ES: Elementary Supernova Spectrum Synthesis User Guide

Table of Contents

1 Programs

- SYN++ --- Consider this a rewrite of the original SYNOW [1] code in modern C++. It has a few further enhancements, a new structured input control file format, and the atomic data files have been repackaged and are more complete than what SYNOW has.
- SYNAPPS --- This code uses the same underlying library code used to build SYN++ to implement a spectrum synthesis calculation within the objective function of a parallel optimization framework. So, SYNAPPS works like an automated SYN++. The idea is for SYNAPPS to do the fitting work so a supernova spectroscopist can do thinking.

2 Software Dependencies

If you only want to use SYN++, then there is only one dependency. If you want to use SYNAPPS, there are three more, and you have to be able to compile and run parallel applications via MPI.

Note that if you use Mac OS X, all of the packages listed below can be obtained quite easily using MacPorts (http://www.macports.org/). For Linux users, consult your package management software (like "aptitude" if you use Ubuntu). Sorry Windows users: You are on your own here.

- CFITSIO (http://heasarc.gsfc.nasa.gov/fitsio/) FITS I/O library. Required for building SYN++ and SYNAPPS. The atomic line list files are stored in FITS format, eliminating a support issue SYNOW had with byte rotation.
- BLAS (http://www.netlib.org/blas/) Basic Linear Algebra Subprograms. Required for building SYNAPPS, but not SYN++.
- LAPACK (http://www.netlib.org/lapack/) Linear Algebra Package. Required for building SYNAPPS, but not SYN++.
- APPSPACK-5.0.1-C3 (https://software.sandia.gov/appspack/downloads/appspack-5.0.1-C3.tar.gz) Asynchronous Parallel Pattern Search, special Computational Cosmology Center version for use with SYNAPPS. Required for building SYNAPPS, but not SYN++. If you visit the APPSPACK [2] website, be sure you get the version of APPSPACK from the "other versions" page that says "special version only for use with SYNAPPS". Otherwise you will have a harder time building it. Note also that BLAS/LAPACK have to be built first.
- MPI (for example, http://www.open-mpi.org/) Message Passing Interface. You need to have MPI compilers and libraries installed in order to use APPSPACK.
- OpenMP (http://openmp.org/wp/) Shared-memory parallel programming. This is an optional dependency for both SYN++ and SYNAPPS. Note that if you want to use SYNAPPS in hybrid parallel mode (with both MPI and OpenMP) that you may need to study your "mpirun" (or equivalent) command line to ensure you get partitioning right.

3 Downloads

- Source code is hosted on GitHub [3] at http://github.com/rcthomas/es/
- The latest distribution version is available at http://github.com/rcthomas/es/downloads
- The atomic line data is available at http://c3.lbl.gov/es/es-data.tar.gz

4 Installation

First, follow the instructions for compiling the other software dependencies, or install them using package manager software like MacPorts or aptitude, depending on your system.

When you install APPSPACK, consult the "platforms" subdirectory there. You may find a platform installation script that is suitable to your situation. Importantly, if you specify a --prefix=$PREFIX argument to configure for APPSPACK, keep track of what value you used for $PREFIX, you'll need it later.

Once the dependencies (including APPSPACK-5.0.1-C3) are installed, you can proceed with ES. It ought to look something like this on a Unix-like system. Suppose you downloaded version 1.00 of ES. Then:

```
$ tar zxvf es-1.00.tar.gz
$ cd es-1.00
$ ./configure
$ make
$ make install
```

You may need to pass options to configure (see "./configure -h" for the full list you can use). Most commonly one will need to use the --prefix flag. If you do not have super-user privileges, you could do something like:

```
$ ./configure --prefix=$HOME/local
```

Thus, the "make install" command will deposit the ES executables and example control files into the installation path. No libraries or header files are installed. If you use the above example, then you get this structure in your file system when all is said and done:

```
$HOME/
    local/
        bin/
            syn++
            synapps
            ...
        share/
            es/
                syn++.yaml
                synapps.yaml
```

An example. Most supernova researchers I know work on Mac OS X. I like to use MacPorts to install my Unix-like packages. This works alright for me on Mac OS X Snow Leopard with MacPorts:

```
./configure
    CXX=g++-mp-4.4 CC=gcc-mp-4.4 F77=gfortran-mp-4.4
    --prefix=$HOME/local
    --with-cfitsio-cpp="-I/opt/local/include"
    --with-cfitsio-libs="-L/opt/local/lib -lcfitsio"
    --with-appspack-cpp="-I$HOME/local/include"
    --with-appspack-libs="-L$HOME/local/lib -lappspack"
```

Note that the above example is included in the script located at "platforms/osx_macports.sh" and you can just run that if you have the same set up. Note that the example assumes that APPSPACK was installed at "--prefix=$HOME/local." Exactly what you do depends largely on taste.

