Signal Monte-Carlo generation for supernova relic neutrinos

Sonia El Hedri

September 23, 2019

We need to generate signals involving a low energy positron (according to a signal-dependent spectrum) and a 2.2 MeV photon (resulting from neutron capture). The general procedure is the following:

- **Input spectra:** We give only energy bins to the code (`emin` and `emax` in practice). Then, we can reweight everything for the different analyses.
- **ZBS files:** Generate ZBS files containing time information and run numbers. These files contain a fixed number of events for each subrun.
- **SKDetSim:** Run SKDetSim using the ZBS files of the previous step as an input. SKDetSim will output skroot files.
- **Lowe reconstruction:** Run the standard event reconstruction algorithms for low energy neutrinos on the SKDetSim output. Outputs ROOT files.
- **Random trigger background:** Superimpose random trigger background events for the AFT trigger window.

The main codes are in

/home/elhedri/SK2p2MeV/mc/generate/ibd_signal/

1 ZBS file generation

The main code for ZBS file generation is `vectgen.F` and should be ran using the following command:

```
./vectgen <run number> <random seed file> <ZBS output> <emin> <emax>
```

The random seed file is generated by

```
./make_random <random seed file>
```

and is just a text file with the seed. I assume we do this for reproducibility. The vector generation file iterates from the first run given to the last run given. For each run, it obtains the timing information (from a separate file) and stores it in the `IDATA` table, ultimately saved into the ZBS file and read by SKDetSim. The arguments emin and emax are the mininimal and maximal positron energies for which we want to generate events. For each subrun, this code generates a fixed number of events, given by the variable `num`, set inside the code. At each iteration, the `spectrum` function is called to generate the following event parameters:

- A random positron energy, between emin and emax
- A random position for the positron in the fiducial volume
- A neutrino, a positron direction, and a neutron. The characteristics of these particles are drawn from distributions built using the IBD cross-section.

The neutron propagation and capture will be modeled by SKDetSim. After setting the event parameters, `spectrum` fills in the corresponding tables for the ZBS output. Then, `vectgen` fills in `IDATA` with the run and timing information for each event.

1.1 Kinematics

We consider that the proton is static in the lab frame. The kinematics is therefore as shown in figure 1.

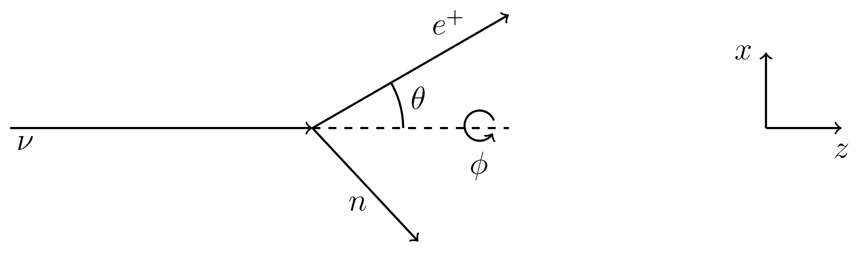

Figure 1: Kinematics and angle definitions for an IBD events. The proton is not moving in the lab frame and is therefore not shown here. θ is the angle between the positron and the neutrino directions and ϕ is the azimuthal angle corresponding to rotations around the z axis.

The positron energy is generated randomly in some user-defined range $[E_{min}, E_{max}]$. Then, we generate θ with a probability distribution defined by the Strumia-Vissani IBD cross-section [1]. The ϕ angle is generated randomly in $[0, 2\pi]$. Then, the other variables can be inferred from energy-momentum conservation. The final positrons and momenta will be stored in the `MC` branch of the skroot tree (SKDetSim output) in the following way:

- **Neutrino:** `pvtxvc[0]`, `pvc[0]`

- **Positron:** `pvtxvc[1]`, `pvc[1]`
- **Neutron:** `pvtxvc[2]`, `pvc[2]`

2 SKDetSim, reconstruction, T2K dummy background

2.1 SKDetSim + lowe reconstruction

We use the most recent version of SKDetSim located in

```
/home/elhedri/skdetsim
```

This version has been modified by Koshio-san to write the neutron capture vertex information in ROOT output files. The `NEUTRON` compiler option turned on in the `GNUMakefile`. This option allows SKDetSim to use a larger time window (800 μs) to generate the positron and the 2.2 MeV gamma in the same event. It also generates the dark noise only up to 18 μs so that the random trigger background can be superimposed on the rest of the event without double counting. We take the input card from

```
/home/sklowe/solar_apr19/gen_mc_gain/supersim_root.card
```

In this card, only the random seeds need to be updated, as the run-dependency is already specified in the ZBS file.

