SNEWPY:
a python code to process supernova simulation data through SNOwGLoBES

Segev Benzvi[1], Tomer Goldhagen[2], Spencer Griswold[1], Anne Graf[3], Jim Kneller[3], Evan O'Connor[4], Navya Uberoi[1], Arkin Worlikar[5]

[1] *University of Rochester, Rochester, NY, USA*
[1] *University of North Carolina - Chapel Hill, Chapel Hill, NC, USA*
[1] *NC State University, Raleigh, NC, US*
[1] *Stockholm University, Stockholm, Sweden*
[1] *Raleigh Charter High School, Raleigh, NC, US*

October 7, 2020

1 Introduction

`SNEWPY` is a software package written in python which forms a three-part data pipeline between supernova simulations and neutrino detector signals. The `SNEWPY` code is designed to interface with the `SNOwGLoBES` code which will also need to be installed by the user. `SNOwGLoBES` is available at `https://github.com/SNOwGLoBES/snowglobes`.

Before we describe the three parts of the pipeline, it is easiest to understand `SNEWPY` if we give the contents of the master file `SNEWPY.py`.

```
#!/usr/bin/env python3

from argparse import ArgumentParser

import to_snowglobes
import run_snowglobes
import from_snowglobes
import tarfile

snowglobes_path = "/path/to/snowglobes/installation/folder/"
model_path = '/path/to/folder/with/simulation/data/'
model_file = 'nakazato-LS220-BH-z0.004-s30.0.fits'
```

```
model_type = 'Nakazato_2013'
transformation_type = 'AdiabtaticMSW_NMO'
output_filename = None
ntbins =  None
deltat = None

theta12 = 33.
theta13 = 9.
theta23 = 45.
m3=None         # in eV
tau3=None       # in s

d=10            # in kpc

mixing_parameters = [theta12,theta13,theta23]
decay_parameters = [m3,tau3]

transformation_parameters = mixing_parameters

tarball = to_snowglobes.generate(model_path, model_file, model_type,
    transformation_type, transformation_parameters, d,
    output_filename, ntbins, deltat)

run_snowglobes.go(snowglobes_path, model_path, tarball)

from_snowglobes.collate(snowglobes_path, model_path, tarball)
```

The first things SNEWPY does is load three modules: to_snowglobes, run_snowglobes, and from_snowglobes. SNEWPY will handle a lot of compressed files in various formats so it also needs the Tarball package. SNEWPY then defines three strings: the path to the SNOwGLoBES installation directory, the path to the folder containing the simulation the user wishes to use, and the name of the supernova model file. In this example we wish to process the model known as nakazato-LS220-BH-z0.004-s30.fits which is a fits formatted file. This model is one of a number of simulations run by Nakazato et al [1] published in a paper written in 2013. As such, it is a Nakazato_2013 model type. Other model types will be described below. The simulation will be processed in combination with a prescription for the flavor transformation through the mantle of the supernova. In this example we are using the AdiabaticMSW_NMO prescription which stands for adiabatic transformation due to the MSW effect using a normal mass ordering. We then provide three variables which allow the user to give the output tarball file a particular name rather than the default name based upon the name of the input, then the user can either supply the number of time bins to be generated (the default is 30) or the time interval between the bins (in which case the number of snapshots is determined by the duration of the simulation).

The AdiabaticMSW_NMO flavor prescription requires the user to specify the vacuum neu-

trino mixing angles often called θ_{12}, θ_{13}, and θ_{23}. These are bundled into a list. Other flavor prescriptions depend upon other parameters. We have given an example for neutrino decay which requires one to specify a mass for the heaviest neutrino and its decay lifetime. Whatever mixing prescription is chosen, all the parameters are collated into a list called `transformation_parameters`. The last parameter that must be defined is the distance to the supernova which is given in units of kpc.

Once the parameters for the flavor transformation have been specified, `SNEWPY` is ready to send data through the pipeline. The first part of the pipeline is to call the function `to_snowglobes.generate` with the previously defined paths, filename etc. The output from the function is the name of the output tarball file the function created. By default this tarball is placed in the same directory as the model file.

The next call is to `run_snowglobes.go` which requires the path to the snowglobes working directory, the path to the tarball created by `to_snowglobes`, and the name of the tarball file with all the snapshots. There is nothing returned by `run_snowglobes.go`. This function unpacks the tarball created by `to_snowglobes.generate` and then runs each snapshot in the tarball through `SNOwGLoBES` for all the different detector configurations `SNOwGLoBES` can calculate. The output of the calculation by `SNOwGLoBES` is placed in the usual /out folder in the snowglobes working directory. `run_snowglobes.go` will then clean up the files it unpacked.

Finally, the third part of the pipeline is `from_snowglobes.collate` which takes the same arguments as `run_snowglobes.go`. This function will collate all the data it finds in the snowglobes working directory /out into the observable channels for each detector. For each detector and each snapshot in time it creates four files, one for each of the four combinations of 'weighted' and 'unweighted' event rates and 'smeared' and 'unsmeared' detector response. Each file will have several columns: the energy and then a column for each of the observable channels. Simple figures of these data files are made using `matplotlib` plotting routines. The files and figures for all the different detectors and snapshots are then packed into a single tarball which is placed into the same folder from which the original simulation data was taken.

