BACHELOR THESIS (BSC)

Dynamics of the Higgs in the proton

Author: Fabian Veider

Supervisor: Prof. Axel Maas

September 7, 2020

Abstract

The objective of this bachelor thesis is to study the dynamics of a post-SM proton postulated by a quantized, non-abelian gauge theory with a Brout-Englert-Higgs effect. Non pertubatively, this leads to bounded states. Therefore, dynamics involving a Higgs-particle are of interest, whose fraction and PDF are the main focus of this thesis. The data to look into is provided by HERWIG++, simulating collisions of protons at the LCD and focusing on Higgs-sensitive $t\bar{t}$ collisions as outcomes. The analysis, numerical and graphical, of total and differential cross sections provides new insight into the subtle mechanics inside the proton caused by the Higgs and its corresponding PDF.

Contents

Abstract **1**

1 Introduction **1**

2 Theoretical approach **2**
2.1 Total cross sections . . . 3
2.2 Differential cross sections . . . 4
2.3 Parton distribution functions . . . 5
2.4 Composite states and why the Higgs . . . 8

3 Technical approach **9**
3.1 Perturbartive treatment . . . 9
3.2 Feynman diagrams . . . 10
3.3 Initial state with the Higgs . . . 15
3.4 Adding the Higgs-PDF . . . 16
3.5 Limiting the search . . . 17

4 Results **18**
4.1 Cross section and dynamics . . . 18
4.2 PDF and Higgs fraction . . . 26

5 Conclusion **28**
5.1 Summary . . . 28
5.2 A look into future research . . . 28

A Ratio plots **29**

B Code **32**
B.1 pdf_yoda.py . . . 32
B.2 master_yoda.py . . . 51
B.3 pdf_plot.py . . . 59

C Bibliography **61**

List of Figures **62**

1 Introduction

Theoretical particle physics attempts to find models, mathematical tools and frameworks to understand the current experiments related to fundamental particles and their interactions. Particle physics limiting itself to the Standard Model results in deficiencies[1], and string theory is still in need of providing experimental ways to prove its theories[2]. Contrarily, particle physics beyond the Standard Model seems to be on the right way to new discoveries. With fields being the main physical quantities, one of its biggest aims is to identify symmetries found in these physical objects. These fields can be expressed as an invariance of the action integral, similar to its classical counterpart. Noether's theorem then provides conserved quantities, bringing us physical quantities that should be observable and leading to a further understanding of the laws of nature.

These symmetries can be further distinguished into so called space-time and internal symmetries. The latter ones, often called local gauge symmetries, are the basis of the similarly called gauge theories used in modern physics. Reaching back to Maxwell's formulation in the 1870s, Einstein's theories about unifying all fundamental forces and Hermann Weyl's attempt of providing global symmetries, gauge theories provide a very useful mathematical construct. Not only do they provide a way to simplify otherwise hard to solve physical problems, they also offer a successful framework for processes in particle physics, in the SM and beyond. Gaining its final breakthrough after Geradus 't Hofft's proof for their renormalizability, even for spontaneous symmetry breaking, raised gauge theories to a standard tool in particle physics.

In this thesis we will go beyond the Standard Model and look at a quantized, non-abelian gauge theory with Brout-Englert-Higgs effects by intoducing a modified proton. To do so, we will be using perturbative methods. The aim is to look at simulations of proton-proton-collisions and the resulting differential cross sections from sensitive ttbar quark decays, hoping to find forms of Gluon-Higgs gH and Higgs-Higgs HH processes that decrease faster than the proton of the Standard Model, here called sm. We will look for a suitable parametrization of the proton-PDF and its effects on differential cross sections depending on the momentum of t quarks along the collision axis called pT_T. This work can be seen as a follow up on work done by [3] and [4].

At first, we will take a look at the theoretical details concerning mentioned cross sections, the importance of PDFs in this framework, and how they help us to better understand how the Higgs-boson contributes to the structure of protons. The necessary data is provided by a high energy Monte Carlo simulation program. Therefore, in chapter 3, details on the program HERWIG++ will be presented. Feynman diagrams of the processes will be discussed, as they provide an intuitive way of looking at the otherwise complex mathematical interactions. In chapter 4, the results from our analysis will be presented and compared to processes and resulting PDF from older work done by [4], looking for similarities and differences. In the end, a short summary and outlook for possible future simulations and theses will be given.

2 Theoretical approach

To start of with, the following chapters provide an introduction into the main theoretical concepts involved. Firstly, we will take a brief look at total cross sections in section 2.1 and their applications in QFT. Additionally, we will introduce differential cross sections in 2.4, as they are the primary physical quantities in this thesis, giving insight on the dynamical processes involved. Since calculations of differential cross sections involve using so-called parton distribution functions (short PDF), section 2.3 will be used to further elaborate on these objects. Then, we will justify why the Higgs particle is necessary here in section 2.4 and add it to the existing system, resulting in dynamical changes discussed in 3.4. The first sections until 2.3 are mainly taken from [5] and [6], while sections 2.4 and onward are heavily influenced by [3] and [7].

2.1 Total cross sections

The physical quantities taken from our (simulated) colliding particles are cross sections. Apart from decay rates, they are the most fundamental experimental quantities in particle physics. Total cross sections serve as the proportionality constants of interaction rates r and incident particle fluxes Φ (see equation 2.1), being defined as the number of interactions per unit time per target particle involved, over the incident flux. The incident flux here is the number of particles crossing a defined area in a certain amount of time.

$$r = \sigma \times \Phi \tag{2.1}$$

The main takeaway is that σ corresponds to the effective cross sectional area each particle has, which is sometimes related to the actual physical cross section. It simply is an expression of the probability of interactions in the framework of QFT. The total cross sections needed for our analysis could be calculated using a relativistic formulation of Fermi's golden rule seen in equations 2.2 and onward, while also adding the particle flux invariant to Lorentz transformations, which is $F = 4E_aE_b(v_a + v_b)$. Here, v_a and v_b represent the velocities of two particles a and b with relativistic momenta p_a and p_b and energies E_a and E_b, scattering into 1 and 2 with momentum p_1 and p_2, see figure 1. This is done by the simulation program, one might use [6] for further information.

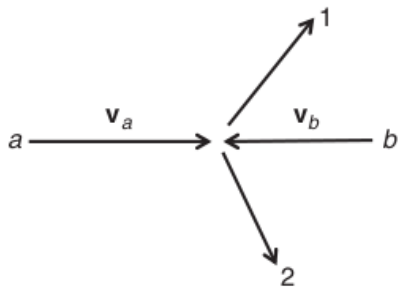


Figure 1: Two body scattering of particles a and b, from [6]

$$\sigma = \frac{(2\pi)^{-2}}{F} \int \left|\mathcal{M}_{fi}\right|^2 \delta^4(E, \mathbf{p}) \frac{\mathrm{d}^3\mathbf{p}_1}{2E_1} \frac{\mathrm{d}^3\mathbf{p}_2}{2E_2} \tag{2.2}$$

$$\delta^4(E, \mathbf{p}) = \delta\left(E_a + E_b - E_1 - E_2\right) \delta^3\left(\mathbf{p}_a + \mathbf{p}_b - \mathbf{p}_1 - \mathbf{p}_2\right) \tag{2.3}$$

$$\mathcal{M}_{fi} = NT_{fi} \tag{2.4}$$

The term $\mathcal{M}_{fi}$ is the Lorentz invariant transitions matrix one obtains when combining the energies of initial and final particles, here E_1 to E_4, and multiplying with T_{fj}. $\delta^4(E, \mathbf{p})$ is the combined delta functional for energies and momenta, needed for integration over phase-space. The element T_{fj} of the transition matrix corresponds to the perturbation series of states $|j\rangle$ and potential field V.

$$T_{fi} = \langle f|V|i\rangle + \sum_{j \neq i} \frac{\langle f|V|j\rangle\langle j|V|i\rangle}{E_i - E_j} + \cdots \tag{2.5}$$

2.2 Differential cross sections

Apart from total cross sections, we will investigate derived quantities. Investigating the change of total cross section related to, for example, an angle or energy, is a regular procedure leading to more detailed description of collision dynamics. Due to practical reasons discussed in 2.3, the focus in this thesis will be the distribution of total cross sections with respect to the transverse momentum from one of the outgoing particles from the collision axis. In our case, this will be the top quark, as it is most sensitive to change and easiest to work with. The resulting differential cross sections describe the number of particles scattered in a certain region dpT_T per time and particle, again over the incident flux. Integrating the differential cross section with respect to the value being distributed over results in the total cross section from above.

$$\sigma = \int \frac{d\sigma}{dpT_T} dpT_T \tag{2.6}$$

In a classical example shown in figure 2, an electron is deflected by a static proton by an angle θ relative to the collision axis, landing in a certain area described by the angle Ω after the collision. As the momentum pT_T does not have this spacial description, this graphical example is used to emphasize the difference to total cross sections. Similarly, the calculation of these values includes the matrix $\mathcal{M}_{fi}$. These are also done by the simulation program used [6], being provided in so-called yoda-files. The main reason they will be included is that there exists a difference in kinematics, seen by [7], for processes with Higgs in the initial state. The explicit formula for a general variable distribution is given below, for further information see [3, p. 33] and [6, p. 73ff].

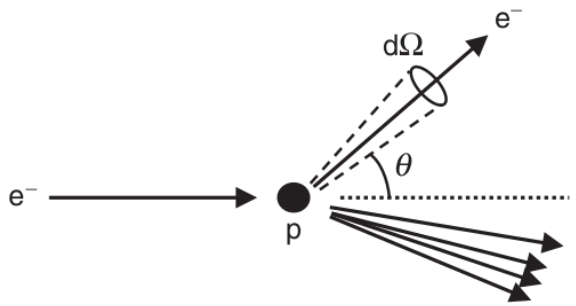


Figure 2: Example of a collision highlighting the differential cross section with respect to the angle Ω

$$\frac{d\sigma}{d\Omega} = \sum_{i,j} \int_0^1 \int_0^1 dx_1 dx_2 PDF^i\left(x_1, Q^2\right) PDF^j\left(x_2, Q^2\right) \frac{d\hat{\sigma}_{i,j\to k}}{d\Omega_k}\left(x_1 p_1, x_2 p_2, Q^2\right) \tag{2.7}$$

Here, i, j mark quark-, antiquark flavours and gluons, the final state kinematics are represented by Ω_k, not necessarily angles. $x_{1,2}$ and $p_{1,2}$ are the fractions and momenta of hadrons, in our case protons. The functions $PDF^{H_i}(x_i, Q^2)$ describe the PDFs of our colliding particles, being the main topic of the next section.

2.3 Parton distribution functions

Protons are being held together by exchanging quarks. In this system of interactions, dynamics will lead to a distribution of quark momenta inside the particle. Here, Parton distribution functions (PDFs in short) serve as a description of said dynamics. The up-quark and down-quark constituents of the proton have PDFs defined in such a way that $u^p(x)\delta x$ and $d^p(x)\delta x$ represent the momentum fraction of these quarks held in the interval $x + \delta x$. Their explicit form depends on the underlying dynamics. As of now, they cannot be obtained from theory because in QCD the coupling constant α_g for gluons is of order $\mathcal{O}(1)$, preventing perturbation theory[8]. Consequently, only from experiments and simulations one can find a suitable form. Normally, one would use experimental data, but that has not been possible as of now[9], urging us to use simulated data as reference. Starting of, it is useful to look at a few simple examples before diving into the more sophisticated case were Higgs particles are added into the pool.

The simplest interaction would be one fundamental particle carrying all the momentum of the proton. In this case, the PDF would just be a Dirac delta-functional, centered at $x = 1$ (i). Next are multiple static constituents, three quarks without any interaction going on between them in our example. The corresponding PDF of all three particles would be a delta-functional, each centered at $x = 1/3$, with an extra normalization of three, so that the full contribution still amounts to one (ii). A possible form, expected by the SM, is having three quarks that interact with each other via gluons. The dynamics would then lead to the delta functions to smear out as momentum exchange happens via gluons (iii), visible in figure 3. Further adding higher order processes, such as virtual quark pair productions from gluons (iv), would increase the PDF at low x values. This stems from the inverse quality of the gluon propagator describing the time-relation of gluons, see [6].

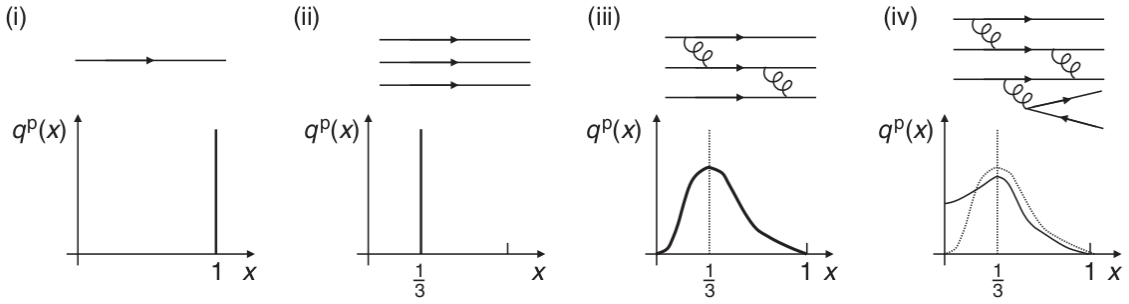


Figure 3: Four special forms of the PDF of quarks $q^p(x)$, taken from [6]

Neglecting contributions made by quarks further down the SM, the dynamic PDF only containing up and down quarks and their antiparticles, while also adding isospin symmetry, gives the following expressions for the structure functions F_1 and F_2 needed in cross sections calculations. These functions describe electric charge and current distributions.

$$F_2(x) = 2xF_1(x) = x\left(\frac{4}{9}u(x) + \frac{1}{9}d(x) + \frac{4}{9}\bar{u}(x) + \frac{1}{9}\bar{d}(x)\right) \tag{2.8}$$

To summarize, the quark–parton model accurately represents phenomena seen in experiments. It explains the disappearance of energy-scale dependence (Q^2) of structure functions, known as Björken scaling, and the Callan-Gross relation, which naturally describes the direct relationship between purely electromagnetic (F_1) and magnetic (F_2) structure functions.

$$F_{1,2}(x, Q^2) \to F_{1,2}(x) \tag{2.9}$$
$$F_2(x) = 2xF_1(x) \tag{2.10}$$

Furthermore, the scale factor x representing the fraction of the momentum carried by a respective particle is directly linked to the energy scale or "mass" of the virtual photon of the deep inelastic lepton-hadron scattering, where M is the mass and v the loss of energy through scattering.

$$x \equiv \frac{Q^2}{Mv} \tag{2.11}$$

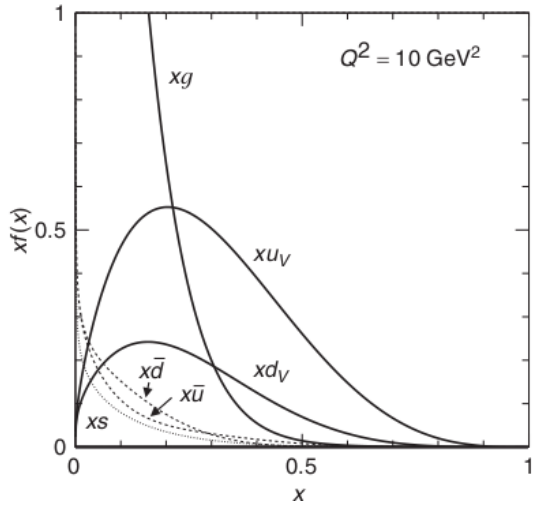


Figure 4: Forms of the PDF made with MRST at $Q^2 = 10GeV$, taken from [6]

To show that x represents a fraction, one can rewrite the scale factor in the following way:

$$x = \frac{Q^2}{Q^2 + W^2 - m_p^2} \tag{2.12}$$

W is the four momentum of the proton and m_p its mass. Since the four-momentum is always greater than the mass ($W^2 \geq m_p^2$) and the negative four-momentum of the exchange particle is greater than zero ($Q^2 \geq 0$), one obtains:

$$0 \leq x \leq 1$$

The case $x = 1$ corresponds to an elastic scattering. In figure 4, the PDFs of the valence quarks xu_V/d_V, the gluon xg and the small contribution of the strange quark xs are shown. These forms serve as a first hint for constructing a suitable PDF for the Higgs. As the parton distribution functions $PDF_p^i(x)$ describe the number densities of certain constituents i in the proton, they naturally have to fulfill momentum and charge sum rules[3, p. 33]:

Conservation of momentum:

$$\sum_i \int_0^1 dx x PDF_p^i(x) = 1 \tag{2.13}$$

Conservation of charge:

$$\sum_i \int_0^1 dx q_i PDF_p^i(x) = e \tag{2.14}$$

The inclusion of the Higgs PDF will result in a reweighting of the other PDFs involved as it impacts the momentum sum rule, transferring momenta contributions to the Higgs. The renormalization needed will be discussed in section 3.3.

2.4 Composite states and why the Higgs

The Higgs field needed to describe the effect of the Higgs in perturbative QFT is introduced as a composite (or local) operator. This corresponds to a local product of fields and their derivatives. There are several possibilities to construct an adequate operator, as any polynomial of fields is possible[10]. A simple choice would be:

$$\mathcal{O}_0(x) = \left(\phi^\dagger \phi\right)(x) \tag{2.15}$$

where $\phi(\vec{x}, t) = \phi(x)$ is the Higgs field with vacuum expectation value. These operators share similarities with hadron operators in QCD, and therefore should be non-perturbative. The precondition for its use is an active Brout-Englert- Higgs (BEH) effect, see [11].

Starting off, one expands the Higgs field in a fixed 't Hooft gauge in the propagator build from 2.15 around the expectation value by $\phi(x) = v + \eta(x)$, and then takes the connected part only. This yields

$$\begin{aligned}\left\langle \mathcal{O}_0^\dagger(x)\mathcal{O}(y)\right\rangle_{con.} &= v^2\langle\eta(x)\eta(y)\rangle \\ &+ v\left\langle \eta^\dagger(x)\eta^\dagger(y)\eta(y) + x \leftrightarrow y\right\rangle \\ &+ \left\langle \eta^\dagger(x)\eta(x)\eta^\dagger(y)\eta(y)\right\rangle\end{aligned} \tag{2.16}$$

If one were to work with standard perturbation theory, the whole expression could be treated physically. Since we are working in the frame of gauge invariant perturbative field theory, the so-called Gribov-Singer ambiguity arises and only the sum of this expression is physical. One could calculate the right hand side with Lattice field theory, but using PDFs is sufficient here. Because of the representation theory of SU(3) and SU(2), there is no way to create a gauge-invariant baryon with quarks alone in the standard model. The simplest operator one can create has the form [12]:

$$P(x) = (qqq\sigma)(x) = (p\sigma)(x),$$

Here, the quarks are all contained in the operator $p(x)$. If we use the FSM mechanism and only look at the LO, we obtain $P \approx vp$ and end up with normal QCD, where the operator $p(x)$ will be examined non-perturbatively. The question now is how to expand the operator in a suitable way. A short qualitative explanation of why the Higgs serves as a candidate to expand the operator follows.

What we want to end up with is a proton that is gauge-invariant, the same in every reference frame. Conclusively, it has to be invariant under strong interactions. Furthermore, because the quarks carry weak charges, we also need invariance with regard to the weak interaction. Because of spin conservation and charge conservation, only weak fundamentally charged bosons qualify as possible candidates[7]. The only particle in the SM fitting these descriptions is the Higgs boson.