Ted Kisner deserves thanks for helping with the ES build system.

5 Atomic Lines

ES makes use of atomic line data files. These are available in the format ES needs on the web (see the Downloads section). They are bundled into a single tar.gz file. Once downloaded, unpack them. You can put them anywhere, but I personally like to place them alongside the sample control files that come with ES. Following the example above where ES is installed into $HOME/local, I would do:

```
$ mv es-data.tar.gz $HOME/local/share/es
$ cd $HOME/local/share/es
$ tar zxvf es-data.tar.gz
$ mv es-data/lines .
$ mv es-data/refs.dat .
```

Then, in the control files for the ES executables, I set:

```
opacity :
    line_dir : /Users/rthomas/local/share/es/lines
    ref_file : /Users/rthomas/local/share/es/refs.dat
```

6 Running SYN++

The "syn++" executable has only one required argument, the name of the YAML [4] control file:

```
$ syn++ syn++.yaml
```

This will compute one or more synthetic spectra and write them to standard output. If more than one spectrum is output, they are separated by a blank line. The format is multi-column ASCII, with the first two columns being wavelength and flux. Sending the result to a file is done by redirect, for example:

```
$ syn++ syn++.yaml > my_spectrum.dat
```

And one can redirect the output to a process like xmgrace [5]:

```
$ syn++ syn++.yaml | xmgrace observed_spectrum.dat -
```

If you run "syn++" from within a scripting language like Python or Ruby, you can capture the output via pipe (no temporary files) and do something interesting with it. Maybe you could automate writing your papers this way.

There are some command-line options.

- The "--verbose" option provides a counter (written to standard error) so you know that SYN++ is doing something and how it's progressing.
- The "--wl-from=SPECTRUM" option makes SYN++ read in the specified spectrum file (three column ASCII format) and use the first column as the wavelength output grid. Especially handy if you need the wavelengths of the spectrum you are fitting to match up with the synthetic one for subtracting.

If you've compiled with OpenMP enabled, you can use multi-core computing to improve SYN++ performance. Don't expect it to scale without bound, I only parallelized part of the code (the slowest part). To take advantage of it, you need to set OMP_NUM_THREADS to something like the number of cores you want to use. If you have a quad-core Opteron, you can do something like:

```
$ export OMP_NUM_THREADS=4
$ syn++ syn++.yaml > my_spectrum.dat
```

It should run faster than with OMP_NUM_THREADS=1. Your default value may already equal the number of cores available on your machine.

A sample SYN++ control file is distributed with the code. After the code is installed, it should be found under the "share/es" directory of the installation, with the name "syn++.yaml." Copy it somewhere and open it up. Note that the ordering of the sections of the control file is not really important, but each variable belonging to a section must be listed under the section header. That is, you cannot put "min_wl" into the "source" section; it goes under "output."

Now, let's go over the example control file, section by section. Note that by convention, all wavelength quantities in the SYN++ and SYNAPPS control files are in Angstroms, all temperatures are in 10^3 K, and all velocities are in 10^3 km/s.

The "output" section controls the wavelength grid of the synthetic spectrum:

```
output :
    min_wl      : 2500.0        # minimum wavelength in AA
    max_wl      : 10000.0       # maximum wavelength in AA
    wl_step     : 5.0           # wavelength spacing in AA
```

Note that the "--wl-from" command line option will override this section. Each of the variables should be easy enough to understand.

