Master Thesis
Parton Showers Near Threshold
Bart Verouden

supervisor: Prof. Dr. E.L.M.P Laenen
Abstract

It has become unthinkable to do modern high energy physics, and especially the interpretation of high energy physics experiments such as those done at the Large Hadron Collider, without the use of numerical simulation programs like event generators. Parton shower algorithms are an essential part of event generators, and it is this topic we will discuss here. In this master thesis we will construct a modified initial state parton shower to simulate the showering off spacelike initial state quarks in a Drell-Yan process that is near its threshold. In order to do so, we will first discuss the underlying physics, such as the parton distribution functions, their evolution equations (DGLAP-equations), and calculate the splitting functions. Next we will construct a conventional final and initial state parton shower, and discuss the different algorithms used. As an intermezzo we will discuss a modified version of the MC@NLO toy model from [21] that, through the appearance of an additional log, gives an indication what to expect in a threshold resummed parton shower. Finally we will discuss some of the theory underlying threshold resummation, and present our model, along with results.
Acknowledgments

I would like to thank my supervisor Professor Eric Laenen for his support and guidance during my master’s research. I believe he has found the right balance between giving me the chance to discover things for myself and interfering when needed, which I found very stimulating. Also his continuous pressing to write out all intermediate results has proven to be invaluable.

Furthermore I would like to thank the PhD students in my office, and the office adjacent to mine, namely Reinier, Michele, Rob, Sander and Lisa, for their moral support and for making my time there fun and inspiring. My gratitude also goes to my fellow smokers for all the time spent on the balcony, regardless of weather. The Nikhef theory group as a whole also have my thanks for the necessary 3 o’clock distraction around the pingpong table.

Finally I would like to give a special thanks to my girlfriend Annelies for her moral support, everlasting interest and endless patience, as well as my parents for their support throughout my studies.
Contents

Abstract iii

Acknowledgments iv

Table of Contents v

1 The QCD-improved Parton Model 1
   1.1 The Parton Distribution Functions 2
   1.2 The naive parton model 3
   1.3 The parton model from field theory 6
   1.4 The parton model and QCD 9
   1.5 Parton branching 17
   1.6 Coherent Branching 25

2 Parton shower simulations 29
   2.1 The Veto Algorithm 29
      2.1.1 The Standard Veto Algorithm 30
      2.1.2 Multivariable Veto algorithm 32
      2.1.3 Numerical Test 33
   2.2 Final state shower 36
      2.2.1 The Sudakov form factor 36
      2.2.2 Kinematics 40
      2.2.3 Evolution 42
      2.2.4 Results 43
   2.3 Initial state shower 46
      2.3.1 Evolution 47
      2.3.2 Kinematics 50
      2.3.3 Results 52

3 MC@NLO toy model with log-term 57
   3.1 MC@NLO 57
      3.1.1 NLO 57
      3.1.2 MC 58
      3.1.3 Matching NLO and MC 58
      3.1.4 Results 59
   3.2 The log model 39
      3.2.1 Exclusive observable 63
      3.2.2 Inclusive observable 65
      3.2.3 Results 66

4 Threshold Resummation 70
   4.1 Theory 70
   4.2 Parton shower implementation 74

A Code 79
   A.1 Code Final State Shower 79
      A.1.1 shower.h 79
      A.1.2 shower.cpp 80
      A.1.3 parton.h 83
      A.1.4 parton.cpp 86
   A.2 Code Initial State Shower 94
      A.2.1 isshower.h 94
      A.2.2 isshower.cpp 96
A.2.3 isparton.h ............................ 105
A.2.4 isparton.cpp .......................... 108
A.3 Code MC@NLO toy model .............. 116
   A.3.1 MCNLO.h ........................... 116
   A.3.2 logterm.c ........................... 117
A.4 Additional Code ......................... 123
   A.4.1 bindat.h ............................ 123
   A.4.2 bindat.cpp .......................... 125
   A.4.3 vec4d.h ............................. 134
   A.4.4 vec4d.cpp ........................... 135
   A.4.5 veto_toy.c ........................... 138
A t small $Q^2$ the proton is observed as a point particle.

(b) At intermediate $Q^2$ the valence quarks are observed.

(c) At large $Q^2$ we also see sea quarks.

Figure 1: Increasing the momentum transfer increases the number of particles seen.

Before the 1970's protons and neutrons were thought to be the fundamental indivisible particles that made up the atomic nucleus. It was, however, unknown what kept the positive charged protons inside the nucleus from repelling each other. It was postulated that some other force had to be present to overcome the electromagnetic repulsion. Since, at small distance scales, it had to be stronger than the electromagnetic force, it was introduced as the "strong" force, and was considered a fundamental force acting directly on the protons. It was discovered that this force was only a residual side-effect of another force acting on particles inside the proton, later to be found quarks and gluons. Nowadays, the term "strong force" is used for the force acting on those particles, whereas the residual force is referred to as the nuclear force. In 1969 Richard Feynman proposed the basic parton model as a way to analyze high-energy hadron collisions [1]. In the basic the parton model, hadrons contain a number of point-like particles called partons. Since the hadron moves at relativistic speeds, it is Lorentz-contracted and time dilated, effectively "freezing" the partons inside the hadron. Hence, colliding particles will scatter incoherently, which was verified experimentally [6]. Later, it was realized that partons describe the same particles we now know as quarks and gluons.