3 Technical approach

3.1 Perturbartive treatment

In the following sections, the change from the SM proton to a slightly modified proton that includes the Higgs particle will be numerically analyzed, using the Monte-Carlo event generator HERWIG++ as primary tool, set at an energy scale of $13TeV$[13][14][15]. Scatter processes made by the simulation will only include LO approximations and the energy-scale dependency Q^2 will be omitted, since including higher order terms (NLO and forth) would increase the time needed significantly and go beyond the scope of what is possible here. It should be noted that the Q^2-dependence of the parton distribution function $PDF(x, Q^2)$ can be calculated with the so called parton evolution functions, known as DGLAP equation[16]. Consequently, as long as one finds a suitable PDF in a certain range of Q^2, one can propagate it through the DGLAP mechanism without having to check multiple energy scales.

As one might imagine, there numerous possible final states for proton-proton collisions. The next section will illustrate how one can reduce that number while still preserving physicality of the underlying processes. To do so, we will look at the underlying dynamics by first introducing relevant Feynman diagrams.

3.2 Feynman diagrams

Feynman diagrams are a key element of particle physics, offering a visual representation of the processes taking part in the interaction and describing the interchanges of particles involved. They feature all time-orderings of particle exchanges and serve as a good foundation of understanding the decay processes analyzed. Some important traits will be discussed below.

Feynman diagrams represent the transition in QFT from one state to another. Initial states are drawn to the left, final states are drawn to the right. The particles drawn in the middle section are virtual, representing the sum of all possible time ordered interactions of the spin-1 bosons, transferring the state from left to right. As of now, these can be gluons, photons, W/Z-bosons and the Higgs boson. As their lifespan depends inversely on their coupling constant α to a given field, gluons and the Higgs in particular have a very short lifespan. Consequently, most of the intermediate particles cannot be measured directly.

When one knows the Feynman diagram of a process, one can find the transition matrix T_{fi} describing the processes. It contains the coupling constants, describing to the probability of a curtain spin-half fermion to emit or absorb a boson of its interaction. Conservation laws of QFT expressed in algebraic terms can be transferred to Feynman diagrams. For example, the direct consequence for color charge conservation is that color-lines of gluons must be drawn either in the initial or final state[3], giving restrictions for possible processes.

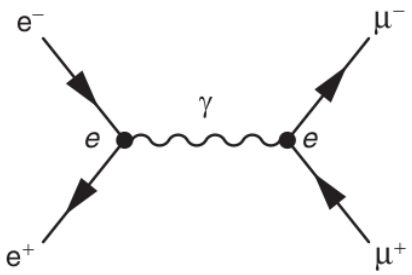


Figure 11: Example of a Feynman diagram for a QED annihilation process of $e^+e^- \to \mu^+\mu^-$, taken from [6]

The arrow of particles always goes from left to right, while they are reversed for anti-particles, which traces back to the unphysical interpretation of antiparticles being particles with negative energies moving backwards in time [6, p. 98]. It should be noted that completely describing one process of initial and final particles is an impossible task, as there exists an infinite number of possible processes and Feynman diagrams. Luckily, the values associated with the transition matrix become smaller if there are more interaction vertices, so when analyzing a specific process, only looking at LO processes should present information with errors small enough to be of value.

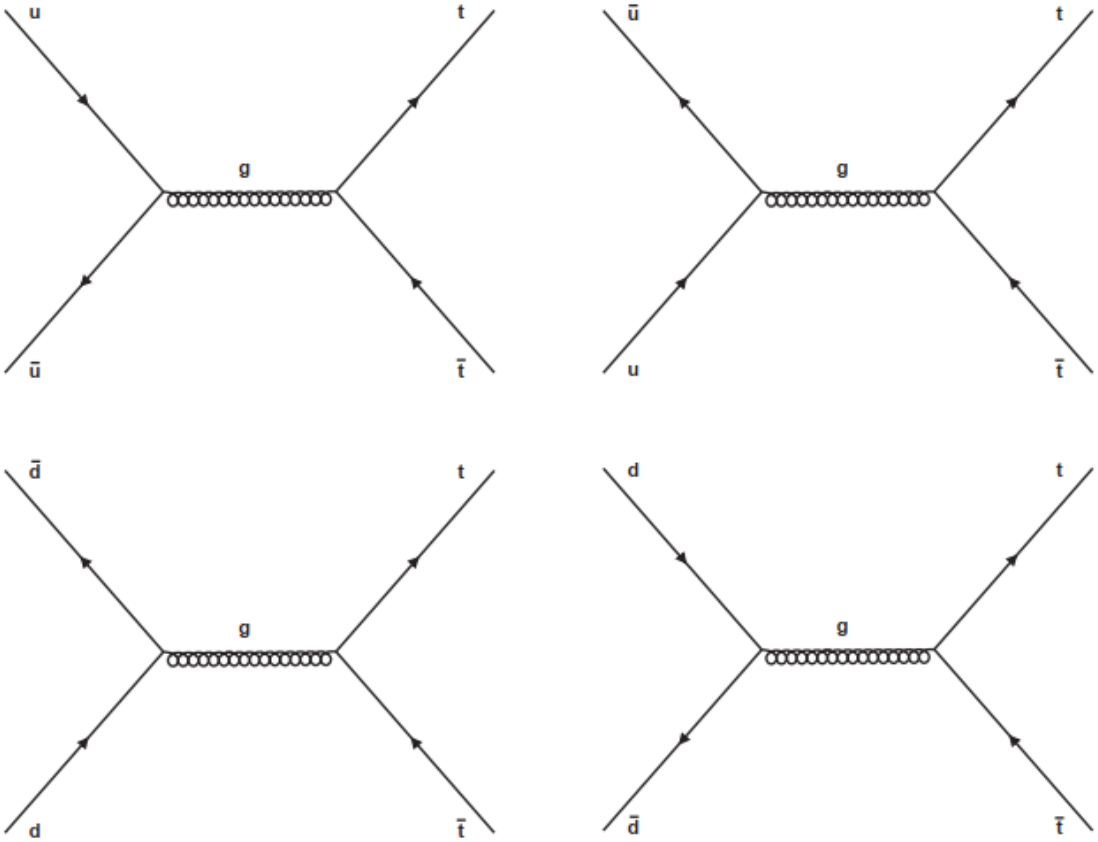


Figure 5: Feynman diagrams for up and down quarks and their antiparticles

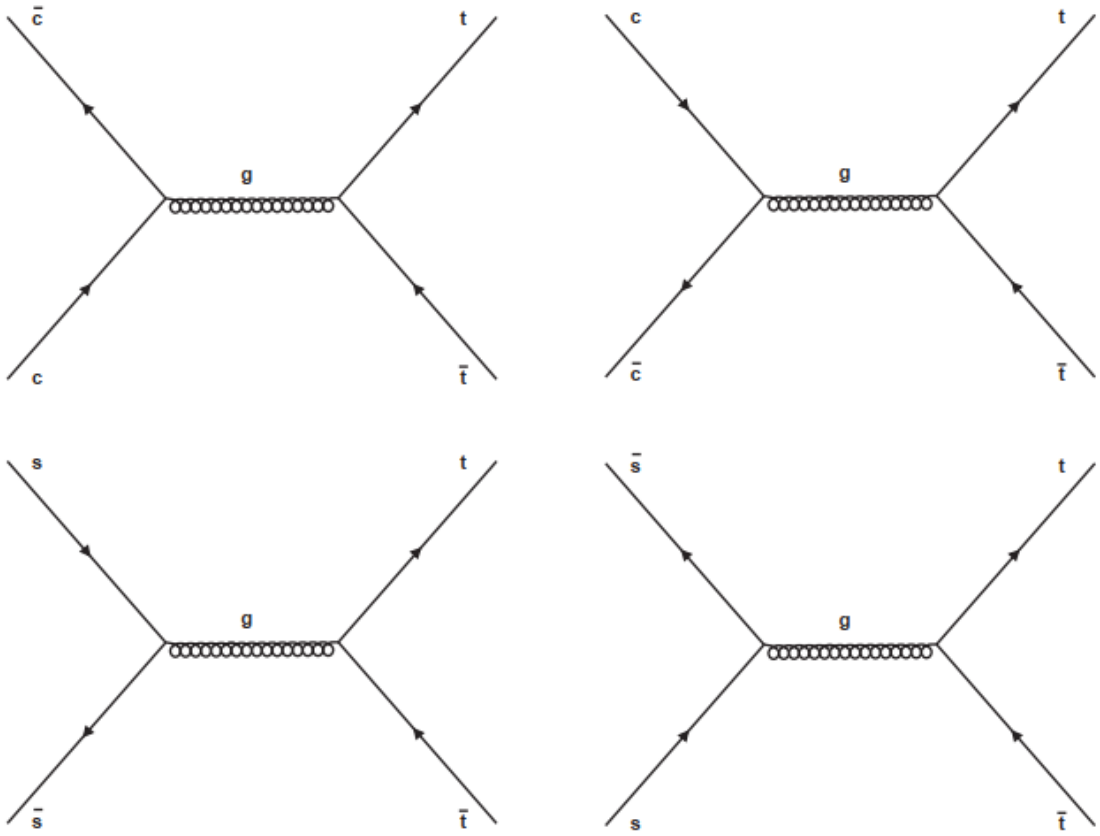


Figure 6: Feynman diagrams for charm and strange quarks and their antiparticles

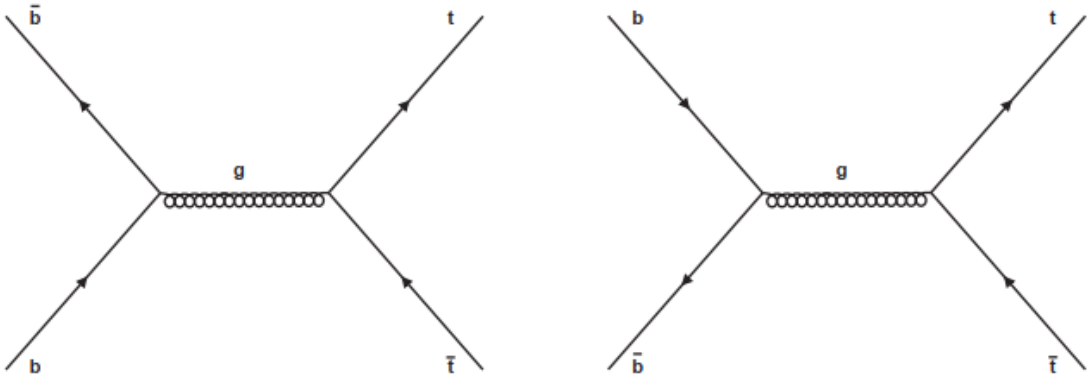


Figure 7: Feynman diagram for bottom quarks and antiparticles

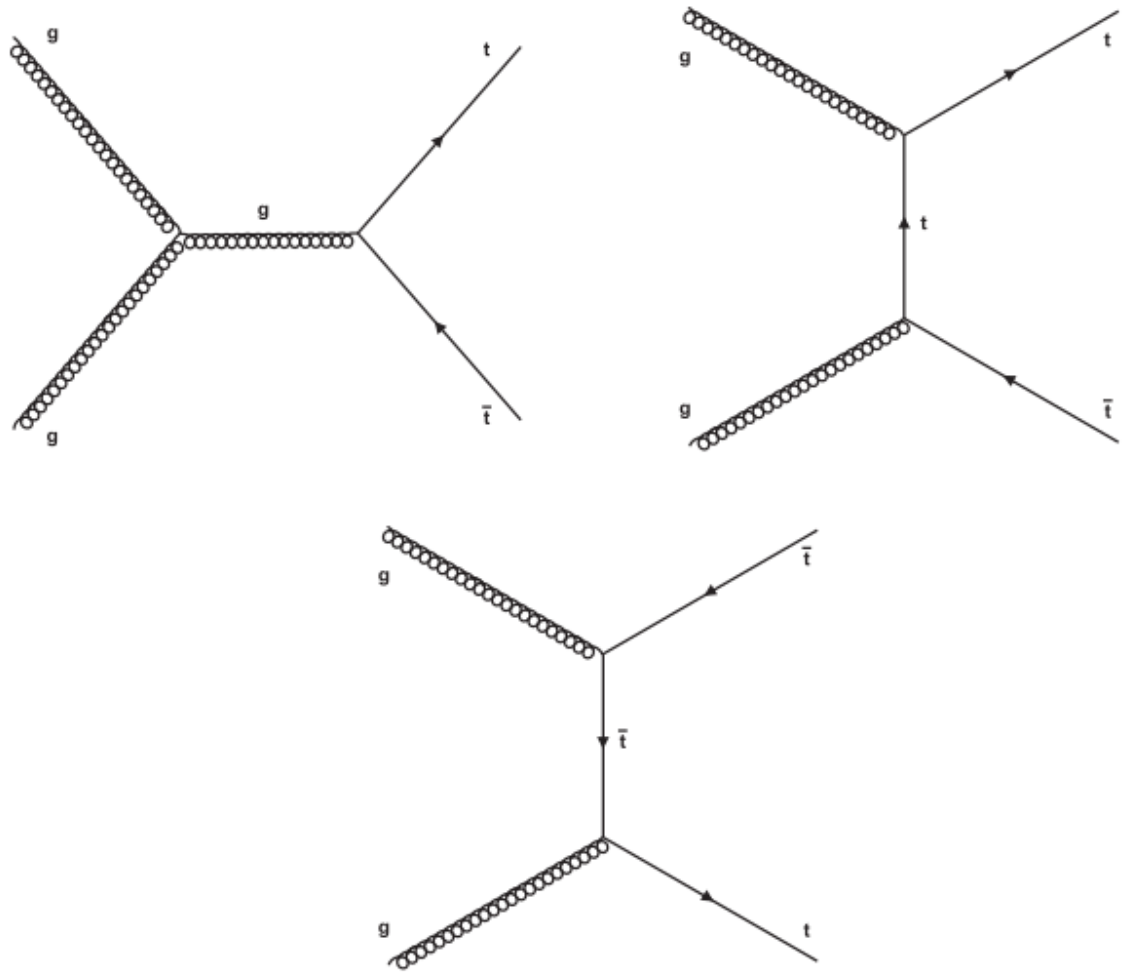


Figure 8: Feynman diagrams for gluons, figure 6 to figure 8 taken and modified from [4]

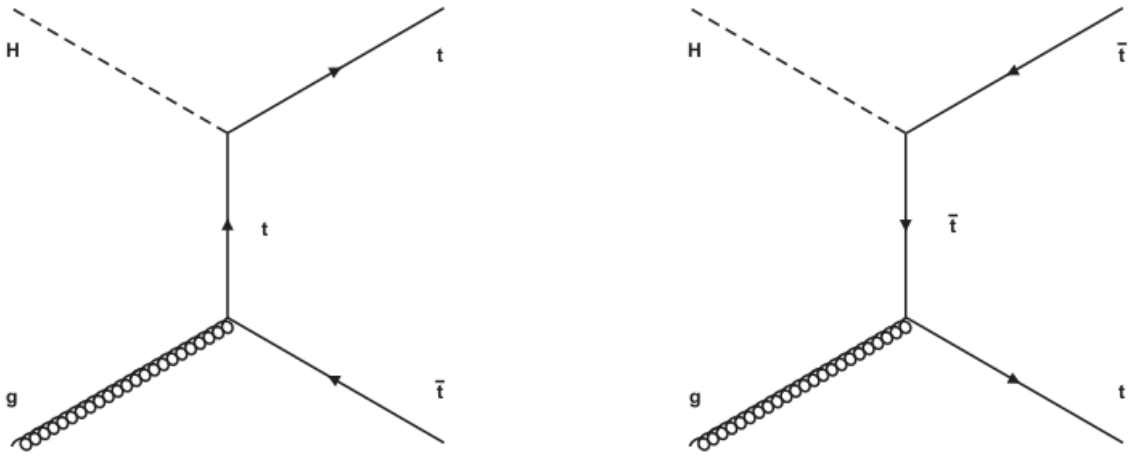
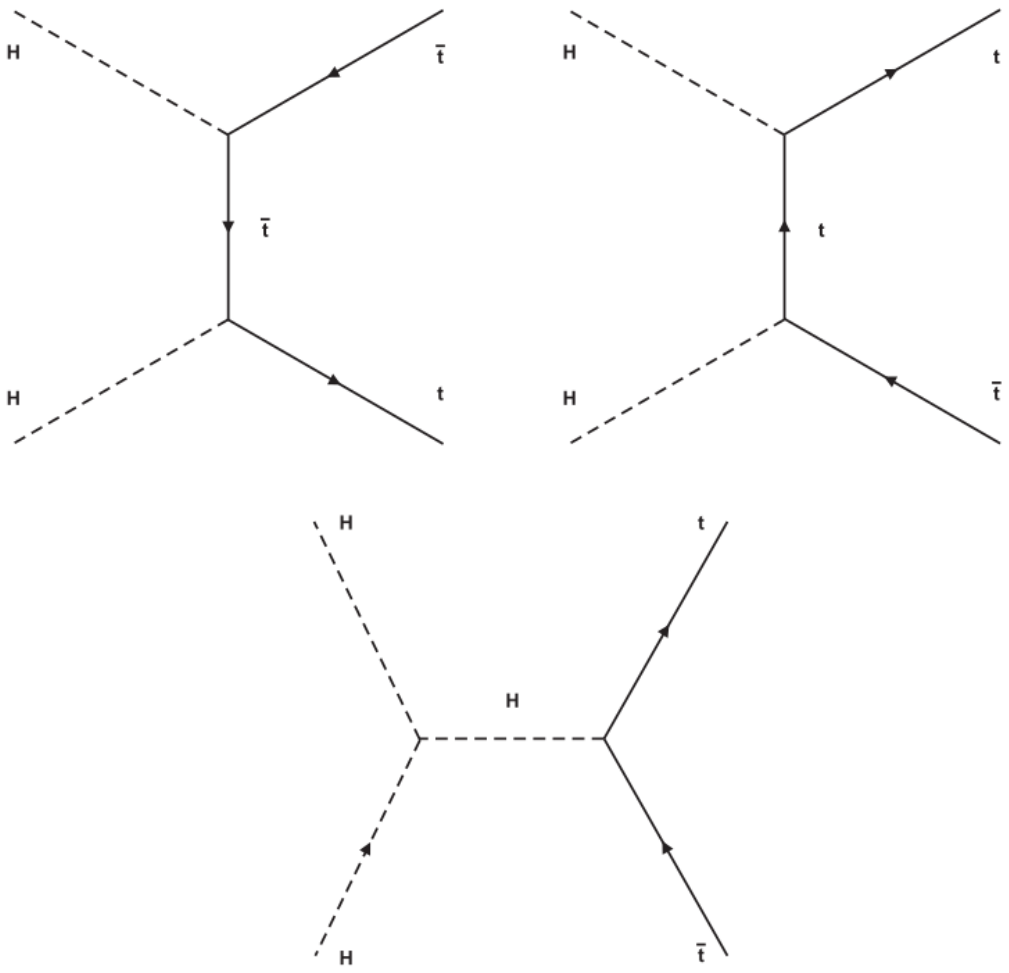


Figure 9: Feynman diagrams for processes involving gluon and Higgs in initial state



.

Figure 10: Feynman diagrams for processes involving a Higgs-Higgs pair in initial state, figure 9 and figure 10 taken and modified from [3]

A full list of Feynman diagrams of used processes in LO are provided in figures 5 to 10, for further reference refer to [6]. The processes for the quark decays and the proton-proton decays are taken from [4, p. 11ff] and [3, p. 29]. Additionally, they are rebuild in Adobe Illustrator to ensure a consistent visual representation. They could also be taken directly from the diagram generator MadGraph included in HERWIG++. In figures 5 to 7, the protons are broken down into their SM subparticles decaying into $t\bar{t}$ particles via gluons g, while figures 8 and 10 show the decay of all the relevant Higgs processes with results $t\bar{t}$ in LO.

