Introdução à Análise de dados em FAE **(01/12/2020)**

Relatório sobre CalcHEP

Professores: Sheila Amaral, Sandro Fonseca, Eliza Melo *Name:* Matheus Pereira Macedo de Sousa

Para a presente análise, foi requerido o download do modelo *Minimal Zp models* que pode ser encontrado no site `https://hepmdb.soton.ac.uk/index.php?mod=user&act=showmodel&id=38` ou na minha conta do github `https://github.com/teteumac/Analise-em-FAE/tree/main/calchep`.
O arquivo encontra-se compactado sob o nome `m38.20111125.124737.tar.gz`

É necessário fazer o envio desse mesmo arquivo para a pasta `models` do `calchep`. Supondo que ele esteja na pasta `Downloads` da sua máquina, com o terminal aberto faz:

```
mv $HOME/m38.20111125.124737.tar.gz $HOME/<diretorio>/calchep/models
```

Para descompacta-lo, entra no diretório `$HOME/calchep/models` e faz

```
tar -vzxf m38.20111125.124737.tar.gz
```

Por padrão, as versões dos `CalcHEP` sempre vem com alguns modelos dentro desta pasta `models` e estão todos eles enumerados. Você precisa alterar o nome dos arquivos, frutos da descompactação para o último número mais 1 da sequência que você tem na sua pasta, exemplo:
o produto pode ser algo assim...
`vars7.mdl`, `prtcls7.mdl`, `lgrng7.mdl`, `func7.mdl`.
Se o último numero da sua sequência de modelos forem 5
`vars5.mdl`, `prtcls5.mdl`, `lgrng5.mdl`, `func5.mdl`
então os arquivos com o final 7 tem que ser alterados para
`vars6.mdl`, `prtcls6.mdl`, `lgrng6.mdl`, `func6.mdl`.

Para acessar a interface do `CalcHEP` abra um terminal e faz:

```
cd $HOME/<diretorio>/calchep/
./calchep
```

Com o programa aberto, você vai escolher o modelo inserido, ele está com o nome `B-L (Full fast)`, clicando nele, clica em seguida em `Force Unit.Gauge= OFF` e troque para `Force Unit.Gauge= ON`, ativando o gauge unitário. Depois clica em `Enter Process`, isso vai te levar para uma outra janela e o *input* de entrada para o processo será

`Enter process: p,p -> m,M`

, ou seja, dois prótons indo em um múon e anti-múon. Aperte `Enter`. Outro ícone irá aparecer para ser preenchido. Nessa etapa ele pede para você compor um objeto composto chamado de p, você vai inserir todos os quarks, ou seja

`composite 'p' consists of: u,U,d,D,c,C,b,B,s,S,G`

. Em seguida ele pede para excluir as contribuições de outros processos, você digitará

`Exclude diagrams with A,Z,H1,H2`

. Após cumprir essas etapas, uma outra janela se abrirá. a sequência segue em. `Square diagrams` e em seguida `Symbolic calculations` e por fim `C-compiler`. Após finalizar essa etapa, outra interface do programa com uma nova janela se abrirá.
Na aba `Subprocess`, contém todos os subprodutos de decaimento dos quarks em múon e anti-múon. Cada um deles resulta numa seção de choque diferente. A seção de choque é encontrada em `1D integration` para uma energia de centro de massa igual a 6500,00 GeV. Abaixo está uma tabela contendo a seção de choque para cada subprocesso.

Subprocesso	Seção de choque(pb)
u U	$4,26 \times 10^{-7}$
U u	$4,26 \times 10^{-7}$
d D	$2,12 \times 10^{-6}$
D d	$2,12 \times 10^{-6}$
c C	$4,26 \times 10^{-7}$
C c	$4,26 \times 10^{-7}$
b B	$2,12 \times 10^{-6}$
B b	$2,12 \times 10^{-6}$
s S	$2,12 \times 10^{-6}$
S s	$2,12 \times 10^{-6}$

Foi calculado a seção de choque para o subprocesso `u U` variando em relação a energia do centro de massa. Os outros subprocessos tem um comportamento igual.

