`nfree`	[int]	number of independent isochrone amplitudes
`ncmd`	[int]	number of CMDs over which to calculate chi-squared
`npix`	[int]	binning factor for CMDs
`dpix`	[real]	size of synthetic CMD pixels (same as in `synth.dat`)
	Parameters for each CMD	
`suffix(i)`	[string]	CMD filename suffix (same as in `synth.dat`)
`xmin(i)`	[real]	calculate chi-squared within these CMD limits
`xmax(i)`	[real]	calculate chi-squared within these CMD limits
`ymin(i)`	[real]	calculate chi-squared within these CMD limits
`ymax(i)`	[real]	calculate chi-squared within these CMD limits
`nbox(i)`	[int]	number of grid boxes within region defined by above limits
	Runtime parameters	
`seed`	[int]	random seed value
`fit_stat`	[int]	0 = use chi-squared; 1 = use Lorentzian; 2 = Use Poisson
`uselog`	[flag]	1 = start searching at logged position in logfile
`iplg`	[flag]	1 = generate plgfile (warning: this file will be very large!)
`gtype`	[flag]	grid definition flag (0 = uniform grid; 1 = custom grid)
`iverb`	[int]	verbosity flag (larger N = more screen output)
`lambda`	[real]	initial size of simplex
`conf`	[real]	confidence interval for computing errorbars (e.g., 1 sigma = 0.68)
`cthr`	[real]	if ($\Delta\chi^2 >$ cthr), consider new location an improvement
`ptol`	[real]	if ($\|p\_new - p\_old\| > ptol$), consider new location different
`ftol`	[real]	if simplex's chi-squared values span less than ftol, signal a zero gradient (i.e., a minimum)
`nvec`	[int]	number of parameter space directions in random-search loop
`ntry`	[int]	number of iterations for determining errorbars

To use `sfh`:

1. **construct the `cmdfile`**. The file's columns are:

   `z_metal`, `log_age`, `filenamestem`

   Column format does not matter. Note that some lines in the `cmdfile` may represent multiple isochrones. In these cases, just choose a representative age/metallicity for the group. These values are only used to identify the amplitudes in the output file. The `filenamestem` needs to point to the synthetic CMD files generated with `synth`.

2. **Decide on a CMD gridding strategy**. `sfh` works by binning the CMDs and comparing the number of data and model stars in each bin. The default is to uniformly bin the CMDs in each dimension with a binsize of `npix`×`dpix`. If you want to use a custom grid instead, write a subroutine to assign gridbox numbers to each CMD pixel, add a call to your subroutine in `grid.f`, and set `gtype` to 1 in `sfh.dat`. You must also add your custom-grid source code files to `sfhcode/Makefile`. An example custom grid can be reviewed in the files `ubbgrid.f`, `bvvgrid.f`, and `viigrid.f`. These grids were constructed for our LMC data, and are fine in densely populated regions of the CMDs and coarse in sparsely populated regions.

   The boxes in each CMD's grid are identified by a single number, the box number. Thus the distribution of stars in the CMDs can be expressed as a two-dimensional array whose indices identify the CMD and the gridbox within that CMD: `nstars(icmd, ibox)`. `icmd` refers to which CMD you are considering, `ibox` is the number of the box in that CMD. Indicate the total number of boxes in each CMD with the `nbox(icmd)` parameter.

3. **Construct the `maskfile`**. the file's columns are:

   `icmd`, `ibox`, `maskflag`

   Column format does not matter. if `maskflag`=1, the stars in (`icmd`, `ibox`) will be excluded from the determination of the fit.

4. **Preprocess the data photometry**. You need to have a separate photometry file for each CMD. The filenames need to be (`datpre` + `sffx(icmd)`), where `datpre` and `sffx(icmd)` are specified in `input/sfh.dat`. The first column in each CMD file is the x-dimension magnitude, and the second column is the y-dimension magnitude. Other columns are ignored, but could be used for (ra, dec) coordinates or other useful information. Note that the `sffx(icmd)` values need to be the same as the `suffix(icmd)` values in `synth.dat`, and that the CMD limits (`xmin`, `xmax`, `ymin`, `ymax`) need to be the same also.

5. **Adjust the sfh runtime parameters.** The default values should work well, but feel free to tweak them.

6. **Run** `sfh`. Depending on the value of `iverb`, it may provide screen output on its progress (status messages, current lowest chi-squared value). Generally, larger values of `iverb` produces more screen output. `iverb`=0 will print no screen messages; `iverb`=3 provides "live" updates of the convergence process. If the run gets interrupted for some reason, you can resume at the last logged position by setting `uselog`=1 (assuming the `logfile` exists).

7. **Evaluate the goodness-of-fit.** When the run is complete, it will print out the lowest chi-squared value found. Divide this by (`nboxes`-`nfree`) to get the reduced chi-squared, which should probably be less than ~ 10 for a good fit.

   To examine the SFH solution, you can use the `sfh.sm` supermongo script (in the `output/` subdirectory). All of the customizable parameters for this script are collected at the top of the file. See the comments in the file for guidance in adjusting these settings.