3.3 Initial state with the Higgs

Since it is assumed that there is a Higgs contribution in the initial state, we want to limit the number of processes to include. The following section will be used to gain a better understanding of the scattering process with initial Higgs and the processes involved leading to said conclusions. As seen in [3], only processes involving top quarks that are coupled to the Higgs particle provide a significant contribution to cross sections. This stems from the fact that the constant of fermion-higgs coupling is the inverse of the Higgs expectation value in the vacuum, $v = 4 \times 10^{-3} GeV^{-1}$. However, top quarks will not appear in the initial process, as Herwig does not yet provide a way of including them there and their PDF's are negligible. As a consequence, the contribution of the Higgs in initial states should lead to an increased effect in processes with top quarks in the final states. Since the total cross section decreases sufficiently fast as one increases the number of particles in the parton state, only decays with two resulting particles will be considered. Because of the color conservation seen in section 3.2, the only possible cases for one Higgs in the initial state and one pair of top quarks after the decay are the pairs gH and HH, giving the prior list of Feynman diagrams. Adding another Z-boson to the final state would increase the number of processes and the sensitivity to the Higgs considerably, which is not done here due to time constraints.

In short, the scattering amplitudes of the SM proton $sm \to t\bar{t}$, modified proton with Higgs $pp \to t\bar{t}$, and processes involving the Higgs $gH \to t\bar{t}$ and $HH \to t\bar{t}$ will be investigated. The impact on the Higgs particle in said dynamics will be discussed in section 4.1, together with the possible Higgs-PDF and its fraction inside the proton in section 4.2. But before that, the problem of how to include the Higgs particle and how this changes other PDFs will be investigated.

3.4 Adding the Higgs-PDF

Before looking at possible ways of coming up with a suitable PDF, it has to be argued how the new PDF can be added to quark- and gluon-PDFs in a way where calculations are bounded by a reasonable scope, while also trying not to lose physical meaning after approximations. To start off with, using momentum sum rules 2.13, it immediately follows that adding the Higgs leads to a change in all other particle PDFs, which could theoretically be solved by doing a global refit of all parameters involved. Since this would blow up calculations, the choice taken here is to normalize the weighting of all the contributions to the modified proton pp and its subprocesses. The whole normalization is then factored out from the PDF

$$PDF_H^{new}(x) = f \times PDF_H^{old}(x), \tag{3.1}$$

Here, the parameter f will be the fraction of the Higgs. One would need to rescale the gluon PDFs used for the proton in HERWIG++ or alternatively the contributing cross sections with gluons in the initial state σ_{gf}, σ_{gg} and σ_{gH}, splitting the SM proton in the simulation. This in turn would mean splitting up the standard model proton, because it is treated as a whole by HERWIG++. The more reasonable alternative is to equally subtract the fraction of the Higgs from all the other PDFs, setting $f_i = f_H = f$. This leads to a momentum sum violation that increases with a narrower peaked PDF for the Higgs. To avoid this violation as much as possible, one can flatten the form of the PDF, suggesting first constraints on the form of the PDF[3, p. 34ff].

Another exclusion is gained by sorting out the PDFs which do not allow values for f while exceeding the maximal experimental error. These errors are based on the precisions of [17] and [18] made by the ATLAS and CMS detectors:

$$\frac{\sigma_{tot}(f)}{\sigma_{sm}} = 1 \pm \Delta_{rel}^{t\bar{t}} = 1 \pm 0.03 \tag{3.2}$$

Since LO approximations lead to a larger deviation of simulation data and experimental results than allowed, it is assumed that the difference scales similarly for all contributions to the total cross section. Hence, all total cross sections σ_{PP}, σ_{gH} and σ_{HH} are rescaled by the ratio of σ_{sm} and σ_{PP}. This results in a quadratic equation, where the smallest, real and positive solution is the desired Higgs fraction.

$$\begin{aligned}
\frac{\sigma_{tot}(f)}{\sigma_{sm}}^{scaled} &= 1 \pm \Delta_{rel}^{t\bar{t}} = \frac{\sigma_{sm}}{\sigma_{PP}} \frac{1}{\sigma_{sm}} \left((1-f)^2 \sigma_{PP} + f(1-f)\sigma_{gH} + f^2 \sigma_{HH} \right) \\
&= (1-f)^2 + f(1-f)\frac{\sigma_{gH}}{\sigma_{PP}} + f^2 \frac{\sigma_{HH}}{\sigma_{PP}} = \left(1 + \frac{\sigma_{HH} - \sigma_{gH}}{\sigma_{PP}} \right) f^2 + \left(\frac{\sigma_{gH}}{\sigma_{PP}} - 2 \right) f + 1 \\
0 &= \left(1 - \frac{\sigma_{HH} - \sigma_{gH}}{\sigma_{PP}} \right) f^2 + \left(\frac{\sigma_{gH}}{\sigma_{PP}} - 2 \right) f \mp \Delta_{rel}^{t\bar{t}}
\end{aligned} \tag{3.3}$$

3.5 Limiting the search

The ansätze made by [3] and [4] serve as starting point for including differential cross sections and their change resulting from said PDFs. The general ansatz for the Higgs PDF $PDF^H_{gen.}$ one could choose based on assumptions made by, again, [4] and [3], has the following form:

$$PDF^H_{gen.}(x) = (1-x)\left(c_1 \exp\left(-c_2 x^2\right) + c_3 \exp\left(-c_4\left(x - c_5\right)\right) + \frac{c_6}{x}\exp\left(-c_7 x^2\right)\right) \tag{3.4}$$

To give an insight into this rather general form of a PDF, the first thing to notice is the factor $1-x$. This is added because the Higgs PDF cannot contain all the momentum of the proton, therefore requiring a term that decreases as the fraction x approaches one. A modified version of this term will be used in the new PDF. Furthermore, an intuitive guess is to assume that the form of the Higgs PDF is similar to one of the quark PDFs, resulting in exponential forms seen in 3.4.

However, since using this full ansatz would take too much time overall, splitting it into smaller pieces and looking at the parts giving results closer to the experiment makes calculations much easier. One would also have to make a generalized PDF of above equation and include on-shell and off-shell PDFs, but since the energy scale used is about $13TeV$, the Higgs will be mostly on shell and this problem can be avoided. Based on prior assumptions, the focus of further research will be the following PDF:

$$PDF^H(x) = (1-x)^a x^b e^{-cx} = PDF^H_R(x) e^{-cx} \tag{3.5}$$

It is based on the PDF made by Reiner [4] of the form $PDF^H_R(x) = (1-x)^a x^b$ while adding an additional degree of freedom by introducing an exponential dampening term.

First, the total cross sections arising from the old density function PDF^H_R will be checked and compared to prior results made by [4], this time including the differential cross sections arising. Then, the range of c will be modified, hoping for the differential cross sections of gH- and HH-processes to result in forms similar to the sm-process.

4 Results

4.1 Cross section and dynamics

Starting with $PDF_R^H(x)$, the raw output data from HERWIG++ was used to calculate the relative error of theoretical and experimental cross sections, providing the range of allowed Higgs percentages. This was checked on the range of $a = [0,2]$, $b = [-2,2]$, $\Delta(a,b) = 0.1$ with 10000 simulations. This served as a check on the simulation program, as the values without the additional term agree, in the range of errors, with the ones calculated by [4]. This results in a best match for total cross sections at $a = 1.6, b = 0.6$, with a Higgs fraction of $f = (2.00 \pm 1.01)\%$. The differential cross sections of gH- and HH-processes, however, do not show the desired form, as seen in figure 12 with 100000 simulations.

In the next step, the new PDF with fixed values $a = 1.6$, $b = 0.6$ was used, with the third value in range $c = [1,1000]$, the density of these values first decreasing with larger c. At first, the allowed Higgs fraction f was investigated by plotting the ratio of experimental and theoretical cross sections, allowing an error of 3% described by 3.2. The change of allowed fractions are presented in figure 25 to 30. As can be seen in the appendix A, small fractions f are allowed in general, while increasing the dampening caused by the parameter c reduces the allowed fraction to $f \leq 5\%$ for large values. The slope of cross section ratios does not change significantly for $c \gtrapprox 10$, and since values of interest are inside that region, no further discussion is necessary.

After first calculations, the density of values was increased in the area of similar structures of differential cross sections for sm, gH and HH. The smallest distance of c-values used was $\Delta c = \pm 2$, as the overall range of c was significantly larger than the ones of a and b before. The results are a shift of the differential cross sections of gH- and HH-processes towards sm-processes for larger c-values. Figures 13 to 20 show the overall structure for the logarithm of differential cross sections $\frac{d\sigma}{dpT_T}$ with respect to the momentum of the t-quark along the collision axis pT_T. To quantify the results, figures 13 to 20 show the logarithmic values of three subprocesses sm, gH and HH. Linearly fitted curves of the form below, in range of relative errors smaller than 50%, are included as well, as can be seen in the ratio subplots:

$$log\left(\frac{d\sigma}{dpT_T}\right) = k \times pT_T + d \tag{4.1}$$

In figure 21, the slopes of 4.1 are calculated with least-squares and drawn in range $c = [1,1000]$, highlighting the overall decrease of steepness for bigger values, especially for the HH-process. Errors are calculated from fitparameters corresponding to upper and lower error bounds of simulation values. Figure 22 further emphasizes the area of where slopes cross, giving an overlap of for k at $c^{gH} = 80 \pm 20$ for the gH-process and $c^{HH} = 51 \pm 7$ for the HH-process. The errors of c-values are estimated by taking the range where the slopes of gH- and HH-processes overlap within their errors. In this area, the density of c-values was increased. For PDF with $c = 80 \pm 20$ and larger, the differential cross section curve for both subprocesses decreases faster than the SM proton, showing a sufficiently fast decline of dynamics involved.

