UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Dipartimento di Fisica "Giuseppe Occhialini"

Corso di Laurea Magistrale in Fisica

Computational Physics Laboratory
QED Feynman Diagrams

Giuseppe Cavallaro

Academic Year 2019/2020

Contents

1 Introduction

In QED we want to compute the scattering probability amplitudes using the tree-representation of Feynman Diagrams. In the Computational Physics Laboratory we want to implement a way to do that for the zero-loop cases. We use two numerical methods:

- The Algebraic Method.
- The Helicity Amplitudes Method.

The Algebraic Method is the simplest method because the computer will calculate gamma matrix traces and their property but has a really big computational cost so we can implement it only in simple diagrams, because in more complex ones we cannot obtain results in decent times. I will use Maxima in order to implement it with a package provided to us in class.

On the contrary, the best way to compute probability amplitudes is using the Helicity Amplitudes Method. I will describe this method shortly and then implement it.

1.1 Helicity Amplitudes

1.1.1 Theoretical Briefing

At the beginning we define γ^μ as the Weyl gamma matrices set, $p = p^\mu = (p^0, p^1, p^2, p^3)$ the four-momentum with mass m and the slashed representation $\not{p} = p^\mu \gamma_\mu$. In order to define the helicity we write two spinors u and v that need to satisfies

$$\not{p} u_r = m u_r(p), \qquad \not{p} = -m v_r(p) \tag{1}$$

with r=1,2. In QED, helicity is the projection of the spin onto the direction of momentum and we cand define an helicity operator as

$$\vec{\Sigma} \cdot \hat{p} = \begin{pmatrix} \vec{\sigma} \cdot \hat{p} & 0 \\ 0 & \vec{\sigma} \cdot \hat{p} \end{pmatrix}$$

where $\vec{\sigma} = (\sigma^1, \sigma^2, \sigma^3)$ are the 2 x 2 Pauli matrices and $\hat{p} = \frac{\vec{p}}{|\vec{p}|}$ is the unit vector at the direction of the momentum of a particle. It is easy to show that the helicity operator commutes with the Dirac hamiltonian:

$$[\vec{\Sigma} \cdot \hat{p}, H] = 0$$

so the helicity is a good quantum number, in fact its value does not change with time

within a given reference frame. It is valid for both massive and massless fermions. Hence, helicity is conserved for both massive and massless particles but is not Lorentz Invariant.

We can now find eigenstates:

$$\frac{\vec{\sigma}\cdot\vec{p}}{|\vec{p}|}\xi_{\pm}(p) = \pm\xi_{\pm}(p) \tag{2}$$

so

$$\xi_{+}(p) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}|+p^3)}}\begin{pmatrix}|\vec{p}|+p^3\\ p^1+ip^2\end{pmatrix}$$

$$\xi_{+}(p) = \frac{1}{\sqrt{2|\vec{p}|(|\vec{p}|+p^3)}}\begin{pmatrix}-p^1+ip^2\\ |\vec{p}|+p^3\end{pmatrix}$$

and we can now define Dirac spinors as helicity eigenstates

$$u_{\lambda}(p) = \frac{1}{\sqrt{2p^0}}\begin{pmatrix}\sqrt{p^0-\lambda|\vec{p}|}\xi_{\lambda}(p)\\ \sqrt{p^0+\lambda|\vec{p}|}\xi_{\lambda}(p)\end{pmatrix}$$

$$v_{\lambda}(p) = \frac{1}{\sqrt{2p^0}}\begin{pmatrix}-\lambda\sqrt{p^0+\lambda|\vec{p}|}\xi_{-\lambda}(p)\\ \lambda\sqrt{p^0-\lambda|\vec{p}|}\xi_{-\lambda}(p)\end{pmatrix}$$

where $\lambda = \pm 1$.

So after spinor construction we can compute the probability amplitude $\mathcal{M}$, in fact after fixing initial and final quantum number we can calculate it as a product between 4-momentum and matrix. After that we can compute $|\mathcal{M}|^2$ for the selected quantum number set and then the total amplitude (unpolarized) it is nothing else the sum of all $|\mathcal{M}|^2$ for every combination of quantum number.

We consider the Feynman Rules for Diagrams in QED:

- for every external lines ingoing the graph we take a factor $u(p,\lambda)$ for initial states, while $v(p,\lambda)$ for final ones.
- for every external lines outgoing the graph we take a factor $\bar{u}(p,\lambda)$ for initial states, while $\bar{v}(p,\lambda)$ for final ones.
- For every fermionic propagator

$$\frac{i(\not{p}+m)}{p^2-m^2+i\epsilon}$$

where p is the momentum in the direction of the fermionic line

- For the photon propagator
$$\frac{i(-g_{\mu\nu})}{p^2 + i\epsilon}$$
- For the vertex a factor $ie\gamma^\mu$.

1.2 Implementation

Now we want to implement the helicity amplitude method while we use maxima for the algebraic method.

In the program we define two classes, **FourVector** and **ComplexFourVector** and we use it to initialize 4-components vector to represent the 4-momenta, spinors and polarization vectors for massless particles. In the class we define the sum, difference, scalar and vectorial products for the 4-vectors. We use the FourVector class to define the Spinor class and its slash operator. In order to define the slash operator we need a **ComplexMatrix** class because contracted four-vectore are 4x4 complex matrix.