Both `run_snowglobes.go` and `from_snowglobes.collate` take an optional fourth argument, the name of a detector you wish to run. If the fourth argument is not supplied, the pipeline will run all the detectors. Below is an example of the inputs to these functions, if the user decided to run the Halo 1 detector:

```
run_snowglobes.go(SNOwGLoBES_path, models_dir, outfile, "halo1")

from_snowglobes.collate(SNOwGLoBES_path, models_dir, outfile, "halo1")
```

The fourth argument must be the same in both functions except for wc100kt30prct and ar40kt. For wc100kt30prct: `run_snowglobes.go` takes in "wc100kt30prct", whereas `from_snowglobes.collate` takes in "wc100kt30prct_events". For ar40kt: `run_snowglobes.go` takes in "ar40kt", while `from_snowglobes.collate` takes in "ar40kt_events".

In what follows we describe in further detail the various model data provided, how each part of `SNEWPY` functions, the options for each component, and what changes can be made.

2 The models

At the present time we have data for hundreds of core-collapse simulations organized into six classes according to the publication describing the simulations. The six classes are

- Bollig_2015
- Janka
- OConnor_2015
- Nakazato_2013
- Sukhbold_2015
- Warren_2020

In each of these folders one can find a README.md file describing the contents of the folders and the publication(s) describing the simulations. In addition we have data from two pair-instability supernovae (PISNe) and two Type-Ia supernovae. These data sets are very different from the core-collapse simulations because a) flavor transformation has already been included and b) the data is already in `SNOwGLoBES` format. The distance to the PISN and Type Ia supernova models was taken to be 10 kpc. In addition, in the Type Ia data sets there are multiple lines of sight.

3 `to_snowglobes`

The purpose of `to_snowglobes` is to take raw supernova simulation data such as that mentioned above, apply a prescription for the flavor transformation that occurred through the mantle of the supernova and accounting for decoherence, and create a set of files packed into a tarball which can be run through SNOwGLoBES. `to_snowglobes` is the part of the code with the most flexibility. The user may change the supernova model fed through the pipeline, the flavor transformation prescription and the parameters used in the prescription, the distance to the supernova, and the number of snapshots produced from the simulation.

The set of available flavor transformations are found in `flavor_transformation.py`. They are:

- NoOscillations - no oscillations.
- AdiabaticMSW_NMO - adiabatic neutrino evolution in the normal mass ordering.
- AdiabaticMSW_IMO - adiabatic neutrino evolution in the inverted mass ordering.
- NonAdiabaticMSWH_NMO - the H resonance mixing the neutrino states 2 and 3 is nonadiabatic, the L resonance is adiabatic.
- NonAdiabaticMSWH_IMO - the H resonance mixing the antineutrino matter states 1 and 3 is completely nonadiabatic, the L resonance is adiabatic.

- TwoFlavorDecoeherence - 50% mixing between the electron neutrinos and the x flavor neutrinos, 50% mixing between the electron antineutrinos and the x flavor antineutrinos.

- ThreeFlavorDecoeherence - 33% mixing between the electron neutrinos and the x flavor neutrinos, 33% mixing between the electron antineutrinos and the x flavor antineutrinos.

- NeutrinoDecay_NMO - Adiabatic evolution through the mantle of the supernova according to the normal mass ordering followed by decay of the heaviest mass neutrino state to the lightest in the vacuum. Note we use the approximation that the energy of the neutrino does not change.

- NeutrinoDecay_IMO - Adiabatic evolution through the mantle of the supernova according to the inverted mass ordering followed by decay of the heaviest mass neutrino state to the lightest in the vacuum. Note again that we use the approximation that the energy of the neutrino does not change.

Each of these classes provies 8 functions: `Pee`, `Pex`, `Pxe`, `Pxx`, `Peebar`, `Pexbar`, `Pxebar`, and `Pxxbar` which, for forwards compatibility, accept two arguments though they are not used except in the case of the two `NeutrinoDecay` classes. The formulae are given at the end of this document. We do not demonstrate here how the formulae were derived: a brief discussion can be found in [2] and a more complete derivation will be provided in a separate document. Some of these flavor transformation prescriptions depend upon parameters. The list of parameters for the flavor transformations are passed to `to_snowglobes` and will be passed again to the flavor transformation class when instantiated.

Before we discuss the models classes, we need to mention to the user that we use a 'Flavor' class to key any array that returns a quantity which has a 'flavor' dimension. The four keys are: nu_e, nu_e_bar, nu_x, nu_x_bar. In this way the user never needs to remember the ordering of the entries in an array if it has a flavor dimension. For example, if we want the electron neutrino element from an array of luminosities assigned to a variable called L, we write L[Flavor.nu_e].

For the six general core-collapse simulation categories listed in section 1, a python class has been written that is capable of reading the data of a file of that type. The six models classes, which are found in `models.py`, exactly match the names of the folders in /models and are

- Bollig_2015,

- Janka,

- OConnor_2015,

- Nakazato_2013,

- Sukhbold_2015,

- Warren_2020.

Each class provides several functions with which the user can access the data from the simulation. The three functions which every model class provides are

- `get_time` - return the array or snapshot times, in units of ms, recorded in the simulation.

- `get_initialspectra(t,E)` - returns a four element array of the neutrino spectra at the neutrinosphere in units of (erg/s) for a given snapshot time t and neutrino energy E. There are just four elements because most supernova simulations do not distinguish between the μ and τ flavors, reporting instead the flavor x which is μ or τ. In some simulations the μ and τ antineutrinos are a separate flavor $\bar{x}$.

