Running NuWro *

Cezary Juszczak

Institute for Theoretical Physics
University of Wrocław

The NuWro Neutrino Event Generator developed by the Wrocław Neutrino Group (WNG) is lightweight but full featured. It handles all interaction types important in neutrino-nucleus scattering as well as DIS hadronization and intranuclear cascade. Its input file, by default `params.txt`, is a plain text file and the output file, by default `eventsout.root`, is a `root`[1] file which can be analyzed by means of the included `myroot` program, or by standard `root`, after loading supplied dictionary library `event1.so`.

1. Installing NuWro

NuWro is a neutrino event generator developed by the Wrocław Neutrino Group. It can be downloaded as a tar ball from:

```
http://borg.ift.uni.wroc.pl/websvn
```

Alternatively, the subversion command:

```
svn export svn://borg.ift.uni.wroc.pl/pub/nuwro
```

can be used to create a directory `nuwro` containing the copy of the current NuWro sources. Then it should be enough to type:

```
cd nuwro
make
```

to build the program, provided the `root` software configured with the Pythia6 library is installed on one's computer.

* Presented by C. Juszczak at the 45th Winter School in Theoretical Physics "Neutrino Interactions: from Theory to Monte Carlo Simulations", Lądek-Zdrój, Poland, February 2–11, 2009.

[1] `root` and `rootcint` are parts of the CERN software http://root.cern.ch/ and `.root` is the file extension of data files used and produced by the software.

1.1. Installing root with Pythia6

Unfortunately the `libPythia6.so` library is not included in the `root` distribution. It must be downloaded and built separately before building root. It is best done by typing either

`build_pythia6.sh`[2] `gfortran`

or

`build_pythia6.sh g77`

depending on which fortran compiler you have.

The resulting `libPythia6.so` file should be placed in the `lib` directory of the `root` source tree[3]. Then the root software should be configured and build with the following commands [4] issued in the `root` sources directory:

```
./configure --with-pythia6-libdir=‘pwd‘/lib
make
```

To be able to run `root` from any location and compile `root` based programs like `nuwro`, the following lines should be added to one's `.bash_profile`[5]

```
export ROOTSYS= path to directory where you made root
export PATH=$PATH:$ROOTSYS/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
```

2. Running Nuwro

To run the program in the default mode type:

```
./nuwro
```

and to use non-default input and output type:

```
./nuwro -i myinput.txt -o myoutput.root
```

Any number of values of the parameters defined in the input file may be overwritten by values specified on the command line as follows:

```
./nuwro -p "parname1=parvalue1" -p "parname2=parvalue2" ...
```

which is very useful when running `nuwro` in batch mode with changing parameter values and output locations.

---

[2] `build_pythia6.sh` is a script by Robert Hatcher <rhatcher@fnal.gov> which can be downloaded from `http://home.fnal.gov/~rhatcher/build_pythia6.sh.txt`

[3] Obtain root sources from `http://root.cern.ch/drupal/content/downloading-root`

[4] Several libraries are needed to build root and Pythia6. Under Ubuntu 9.04 the following packages must be installed: `g++, gfortran, libX11-dev, libxft-dev, x11proto-xext-dev, libXpm-dev, libXext-dev.`

[5] For a system wide installation, a file `root.sh` consisting of these three lines should be placed in `/etc/profile.d/` directory, instead. Under Ubuntu it is convenient to put them to `.bashrc`

NuWro input file

All the parameters of the NuWro generator are read from a file, usually `params.txt`. The structure of this file is quite simple – each line is either a comment (if it begins with `#`):

```
# This is an example of a comment line
```

or a substitution:

```
parameter_name = parameter_value
```

Parameter names coincide with the corresponding `C++` variable names inside the program. Only four parameter types are used: `int` (integer number), `double` (floating point number), `vec` (3D vector initialized by three white space separated numbers) and `string` (stretches to the end of line).

The meaning of the parameters is briefly explained and their possible values listed in the commented lines of the `params.txt` file itself. Let us summarize the meaning of the most important the parameters here.

Basic parameters

The test events are not stored in the output file, but the average of their weights becomes the total cross section in each channel:

```
number_of_test_events = 1000000
```

The number of unweighted events to be stored in the output is given by:

```
number_of_events = 500000
```

The beam definition

At present, only beams of identical neutrinos flying in the same direction are allowed. The beam direction coordinates x, y, z and neutrino PDG code must be specified e.g.

```
beam_direction     = 0 0 1
beam_particle      = 14
```

The neutrino energy given in MeV can be either fixed e.g.

```
beam_energy = 1000
```

or given as a histogram encoded in the sequence of numbers: E_{min}, E_{max}, n_1, n_2, n_3 $\ldots$, n_k.

```
beam_energy = 1000 4000 1 2 3
```

The number of beans k is inferred from the length of the sequence and the bean width is $(E_{max} - E_{min})/k$. Definitions of a few popular beams are included in the `params.txt` file and it is enough to uncomment the corresponding (sometimes very long) line to use one of them.

The target nucleus definition

The target nucleus is defined by the following parameters:

```
nucleus_p       = 8   // number of protons
nucleus_n       = 8   // number of neutrons
nucleus_density = 1   // 1 - constant, 2 - realistic density
```

and the switch identifying the nucleus model to be used:

```
nucleus_target = 1
```

with the following allowed values: **0** - free nucleon; **1** - Fermi gas; **2** - local Fermi gas; **3** - FG with Bodek-Ritchie momentum distribution; **4** - "effective" spectral function (carbon or oxygen); **5** - deuterium.

