MCgrid Version 2.0 User Guide

mcgrid@projects.hepforge.org

Contents

1 Introduction

MCgrid is a software package that provides access to the APPLgrid and fastNLO interpolation tools for Monte Carlo event generator codes. This is done by providing additional tools to the Rivet analysis system for the construction of MCgrid enhanced Rivet analyses. The interface is based around a one-to-one correspondence between a Rivet histogram class and a wrapper for an APPLgrid/fastNLO interpolation grid. The Rivet system provides all of the analysis tools required to project a Monte Carlo weight upon an experimental data bin, and the MCgrid package provides the correct conversion of the event weight to an APPLgrid/fastNLO fill call, fully accounting for the statistical subtitles in the process and the correct treatment of Catani-Seymour counter terms in the event weights. MCgrid has been tested and designed for use with the SHERPA event generator, however as with Rivet the package is suitable for use with any code which can produce events in the `HepMC` event record format.

2 Software setup

2.1 Installation and build dependancies

The MCgrid package is supplied as an external library which may be used when constructing Rivet analyses. It has a few basic dependancies, namely,

- Rivet version 2.2.0 or later.
- APPLgrid version 1.4.56 or later.
- fastNLO Toolkit version 2.3pre-2125 or later.
- Optionally pkg-config for path management.

Both APPLgrid and fastNLO are optional, but at least one of them must be available.

In order to install the MCgrid examples and test code you should additionally have the LHAPDF and HOPPET [2] packages installed.

MCgrid may be configured and installed in the conventional way with the autotools build system. In the mcgrid directory you should perform:

```
./configure --prefix=[installation-dir]
make && make install
```

Additionally is one important configuration option to be noted.

- --disable-sherpafill

This option disables the default fill behaviour of MCgrid which takes into account the PDF structure of event weights originating from the SHERPA [1] event generator and enables the generic fill mode. You should enable this if you wish to use a different event generator with MCgrid and the PDF dependance of the supplied weights is via a simple multiplicative factor as described in [3].
If you want to use MCgrid in conjunction with SHERPA, you should at least use the following options when configuring it:

```
./configure --enable-rivet=[installation-dir of rivet] \
  --enable-hepmc2=[installation-dir of hepmc2]
```

Example SHERPA event generation configuration files ("run cards") used for grid creation with MCgrid can be found on the MCgrid hepforge wegpage.

2.2 Linking with a Rivet analysis.

To include MCgrid functionality in your analysis, you should supply the usual rivet−buildplugin script with additional flags providing the paths to the package. The installation procedure provides the system with a pkg−config.pc file to provide path information. A typical command for building a Rivet plugin would therefore be:

```
rivet−buildplugin [RivetAnalysis.so] [RivetAnalsis.cc]  \
   $(pkg−config mcgrid −−cflags) $(pkg−config mcgrid −−libs)
```

An set of example analyses and a typical Makefile are provided on the MCgrid hepforge webpage.

3 Implementing MCgrid tools in an analysis

3.1 Required modifications

To use the MCgrid tools, there are three modifications that must be made to your Rivet analyses to enable the package. Firstly the MCgrid headers should be included at the top of the analysis code:

```
    #include "mcgrid/mcgrid.hh"
```

Secondly, in the analysis phase of the code, the MCgrid event handler must be called for every event passed to Rivet. This is done by adding the following line to the start of the analysis phase:

```
    MCgrid::PDFHandler::HandleEvent(event, histoDir());
```

Finally in the finalise phase, the event handler must be cleared and exported by adding the following as the final line in the finalise phase:

```
    MCgrid::PDFHandler::CheckOutAnalysis(histoDir());
```

With these modifications you have a barebones MCgrid enabled Rivet analysis. An example of this minimal modification, MCGRID_BASIC is given in the examples package.