We use two objects **Enum** in order to define the type of our FourVectors (spinors, 4momentum or polarization) and the helicity.

We want to study the difference between the algebraic and helicity-amplitude method in some different cases.

1.2.1 Electron-Positron Annihilation

The electron-positron energy annihilation is

$$e^- + e^+ \to \gamma + \gamma$$

and the two diagrams (see figure(1)).

and we sum the two diagrams to obtain the total amplitude $\mathcal{M}$.

In this case we have photons so we need to define the 4-vector polarization $\epsilon_\mu(p)$ that has the two property

$$\epsilon_\mu(p,\lambda)e^\mu(p,\lambda)$$
$$p_\mu e^\mu(p) = 0$$

with helicity $\lambda = 0, \pm 1$. So the polarizations are

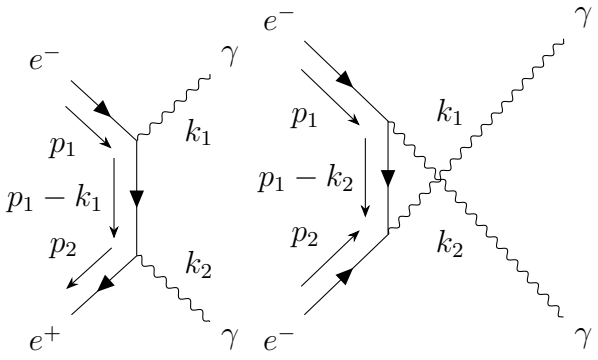

Figure 1: Electron-positron annihilation in the two channels

$$\epsilon^\mu(p,+) = \frac{1}{\sqrt{2}}(-\epsilon_1^\mu(p) + i\epsilon_2^\mu(p))$$

$$\epsilon^\mu(p,-) = \frac{1}{\sqrt{2}}(-\epsilon_1^\mu(p) - i\epsilon_2^\mu(p))$$

$$\epsilon^\mu(p,0) = \frac{E}{m|\vec{p}|}(\frac{|\vec{p}|^2}{E}, p_x, p_y, p_z)$$

where

$$\epsilon_1^\mu(p) = \frac{1}{p_T}(0, -p_y, p_x, 0)$$

$$\epsilon_2^\mu(p) = \frac{1}{|\vec{p}|p_T}(0, p_x p_z, p_y p_z, -p_T^2)$$

$$\epsilon_0^\mu(p) = \frac{E}{m|\vec{p}|}(\frac{|\vec{p}|^2}{E}, p_x, p_y, p_z)$$

For the photons the 0 polarization does not contribute.

We can now write amplitudes as

$$\mathcal{M}_1 = \bar{v}(p_2)\not{\epsilon}(k_2)\frac{\not{p}_1 - \not{k}_1 + m}{(p_1 - k_1)^2 - m^2}\not{\epsilon}(k_1)u(p_1)$$

$$\mathcal{M}_2 = \bar{v}(p_2)\not{\epsilon}(k_1)\frac{\not{p}_1 - \not{k}_2 + m}{(p_1 - k_2)^2 - m^2}\not{\epsilon}(k_2)u(p_1)$$

The Maxima code:

```
declare(d,symmetric);
n_dim:4;
conjugamp_gamma(expr):=rest(rest(reverse(expr),-1),1);
conjugamp(expr):=coefsum(map(lambda([z], [first(z),
```

```
conjugamp_gamma(last(z))]), coefsm(expr,[gamma_prod])));
denom(k):=distrib_d(d(k,k));
k2:p1+p2-k1;
t:-s-u+2*m^2;
setup_distr_symp([p1,p2,k1],[d(p1,p1)=m^2, d(k1,k1)=0, d(p2,p2)=m^2,
d(p1,p2)=s/2-m^2,d(p1,k1)=(-t+m^2)/2, d(k1,p2)=(-u+m^2)/2]);
amp1: gamma_prod([p2,-m],mu,[p1-k1,m],nu,[p1,m])/(denom(p1-k1)-m^2);
amp2: gamma_prod([p2,-m],nu,[p1-k2,m],mu,[p1,m])/(denom(p1-k2)-m^2);
res:multgamma(amp1+amp2,conjugamp(amp1+amp2));
res:factor(distrib_all(contract_d(trace_all(res),[mu,nu])) / 4);
```

Listing 1: Bhabha Scattering with the Algebraic Method in Maxima

For the Helicity Amplitude method now we use python because it is more simple and with performance similar to the C ++ for this case. We introduce a function **Build** that construct the four-momentum object as a spinor or a vector after a check to the **enum** object (Which) that contain the information to the type of four vector and the helicity.

```
uSpinor1 = FourVector.Build(p1, Which.uSpinor, hel1)
vbarSpinor2 = FourVector.Build(p2, Which.vbarSpinor, hel2)
phVector1 = FourVector.Build(k1, Which.epsVector, hel3)
phVector2 = FourVector.Build(k2, Which.epsVector, hel4)
num1 = (((vbarSpinor2*~phVector2)*(~(p1-k1)+m))*~phVector1)*uSpinor1
num2 = (((vbarSpinor2*~phVector1)*(~(p1-k2) + m))*~phVector2)*uSpinor1
amp = (num1/((p1-k1).Squared()-m**2)+num2/((p1-k2).Squared()-m**2))
```

Listing 2: Electron-Positron Annihilation with Helicity Amplitude Method in Python

where is an operator that convert a Complex Matrix object in a contracted Four Vector, uSpinor1 and vbarSpinor2 are spinors, while phVector1 and phVector2 are vector. Then the modulus will be sum for every combination of helicity (hel1, hel2, hel3, hel4) in order to obtain the total amplitude. So we consider 4-momenta with $m = 1$ and $s = 8$ and $u = -4.414213562373095$, the computed amplitudes are

$$|\mathcal{M}_{alg}|^2 = 7.102040816326533$$

$$|\mathcal{M}_{hamp}|^2 = 7.102040816325352$$

that are compatible to the theoretical value.

