ALP Sensitivity Study: Software Files

Sophie Middleton

June 2024

1 Samples

Using the MadGraph http://madgraph.phys.ucl.ac.be/ generator two sets of samples have been made:

- Primakoff production;
- Photon fusion production.

The physics involved is described in the accompanying document. These were made with help of a theoretical physicist, I can talk you through the production code, but its beyond the scope of what you need to know at this point.

The resulting events are stored as LHE file (Les Houches file format, a common generator output). You can open the files in a text editor to inspect their content. There are a few things to check. Scroll through one of the files, look for the following block:

```
<MGGenerationInfo>
  # Number of Events          :          50000
  # Integrated weight (pb)    :          631.05
</MGGenerationInfo>
```

what this tells us is that this file had 50000 generated events and MadGraph calculated the cross-section to be 631.05 pb. If you are not familar with the term "cross-section" it simply acts as a measure of the probability that a specific process will take place and its common unit is that barn(https://en.wikipedia.org/wiki/Barn_(unit). Here the units are pb - so picobarns. This doesnt really mean much at this point, but when we start looking at all the samples, relative difference will start to tell you which are more likley to occur in our experiment.

For the Primakoff files you will see that these were made for a specific coupling

```
# ALP width (for gag = 1e-3/GeV) = 3.9788735772973844e-11
```

The accompanying detailed document explains the significance of these values.

1.1 Photon fusion

The photon fusion samples are create with exactly 50K events for each of the following ALP masses:

- 10 MeV/c^2
- 50 MeV/c^2
- 100 MeV/c^2
- 150 MeV/c^2
- 200 MeV/c^2

- 300 MeV/c^2
- 400 MeV/c^2
- 500 MeV/c^2

The folder names reflect the mass choices (or you can check in the LHEs to make sure).

1.2 Primakoff

The Primakoff samples are made from a set of photons collected from a beam simulation of LDMX with 8GeV electrons coming in. All the photons produced at the target are collected and these are used to generate the ALPs. Due to the way we simulate this type of production, there are a variable number of events per ALP mass. These Primakoff events were made using code developed by Nikita Bilnov: https://github.com/nblinov/primakoff.

- 10 MeV/c^2 -
- 50 MeV/c^2
- 100 MeV/c^2
- 150 MeV/c^2
- 200 MeV/c^2
- 300 MeV/c^2
- 400 MeV/c^2
- 500 MeV/c^2

2 Reading LHEs to Python

Back in 2021 I made some scripts that could do this for another study: https://github.com/sophiemiddleton/ReadLHE/blob/master/lhereader.py I will need to test these for the present scenario.

Here is an example for the ALP samples: https://github.com/sophiemiddleton/ReadLHE/blob/master/ALPExample_python.py

To run:

```
ldmx python3 ALPExample_python.py
```

To make this work with primakoff samples, we need to add a line to the header of each LHEs:

```
<LesHouchesEvents version="3.0">
```

and at the end:

```
</LesHouchesEvents>
```

This is because these files were custom made by Nikita's code.
A small explanation about an event:

```
<event>
6 2 1 -1 0.0072992700729927005 0.1081
      22 -1    0    0    0    0 +2.3501465321e-03 -3.3189296722e-03 +6.9058828125e+00
     623 -1    0    0    0    0 +0.0000000000e+00 +0.0000000000e+00 +0.0000000000e+00
     666  2    1    2    0    0 -3.2135711517e-02 -2.3073132941e-02 +6.8876407762e+00
     623  1    0    0    0    0 +3.4485858049e-02 +1.9754203269e-02 +1.8242036342e-02
```

```
        22   1     3     0     0     0  -1.6912474015e-01  -1.9666690768e-01  +2.7086827634e+00
        22   1     3     0     0     0  +1.3698902863e-01  +1.7359377474e-01  +4.1789580127e+00
</event>
```

The first column is the pdg number (22 = photon, 623 = Tungsten nucleus, 666= ALP). The second column says if it is outgoing or incoming. Here we have an incoming (-1) photon and a virtual (2) ALP and 2 outgoing photons. The other columns are the px,py,pz, energy etc.