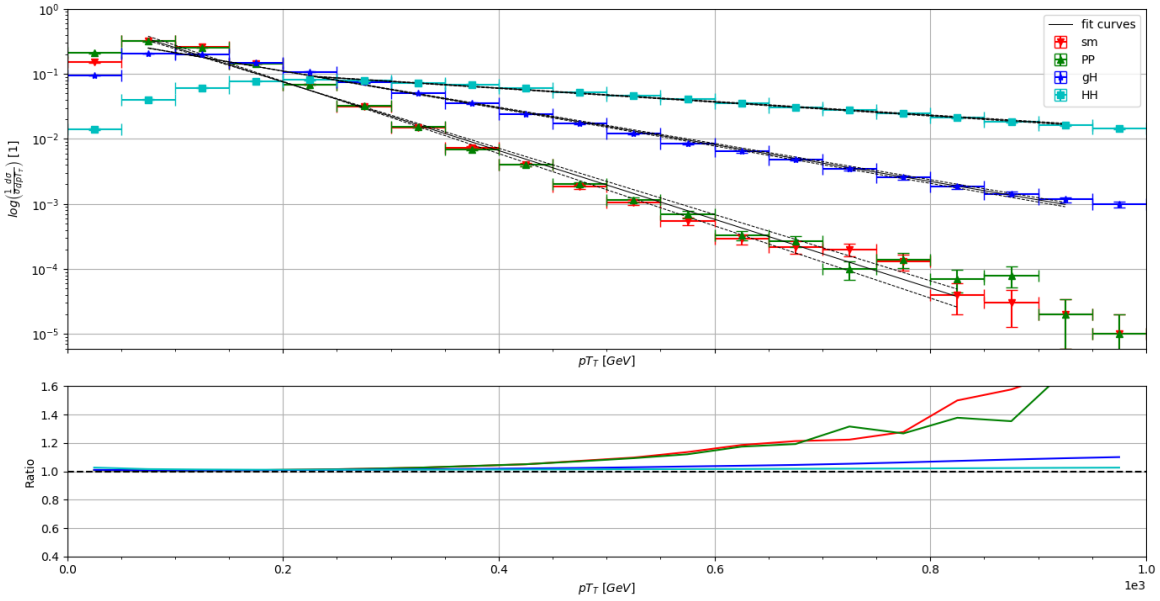


Figure 12: $PDF = (1-x)^{1.6}x^{0.6}$

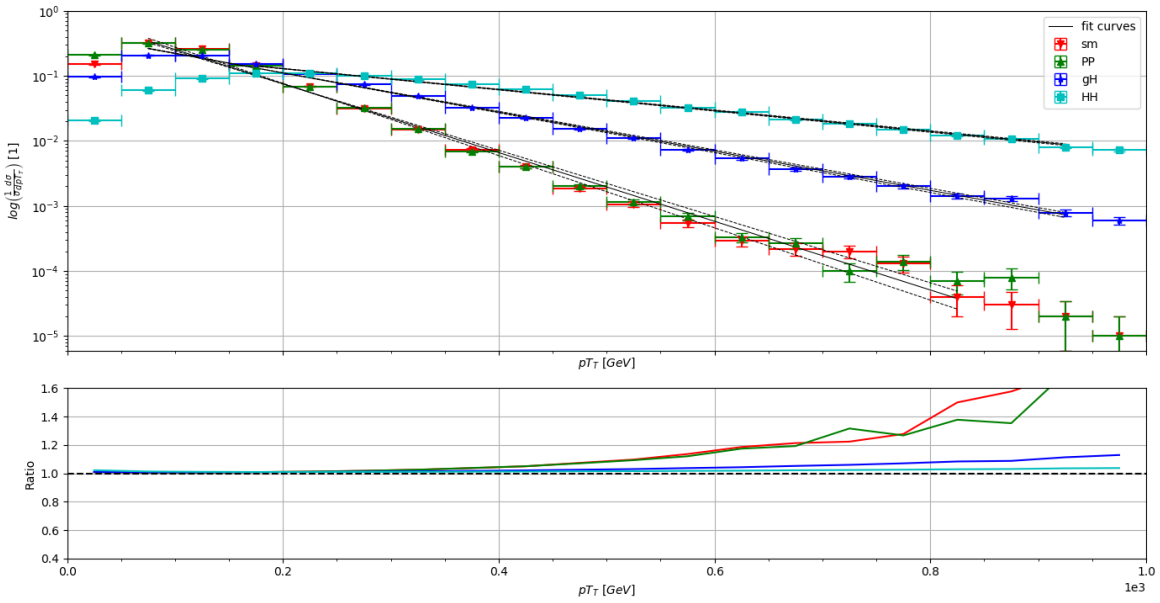


Figure 13: $PDF = (1-x)^{1.6}x^{0.6}e^{-5x}$

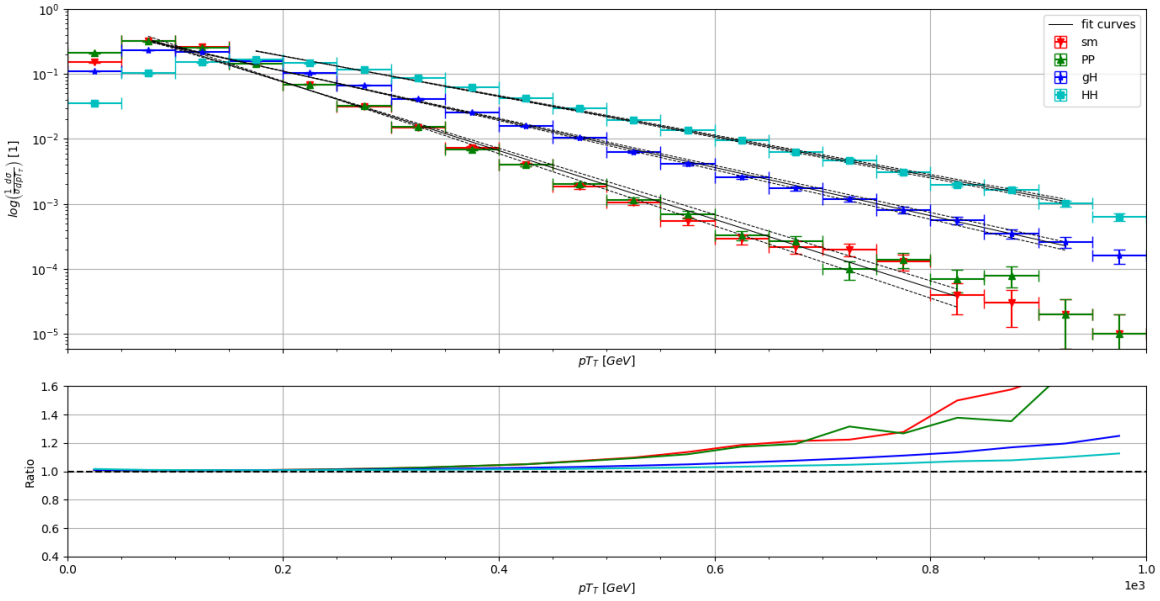


Figure 14: $PDF = (1-x)^{1.6}x^{0.6}e^{-20x}$

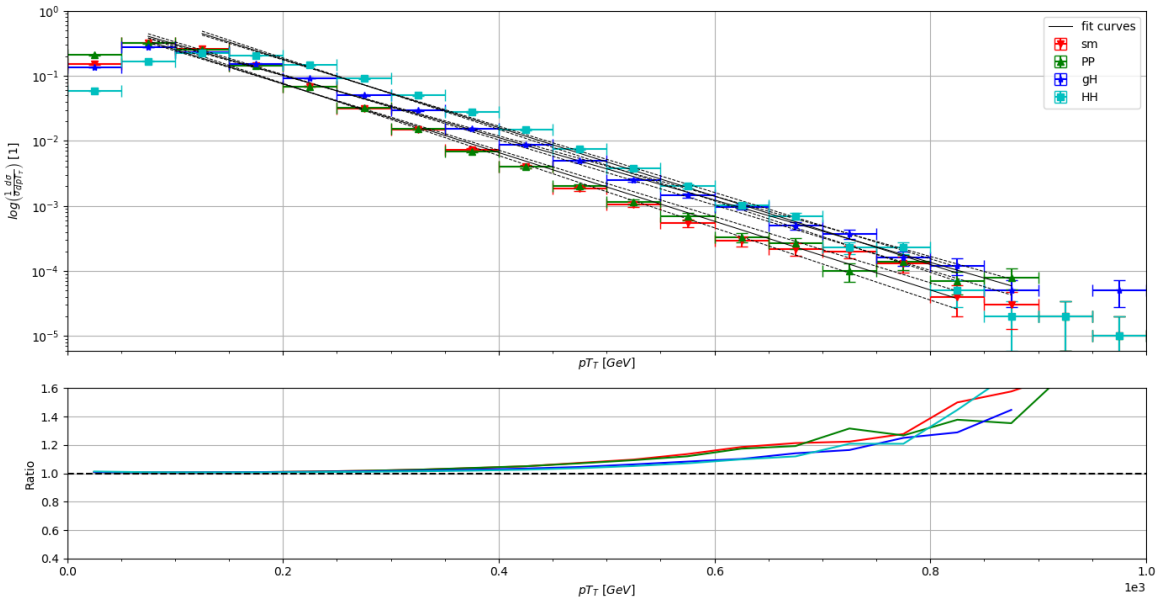


Figure 15: pT_T for $PDF = (1-x)^{1.6}x^{0.6}e^{-50x}$

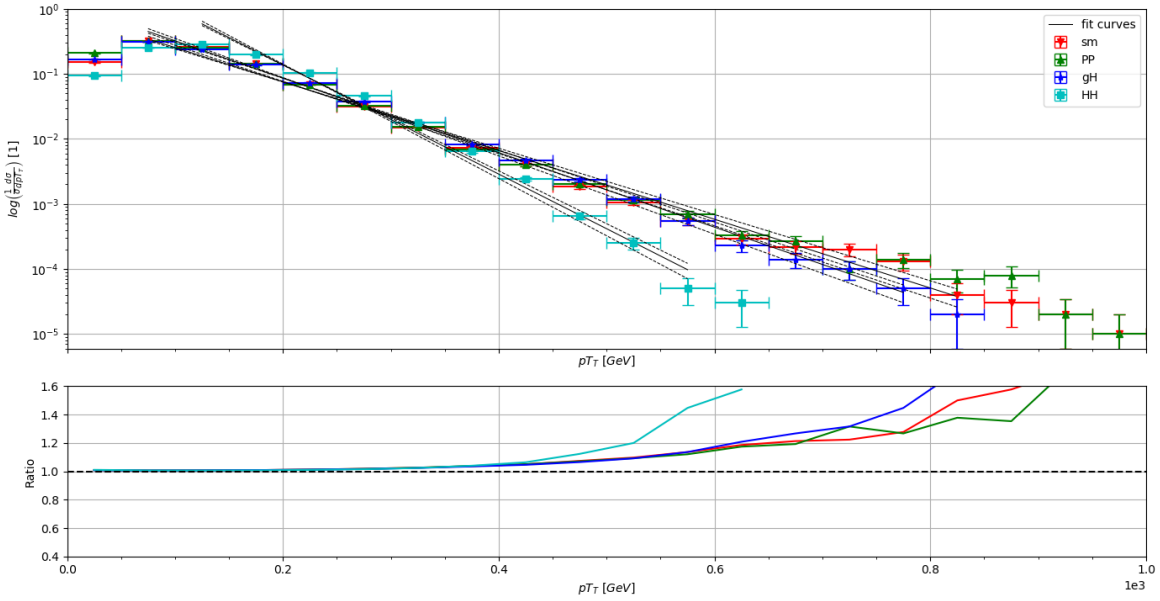


Figure 16: $PDF = (1-x)^{1.6}x^{0.6}e^{-100x}$

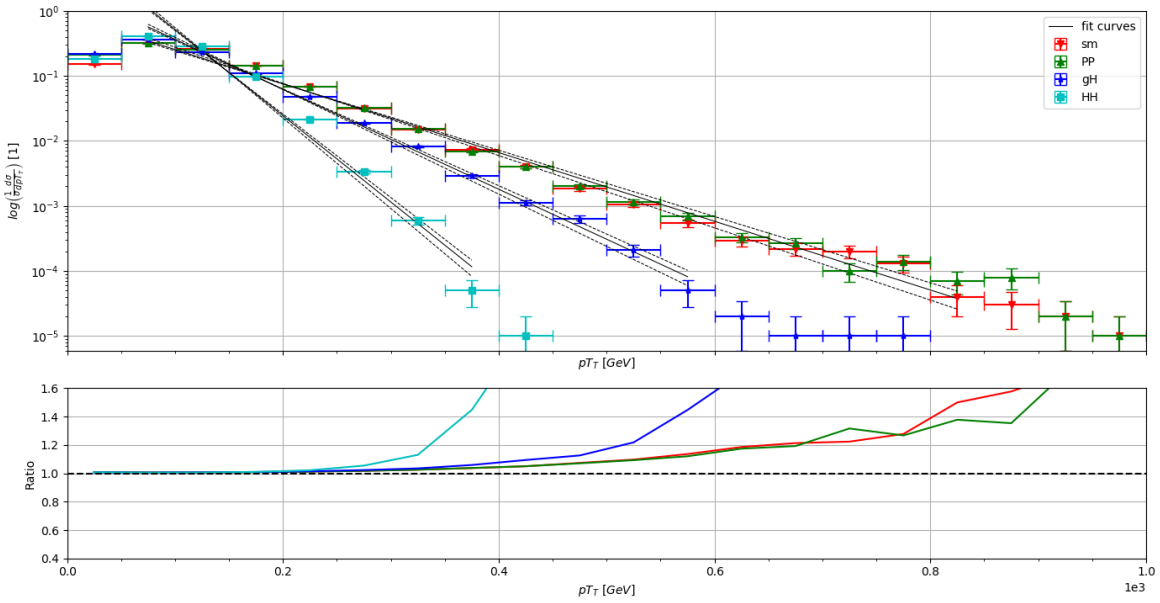


Figure 17: $PDF = (1-x)^{1.6}x^{0.6}e^{-250x}$

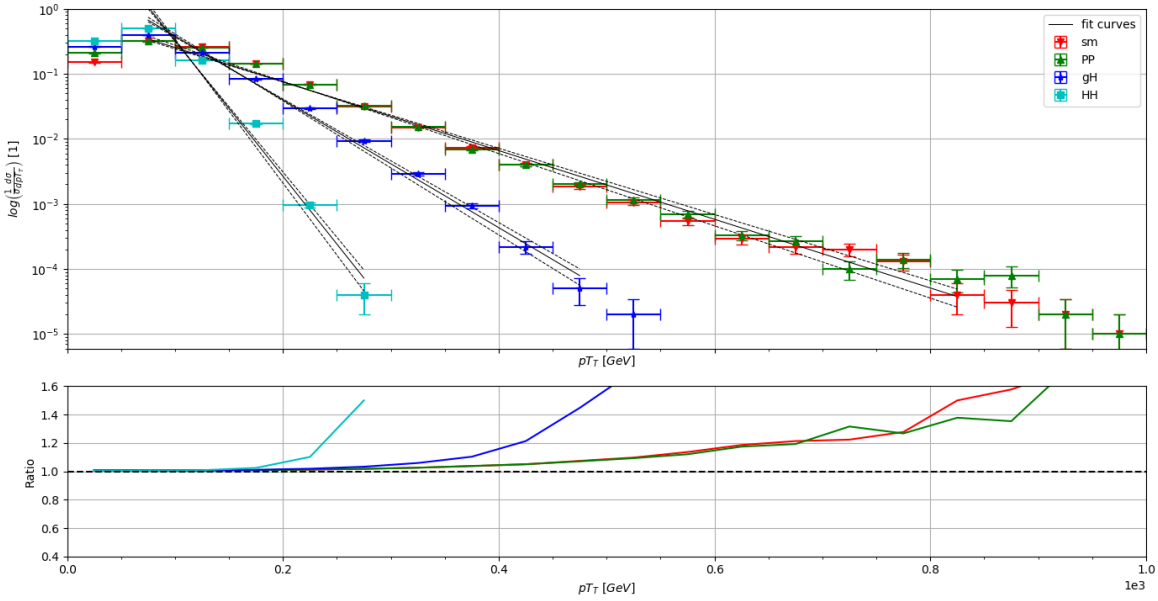


Figure 18: $PDF = (1-x)^{1.6}x^{0.6}e^{-500x}$

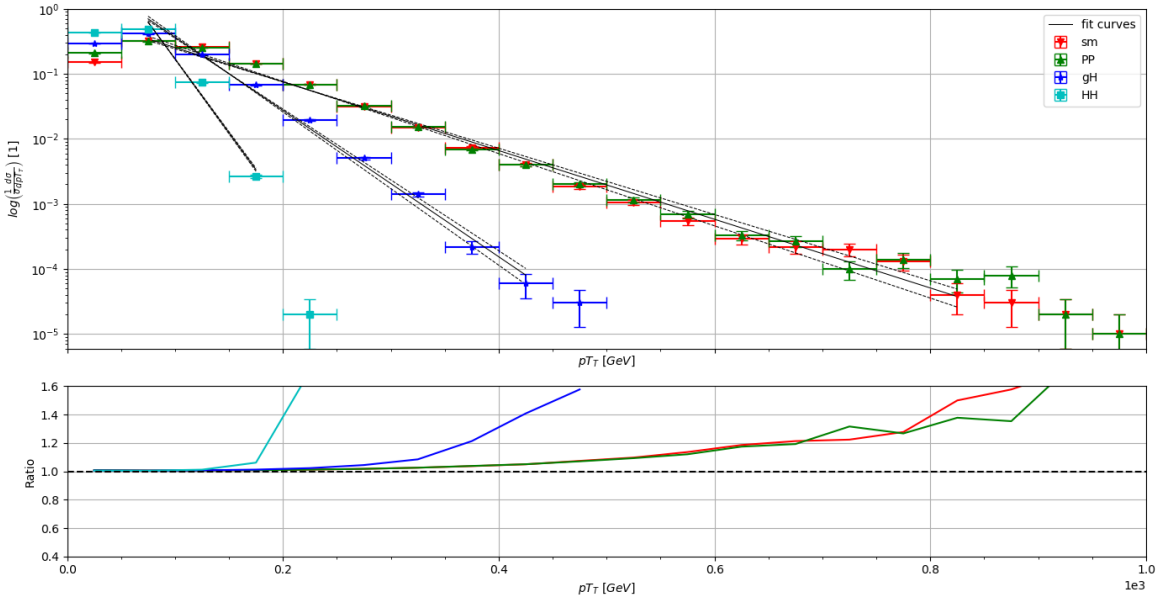


Figure 19: $PDF = (1-x)^{1.6}x^{0.6}e^{-750x}$

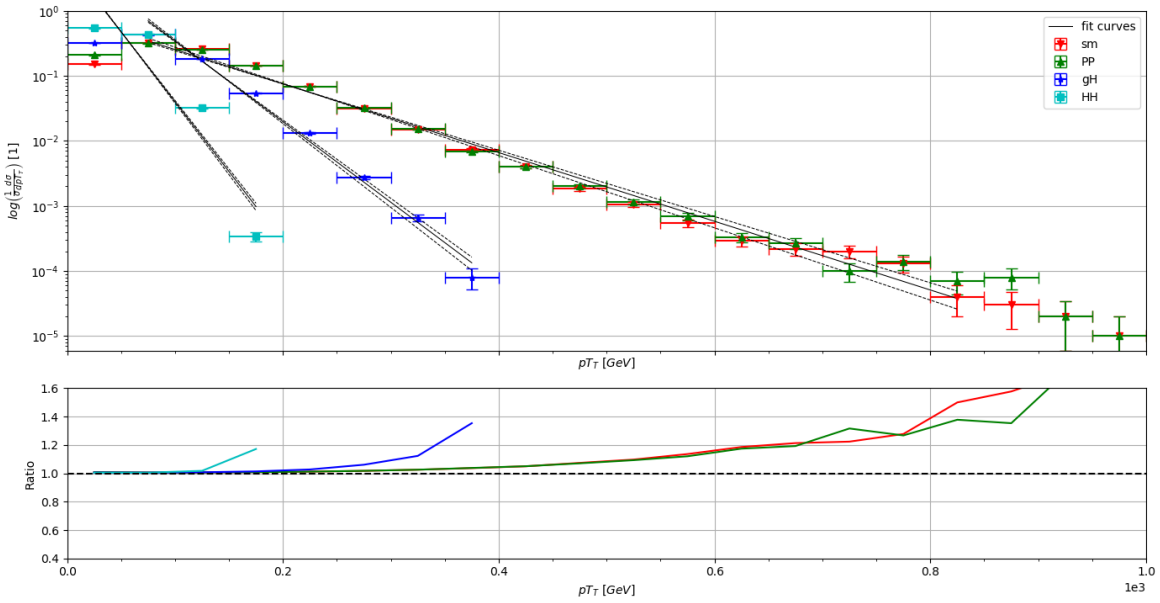


Figure 20: $PDF = (1-x)^{1.6}x^{0.6}e^{-1000x}$

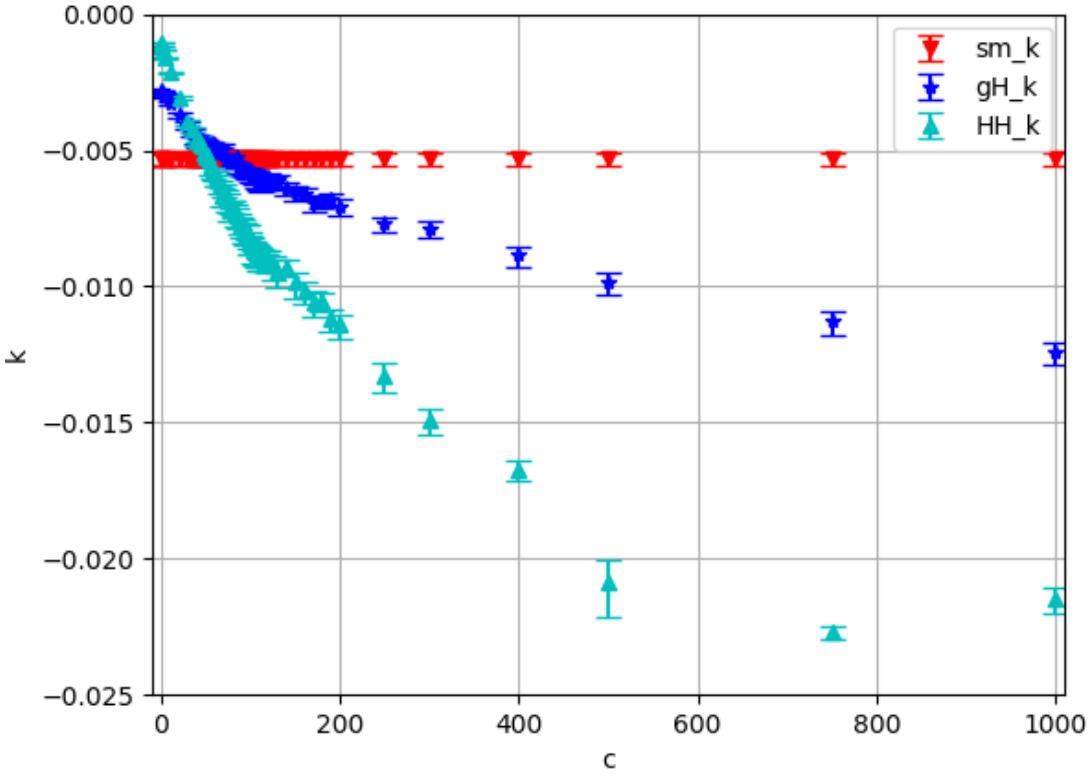


Figure 21: Slopes of fitcurves from 4.1

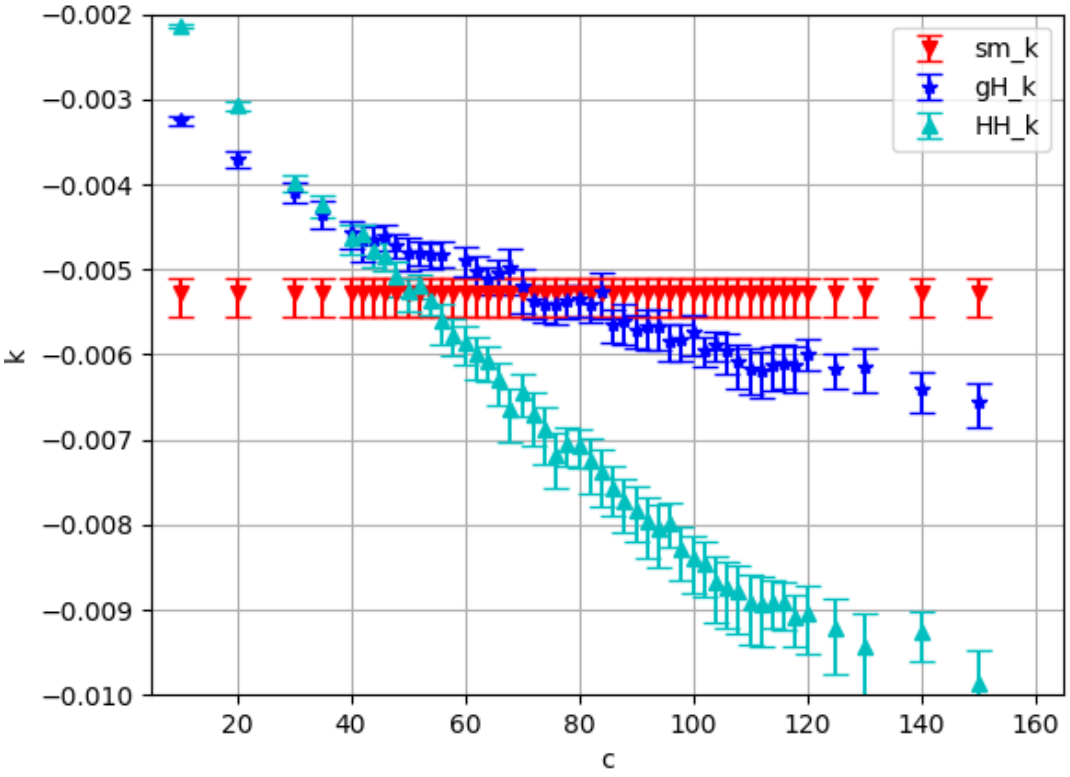


Figure 22: Slopes of fitcurves from 4.1 in overlap area

4.2 PDF and Higgs fraction

To give some insight in the PDFs involved, the prior PDF without the exponential term is drawn in figure 23, together with a selected few in range $c = [1, 1000]$ of modified ones. Figure 24 gives a comparison of PDFs where the slope of gH- and HH-processes crosses the SM proton slope, while the latter one satisfies overall decline of dynamics. The fractions of the Higgs contribution in each case are listed as well. We can see a decrease from $f = 2.00\%$ for $c = 0$ to $f = 1.51\%$ for $c = 1000$. Only smaller dampenings made by the exponential term e^{-cx} significantly contribute to a decrease in f, from $c = 100$ it stays constant with respect to numerical approximations. The corresponding PDFs are as follows, the second one being the more significant one as both processes gH and HH decline faster than the proton of the SM.