3.2 Booking subprocess PDFs

After the basic modifications, you need to specify a subprocess PDF combination. This details which QCD subprocesses contribute to the full process in question, and how the individual parton-parton subchannels are categorised into said subprocesses. This information is provided by APPLgrid lumi_pdf config files or by fastNLO steering files. For the details of how these files may be obtained from SHERPA or constructed by hand, refer to Appendix A.

Create a subprocessConfig instance to specify a subprocess combination for each process in the analysis:

```
MCgrid::subprocessConfig subproc(configFileName,
                                 beam1Type, beam2Type);
```

Where configFileName is a std::string providing the filename of the subprocess config name. APPLgrid lumi_pdf config files should be installed to the APPLgrid share folder. fastNLO steering files should be located in the working directory. beam1Type↩ and beam2Type specify whether the beam types used in the config file, either for proton or anti-proton beams where the quark flavours should be switched when performing a fill. For an LHC analysis an example call for use with APPLgrid would be:

```
const string PDFname("atlas_inclusivejets.config");
MCgrid::subprocessConfig subproc(PDFname,
                                 MCgrid::BEAM_PROTON,
                                 MCgrid::BEAM_PROTON);
```

Or when using fastNLO for a Tevatron analysis where the second beam is antiprotons in the event generation:

```
const string PDFname("cdf_zrapidity.str");
MCgrid::subprocessConfig subproc(PDFname,
                                 MCgrid::BEAM_PROTON,
                                 MCgrid::BEAM_ANTIPROTON);
```

An important config file that is provided by default in APPLgrid is the basic.config file.[1] In this subprocess config all 121 partonic channels are active. If you do not have a specific subprocess identification file for your analysis, it is always possible to use this subprocess PDF. However the resulting grid will be significantly larger than a typical grid produced with subprocess identification enabled.

[1]A corresponding fastNLO steering file can be generated from basic.config using the createFastNLOSteering.py script located in the MCgrid tarball.

A few examples of subprocess config files are provided in the `examples/subproc` and in the `examples/fastnlo−steerings` folders.

3.3 Initialising APPLgrids and fastNLO tables in your analysis

With the subprocess PDFs initialised it is time to set up the interpolating grids themselves. Firstly the `Rivet` analysis should be implemented and checked as in a standard analysis using only the histogram classes. Once the user is satisfied with the analysis, they should add to the analysis class their grid classes.

For every `Rivet` histogram for which the user wishes to construct a corresponding `APPLgrid` and/or `fastNLO` table, they should add an `MCgrid::gridPtr` instance to the analysis class' private attributes. For example:

```
private:
   /// Rivet Histograms
   Histo1DPtr _h_distribution;
   Histo1DPtr _h_xsection;

   // Interpolation grids
   MCgrid::gridPtr _g_distribution;
   MCgrid::gridPtr _g_xsection;
```

The naming of the `gridPtr` objects is left to the user, however it's recommended that they explicitly reference the histogram they are to be based upon.

Now, in the `init()` phase where your histograms are initialised, the `MCgrid::gridPtr` instances should also be initialised with the following function:

```
   MCgrid::gridPtr MCgrid::bookGrid(
       // Corresponding Rivet histogram
       const Rivet::Histo1DPtr hist,
       // Result of Rivet histoDir() call
       const std::string histoDir,
       // Either an APPLgrid or a fastNLO config object
       T config
   );
```

Where the struct `config` specifies the configuration of the interpolation grid. `APPLgrid` and `fastNLO` need different input to initialise their interpolation grids, so there is a config struct for both of them:

```
// Use this to book an APPLgrid
MCgrid::applGridConfig applgrid_config(
    // LO of the process
    const int lo,
    // A subprocess config file name
    const subprocessConfig subprocess_config,
    // A grid architecture config object
    const applGridArch applgrid_arch,
    // The minimum x value of the grid
    const double x_min,
    // The maximum x value of the grid
    const double x_max,
    // The minimum scale of the grid
    const double q2_min,
    // The maximum scale of the grid
    const double q2_max,
);

// Use this to book a fastNLO table
MCgrid::fastnloConfig fastnlo_config(
    // LO of the process
    const int lo,
    // A subprocess config file name
    const subprocessConfig subprocess_config,
    // A grid architecture config object
    const fastnloGridArch fastnlo_arch,
    // The center-of-mass energy of the events
    const double com_energy
);
```