1.2.2 Positron-Electron pair production

We can create an electron-positron pair in this way

$$\gamma + \gamma \to e^- + e^+$$

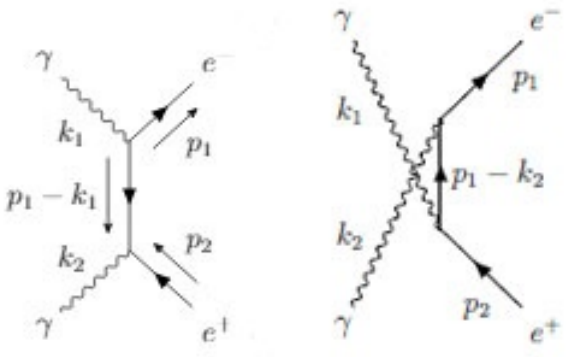

Figure 2: Positron-Electron pair production in the two channels

The two amplitudes are

$$\mathcal{M}_1 = \bar{u}(p_1)\not{\epsilon}(k_1)\frac{\not{p}_1 - \not{k}_1 + m}{(p_1 - k_1)^2 - m^2}\not{\epsilon}(k_2)v(p_2)$$

$$\mathcal{M}_2 = \bar{u}(p_1)\not{\epsilon}(k_2)\frac{\not{p}_1 - \not{k}_2 + m}{(p_1 - k_2)^2 - m^2}\not{\epsilon}(k_1)v(p_2)$$

Maxima and Helicity amplitudes codes are:

```
declare(d,symmetric);
n_dim:4;
conjugamp_gamma(expr):=rest(rest(reverse(expr),-1),1);
conjugamp(expr):=coefsum(map(lambda([z], [first(z),
conjugamp_gamma(last(z))]), coefsm(expr,[gamma_prod])));
denom(k):=distrib_d(d(k,k));
p2:k1+k2-p1;
t:-s-u+2*m^2;
setup_distr_symp([k1,k2,p1],[d(k1,k1)=0, d(k2,k2)=0, d(p1,p1)=m^2,
d(k1,k2)=s/2, d(p1,k1)=(-t+m^2)/2, d(p1,k2)=(-u+m^2)/2]);
amp1: gamma_prod([p1,m],nu,[p1-k1,m],mu,[p2,-m])/(denom(p1-k1)-m^2);
amp2: gamma_prod([p1,m],mu,[p1-k2,m],nu,[p2,-m])/(denom(p1-k2)-m^2);
res:multgamma(amp1+amp2,conjugamp(amp1+amp2));
res:factor(distrib_all(contract_d(trace_all(res),[mu,nu])) / 4);
```

Listing 3: Positron-Electron pair production with the Algebraic Method in Maxima

and

```
ubarSpinor1 = FourVector.Build(p1, Which.ubarSpinor, hel1)
vSpinor2 = FourVector.Build(p2, Which.vSpinor, hel2)
phVector1 = FourVector.Build(k1, Which.epsVector, hel3)
```

```
phVector2 = FourVector.Build(k2, Which.epsVector, hel4)
num1 = (((ubar1*~phVector1)*(~(p1-k1)+m))*~phVector2)*vSpinor2
num2 = (((ubar1*~phVector2)*(~(p1-k2)+m))*~phVector1)*vSpinor2
amp = (num1/((p1-k1).Squared()-m**2)+numerator2/((p1-k2).Squared()-m**2))
```

Listing 4: Positron-Electron pair production with Helicity Amplitude Method in Python

So we consider 4-momenta with $m = 100$ and $s = 8$ and $u = -89.98271772749182$, the computed amplitudes are

$$|\mathcal{M}_{alg}|^2 = 21.23410953007651$$

$$|\mathcal{M}_{hamp}|^2 = 21.23410953007657$$

that are compatible to the theoretical value.

