Practical 6: Crosschecking our Generation and Analysis

Today we will generate data in Python and use both our analysis and MadAnalysis to see if we get the same cross section and angular distribution. We will also try generating some data in MadGraph, and run both analyses to see if they match.

Firstly, lets install MadAnalysis. To ensure everything runs smoothly, lets create a new virtual environment. First, we navigate to the `MG5_aMC_v3_5_7` folder and run

```
python3 -m venv venv
```

and activative it using

```
source venv/bin/activate
```

We install the dependancies

```
pip install six
```

Let's run MadGraph

```
./bin/mg5_aMC
```

and run

```
install MadAnalysis5
```

For this practical, it is best to place our directory `ComputationalHEP` into the MadGraph directory `MG5_aMC_v3_5_7`. This can be done with the `mv` command.

- *Our first task in order to be able to compare the MadAnalysis with our analysis is to ensure both programs are using the same cuts. This means we should implement the cut based on the value of ΔR_{ij}, rather than $\cos\theta$, in our code in the same way MadGraph does.*

We can then run both programs to see if we get the same cross-section. This should hopefully agree. However, this doesn't prove our code to be correct - we may have two compensatory mistakes in our generation and analysis that negate eachother to give us the same answer as MadGraph. To make sure this isn't the case, we can generate data with our code then analyse in MadAnalysis, and generate data with MadGraph and then analyse with our code. If these match, then this is a great sign!

- *So, our next task will be to write a function to import data and analyse generated by MadGraph.*

Recall that we can obtain a dataset from MadGraph by running

```
generate e+ e- > d d~ g / z
output quicktest
launch
```

This will generate a dataset and compute the cross section. We can then

```
cd quicktest
cd Events
cd run_01
ls
```

and should see a file `unweighted_events.lhe.gz`. This contains the data generated by MadGraph, in compressed form. Inside our `run.py`, we should add a new case to `__main__` for our analysis of MadGraph files. Already, there is a parser defined, `parser_rratio_analyze_events_experiment` You should create a new experiment file, similar to `rratio_differential.py`, where instead of generating phase space points in `integrand()`, you import the MadGraph data. Of course, we will need to parse the data in a form that is compatible with our code. The data is stored in a `.lhe` file, which is a complicated format. Fortunately, we can take advantage of MadGraph's own parser. To do this, take a look at `utils/lhe_parser.py`. Note the code

```
    try:
    if os.getenv('MG_ROOT_PATH') is not None:
        sys.path.insert(0, str(os.getenv('MG_ROOT_PATH')))
    else:
        sys.path.append('../')
    from madgraph.various.lhe_parser import EventFile, Event, Particle  # type: ignore
except Exception as e:
    print(f"Could not load Magraph lhe parser: {e}. Specify madgraph root path
    with env. variable MG_ROOT_PATH.")  # nopep8
    sys.exit(1)
from CHEP.utils import CHEP_TEMPLATES
```

This suggests that if we place our `CHEP` folder inside the MadGraph folder, our code will be able to use the built-in Madgraph parser in order to interpret the file.

We then have a daughter class of MadGraph's own `EventFile`,

```

class CHEPEventFile(EventFile):
    def __init__(self, file_path, *args, mode='r', **kwargs):
```

```
        super().__init__(file_path, *args, mode=mode, **kwargs)
        if mode == 'w':
            self.banner = EventFile(os.path.join(CHEP_TEMPLATES,
            "template_events_file.lhe"), mode='r').get_banner()  # nopep8
            self.banner.write(self, close_tag=False)
            self.write("\n"*10)
        self.banner = self.get_banner()
```

We can then run our code with, for example 10 iterations,

```
    python3 run.py rratio_differential -ni 10
```

Once this is implemented, running our analysis on the MadGraph data should give us a similar result to the data we generated ourself!

- *The final task will be to analyse our generated data with MadAnalysis.*

Note that in `rratio_differential.py` we can see a line of code already

```
        event_file = CHEPEventFile("./epem_ddxg.lhe", mode='w')
```

Note that the `integrand()` method takes this as an input argument. We can see in this method that if the phase space point passes our cuts, we already write to the event file. So, after running our code we should already see this file in our directory. There is also code

```
        unweighted_event_file = CHEPEventFile("./epem_ddxg.lhe", mode='r')
    unweighted_event_file.unweight("./unweighted_epem_ddxg.lhe")
    unweighted_event_file.close()
```

and thus we should also see an unweighted version in the directory.

Inside `ComputationalHEP/CHEP/templates`, you should find a template specification file `run.ma5` This is a file that we can run with MadGraph to automatically generate our cross sections and analysis without having to manually type each command in MadGraph. We should open this file

```
    vim ComputationalHEP/CHEP/templates/run.ma5
```

Inside there are lines

```
    import ./epem_ddxg.lhe as weightedEvents
  import ./unweighted_epem_ddxg.lhe as unweightedEvents
```

By changing these to directory and name of our generated files (if they're not already in the current directory), we can import and run MadAnalysis on them with

```
./bin/mg5_aMC ComputationalHEP/CHEP/templates/run.ma5
```

Hopefully, you get cross-sections and histograms that look like those produced by your analysis!