$$PDF^H_{HH}(x) = (1-x)^{1.6} x^{0.6} e^{-(55\pm5)x} \tag{4.2}$$

$$PDF^H_{gH}(x) = (1-x)^{1.6} x^{0.6} e^{-(75\pm15)x} \tag{4.3}$$

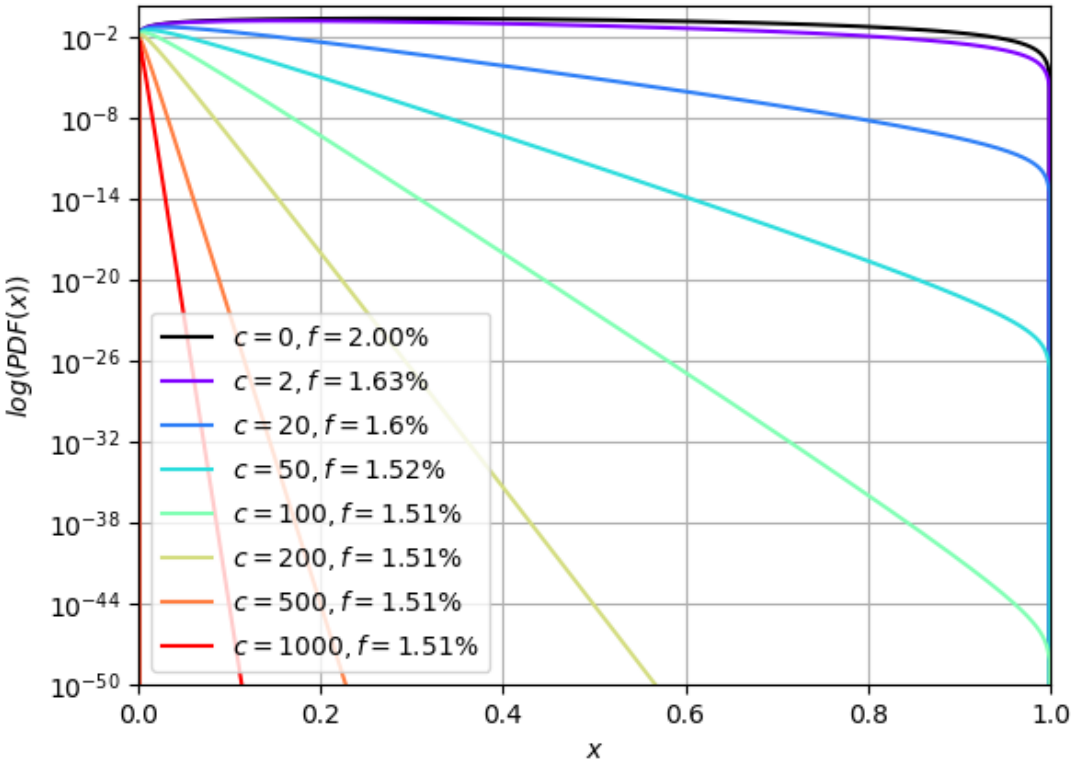


Figure 23: Comparison with old $PDF(x) = (1-x)^{1.6} x^{0.6}$

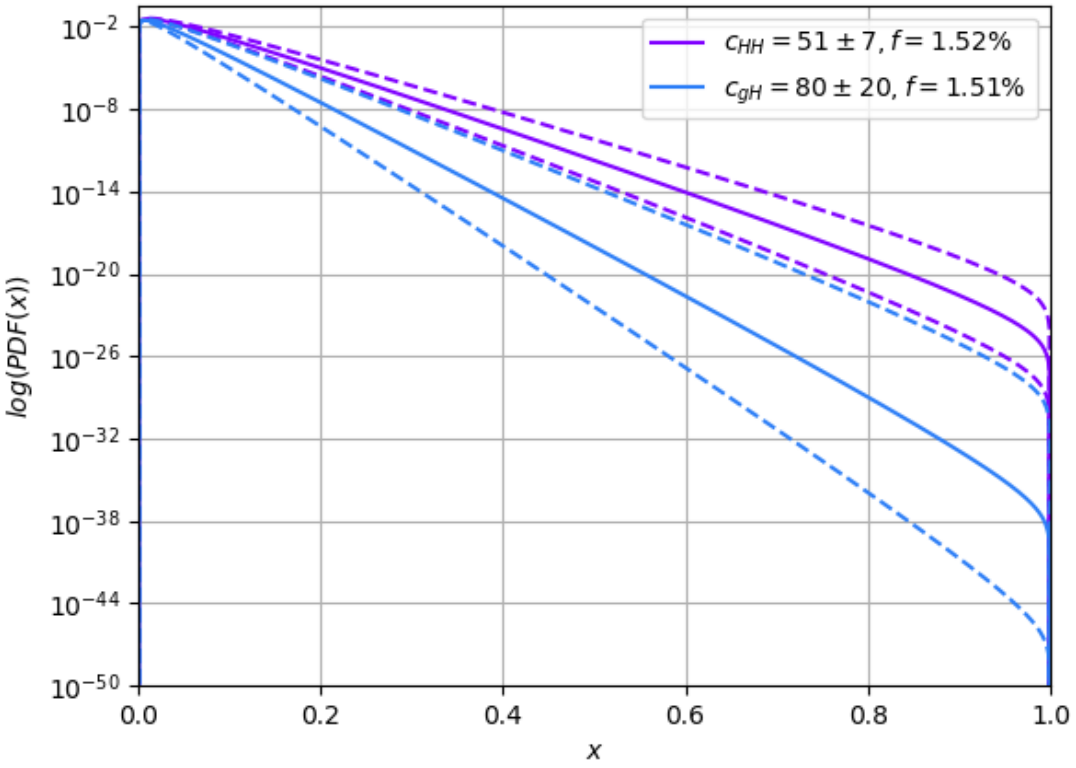


Figure 24: $PDF^H_{HH}(x)$ and $PDF^H_{gH}(x)$

5 Conclusion

5.1 Summary

This thesis had the purpose of looking at the structure of the Higgs PDF with regard to dynamics of differential cross sections and their dependence on momentum distribution. To analyze this behaviour, HERWIG++ was used in LO approximation at an energy scale of $Q^2 = 13TeV$ with $N = 100000$ simulations. The basis of this research was the PDF used by [4] of the form $PDF_R(x) = (x-1)^a x^b$ with an additional dampening term e^{-cx}. At first, the results done by Reiner were varified in range of errors. Then, by adding the extra term with c-values in range $[1, 1000]$, the differential cross sections $\frac{d\sigma}{pT_T}$ depending on pT_T of the SM proton sm, modified proton pp and processes involving the Higgs particle gH and HH were analyzed. The results were a decline in steepness of gH- and HH-processes for larger c, described by fitfunctions of logarithmic values $log(\frac{d\sigma}{dpT_T})$. The PDFs describing declines of gH- and HH-subprocesses faster than sm are described by

$$PDF(x) = (1-x)^{1.6} x^{0.6} e^{-(80\pm20)x} \tag{5.1}$$

with corresponding fraction of the Higgs $f = (1.51 \pm 0.20)\%$. The up-quark therefore seems to be the dominant part in the proton, while the peak of dampened Higgs PDF is shifted towards the left by a considerably amount of $x = 0.27 \rightarrow x = 0.01$.

5.2 A look into future research

As only the $pp \rightarrow t\bar{t}$ processes were taken into consideration, one of the next steps could be the combination with $pp \rightarrow t\bar{t}Z$ processes. Additionally, one could include other dynamic variables such as the pseudorapidity η and polar angle in the transverse plane ϕ, also contained in the output files of HERWIG++. When taking the planned circular collider FCC at CERN into consideration, one can hope for reductions in the current limitations of experimental data, allowing for greater insight in future papers.

A Ratio plots

The ratio plots used and discussed in section 4.1 are listed, giving insight on possible fraction for the Higgs particle depending on the PDF parameters used.

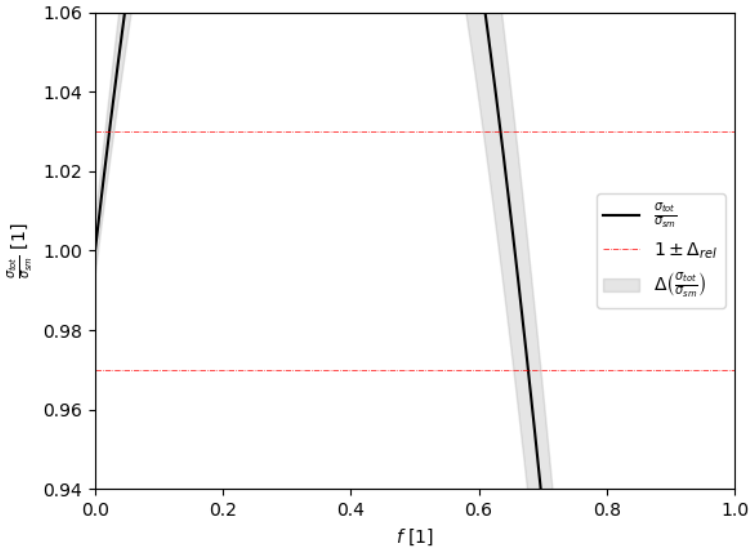


Figure 25: Allowed higgs fraction f for PDF_R^H

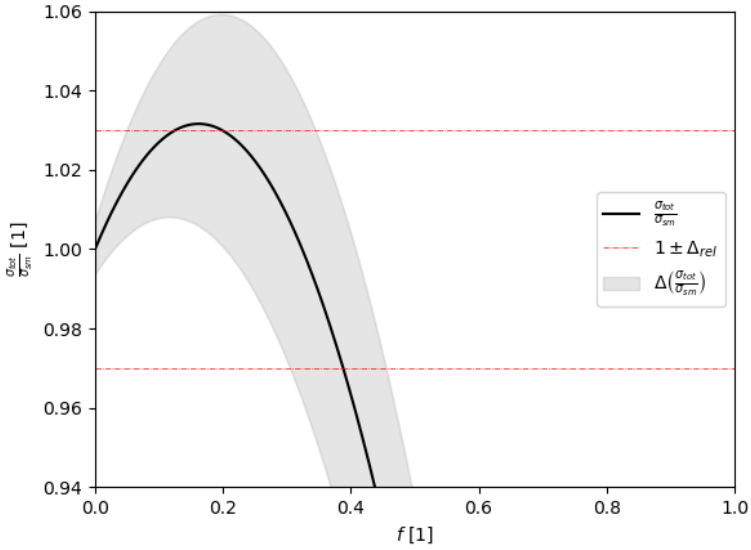


Figure 26: Allowed higgs fraction f for $c = 1$

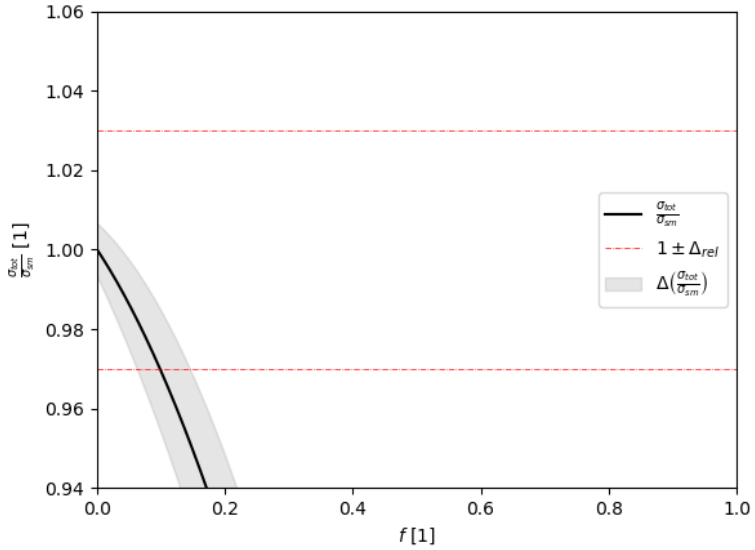


Figure 27: Allowed higgs fraction f for $c = 2$

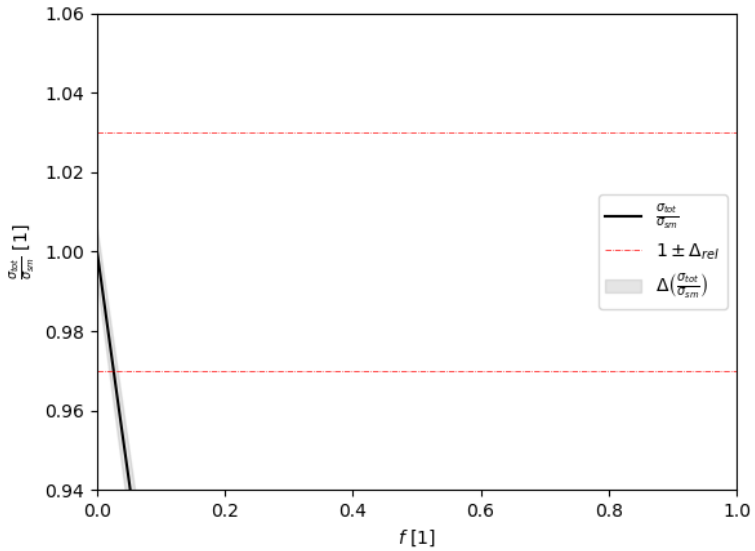


Figure 28: Allowed higgs fraction f for $c = 5$

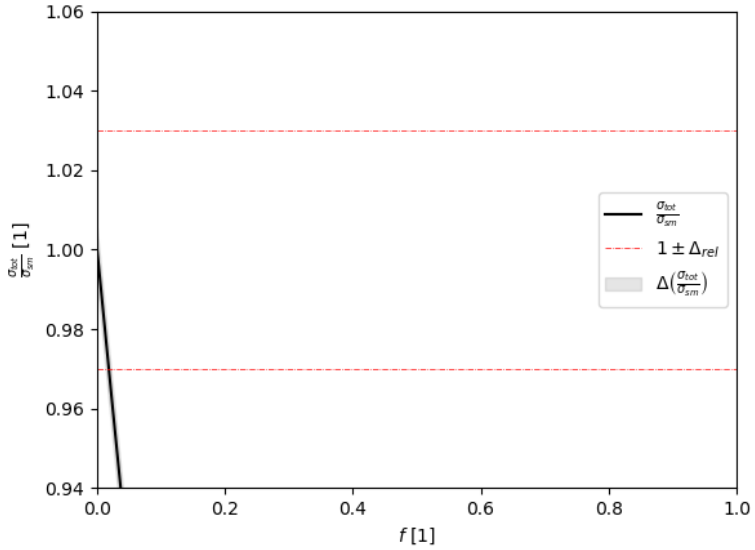


Figure 29: Allowed higgs fraction f for $c = 10$

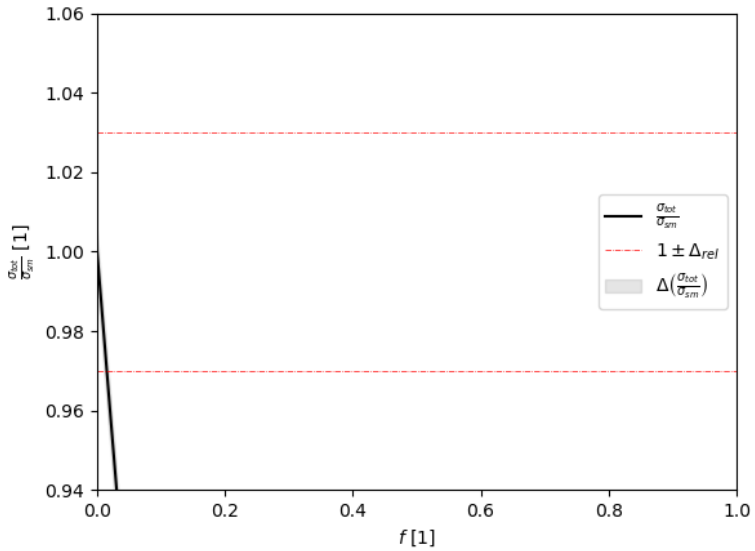


Figure 30: Allowed higgs fraction f for $c = 1000$

B Code

B.1 pdf_yoda.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: fabianv
"""

import os
import glob
import matplotlib.pyplot as plt
from matplotlib.pyplot import cm
import shutil
import subprocess
import datetime
import numpy as np
from scipy import stats
import fileinput
import master_yoda # own library master_yoda.py seen below
import enum
import scipy
from scipy.optimize import curve_fit

# Change directory of the bash file here,
# must be contained in Herwig subdirectory
prog_path = os.path.dirname(__file__)
bash_name = 'bash_TTbar.sh'

# checks if file is empty :
def is_file_empty(file_path):
return os.path.exists(file_path) and os.stat(file_path).st_size == 0

# Enum for size units
class SIZE_UNIT(enum.Enum):
BYTES = 1
KB = 2
MB = 3
GB = 4

# Convert size units
def convert_unit(size_in_bytes, unit):
   if unit == SIZE_UNIT.KB:
      return size_in_bytes/1024
   elif unit == SIZE_UNIT.MB:
      return size_in_bytes/(1024*1024)
   elif unit == SIZE_UNIT.GB:
      return size_in_bytes/(1024*1024*1024)
   else:
      return size_in_bytes
```

```
# Get file in size in given unit like KB, MB or GB
def get_file_size(file_name, size_type = SIZE_UNIT.BYTES ):
   size = os.path.getsize(file_name)
      return convert_unit(size, size_type)

# Find exponent of number
def findExponent(number):

   number = np.abs(number)
   index = 0

   if 1 <= number < 10:
      return index

   elif 0 <= number < 1:
      while True:
         index -= 1
         number *= 10
         if number >= 1:
            return index
   else:
      while True:
         index += 1
         number /= 10
         if number < 10:
            return index


# Solve quadratic equation, return solution(s) smaller than one,
# error message otherwise
def quadsol(a,b,c):
   sign_q1 = 1
   sign_q2 = -1
   if a is None or b is None or c is None:
      ValueError('Something went wrong with the total cross sections, look at
          data files for further information')
      return None, None

   if b**2 - 4*a*c < 0:
      ValueError('Complex results!')
         return None, None

   q1 = -b + np.sqrt(b**2-4*a*c)
   q2 = -b - np.sqrt(b**2-4*a*c)
   f1 = q1/(2*a)
   f2 = q2/(2*a)

if f1 is not None and f2 is not None:

   if (0 < f1 <= 1) and (0 < f2 <= 1):
      f = np.min((f1,f2))
      if f == f1:
         return f1, sign_q1
```

```
        else:
            return f2, sign_q2
    elif 0 < f1 <= 1:
        return f1, sign_q1
    else:
        return f2, sign_q2
elif f1 is not None and (0 < f1 <= 1):
    return f1, sign_q1
elif f2 is not None and (0 < f2 <= 1):
    return f2, sign_q2
else:
    ValueError('Both values return None or are not in the range of 0 and 1!')
    return None, None

############################################################
#       Start of main programm, main parameters     #
############################################################

# Set to true to start simulation/delete files,
# otherwise work with existing files
start = False
delete = False
seed_delete = []

# Decide on values for PDF, simulation and error for Herwig++
seed = [133]
number = [100000]
c = np.linspace(0,1,1000)
delta_rel = 0.03

# Parameters for PDF
#a_par = np.linspace(0.0,2.0,20)
#b_par = np.linspace(-2.0,2.0,40)
a_par = np.linspace(1.6,1.6,1)
b_par = np.linspace(0.6,0.6,1)
c_par = np.array([0,1,2,5,10,20,30,40,50,55,60,62,64,66,68,70,\
72,74,76,78,80,82,84,86,88,90,92,94,96,98,100,\
102,104,106,108,110,112,114,116,118,120,125,130,\
140,150,160,170,180,190,200,250,300,400,500,750,1000])

# Initialize linear fit parameter list and errors of dsig values
# Slope of fit curve
lin_k_sm = []
lin_k_sm_err1 = []
lin_k_sm_err2 = []

lin_k_GH = []
lin_k_GH_err1 = []
lin_k_GH_err2 = []

lin_k_HH = []
lin_k_HH_err1 = []
lin_k_HH_err2 = []
```

```
# Intersection of fit curve
lin_d_sm = []
lin_d_sm_err1 = []
lin_d_sm_err2 = []

lin_d_GH = []
lin_d_GH_err1 = []
lin_d_GH_err2 = []

lin_d_HH = []
lin_d_HH_err1 = []
lin_d_HH_err2 = []

# Delete unnecessary files by seed name
if delete == True:
# Deletes files with name format {file_begin}-S{seed}.{file_end}
   master_yoda.deleteYoda(seed_delete)

else:
   print('Nothing deleted.')

if master_yoda.deleteYoda.has_been_called:

   message = 'Files with seed(s) {} have been deleted!'.format(seed_delete)
   print('''
{}\n################################################################
'''.format(message))

#############################################################
#   generate PDF with np.root(), np.exp(), np.power() #
#############################################################

# Parameters for PDF, maximum of three
if c_par is None:
   print('Values c1 and c2 being tested:\nc1 = {0}\nc2 =
       {1}'.format(a_par,b_par))
else:
   print('Values c1, c2 and c3 being tested:\nc1 = {0}\nc2 = {1}\nc3 =
       {2}'.format(a_par,b_par,c_par))

# Set True if completely new PDF was used
renew = True

# PDF form in latex format for title
for ind1 in range(len(a_par)):
   for ind2 in range(len(b_par)):
      for ind3 in range(len(c_par)):
      # Change PDF form in PDF_latex and PDF_python here
      PDF_latex = r'(1-x)^{{c_{{1}}}}x^{{c_{{2}}}}e^{{-c_{{3}}x}}, c_{{1}} = {0},
          c_{{2}} = {1}, c_{{3}} =
          {2}'.format(a_par[ind1],b_par[ind2],c_par[ind3])
      PDF_fig = 'PDF_' + PDF_latex.split(',')[0]
```

```
      # PDF form to change PDF in HiggsPDF.cc
      PDF_python = 'np.power(1.0-x,{0})*np.power(x,{1})*exp(-{2}*x)'\
      .format(a_par[ind1],b_par[ind2],c_par[ind3])
      [PDF_old, PDF_new] = master_yoda.changeYoda(PDF_python)

      # Further calculatins for all different seeds s and numbers N
      for i in range(len(seed)):
         for j in range(len(number)):

# Start of for loops, shifted in for readability
# Change values in bash file for simulation and call Herwig++ for MC simulations
if start == True:
   master_yoda.bashYoda(bash_name, num_herwig=number[j], seed_herwig=seed[i])
   subprocess.call(['./{}'.format(bash_name)])
else:
   print('No new PDF, just data analysis')

if master_yoda.bashYoda.has_been_called:
   print('''
\n############################################################\n
Simulation started!\nOld PDF used: {0}\nNew PDF used: {1}
############################################################\n
   '''.format(PDF_old[9:],PDF_new[9:]))