1.2.3 Bhabha Scattering

We know that Bhabha scattering is

$$e^+ + e^- \to e^+ + e^-$$

and the two feynman diagrams

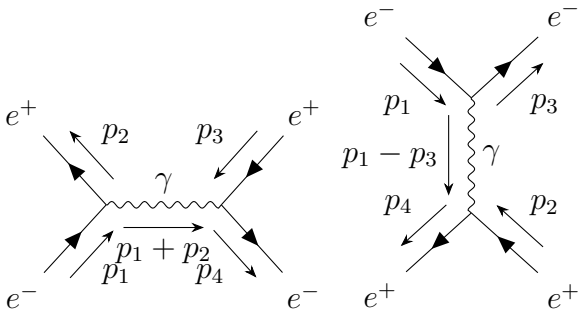

Figure 3: Bhabha Scattering in the two channel

This is the Maxima code for the algebraic method

```
declare(d,symmetric);
n_dim:4;
conjugamp_gamma(expr):=rest(rest(reverse(expr),-1),1);
conjugamp(expr):=coefsum(map(lambda([z], [first(z),
conjugamp_gamma(last(z))]), coefsm(expr,[gamma_prod])));
denom(k):=distrib_d(d(k,k));
p2:k1+k2-p1;
```

```
t:-s-u+2*m^2;
setup_distr_symp([k1,k2,p1],[d(k1,k1)=0, d(k2,k2)=0, d(p1,p1)=m^2,
d(k1,k2)=s/2, d(p1,k1)=(-t+m^2)/2, d(p1,k2)=(-u+m^2)/2]);
amp1: gamma_prod([p1,m],nu,[p1-k1,m],mu,[p2,-m])/(denom(p1-k1)-m^2);
amp2: gamma_prod([p1,m],mu,[p1-k2,m],nu,[p2,-m])/(denom(p1-k2)-m^2);
res:multgamma(amp1+amp2,conjugamp(amp1+amp2));
res:factor(distrib_all(contract_d(trace_all(res),[mu,nu])) / 4);
```

Listing 5: Bhabha Scattering with the Algebraic Method in Maxima

For the Bhabha scattering we use C++ at lesson, so we need to implement spinor and vectors

```
typedef Momentum: double [4];
typedef struct Spinor {complex s[4]; Momentum k[4];}
typedef struct Vector {complex v[4]; Momentum k[4];}
typedef Polarization: int;
```

Now we define spinors and polarization vectors for every external lines of the diagram

```
Spinor uSpinor(momentum k, int Pol);
Spinor vSpinor(momentum k, int Pol);
Spinor ubarSpinor(momentum k, int Pol);
Spinor vbarSpinor(momentum k, int Pol);
Vector PolVector(momentum k, int Pol);
```

Then we define two function that return a spinor object and use it to compute vertex between to spinors or a spinor and photon. Now we define the propagators:

```
Spinor SpinorPropagator(spinor S);
Vector VectorPropagator(Vector V);
```

Then we define functions that contract polarization and vectors.

```
complex SpinorSpinor(Spinor S1, Spinor S2);
Vector VectorVector(Vector V1, VectorV2);
```

Then we can compute the amplitude of the bhabha scattering with the helicity method

```
momentum k1,k2,k3,k4;
int lam1,lam2,lam3,lam4
double tot=0;
for(lam1=-1; lam1<2; lam1+=2) {
for(lam2=-1; lam2<2; lam2+=2) {
for(lam3=-1; lam3<2; lam3+=2) {
for(lam4=-1; lam4<2; lam4+=2) {
complex amp = VectorVector(
SpinorSpinor_vector(ubarSpinor(k3,lam3),uSpinor(k1,lam1)),
VectorPropagator(
SpinorSpinor_vector(vbarSpinor(k2,lam2),vSpinor(k4,lam4))
```

```
)
+
VectorVector(
SinorSpinor_vector(ubarSpinor(k3,lam3),vSpinor(k4,lam4)),
VectorPropagator(
SinorSpinor_vector(vbarSpinor(k2,lam2),uSpinor(k1,lam1))
);
tot=tot+modulus(amp)
}}
}
}
```

Listing 6: Bhabha Scattering with Helicity Amplitude Method in C++

We consider 4-momenta with $m = 1$ and $s = 8$ and $u = -3.41421356237309$, the computed amplitudes are

$$|\mathcal{M}_{alg}|^2 = 325.6382393293399$$

$$|\mathcal{M}_{hamp}|^2 = 325.6382393294532$$

that are compatible to the true value.

1.2.4 Compton Scattering

Compton Scattering is a process that involve an electron and a photon, the photon change its motion direction after the scattering:

$$e^- + \gamma \to e^- + \gamma$$

and the feynamn diagrams in the two channels are

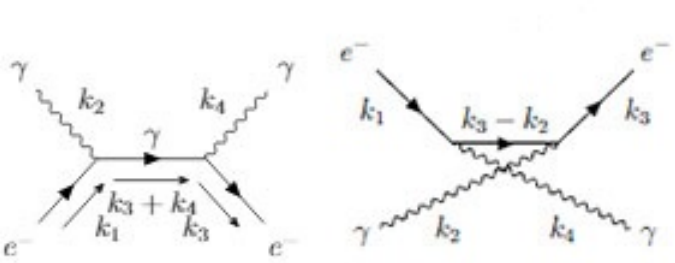

Figure 4: Compton Scattering in the two channels

and the single amplitudes that we need to sum in order to compute the total amplitude are

$$\mathcal{M}_1 = \bar{u}(k_3)\not{\epsilon}(k_4)\frac{\not{k}_3 + \not{k}_4 + m}{(k_3+k_4)^2 - m^2}\not{\epsilon}(k_2)u(k_1)$$

$$\mathcal{M}_2 = \bar{u}(k_3)\not{\epsilon}(k_2)\frac{\not{k}_3 - \not{k}_2 + m}{(k_3-k_2)^2 - m^2}\not{\epsilon}(k_4)u(k_1)$$