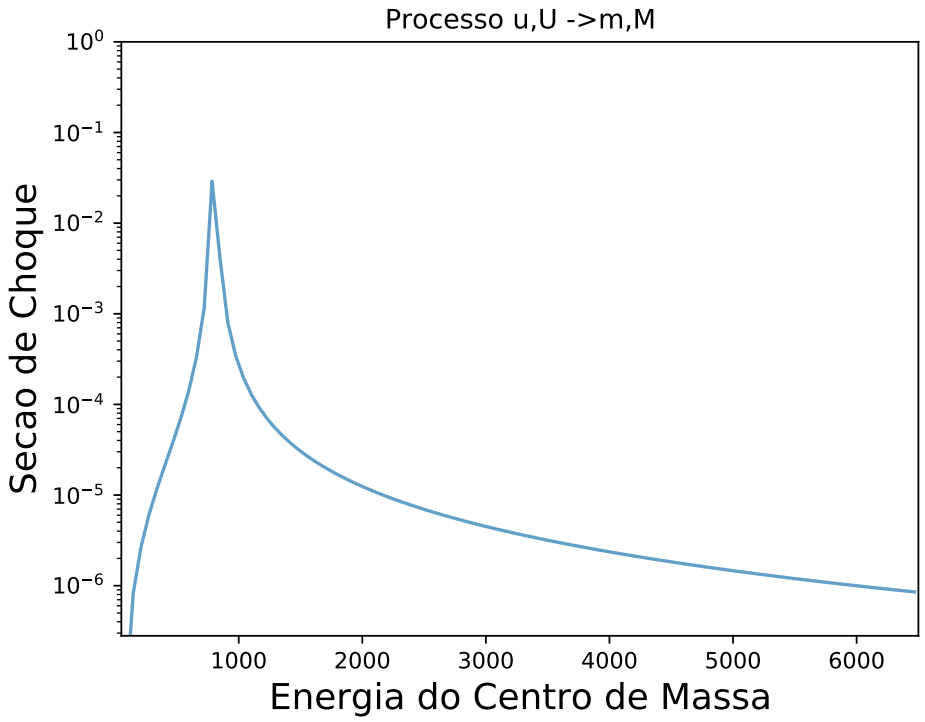

Figura 1

Para obter esse plot, clica na aba `1D integration` e seleciona a última opção `sigma*v plots`. `Pcm` significa Energia do Centro de Massa, ao clicar nela, pode optar por escolher os limites em que será gerado o gráfico e o número de pontos. O limite escolhido foi 50GeV e 6500GeV. Surgirá o plot na tela e clicando em qualquer tecla do teclado ele te levara a outra janela. Clique em `Save Plot in file`, escolha o nome para o arquivo e em seguida qual a extensão. Por opção, a minha escolha foi `Python`. Isso gerará um código na linguagem escolhida e você pode mandar executar o código e ganhará o gráfico acima (fazendo as devidas modificações por razões estéticas).
Abaixo está o código gerado pelo `calcHEP` para a Figura 1

```
import os
import matplotlib.pyplot as plt
import numpy as np
from math import exp, log, sqrt
from scipy import interpolate
#--------------- data input ----------------------#
y0 = np.genfromtxt('plot_1.tab',dtype=float, comments='#',unpack=True)
#--------------- writing plots -----------------#
#fig, ax = plt.subplots(figsize=(6,3.8))
```

```
#plt.subplots_adjust(bottom=0.20,right=0.75,left=0.15)
plt.title('Processo u,U ->m,M')
plt.xlabel('Energia do Centro de Massa',fontsize=16)
plt.ylabel('Secao de Choque',fontsize=16)
plt.xscale('linear')
plt.yscale('log')
xmin=5.000000E+01
xmax=6.500000E+03
plt.xlim(xmin,xmax)
plt.ylim(2.799029E-07,1.00)
xfirst=xmin+0.5*(xmax-xmin)/101
xlast=xmax -0.5*(xmax-xmin)/101
x0=np.linspace(xfirst,xlast,101)
plt.plot(x0, y0[0:101],alpha=0.7, label='v*sigma[pb]', linestyle='-')
plt.savefig('plot_1.pdf')
plt.show()
```

Verificamos como a largura do *Zp* varia com a sua massa. Para isso, temos que trocar o processo, de espalhamento para decaimento. Portanto, no começo do programa, onde tinha antes `p,p -> m,M`, vamos trocar para o decaimento do *Zp*(*Zprime boson*), da seguinte forma:

```
Enter process:  Zp -> 2*X
Exclude diagram with A,Z,H1,H2
Exclude X-particles
```