- `get_oscillatedspectra(t,E)` - returns a four element array of the neutrino spectra in units of (erg/s) for a given snapshot time t and neutrino energy E after a flavor prescription has been applied. The effect of flux dilution due to distance to the supernova is not included. Again, the value for a particular flavor in the array is accessed by using the keys provided by the Flavor class e.g. [Flavor.nu_x]. The oscillated spectra Φ'_{ν_α} are related to the initial spectra Φ and Φ_{ν_x} by the following equations:

$$\begin{aligned}
\Phi'_{\nu_e} &= P_{ee}\Phi_{\nu_e} + P_{ex}\Phi_{\nu_x} && (1)\\
\Phi'_{\nu_x} &= P_{xe}\Phi_{\nu_e} + P_{xx}\Phi_{\nu_x} && (2)\\
\Phi'_{\bar{\nu}_e} &= \bar{P}_{ee}\bar{\Phi}_{\nu_e} + \bar{P}_{ex}\bar{\Phi}_{\nu_x} && (3)\\
\Phi'_{\bar{\nu}_x} &= \bar{P}_{xe}\bar{\Phi}_{\nu_e} + \bar{P}_{xx}\bar{\Phi}_{\nu_x}. && (4)
\end{aligned}$$

The quantities P_{ee}, P_{ex} etc. are the eight functions provided by the flavor transformation model mentioned earlier. In the code they are passed the snapshot time t and energy E but at the present time, these arguments are not used for anything. In future versions of `SNEWPY` we plan to include time and energy dependence.

In addition to these three functions, the supernova model classes also provide additional functions for accessing information about the simulation. These functions include

- `get_revival_time` - returns the time in units of ms at which the shock was revived.

- `get_mean_energy(flavor)` - returns an array of mean energies for each flavor in units of MeV at the snapshot times recorded by the simulation for a given neutrino flavor.

- `get_meansq_energy(flavor)` - returns an array of mean square energies for each flavor in units of MeV at the snapshot times recorded by the simulation for a given neutrino flavor.

- `get_rootmeansq_energy(flavor)` - returns an array of square root of the mean square energies for each flavor in units of MeV^2 at the snapshot times recorded by the simulation for a given neutrino flavor.

- `get_luminosity(flavor)` - returns an array of luminosities in units of erg/s for each flavor at the snapshot times recorded by the simulation for a given neutrino flavor.

- `get_pinch_param(flavor)` - returns a four element array of the pinch parameters for each flavor at the snapshot times recorded by the simulation for a given neutrino flavor.
- `get_EOS` - returns a string which indicates the equation of state used in the simulation.
- `get_progenitor_mass` - returns the ZAMS mass of the progenitor in units of solar masses.

Not all model classes provide all of these functions - what is available depends upon the contents of the data files. Some of these additional functions are invoked by `get_initialspectra` and `get_oscillatedspectra` so are essential to the code, others are provided to the user in case they prove useful.

The two PISN models and two TypeIa data sets are different. Neutrino oscillations have already been applied and the data is already in `SNOwGLoBES` format. For these data sets there is is no need to call `to_snowglobes`. A class in models.py has been provided to interface with data. The class is called SNOwGLoBES and has just one function, `get_fluence(t)` with t the time.

Now that `models.py` and `flavor_transformation.py` have been described, we can go through the function `to_snowglobes.generate`. After entering the function the user will see that the first thing done is to identify the model and flavor transformation classes. With these two identifications, a `snmodel` object is created which is given the name of the location of the input file and the flavor transformation class. With the model instantiated, we use the `get_time function` to find the limits of the simulation. Then, depending upon whether the number of time bins and/or deltat were set, the number of snapshots and the time spacing to be used are determined. The name for the output tarball is then set and a tar file created and opened. If no name for the tarball is supplied, the default behavior is to take the stem of the input file name and append pieces to identify the flavor transformation used, the tmin and tmax, the number of time bins, and the distance to the supernova. This tarball will be placed in the same directory as the input simulation data file. A small file called `parameterinfo` containing the values of the parameters used in the flavor transformation is inserted into the tarball. The code then loops over the snapshot times to be generated and for each snapshot creates a table with seven columns. In order the columns are: the energy in GeV, the fluence of electron neutrinos in $/cm^2$, the fluence of muon neutrinos in the same units, the fluence of tau neutrinos, and then the same for the three antineutrino flavors in the same order. The code then loops over the neutrino energies from 0 to 0.1 GeV in 501 steps of 200 keV. Using the energy and time, it invokes the `get_oscillatedspectra` function of the snmodel class then multiplies by the timebin width, divides by $4\pi d^2$ (after converting d to cm), and multiplies by the energy bin width of 200 keV. The column for ν_x and $bar\nu_x$ is repeated twice. This is the `SNOwGLoBES` format. The table is then inserted into a file which is given a name to indicate the simulation used and the snapshot number. The file is then packed into the tarball. The code then proceeds to the next time snapshot. Once all the snapshots have been generated the code returns the name of the tarball.

4 run_snowglobes

The purpose of `run_snowglobes` it to take a compressed file containing fluence files in SNOwGLoBES format and run them through the SNOwGLoBES software. In essence `run_snowglobes` can be used instead of the `supernova.pl` script that comes with SNOwGLoBES. In fact, much of `run_snowglobes` is a literal translation of `supernova.pl` into python. `run_snowglobes` also creates a supernova.glb file then invokes `supernova`. The mandatory arguments passed to `run_snowglobes` are the location of the SNOwGLoBES installation directory, the path to the folder containing the compressed file, and the name of the compressed file. A fourth optional argument is the name of the detector to be run. If the fourth argument is not supplied then all detectors are run.