1 The QCD-improved Parton Model

Since no free quarks or gluons are found in experiments, the coupling constant of the strong interaction $\alpha_s$ must increase at larger distance scales, i.e. smaller momentum transfer $Q^2$, making the theory non-perturbative in the coupling constant. Quantum Chromo Dynamics (QCD) tells us that the evolution of the coupling constant is given by

$$ Q \frac{d}{dQ} \alpha_s(Q) = \beta_0 \frac{\alpha_s^2(Q)}{2\pi} - \beta_1 \frac{\alpha_s^3(Q)}{8\pi} - \ldots $$

(1)

It was found that that a hadron consists of 3 valence quarks, e.g. a proton consists of two up and one down quarks. When looking closer, i.e. higher momentum transfer, one finds that the number of quarks increases (see Figure [1]). These extra quarks are called the sea quarks, and always appear in a quark anti-quark pair. Hence, the sum of the charges of all quarks found inside the proton remains one. However the sum of the momenta of the quarks did not add up to the momentum of the proton. Apparently a part of the momentum is carried by other partons. These turned out to be gluons, as was later seen in the form of 3-jet events at the $e^+e^-$ collider PETRA in Hamburg [2].

So depending on the momentum transfer $Q^2$ the distribution of observed partons changes. This distribution is described by the Parton Distribution Functions (PDF’s): $\phi_g(x,Q)$, $\phi_f(x,Q)$ and $\phi_{\bar{f}}(x,Q)$. Here $g$ stands for gluon and $f$ and $\bar{f}$ are the different types of flavor of quarks and anti-quarks.
1.1 The Parton Distribution Functions

The PDF’s give the differential chance to find a parton carrying a fraction $x$ of the total momentum of the hadron when probing the hadron with a momentum transfer $Q^2$. Obviously we have

$$\int_0^1 dx \sum_i \phi_i(x) = 1$$

(2)

at any scale $Q^2$, which is known as the same rule. As we will later derive, the evolution of the PDF’s is described by an integro-differential equation:

$$Q \frac{d}{dQ} \phi_i(x, Q) = \frac{\alpha_s(Q)}{2\pi} \sum_j \int_0^1 \frac{dy}{y} P_{ij}(y) \phi_j\left(\frac{x}{y}, Q\right),$$

(3)

where $P_{ij}(x)$ are the so-called splitting functions. They can be interpreted as the probabilities of finding a parton of type $i$ in a parton of type $j$ with a fraction $x$ of the longitudinal momentum of the parent parton (and a transverse momentum much less than $Q^2$). We will discuss the splitting functions in more detail in section 1.5.

We are going to try and solve Eq. (3) using a Mellin transform. The Mellin transform $\tilde{f}(N)$ of a function $f(x)$ is defined as

$$\tilde{f}(N) = \int_0^1 dx x^{N-1} f(x),$$

(4)

and its inverse

$$f(x) = \int_{c-i\infty}^{c+i\infty} \frac{dN}{2\pi i} e^{-N \ln x} \tilde{f}(N).$$

(5)

For simplicity we will assume that there is only one type of parton. Performing the Mellin transform on both sides of Eq. (3) we get

$$Q \frac{d}{dQ} \tilde{\phi}(N, Q) = \frac{\alpha_s(Q)}{2\pi} \int_0^1 dx x^{N-1} \int_0^1 \frac{dy}{y} P(y) \phi\left(\frac{x}{y}, Q\right).$$

(6)

Since $y \geq x$, we can write the r.h.s. of Eq. (6) as

$$\frac{\alpha_s(Q)}{2\pi} \int_0^1 dx x^{N-1} \int_0^1 \frac{dy}{y} P(y) \int_0^1 dz \phi(z, Q) \delta(z - \frac{x}{y})$$

$$= \frac{\alpha_s(Q)}{2\pi} \int_0^1 dx x^{N-1} \int_0^1 dy \int_0^1 dz \delta(x - yz) P(y) \phi(z, Q),$$

since $\delta(x - yz)$ ensures that $y = \frac{z}{x} \geq x$. Hence, Eq. (6) simplifies to

$$Q \frac{d}{dQ} \tilde{\phi}(N, Q) = \frac{\alpha_s(Q)}{2\pi} \tilde{P}(N) \tilde{\phi}(N, Q).$$

(7)

We shall solve Eq. (7) using separation of variables. We have

$$\frac{1}{\phi(N, Q)} \frac{\partial}{\partial Q} \tilde{\phi}(N, Q) = \frac{\alpha_s(Q)}{2\pi} \frac{\tilde{P}(N)}{Q}$$

$$\Rightarrow \int_{\tilde{\phi}(N, Q_0)}^{\tilde{\phi}(N, Q)} \frac{1}{\phi(N, Q')} d\tilde{\phi}(N, Q') = \int_{Q_0}^{Q} \frac{\alpha_s(Q')}{2\pi} \tilde{P}(N) \frac{dQ'}{Q'}. \quad (8)$$

Eq. (1) tells us that

$$d\alpha_s(Q) = \frac{\partial \alpha_s(Q)}{\partial Q} dQ = \frac{\beta(\alpha_s)}{Q} dQ \Rightarrow dQ = \frac{Q}{\beta(\alpha_s)} d\alpha_s.$$
Figure 2: Deep inelastic lepton-proton scattering.