The "grid" section controls the velocity, line opacity, and line source function grids:

```
grid :
    bin_width   : 0.3           # opacity bin size in kkm/s
    v_size      : 100           # size of line-forming region grid
    v_outer_max : 30.0          # fastest ejecta velocity in kkm/s
```

Most of the time you will not mess with any of these variables. The most important one to know about is "v_outer_max," which is set to a value at least as big as any "v_outer" you are going to specify in any of the setups. This setting establishes the wavelength limits of the atomic line list to be loaded for all setups. You might think that the output wavelength limits were enough to establish this, but actually more lines in the blue and red are needed to establishe the source function and compute the full spectrum.

The next section is the "opacity" section:

```
opacity :
    line_dir    : /usr/local/share/es/lines     # path to atomic line data
    ref_file    : /usr/local/share/es/refs.dat  # path to ref. line data
    form        : exp           # parameterization (only exp for now)
    v_ref       : 10.0          # reference velocity for parameterization
    log_tau_min : -2.0          # opacity threshold
```

The first two variables contain the path to the atomic line list directory on disk. The "form" variable should always be set to "exp" until we add new parameterizations; it is the same for all opacity profiles in all setups. The value of "v_ref" is the reference velocity for all opacity profiles, they are scaled to the value of "log_tau" at this velocity (given by each profile in each setup). Bins with integrated opacity (over all lines in a bin) smaller than "log_tau_min" are ignored in the calculation of the source function and spectrum.

Next is the "source" section, which is very simple:

```
source :
    mu_size     : 10            # number of angles for source integration
```

Again, you don't really need to mess with this. However, you might find a performance boost if you are using OpenMP and this number is set to a multiple of the number of cores available.

And next is "spectrum" which controls how the output spectrum is calculated:

```
spectrum :
    p_size      : 60            # number of phot. impact parameters for spectrum
    flatten     : No            # divide out continuum or not
```

First is the number of impact parameter rays subtending the photosphere as viewed in projection from infinity. More rays are added going out to "v_outer" as needed to integrate the projected flux. The "flatten" option computes the spectrum without the underlying thermal continuum or any warping parameters.

Each synthetic spectrum computation is governed by a "setup." Multiple setups can be placed into a SYN++ YAML control file. They are simply expressed as YAML lists: Each setup is preceded on its first line by a "-" character. This is useful, since if different setups all use the same ions, the line list is only loaded once per run of SYN++, not per setup. Even if the list of ions changes from setup to setup, those ions in common between subsequent setups will not be dumped and re-loaded. Here's the start of the "setups" section and the first setup:

```
setups :
    -   a0      :  1.0          # constant term
        a1      :  0.0          # linear warp term
        a2      :  0.0          # quadratic warp term
        v_phot  :   8.0         # velocity at photosphere (kkm/s)
        v_outer :  30.0         # outer velocity of line forming region (kkm/s)
        t_phot  :  12.0         # blackbody photosphere temperature (kK)
        ions    :  [ 1601, 2201, 2401, 2601 ]   # ions (100*Z+I, I=0 is neutral)
        active  :  [  Yes,  Yes,  Yes,  Yes ]   # actually use the ion or not
        log_tau :  [  0.1,  1.0,  1.0,  1.0 ]   # ref. line opacity at v_ref
        v_min   :  [ 10.0, 10.0, 10.0, 10.0 ]   # lower cutoff (kkm/s)
        v_max   :  [ 30.0, 30.0, 30.0, 30.0 ]   # upper cutoff (kkm/s)
        aux     :  [  1.0, 10.0, 10.0, 10.0 ]   # e-folding for exp form
        temp    :  [ 10.0, 10.0, 10.0, 10.0 ]   # Boltzmann exc. temp. (kK)
```

The parameters "a0," "a1," and "a2" are the coefficients of a quadratic warping function that can be multiplied by the synthetic spectrum once it is computed. This can be helpful if the target spectrum has a low-frequency trend that SYN++ cannot replicate (a sharp photosphere blackbody and no electron scattering has its limitations).

Parameters "v_phot" and "v_outer" are the inner and outer boundaries of the line-forming region, respectively. Remember that no "v_outer" may exceed "v_outer_max" in any of the setups. Parameter "t_phot" is the Blackbody temperature of the photospheric lower boundary of the line-forming region.