############################################################
#        general information from yoda files        #
############################################################

file_size = SIZE_UNIT.KB

# Rename files from simulation
if master_yoda.bashYoda.has_been_called:

   # Get new yoda- and outfiles, sorted alphabetically
   yoda_files = sorted(glob.glob('*S{}.yoda'.format(seed[i])))
   out_files = sorted(glob.glob('*S{}.out'.format(seed[i])))

   # Rename yoda and out files for each individual choice of parameters
   if c_par is None:
      for file in yoda_files:
         os.rename(file,file[:-5] +
             '_c1_{0}_c2_{1}_N_{2}.yoda'.format(a_par[ind1],b_par[ind2],number[j]))
      for file in out_files:
         os.rename(file,file[:-4] +
             '_c1_{0}_c2_{1}_N_{2}.out'.format(a_par[ind1],b_par[ind2],number[j]))

      yoda_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_N_{3}.yoda'\
      .format(seed[i],a_par[ind1],b_par[ind2],number[j])))
      out_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_N_{3}.out'\
      .format(seed[i],a_par[ind1],b_par[ind2],number[j])))

   else:
```

```
      for file in yoda_files:
         os.rename(file,file[:-5] + '_c1_{0}_c2_{1}_c3_{2}_N_{3}.yoda'\
         .format(a_par[ind1],b_par[ind2],c_par[ind3],number[j]))
      for file in out_files:
         os.rename(file,file[:-4] + '_c1_{0}_c2_{1}_c3_{2}_N_{3}.out'\
         .format(a_par[ind1],b_par[ind2],c_par[ind3],number[j]))

      yoda_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_c3_{3}_N_{4}.yoda'\
      .format(seed[i],a_par[ind1],b_par[ind2],c_par[ind3],number[j])))
      out_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_c3_{3}_N_{4}.out'\
      .format(seed[i],a_par[ind1],b_par[ind2],c_par[ind3],number[j])))

# Work through already generated data
else:
   if c_par is None:

      yoda_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_N_{3}.yoda'\
      .format(seed[i],a_par[ind1],b_par[ind2],number[j])))
      out_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_N_{3}.out'\
      .format(seed[i],a_par[ind1],b_par[ind2],number[j])))

   else:

      yoda_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_c3_{3}_N_{4}.yoda'\
      .format(seed[i],a_par[ind1],b_par[ind2],c_par[ind3],number[j])))
      out_files = sorted(glob.glob('*S{0}_c1_{1}_c2_{2}_c3_{3}_N_{4}.out'\
      .format(seed[i],a_par[ind1],b_par[ind2],c_par[ind3],number[j])))

yoda_sz_date = [(file, os.path.getsize(file), os.path.getmtime(file)) for file
    in yoda_files]
for file, size, mtime in yoda_sz_date:
   size = get_file_size(prog_path, file_size)
   date_str = datetime.datetime.fromtimestamp(mtime).strftime('%Y-%m-%d-%H:%M')
   print('{0}: size = {1} KB, date of last change =
       {2}'.format(file,size,date_str))

print(yoda_files,out_files)

############################################################
#                  read output files                       #
############################################################

# Decide if unit should be b or nb, likewise for b/sr or nb/sr
unit_sig = 'b'
unit_dsig = 'b/sr'

if unit_sig == 'b':
   conv = True
else:
   conv = False

# Total cross section in b (Barn), add convert = False to function to get total
    cross section in nb
```

```
sm_out, sm_out_err = master_yoda.readOut(out_files[3], convert=conv)
PP_out, PP_out_err = master_yoda.readOut(out_files[2], convert=conv)
GH_out, GH_out_err = master_yoda.readOut(out_files[0], convert=conv)
HH_out, HH_out_err = master_yoda.readOut(out_files[1], convert=conv)

sigma_out = [sm_out, PP_out, GH_out, HH_out]
sigma_err = [sm_out_err, PP_out_err, GH_out_err, HH_out_err]

# Differential cross section
data_sm, dicdata_sm, scale_sm = master_yoda.readYoda(yoda_files[3])
data_PP, dicdata_PP, scale_PP = master_yoda.readYoda(yoda_files[2])
data_GH, dicdata_GH, scale_GH = master_yoda.readYoda(yoda_files[0])
data_HH, dicdata_HH, scale_HH = master_yoda.readYoda(yoda_files[1])

############################################################
#         Calculate sigma and Higgs fraction        #
############################################################

# calculate total cross section and fraction of higgs
sigma_a = 1 - (HH_out-GH_out)/PP_out
sigma_b = GH_out/PP_out - 2
sigma_c1 = -delta_rel
sigma_c2 = +delta_rel

#Total cross sections derived from various c parameters and their errors
sigma_tot_list = []
sigma_tot_err_list= []

for k in range(len(c)):

   sigma_tot, sigma_tot_err = master_yoda.sigmaYodac(sigma_out, sigma_err, c[k])
   sigma_tot_list.append(sigma_tot)
   sigma_tot_err_list.append(sigma_tot_err)

# Value of higgs fraction and error
higgs_frac1, sign1 = quadsol(sigma_a, sigma_b, sigma_c1)
higgs_frac2, sign2 = quadsol(sigma_a, sigma_b, sigma_c2)

# Check which calculation results in the correct value for the fraction of the
    higgs
if higgs_frac1 is not None and higgs_frac2 is not None:

   if higgs_frac1 < 0 and 0 < higgs_frac2 <= 1:
      higgs_frac2 = np.round(higgs_frac2,4)
      higgs_frac = higgs_frac2
      higgs_frac_err = master_yoda.higgsError(higgs_frac2, sigma_out, sigma_err,
          sign2, sign2*delta_rel)
      higgs_frac_err = np.round(higgs_frac_err,4)

   elif higgs_frac2 < 0 and 0 < higgs_frac1 <= 1:
      higgs_frac1 = np.round(higgs_frac1,4)
      higgs_frac = higgs_frac1
```

```
            higgs_frac_err = master_yoda.higgsError(higgs_frac1, sigma_out, sigma_err,
                sign1, sign1*delta_rel)
            higgs_frac_err = np.round(higgs_frac_err,4)

        else:
            higgs_frac = 0
            higgs_frac_err = 0

    elif higgs_frac1 is not None:

        if 0 < higgs_frac1 <= 1:
            higgs_frac1 = np.round(higgs_frac1,4)
            higgs_frac = higgs_frac1
            higgs_frac_err = master_yoda.higgsError(higgs_frac1, sigma_out, sigma_err,
                sign1, sign1*delta_rel)
            higgs_frac_err = np.round(higgs_frac_err,4)
        else:
            higgs_frac = 0
            higgs_frac_err = 0

    elif higgs_frac2 is not None:

        if 0 < higgs_frac2 <= 1:
            higgs_frac2 = np.round(higgs_frac2,4)
            higgs_frac = higgs_frac2
            higgs_frac_err = master_yoda.higgsError(higgs_frac2, sigma_out, sigma_err,
                sign2, sign2*delta_rel)
            higgs_frac_err = np.round(higgs_frac_err,4)
        else:
            higgs_frac = 0
            higgs_frac_err = 0

    else:
        higgs_frac = 0
        higgs_frac_err = 0

    ############################################################
    #    Total cross sections derived from higgs fraction      #
    ############################################################

    # Total cross sections derived from various c parameters and higgs fraction f
    sigma_c_list = []
    sigma_c_err_list= []

    for k in range(len(c)):
        #sigma_cf, sigma_cf_err = master_yoda.sigmaYoda2(sigma_out, sigma_err, c[k],
            higgs_frac, higgs_frac_err) # Depending on c and f
        sigma_c, sigma_c_err = master_yoda.sigmaYodac(sigma_out, sigma_err, c[k]) #
            Depending on c
        sigma_c_list.append(sigma_c)
        sigma_c_err_list.append(sigma_c_err)
```

```
# sigma_f, sigma_f_err = master_yoda.sigmaYoda(sigma_out, sigma_err, higgs_frac)
    # Not depending on higgs_frac_err
sigma_f, sigma_f_err = master_yoda.sigmaYodaf(sigma_out, sigma_err, higgs_frac,
    higgs_frac_err)
sigma_tot_exp = sigma_f/sm_out

############################################################
#           Graph for acceptable values for c       #
############################################################

plt.close()

fig1 = plt.figure()

plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))

# Convert list of total cross section and errors into array for plotting
sigma_tot_array = np.array(sigma_tot_list)
sigma_tot_err_array = np.array(sigma_tot_err_list)

# Modified P'P' cross section depending on f and c
sigma_c_array = np.array(sigma_c_list)
sigma_c_err_array = np.array(sigma_c_err_list)

# Error for sig_tot/sig_sm
sigma_c_sm = sigma_c_array/sm_out
sigma_c_sm_err = np.abs(1/sm_out)*sigma_c_err_array +
    np.abs(-1*sigma_c_array/(sm_out**2))*sm_out_err

# Print readable version of numbers in terminal
sigma_c_pow = findExponent(sigma_c_list[0])
sigma_f_pow = findExponent(sigma_f)

sigma_c_0 = np.round(sigma_c_list[0]*10**(-sigma_c_pow),2)
sigma_c_err0 = np.round(sigma_c_err_list[0]*10**(-sigma_c_pow),2)

sigma_c_end = np.round(sigma_c_list[-1]*10**(-sigma_c_pow),2)
sigma_c_errend = np.round(sigma_c_err_list[-1]*10**(-sigma_c_pow),2)

sigma_f_0 = np.round(sigma_f*10**(-sigma_f_pow),2)
sigma_f_err0 = np.round(sigma_f_err*10**(-sigma_f_pow),2)

print('''
############################################################\n
Total cross section for {0} c values from {1} to {2}:\n
Total cross section for c:\n({3} +/- {9})*10e{7} {11} for c = {1}\n({4} +/-
    {10})*10e{7} {11} for c = {2}\n
Total cross section for f:\n({5} +/- {6})*10e{8} {11}\n
############################################################
'''.format(np.size(c), c[0], c[-1], sigma_c_0, sigma_c_end, sigma_f_0,
    sigma_f_err0, sigma_c_pow, sigma_f_pow, sigma_c_err0, sigma_c_errend,
    unit_sig))
```

```
# Find the best value for c parameter and check if in range of possible values
c_close = min(sigma_c_list, key=lambda x:abs(x-sigma_f))
index = 0
for loop in range(len(sigma_c_list)):
   index += 1
   if c_close == sigma_c_list[loop]:
      c_value = c[index]
      c_value = np.round(c_value,3)
      print('Best value for c parameter is: {}'.format(c_value))

# Plot derivation of total cross sections with c value
plt.plot(c,sigma_c_sm, color = 'k', label =
    r'$\frac{\sigma_{tot}}{\sigma_{sm}}$')
dotted = plt.axhline(y=1-delta_rel, color = 'r', linestyle = '-.', linewidth =
    0.5, label = r'$1 \pm \Delta_{rel}$')
plt.axhline(y=1+delta_rel, color = 'r', linestyle = '-.', linewidth = 0.5)
plt.fill_between(c,sigma_c_sm + sigma_c_sm_err,sigma_c_sm - sigma_c_sm_err,
    color = 'gray', alpha =0.2, label =
    r'$\Delta\left(\frac{\sigma_{tot}}{\sigma_{sm}}\right)$')
plt.axis (xmin = 0, xmax = np.max(c),ymin = 1-2*delta_rel, ymax = 1+2*delta_rel)
plt.xlabel(r'$c$ $[1]$')
plt.ylabel(r'$\frac{\sigma_{tot}}{\sigma_{sm}}$ $[1]$')
plt.legend()

#############################################################
#     x-values and y-values from the dicdata files  #
#############################################################

# sm file
[x_sm, x_sm_err, y_sm, y_sm_err, y_sm_err_rel, num_Entries_sm] =
    master_yoda.valueYoda(dicdata_sm, scale=scale_sm)

# PP file
[x_PP, x_PP_err, y_PP, y_PP_err, y_PP_err_rel, num_Entries_PP] =
    master_yoda.valueYoda(dicdata_PP, scale=scale_PP)

# GH file
[x_GH, x_GH_err, y_GH, y_GH_err, y_GH_err_rel, num_Entries_GH] =
    master_yoda.valueYoda(dicdata_GH, scale=scale_GH)

# HH file
[x_HH, x_HH_err, y_HH, y_HH_err, y_HH_err_rel, num_Entries_HH] =
    master_yoda.valueYoda(dicdata_HH, scale=scale_HH)

y_sm_rel = (y_sm + y_sm_err)/y_sm
y_PP_rel = (y_PP + y_PP_err)/y_PP
y_GH_rel = (y_GH + y_GH_err)/y_GH
y_HH_rel = (y_HH + y_HH_err)/y_HH

#############################################################
#               Graph of dsig and p_T                #
#############################################################
```

```
# Different subplots for each subprocess

# Define shape of subplot
nrow = 2
ncol = 2
wid = 2*x_sm_err
cap = 5

fig2, axis1 = plt.subplots(nrow, ncol, figsize=(14, 8))

axis1[0,0].bar(x_sm, y_sm, yerr=y_sm_err, width=wid, color='r', capsize=cap)
axis1[0,1].bar(x_PP, y_PP, yerr=y_PP_err, width=wid, color='g', capsize=cap)
axis1[1,0].bar(x_GH, y_GH, yerr=y_GH_err, width=wid, color='b', capsize=cap)
axis1[1,1].bar(x_HH, y_HH, yerr=y_HH_err, width=wid, color='c', capsize=cap)

run_str = [['sm','PP'],['GH','HH']]

for k in range(nrow):
   for l in range(ncol):
      axis1[k,l].set_title(r'$d\sigma$ for {}-subprocess'.format(run_str[k][l]))
      axis1[k,l].set_xlabel(r'$pT_{T}$ $[GeV]$')
      axis1[k,l].set_ylabel(r'$\frac{1}{\sigma}\frac{d\sigma}{dpT_{T}}$ $[1]$')
      axis1[k,l].grid()
      axis1[k,l].set_ylim(0,np.max([y_sm,y_PP,y_GH,y_HH])*1.1)

plt.tight_layout()


# All subprocsesses in one figure, not used in main routine

fig3, (axis1, axis2) = plt.subplots(2, 1, sharex=True,
    gridspec_kw={'height_ratios': [2, 1]}, figsize=(14, 8))

                axis1.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
axis1.errorbar(x_sm, y_sm, xerr=x_sm_err, yerr=y_sm_err, color='r', capsize=cap,
    label= 'sm', fmt='v')
axis1.errorbar(x_PP, y_PP, xerr=x_sm_err, yerr=y_PP_err, color='g', capsize=cap,
    label= 'PP', fmt='^')
axis1.errorbar(x_GH, y_GH, xerr=x_sm_err, yerr=y_GH_err, color='b', capsize=cap,
    label= 'gH', fmt='*')
axis1.errorbar(x_HH, y_HH, xerr=x_sm_err, yerr=y_HH_err, color='c', capsize=cap,
    label= 'HH', fmt='s')

axis1.set_yscale('log', nonposy='clip')

axis1.set_xlabel(r'$pT_{T}$ $[GeV]$')
axis1.set_ylabel(r'$\frac{1}{\sigma}\frac{d\sigma}{dpT_{T}}$ $[1]$')
axis1.set_xlim(0,x_sm[-1]+x_sm_err[-1])
axis1.set_ylim(0,1)
axis1.set_xticks(x_sm-25,minor=True)
axis1.legend(loc="upper right")
axis1.grid()
plt.tight_layout()
```

```
# Ratio of dsig + err and dsig
axis2.plot(x_sm,y_sm_rel,'-r')
axis2.plot(x_PP,y_PP_rel,'-g')
axis2.plot(x_GH,y_GH_rel,'-b')
axis2.plot(x_HH,y_HH_rel,'-c')

axis2.plot([0,x_sm[-1]+50],[1,1],'--k')
axis2.set_ylabel('Ratio')
axis2.set_xlabel(r'$pT_{T}$ $[GeV]$')
axis2.set_xlim(0,x_sm[-1]+x_sm_err[-1])
axis2.set_ylim(0.4,1.6)
axis2.grid()

fig3.tight_layout(rect=[0, 0.03, 1, 0.95])

#############################################################
#      Use data from max onwards for linfit         #
#############################################################

# From max onwards, plot HH, GH and sm with a polynomial of the form a*p_T+b and
    then plot a in comparison to c

# Allowed relative error
err_rel = 1.50

# Index of peak value
y_sm_start = y_sm.tolist().index(np.max(y_sm))
y_GH_start = y_GH.tolist().index(np.max(y_GH))
y_HH_start = y_HH.tolist().index(np.max(y_HH))

# Index of final value
y_sm_fin = [ind11 for ind11,value in enumerate(y_sm_rel) if value > err_rel]
y_GH_fin = [ind11 for ind11,value in enumerate(y_GH_rel) if value > err_rel]
y_HH_fin = [ind11 for ind11,value in enumerate(y_HH_rel) if value > err_rel]

if not y_sm_fin:
   y_sm_fin = len(y_sm)-1
else:
   y_sm_fin = y_sm_fin[0]

if not y_GH_fin:
   y_GH_fin = len(y_GH)-1
else:
   y_GH_fin = y_GH_fin[0]

if not y_HH_fin:
   y_HH_fin = len(y_HH)-1
else:
   y_HH_fin = y_HH_fin[0]

x_sm_fit = x_sm
x_GH_fit = x_GH
```

```
x_HH_fit = x_HH

x_sm_fit_err = x_sm_err
x_GH_fit_err = x_GH_err
x_HH_fit_err = x_HH_err

y_sm_fit = y_sm
y_GH_fit = y_GH
y_HH_fit = y_HH

y_sm_fit_err = y_sm_err
y_GH_fit_err = y_GH_err
y_HH_fit_err = y_HH_err

# Only include values from peak to rel. uncertainty smaller err_rel
x_sm_fit = x_sm_fit[y_sm_start:y_sm_fin]
x_GH_fit = x_GH_fit[y_GH_start:y_GH_fin]
x_HH_fit = x_HH_fit[y_HH_start:y_HH_fin]

x_sm_fit_err = x_sm_fit_err[y_sm_start:y_sm_fin]
x_GH_fit_err = x_GH_fit_err[y_GH_start:y_GH_fin]
x_HH_fit_err = x_HH_fit_err[y_HH_start:y_HH_fin]

y_sm_fit = y_sm_fit[y_sm_start:y_sm_fin]
y_GH_fit = y_GH_fit[y_GH_start:y_GH_fin]
y_HH_fit = y_HH_fit[y_HH_start:y_HH_fin]

y_sm_fit_err = y_sm_fit_err[y_sm_start:y_sm_fin]
y_GH_fit_err = y_GH_fit_err[y_GH_start:y_GH_fin]
y_HH_fit_err = y_HH_fit_err[y_HH_start:y_HH_fin]

# Linear fit comparison plot, upper and lower errors from statistical errors

# Delete zero value as they cannot be used for log comparison
GH_zero = len(y_GH_fit[y_GH_fit == 0])
HH_zero = len(y_HH_fit[y_HH_fit == 0])

#print(GH_zero,HH_zero)
if GH_zero > 0:
   x_GH_fit = x_GH_fit[:-GH_zero]
   x_GH_fit_err = x_GH_fit_err[:-GH_zero]
   y_GH_fit = y_GH_fit[:-GH_zero]
   y_GH_fit_err = y_GH_fit_err[:-GH_zero]

if HH_zero > 0:
   x_HH_fit = x_HH_fit[:-HH_zero]
   x_HH_fit_err = x_HH_fit_err[:-HH_zero]
   y_HH_fit = y_HH_fit[:-HH_zero]
   y_HH_fit_err = y_HH_fit_err[:-HH_zero]

#print('fitted data: GH\n {0}\nHH\n {1}'.format(y_GH_fit,y_HH_fit))
y_sm_fit_log = np.log10(y_sm_fit)
y_GH_fit_log = np.log10(y_GH_fit)
```