So we get

$$
\ln \frac{\tilde{\phi}(N, Q)}{\tilde{\phi}(N, Q_0)} = \int_{\alpha_s(Q_0)}^{\alpha_s(Q)} d\alpha_s \frac{\tilde{P}(N)}{2\pi \beta(\alpha_s)}
$$

$$
\Rightarrow \tilde{\phi}(N, Q) = \tilde{\phi}(N, Q_0) \exp \left( \int_{\alpha_s(Q_0)}^{\alpha_s(Q)} \frac{d\alpha_s}{\beta(\alpha_s)} \frac{\tilde{P}(N)}{2\pi \alpha_s} \right). \quad (9)
$$

Entering the expansion of $\beta(\alpha_s)$ from Eq. (1) into Eq. (9), and keeping only the $\beta_0$-term, the exponent in the r.h.s. becomes

$$
- \int_{\alpha_s(Q_0)}^{\alpha_s(Q)} \frac{d\alpha_s}{\alpha_s} \frac{\tilde{P}(N)}{\beta_0} = - \ln \frac{\alpha_s(Q)}{\alpha_s(Q_0)} \frac{\tilde{P}(N)}{\beta_0} = \ln \left( \frac{\alpha_s(Q)}{\alpha_s(Q_0)} \right)^{-\frac{\tilde{P}_0}{\alpha_s}}.
$$

Hence

$$
\tilde{\phi}(N, Q) = \tilde{\phi}(N, Q_0) \left( \frac{\alpha_s(Q)}{\alpha_s(Q_0)} \right)^{-\frac{\tilde{P}_0}{\alpha_s}}.
$$

We can conclude that the evolution of the PDF is determined by the evolution of the running coupling, and by the "anomalous dimension" $\tilde{P}(N)$.

1.2 The naive parton model

In this section we will derive a naive parton model by assuming that the structure functions of the proton, measured by deep inelastic scattering, obey the Bjorken scaling law. This means that in the Bjorken limit, which is the limit where the momentum transfer $Q^2 \to \infty$, the structure functions $F_i(x, Q^2)$ scale. That is

$$
F_i(x, Q^2) \to F_i(x),
$$

where $x$ is some dimensionless parameter defined below. This would imply that the proton consists of pointlike constituents, because otherwise the structure functions would depend on the ratio $\frac{Q}{Q_0}$, where $\frac{1}{Q_0}$ would be some measure indicating the size of the constituents.

If we look at a deep inelastic lepton-proton scattering as shown in Fig. 2 we can define the standard deep inelastic variables, which are (in the proton’s rest frame) given by

$$
Q^2 = -q^2
$$

$$
M^2 = p^2
$$

$$
\nu = p \cdot q = p \cdot (k - k') = M (E - E')
$$

$$
x = \frac{Q^2}{2\nu} = \frac{Q^2}{2M(E - E')}
$$

$$
y = \frac{q \cdot p}{k \cdot p} = \frac{M(E - E')}{ME} = 1 - \frac{E'}{E},
$$

(12)
where \( p \) is the momentum of the proton, \( k \) and \( k' \) the momentum of respectively the incoming and outgoing lepton, and \( q \) the momentum transfer given by

\[
q^\mu = k^\mu - k'^\mu. \tag{13}
\]

The structure functions of the proton are defined by the lepton-proton scattering cross section. For the process \( l p \rightarrow l X \), the differential cross section reads

\[
d^2\sigma/dxdy = \left[ \frac{8\pi\alpha^2 ME}{Q^4 \left(1 + (1 - y)^2\right)} \right] \left(2xF_1 + (1 - y)(F_2 - 2xF_1) - \frac{M}{2E} xyF_2\right), \tag{14}
\]

the structure of which will be derived in the next section. In Eq. (14) we have ignored the possibility of the interaction by a neutral \( Z \)-boson instead of a photon. This approximation is valid as long as \( Q^2 \ll M_z^2 \). Using

\[
y = \frac{E - E'}{E} = \frac{\nu}{ME} = \frac{Q^2}{2xME}, \tag{15}
\]

we can rewrite Eq. (14) as

\[
d^2\sigma/dxdQ^2 = \frac{d^2\sigma}{dxdy dQ^2} = \frac{d^2\sigma}{dxdy 2xME} = \frac{4\alpha^2\pi}{Q^4} \left[ (1 + (1 - y)^2) F_1 + \frac{1-y}{x} (F_2 - 2xF_1)\right], \tag{16}
\]

where we have neglected the mass of the proton.

Now consider the deep inelastic scattering of a lepton and a quark carrying a fraction \( \xi \) of the proton’s mass, i.e. \( p_q^\mu = \xi p^\mu \). In terms of the Mandelstam variables \( \hat{s} = (k + p_q)^2 \), \( \hat{t} = (k - k')^2 \) and \( \hat{u} = (p_q - k')^2 \), the amplitude squared for the reaction

\[
e^- (k) + q(p_q) \rightarrow e^- (k') + q (p'_q) \tag{17}
\]

is given, according to chapter 3 of [2], by

\[
\sum |\mathcal{M}|^2 = 2e_q^2 e^4 \hat{s}^2 \hat{u}^2 = \frac{16\pi}{\hat{t}^2}, \tag{18}
\]

where the \( \sum \) denotes the average over the initial state’s and the sum over the final state’s spins and colors. The “\(^-\)“ indicates that we are considering a quark instead of a proton. Using Eq. (12), we can rewrite the Mandelstam variables

\[
\hat{t} = (k - k')^2 = q^2 = -Q^2 \\
\hat{u} = (p_q - k') = -2p_q \cdot k' = -2\xi ME' \\
= 2\xi E \left(\frac{E'}{E}\right) = 2p_q \cdot k \left(1 - \frac{E'}{E} - 1\right) = \hat{s}(y - 1). \tag{19}
\]