The next two variables, "ions" and "active" are not really parameters, but rather ion labels for each opacity profile, and a boolean value for whether or not to actually turn the opacity profile on or off. Being able to do this when iteratively fitting can be handy. The format of the ion species is 100 * atomic number + ionization state, with an ionization state of "0" being neutral.

The last five variables control each opacity profile. The opacity profiles are listed from left to right, in columns. The same ion may be listed more than once, but there are precedence rules for opacity profiles with repeated ions. First, each successive profile which repeats an ion gets precedence to set the reference opacities between its specified values of "v_min" and "v_max." Second, the rightmost temperature listed for an ion wins. These two rules also hold for SYNOW, and though arbitrary we have kept to them.

The opacity profile parameters in more detail. Parameter "log_tau" is the base-10 logarithm of the reference line opacity at the value of "v_ref" set in the "opacity" section. Parameters "v_min" and "v_max" are the limits in velocity of the opacity profile, but "v_phot" and "v_outer" set hard limits. The "aux" parameter is a generic name for whatever the opacity "form" needs it to be. For an "exp" form, it is the e-folding length of the opacity profile. Finally, "temp" is the Boltzmann excitation temperature for parameterizing line strengths.

7 Running SYNAPPS

The "synapps" executable has only one required argument, the name of the YAML control file:

```
$ mpirun -np 16 synapps synapps.yaml
```

Your command line may look different, depending on MPI. While SYNAPPS runs it channels APPSPACK, which is set to be quite chatty, and when it starts up it will output a bunch of diagnostics about the problem. Then it will start solving, and you can watch as each new best minimum of the objective function is found. When it's done, it will output a summary.

If you have enabled OpenMP and want to use OpenMP to increase performance of the objective function, you need to hierarchically distribute your cores. This should not be too hard, but it may vary from system to system:

```
$ export OMP_NUM_THREADS=2
$ mpirun -npernode 4 synapps synapps.yaml
```

Here we specify that each worker gets 2 OpenMP threads with which to compute the objective function, but there are 4 MPI tasks per node. The above example is for a case where there are 8 cores total per node.

You will likely want to capture this output into a log file, because it can come in handy later. For example, in bash:

```
$ mpirun -np 16 synapps synapps.yaml | tee synapps.log
```

The tee command will let you watch the progreess of the code but also save it to the "synapps.log" file.

Let's examine the "synapps.yaml" example YAML control file that came with ES, it should be under "share/es" in the install tree. The sections "grid," "opacity," "source," and "spectrum" are the same as with SYN++, so we skip those. Also, note the absence of the "output" section. This is because the wavelength grid for the output will be determined by the target spectrum being fit. There are two new sections, "evaluator" and "config" which I discuss below.

For "evaluator," things look like this:

```
evaluator :
    target_file : "target.dat"  # spectrum to fit (format: wl, flux, flux_error)
    vector_norm : 2             # objective function norm
    regions     :
        apply   : [   Yes,   No,  Yes ] # fit this wavelength region or not
        weight  : [     0,    1,    1 ] # weight the region by this number
        lower   : [     0, 3900, 5600 ] # min. wavelength for region definition
        upper   : [ 10000, 4100, 6400 ] # max. wavelength for region definition
```

This section controls how the objective function is evaluated. This depends on the target file being fit, what vector norm is being used, and the regions of the target spectrum to be fit.

The target file is a three-column ASCII input file with columns wavelength, flux, and flux error; note the flux error is not the variance, it should be like standard error. Of course the spectrum must be in rest-frame: There are better codes out there for determining the redshift and phase of your spectrum! [6]

The "vector_norm" parameter is the exponent of an L-norm specification. So, a value of 1 is like absolute value, a value of 2 is like a chi-squared, etc. Note that the residuals that go into the calculation are all weighted by the standard error.

The "regions" sub-section specifies whether there are parts of the target spectrum to be ignored (for telluric lines, or simply there is known to be no line opacity to fit in certain parts). The regions are listed in columns, and if no regions are listed then the entire spectrum is fit. The way regions work is that they are applied from left to right by logical "or" if they are applied. In the above example, first we mask out everything between a wavelength of 0 and 10000 (that is, weight it by 0). The next region is skipped (apply=No) altogether. The last region is applied, activating SYNAPPS to fit the region between 5600 and 6400 AA.