The entirety of `run_snowglobes` is composed of the `go` function. The first thing that `run_snowglobes.go` does is unpack the input tarfile into the /fluxes folder of the `SNOwGLoBES` installation directory. This tarfile could have been created by `to_snowglobes`, or it could be created by the user, as long as the format is consistent with the format of the `to_snowglobes` generated tarfiles. `run_snowglobes` also accepts zipfiles.

Next, a new `/out` folder is created in the `SNOwGLoBES` installation directory. This effectively removes any leftover data from previous runs ensuring there is no cross-contamination if `SNEWPY` is run multiple times.

Then the driver kicks in. There are two versions, one that runs all the detectors, if the optional fourth argument in `run_snowglobes.go` is not included, and one that runs only the specified detector if the fourth argument is included. If all detectors are to be run, the driver consists of 12 individual calls of the function `format_globes_for_supernova`, one for each detector. More can be added as more detectors are created, and current ones can be commented out here, if the user does not wish them to run. `format_globes_for_supernova` takes four inputs: the name of a flux file, the detector material, the detector name, and a weight value (if the weight value is anything other than the number 1, weights will be applied. As the code stands now, the string "weight" is supplied as the fourth argument, letting the user know that weights are being applied). Below is an example of the function call for the 40 kiloton Argon detector:

```
if format_globes_for_supernova(IndividualFluxFile, "argon", "ar40kt",
"weight") == "Complete":
    os.system('echo "Finished {0}||ar40kt"'.format(IndividualFluxFile))
```

It is written as an if statement because many different files provide data to this function, and if one of the files is not found, then an "incomplete" is returned.

When only a single detector is run, the driver determines the detector material from the inputted detector name. Then, `format_globes_for_supernova` is run, with the same four inputs as above. The function call is this:

```
if format_globes_for_supernova(IndividualFluxFile, detector_material,
detector_input, "weight") == "Complete":
    os.system('echo "Finished {0}||{1}"'.format(IndividualFluxFile,
    detector_output))
```

Both drivers also include a subroutine that calculates what percent of the calculations have been completed, and then prints that value to the screen, for the user's enlightenment.

`format_globes_for_supernova` is the main function within `run_snowglobes.go`. It reformats data from a bunch of different files in order to run `supernova`, the executable that is the heart of SNOwGLoBES. First, it creates a `supernovae.glb` in the same directory as SNOwGLoBES, writes the preamble to it, and then checks that the flux files exist. Since `run_snowglobes` pulls in data from many different files, it checks the existence of these files first, and if any of these files do not exist in their specified path, then the program returns an Incomplete and notifies the user that there is an issue. After checking for the existence of the flux file, the modified flux is written to the `supernova.glb`, replacing the flux file name with the input argument. Next, the same thing happens with the channel. The existence of `ChannelFileName` is verified, and then the channel names from each line is written to the `supernova.glb`. Below is an example of the channel file, in this case for lead. Thus, this file corresponds to the Halo 1 and Halo 2 detectors.

```
nue_e 0 + e 82.
nuebar_e 1 - e 82.
numu_e 2 + m 82.
numubar_e 3 - m 82.
nutau_e 4 + t 82.
nutaubar_e 5 - t 82.
nue_Pb208_1n 6 + e 1.
nue_Pb208_2n 7 + e 1.
nc_nue_Pb208_1n 8 + e 1.
nc_nuebar_Pb208_1n 9 - e 1.
nc_numu_Pb208_1n 10 + m 1.
nc_numubar_Pb208_1n 11 - m 1.
nc_nutau_Pb208_1n 12 + t 1.
nc_nutaubar_Pb208_1n 13 - t 1.
nc_nue_Pb208_2n 14 + e 1.
nc_nuebar_Pb208_2n 15 - e 1.
nc_numu_Pb208_2n 16 + m 1.
nc_numubar_Pb208_2n 17 - m 1.
nc_nutau_Pb208_2n 18 + t 1.
nc_nutaubar_Pb208_2n 19 - t 1.
```

The first entry (nue_e, nuebar_e, etc.) corresponds to the channel names. The second entry is an integer that increases by one, down the list. The third entry is either a + or a -. The + refers to a neutrino type, whereas the - refers to an anti-neutrino type. The fourth entry is either an e, m, or t, which further specifies the type of interaction. e is electron, m is muon, and t is tau. The final entry (82., 1.) corresponds to the weighting factor (which will be explained later, when discussing weights).

Then the existence of `detector_configurations.dat` is verified. That file looks like:

```
# Configuration/  mass in kton / target normalization
wc100kt30prct  100.     0.1111
wc100kt15prct  100.     0.1111
ar40kt         40.      0.025
scint20kt      20.      0.1429
halo1          0.079    0.004808
halo2          1.       0.004808
novaND         0.3      0.107
novaFD         14.      0.107
d2O            .001     0.1
wc100kt30prct_he 100.   0.1111
ar40kt_he      40.      0.025
icecube        51600.   0.1111
```

The first entry is the detector name, the second is the mass of the detector in kilotons, and the third is the target normalization factor, which is used later in determining the `TargetMass`. Unlike previous files, these values are not copied directly into `supernova.glb`; instead, the three separate columns are split into arrays, and those arrays are used to check that all of the detectors, their masses, and their target normalization factors, are present. The `TargetMass` is calculated for each detector by multiplying the given mass in kilotons by the target normalization factor. Then the modified detector is written to the `supernova.glb`, and the mass is replaced with the `TargetMass`. Next, the paths to the applicable cross section files are copied over to the `supernova.glb`. The modified channels and their post smearing efficiencies are also copied over, once the files containing that information are verified to exist.