Now using the standard result for massless \( 2 \rightarrow 2 \) scattering

\[
\frac{d\hat{\sigma}}{dt} = \frac{1}{16\pi \hat{s}^2} \sum |\mathcal{M}|^2 \tag{20}
\]

we get

\[
\frac{d\hat{\sigma}}{dQ^2} = \frac{d\hat{\sigma}}{dt} \frac{2e_q^2 e^4 \hat{s}^2 + \hat{u}^2}{16\pi \hat{s}^2 \hat{t}^2} = \frac{2\pi\alpha^2 e_q^2}{Q^2 \hat{s}^2} \left(\hat{s}^2 + \hat{u}^2(y - 1)^2\right) = \frac{2\pi\alpha^2 e_q^2}{Q^2} \left(1 + (1 - y)^2\right). \tag{21}
\]
where we used that $\alpha = \frac{e^2}{4\pi}$. The mass-shell condition for the outgoing quark is

$$0 = p_q'^2 = (p_q + q)^2 = q^2 + 2p_q \cdot q = 2p \cdot q \left( \frac{-Q^2}{2p \cdot q} + \xi \right) = -2p \cdot q(\xi - x)$$

which implies that $\xi = x$. Now by writing

$$\int_0^1 dx \delta(x - \xi) = 1,$$

we get

$$\frac{d^2\hat{\sigma}}{dx dQ^2} = 4\pi \alpha^2 \frac{(1 + (1 - y)^2)}{Q^2} \frac{1}{2} e_q^2 \delta(x - \xi).$$

Comparing Eqs. (16) and (24), we see that we have

$$\hat{F}_2 = 2x \hat{F}_1 = xe_q^2 \delta(x - \xi).$$

This suggests that $F_2(x)$ probes a constituent with a momentum fraction of $\xi = x$. Actual measurements clearly show that the structure functions are distributions in $x$, instead of a delta function. This means that the proton exists of constituents with a range of momentum fractions. This has lead to the ‘naive parton model’\cite{3}, in which the structure functions for the proton are

$$F_2 = 2xF_1 = \sum_{q,d} \int_0^1 d\xi q(\xi) xe^2_q \delta(x - \xi)$$

$$= \sum_{q,d} e^2_q xq(x),$$

where $q(\xi) d\xi$ represents the probability that a quark $q$ carries a momentum fraction between $\xi$ and $\xi + d\xi$. The relation

$$F_2 = 2x f_1$$

in Eq. (26), is known as the Callan-Gross relation, and, when observed, was a first indication of the fermionic nature of the quarks, of which it is a direct consequence. Note that the two terms in Eq. (16) correspond to the absorption of transversly ($F_1$) and longitudinally ($F_2 - 2xF_1$) polarized virtual photons. The Callan-Gross relation follows from the fact that, because of the Ward identity, a spin-$\frac{1}{2}$ fermion cannot absorb a longitudinally polarized photon.

When the proton is probed at a momentum scale $Q$, all flavors of quarks with mass $m_q \ll Q$ are to be considered. For example, if we consider 4 quark flavors, we have for the scattering of a lepton off a proton by means of a photon exchange

$$F_2(x) = x \left[ \frac{4}{9} (u(x) + \bar{u}(x) + c(x) + \bar{c}(x)) 
+ \frac{1}{9} (d(x) + \bar{d}(x) + s(x) + \bar{s}(x)) \right].$$

As it turns out, if we also consider $Z$-exchange for the lepton-proton scattering, and $W$-exchange for neutrino-proton scattering\cite{2}, the picture emerges that the proton exists of three valence quarks, and a sea of quark-antiquark pairs

$$u(x) = u_v(x) + S(x)$$
$$d(x) = d_v(x) + S(x)$$
$$S(x) = \bar{u}(x) = \bar{d}(x) = s(x) = \bar{s}(x),$$

\hspace{1cm} (29)
with the following sum rules
\[ \int_0^1 dx u_\nu(x) = 2 \quad \int_0^1 dx d_\nu(x) = 1 \]
\[ \sum_q \int_0^1 dx [q(x) + \bar{q}(x)] = \int_0^1 dx [u_\nu(x) + d_\nu(x) + 6S(x)] \approx 0.5. \] (30)

This indicates that the quarks carry about 50% of the protons' momentum, which is an experimental result. The other half is carried by gluons which do not couple to electric charge.