   If there is a problem with the SFH fit (e.g., high chi-squared value), you can use the following diagnostics (each of these programs has its own Chapter in this manual):

   (a) the `repop` program produces artificial stellar photometry based on the best-fit amplitudes, to be compared to the data CMDs.

   (b) the `mkimages.sh` script in the `grid/` subdirectory, which converts the gridbox populations in the `chifile` into images, and embeds them into an HTML document for easy comparison.

   (c) the `testchi` program allows you to tweak the SFH amplitudes and parameter values. It then calculates a new chi-squared value for the new model.

Chapter 5

geteep and interp

geteep and interp are used to interpolate between isochrones to increase the nominal matallicity and/or age resolution. To compile the programs, cd to the interpcode directory and type make. This will create the geteep and interp programs in the root SFH-1.1 directory.

geteep identifies empirical "equivalent evolutionary points" (EEPs) in each isochrone. True EEPs are stellar evolution events which can be identified in the isochrones, such as "the main sequence turn-off" or "the tip of the red giant branch". These are very useful for interpolating between isochrones. In fact, the older Padua isochrones provide EEP tables for this purpose, but the new isochrones published in 2002 did not have this information (StarFISH includes the older EEP tables, but we don't make use of them since they are missing for some isochrones).

Since we don't have EEP tables for all of the isochrones, we need a way to identify EEPs (or something like them), which is what geteep does. It identifies empirical EEPs (eEEPs) as local extrema in color or brightness, along the isochrone track in one of the CMDs. Many of these points correspond to "true" EEPs (e.g., the TRGB is a local brightness maximum). The point is to identify points which represent the same evolutionary state in two similar isochrones; we feel that the color/brightness extrema do this well.

The geteep input file has a two-line header. The first line contains an integer equal to the number of magnitudes representing each point in the isochrones. The second line contains an integer identifying the magnitude which will be the vertical "brightness" axis in the target CMD (in which eEEPs will be identified). The horizontal axis is automatically set to be the color index (mag(i-1) - mag(i)), so don't choose "1" for the vertical axis. The remainder of the file lists the isochrone files to be processed, and the list of output isochrones in which eEEPs have been identified.

Once you have run geteep to identify and label the eEEPs in each isochrone, you can run interp to interpolate between the isochrones. The interp input file has three header lines at the top which identify the number of magnitudes, the vertical-axis magnitude, and the fractional distance between the bracketing isochrones for the target interpolated isochrones. For example, if the bracketing isochrones have metallicity 0.004 and 0.008, and the target isochrones are to have metallicity 0.005, the fractional distance should be 0.25. (This assumes linear variation with metallicity, which is probably Ok over small changes in metallicity).

The rest of the `interp` input file is the list of isochrones to be interpolated. In each row, column 1 is the first bracketing isochrone, column 2 is the second bracketing isochrone, and column 3 is the target isochrone. Again, there is no strict column format; each filename can be up to 40 characters long.

Chapter 6

testpop

testpop constructs artificial stellar photometry based on a user-specified star formation history. It is basically a different front-end to the synth code base. Build testpop by typing make in the testpop directory. testpop reads a (probably modified) version of the synth input file, which must be named testpop/synth.dat. It also reads its own input parameter file, which is specified on the command line:

```
% ./testpop < input_file
```

The input file should contain the following 8 header lines:

pre	[string]	a prefix for the output photometry files (up to 8 characters)
lockflag	[flag]	1 = use locked amplitudes (use lockfile in synth.dat)
nstars	[int]	an amp of 1.0 will draw nstars masses from the isochrone
dmod	[real]	delta-distance modulus. For adjusting the DM of the isochrones
fext	[real]	additional extinction multiplier
gamma	[real]	IMF slope (testpop recalculates OPs)
fbinary	[real]	Binary fraction
sfrflag	[flag]	1 = amps are expressed as $M_\odot/yr$; 0 = amps are Nstars/bin

The values for gamma and fbinary supercede the values in synth.dat. Immediately following these 8 parameters, the input file should contain N lines with the following columns:

Amp_{SFH}, Z_{metal}, $log(age)$, num_{iso}

There should be one line for each isochrone in the isofile (if lockflag=0), or one line for each independent isochrone group (if lockflag=1). testpop determines the number of stars to generate for isochrone i according to the formula:

$$N(i) = nstars \times Amp(i) \times frac(i) \times \Delta t(i)/num_{iso}(i)$$

where: $frac(i)$ is the fraction of the full mass range that is represented on the isochrone ($frac(i)$ decreases with age, as more stars have evolved to their non-luminous end-states); $\Delta t(i)$ is the duration of the age bin, in Gyr; and $num_{iso}(i)$ is the number of isochrones in the current isochrone group. If sfrflag=0, then $\Delta t(i) = 1.0$ for all i. If lockflag=0, then $num_{iso}(i) = 1$ for all i.

Be aware that, in general, many more stars are generated than actually appear in the CMDs, because a large fraction of the generated stars will be too faint to detect.

`testpop` outputs `ncmd` output photometry files, each containing the following data:

mag_x, mag_y, Δ_x, Δ_y

These files may be used directly as input data to the `sfh` program.