Em seguida, entre em `Square diagrams` e em seguida `Symbolic calculations` e `C-compiler`. Ao abrir novamente uma outra janela, clica entre *Total width*(último ícone da janela do canto direito). O segundo ícone da janela que for abrir, clica para que apare`Partial widths [GeV]`. Um *print* da tela mostra como a largura da massa do *Zprime boson* varia para cada subprocesso. A largura total é dada por 8.145 GeV

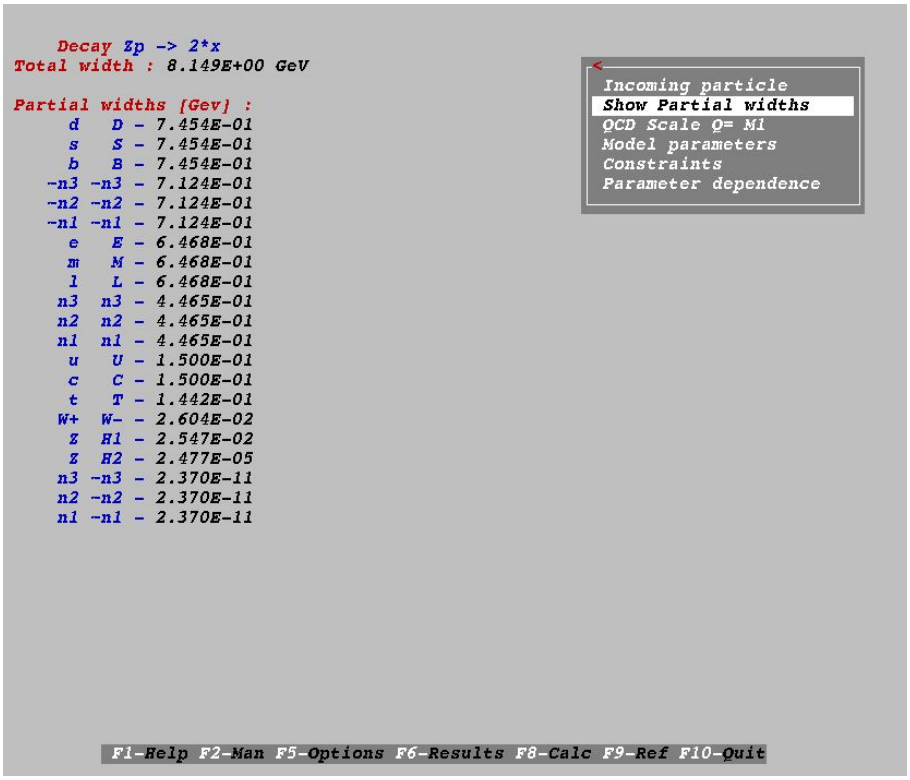

Figura 2

Usando o *batch methods*, um código com as instruções para ser executado no batch foi feito. Abaixo, segue as instruções mencionada.

```
Model:          B-L (Full fast)
Model changed: False
Gauge:          Unitary

Process:   p,p->m,M
Composite: p=u,U,d,D,s,S,c,C,b,B,G
Remove: Z,A,H1,H2

pdf1:       cteq6l1 (proton)
pdf2:       cteq6l1 (proton)

p1:         6500
p2:         6500

Run parameter: MZp
Run begin:     1000
Run step size: 500
Run n steps:   3

alpha Q :            M12

Cut parameter:    n(m)
Cut invert:       False
Cut min:          -100
Cut max:          100

Cut parameter:    n(M)
Cut invert:       False
Cut min:          -100
Cut max:          100

Cut parameter:    M(m,M)
Cut invert:       False
Cut min:          50
Cut max:

Kinematics :    12 -> 3,4

Regularization momentum: 34
Regularization mass:    MZp
Regularization width:   wZp
Regularization power:    2

#Dist parameter:    M(m,M)
#Dist min:          400
#Dist max:          3000
#Dist n bins:       150
#Dist title:        p,p->m,M
#Dist x-title:      Massa invariante (GeV)

Dist parameter:    T(m)
Dist min:          30
Dist max:          600
Dist n bins:       150
Dist title:        p,p->m,M
Dist x-title:      Momentum Transverso (GeV)

Dist parameter:    N(m)
Dist min:          -3.14
Dist max:          3.14
Dist n bins:       150
```

```
Dist title:         p,p->m,M
Dist x-title:       Rapidez

Dist parameter:     Y(m,M)
Dist min:           -3.14
Dist max:           3.14
Dist n bins:        150
Dist title:         p,p->m,M
Dist x-title:       Pseudo-rapidez

Number of events (per run step):  1000
Filename:                         zprime_mm_events
NTuple:                           True

Parallelization method:      local

nSess_1:   20
nCalls_1:  100000
nSess_2:   20
nCalls_2:  100000
```