The post smearing efficiencies are stored in the effic folder, labeled by detector and type of interaction. Each efficiency file contains a list of values, which models the specific detector's efficiency as a function of detected energy. These are the only files copied into the `GLOBESFILE` which are not required. If an efficiency file is not provided in the correct place for a detector that is being run, the program will just assume a 100 % efficiency, which is equivalent to the list in the corresponding file being full of ones.

After all of those files are read into the `supernova.glb`, the postamble is copied over and the `supernova.glb` is closed. The `supernova.glb` is moved to the SNOwGLoBES directory, and the following two lines of code actually invokes supernova:

```
ComString = "{0} {1} {2} {3}".format(ExeName, FluxNameStem,
ChannelFileName, ExpConfigName)
        call(ComString, shell = True)
```

The variables taken into ComString are as follows:

- `ExeName` - "bin/supernova" aka the executable
- `FluxNameStem` - The path to one of the flux files unpacked from the inputted tarfile; the code runs for each FluxNameStem

- `ChannelFileName` - The path to the channel file for the current running detector; the code will run for each channel file if all detectors are to be run, or for just the channel file that corresponds to the single inputted detector

- `ExpConfigName` - The name of the detector (icecube, wc100kt30prct, wc100kt30prct_he, wc100kt15prct, wc100kt15prct_he, hyperk30prct, ar40kt, halo1, halo2, ar40kt_he, novaND, novaFD, scint20kt)

The final action `format_globes_for_supernova` performs is to call the `apply_weights` function, unless the final argument supplied to `format_globes_for_supernova` is 1 (in which case `apply_weights` will not be called, and only the unweighted files and plots will be created).

`apply_weights` take in four inputs: smearing, FluxNameStem, ChannelFileName, and ExpConfigName. `apply_weights` is called twice, with two different inputs for smearing: `"events_u"` and `"unsmeared"`. The `"events_u"` corresponds to smeared, and `"unsmeared"` is, predictably, unsmeared. This is to ensure weighted files are created from both the smeared and unsmeared files. The last three inputs to the `apply_weights` function were explained earlier.

`apply_weights` first copies the weighting factor from the channel file (the third entry), and stores these as the variable `NumTargetFactor`. Then the unweighted file is copied to the weighted file, except that the event rates are now weighted (found by multiplying the original event rate by `NumTargetFactor`).

Once that function has run, all of the output from `run_snowglobes` has been generated. For every flux file within the inputted tarfile, four files are created for each channel for each detector that is run. These four files correspond to the different combinations of smeared, unsmeared, weighted, and unweighted. The different channels for each detector are found within the channel file for the material the detector runs on. For example, if the 30 kiloton Water Cherenkov detector is run, then files are created for each of the following channels:

- `ibd`
- `nue_e`
- `nuebar_e`
- `numu_e`
- `numubar_e`
- `nutau_e`
- `nutaubar_e`
- `nue_O16`
- `nuebar_O16`
- `nc_nue_O16`
- `nc_nuebar_O16`
- `nc_numu_O16`
- `nc_numubar_O16`
- `nc_nutau_O16`
- `nc_nutaubar_O16`

For each channel, the four files created are: smeared unweighted, unsmeared unweighted, smeared weighted, and smeared unweighted. Each of files have two columns: the first is the energy bin in GeV, and the second is the number of events.

The filenames are used in the next module `from_snowglobes` to differentiate between their contents, so the nomenclature is very precise. Below are a few examples of file names.

- `sqatest_nc_nutaubar_Pb208_2n_halo2_events_smeared_weighted.dat`

- `sqatest_nuebar_e_halo2_events_smeared_unweighted.dat`

- sqatest_nue_Ar40_ar40kt_events_unsmeared_weighted.dat
- sqatest_ibd_wc100kt30prct_events_unsmeared_weighted.dat

The first value is the name of the fluxfile (in this case, sqatest), then the type of interaction (for these files that is: _nc_nutaubar_Pb208_2n_; _nuebar_e_; _nue_Ar40; _ibd), and then the detector (halo2, ar40kt, and wc100kt30prct). After the word events is the smear value (either _smeared or _unsmeared) and the weight value (either _weighted or _unweighted).

5 from_snowglobes

The purpose of from_snowglobes is to collate the output generated by SNOwGLoBES into the observable channels of each detector in the four combinations of weighted and unweighted event rates and with and without applying detector energy smearing. The mandatory arguments passed to from_snowglobes are the location of the SNOwGLoBES installation directory, a path where the output file will be placed, and the name of the compressed file that was run through run_snowglobes. Again, a fourth optional argument is the name of the detector that was run. If the fourth argument is not supplied then all detectors are assumed. The output from from_snowglobes is a compressed file with a name which is the modified version of the name of the file that was run through SNOwGLoBES which is placed in the location specified.