Chapter 7

repop

`repop` is nearly identical to `testpop`. The only difference is that the input SFH amplitudes are generated from the SFH amplitudes output by `sfh`. `repop` constructs an artificial stellar population based on the best-fit SFH solution, so that it can be compared to the input population.

Before using `repop`, you must first generate the input SFH amplitudes file using `mkinput.awk`:

```
% cd repop
% gawk -f mkinput.awk -v pre=file_prefix sfh_output_file
```

You may need to modify this script to make it suitable to your data. Specifically, the number of isochrone groups and number of isochrones per group are hard-coded, as are the metallicity strings.

Now you can actually run `repop`, specifying the new input file on the command line:

```
% repop < file_prefix.input
```

The number of stars constructed by `repop` is probably different from the number of data stars. To generate the same number of stars, follow these steps:

1. use `wc` to count the number of stars in one of repop's output CMD files, and the corresponding input data CMD file.
2. multiply `nscale` (in the `repop` input file) by N(data)/N(repop)
3. run `repop` again

The `repop.sm` script can be used to display the CMDs of the generated population. Again, you may need to modify this script. This script as written displays the repopulated B-V,V CMD alongside the original B-V,V CMD for easy comparison. It takes two arguments, the prefixes of the repopulated and original photometry files.

Chapter 8

testchi

testchi provides an interactive front-end to the fitstat.f subroutine, which calculates the fitting statistic of a given model SFH. Compile testchi by typing make in the testchi directory (the executable is placed in the root SFH-1.1 directory).

The testchi input file is a simplified version of the sfh input file, with several parameters removed. The testchi input file contains the following parameters:

Filenames		
datpre	[string]	input data file prefix
cmdfile	[string]	Sample synthetic CMDs description file
maskfile	[string]	Allows one to ignore specified regions of the CMDs
holdfile	[string]	file w/ SFH amplitudes that should be held fixed
ampfile	[string]	the input model SFH amplitudes to test
chifile	[string]	output file containing grid populations and chi-squared values
Synthetic CMD parameters		
nfree	[int]	number of independent isochrone amplitudes
ncmd	[int]	number of CMDs over which to calculate chi-squared
npix	[int]	binning factor for CMDs
dpix	[real]	size of synthetic CMD pixels (same as in synth.dat)
Parameters for each CMD		
suffix(i)	[string]	CMD filename suffix (same as in synth.dat)
xmin(i)	[real]	calculate chi-squared within these CMD limits
xmax(i)	[real]	calculate chi-squared within these CMD limits
ymin(i)	[real]	calculate chi-squared within these CMD limits
ymax(i)	[real]	calculate chi-squared within these CMD limits
nbox(i)	[int]	number of grid boxes within region defined by above limits
Runtime parameters		
fit_stat	[int]	0 = use chi-squared; 1 = use Lorentzian; 2 = Use Poisson
gtype	[flag]	grid definition flag (0 = uniform grid; 1 = custom grid)
iverb	[int]	verbosity flag (larger N = more screen output)

To use testchi, adjust the parameters in testchi.dat (or create a new input file) and run the program:

```
% ./testchi < input/testchi.dat
```

The chi-squared value resulting from a comparison of the data and the model described by the outfile is printed to the screen, and the gridbox populations are written to the chifile.

testchi is very useful for quickly seeing changes in the chi-squared value when the SFH amplitudes (or other parameters) are changed. You can examine the distribution of chi-squared in the CMDs using the chifile, and the mkimages.sh script.

Chapter 9

mkimages.sh

The `grid` subdirectory contains the script `mkimages.sh`, which generates images of the CMDs from the grid box populations in the `chifile`, and generates an HTML document which displays the images in a grid. The HTML document provides a very convenient way to examine the quality of the model fit, and to diagnose what may be causing a poor fit.

The script requires files named `grid/gridN.index` which contains a description of the grid boxes in each of your CMDs (each CMD has its own index file). The columns are:

`ix`, `iy`, `ibox`

where (`ix`, `iy`) are the synthetic CMD pixel coordinates, and `ibox` is the grid box to which that pixel belongs. You can modify the program `mkgrid.f` to generate the gridN.index files for your CMDs.

Once you have valid `gridN.index` files, you can use the `mkimages.sh` script to generate the images and HTML page. Before running the script, there are several parameters which should be adjusted; see the "CUSTOMIZATION" section of the `mkimages.sh` file.

When running the script, you specify a filename prefix for the HTML document and images, and the name of the `chifile` on the command line:

```
% ./mkimages.sh c1978 ../output/c1978.chi
```

The script will generate 4 images per CMD, which will be placed in the `images/` subdirectory. The images represent the grid box populations of the model and of the data; the difference between model and data, and the contribution of each grid box to the fitting statistic. You can view all of the images using the HTML document that is created by the script.

Bibliography

[Bertelli et al. (1994)] Bertelli, G., Bressan, A., Chiosi, C., Fagotto, F., & Nasi, E. 1994, *A&AS*, 106, 275

[Girardi et al. (2000)] Girardi, L., Bressan, A., Bertelli, G., & Chiosi, C. 2000, *A&AS*, 141, 371