Université Catholique de Louvain

LPHY2131 : Particle Physics I

Laboratoire Numérique : masses des bosons Z et W.

Année académique 2015-2016

Professeurs :
Vincent Lemaitre
Christophe Delaere

Table des matières

1 Introduction

2 Simulation avec MadGraph5 et Delphes

2.1 Génération d'événements avec MadGraph5 pour le Z

Pour le moment, nous sommes dans la partie "simulation" du laboratoire, c'est à dire que nous voulons générer des événements numériquement avec des méthodes Monte Carlo afin des les confronter dans un second temps aux données du CMS de 2010. Pour ce faire nous allons utiliser plusieurs logiciels, et le premier est MadGraph5 qui permet de générer des évènements choisis. Nous aimerions simuler la production du Z à partir de collisions du type $p\ p$ (proton-proton), et pour cela il faut fournir à MadGraph5 les états asymptotiques de la collision. Nous allons nous concentrer sur les événements $[p\ p\ \rightarrow\ e-\ e+]$ et $[p\ p\ \rightarrow\ mu+\ mu-]$, c'est à dire les événements où un Z s'est désintégré en deux électrons ou deux muons.

La première chose à faire est d'ouvrir MadGraph5 en utilisant la commande *cd madgraph5*. Ensuite on rentre dans le programme en tappant *./bin/mg5_aMC*. Maintenant il faut générer le processus désiré, créer un dossier dans MadGraph et lancer la simulation. Voici les lignes de code pour simuler le processus $[p\ p\ \rightarrow\ e-\ e+]$ et créer un dossier nommé "ppee" :

generate p p > e- e+
output ppee
launch ppee

Si on veut générer du muon, il suffit de remplacer e par mu. Attention, avec ces commandes, tous les calculs et simulations de MadGraph se font au LO (leading order). Si on veut être plus précis et faire du NLO (next to leading order) il faut ajouter *[QCD]* à la fin de la première commande, celle qui génère l'évènement.

Si vous avez généré du LO, après avoir entré la commande "*launch*", MadGraph affiche une série de lignes numérotés (qui sont des options de simulation) et la première est

1 Run the pythia shower/hadronization : pythia=OFF

Pythia est le programme qui permet de générer des jets de particules due à la hadronization des quarks, il est donc important de l'activer afin d'avoir une simulation qui reproduit bien le nombre de jets. Pour ce faire il suffit d'entrer "1" dans la console. Maintenant MadGraphe affiche les mêmes lignes que précédemment mais la première se termine par "pythia = ON" ; vous pouvez donc appuyer sur "enter"pour continuer.

De nouveau une série de lignes numérotés s'affiche ; ce sont d'autres options du run que vous vous apprêtez à lancer. Entrer "2" dans la console pour avoir accès à la run card de l'évènement.

Vous pouvez modifier les paramètres comme vous voulez en fonction du nombre d'évènements que vous voulez, de certaines coupures éventuelles sur des paramètres (comme le moment transverse maximum, etc). Cependant, vous devrez toujours modifier l'énergie des faisceaux de protons. En effet, par défaut elles sont mises à 6500 GeV chacune, ce qui fait 13 TeV dans le centre de masse. Or en 2010 le LHC tournait avec 7 TeV au centre de masse, il faut donc remplacer 6500 par 3500 pour les deux faisceaux.

Cette run card qui s'affiche dans la console est ouverte avec l'éditeur de texte "vim" de linux, et ce n'est pas toujours très intuitif lorsqu'on n'y est pas habitué. Voici quelques règles de base :

- Pour pouvoir modifier des paramètres, il faut au préalable appuyer sur "a", et pour revenir au mode initial pressez "*esc*".

– Une fois les modifications terminées, pour quitter et sauvegarder il faut entrer " *:wq*".

Dans le cas où vous avez généré du NLO, vous n'avez pas besoin d'activer pythia après avoir appuyé sur "*launch*", par contre il faut toujours modifier la run card. Elle est légèrement différente de celle pour le LO mais fort similaire et fonctionne de la même manière.

Tout est donc prêt, appuyez sur "enter" et le processus se met en route. Cela peut prendre un certain temps.

2.2 Génération d'événements avec MadGraph5 pour le W

Pour le boson W il n'y a pas beaucoup de différence, si ce n'est que nous allons nous intéresser en même temps au W^+ et au W^-. Pour générer les deux, voici les lignes de codes à écrire :

generate p p > l- ve ~
add process p p > l+ ve

où l désigne un lepton, et doit être remplacé par e ou mu selon le processus désiré. Pour le reste, tout est identique à la marche à suivre pour le Z.