To discuss the "config" section, we break it up into parts. The top of the "config" section looks like:

```
config :
    fit_file    : "target.fit"      # when done, put answer here
    cache_file  : "target.cache"    # evaluated point cache
```

Pretty straightforward, the final best fit will be output to the file given by "fit_file" and the name of the cache input/output file is given by "cache_file."

Next comes the configuration of scalar parameters:

```
a0      : { fixed:  No, start:    1, lower:   0, upper: 10, scale: 10 }
a1      : { fixed:  No, start: -2.6, lower: -10, upper: 10, scale: 20 }
a2      : { fixed:  No, start: -5.0, lower: -10, upper: 10, scale: 20 }
v_phot  : { fixed:  No, start: 10.7, lower:   5, upper: 15, scale: 10 }
v_outer : { fixed: Yes, start:   30, lower:  15, upper: 30, scale:  1 }
t_phot  : { fixed:  No, start: 11.4, lower:   5, upper: 25, scale: 20 }
```

These parameters should be recognizable from the discussion of the SYN++ YAML control file. Each parameter has 5 specifications for SYNAPPS. First is "fixed," which is "Yes" if the parameter is to be left at the "start" value and "No" if you want SYNAPPS to fit for it. Keyword "start" is the initial value for the parameter. The "lower" and "upper" keywords give the lower and upper bounds constraints for the parameter, and the "scale" keyword basically gives SYNAPPS a way to rescale the parameter in a numerically convenient way (think of it as a normalizing factor). In the above example, "v_outer" will be held fixed to 30,000 km/s, but all the other parameters will be fit for by SYNAPPS.

Next after the scalar parameters, we see some lists. These are not of parameters, but are switches and labels that go with each of the opacity profiles that follow:

```
ions        :  [ 1401, 1100, ]
active      :  [  Yes,   No, ]
detach      :  [   No,   No, ]
```

The "ions" labels and "active" switches should be familiar from the SYN++ YAML control file discussion. The "detach" switch is used to disengage a linear equality constraint that fixes "v_min" of the opacity profile to the value of "v_phot." Disengaging this constraint will "detach" the profile and allow "v_min" and "v_phot" to be fit independently. To be clear: Setting it to "no" forces "v_min" to be exactly equal to "v_phot." If you set detach to "no" then be sure the "start" value of "v_min" equals "v_phot" or you will get an "infeasible point" crash.

Finally we discuss one of the opacity profile parameters only ("log_tau"), the extension to the other parameters is trivial:

```
log_tau     :
    fixed   :  [   No,   No, ]
    start   :  [ 1.14,    0, ]
    lower   :  [   -2,   -2, ]
    upper   :  [    2,    2, ]
    scale   :  [    1,    1, ]
```

Each opacity profile has parameters "log_tau," "v_min," "v_max," "aux," and "temp." Each of these in turn has the 5 keywords to specify, as in the case of the scalar parameters. They have the same function as in that case.

In all cases, be sure that the "start" value is between the "lower" and "upper" bounds constraints or you will get an infeasible point error.

8 Python Code Included

There are a couple scripts and modules that I wrote for managing the YAML control files. One of them takes a SYNAPPS YAML control file and the SYNAPPS log and turns the last best fit into a SYN++ YAML control file. You can then turn ions on or off or tweak the fit if you like, or think you can do better than SYNAPPS.

9 Citing SYN++ or SYNAPPS

A code description paper is in the works. When it is published, please cite that. Until then, please make some form of acknowledgement that you used this code.

10 Frequently Asked Questions

10.1 Is a precompiled binary version available?

Sorry, no can do. APPSPACK is licensed under LGPL, and ES is licensed under BSD. I cannot link APPSPACK into ES and distribute the product as a binary. If you have problems compiling, please don't hesitate to ask for help.

10.2 How can I get a PDF copy of this manual?

You can create it yourself from the README using rst2pdf, or you can get a copy of it here: http://c3.lbl.gov/es/es_manual.pdf.

10.3 Why does SYNAPPS crash right away, with an "infeasible point" message?

You may have specified in your control file a starting point that is considered infeasible. That is, you might have a variable set outside its bounds constraints.

Another way this could happen is if you are re-using a previously written cache file, but the number of parameters cached is different. Also, be aware that if you re-use a cache file and do something like swap one ion out for another, the cache will be invalidated and you cannot re-use it with the new run.