1.3 The parton model from field theory

To derive the structure of Eq. (14), we again consider the process shown in Fig. 2, but now from a field theoretical perspective. The amplitude for the process \( lP \rightarrow lX \), where we again neglect the contribution by means of a Z-boson, is given by
\[ A = e\bar{u}(k') \gamma^\alpha u(k) \frac{1}{q^2} \langle X | j_\alpha(0) | P \rangle , \] (31)
where \( j_\alpha \) represents the current operator to which the exchange photon couples. The meaning of its coordinate taken at zero will become clear later. The most interesting feature about this structure is that the cross section, which is proportional to the amplitude squared, can be written as the product of a purely leptonic, and a purely hadronic part:
\[ \frac{d^2\sigma}{dx dy} \propto L_{\alpha\beta} W^{\alpha\beta} , \] (32)
where \( L \) and \( W \) are the leptonic and the hadronic parts respectively. The leptonic part is determined straightforwardly by QED:
\[ L_{\alpha\beta} = e^2 \text{Tr} [\gamma^\mu \gamma^\nu \gamma_{\alpha\beta}] = 4e^2 k^\mu k^\nu (\eta_{\mu\nu} - \eta_{\mu\alpha} \eta_{\nu\beta} + \eta_{\mu\beta} \eta_{\nu\alpha}) = 4e^2 (k_\alpha k'_\beta + k_\beta k'_\alpha - \eta_{\alpha\beta} k^2) . \] (33)
The hadronic part now contains all the information about the interaction of the electromagnetic current \( j_\alpha \) and the proton
\[ W_{\alpha \beta}(p,q) = \frac{1}{4\pi} \sum_X \left\langle P \right| j^\dagger_\beta(0) | X \rangle \langle X | j_\alpha(0) | P \rangle \left(2\pi\right)^4 \delta(q+p-p_X) \]
\[ = \frac{1}{4\pi} \sum_X \int d^4z e^{iz(q+p-p_X)} \left\langle P \right| j^\dagger_\beta(0) | X \rangle \langle X | j_\alpha(0) | P \rangle \]
\[ = \frac{1}{4\pi} \int d^4z e^{iz} \sum_X \left\langle P \right| e^{izp_X} j^\dagger_\beta(0) e^{-izp_X} | X \rangle \langle X | j_\alpha(0) | P \rangle \]
\[ = \frac{1}{4\pi} \int d^4z e^{iz} \sum_X \left\langle P \right| \left(\bar{X} | j_{\beta}(z) j_\alpha(0) \right) P \rangle \]
\[ = \frac{1}{4\pi} \int d^4z e^{iz} \left\langle P \right| \left(\bar{X} | j_{\beta}(z) j_\alpha(0) \right) P \rangle , \] (34)
where in the second line we introduced an integral representation for the delta function, in the fourth line we use the fact that \( p \) and \( p_X \) are the eigenvalues of the operator \( \hat{p} \) acting on \( P \) respectively \( X \), and in the fifth line we used the completeness of the final states \( X \). The second
term in the commutator vanishes because \( J_{\beta}^\dagger \) \( (z) \, |P\rangle = 0 \), since \( |P\rangle \) denotes the ground state of the proton, and \( J_{\beta}^\dagger \) lowers the energy of what it acts upon. The factor \( \frac{1}{4\pi^2} \) introduced in the first line is just a convention.

The hadronic part is a rank two tensor which must be symmetric, and parity and Lorentz invariant. The most general form is thus

\[
W_{\alpha\beta} = A\eta_{\alpha\beta} + Bq_{\alpha}q_{\beta} + C(p_{\alpha}q_{\beta} + p_{\beta}q_{\alpha}) + Dp_{\alpha}p_{\beta},
\]

where \( A, B, C \) and \( D \) are functions of \( p^2 \) and \( q^2 \), or, equivalently, of \( x \) and \( Q^2 \). Furthermore, since the electromagnetic current is conserved, we know that

\[
0 = q^\alpha W_{\alpha\beta} = Aq_{\beta} + Bq^2q_{\beta} + C(p \cdot q)q_{\beta} + Cq^2p_{\beta} + D(p \cdot q)p_{\beta}
\]

\[
\Rightarrow \begin{cases} 
A + Bq^2 + C(p \cdot q) = 0 \\
Cq^2 + D(p \cdot q) = 0
\end{cases}.
\]

If we now choose

\[
A = W_1(x, Q^2), \quad D = W_2(x, Q^2),
\]

we get

\[
C = -\frac{(p \cdot q)}{q^2}W_2(x, Q^2) = \frac{1}{2x}W_2(x, Q^2),
\]

and

\[
B = -\frac{A + (p \cdot q)C}{q^2} = -\frac{1}{q^2}W_1(x, Q^2) + \left(\frac{1}{2x}\right)^2 W_2(x, Q^2).
\]

Hence we find for \( W_{\alpha\beta} \) the most general form:

\[
W_{\alpha\beta} = \left(\eta_{\alpha\beta} - \frac{q_{\alpha}q_{\beta}}{q^2}\right)W_1(x, Q^2) + \left(\eta_{\alpha\beta} - \frac{q_{\alpha}q_{\beta}}{q^2}\right)W_2(x, Q^2) + \left(\frac{1}{2x}\right)^2 q_{\alpha}q_{\beta} + p_{\alpha}p_{\beta}
\]

\[
= \left(\eta_{\alpha\beta} - \frac{q_{\alpha}q_{\beta}}{q^2}\right)W_1(x, Q^2) + \left(\frac{1}{2x}\right)^2 q_{\alpha}q_{\beta} + p_{\alpha}p_{\beta}
\]

\[
= \left(\eta_{\alpha\beta} - \frac{q_{\alpha}q_{\beta}}{q^2}\right)W_1(x, Q^2) + \left(p_{\alpha} + \frac{1}{2x}q_{\alpha}\right)\left(p_{\beta} + \frac{1}{2x}q_{\beta}\right)W_2(x, Q^2).
\]

Hence, the functions \( W_1 \) and \( W_2 \) contain the information about how the hadron is 'seen' by the photon. If we now contract this with the leptonic part and compare this with Eq. (14), we get

\[
F_1(x, Q^2) = W_1(x, Q^2)
\]

\[
F_2(x, Q^2) = \nu W_2(x, Q^2).
\]

To work with the hadronic tensors we now define two lightlike vectors \( p \) and \( n \), such that \( p \cdot n = 1 \). It is now possible to write any four vector as

\[
k^\mu = ap^\mu + bn^\mu + k_T^\mu,
\]

where \( k_T \) is a spacelike two dimensional four vector such that

\[
p^2 = n^2 = k_T \cdot p \cdot k_T = 0.
\]

