Physics 450 – Computer Programs for Parton Calculations

On the Physics 450 course web site, there is a file `collider.tar.gz` that contains computer programs for computing cross sections at hadron colliders. This note gives some basic documentation on these programs. If you have any difficulty following these instructions, please email the professor for help.

How do I unpack the distribution and run the programs?

Log into a UNIX machine (elaine at Stanford, flora at SLAC, or your LINUX or Mac OS X laptop). Create a new directory by typing `mkdir collider`. Use your Web browser to download the file `collider.tar.gz`, and store it in this directory. Type `cd partons` to enter the directory, and type

```
gunzip collider.tar.gz
tar -xvf collider.tar
```

The distribution should then unpack, and you will find the directories:

```
collider/ engine/ examples/ processes/
```

The example programs are in the directory `examples`. You can modify these files with your favorite text editor. To compile and run one of the programs given, for example, `plotxf.cpp`, type `make plotxf.ex`, then `plotxf.ex` (or `./plotxf.ex`). The compilation is controlled by the `makefile` in the `examples` directory. If you create a program with a new name, it should compile and run in the same way. If you create new *process* classes, for new parton reactions, you should add these classes to the makefile in the line labelled `processes`. On Mac OS X systems, you might have to change `g++` to `c++` at the top of the `makefile` to get the programs to compile.

The programs in this distribution are written in `C++`. The directory `engine` contains the basic reference classes and the Monte Carlo integrator (based on the program VEGAS, by Peter Lepage). The directory `collider` contains the programs that compute parton distribution functions and collider cross sections. At the moment, the only parton distributions available in the package are the fits by Martin, Roberts, Stirling, and Thorne (see, e.g., hep-ph/0110215). The directory `processes` contains the programs that compute the differential cross sections for parton-parton reactions.

Hopefully, you can just leave the `C++` classes in `engine` and `collider` alone. For some of the exercises in the course, you will have to write new classes representing parton reactions. Such classes should be derived from the class `process` defined in the file `process.h` in the `process` directory. This directory also includes the class `twojet`, which gives an implementation of parton-parton scattering to two partons (e.g., $q\overline{q} \to gg$). You can take this as a model for other `process` classes.

The output of the programs is in the form of gnuplot graphics files. To view a gnuplot file, type `gnuplot` to open the gnuplot command-line environment. Then, to view the file `A.gp`, type `load 'A.gp'`. To print the file, you can either take a screen shot or convert it to

postscript. To convert the file to postscript, here are the steps. In the gnuplot environment, type:

```
> set terminal postscript color
> set output 'A.ps'
> load 'A.gp'
```

The plot `A.gp` will be transferred to a postscript file `A.ps`.

What do the example programs, in the `examples` directory, actually do?

The program `plotxf.cpp` produces a plot of the evolution of the up quark parton distribution in the proton, similar to Peskin and Schroeder, Fig. 17.21, and also a plot of the evolution of the gluon distribution in the proton. (The up quark plot differs in detail from Fig. 17.21 because a different fit to the deep inelastic scattering data was used.)

The program `Tevatronjets.cpp` produces a plot of the p_T distribution of 2-jet events in $p\overline{p}$ collisions at 2 TeV in the center of mass, and also a plot of the distribution of 2-jet invariant mass in these events. The latter plot will be similar to the plot in Peskin and Schroeder, Fig. 17.13. The program also produce a integrated histogram of the number of events with a jet with p_T greater than a given value.

What is going on in `plotxf`, and how can I change it?

The core working part of `plotxf` is the `for` loop (line 28 or line 59) that steps over `x` from 0 to 1, evaluates the parton distribution, and sends the data to a gnuplot file. The parton distributions are computed by the C++ class `P` of the type `MRST`. The parton distribution for the up quark in the proton is accessed as

```
P.xfu(x,Q)
```

Note that this function returns $xf_u(x,Q)$, which is smoother than $f_u(x,Q)$. The other parton distributions in the proton are returned by

```
P.xfd(x,Q) P.xfs(x,Q) P.xfc(x,Q) P.xfb(x,Q) P.xfg(x,Q)
P.xfubar(x,Q) P.xfdbar(x,Q) P.xfsbar(x,Q) P.xfcbar(x,Q) P.xfbbar(x,Q)
```

The material in lines 11–14 and 41–45 define `gnuplot` objects, which create gnuplot graphics files. For example, line 11 creates a `gnuplot` object `Gu`, writing to a file `uquarks.gp`, with a plot with x limts 0 to 1 and y limits 0 to 1. Line 44 is the command that sets the `gnuplot` object `Gg` to have a plot with a log scale in the x direction. (Note that the corresponding `for` loop has exponential steps.) Please examine the lines 27, 32, 36. For each curve in the gnuplot, you must `open` and `close` the `gnuplot` object, and when you are done with the object, you must `finish` it.

What is going on in `Tevatronjets`, and how can I change it?