De posse dessas instruções, salve esse código como `batch_file` no memso diretório do `calchep` que está o `calchep_batch`. Execute da seguinte maneira

```
./calchep_batch batch_file
```

Os *outputs* se encontra no diretório `$HOME/<diretorio>/calchep/batch_results`. Temos então o resultado da massa do *Zprime boson* variando em relação a seção de choque (`zprime_mm_events-cs.jpg`).

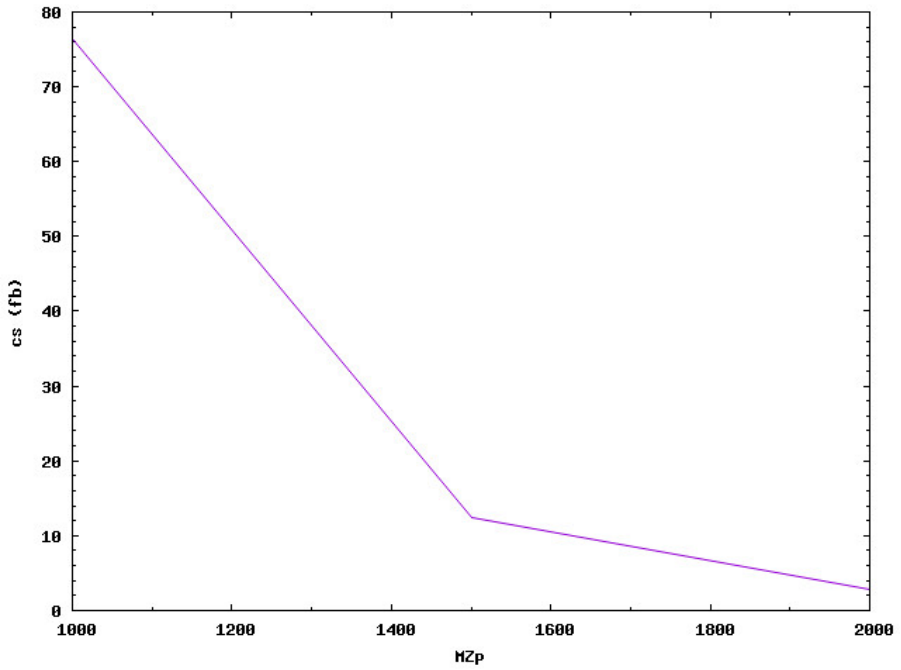

Figura 3: Variação da massa do *Zprime boson* por seção de choque

O *batch mode* do `calcHEP` gera um *output* "zipado"em formato *.lhe*. Precisamos primeiramente descompactá-lo, para isso, com o terminal aberto no diretório que se encontra o arquivo (geralmente em `/$HOME/<diretorio>/calchep/bat` fazer:

```
gunzip zprime_mm_events-MZp000.lhe.gz
```

Ele irá gerar um arquivo do tipo `zprime_mm_events-MZp000.lhe`. O formato `.lhe` guarda as informações do processo em questão. Para extrairmos informações das variáveis `Pt`, `MassaInvariante`, `Rapidez` e `Pseudo-rapidez` são obtidas a partir de uma macro em root para ler o processo físico armazenado em `.lhe` e alocar em uma Ntupla no formato .root. A macro se encontra abaixo.