We can choose \( p \) to be the four momentum of the (hadronic) target in the approximation that we ignore its mass. This would give

\[
P^\mu = p^\mu + \frac{M^2}{2}n^\mu,
\]

\[
q^\mu = \nu n^\mu + q_T^\mu,
\]

where \( q_T^2 = -q^2 = Q^2 \). For simplicity we shall ignore the mass of the proton, in which case \( p^\mu = P^\mu \). Using Eq. (14) we can project the functions \( W_{1,2} \) out of the hadronic tensor

\[
\nu n^\alpha n^\beta W_{\alpha\beta} = \nu \left(n^2 - \frac{(n \cdot q)^2}{q^2}\right)W_1 + \nu \left(p \cdot n + \frac{n \cdot q}{2x}\right)^2 W_2 = \nu W_2 = F_2,
\]
Figure 3: Handbag diagram.

since \( n \cdot q = \nu n^2 + n \cdot q_T = 0 \) because of Eq. (43), and

\[
\frac{4x^2}{\nu} \rho^\rho W_{\alpha\beta} = \frac{4x^2}{\nu} \left( p^2 - \frac{(p \cdot q)^2}{q^2} \right) W_1 + \frac{4x^2}{\nu} \left( p^2 + \frac{p \cdot q}{2x} \right)^2 W_2
\]

\[
= \frac{4x^2}{\nu} \left( -\frac{\nu^2}{q^2} W_1 + \frac{\nu^2}{4x^2} W_2 \right) = \nu W_2 - 2x W_1 = F_L \equiv F_2 - 2xF_1. \tag{46}
\]

The naive parton model assumes that the photon scatters incoherently off pointlike constituents. To calculate the amplitude \( W_{\alpha\beta} \) we must square the corresponding matrix element, which is identical to multiplying it by its complex conjugate. This can be pictorially represented by attaching the mirror image of the Feynman diagram to the outgoing side of the original diagram, inserting a dashed line where the two diagrams meet to indicate that the lines are to be taken on mass-shell. The hadronic part of the amplitude can therefore be obtained from a simple handbag diagram as shown in Fig. 3.

\[
W_{\alpha\beta} = e_q^2 \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \gamma^\beta (k + q) \gamma^\alpha B(k, p) \right] \delta \left( (k + q)^2 \right), \tag{47}
\]

where the delta function indicates the mass-shell constraint for the outgoing quark. We can write the four momentum \( k^\mu \) of the quark as in Eq. (42). For the naive parton model to make sense we have to assume that the component of \( k^\mu \) transverse to the proton momentum is small, i.e. \( k^\mu \approx \xi p^\mu \). Hence we write

\[
k^\mu = \xi p^\mu + An^\mu + k_T^\mu. \tag{48}
\]

From

\[
k^2 = 2A\xi - k_T^2, \tag{49}
\]

where \( k_T^2 = -k_T \cdot k_T \geq 0 \) since \( k_T \) is spacelike, we derive

\[
A = \frac{k^2 + k_T^2}{2\xi}, \tag{50}
\]

resulting in

\[
k^\mu = \xi p^\mu + \frac{k^2 + k_T^2}{2\xi} n^\mu + k_T^\mu. \tag{51}
\]

Since we assume that the transverse momentum \( k_T^2 \), and also the virtuality \( k^2 \) of the quark are small, we can rewrite the delta function in Eq. (47), as

\[
\delta \left( (k + q)^2 \right) = \delta(k^2 + q^2 + 2k \cdot q) \approx \delta \left( 2\xi \nu + 2q_T \cdot k_T - Q^2 \right) \approx \delta \left( 2\xi \nu - Q^2 \right) = \frac{1}{2\nu} \delta \left( \xi - x \right). \tag{52}
\]
This leads to

\begin{align}
F_2 = \nu n^\alpha n^\beta W_{\alpha\beta} &= \frac{e_q^2}{2} \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \hat{p} (\hat{k} + \hat{q}) \hat{n} B(k, p) \right] \delta(\xi - x) \\
&= \frac{e_q^2}{2} \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \left( 2\hat{p}(\hat{k}\cdot\hat{n} + \hat{q}\cdot\hat{n}) + n^2(\hat{k} + \hat{q}) \right) B(k, p) \right] \delta(\xi - x) \\
&= e_q^2 \frac{\nu}{x} \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \hat{p} B(k, p) \right] \delta(\xi - x) \\
&= e_q^2 \nu x q(x),
\end{align}

where the quark distribution is

\begin{equation}
q(x) = \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \hat{p} B(k, p) \right] \delta(n \cdot k - x).
\end{equation}

Hence, the structure function $F_2$ does not violate Bjorken scaling, i.e. it depends only on the dimensionless variable $x$, not on $Q^2$. We can do the same for the longitudinal structure function

\begin{align}
F_L = F_2 - 2xF_1 &= \frac{4x^2}{\nu} \nu^\alpha \nu^\beta W_{\alpha\beta} \\
&= \frac{2x^2}{\nu^2} \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \hat{p} (\hat{k} + \hat{q}) \hat{n} B(k, p) \right] \delta(\xi - x) \\
&= \frac{2x^2}{\nu^2} \int \frac{d^4k}{(2\pi)^4} \text{Tr} \left[ \hat{p} B(k, p) \right] \left( \frac{k^2 + q^2}{x} + 2\nu \right) \delta(\xi - x) \\
&= O(\nu^{-1})
\end{align}

which vanishes in the Bjorken limit $Q^2 \to \infty$. We therefore again retrieve the Callan-Gross relation, Eq. (27)

\begin{equation}
F_2 = 2xF_1,
\end{equation}

which is typical for the spin-$\frac{1}{2}$ nature of the constituents as, explained after Eq. (27).