Where the struct `subprocess_config` can be created as described in sec. 3.2. The `applGridArch` and `fastnloGridArch` structs specify the architecture of the grid interpolation. Again, there is one constructor for each implementation:

```
    // Use this to specify an APPLgrid architecture
    MCgrid::applGridArch applgrid_arch(
        // Number of grid points
        const int nX,
        const int nQ2,
        // Order of interpolation
        const int xOrd,
        const int Q2Ord
    );

    // Use this to specify an APPLgrid architecture
    MCgrid::fastnloGridArch fastnlo_arch(
        // Number of grid points
        const int nX,
        const int nQ,
        // Interpolation kernels
        std::string const xKernel,
        std::string const qKernel,
        // Distance measures
        std::string const xDistanceMeasure,
        std::string const qDistanceMeasure
    );
```

For possible values for the kernels and distance measures of `fastNLO` tables, see the `mcgrid.hh` header. Some examples for architectures are predefined in the `MCgrid` namespace: `lowPrecAPPLgridArch`, `medPrecAPPLgridArch`, `highPrecAPPLgridArch` or `lowPrecFastNLOArch`, `medPrecFastNLOArch`, `highPrecFastNLOArch`.

As an example, consider the construction of an `APPLgrid` for a Drell-Yan Z-rapidity analysis where events are generated with a fixed scale of M_z^2:

```
    // Grid architecture and configuration
    MCgrid::applGridArch   arch(50,1,5,0);
    MCgrid::applGridConfig config(0, subproc, arch,
                                  1E-5, 1, 8315.18, 8315.18);

    /// Book histogram and corresponding grid
    _h_xsection = bookHisto1D(1, 1, 1);
    _g_xsection = MCgrid::bookGrid(_h_xsection, histoDir(),
                                   config);
```

3.4 Filling and finalising the grids

In the `analyse` phase of your `Rivet` analysis, both the histograms and grid classes must be populated after the experimental cuts and analysis tools are applied as usual.

Once you have performed your event selection and are ready to fill a histogram, you simply have to fill the corresponding `gridPtr` also.

```
    _h_distribution->fill(coord, weight);   // Histogram fill
    _g_distribution->fill(coord, event);    // grid fill
```

Here `coord` specifies the value of the histogrammed quantity for that event, `weight` is the usual event weight and `event` is the `Rivet::Event` object passed to the `analyse` method.

Finally the normalisation of the grids should be set, and the `APPLgrid .root` files or `fastNLO .tab` files exported for use. This is accomplished in the `finalise` phase of the analysis. For the normalisation the treatment of the grids is once again analogous to that of the histograms[2]. For each histogram/grid pair to be scaled the following should be called:

```
    // Histogram normalisation
    scale(_h_distribution, normalisation);
    // Grid normalisation
    _g_distribution->scale(normalisation);
```

And finally the grids should be written to file.

[2]It should be noted that in MCgrid, a function analogous to the `Rivet` `normalise` method is not provided. This is an intentional choice, as under PDF variation the resulting predictions cannot be guaranteed to be normalised to one. The user should utilise the scale method as described.

```
    _g_distribution->exportgrid();
```

The filename of the grid will be based automatically upon the id of the corresponding histogram.

3.5 Active flavours

Call `setNumberOfActiveFlavors(n)` in the `init` phase of your `Rivet` analysis to change the number of active flavours used by `MCgrid` internally. This is important if you want to fill NLO events generated with Catani-Seymour subtraction, but the active flavours in the generation does not match the `MCgrid` default of 5 (i.e. only the top quark is excluded)[3]. For example, if you set the bottom quark to be massive in `SHERPA`, add the following line to the `init` phase of your `Rivet` analysis to exclude the bottom quark:

```
    MCgrid::setNumberOfActiveFlavors(4);
```

Note that `MCgrid` assumes that the active flavours are the first n elements of the following list: up, down, strange, charm, bottom, top.