This is the Maxima code for the algebraic method

```
declare(d,symmetric);
n_dim:4;
conjugamp_gamma(expr):=rest(rest(reverse(expr),-1),1);
conjugamp(expr):=coefsum(map(lambda([z], [first(z),
conjugamp_gamma(last(z))]), coefsm(expr,[gamma_prod])));
denom(k):=distrib_d(d(k,k));
k4:k1+k2-k3;
t:-s-u+2*m^2;
setup_distr_symp([k1,k2,k3],[d(k1,k1)=m^2,d(k2,k2)=0,d(k3,k3)=m^2,
d(k1,k2)=(s-m^2)/2,d(k1,k3)=-t/2+m^2,d(k2,k3)=(-u+m^2)/2]);
amp1: gamma_prod([k3,m],mu,[k3+k4,m],nu,[k1,m])/(denom(k3+k4)-m^2);
amp2: gamma_prod([k3,m],nu,[k3-k2,m],mu,[k1,m])/(denom(k3-k2)-m^2);
res:multgamma(amp1+amp2,conjugamp(amp1+amp2));
res:factor(distrib_all(contract_d(trace_all(res),[mu,nu])) / 4);
```

Listing 7: Compton Scattering with the Algebraic Method in Maxima

and the Helicity Amplitudes Method (in Python)

```
ubarSpinor3 = FourVector.Build(k3, WhichVector.ubarSpinor, h3)
uSpinor1 = FourVector.Build(k1, WhichVector.uSpinor, h1)
phVector2 = FourVector.Build(k2, WhichVector.epsVector, h2)
phVector4 = FourVector.Build(k4, WhichVector.epsVector, h4)
num1 = (((ubarSpinor3*~phVector4)*(~(k3+k4)+m))*~phVector2)*uSpinor1
num2 = (((ubarSpinor3*~phVector2)*(~(k3-k2)+m))*~phVector4)*uSpinor1
amp = (numerator1/((k3+k4).Squared()-m**2)+num2/((k3-k2).Squared()-m**2))
```

Listing 8: Compton Scattering with Helicity Amplitude Method in Python

We consider 4-momenta with $m = 1$ and $s = 5.829631555131293$ and $u = -3.24384511750439$, the computed amplitudes are

$$|\mathcal{M}_{alg}|^2 = 3.81137659049842$$

$$|\mathcal{M}_{hamp}|^2 = 3.81137659049857$$

that are compatible to the true value.

1.2.5 Increase photon number in the Compton Scattering

Now we want to study the computational efficancy of the algebraic methond and the helicity ones by adding a photon to the compton scattering so:

$$e^- + \gamma \to e^- + \gamma + \gamma$$

and it's report here one of the six diagrams that we need to study in Figure 5.

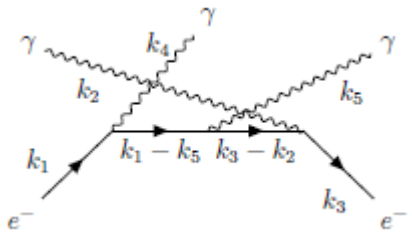

Figure 5: Compton Scattering with one photon more

The total amplitude is:

$$|\mathcal{M}|^2 = \left(\sum_{i=1}^{6} \mathcal{M}_i\right)\left(\sum_{i=1}^{6} \mathcal{M}_i^*\right)$$

and we need to compute in Maxima 36 products of amplitudes. The computational cost of the algebraic method in Maxima is really big and the computer after 18 hour can't show a result. With the helicity amplitude method we don't have the same problem because the ALU of the CPU is able to compute several products of vectors and matrix. This is the code in python:

```
uSpinor1 = FourVector.Build(k1, Which.uSpinor, hel1)
ph2 = FourVector.Build(k2, Which.epsVector, hel2)
ubarSpinor3 = FourVector.Build(k3, Which.ubarSpinor, hel3)
phVector4 = FourVector.Build(k4, Which.epsVector, hel4)
phVector5 = FourVector.Build(k5, Which.epsVector, hel5)
num1 = (((((ubarSpinor3*~phVector5)*(~(k3+k5)+m))*~phVector4)*(~(k1+k2)+m
    ))*~phVector2)*uSpinor1
num2 = (((((ubarSpinor3*~phVector5)*(~(k3+k5)+m))*~phVector2)*(~(k1-k4)+m
    ))*~phVector4)*uSpinor1
num3 = (((((ubarSpinor3*~phVector2)*(~(k3-k2)+m))*~phVector4)*(~(k1-k5)+m
    ))*~phVector5)*uSpinor1
num4 = (((((ubarSpinor3*~phVector4)*(~(k3+k4)+m))*~phVector5)*(~(k1+k2)+m
    ))*~phVector2)*uSpinor1
num5 = (((((ubarSpinor3*~phVector4)*(~(k3+k4)+m))*~phVector2)*(~(k1-k5)+m
    ))*~phVector5)*uSpinor1
```

```
num6 = (((((ubarSpinor3*~phVector2)*(~(k3-k2)+m))*~phVector5)*(~(k1-k4)+m
    ))*~phVector4)*uSpinor1
amp = (num1/(((k3+k5).Squared()-m**2)*((k1+k2).Squared()-m**2))+num4/(((
    k3+k4).Squared()-m**2)*((k1+k2).Squared()-m**2))+num2/(((k3+k5).
    Squared()-m**2)*((k1-k4).Squared()-m**2))+num5/(((k3+k4).Squared()-m
    **2)*((k1-k5).Squared()-m**2))+num3/(((k3-k2).Squared()- m**2)*((k1-k5
    ).Squared()-m**2))+num6/(((k3-k2).Squared()-m**2)*((k1-k4).Squared()-m
    **2)))
```