These errors are not that helpful, but they are at least thrown as exceptions we could catch and handle. It's on the roadmap to work on this issue.

10.4 Why doesn't SYNAPPS figure out the ions for me?

SYNAPPS uses an optimizer which does not handle categorical variables. Using categorical variables would be one way to potentially solve the problem. We're looking at other optimizers all the time. Stay tuned.

Another approach would be to work out a bunch of heuristics for it, but that'd all be pre-processing and wouldn't have to be part of SYNAPPS. It's an interesting problem.

10.5 How do I stop SYNAPPS and still get the fit output?

The answer is sort of at an Easter-egg level. I would like to think of a better way to do this. The first step is to run the "ps" command to find out the process ID's of all the MPI tasks running synapps. Then, take the first synapps process ID (NOT the "mpirun" one) that is listed in numerically ascending order. Then send it a TERM signal like:

```
$ kill -15 PID
```

If you did it right, SYNAPPS will catch the signal and tell all the workers to shut down. The master will run an evaluation of the most recent best fit it has and write it out to the file requested in the "config" section of the YAML control file.

On most batch queue systems you don't get enough time to write out before the KILL signal is sent, and I think that goes to the process leader (not the SYNAPPS master process).

10.6 Why use YAML for control file input?

SYN++ and SYNAPPS use structured input files. The older SYNOW code uses a Fortran feature called "namelist input" which can map values into program variables at runtime. I know of nothing similar that is just built into C++ that could compete (let alone go farther). The easiest solution is to flatten the input into a sequence of unique key key-value pairs, but this produces rather large control files. More complicated solutions involve writing your own parser, which I believe is best left to the professionals.

So, early on I experimented with embedding the Lua [7] interpreter into the code. This worked rather well, and raised a numer of very interesting possibilities for user customization. But, I found that it would be very useful to be able to have a code serialize an internal state, and then use that serialized state as input to another (or the same) code. Lua doesn't do this. I also had a hard time convincing myself people would learn Lua (but really, check it out, especially if you are a developer). I also doubted people would want to mess with XML, I wanted it to be easier to read.

YAML [8] seemed the best option. It handles structure intuitively by indenting and blocking. YAML documents can be created or ingested by programs written in many languages, and we include some Python scripts that help users manage these. The only thing it's lacking is that its output is not completely deterministically formatted, so some of the Python included in ES is dedicated to just formatting it nicely. One day we may set up a web-service where spectra could be fit using cloud computing (I am not joking here) and YAML is easily converted to JSON, so it might be an additional advantage.

I hope that the choice of YAML isn't alienating and that people can get used to the idea of indentation for structuring their input. It's common in Python, so people do seem to be able to get used to it.

10.7 What does "ES" mean?

ES stands for "Elementary Supernova," a term used by Jeffery and Branch (1990) [9] to describe a conceptual model that serves as the basis for a number of computer codes used in the analysis of supernova spectra. See Branch, Baron, and Jeffery (2003) [10] and references therein to learn more.

SYNOW is the classic example ES code. Originally written by David Branch, and then updated and modernized (in 1993) by Adam Fisher [11] at the University of Oklahoma, SYNOW has become a widely

used tool among supernova scientists. Learning to use SYNOW is a rite of passage for many budding supernova spectroscopists.

11 Author

ES was developed by R. C. Thomas at the Computational Cosmology Center, Lawrence Berkeley Lab.

12 License

ES is distributed under the terms of the BSD software license. See the LICENSE file for further details and contact information.

13 Copyright

---

1 http://www.nhn.ou.edu/~parrent/synow.html
2 https://software.sandia.gov/appspack/version5.0/index.html
3 http://github.com/
4 http://en.wikipedia.org/wiki/YAML
5 http://plasma-gate.weizmann.ac.il/Grace/
6 http://adsabs.harvard.edu/abs/2007ApJ...666.1024B
7 http://www.lua.org/
8 http://www.yaml.org/
9 http://adsabs.harvard.edu/abs/1990sjws.conf..149J
10 http://adsabs.harvard.edu/abs/2003LNP...598...47B
11 http://adsabs.harvard.edu/abs/2000PhDT.....6F