The entirety of from_snowglobes is composed of the collate function. Within this function, the tarfile (either outputted from to_snowglobes or separately provided by the user) is opened, and the names of the flux files are stored for later use. The files are not unpacked. Then, much like run_snowglobes, one of two drivers is called, depending whether the user specified a particular detector, or all detectors are being run (as is the default). If all detectors are run, the driver iterates through all flux files, both values for the smearing, and then both unweighted and weighted, and calls the add_funct for each detector. Below are a few examples of the calls for different detectors.

```
add_funct (str(single_flux), "wc100kt15prct", weightval, smearval, "nc",
"_e_","ibd", "nue_O16", "nuebar_O16")
add_funct (str(single_flux), "halo1", weightval, smearval, "nc", "_e_",
"Pb208_1n", "Pb208_2n")
```

add_funct takes in five arguments:

- Name of Flux File - For each function call, this argument is inputted in the form of str(single_flux), and is the variable through which the different flux files within the input tarfile iterate through
- Detector Name - In the above examples, "wc100kt15prct" and "halo1"
- Weight Value - For each function call, this argument takes the form of weightval, which corresponds to both the weighted and unweighted forms

- `Smear Value` - For each function call, this argument takes the form of `smearval`, which corresponds to both the smeared and unsmeared forms

- `*arg` - The last argument can take in different numbers of variables. In the above examples `*arg` has 5 variables (`"nc", "_e_", "ibd", "nue_O16", "nuebar_O16"`) and 4 variables (`"nc", "_e_", "Pb208_1n", "Pb208_2n"`). The different `*arg` values define the different categories that are summed together. For example, `"nc"` stands for Neutral Current, and all Neutral Current events for the same detector, flux file, smear, and weight, are added together. The same happens for all files with `"_e_"` which encompasses all mu, tau, anti-mu, and anti-tau type interactions.

It should be noted that for detectors with a high energy counterpart, the function calls look like so:

```
add_funct (str(single_flux), "ar40kt_events", weightval, smearval, "nc",
"_e_","nue_Ar40", "nuebar_Ar40")
add_funct (str(single_flux), "ar40kt_he", weightval, smearval, "nc", "_e_",
"nue_Ar40", "nuebar_Ar40")
```

Each input is consistent with above, except that the detector name includes a `_events` suffix, and the high energy counterpart includes a `_he` suffix.

If a single detector is indicated, the driver operates in much the same way, but only completes the call for the indicated detector. Again, much like in `run_snowglobes`, the drivers include a subroutine that calculates what percent of the plots have been created, and then prints that value to the screen in order to provide the user with some information about the progress.

`add_funct` starts by redefining different input values so that they are properly formatted for display on plots. Because `from_snowglobes` predominantly runs by separating out files based on what information is included in the filename, many of the delineations must include a _, in order to only grab the necessary files. For example, one of the `*arg` values is `"_e_"`, meaning that all the files with `"_e_"` in the filename correspond to a mu, tau, anti-mu, or anti-tau interaction, and thus must be summed together. However, if the `*arg` value was simply `"e"`, nearly every file would be grabbed, since there would be no discrimination between the e in the word `"smeared"` and the e in `_nutau_e_halo1`. Thus the underscores are used as delineation indicators in many cases, and need to be removed prior to using some input values as plot labels.

Different files produced from `run_snowglobes` have to be summed together, to model what one would actually see from a detector. The interactions that are indistinguishable from each other are added together within the time bins, and then plotted as cumulative interactions. The files are only summed for their respective flux file, detector, and smearing and weighting values. The following files are added together:

- `nc` - Neutral Current

- `_e_` - $\nu_x + e^-$ elastic scattering

- `ibd` - Inverse Beta Decay

- `nue_O16` - ν_e ^{16}O
- `nuebar_O16` - $\bar{\nu}_e$ ^{16}O
- `Pb208_1n` - ^{208}Pb 1 neutron
- `Pb208_2n` - ^{208}Pb 2 neutrons
- `nue_Ar40` - ν_e ^{40}Ar
- `nuebar_Ar40` - $\bar{\nu}_e$ ^{40}Ar
- `nue_C12` - ν_e ^{12}C
- `nuebar_C12` - $\bar{\nu}_e$ ^{12}C

The actual addition is done by iterating through the different values of `*arg`, which correspond to the different chunks the files will be split into. For each of those `*arg` values, the events for each energy bin are added together and appended to dictionaries. `compile_dict` contains keys of the energy bins, with values as lists of the summed energies. The index of the list corresponds to the index of the `*arg` value for which those energies were summed. This dictionary is translated over to a condensed data file with an Energy column and either four or five other columns, corresponding to the `*arg` values. The condensed files are saved with the same title as the uncondensed files, except the word `Collated` appears in the name, and the type of interaction does not.

One by one, those list indices from the `compile_dict` values are transferred to `new_dict`, which plots a single line on the plot. Once all items in `compile_dict` have been exhausted, the plot is saved with similar nomenclature to the files from `run_snowglobes`. An example of a plot name is shown below:

```
nakazato-LS220-BH-z0.004-s30.0.tbin30.NoTransformation.-0.110,0.342,30-
10.0kpc_novaFD_smeared_unweighted_log_plot.png
```

The name of the plot file includes, in order: the name of the flux file it originated from (`nakazato-LS220-BH-z0.004-s30.0.tbin30.`), the type of transformation applied (`NoTransformation`), tmin, tmax and number of time slices (`-0.110,0.342,30-10.0kpc`, the detector (`novaFD`), whether it is smeared or unsmeared, whether it is weighted or unweighted, and then `_log_plot.png`.