Listing 9: Double Compton Scattering with Helicity Amplitude Method in Python

2 Hadronic Cross Section

We want now to study an hadronic collision process, i.e the pair production of top quarks (top-antitop) in LHC. The center of mass energy is 13 TeV and we collide a couple of protons. At the beginning of the process we can see components of the proton (quarks $u, \bar{u}, d, \bar{d}$ and gluons g) scatter producing heavy quarks pairs. The cross section is

$$\sigma = \int dx_1 dx_2 \sum_{ij} f_i(x_1, \mu) f_j(x_2, \mu) d\sigma_{ij}(x_1 P_1, x_2 P_2) \tag{1}$$

The functions $f_i(x, \mu)$ represent the probability to find the component of the proton with flavor i (quark, antiquark or gluon) in the proton with a fraction x of the proton's momentum.

We can take the energy μ as equal to the heavy quark's mass (top). Also $d\sigma_{ij}(x_1 P_1, x_2 P_2)$ represent the differential cross section of the process that two components i and j of the protons with momenta $x_1 P_1$ and $x_2 P_2$ create a pair of heavy quarks.

We can write $d\sigma_{ij}$ in two different ways, one for quarks and one for gluons:

$$\frac{d\sigma_{q\bar{q}}}{d\Omega} = \frac{\alpha_s^2}{9s^2}\sqrt{1 - \frac{4m_Q^2}{s}}[(m_Q^2 - t)^2 + (m_Q^2 - u)^2 + 2m_Q^2 s] \tag{2}$$

$$
\begin{aligned}
\frac{d\sigma_{gg}}{d\Omega} =& \frac{\alpha_s^2}{32s}\sqrt{1-\frac{4m_Q^2}{s}}\left[\frac{6}{s^2}(m_Q^2-t)(m_Q^2-u)+\right.\\
&-\frac{m_Q^2(s-4m_Q^2)}{3(m_Q^2-t)(m_Q^2-u)}+\\
&+\frac{4}{3}\frac{(m_Q^2-t)(m_Q^2-u)-2m_Q^2(m_Q^2+t)}{(m_Q^2-t)^2}+\\
&+\frac{4}{3}\frac{(m_Q^2-t)(m_Q^2-u)-2m_Q^2(m_Q^2+u)}{(m_Q^2-u)}+\\
&-3\frac{(m_Q^2-t)(m_Q^2-u)+m_Q^2(u-t)}{s(m_Q^2-t)^2}+\\
&\left.-3\frac{(m_Q^2-t)(m_Q^2-u)+m_Q^2(t-u)}{s(m_Q^2-u)}\right]
\end{aligned}
\tag{3}
$$

where Ω is the scattering angle. In the quark-antiquark case $\sigma_{q\bar{q}}$ can partecipate $u\bar{u}$ and $\bar{u}u$, $d\bar{d}$ and $\bar{d}d$. The solid angle is $d\Omega = dcos\theta d\phi$ and u and t depends only on θ. s, t, u are Mandelstam variables, α_s is the coupling constant and m_Q is the heavy quark mass producted by the scattering (the top quark).

I report here the algorithm to compute the cross section in C++

```
// arg[0] = theta
// arg[1] = x1
// arg[2] = x2
double TotalCrossSection(double * arg) {
  double mQ = 172.5;
    double m2 = pow(mQ, 2);
    double x1 = arg[1];
  double x2 = arg[2];
    double th_a = 0;     // min range of theta
  double th_b = M_PI;  // max range of theta
  double th   = th_a + (th_b - th_a) * arg[0]; // theta in (th_a, th_b)

    double s = x1 * x2 * S;
    if (s < 4*m2) return 0.0;
    double t = m2 - (s/2) * ( 1 - sqrt(1 - 4*m2/s)*cos(th) );
    double u = 2*m2 - t - s;
    double res = 0.0;
  double temp;

    // loop all quarks and gluons
    for (int fl = -2; fl <= 2; fl++) {
    if (fl == 0) temp = ggCrossSection(s, t, u, mQ);
```

```
23      else temp = qqCrossSection(s, t, u, mQ);
24
25          // xfxM() returns x*f(x, mQ, fl), so it must be divided by x
26          temp *= xfxM(1, x1, mQ, fl) * xfxM(1, x2, mQ, -fl) / (x1 * x2);
27      res += temp;
28    }
29    res *= 2 * M_PI * (th_b - th_a) * sin(th);
30      return res;
31  }
```

2.0.1 Implementation

We want to compute the integral (1), generally if we have N particles in final state, we need, in order to describe the phase-space, $3N-4$ parameters, i.e 3 components for the momentum of every particles without 4 parameters for the conservation of energy and momentum. Our case is about an hadronic collision so we need to add two parameters, the fraction of the initial momentum of the quark and gluons (the components of the hadron) that collide. So we need a Monte Carlo method to compute the cross section. A Monte Carlo method is a method that use random numbers. We integrate on 3 parameters only, the fractions of momentum and theta.

In order to increase the performance of our Monte Carlo integrators we introduce an adaptive method that generate a grid of point and make the algorithm generate more random point where the function that we want to integrate is greater.