In cases where the Fermi Gas model is used it is possible to specify:

```
nucleus_E_b      =  27 // nucleon bounding Energy in MeV
nucleus_kf       = 225 // Fermi momentum in MeV
```

The physics effects switches

Nonzero values of the switches:

`dyn_qel_cc`, `dyn_res_cc`, `dyn_dis_cc`, `dyn_coh_cc`,
`dyn_qel_nc`, `dyn_res_nc`, `dyn_dis_nc`, `dyn_coh_nc`

indicate that quasi elastic, resonant, deep inelastic, and coherent events should be generated. There are `nc`/`cc` variants to separately control generation of neural current and charge current events.

The quasi-elastic cross sections depend on the values of axial masses:

```
qel_nc_axial_mass= 1030   //MeV
qel_cc_axial_mass= 1100   //MeV
```

and the choice of the form factors:

```
qel_cc_vector_ff_set = 1   // only 1 is possible
qel_cc_axial_ff_set = 1    // 1 - dipole form factors,
                           // 2, 3, 4 - two-fold parabolic modifications
```

By specifying a nonzeoro value of the parameters:

```
flux_correction = 1
qel_relat       = 1
```

the difference in neutrino flux between nucleon and nucleus frames is accounted for.
Finally, running the cascade code with Pauli-blocking of the intermediate nucleons, is achieved by the lines:

```
kaskada_on   = 1
pauli_blocking   = 1
```

Some parameters are not listed here because they are used only for testing `nuwro` and setting them to nondefault values would result in obtaining incorrect results.

3. Analysing the output

The output of `nuwro` is a `root` file, usually `eventsout.root`. This file contains only one object `treeout` which is a `TTree` with a single branch `e` containing the `event` objects. The easiest way to access this file is by means of the `myroot` program:

```
./myroot eventsout.root
```

which is build together with `nuwro`. It is a version of `root` containing a dictionary for the class `event` and all its dependencies. Each `event` contains the following data:

`params` - parameters read from the input file and the command line.

`flags` - set of booleans (`coh, qel, dis, res, nc, cc, anty`) to easily filter events based on primary vertex interaction type.

`dyn` - integer identifying the dynamics used in the primary vertex.

`in, tmp, out, post, all` - are STL[6] vectors of `particles` (incoming, temporary, outgoing, post, and all the particles including the intermediate cascade particles).

`weight` - a number proportional to the cross section.

It has also a number of useful methods: `q2(), s(), n(), W(), nof()`, etc.

A `particle` is a Lorentz fourvector with coordinates `t, x, y, z` denoting its energy and momentum components, supplied with is pdg code, mass, position fourvector `r` (used only by the cascade code) and some less important attributes. Several methods are added with self explanatory names like `E(), Ek(), momentum(), v(), ....`

[6] STL - stands for the C++ Standard Template Library

By convention `in[0]` is always the beam particle and `out[0]` is the outgoing lepton. Thanks to the way the `root` interpreter handles STL vectors, that you can type `out[0].t` for the outgoing lepton energy, `out.Ek()` for the energy of all outgoing particles and `@out.size()` for the number of outgoing particles (here `out[0]` stands for the first outgoing particle, `out` for any outgoing particle and `@out` for the whole vector of outgoing particles).

The possibility to use native C++ methods from the root interpreter significantly simplifies the analysis. It is possible to obtain many interesting plots with just one command:

```
treeout->Draw("out.mass()"); //masses of outgoing particles.

treeout->Draw("n()");  // number of outgoing particles

treeout->Draw("nof(111)","flag.dis*flag.cc");
// number of π^0 produced in deep inelastic charge current events

treeout->Draw("out.momentum()","out.pdg==111");
// momenta of outgoing π^0.

treeout->Draw("all.r.x:all.r.y:all.r.z");
// places where the interactions took place in the cascade code

treeout->Draw("q2()","flag.dis");
// proportional to dσ/dq^2 differential cross section in the DIS channel
```

In more complicated cases it is possible to write a script or a root based C++ program to perform the analysis. It is also possible to add more methods to the `particle` and `event` classes. More information on NuWro can be found on `http://wng.ift.uni.wroc.pl/nuwro`.

REFERENCES

[1] J.A. Nowak, *Construction of an event generator...* , PhD Thesis (in polish), Wrocław Univ. 2006, http://wng.ift.uni.wroc.pl/wng/papers/Nowak_PhD.ps

[2] J.A. Nowak, *Wrocław neutrino event generator*, Phys. Scr. T127 (2006) 70

[3] J.A. Nowak, J.T. Sobczyk, *Hadron production in Wrocław Neutrino Event Generator*, Acta Phys. Pol. B37 (2006) 2371

[4] J. Sobczyk, *NuWro – Monte Carlo generator of neutrino interactions*, in proceedings of 10th International Workshop on Neutrino Factories, Super beams and Beta beams (NuFact08), PoS(NUFACT08) 141

[5] C. Juszczak, J.A. Nowak, J.T. Sobczyk, *Simulations from a new neutrino event generator*, Nucl. Phys. B (Proc. Suppl.) 159 (2006) 211