4 Executing your MCgrid/Rivet analysis

As is typical with the `APPLgrid` and `fastNLO` package, to fill their produced grids two runs of the analysis must be performed. The first, or phasespace fill run, determines the relative statistics of each partonic channel in the process such that their statistical samples may be combined correctly, and also establishes the boundaries of the x, Q^2 phase space for each of the interpolation grids as explained in [5] and [6]. The second run actually populates the grids with the Monte Carlo weights. It is therefore typically sufficient to perform a run with a smaller but representative event sample for the phase space run, and only run the full event sample for the full fill.

The modified `Rivet` analysis produced with `MCgrid` utilities can be uses as a completely conventional `Rivet` analysis, running over `HepMC` event record files, or indeed streamed via a `FIFO` pipe or straight from an event generator.

The first run of the analysis will produce an `MCgrid` results directory in the current working directory, and export an event count file along with the optimised `APPLgrid/fastNLO` phase space grid to `mcgrid/<analysis name>/phasespace/`. The second, fill run, looks for these files and reads them in preparation for the fill. The final `APPLgrid/fastNLO` files are exported into the directory `mcgrid/<analysis name>/` at the end of the second run.

[3]This is also the default of `SHERPA`.

Subsequent runs would fill more grids. A counter suffix in the name of the exported file is automatically used to prevent overwriting of existing grids.

4.1 Environment variables

The behaviour of MCgrid can be customised using the following environment variables:

- `MCGRID_DISABLED` If this variable is defined and not set to "0", "false" or an empty string, then `MCgrid` is disabled, i.e. will do nothing. Use this to temporarily disable `MCgrid` without switching or modifying the analysis.
- `MCGRID_OUTPUT_PATH` Use this variable to customise the path used by `MCgrid` for exporting final grids. It can be relative or absolute. The default output path is `mcgrid/`.
- `MCGRID_PHASESPACE_PATH` Use this variable to customise the path used by `MCgrid` for reading and writing phasespace information and final grids. It can be relative or absolute. The default phasespace path is `MCGRID_OUTPUT_PATH`.

4.2 Parallelisation and grid combination

In the case of very large statistics Monte Carlo runs, it may be advantageous to parallelise the calculation to provide a substantial speed boost in the generation of the `APPLgrid/ fastNLO` files. It should be noted however that the phase space information provided from the first run must be used by all subsequent parallel runs to ensure the correct combination of the final grids. Therefore the phase space run may not be parallelised. However, as mentioned previously, a representative sample rather than the full event record may be used to determine the phase space information. This data may then be provided to several parallel fill runs. Combination of the produced grids is done by the standard tools provided with the `APPLgrid/fastNLO` packages, `applgrid−combine` and `fnlo−merge`.

References

[1] T. Gleisberg, S. .Hoeche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert and J. Winter, JHEP **0902** (2009) 007 [arXiv:0811.4622 [hep-ph]].

[2] G. P. Salam and J. Rojo, Comput. Phys. Commun. **180** (2009) 120 [arXiv:0804.3755 [hep-ph]].

[3] L. Del. Debbio, N. P. Hartland, S. Schumann, [arXiv:1312.4460 [hep-ph]].

[4] M. Dobbs and J. B. Hansen, Comput. Phys. Commun. **134** (2001) 41.

[5] T. Carli, D. Clements, A. Cooper-Sarkar, C. Gwenlan, G. P. Salam, F. Siegert, P. Starovoitov and M. Sutton, Eur. Phys. J. C **66** (2010) 503 [arXiv:0911.2985 [hep-ph]].