We define the grid as a function $h(y) = x$ then generate y points to compute x coordinate of the function as $f(h(y))$. After defining the grid we need to generate events defining some parameters:

- nGrid = 100000, is the number of cycles necessary for the grid generation
- nInt = 1000, the number of intervals of the x axis, divided in order to generate the grid
- nRef = 5, is the number of time the algorithm call the grid and refine it
- nCycl = 1000000, the number of cycles of the Monte Carlo algorithm used to compute the integrals

For the generation of events is required the definition of an upper bound for every intervals of the x axis. Then with an "Hit and Miss" we can generate points that follows the integrate function needed. The integrator need to generate the parameters x_1,x_2 and θ, it generate a number between 0 and 1 that it is good for the fractions but not for the angle theta that will be multiplied by a factor π in order to obtain a

distribution between 0 and π. I report here a way to generate the parameters using the hit-miss method in C++

```
void IntegrationGeneration::generatePoint(double * xval, int &miss, int &
    ub_violation) {

  int l[Ndim];
  double r[Ndim];
  double rr;
  double fval;
  double upper_b;

    for (int k = 0; k < Ndim; k++) r[k] = unif(re);
    for (int k = 0; k < Ndim; k++) {
        for (int j = 1; j <= NumIntervals; j++) {
            if (Ti[k][j] > r[k]) { l[k] = j; break; }
        }
        xval[k] = xi[k][l[k]-1] + ( r[k] - Ti[k][l[k]-1] ) / ( Ti[k][l[k
    ]] - Ti[k][l[k]-1] ) * ( xi[k][l[k]] - xi[k][l[k]-1] );
    }
    // Hit-And-Miss
    fval = f(xval);
    rr = unif(re);
    upper_b = 1.0;
    for (int k = 0; k < Ndim; k++) upper_b *= Si[k][l[k]];
    if (fval > upper_b) ub_violation++;

    while (rr > fval / upper_b) {
        miss++;
    for (int k = 0; k < Ndim; k++) r[k] = unif(re);
    for (int k = 0; k < Ndim; k++) {
      for (int j = 1; j <= NumIntervals; j++) {
        if (Ti[k][j] > r[k]) { l[k] = j; break; }
      }
      xval[k] = xi[k][l[k]-1] + ( r[k] - Ti[k][l[k]-1] ) / ( Ti[k][l[k]]
    - Ti[k][l[k]-1] ) * ( xi[k][l[k]] - xi[k][l[k]-1] );
    }
        // Now the Hit-And-Miss
    fval = f(xval);
    rr = unif(re);
    upper_b = 1;
    for (int k = 0; k < Ndim; k++) upper_b *= Si[k][l[k]];
    if (fval > upper_b) ub_violation++;
    }
}
```

2.0.2 Results

The variables that we need in order to compute the integral are the fractions of momentum x_1 and x_2 and the scattering angle θ. Of the $d\sigma_{ij}$ we know s as the scattering energy, fixed to 13 TeV and m_Q, the top mass, is 172.5 GeV. The coupling constant α_s is known and it is $\alpha_s = 0.118$. t and u depends on the scattering angle and s.

The computation of the integral use this relation

$$I = \frac{1}{N}\sum_{\alpha=1}^{N} f(\vec{h}(\vec{y}_\alpha)) \prod_{d}^{k=1} \left.\frac{dh^k(y)}{dy}\right|_{y=y_\alpha^k}$$

where for every dimensions $k = 1, ..., d$ we have a sequence $x_j^k (j = 1, .., m)$ of point that represent the extremes of the range $[0, 1]$ so $x_l^k \frac{l}{m}$ and we produce N sequence of random numbers y_α^k where $k = 1, d$ and $\alpha = 1, .., N$. So it's true

$\left.\frac{dh^k(y)}{dy}\right|_{y=y_\alpha^k} = \frac{x_l^k - x_{l-1}^k}{1/m} = (x_l^k - x_{l-1}^k)m, \qquad l = [y_\alpha^k \times m] + 1$

and we can estimate the integral and its error by the sum of al contributions

$$I = \frac{1}{n}\sum_{\alpha=1}^{N} d_\alpha, \quad \Delta = \frac{1}{n}\sum_{\alpha=1}^{N} d_\alpha^2, \quad Err = \sqrt{\frac{\Delta - I^2}{N}}$$

The result of the integration and its Monte Carlo error is

$$\sigma = (567.4689847238 \pm 4.9364328927)pb$$

Now we want to generate some triads of events (x_1,x_2 and θ) in order to compute the invariant mass of the system and the transverse momentum of the top.

The energy of this collision is 13 TeV, we can define the Mandelstam variable S as $S = (P_1 + P_2)^2$ where P_1 and P_2 are the momenta of the protons in initial state so $\sqrt{S} = 13$ TeV. Now, if we see the system from the center of mass point of view we and we put the mass of the protons to 0 then the two protons will have the same energy E and the module of the momenta will be equal but in opposite direction. So $S = 4E^2$ and $P_1^2 = P_2^2 = 0$ (because we consider only the z-direction) and the module $|P_1| = |P_2| = E = 6.5$ TeV. We want to study the scattering

$$q(p_1) + \bar{q}(p_2) \to Q + \bar{Q}$$

of the components of protons where $p_1 = x_1 P_1$ and $p_2 = x_2 P_2$ are the fractions of

the proton's momenta. We consider the components massless and the direction of the momenta is opposite on the z-axis so p_1 and p_2 are

$$p_1 = \begin{pmatrix} x_1E \\ 0 \\ 0 \\ x_1E \end{pmatrix} \quad p_2 = \begin{pmatrix} x_2E \\ 0 \\ 0 \\ -x_2E \end{pmatrix}$$

So if we consider the Mandelstam variable S, we know for the massless particles that $S = 2P_1P_2$ and we can write

$$s = (x_1P_1 + x_2P_2)^2 = x_1x_2 2P_1P_2 = x_1x_2S$$

and this is the invariant mass. Also we can write the other mandelstam variables t, u:

$$t = (p_1 - k_1)^2 = m_Q^2 - \frac{s}{2}\left(1 - \sqrt{1 - \frac{4m_Q^2}{s}}cos\theta\right)$$

$$u = 2m_Q^2 - s - t$$

.