The key working part of `Tevatronjets` is a Monte Carlo integrator implementing a cross section formula:

$$\sigma = \int dx_1 \int dx_2 \int d\cos\theta_* \cdot f_1(x_1,Q) f_2(x_2,Q) \frac{d\sigma}{d\cos\theta_*}(\hat{s},\hat{t},\hat{u}) \qquad (1)$$

where f_1 and f_2 are parton distributions. The integrator is implemented in a `collider` object, two of which are defined in lines 60 and 62. The `collider` object depends on a collider type (`pp` or `ppbar`), a center of mass energy, a `partons` object (here, `Pset` defined in line 39), `cuts` object (here, `Call` and `Cone` defined in lines 37 and 38), and a `process` object (here, `TJ`, defined in line 36).

The differential cross section in (1) is computed by the `process` object. Here, that object is belongs to the `twojet` class defined in `twojet.h, twojet.cpp`, which implements parton-parton scattering.

The region of integration in (1) is specified by the `cuts` object. The class `cutforall` in line 37 places the restriction that all final-state particles have p_T greater than `ptmin` and $|\eta|$ less than `etamax`. The class `cutforone` in line 38 imposes this requirement for either, but not necessarily both, final particles. It is appropriate to use the first kind of cut for a two-jet measurement and the second for a one-jet measurement.

The material in lines 43-58 illustrates an additional feature of the `process` class. The class methods used here either allow all combinations of initial-state partons, or restrict the calculation to q and $\overline{q}$ initial states only, to qg and $\overline{q}g$ initial states, or to gg initial states.

Lines 66 and 68 illustrate the `histogram` class in the package. A histogram is created by specifying the lower limit, the upper limit, and the bin size. A histogram with the appropriate number of bins, plus an overflow and an underflow bin, will be created. To add a data point to the histogram `H1`, call `H1.add(data)` or `H1.add(data, weight)`, as illustrated in lines 92 and 118. In this program, the Monte Carlo integrator returns points with weights from the integration region (lines 90 and 116), and these weights are then used to construct the histograms. The integral histogram works in the same fashion.

Before selecting points with the Monte Carlo integrator, it is best to let the integrator sample the integrated function and adapt itself for best efficiency. This is the purpose of lines 81 and 108. The argument of the method `prepare` used in these lines is the number of calls to the integrand used in the adaptation process.

The `collider` integrator returns a point as an object `L` of the type `LVlist`, belonging to the collider class, which contains the 4-vectors of the final-state particles. Notice how methods of `L` are called in lines 80 and 106. Some of the methods available for `L` are:

`L.E(i), L.p(i), L.cost(i), L.pt(i), L.eta(i), L.mass(i), L.mass(i,j)`

These methods take arguments `i,j = 1,2` for the two final particles. A full definition of the `LVlist` class is given in the file `engine/LClasses.h`.

Finally, lines 99, 126, and 129 write out the histograms to gnuplot files. The method used `printScaledHistogram`, rescales the histogram so that it is normalized to the value given in the second argument. The value given in the third argument specifies the color of the line with which the histogram is drawn on the plot.

How do I create my own `process` class?

To add more processes to this framework, define each one as a new `C++` class derived from `process`. To set up such a class, create files `myprocess.h` and `myprocess.cpp`, in the

processes directory. The file myprocess.h should be a copy of the file twojet, with the name changed from twojet to myprocess in all relevant places. The file myprocess.cpp should define the class constructor and the function that computes the cross sections. A model for the class constructor can be found in twojet.cpp, line 4. The arguments of the process constructor are the masses of the final-state particles. A model for the crosssections(i,j) method can be found in the rest of this file. You should be aware that (1) crosssections(i,j) is used only for cases $i \geq j$, so it only needs to be defined for these cases; (2) because the cross section is integrated over $-1 < \cos\theta < 1$, a cross section for a process with identical particles in the final state should be divided by 2.

At the time that crosssections(i,j) is called, many kinematic variables of the collision have already been defined. These include E, p, E1, E2, M1, M2, cost, sint, shat, that, uhat, respectively, the initial CM particle energy, the final CM particle momentum, the two final CM particle energies, the two final CM particle masses, the values of $\cos\theta$ and $\sin\theta$ in the CM system, and the values of the three Mandelstam variables. A quantity Q which measures the hardness of the reaction has also been defined, by

$$Q^2 = \frac{1}{4}(M_1 + M_2)^2 + p_T^2 \ . \tag{2}$$

Any of these variables can be used in the program that computes the cross sections.

The file engine/hepdata.h, which is included in twojets.h, defines a number of useful quantities that can be used in cross section calculations, including mw, mz, mt, alpha, alphas(Q), sw, cw; the last two variables are $\sin\theta_w$, $\cos\theta_w$. This file also defines fb to be the conversion factor from $(\mathrm{GeV})^{-2}$ to fb. For clarity, all masses in the current program are given in GeV and all cross sections in fb; it might be good to stick to these conventions in your new classes.

A new class derived from process can be used in a collider cross section computation exactly as twojet is used in the program Tevatronjets.cpp, line 26. You will need to add the new process to the makefile to compile the the new program.