2.3 Création de fichier ROOT avec Delphes

Dans le dossier MadGraph, ouvrez le dossier qui porte le nom que vous avez choisi pour votre processus. Ensuite allez dans le dossier "Events" et entrer dans le dossier "run_x" où x est le numéro du run que vous voulez analyser. Il y a un fichier avec l'extension ".hep.gz", c'est celui là qui nous intéresse pour la suite. Nous appellerons ce fichier "test.hep.gz" pour simplifier. Il faut copier ce fichier et le mettre dans le dossier "delphes". Faites de même avec le fichier "CreateTree.C" qui se trouve dans le dossier "PP1". Vous pouvez les mettre dans d'autres dossiers et les exécuter en fonction, mais toutes les lignes de code qui suivent marchent telles quelles uniquement si les fichier sont simplement mis dans le dossier "delphes".

La première chose à faire est de créer un ficher "test.hep". Pour ça ouvrez le dossier "delphes" dans le terminal et entrez "*gunzip test.hep.gz*". Cela produit le fichier voulu. Maintenant nous allons faire tourner Delphes dessus. Avant toute chose, allez dans le dossier "PP1" et copiez le fichier "delphes_card_CMS_mod.tcl" dans le dossier "cards" de delphes. Pour créer un fichier Root (par exemple output.root) à partir de notre fichier "test.hep" en utilisant la carte de delphes voulue, entrez la commande :

./DelphesSTDHEP cards/delphes_card_CMS_mod.tcl output.root test.hep

Le fichier output.root est un Tree qui contient énormément d'informations, mais pour rendre la simulation compatible avec les données nous devons créer un Tree plus petit avec uniquement les variables qui nous intéressent. Pour cela, il faut faire tourner la macro "CreateTree.C" dans root. Cette fonction prend trois arguments : le nom du fichier à traiter, le nom du fichier de sortie et un entier x. Si vous mettez $x = 0$ cela veut dire que vous vous intéressez au Z, et si vous mettez $x = 1$ c'est pour le W. Voici les lignes de commandes pour exécuter ce code et sortir un fichier "tree.root" à partir de "output.root" :

root
.x CreateTree.C("output.root", "tree.root", 0)

Nous avons maintenant un arbre avec les même branches que celui contenant les données, ce qui facilitera grandement l'analyse en parallèle des deux.

3 Introduction à ROOT

Dans ce laboratoire vous allez beaucoup utiliser le logiciel ROOT. Il existe pas mal de documentation et de tutoriels sur internet. Cependant, voici tout de même quelques commandes et concepts de bases pour commencer.

Pour ouvrir ROOT il suffit d'entrer la commande *root* dans le terminal. Il faut bien veiller à se trouver dans le dossier qui contient les codes qu'on veut faire tourner. Par exemple, si on veut faire tourner un code qui se trouve dans le dossier "delphes", il faut ouvrir le répertoire delphes avec *cd delphes*.

Root est un programme qui permet, entre autre, de produire des histogrammes de différentes variables liées entre elles. Les objets que nous utiliserons le plus sont les arbres (Tree). Un arbre est constitué de plusieurs branches (et chaque branche peut contenir des feuilles). Par exemple dans notre problème, nous avons plusieurs variables d'intérêt, comme l'impulsion transverse et la masse invariante. Avec ROOT il est possible de mettre l'histogramme correspondant à chaque variable sur une branche d'un même arbre. Imaginons que nous ayons l'arbre "Tuto" avec les branches "PT" et "invMass". L'avantage de mettre cela dans un même arbre est que nous pouvons visualiser un histogramme en imposant des conditions sur une autre variable. Par exemple, si nous voulons voir la masse invariante en imposant que le PT soit plus grand que $30GeV$ et que la masse soit inférieure à $100GeV$ il suffit de taper :

Tuto->Draw("invMass","invMass< 100 && PT> 30")

Pour exécuter les opérations voulues, il y a plusieurs possibilités. Soit vous entrez directement les commandes dans la console, soit vous écrivez une macro.C que vous exécutez en entrant *.x macro.C* dans ROOT. Au niveau de la macro, il peut soit s'agir d'un script contenant une série d'instruction, soit d'un programme qui a la structure habituelle d'un programme C++. Peu importe l'option choisie, la programmation se fait toujours en C++.

Lorsque vous êtes dans ROOT, si vous voulez ouvrir le browser pour avoir accès aux différents fichiers, entrez simplement : *TBrowser t;*

Dans le dossier "tuto_root" dans "PP1" se trouvent des codes commentés montrant quelques opérations simples, comme lire un fichier, créer un histogramme, etc.

4 Application au Z et au W

4.1 Arbre

Voici les branches de l'arbre que vous aller analyser. L'arbre des données se nomme *data.root* et se trouve dans le dossier "PP1". La simulation créera un arbre semblable grâce au programme "CreateTree.C".