Four plots are generated per detector for each flux file. Below are four plots generated for the Water Cherenkov 100 Kiloton 30 percent detector, run through the Livermore flux file.

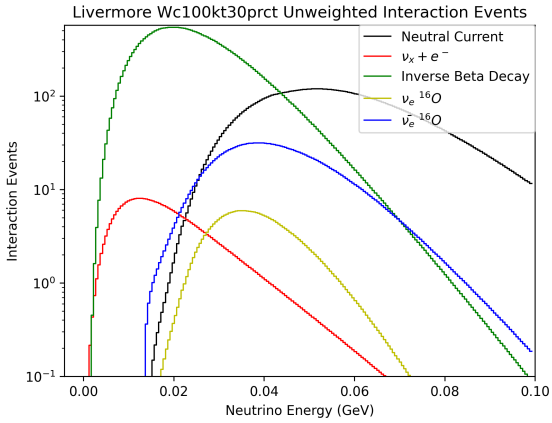

Figure 1: Unweighted Unsmeared

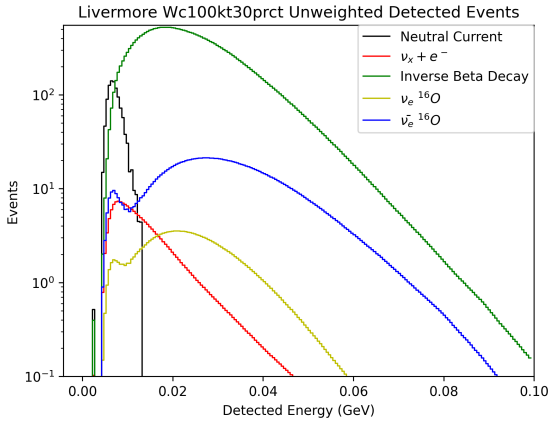

Figure 2: Unweighted Smeared

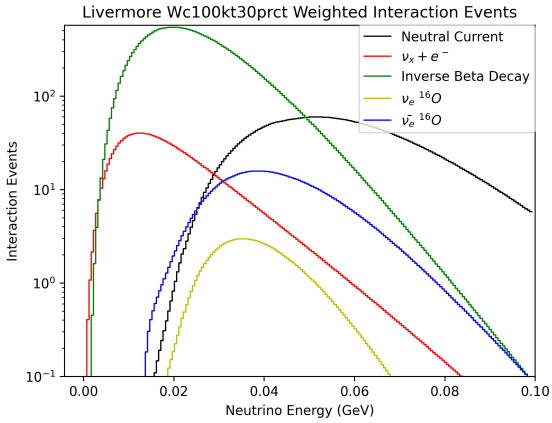

Figure 3: Weighted Unsmeared

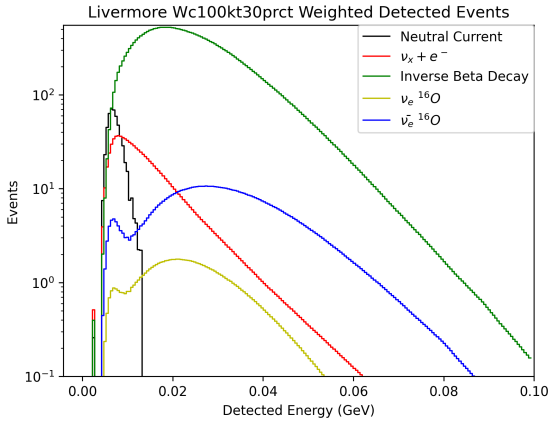

Figure 4: Weighted Smeared

After all of the plots and condensed files are generated, all of the files generated by `run_snowglobes` are deleted. A new tarfile is created with the same name as the input tarfile, with `Output.tar.gz` appended to the end.

References

[1] Ken'ichiro Nakazato, Kohsuke Sumiyoshi, Hideyuki Suzuki, Tomonori Totani, Hideyuki Umeda, and Shoichi Yamada. Supernova neutrino light curves and spectra for various progenitor stars: From core collapse to proto-neutron star cooling. *The Astrophysical Journal Supplement Series*, 205(1):2, Feb 2013. ISSN 1538-4365. doi: 10.1088/0067-0049/205/1/2. URL `http://dx.doi.org/10.1088/0067-0049/205/1/2`.

[2] Shunsaku Horiuchi and James P. Kneller. What can be learned from a future supernova neutrino detection? *Journal of Physics G Nuclear Physics*, 45(4):043002, April 2018. doi: 10.1088/1361-6471/aaa90a.

A The Flavor Transformation Class Formulae

In this appendix we provide the formulae for the P's which appear in equations (1) - (4) that the user can find in the flavor transformation prescriptions in `flavor_transforamtion.py`. In these equations the quantities $D_{\alpha i} = —\mathrm{U}_{\alpha i}|^2$ where $U_{\alpha i}$ are the elements of the vacuum mixing matrix. In terms of the angles θ_{12}, θ_{13}, θ_{23} the expressions for the D's are: $D_{e1} = \cos^2\theta_{12}\,\cos^2\theta_{13}$, $D_{e2} = \sin^2\theta_{12}\,\cos^2\theta_{13}$, $D_{e3} = \sin^2\theta_{13}$.