3 Production Scripts

3.1 Making ALP Reconstructed .root samples

To pass the generator out put (LHE file) into realistic data products, similar to what we might see in the real LDMX detector we need to run the following command:

```
ldmx fire fireALP.py
```

(fireAlP.py is one of the config files in the directory)

This will take in the vertices (production points) of the photons from the generator and pass these through our particle detector. We simulate all the particle interactions they might have and the responses of our detector to. The end result will be something similar to what we could really detect in the experiment.

You will need to change the path to the file you want to reconstruct, output will be named root file, change name accordingly. Before running this command you should edit the block of code inside the fireALP.py file (using a text editor):

```
# edit these for each run:
filepath = "/Users/sophie/LDMX/software/test/"
infilename = "m10_prima.lhe"
outfilename = "m10_prima_reco.root"
nevents = 100
```

Make sure that the path corresponds to the location of the LHEs (which you can download from the google drive), edit the filenames accordingly and if you want to run over all events simply remove the nevents option. For the first pass, I would stick with 100 events.

3.2 Making the Secondary Photons for Primakoff

Here I describe how I collect the secondary photons, this is something you do not need to know but is here for completeness.

The following script makes a set of secondary photons which are then input into the Primakoff generator. A plane is placed at the back of the detector (a virtual plane, in the simulation). All photons are collected:

- photonProduction.py

This script should be run again using 'ldmx fire' is simply fires 8GeV electrons at the target and collects secondary photons. These are passed to the MadGraph based Primakoff generator.

4 Analysis

4.1 Analysing the Reconstructed ALP samples

We can develop and train a boosted-decision tree (BDT) to help us separate our signal (ALP) from the dominent background (PN). BDTs are commonly used in HEP as a form od multi-variant analysis. Essentially when one simple cut isn't enough to split a sample into definite background and definite signal, we combine a set of features which all have some smaller discriminant power, but when combined provide a powerful discriminant.

Part of the project will be to build this BDT, using features which we can reconstruct in the experiment's detector systems, focussing primarily on the HCAL.

Once you have the reconstructed ldmx-sw output, run the MakeRootTree.py file. This will produce an NTuple with the relevant fields for the BDT. The MakeRootTree script must be ran using pyROOT in the LDMX environment, for example:

```
ldmx python3 MakeRootTree.py --ifile reco.root --mass 100 --label primakoff
```

where reco.root is the output of passing the ALP LHE through LDMX-sw (using fireALP.py). The "mass" option is for the ALP mass you have simulated. This script outputs another root file with a selection of "features". These will be input to the BDT later on.

4.2 Scripts for other samples

To assess the sensitivity of our experiment we must think about how we might discriminate against the dominant photo-nuclear (PN) background. To do this discriminating between the photons from our visible ALP decay and the neutrons from the PN events is key.

We could look at samples of pure photons and neutrons to help guide our approach:

- fireGun.py - can use to create gun samples of neutron (particle = 'neutron') and photon (particle = 'gamma') of given energy (energy = X GeV). Run this script using the 'ldmx fire' command.

However, these are not realistic samples. We can create PN background samples, and reconstruct these in the experiment using the following script:

- firePN.py

These can bother be ran using the "ldmx fire" command within the LDMX environment.

5 BDT

5.1 TMVA

There are two scripts for running TMVA:

- TMVAClassification.C - training
- TMVAApplication.C - applying

To run the classification scripts on a given sample, you will need to edit the following line:

```
TString fnameSig = ``/<full path to files/signal.root";
inputSig = TFile::Open( fnameSig ); // check if file in local directory exists
TFile *inputBkg(0);
TString fnameBkg = ``/<full path to files/bkg.root";
```

to use the signal and background files with the full path. The input root files here should be output from MakeRootTree.py.

To apply the trained MVA to a given ALP or PN sample edit the line in the TMVAClassification-Application.C:

```
TString fname = ``<full path to file>/<name of ALP or PN file>.root";
```

again this file should be the output of the MakeRootTree.py script for that specific ALP sample. This TMVA script will output a file called output.root.

5.2 Python ML

I also looked at how things change if we use any of the python packages. Running the comparePython-MVA.py script shows the efficiency for several scikit-hep based algorithms. One extension could be to see if these provide better signal to background separation than the TMVA.