### 1.4 The parton model and QCD

In QCD it is found that the Bjorken scaling of the structure functions is broken by logarithms of $Q^2$. This is because the assumption that the transverse momenta of the quarks is small, does no longer hold in QCD. By emitting a gluon it is possible for a quark to get a transverse momentum up to $k_T^2 \sim Q^2$. This will lead to contributions proportional to $\alpha_s \log Q^2$ which breaks scaling as we will show in this section. In order to see this, we will calculate the structure functions of a quark which can emit a gluon. This is the $O(\alpha_s)$ correction to the parton model result $F_2 = e_q^2 \delta(x - \xi)$, as found in section 1.2.
We will first calculate the structure function $\hat{F}_2$ from the scattering of a virtual photon off a quark, as shown in Fig. 4(a), with momentum $p^\mu$, i.e. $\xi = 1$, to establish a normalization. The scattering amplitude is given by

$$M_\alpha = -ie_q \bar{u}(l)\gamma_\alpha u(p),$$

and hence the squared matrix element is

$$\sum |M|^2_{\alpha\beta} = e^2_q \bar{u}(p)\gamma_\beta u(l)\bar{u}(l)\gamma_\alpha u(p) = \frac{1}{2} e^2_q \text{Tr}[\gamma_\beta\gamma_\alpha],$$

where we summed, and averaged over spins. To check the normalization for $\hat{F}_2$ we have to calculate the quantity

$$\nu n^\alpha n^\beta \sum |M|^2_{\alpha\beta} = \nu e^2_q \text{Tr}[n^\beta n^\alpha] = 4\nu e^2_q n \cdot l = 4\nu e^2_q.$$  

The one-dimensional phase space is

$$d\Phi_1 = 2\pi \delta (p+q)^2 = \frac{\pi}{\nu} \delta (1-x),$$

where we used Eq. (52) with $\xi = 1$. If we combine this, we see that, to get the correct expression for $\hat{F}_2$ as in Eq. (25), we need to insert a factor of $\frac{1}{4\pi}$, which is referred to in [2] as the “flux factor”, and which comes from Eq. (34). We then get back the naive parton model result with $\xi = 1$

$$\hat{F}_2(x) = e^2_q \delta (1-x).$$

Next, we will consider the processes of Fig. 4(b,c), where the parton emits a gluon, respectively before and after interacting with the photon. Since these are indistinguishable, there are four terms in the amplitude squared. These four terms are represented by the four diagrams shown in Fig. 5. As it turns out, in the light-cone gauge, only the first diagram, Fig. 5(a), gives a logarithmic divergence, where the other diagrams give only finite corrections. Therefore we will focus on this contribution and leave the finite part undetermined. Notice that Fig. 5(a) has the structure of a handbag diagram, as in Fig. 3, where the amplitude $B$ is a single gluon exchange. The two-dimensional phase space for this contribution is

$$d\Phi_2 = \int \frac{d^4r}{(2\pi)^4} \frac{d^4l}{(2\pi)^4} \delta^+(r^2) \delta^+(l^2) (2\pi)^4 \delta^4 (p+q-r-l).$$
Performing one of the integrals and introducing \( k^\mu = p - r = l - q \) as the momentum of the struck proton we get
\[
d\Phi_2 = \frac{1}{4\pi^2} \int d^4k \delta^+( (p - k)^2 ) \delta^+( (k + q)^2 ) .
\]
(63)

We can again decompose \( k^\mu \) in terms of \( p^\mu, n^\mu \) and a transverse vector \( k_T^\mu \), as in Eq. (51):
\[
k^\mu = \xi p^\mu + \frac{k_T^2 + k^2}{2\xi} n^\mu + k_T^\mu .
\]

For simplicity we choose
\[
p = (p, 0, 0, p)
\]
\[
n = \left( \frac{1}{2p}, 0, 0, -\frac{1}{2p} \right),
\]
and since \( k_T \cdot p = k_T \cdot n = 0 \) we have
\[
k_T = \left( 0, k_T^{(1)}, k_T^{(2)}, 0 \right),
\]
hence
\[
k = \left( \xi p + \frac{k_T^2 + k^2}{2p} \frac{1}{2\xi}, k_T^{(1)}, k_T^{(2)}, \xi p - \frac{k_T^2 + k^2}{2p} \right).
\]
(66)

Now to rewrite the four dimensional volume element \( d^4k \) in the variables \( k^2, \xi, k_T^{(1)} \) and \( k_T^{(2)} \), we have to calculate the Jacobian which is
\[
J = \begin{vmatrix}
1 & 1 & 1 & 1 \\
\frac{k_T^{(1)}}{2p} & 0 & 0 & -\frac{k_T^{(1)}}{2p} \\
\frac{k_T^{(2)}}{2p} & 1 & 0 & -\frac{k_T^{(2)}}{2p} \\
\frac{k_T^2 + k^2}{2p} & 0 & 1 & -\frac{k_T^2 + k^2}{2p}
\end{vmatrix}
= \frac{1}{2\xi} \left( p + \frac{k_T^2 + k^2}{2} \frac{1}{2p} \right) + \frac{1}{2\xi} \left( p - \frac{k_T^2 + k^2}{2} \frac{1}{2p} \right)
= \frac{1}{2\xi}.
\]
(68)