Now let's compute the transverse momentum p_T but we are in the center of mass of the protons, not of the components so we must apply a boost along the z axis to bring the system in the component's center of mass. p_T is dependent on the p_x, p_y, that are not modified by the boost. We can write the boost only for the energy and p_z component:

$$\begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \begin{pmatrix} x_1E \\ x_1E \end{pmatrix} = \begin{pmatrix} \gamma(x_1E - \beta x_1E) \\ \gamma(x_1E - \beta x_1E) \end{pmatrix}$$

$$\begin{pmatrix} \gamma & -\beta\gamma \\ -\beta\gamma & \gamma \end{pmatrix} \begin{pmatrix} x_2E \\ -x_2E \end{pmatrix} = \begin{pmatrix} \gamma(x_1E + \beta x_2E) \\ -\gamma(x_2E + \beta x_2E) \end{pmatrix}$$

So the total energy of the system is

$$E_{tot} = \gamma(x_1E - \beta x_1E) + \gamma(x_2E + \beta x_2E)$$

for $\beta = \frac{x_1 - x_2}{x_1 + x_2}$.

For the transverse momentum of the top quarks we need to write the momenta of the quarks

$$k_1 = \begin{pmatrix} E' \\ k_x \\ k_y \\ k_z \end{pmatrix} \qquad k_2 = \begin{pmatrix} E' \\ -k_x \\ -k_y \\ -k_z \end{pmatrix}$$

Finally we can write the transverse momentum as

$$k_T = \sqrt{k_x^2 + k_y^2}$$

.

For $k^2 = |\vec{k}_1|^2 = |\vec{k}_2|^2 = k_T^2 + k_z^2$:

the transverse momentum become

$$k_T^2 = k^2 sin^2\theta$$

and

$$k_z^2 = k^2 cos^2\theta = k_T^2 cot^2\theta$$

so the energy E' can be written

$$E' = \sqrt{m_Q^2 + k_T^2 + k_z^2} = \sqrt{m_Q^2 + k_T^2(1 + cot^2\theta)}$$

and for the conservation of energy $E_{tot}^2 = (2E')^2$

$$k_T^2 = \frac{E_{tot}^2 - 4m_Q^2}{4(1 + cot^2\theta)}$$

Here we can see the plots of the transverse momentum and the $\sqrt{s}$ distributions for N=100000.

We can see for the second plot that the result is never zero, because the minimum is $s = m_t^2 + m_{\bar{t}}^2$

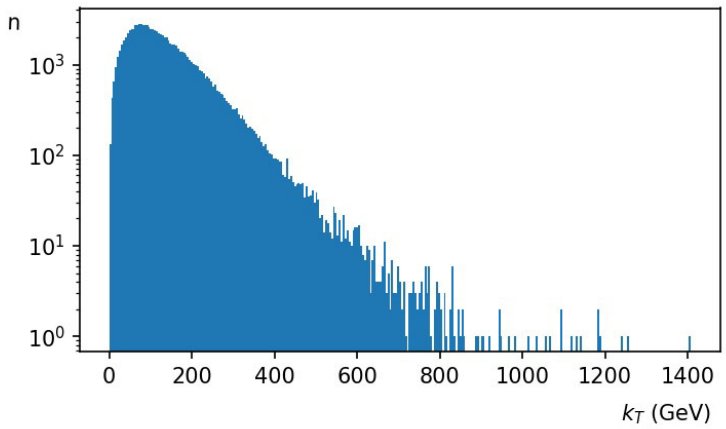

Figure 6: Distribution of the Transverse Momentum

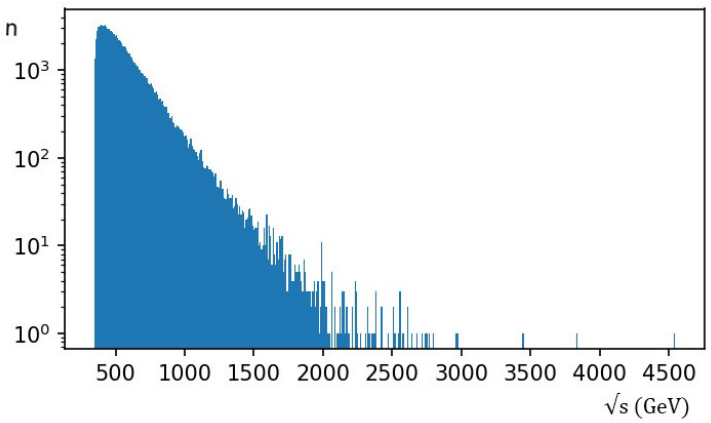

Figure 7: Distribution of the $\sqrt{s}$