[6] D. Britzger *et al.* [fastNLO Collaboration], arXiv:1208.3641 [hep-ph].

[7] T. Gleisberg and S. Hoeche, JHEP **0812** (2008) 039 [arXiv:0808.3674 [hep-ph]].

[8] F. Krauss, R. Kuhn and G. Soff, JHEP **0202** (2002) 044 [hep-ph/0109036].

A Subprocess Identification Scripts

The subprocess identification config files of `APPLgrid` list the partonic components of each of the N_{sub} distinct subprocesses present in the calculation. For each subprocess there are a set of $N_{pair}^{(isub)}$ parton-parton pairs that contribute to it. The configuration file denotes these as so:

```
[Flag for removal of CKM matrix elements = 0 or 1]
0 [pair1] [pair2] .. [pairN_0]
1 [pair1] [pair2] .. [pairN_1]
..
[Nsub]
```

Where the pairs are denoted by integer pairs in the LHA basis, neglecting the top quark:

$\bar{b}$	$\bar{c}$	$\bar{s}$	$\bar{u}$	$\bar{d}$	g	d	u	s	c	b
-5	-4	-3	-2	-1	0	1	2	3	4	5

The `APPLgrid` package searches for these configuration files in it's `share` path which can be found by using:

```
applgrid-config --share
```

In `MCgrid` the first parameter in the configuration should always be set to zero, as the `APPLgrid` functionality of CKM matrix element variations is not available in the package. However the loss of this feature will only impact calculations where the CKM elements enter only in the vertex connecting the two incoming partons.

As an example configuration, consider a hypothetical process who's only partonic sub-processes consist of $U\bar{U}$ and gD channels where U denotes an up-type quark and D a down-type. The configuration file for `APPLgrid` would then be:

```
0
0 2 -2 4 -4 # UUBar
1 0 1 0 3 0 5 # gD
```

An important point is that these configuration files refer to the numbering scheme for *proton* distributions. In the case where the user wishes to use a calculation with an initial state antiproton beam, the signs on the antiproton beam flavours should be flipped. For example, for a $p\bar{p}$ beam our previous configuration file would become:

```
0
0 2 2 4 4 # UUBar (ppbar)
1 0 -1 0 -3 0 -5 # gD (ppbar)
```

Such that the correct PDF treatment of the antiproton beam is taken into account. Examples of subprocess configurations for both pp and $p\bar{p}$ beams can be found in the examples package.

A simple python script is provided in the `MCgrid` package for the automated generation of `APPLgrid` `lumi_pdf` configuration files from the output of either of the two matrix element generators present in `SHERPA`, `COMIX`[7] and `AMEGIC++`[8]. The user may choose to either construct the appropriate configuration file by hand or make use of this script.

The tool can be found at `mcgrid/scripts/identifySubprocs.py`.

The operation of the identification script is straightforward. Taking the `SHERPA` run card which you will use for the full event generation run, you should run with only a handful of events, which is sufficient for the generation of the process information required to form the subprocess configurations. You should then run the script with the produced process database as an argument. The process database is typically found in the generated `Process` directory.

```
identifySubprocs.py --beamtype=[pp/ppbar/pbarp] Process.db
```

Where the argument specifies the beam types used in the event generation. This ensures that the quark flavours are mapped correctly to the proton PDF basis. This script will then produce a `subprocs.config` file to be used in your `MCgrid` analysis.

For `fastNLO` you must provide a steering file in the working directory with the subprocess identification. The format is similar to the way described above for `APPLgrid`. Example steering files are provided in the examples package. A python script is provided in the `MCgrid` package for the automated translation of a `fastNLO` steering file from an `APPLgrid` configuration file, which in turn can be automatically generated from the processes written out from `SHERPA` as described above.

The tool can be found at `mcgrid/scripts/createFastNLOSteering.py` and is used as follows:

```
createFastNLOSteering.py --beamtype=[pp/ppbar/pbarp] \
    subproc.config
```