- nMuons : nombre de muons
- MuonsPt : impulsion transverse des muons
- MuonsEta : pseudo-rapidité des muons
- MuonsPhi : angle azimuthal des muons
- muonIsolation : rapport entre la calo-isolation et le pt des muons
- nElectrons : nombre d' électrons
- ElectronsPt : impulsion transverse des électrons
- ElectronsEta : pseudo-rapidité des électrons
- ElectronsPhi : angle azimuthal des électrons
- electronIsolation : rapport entre la calo-isolation et le pt des électrons

- nJets : nombre de jets
- JetsPt : impulsion transverse des jets
- JetsEta : pseudo-rapidité des jets
- JetsPhi : angle azimuthal des jets
- invMass : masse invariante des 2 e ou 2 mu de plus grand pt (si l'événement à été selectionné, c'est à dire si il y avait au moins deux e ou deux mu)
- deltaR_Muons : racine carrée de la somme des carrés des différences en η et ϕ des 2 mu de plus grand pt (si l'événement à été selectionné)
- deltaR_Electrons : racine carree de la somme des carrés des differences en η et ϕ des 2 e de plus grand pt (si l'événement à été selectionné)
- MET_pt : impulsion transverse manquante
- MET_phi : angle azimuthal de l'impulsion transverse manquante
- MET_eta : pseudo-rapidité de l'impulsion transverse manquante

Attention ! Pour le Z, la masse invariante de la branche "invMass" est correcte. Par contre, pour le W il va falloir calculer la masse transverse dans l'analyse des données, donc la branche "invMass" est sans importance dans ce cas.

4.2 Fonctions utiles

Vous trouverez ici une liste non exhaustive des classes et méthodes qui vous seront utiles (pour la documentation complète, rendez-vous sur https ://root.cern.ch/).

TFile Variable contenant un fichier.
- Open(string *nom du fichier*) : ouvre le fichier spécifié.

TTree Variable contenant un arbre.
- SetBranchAddress(string *nom de la branche*, type &*value*) : met l'adresse de la branche spécifiée de l'arbre à l'adresse de *value* (le type de *value* doit correspondre au type des données de la branche).

TH1F Variable contenant un histogramme uni-dimensionnelle de float.
- TH1F(string *nom de l'histogramme*, string *titre de l'histogramme*, int *nombre de bins*, float *valeur minimale*, float *valeur maximale*) : constructeur.
- SumW2() : ajoute une erreur sur le nombre dans chaque bin, qui est la racine carrée de ce nombre.
- Fill(float *value*) : remplit l'histogramme avec la valeur *value*.
- Scale(float *échelle*) : multiplie toutes les entrées de l'histogramme par l'échelle.
- SetLineColor(*couleur*) : change la couleur de la ligne de tracer du haut des bins (les couleurs sont *kRed*, *kBlue*, *kYellow*, ...).
- SetFillColor(*couleur*) : change la couleur de remplisage de l'histogramme (voir options de Draw).
- SetMaximum(float *max*) : ajuste la hauteur de l'axe vertical du graphe à la valeur *max*.
- Draw(string *options*) : trace l'histogramme, avec les options éventuelles *same* (trace l'histogramme sur le même graphe que le précédent), et *hist* (fait un histogramme plein).
- Fit(string *nom de la fonction*, string *options*) : fitte l'histogramme avec la fonction précisée (voir TF1) avec les options éventuelles *R* (utilise le même range pour le fit que le range de la fonction), *L* (fit en likelihood plutôt qu'en chi carré).

TF1 Variable contenant une fonction uni-dimensionnelle.
- TF1(string *nom de la fonction*, *formule*, float *valeur minimale*, float *valeur maximale*, int *nombre de paramètres*) : constructeur. Si la formule est expilicite, e.g. [0]*TMath : :Cos([1]*x), il n'est pas nécessaire de préciser le nombre de paramètres, sinon, soit on utilise une fonction

prédéfinie de Root, e.g. *gaus*, soit une fonction définie par l'utilisateur dans un fichier *fonction.C* (ne pas oublier alors la ligne gROOT->LoadMacro("fonction.C");!!!), auquel cas il faut préciser le nombres de paramètres.
- SetParameter(int *numéro du paramètre*, float *value*) : modifie la valeur d'un paramétre.
- GetParameter(int *numéro du paramètre*, float *value*) : accède à la valeur d'un paramétre.

TLegend Variable contenant la légende d'un histogramme.
- TLegend(float *x minimal*, float *y minimal*, float *x maximal*, float *y maximal*) : constructeur. Spécifie la position du cadre de la légende.
- AddEntry((TH1F *nom de l'histogramme*, string *nom affiché dans la légende*, string *options*) : ajoute une entrée, avec comme option *l* pour les histogrammes en lignes et *f* pour ceux pleins.
- Draw(string *options*) : voir TH1F.

THStack Variable superposant plusieurs histogrammes.
- THStack((string *nom du stack*,"") : constructeur (les "" vides semblent nécessaires).
- Add(TH1F *nom de l'histogramme*) : superpose l'histogramme à ceux déjà présents (le dernier stacké est au dessus).
- même fonctions que TH1F.