A.1 No Transformation

$P_{ee} = P_{xx} = 1 \quad P_{ex} = P_{xe} = 0$

$\bar{P}_{ee} = \bar{P}_{xx} = 1 \quad \bar{P}_{ex} = \bar{P}_{xe} = 0$

A.2 AdiabaticMSW_NMO

$P_{ee} = D_{e3} \qquad P_{ex} = 1 - P_{ee}$
$P_{xx} = (1 + P_{ee})/2 \quad P_{xe} = (1 - P_{ee})/2$

$\bar{P}_{ee} = D_{e1} \qquad \bar{P}_{ex} = 1 - \bar{P}_{ee}$
$\bar{P}_{xx} = (1 + \bar{P}_{ee})/2 \quad \bar{P}_{xe} = (1 - \bar{P}_{ee})/2$

A.3 AdiabaticMSW_IMO

$P_{ee} = D_{e2} \qquad P_{ex} = 1 - P_{ee}$
$P_{xx} = (1 + P_{ee})/2 \quad P_{xe} = (1 - P_{ee})/2$

$\bar{P}_{ee} = D_{e3} \qquad \bar{P}_{ex} = 1 - \bar{P}_{ee}$
$\bar{P}_{xx} = (1 + \bar{P}_{ee})/2 \quad \bar{P}_{xe} = (1 - \bar{P}_{ee})/2$

A.4 NonAdiabaticMSWH_NMO

$P_{ee} = D_{e2} \qquad P_{ex} = 1 - P_{ee}$
$P_{xx} = (1 + P_{ee})/2 \quad P_{xe} = (1 - P_{ee})/2$

$\bar{P}_{ee} = D_{e1} \qquad \bar{P}_{ex} = 1 - \bar{P}_{ee}$
$\bar{P}_{xx} = (1 + \bar{P}_{ee})/2 \quad \bar{P}_{xe} = (1 - \bar{P}_{ee})/2$

A.5 NonAdiabaticMSWH_IMO

$P_{ee} = D_{e2} \qquad P_{ex} = 1 - P_{ee}$
$P_{xx} = (1 + P_{ee})/2 \quad P_{xe} = (1 - P_{ee})/2$

$\bar{P}_{ee} = D_{e1} \qquad \bar{P}_{ex} = 1 - \bar{P}_{ee}$
$\bar{P}_{xx} = (1 + \bar{P}_{ee})/2 \quad \bar{P}_{xe} = (1 - \bar{P}_{ee})/2$

A.6 TwoFlavorDecoherence

$$P_{ee} = 1/2 \qquad P_{ex} = 1 - P_{ee}$$
$$P_{xx} = (1 + P_{ee})/2 \quad P_{xe} = (1 - P_{ee})/2$$

$$\bar{P}_{ee} = 1/2 \qquad \bar{P}_{ex} = 1 - \bar{P}_{ee}$$
$$\bar{P}_{xx} = (1 + \bar{P}_{ee})/2 \quad \bar{P}_{xe} = (1 - \bar{P}_{ee})/2$$

A.7 ThreeFlavorDecoherence

$$P_{ee} = 1/3 \qquad P_{ex} = 1 - P_{ee}$$
$$P_{xx} = (1 + P_{ee})/2 \quad P_{xe} = (1 - P_{ee})/2$$

$$\bar{P}_{ee} = 1/3 \qquad \bar{P}_{ex} = 1 - \bar{P}_{ee}$$
$$\bar{P}_{xx} = (1 + \bar{P}_{ee})/2 \quad \bar{P}_{xe} = (1 - \bar{P}_{ee})/2$$

A.8 NeutrinoDecay_NMO

With τ the lifetime and m the mass of the heaviest neutrino, the decay width is $\Gamma = mc/(E\tau)$. We assume the heaviest neutrino decays to the lightest neutrino only and the energy of the neutrino does not change. Future versions of `SNEWPY` will correct this assumption. If d is the distance to the supernova then

$$P_{ee} = D_{e1}[1 - \exp(-\Gamma d)] + D_{e3}\exp(-\Gamma d)$$
$$P_{ex} = D_{e1} + D_{e2}$$
$$P_{xx} = 1 - P_{ex}/2$$
$$P_{xe} = (1 - P_{ee})/2$$

$$\bar{P}_{ee} = D_{e1}$$
$$\bar{P}_{ex} = D_{e1}[1 - \exp(-\Gamma d)] + D_{e2} + D_{e3}\exp(-\Gamma d)$$
$$\bar{P}_{xx} = 1 - \bar{P}_{ex}/2$$
$$\bar{P}_{xe} = (1 - \bar{P}_{ee})/2$$

A.9 NeutrinoDecay_IMO

With the same definitions as in NeutrinoDecay_NMO but now

$$P_{ee} = D_{e2}\exp(-\Gamma d) + D_{e3}[1 - \exp(-\Gamma d)]$$
$$P_{ex} = D_{e1} + D_{e3}$$
$$P_{xx} = 1 - P_{ex}/2$$
$$P_{xe} = (1 - P_{ee})/2$$

$$\bar{P}_{ee} = D_{e3}$$
$$\bar{P}_{ex} = D_{e1} + D_{e2}\exp(-\Gamma d) + D_{e3}[1 - \exp(-\Gamma d)]$$
$$\bar{P}_{xx} = 1 - \bar{P}_{ex}/2$$
$$\bar{P}_{xe} = (1 - \bar{P}_{ee})/2$$