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <cstdlib>
#include <iomanip>
#include "TTree.h"
#include "TFile.h"
#include "Math/Vector4D.h"

using namespace std;

//Parameters
char* filename  = "/home/matheus/calchep/batch_results/zprime_mm_events-MZp1000.lhe";
char* outputName = "Zprime_boson";
char* treeName = "LHE_Tree";

typedef ROOT::Math::LorentzVector<ROOT::Math::PxPyPzE4D<float> > LorentzVector;

//fill tree for only maxEvents number of events
int numEvents = 0; //initializing counter of events
int maxEvents = 1e9; //if you want to fill all events, make maxEvents huge

//function to take store reweights
float store_reweights(string line) {

        //float wgt;
        int begin = line.find(">"); //beginning of number
        int end = line.find("<",begin);  //end of number
        float wgt= ::atof(line.substr(begin+1,end-(begin+1)).c_str()); //defines wgt
            as substring of line holding reweight
        //between begin and end
        return wgt;
}

int looper(){


        //Declare TTree and TFile
        TFile *file = new TFile(Form("%s.root", outputName), "RECREATE");
        TTree *tree = new TTree("tree", Form("%s",treeName));  //tree called "tree"

        //Declare variables that will be stored in tree
        vector <int> pdgID;
        vector <int> status;
        vector <int> mother_1;
        vector <int> mother_2;
        vector <int> colour_1;
        vector <int> colour_2;
        vector <LorentzVector> four_momentum;  //encodes 4-momentum and position
        vector <LorentzVector> four_position;

        int nParticles;      //variables in row1, first row beneath "<event>"
        int process_number;
        float weight;
        float energy_scale;
        float QED_coupling;
        float QCD_coupling;

        vector <float> reweight; //variable not in main block of code

        //Match up variable with branch
```

```
63
64          //variables to be filled in second_line_below loop
65          tree->Branch("pdgID", &pdgID);
66          tree->Branch("status", &status);
67          tree->Branch("mother_1", &mother_1);
68          tree->Branch("mother_2", &mother_2);
69          tree->Branch("colour_1", &colour_1);
70          tree->Branch("colour_2", &colour_2);
71          tree->Branch("four_momentum", &four_momentum);
72          tree->Branch("four_position", &four_position);
73
74          //filled separate from first_below_event and second_line_below loops
75          tree->Branch("reweight",&reweight);
76
77          //filled in first_below_event loop
78          tree->Branch("nParticles",&nParticles);
79          tree->Branch("process_number",&process_number);
80          tree->Branch("weight",&weight);
81          tree->Branch("energy_scale",&energy_scale);
82          tree->Branch("QED_coupling",&QED_coupling);
83          tree->Branch("QCD_coupling",&QCD_coupling);
84
85          string line; //line of LHE file, to be looped over in main part of looper()
86
87          bool first_line_below = false; //whether on line right below event
88          bool second_line_below = false; //between <event> and </event>, is turned to
                true one line after "first_line_below", both turn off after filling tree
89          bool is_reweight = false;  //notes whether data contains reweight data
90
91          //declare vectors used to fill other variables
92          vector<float> row;  //row used to fill variables in second_line_below loop
93          vector<float> row1; //row used to fill variables in first_line_below part of
                looper()
94
95          //Opening data file
96          fstream myfile (filename, ios_base::in);
97
98          while (getline(myfile,line)){  //loops through file and fills line(the
                variable) with that line of the file
99
100
101
102             if (numEvents > maxEvents) break;  //fills tree only for maxEvents
                    number of events
103
104             //if on 2nd line below "<event>", fill variables(row1 and reweight
                    entries filled in other loops)
105             if (second_line_below == true) {
106                     do {
107                             if (line.find('#') != string::npos) break;  //if line
                                    contains '#', line does not contain data anymore
                                   , break out of loop
108
109                             //iss contains line
110                             istringstream iss;   //opening string stream
111                             iss.str(line);      //copying line into stream
112
113                             float val;
114
115                     while (iss >> val) row.push_back(val);  //fill row with data
                            from line
116
117                             //Now split row into vectors for each variable
```

```
                                pdgID.push_back(row[0]);
                                status.push_back(row[1]);
                                mother_1.push_back(row[2]);
                                mother_2.push_back(row[3]);
                                colour_1.push_back(row[4]);
                                colour_2.push_back(row[5]);
                                LorentzVector four_momentum_temp;
                                four_momentum_temp.SetPx(row[6]);
                                four_momentum_temp.SetPy(row[7]);
                                four_momentum_temp.SetPz(row[8]);
                                four_momentum_temp.SetE(row[9]);
                                four_momentum.push_back(four_momentum_temp);
                                LorentzVector four_position_temp;
                                four_position_temp.SetXYZT(row[10],row[11],row[12],0)
                                    ;  //no time information in file, we have set t=0
                                four_position.push_back(four_position_temp);