The output of SKDetSim is a skroot file that is passed to `lowfit_sk4_gain_corr_mc`, a code that computes all the lowe parameters (reconstructed energy, etc...). Here, we reused the solar neutrino code (see `/home/sklowe/solar_apr19/gen_mc/`) that calls the SKOFL code `lfallfit_sk4`.

2.2 Random trigger background

The code to superimpose random trigger background on a skroot file is in

```
/home/elhedri/SK2p2MeV/mc/generate/combine/incorporate.C
```

and the command to run it is

```
./incorporate <output file> <input file> <run number>
```

Ideally, for each SK-IV run we would need to inject the T2K background corresponding to this specific run. In practice this is not possible since some runs do not have any T2K dummy (or even beam) data. Given that the time dependence of the T2K dummy events is relatively weak, we divide the entire SK-IV run period into 10 bins of equal duration (about one year each). For MC signal events corresponding to a given run, we then inject T2K dummy data randomly taken from the corresponding one-year bin. In principle, with this method, one T2K event could be used for multiple MC events, but as long as we have enough T2K events this situation should be rare.

For a given MC event and a given (appropriately chosen) T2K event, the injection procedure is rather straightforward. Since the total SHE+AFT time window is 535 μs and the first 18 μs of noise are already simulated by SKDetSim, we only need to inject 517 μs worth of T2K events. Since the time window for T2K dummy events is 1 ms long, we inject hits from either the first or the second half of it (this choice is made at random). We can also reuse T2K beam events, but in this case, we take hits only from the first 500 μs of the event window and fill in the remaining 17 μs by taking hits from another T2K event.

3 Running the code on the cluster

The code includes a series of scripts designed to run the code on the sukap cluster. These codes are in

```
/home/elhedri/SK2p2MeV/mc/generate/ibd_signal/scripts/
```

The input/output folders as well as the cluster queue and the executables are specified a as environment variables in `exec_card.sh`. The vector file generation codes are then run by

```
./qsub_vectgen.sh -f <start run> -l <end run> -e <emin> -E <emax>
```

For each energy bin [emin, emax], a subfolder is created in the output folder specified in `exec_card.sh` and will contain the final vector files. Once these vector files are ready, SKDetSim and lowfit can then be run by doing

```
./qsub_skdetsim.sh <start run> <end run> <emin> <emax>
```

The ROOT outputs are saved in the same way as for the vector files. Then, if one is interested in neutron tagging, the T2K injection can be run by doing

```
./qsub_combine.sh <start run> <end run> <emin> <emax>
```

4 Reweighting

In order to plot distributions for a given neutrino/positron energy spectrum, we need to reweight our events appropriately. This reweighting can be done by using the function `weight(float enu, float elow, float ebin, float *nuspectrum)` in

```
/home/elhedri/SK2p2MeV/mc/generate/combine/dsigma.cpp
```

Here, we assume that the neutrino spectrum is given as a histogram with evenly-sized bins. `enu` is the neutrino energy (taken from the `MC` branch of the skroot tree), `elow` is the lowest neutrino energy considered in the histogram, `ebin` is the bin width, and `nuspectrum` is the bin content. This function can be called by an external code, if `dsigma` is linked. An example is given in

```
/home/elhedri/SK2p2MeV/mc/generate/combine/examples/read_reweight.cpp
```

This folder also contains a `Makefile` and a script to run the code.

References

[1] A. Strumia and F. Vissani, Phys. Lett. B **564**, 42 (2003) doi:10.1016/S0370-2693(03)00616-6 [astro-ph/0302055].