```
y_HH_fit_log = np.log10(y_HH_fit)

y_sm_fit_logu = np.log10(y_sm_fit+y_sm_fit_err)
y_GH_fit_logu = np.log10(y_GH_fit+y_GH_fit_err)
y_HH_fit_logu = np.log10(y_HH_fit+y_HH_fit_err)

y_sm_fit_logd = np.log10(y_sm_fit-y_sm_fit_err)
y_GH_fit_logd = np.log10(y_GH_fit-y_GH_fit_err)
y_HH_fit_logd = np.log10(y_HH_fit-y_HH_fit_err)

k_sm, d_sm, r_sm, p_sm, std_err_sm = stats.linregress(x_sm_fit, y_sm_fit_log)
k_GH, d_GH, r_GH, p_GH, std_err_GH = stats.linregress(x_GH_fit, y_GH_fit_log)
k_HH, d_HH, r_HH, p_HH, std_err_HH = stats.linregress(x_HH_fit, y_HH_fit_log)

k_sm_u, d_sm_u, r_sm_u, p_sm_u, std_err_sm_u = stats.linregress(x_sm_fit,
    y_sm_fit_logu)
k_GH_u, d_GH_u, r_GH_u, p_GH_u, std_err_GH_u = stats.linregress(x_GH_fit,
    y_GH_fit_logu)
k_HH_u, d_HH_u, r_HH_u, p_HH_u, std_err_HH_u = stats.linregress(x_HH_fit,
    y_HH_fit_logu)

k_sm_d, d_sm_d, r_sm_d, p_sm_d, std_err_sm_d = stats.linregress(x_sm_fit,
    y_sm_fit_logd)
k_GH_d, d_GH_d, r_GH_d, p_GH_d, std_err_GH_d = stats.linregress(x_GH_fit,
    y_GH_fit_logd)
k_HH_d, d_HH_d, r_HH_d, p_HH_d, std_err_HH_d = stats.linregress(x_HH_fit,
    y_HH_fit_logd)

# Errors for fitplot from statistical errors
x_sm_fitline = np.linspace(x_sm_fit[0],x_sm_fit[-1])
x_GH_fitline = np.linspace(x_GH_fit[0],x_GH_fit[-1])
x_HH_fitline = np.linspace(x_HH_fit[0],x_HH_fit[-1])

y_sm_fitline = 10**(k_sm*x_sm_fitline + d_sm)
y_GH_fitline = 10**(k_GH*x_GH_fitline + d_GH)
y_HH_fitline = 10**(k_HH*x_HH_fitline + d_HH)

y_sm_fitdotline_d = 10**(k_sm_d*x_sm_fitline + d_sm_d)
y_GH_fitdotline_d = 10**(k_GH_d*x_GH_fitline + d_GH_d)
y_HH_fitdotline_d = 10**(k_HH_d*x_HH_fitline + d_HH_d)

y_sm_fitdotline_u = 10**(k_sm_u*x_sm_fitline + d_sm_u)
y_GH_fitdotline_u = 10**(k_GH_u*x_GH_fitline + d_GH_u)
y_HH_fitdotline_u = 10**(k_HH_u*x_HH_fitline + d_HH_u)

min_lin_m = min(min(y_sm_fitline),min(y_GH_fitline),min(y_HH_fitline))
min_lin_d =
    min(min(y_sm_fitdotline_d),min(y_GH_fitdotline_d),min(y_HH_fitdotline_d))
min_lin_u =
    min(min(y_sm_fitdotline_u),min(y_GH_fitdotline_u),min(y_HH_fitdotline_u))

max_lin_m = max(max(y_sm_fitline),max(y_GH_fitline),max(y_HH_fitline))
```

```
max_lin_d =
    max(max(y_sm_fitdotline_d),max(y_GH_fitdotline_d),max(y_HH_fitdotline_d))
max_lin_u =
    max(max(y_sm_fitdotline_u),max(y_GH_fitdotline_u),max(y_HH_fitdotline_u))

# Add parameters to list for plot later
lin_k_sm.append(k_sm)
lin_d_sm.append(d_sm)
lin_k_GH.append(k_GH)
lin_d_GH.append(d_GH)
lin_k_HH.append(k_HH)
lin_d_HH.append(d_HH)

lin_k_sm_err1.append(np.abs(k_sm-k_sm_d))
lin_d_sm_err1.append(np.abs(d_sm-d_sm_d))
lin_k_GH_err1.append(np.abs(k_GH-k_GH_d))
lin_d_GH_err1.append(np.abs(d_GH-d_GH_d))
lin_k_HH_err1.append(np.abs(k_HH-k_HH_d))
lin_d_HH_err1.append(np.abs(d_HH-d_HH_d))

lin_k_sm_err2.append(np.abs(k_sm-k_sm_u))
lin_d_sm_err2.append(np.abs(d_sm-d_sm_u))
lin_k_GH_err2.append(np.abs(k_GH-k_GH_u))
lin_d_GH_err2.append(np.abs(d_GH-d_GH_u))
lin_k_HH_err2.append(np.abs(k_HH-k_HH_u))
lin_d_HH_err2.append(np.abs(d_HH-d_HH_u))

# Linear regress comparison plot
fig4 = plt.figure()
ax = plt.axes()

plt.errorbar(x_sm_fit, y_sm_fit, xerr=x_sm_fit_err, yerr=y_sm_fit_err,
    color='r', capsize=cap,
label= 'sm', fmt='v')
plt.errorbar(x_GH_fit, y_GH_fit, xerr=x_GH_fit_err, yerr=y_GH_fit_err,
    color='b', capsize=cap,
label= 'gH', fmt='*')
plt.errorbar(x_HH_fit, y_HH_fit, xerr=x_HH_fit_err, yerr=y_HH_fit_err,
    color='c', capsize=cap,
label= 'HH', fmt='^')

ax.plot(x_sm_fitline, y_sm_fitline, '-k', label='fit curves',linewidth=0.75)
ax.plot(x_sm_fitline, y_sm_fitdotline_d, '--k',linewidth=0.75)
ax.plot(x_sm_fitline, y_sm_fitdotline_u, '--k',linewidth=0.75)

ax.plot(x_GH_fitline, y_GH_fitline, '-k',linewidth=0.75)
ax.plot(x_GH_fitline, y_GH_fitdotline_d, '--k',linewidth=0.75)
ax.plot(x_GH_fitline, y_GH_fitdotline_u, '--k',linewidth=0.75)

ax.plot(x_HH_fitline, y_HH_fitline, '-k',linewidth=0.75)
ax.plot(x_HH_fitline, y_HH_fitdotline_d, '--k',linewidth=0.75)
ax.plot(x_HH_fitline, y_HH_fitdotline_u, '--k',linewidth=0.75)
```

```
ax.set_yscale('log', nonposy='clip') # Logarithmic scale

plt.grid()
plt.legend()
plt.xlim((0,1000))
plt.ylim((0.5*min(min_lin_d,min_lin_m,min_lin_u),\
2*max(max_lin_d,max_lin_m,max_lin_u)))
plt.xlabel(r'$pT_{T}$ $[GeV]$')
plt.ylabel(r'$\frac{1}{\sigma}\frac{d\sigma}{dpT_{T}}$ $[1]$')

# Data and linear regress plot
fig5, (axis1, axis2) = plt.subplots(2, 1, sharex=True,
    gridspec_kw={'height_ratios': [2, 1]}, figsize=(14, 8))

axis1.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
axis1.errorbar(x_sm, y_sm, xerr=x_sm_err, yerr=y_sm_err, color='r', capsize=cap,
label= 'sm', fmt='v')
axis1.errorbar(x_PP, y_PP, xerr=x_sm_err, yerr=y_PP_err, color='g', capsize=cap,
label= 'PP', fmt='^')
axis1.errorbar(x_GH, y_GH, xerr=x_sm_err, yerr=y_GH_err, color='b', capsize=cap,
label= 'gH', fmt='*')
axis1.errorbar(x_HH, y_HH, xerr=x_sm_err, yerr=y_HH_err, color='c', capsize=cap,
label= 'HH', fmt='s')

axis1.plot(x_sm_fitline, y_sm_fitline, '-k', label='fit curves',linewidth=0.75)
axis1.plot(x_sm_fitline, y_sm_fitdotline_d, '--k',linewidth=0.75)
axis1.plot(x_sm_fitline, y_sm_fitdotline_u, '--k',linewidth=0.75)

axis1.plot(x_GH_fitline, y_GH_fitline, '-k',linewidth=0.75)
axis1.plot(x_GH_fitline, y_GH_fitdotline_d, '--k',linewidth=0.75)
axis1.plot(x_GH_fitline, y_GH_fitdotline_u, '--k',linewidth=0.75)

axis1.plot(x_HH_fitline, y_HH_fitline, '-k',linewidth=0.75)
axis1.plot(x_HH_fitline, y_HH_fitdotline_d, '--k',linewidth=0.75)
axis1.plot(x_HH_fitline, y_HH_fitdotline_u, '--k',linewidth=0.75)

axis1.set_yscale('log', nonposy='clip')
axis1.set_xlabel(r'$pT_{T}$ $[GeV]$')
axis1.set_ylabel(r'$\frac{1}{\sigma}\frac{d\sigma}{dpT_{T}}$ $[1]$')
axis1.set_xlim(0,x_sm[-1]+x_sm_err[-1])
axis1.set_ylim(0,1)
axis1.set_xticks(x_sm-25,minor=True)

axis1.legend(loc="upper right")
axis1.grid()
plt.tight_layout()

# Ratio of dsig + err and dsig
axis2.plot(x_sm,y_sm_rel,'-r')
axis2.plot(x_PP,y_PP_rel,'-g')
axis2.plot(x_GH,y_GH_rel,'-b')
axis2.plot(x_HH,y_HH_rel,'-c')
```

```
axis2.plot([0,x_sm[-1]+50],[1,1],'--k')
axis2.set_ylabel('Ratio')
axis2.set_xlabel(r'$pT_{T}$ $[GeV]$')
axis2.set_xlim(0,x_sm[-1]+x_sm_err[-1])
axis2.set_ylim(0.4,1.6)
axis2.grid()

fig5.tight_layout(rect=[0, 0.03, 1, 0.95])


############################################################
#              Save figures and data                       #
############################################################

# Decide on name for directory of new figures

# Make new directory folder for PDF with completele new shape
if renew == True:
   fig_dir = PDF_fig
   fig_path = os.path.join(prog_path,fig_dir)

results_dir = os.path.join(prog_path, fig_path)

# Check if directory exists, otherwise create it
if not os.path.isdir(results_dir):
   os.makedirs(results_dir)

if c_par is None:

   name1 = '{0}/c_total_N{1}_S{2}_c1_{3}_c2_{4}.png'\
   .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2])
   name2 = '{0}/PDF_subplots_N{1}_S{2}_c1_{3}_c2_{4}.png'\
   .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2])
   name3 = '{0}/PDF_fullplot_N{1}_S{2}_c1_{3}_c2_{4}.png'\
   .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2])
   name4 = '{0}/PDF_fitlinear_N{1}_S{2}_c1_{3}_c2_{4}.png'\
   .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2])
   name5 = '{0}/PDF_fullplot_and_lineplot_N{1}_S{2}_c1_{3}_c2_{4}.png'\
   .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2])

   with open('{0}/PDF_data_N{1}_S{2}_c1_{3}_c2_{4}.txt'.format(results_dir,
       number[j], seed[i], a_par[ind1], b_par[ind2]),'w') as data_file:
      text = '''
      PDF used           : {0}
      Number of simulations: {1}
      Seed used          : {2}\n
      Total cross section for {3} c values from {4} to {5}:\n
      Total cross section for c: ({6} +/- {7})*10e{8} {9} for c = {4}
      ({10} +/- {11})*10e{8} {9} for c = {5}
      Total cross section for f:\n({12} +/- {13})*10e{8} {9}\n
      Total higgs fraction : {14} +/- {15}
      '''.format(PDF_latex, number[j], seed[i], np.size(c), c[0], c[-1],
          sigma_c_0, sigma_c_err0, sigma_c_pow, unit_sig, sigma_c_end,
```

```
            sigma_c_errend, sigma_f_0, sigma_f_err0, higgs_frac, higgs_frac_err)
        data_file.write(text)

else:

    name1 = '{0}/c_total_N{1}_S{2}_c1_{3}_c2_{4}_c3_{5}.png'\
    .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2],
        c_par[ind3])
    name2 = '{0}/PDF_subplots_N{1}_S{2}_c1_{3}_c2_{4}_c3_{5}.png'\
    .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2],
        c_par[ind3])
    name3 = '{0}/PDF_fullplot_N{1}_S{2}_c1_{3}_c2_{4}_c3_{5}.png'\
    .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2],
        c_par[ind3])
    name4 = '{0}/PDF_fitlinear_N{1}_S{2}_c1_{3}_c2_{4}_c3_{5}.png'\
    .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2],
        c_par[ind3])
    name5 = '{0}/PDF_fullplot_and_lineplot_N{1}_S{2}_c1_{3}_c2_{4}_c3_{5}.png'\
    .format(results_dir, number[j], seed[i], a_par[ind1], b_par[ind2],
        c_par[ind3])

    with
        open('{0}/PDF_data_N{1}_S{2}_c1_{3}_c2_{4}_c3_{5}.txt'.format(results_dir,
        number[j], seed[i], a_par[ind1], b_par[ind2], c_par[ind3]),'w') as
        data_file:
        text = '''
        PDF used           : {0}
        Number of simulations: {1}
        Seed used          : {2}\n
        Total cross section for {3} c values from {4} to {5}:\n
        Total cross section for c: ({6} +/- {7})*10e{8} {9} for c = {4}
        ({10} +/- {11})*10e{8} {9} for c = {5}
        Total cross section for f:\n({12} +/- {13})*10e{8} {9}\n
        Total higgs fraction : {14} +/- {15}
        '''.format(PDF_latex, number[j], seed[i], np.size(c), c[0], c[-1],
            sigma_c_0, sigma_c_err0, sigma_c_pow, unit_sig, sigma_c_end,
            sigma_c_errend, sigma_f_0, sigma_f_err0, higgs_frac, higgs_frac_err)
        data_file.write(text)

# Save figures in directory specified by name
fig1.savefig(name1)
fig2.savefig(name2)
fig3.savefig(name3)
fig4.savefig(name4)
fig5.savefig(name5)

# End of shifted for loops

# Plot polynomial degree and c values

# Find value of c with best overlap of different slopes

fig6 = plt.figure()
```

```
ax = plt.axis()
plt.errorbar(c_par, lin_k_sm, yerr=[lin_k_sm_err1,lin_k_sm_err2], color='r',
    capsize=cap, label='sm_a', fmt='v')
plt.errorbar(c_par, lin_k_GH, yerr=[lin_k_GH_err1,lin_k_GH_err2], color='b',
    capsize=cap, label='GH_a', fmt='*')
plt.errorbar(c_par, lin_k_HH, yerr=[lin_k_HH_err1,lin_k_HH_err2], color='c',
    capsize=cap, label='HH_a', fmt='^')

plt.xlabel('c')
plt.ylabel('k')
plt.xlim((c_par[0]-10,c_par[-1]+10))
plt.ylim((-0.025,0))
plt.grid()
plt.legend()
name6 = '{0}/PDF_lin_fit_para_S{1}.png'.format(results_dir,seed[0])
fig6.savefig(name6)

fig7 = plt.figure()
ax = plt.axis()
plt.errorbar(c_par[4:53], lin_k_sm[4:53],
    yerr=[lin_k_sm_err1[4:53],lin_k_sm_err2[4:53]], color='r', capsize=cap,
    label='sm_k', fmt='v')
plt.errorbar(c_par[4:53], lin_k_GH[4:53],
    yerr=[lin_k_GH_err1[4:53],lin_k_GH_err2[4:53]], color='b', capsize=cap,
    label='gH_k', fmt='*')
plt.errorbar(c_par[4:53], lin_k_HH[4:53],
    yerr=[lin_k_HH_err1[4:53],lin_k_HH_err2[4:53]], color='c', capsize=cap,
    label='HH_k', fmt='^')

plt.xlabel('c')
plt.ylabel('k')
plt.xlim((c_par[4]-5,c_par[53]+5))
plt.ylim((-0.010,-0.002))
plt.grid()
plt.legend()
name7 = '{0}/PDF_lin_fit_para2_S{1}.png'.format(results_dir,seed[0])
fig7.savefig(name7)
```

B.2 master_yoda.py

```
import numpy as np
import subprocess
import os
import re
import glob
import functools

from collections import OrderedDict

num_format = re.compile('-?\ *\d+\.?\d*(?:[Ee]\ *-?\ *\d+)?')
run_str = ['sm','PP','GH','HH']

############################################################
def trackYoda(function):
# Track if function has been called

   @functools.wraps(function)
   def wrapper(*args, **kwargs):

      wrapper.has_been_called = True
      return function(*args, **kwargs)
   wrapper.has_been_called = False
   return wrapper

############################################################
def readYoda(file_path, decay='pT_T', num_col_expect=10000, raw=False):
# Reads .yoda files file_path and places dsig-values into array data
# ordered dictionary dict_data
# Change decay to access different ouputs of yoda file
# Change number of collisions expected when changing number of simulations
# if the simulation terminated early
# Set raw=True when using RAW/... data

   def convert(line):

      line = line[:-1].split('\t')

      return[float(elem) for elem in line]

   state = 0
   data = []
   dict_data = OrderedDict()
   yoda_key = ['xlow', 'xhigh', 'sumw', 'sumw2', 'sumwx', 'sumwx2',
       'num_Entries']

   with open(file_path, 'rt') as f:

   if raw == True:
      decay_line = 'Path: /RAW/TTBAR/{0}\n'.format(decay)
      col_line = 'Path: /RAW/_EVTCOUNT\n'
```

```
    else:
        decay_line = 'Path: /TTBAR/{0}\n'.format(decay)
        col_line = 'Path: /_EVTCOUNT\n'
    table_line = '# xlow\t xhigh\t sumw\t sumw2\t sumwx\t sumwx2\t numEntries\n'
    table_end_line = 'END YODA_HISTO1D_V2\n'
    col_table_line = '# sumW\t sumW2\t numEntries\n'

    for line in f:
        if state == 0 and line == decay_line:
            state = 1
        # Find scale value of data
        elif state == 1 and line.startswith('ScaledBy:') == True:
            state = 2
            scale = [float(x) for x in re.findall(num_format, line)]
            scale = scale[0]

        elif state == 2 and line == table_line:
            state = 3

        elif state == 3:
            if line == table_end_line:
                state = 4
                data = np.transpose(np.array(data))
            else:
                data.append(convert(line))


        elif state == 4 and line == col_line:
            state = 5

        elif state == 5 and line == col_table_line:
            state = 6

        elif state == 6:
            num_col = convert(line)

            if num_col[2] != num_col_expect:
                ValueError('Simulation stopped after {0} collisions;\
                expected {1} collisions!\n'.format(num_col[2], num_col_expect))

            for k in range(0, np.size(yoda_key)):
                dict_data[yoda_key[k]] = data[k]

            return data, dict_data, scale

    return None

############################################################
def readOut(file_path, num_col_expect=10000, convert=True):
# Read .out file and extract total cross section
# Convert being True results in b units, otherwise nb

    def numAfterp(number):
```

```
# Find number after point

   num_str = str(number)
   if not '.' in num_str:
      return 0
   return len(num_str) - num_str.index('.') - 1

def signDig(numb):
# returns significant digits as integer
# minus added if  signDig after point, otherwise positive
# allows for  scientific notation

   num_str = str(float(numb))

   if not '.' in num_str:
      return len(num_str)
   else:
      while num_str[len(num_str)-1] == '0':
         num_str = num_str[:-1]
      if (len(num_str) - num_str.index('.') - 1) == 0:
         num_str = num_str[:-1]
         return len (num_str)
      else:
         return -(len(num_str) - num_str.index('.') - 1)

# open .out file
with open(file_path, 'rt') as file:

   sigma_lines = file.readlines()

   for line in sigma_lines:

      # find sm, PP, GH or HH value
      if line.startswith\
      (('ME[g,g->tbar,t]','ME[g,h0->tbar,t]','ME[h0,h0->tbar,t]')) == True:

         sigma_value = [float(number) for number in re.findall(num_format,
             line[61:78])]

         sigma = sigma_value[0]*(10**sigma_value[2])

   # eliminate round off errors
         if numAfterp(sigma) > abs(sigma_value[2]) + 10:
            sigma = round(sigma, int(abs(sigma_value[2])))

   # number of digits for error of sigma
         if not sigma_value[2] == 0:
            sigma_err = sigma_value[1]*(10**sigma_value[2])
         else:
            digit = signDig(sigma)
            if digit > 0:
               sigma_err = sigma_value[1]*(10**(digit-1))
            else:
```

```
                sigma_err = sigma_value[1]*(10**(digit))

            if convert == True:
               sigma = sigma*(10**(-9))
               sigma_err = sigma_err*(10**(-9))


      return sigma, sigma_err

############################################################
def valueYoda(dic_path, scale, component='sumw2', precision=6):
# Obtain values, errors and standard deviation
# from dictionary via readYoda and put them into arrays for further use
# Change component to compare different data types, namely sumw, sumwx and sumwx2
# Use scale factor from readYoda function to obtain correct units for values
# Change precision to allow for bigger or smaller errors in the sum of dsig

   # Get values from dictionaries
   x_low = dic_path['xlow']
   x_high = dic_path['xhigh']
   x = (x_low + x_high)/2
   x_err = (x_high - x_low)/2
   y = dic_path['sumw2']
   y = y/scale
   num_Entries = dic_path['num_Entries']

   y_err = np.zeros(len(y))
   y_err_rel = np.zeros(len(y))

   # Check if values add up to one
   sum_part = 0

   for ind3 in range(len(y)):
      num = y[ind3]
      sum_part += num
      if np.round(sum_part,6) != 1:
         ValueError('Sum does not add up to 1, look at data!')