                                row.clear();  //clears row so we can fill it again
                                    with next line of data

                        } while (getline(myfile,line));

                                first_line_below = false;
                                second_line_below = false;  // turn first_line_below,
                                     second_line_below off so we know we're done with
                                     those loops
                        }

                        //Line right below <event>(row1 variables)
                        if (first_line_below == true) {
                                second_line_below = true; //sets up loop over
                                    second_line_below variables

                                istringstream iss;
                                iss.str(line);

                                float val1;

                                while (iss >> val1) row1.push_back(val1);  //fill
                                    row1 with numbers in line

                                        nParticles = row1[0];
                                        process_number = row1[1];
                                        weight = row1[2];
                                        energy_scale = row1[3];
                                        QED_coupling = row1[4];
                                        QCD_coupling = row1[5];

                                        row1.clear(); //clears row1 for next event

                                }

                        //Store reweights
                        if (line.find("wgt id") != string::npos) { //if line contains
                             wgt id, start filling reweight

                                //wgt is substring of line that contains the reweight
                                     information
                                float wgt = store_reweights(line);
                                reweight.push_back(wgt); //fill reweight vector

                                is_reweight = true; //note that events contain
                                    reweight information
```

```
172                         }
173
174                         //Check every line after </event> to see if we've reached the
                                next event
175                         if ((line.find("<event>") != string::npos)) {  //if line
                                contains  "<event>", we turn first_line_below to true(
                                begins filling row1 variables)
176
177                             //if file doesn't contain reweight info, fill
                                    reweight with nonsense
178                             if (is_reweight == false) reweight.push_back(-5);
179
180                             if (pdgID.size() != 0){
181                                 tree->Fill();   //only fill tree if vectors
                                        are nonempty(pdgID picked arbitrarily)
182                                 numEvents++;
183
184                             }
185
186                             //clear vectors so we can fill them again for next
                                    event
187                             pdgID.clear();
188                             status.clear();
189                             mother_1.clear();
190                             mother_2.clear();
191                             colour_1.clear();
192                             colour_2.clear();
193                             reweight.clear();
194                             four_momentum.clear();
195                             four_position.clear();
196
197                             first_line_below = true; //begins process to move
                                    onto next event
198                         }
199
200                 }
201
202         tree->Fill(); //last event isn't filled, because there's no next "<event>" to
                trigger it
203
204         file->cd();
205         tree->Write();
206
207         return 0;
208 }
```

Com isso, temos o *output* gerado `Zprime_boson.root`. A parte final, resulta nos gráficos que se seguem. O eixo y é a seção de choque

Figura 4: Massa Invariante do par muon antimuon

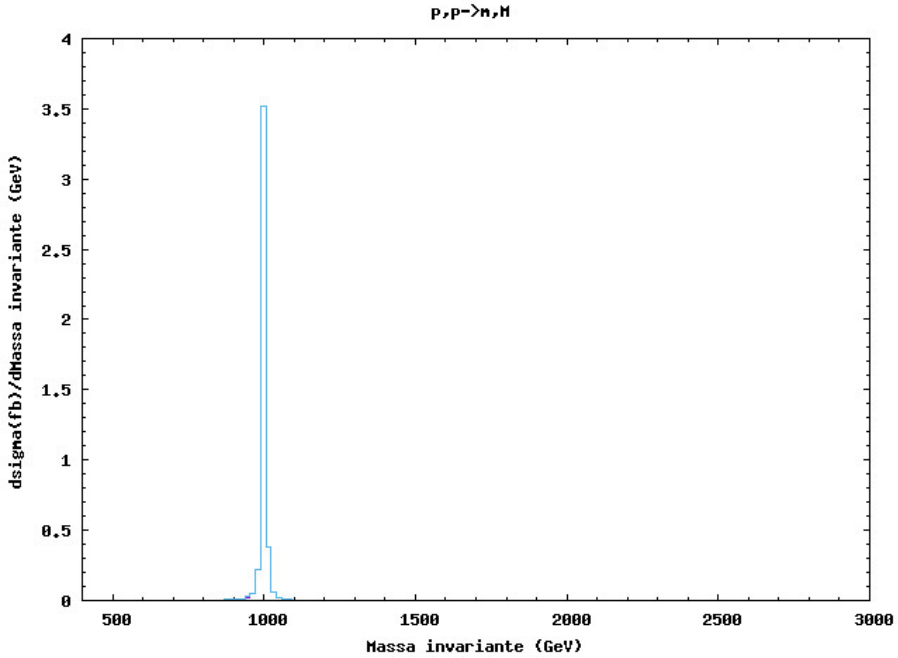

Figura 5: Momentum Transverso do Muon

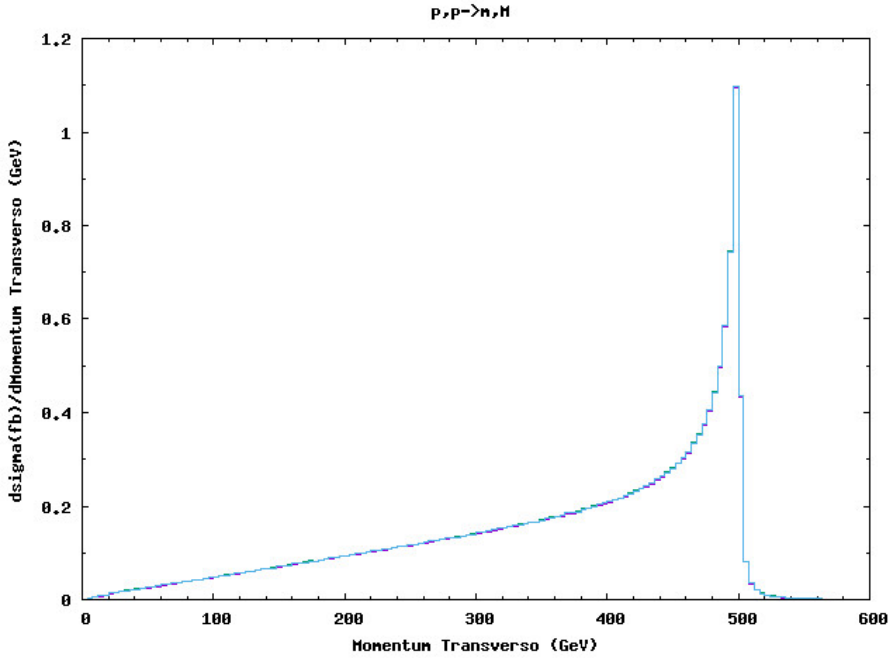

Figura 6: Rapidez

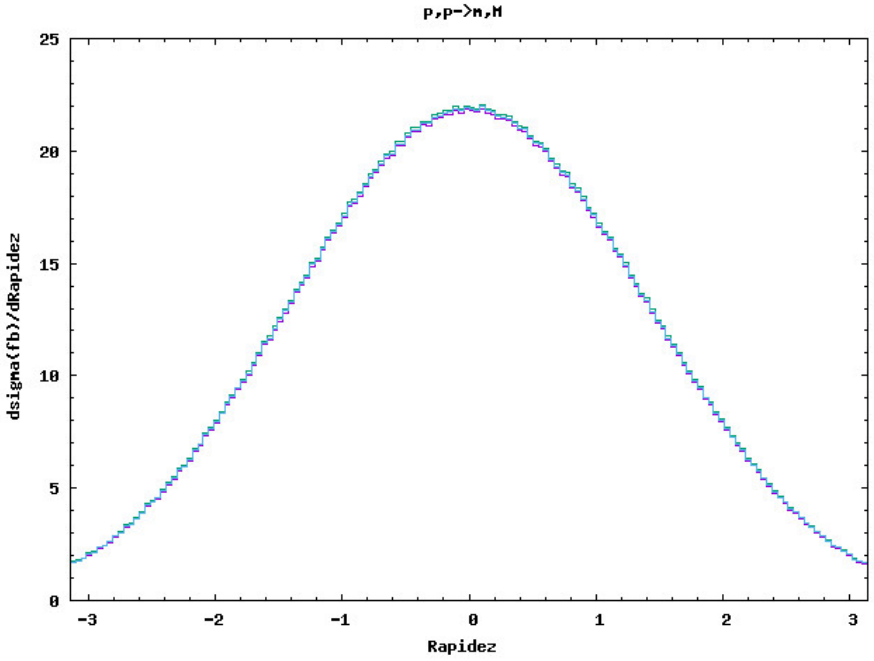

Figura 7: Pseudo-rapidez do par muon-antimuon