   # Add errors
   for ind4 in range(len(num_Entries)):
      if num_Entries[ind4] == 0:
         y_err[ind4] = 0
         y_err_rel[ind4] = 0
      elif y[ind4] == 0:
         y_err[ind4] = y[ind4]/np.sqrt(num_Entries[ind4])
         y_err_rel[ind4] = 10
      else:
         y_err[ind4] = y[ind4]/np.sqrt(num_Entries[ind4])
         y_err_rel[ind4] = y_err[ind4]/y[ind4]

   return x, x_err, y, y_err, y_err_rel, num_Entries

############################################################
```

```
@trackYoda
def bashYoda(bash_path, num_herwig=1000, seed_herwig=7):
# Creates bash file to call in terminal for simulation
# Change seed, number of collisions or name of files
# Change results for different processes

   with open('bash_TTbar.sh','wt') as file:

      #run_str = ['sm','PP','GH','HH']
      herwig_start = '''#!/bin/bash\n
# bash file, see master_yoda.py for more detail
ve() { source $1/bin/activate; }
ve .
'''

      plugin = '''
make IntrinsicHiggs.so
rivet-buildplugin TTBAR.cc
export
    RIVET_ANALYSIS_PATH=/home/fabianv/Documents/University/Bachelor_Thesis/Herwig_update\n
'''
      generate = ''

      for k in range(np.size(run_str)):
         sim = '''
find . \( -type d -name '*scratch*' -or -type d -name '*Herwig-cache*' \) -exec
    rm -rf {{}} \;
Herwig read {0}.in
Herwig run {0}.run -N {1} -s {2}
         '''.format(run_str[k],num_herwig,seed_herwig)
         generate = generate + sim


      text = herwig_start + plugin + generate
      file.write(text)

      return None

############################################################
@trackYoda
def changeYoda(PDF_py):
# Get shape of pdf in HiggsPDF.cc and change it

############################################################
   def formulaC(pdf_py):
   # Change python formula from numpy to C++ formula
   # Can contain powers, roots and exponentials

      dict = {
   'np.power': 'pow',
   'np.sqrt' : 'sqrt',
   'np.exp' : 'exp',
   }
```

```
    for key, value in dict.items():
        pdf_c = pdf_py.replace(key, value)

        return ' return {};\n'.format(pdf_c)
#############################################################


    script_dir = os.path.dirname(__file__)

    def listToString(s):
    # Change list to string

        str1 = ''
        return (str1.join(s))

    PDF = formulaC(PDF_py)

    for outfile in glob.glob(os.path.join(script_dir,'HiggsPDF.cc')):
        with open(outfile,'rt') as file:

            lines1 = file.readlines()
            lines2 = listToString(lines1)

            PDF_old = lines1[39]
            PDF_new = PDF

        with open(outfile,'wt') as file:

            newlines = lines2.replace(PDF_old,PDF_new)
            file.write(newlines)

            return PDF_old, PDF_new

#############################################################
@trackYoda
def deleteYoda(seed=[]):
# Delete all files connected to seed

    if seed == []:
        return None
    else:
        for i in range(np.size(seed)):

            herwig_files = glob.glob('*-S{}*'.format(seed[i]))

            for file in herwig_files:

                os.remove(file)

            return None

#############################################################
```

```
def sigmaYodac(sigma, sigma_err, c):
# Calculate total cross section and errors using sigma values and test parameter
    c
# scaled by sig_sm/sig_PP

   sigma_tot_c = ((1-c)**2)*sigma[0] +  ((1-c)*c)*sigma[0]*sigma[2]/sigma[1]\
   + (c**2)*sigma[0]*sigma[3]/sigma[1]

   sigma_tot_sm_err = 1-(c**2) + c*(1-c)*sigma[2]/sigma[1] +
       (c**2)*sigma[3]/sigma[1]
   sigma_tot_PP_err = -c*(1-c)*sigma[0]*sigma[2]/(sigma[1]**2)\
   - (c**2)*sigma[0]*sigma[3]/(sigma[1]**2)
   sigma_tot_gH_err = c*(1-c)*sigma[0]/sigma[1]
   sigma_tot_HH_err = (c**2)*sigma[0]/sigma[1]

   sigma_tot_c_err = np.abs(sigma_tot_sm_err)*sigma_err[0]\
   +np.abs(sigma_tot_PP_err)*sigma_err[1]+\
   np.abs(sigma_tot_gH_err)*sigma_err[2]+np.abs(sigma_tot_HH_err)*sigma_err[3]

   # Check if error is reasonable
   if sigma_tot_c_err >= sigma_tot_c:
      sigma_tot_c_err = sigma_tot_c

   return sigma_tot_c, sigma_tot_c_err

############################################################
def sigmaYodaf(sigma, sigma_err, f, f_err):
# Calculate total cross section and errors using sigma values and higgs fraction
    f
# scaled by sig_sm/sig_PP

   sigma_tot_f = ((1-f)**2)*sigma[0] +  ((1-f)*f)*sigma[0]*sigma[2]/sigma[1]\
   + (f**2)*sigma[0]*sigma[3]/sigma[1]

   sigma_tot_sm_err = 1-(f**2) + f*(1-f)*sigma[2]/sigma[1]+
       (f**2)*sigma[3]/sigma[1]
   sigma_tot_PP_err = -f*(1-f)*sigma[0]*sigma[2]/(sigma[1]**2)\
   - (f**2)*sigma[0]*sigma[3]/(sigma[1]**2)
   sigma_tot_gH_err = f*(1-f)*sigma[0]/sigma[1]
   sigma_tot_HH_err = (f**2)*sigma[0]/sigma[1]

   sigma_tot_frac_err = 2*f*sigma[0]*((sigma[3]-sigma[2])/sigma[1]+1)\
   + sigma[0]*(sigma[2]/sigma[1] - 2)

   # Add additional error term arising from higgs fraction error
   sigma_tot_f_err = np.abs(sigma_tot_sm_err)*sigma_err[0]\
   +np.abs(sigma_tot_PP_err)*sigma_err[1] +
       np.abs(sigma_tot_gH_err)*sigma_err[2]\
   +np.abs(sigma_tot_HH_err)*sigma_err[3]+np.abs(sigma_tot_frac_err)*f_err

   # Check if error is reasonable
   if sigma_tot_f_err >= sigma_tot_f:
      sigma_tot_f_err = sigma_tot_f
```

```
    return sigma_tot_f, sigma_tot_f_err


############################################################
def higgsError(fraction, sigma, sigma_err, sign, delta_rel = 0.03):
# Calculate error for higgs fraction f

    # Using shortcuts for simplification
    sigma_a = 1 - (sigma[3] - sigma[2])/sigma[1]
    sigma_b = sigma[2]/sigma[1] - 2
    sigma_root = np.sqrt(sigma_b**2-4*(sigma_a)*delta_rel)

    # Calculate partial derivates of PP, GH and HH,
    # plus and minus for positive and negate delta_rel
    dPP = 1/(4*sigma_a**2)*((sigma[2]/(sigma[1]**2)\
    +sign*sigma_root**2*sigma_a*(-2*sigma_b*sigma[2]/(sigma[1]**2)\
    -4*delta_rel*(sigma[3]-sigma[2])/(sigma[1]**2))\
    -4*fraction*sigma_a*(sigma[3]-sigma[2])/(sigma[1]**2)))
    dPP = np.abs(dPP)

    dGH =
        1/(4*sigma_a**2)*((-1/sigma[1]+sign*sigma_root**2*sigma_a*(2*sigma_b/sigma[1]\-4*delta_re
        - 4*fraction*sigma_a/sigma[1]))
    dGH = np.abs(dGH)

    dHH =
        1/(4*sigma_a**2)*((sign*sigma_root**2*sigma_a*(4*delta_rel/sigma[1])\+4*fraction*sigma_a/
    dHH = np.abs(dGH)

    f_err = dPP*sigma_err[1]+dGH*sigma_err[2]+dHH*sigma_err[3]

    # Check if error is reasonable, just for practice reasons
    if f_err is not None:
       if f_err > 0.1:
          f_err = 0.1
          return abs(f_err)
       else:
          return abs(f_err)
    else:
       f_err = 1
       return abs(f_err)
```

B.3 pdf_plot.py

```
import numpy as np
import matplotlib.pyplot as plt
import os
from matplotlib.pyplot import cm

prog_path = os.path.dirname(__file__)

a_par = np.linspace(1.6,1.6,1)
b_par = np.linspace(0.6,0.6,1)
c_par = np.array([2,20,50,100,200,500,1000])

# Taken from Data-files provided by pdf_yoda.py
c_par_gH = np.array([51,80])
f = np.array([0.0163,0.0160,0.0152,0.0151,0.0151,0.0151,0.0151])
f_gH = np.array([0.0152,0.0151])
c_par_gH_err = np.array([7,20])

y_val = np.power(1.0-x_val,1.6)*\np.
power(x_val,0.6)*np.exp(-1*c_par_gH[ind2]*x_val)
y_val_err = np.power(1.0-x_val,1.6)*
\np.power(x_val,0.6)*np.exp(-1*c_par_gH[ind2]*x_val)
y_val_d = np.power(1.0-x_val,1.6)*
\np.power(x_val,0.6)*np.exp(-1*(c_par_gH[ind2]-c_par_gH_err[ind2])*x_val)
y_val_u = np.power(1.0-x_val,1.6)*
\np.power(x_val,0.6)*np.exp(-1*(c_par_gH[ind2]+c_par_gH_err[ind2])*x_val)

############################
# Figure with the old PDF #
############################

fig8 = plt.figure()
ax = plt.axes()

# Initialize color
color1 = iter(cm.rainbow(np.linspace(0,1,len(c_par))))

x_val = np.linspace(0,1,1000)
y_val_old = np.power(1.0-x_val,1.6)*np.power(x_val,0.6)

# Plot old PDF
plt.plot(x_val,y_val_old, label=r'$c=0, f=2.00\%$',color='k')

#Plot new PDFs
for ind1 in range(len(c_par)):
   c = next(color1)
   y_val =
       np.power(1.0-x_val,1.6)*np.power(x_val,0.6)*np.exp(-1*c_par[ind1]*x_val)
   plt.plot(x_val,y_val, label=r'$c={0},
       f={1}\%$'.format(c_par[ind1],100*f[ind1]), color=c)
```

```
plt.xlabel(r'$x$')
plt.ylabel(r'$log\left(PDF(x)\right)$')
ax.legend()
ax.set_yscale('log', nonposy='clip')
plt.grid()
plt.xlim((0,1))
plt.ylim((10**(-50),10*np.max(y_val_old)))
fig8.savefig(prog_path + '/Higgs_PDF_with_old.png')

plt.close()

##############################
# Figure without the old PDF #
##############################

fig9 = plt.figure()
ax = plt.axes()

# Initialize color
color2 = iter(cm.rainbow(np.linspace(0,1,len(c_par))))

x_val = np.linspace(0,1,1000)
pdf_c_max = np.zeros(len(c_par_gH))
process = ['HH','gH']

#Plot new PDFs
for ind2 in range(len(c_par_gH)):
   c = next(color2)

   plt.plot(x_val,y_val, label=r'$c_{{{0}}}={1} \pm {2},\
   f={3}\%$'.format(process[ind2],c_par_gH[ind2],\
   c_par_gH_err[ind2],100*f_gH[ind2]), color=c)
   plt.plot(x_val,y_val_d, linestyle='--', color=c)
   plt.plot(x_val,y_val_u, linestyle='--', color=c)

   pdf_c_max[ind2] = max(y_val)

pdf_max = max(pdf_c_max)

plt.xlabel(r'$x$')
plt.ylabel(r'$log\left(PDF(x)\right)$')
ax.legend()
ax.set_yscale('log', nonposy='clip')
plt.grid()
plt.xlim((0,1))
plt.ylim((10**(-50),10*pdf_max))

fig9.savefig(prog_path + '/Higgs_PDF_without_old.png')
plt.close()
```

C Bibliography

[1] Avnish and Kirtiman Ghosh. *Multi-charged TeV scale scalars and fermions in the framework of a radiative seesaw model*. 2020. arXiv: 2007.01766 [hep-ph].

[2] Elias Kiritsis. *String theory in a nutshell*. Vol. 21. Princeton University Press, 2019.

[3] Fernbach Simon. "Higgs-PDF study in proton-proton collisions". Master Thesis. Uni Graz, May 28, 2019.

[4] Reiner Franziska. "The form of the Higgs-PDF in the proton". Bachelor Thesis. Uni Graz, Sept. 19, 2019.

[5] 't Hooft Gerard. *Gauge theories*. July 1, 2020. URL: http://www.scholarpedia.org/article/Gauge_theories (visited on 07/01/2020).

[6] Thomson Mark. *Modern particle physics*. Cambridge University Press, 2013.

[7] Fernbach Simon et al. "Constraining the Higgs boson valence contribution in the proton". In: *Physical Review D* 101.11 (June 2020). ISSN: 2470-0029. DOI: 10.1103/physrevd.101.114018. URL: http://dx.doi.org/10.1103/PhysRevD.101.114018.

[8] Martha Constantinou et al. *Parton distributions and lattice QCD calculations: toward 3D structure*. 2020. arXiv: 2006.08636 [hep-ph].

[9] Maas Axel. "Brout–Englert–Higgs physics: From foundations to phenomenology". In: *Progress in Particle and Nuclear Physics* 106 (2019), pp. 132–209.

[10] Bonneau G. "Local operator". In: *Scholarpedia* 4.9 (2009). revision #91439, p. 9669. DOI: 10.4249/scholarpedia.9669.

[11] Maas Axel. "Field theory as a tool to constrain new physics models". In: *Modern Physics Letters A* 30.29 (2015), p. 1550135.

[12] Török Pascal and Maas Axel. *On observable particles in theories with a Brout-Englert-Higgs effect*. 2018. arXiv: 1806.11373 [hep-lat].

[13] Bellm Johannes et al. "Herwig 7.0/Herwig++ 3.0 release note". In: *The European Physical Journal C* 76.4 (Apr. 2016). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-016-4018-8. URL: http://dx.doi.org/10.1140/epjc/s10052-016-4018-8.

[14] Johannes Bellm et al. *Herwig 7.1 Release Note*. 2017. arXiv: 1705.06919 [hep-ph].

[15] Bähr Manuel et al. "Herwig++ physics and manual". In: *The European Physical Journal C* 58.4 (Nov. 2008), pp. 639–707. ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-008-0798-9. URL: http://dx.doi.org/10.1140/epjc/s10052-008-0798-9.

[16] A. V. Kotikov and A. I. Onishchenko. *DGLAP and BFKL equations in $\mathcal{N} = 4$ SYM: from weak to strong coupling*. 2019. arXiv: 1908.05113 [hep-th].

[17] G. Aad et al. "Measurement of the $t\bar{t}$ production cross-section and lepton differential distributions in $e\mu$ dilepton events from pp collisions at $\sqrt{s} = 13\,\text{TeV}$ with the ATLAS detector". In: *The European Physical Journal C* 80.6 (June 2020). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-020-7907-9. URL: http://dx.doi.org/10.1140/epjc/s10052-020-7907-9.

[18] Matteo M. Defranchis. *Measurements of the inclusive* $\mathrm{t\bar{t}}$ *production cross section at the ATLAS and CMS experiments*. 2019. arXiv: 1901.10898 [hep-ex].

List of Figures

1 Two body scattering of particles a and b . . . 3
2 Example of a collision highlighting the differential cross section with respect to the angle Ω . . . 4
3 Four special forms of the PDF of quarks $q^p(x)$ . . . 5
4 Forms of the PDF made with MRST at $Q^2 = 10GeV$, taken from [6] . . . 6
11 Example of a Feynman diagram for a QED annihilation process of $e^+e^- \to \mu^+\mu^-$ 10
5 Feynman diagrams for up and down quarks and their antiparticles . . . 11
6 Feynman diagrams for charm and strange quarks and their antiparticles . . . 12
7 Feynman diagram for bottom quarks and antiparticles . . . 12
8 Feynman diagrams for gluons . . . 13
9 Feynman diagrams for processes involving gluon and Higgs in initial state . 13
10 Feynman diagrams for processes involving a Higgs-Higgs pair in initial state 14
12 $PDF = (1-x)^{1.6}x^{0.6}$ . . . 19
13 $PDF = (1-x)^{1.6}x^{0.6}e^{-5x}$ . . . 19
14 $PDF = (1-x)^{1.6}x^{0.6}e^{-20x}$ . . . 20
15 pT_T for $PDF = (1-x)^{1.6}x^{0.6}e^{-50x}$ . . . 20
16 $PDF = (1-x)^{1.6}x^{0.6}e^{-100x}$ . . . 21
17 $PDF = (1-x)^{1.6}x^{0.6}e^{-250x}$ . . . 21
18 $PDF = (1-x)^{1.6}x^{0.6}e^{-500x}$ . . . 22
19 $PDF = (1-x)^{1.6}x^{0.6}e^{-750x}$ . . . 23
20 $PDF = (1-x)^{1.6}x^{0.6}e^{-1000x}$ . . . 23
21 Slopes of fitcurves from 4.1 . . . 24
22 Slopes of fitcurves from 4.1 in overlap area . . . 25
23 Comparison with old $PDF(x) = (1-x)^{1.6}x^{0.6}$ . . . 26
24 $PDF^H_{HH}(x)$ and $PDF^H_{gH}(x)$ . . . 27
25 Allowed higgs fraction f for PDF^H_R . . . 29
26 Allowed higgs fraction f for $c = 1$ . . . 29
27 Allowed higgs fraction f for $c = 2$ . . . 30
28 Allowed higgs fraction f for $c = 5$ . . . 30
29 Allowed higgs fraction f for $c = 10$ . . . 31
30 Allowed higgs fraction f for $c = 1000$ . . . 31