So we have
\[
d^4k = \frac{d\xi}{2\xi} dk^2 d^2k_T.
\]
(69)

Now writing \( k^2 = -|k^2| \), we get
\[
(p - k)^2 = \left( 1 - \xi \right) p - \frac{k_T^2 - |k^2|}{2\xi} n - k_T
\]
\[
= \left( 1 - \xi \right) \frac{k_T^2 - k_T^2 - 1 - \xi k_T^2}{\xi} = (1 - \xi) \frac{k_T^2}{\xi} - \frac{k_T^2}{\xi},
\]
\[
(k + q)^2 = -|k^2| - Q^2 + 2k \cdot q = 2\xi \nu - 2q_T \cdot k_T - |k^2| - Q^2,
\]
where we used that \( k_T \cdot k_T = -k_T^2 \), and \( q^\mu \) is as in Eq. (44). Using these expressions we can rewrite the phase space integral to
\[
d\Phi_2 = \frac{1}{4\pi^2} \int d\xi \frac{k_T^2 + k_T}{2\xi} \delta \left( \frac{k_T^2}{\xi} - (1 - \xi) \frac{|k^2|}{\xi} \right) \delta \left( 2\xi \nu - Q^2 - |k^2| - 2q_T \cdot k_T \right)
\]
\[
= \frac{1}{16\pi^2\nu} \int d\xi dk^2 dk_T^2 d\theta \delta \left( k_T^2 - (1 - \xi) |k^2| \right) \delta \left( \xi - x - \frac{|k^2| + 2q_T \cdot k_T}{2\nu} \right)
\]
(72)
with $0 < \theta < \pi$.

The matrix element corresponding to Fig. 4(b) is

$$-ig_q \bar{u}(l) \gamma^\alpha \frac{1}{k^2} \ell t^A u(p),$$

(73)

leading to

$$\sum |M|_{\alpha\beta}^2 = \frac{1}{4\pi} \epsilon^*_{\mu} \epsilon_{\nu} \sum_{\text{pol}} C_F \text{Tr} \left[ \gamma^\beta \left( k + q \right) \gamma^\alpha \ell \ell^* k \right] |k^2| \frac{1}{k^4}.$$  \hspace{1cm}

(74)

To perform the sum over the polarization of the gluon, we use

$$\sum_{\text{pol}} \epsilon_{\mu} \epsilon^*_{\nu} = -g_{\mu\nu} + \frac{n_{\mu} r_\nu + n_{\nu} r_\mu}{n \cdot r},$$

(75)

with $r_\mu$ the 4-momentum of the gluon.

We can now project out the contribution to $\hat{F}_2$ by using the vector $n$ as in Eq. (45). Using the kinematic relations we find

$$\frac{1}{4\pi} \alpha_s \sum |M|_{\alpha\beta}^2 = \frac{8 \epsilon_q^2 \alpha_s}{|k^2|} \xi P(\xi),$$

(76)

where $P(\xi)$ is known as the splitting function:

$$P(\xi) = C_F \frac{1 + \xi^2}{1 - \xi}.\hspace{1cm}

(77)$$

Eq. (76) can be derived by using a symbolic manipulation system, such as FORM\[4\]. Code to such a calculation is shown in Fig. 6. The appearance of the $\frac{1}{|k^2|}$ together with the $dk^2$ in the phase space integral leads to a logarithmic divergence. We will not do the calculation here, but to summarize, the four diagrams of Fig. 5, together with the leading order diagram give a structure function

$$\hat{F}_2(x, Q^2) = e_q^2 x \left[ \delta(1 - x) + \frac{\alpha_s}{2\pi} \left( P(x) \ln \frac{Q^2}{\kappa^2} + C(x) \right) \right],$$

(78)

where $\kappa$ is a small cut-off value for the momentum integration of the struck parton and $C$ is a calculable finite function. This regularization method is just a choice. We could also have chosen dimensional regularization, but in that case the breaking of Bjorken scaling would not be manifest.

Hence we see that at next-to-leading order (NLO) the structure function is $Q^2$ dependent, thus breaking Bjorken scaling by logarithms of $Q^2$. The quark distribution function that emerges from Eq. (78)

$$q(x, Q^2) = \delta(1 - x) + \frac{\alpha_s}{2\pi} \left( P(x) \ln \frac{Q^2}{\kappa^2} + C(x) \right)$$

(79)

is, however, not complete. To obtain the full result to this order in perturbation theory, we need to include diagrams with virtual gluons, such as those shown in Fig. 7. The calculations of these diagrams is straightforward using standard loop-integral methods. The ultraviolet divergences appearing in the individual diagrams cancel in the sum. There is however a physical argument provided by Altarelli and Parisi\[3\], that provides a shortcut to the answer. Since the outgoing quark is on shell, all the diagrams including virtual gluons are proportional to $\delta \left( (p + k)^2 \right) = \delta (-Q^2 + 2\nu) \propto \delta(1 - x)$. So the splitting function becomes something like

$$P(x) \rightarrow P(x) + K \delta(1 - x).$$

(80)

Furthermore, we can say that the final splitting function must integrate to zero, since otherwise the number of quarks, which is given by $\int_0^1 dx \ q(x, Q^2)$ would be $Q$ dependent, which would violate baryon number conservation. This can be achieved with

$$P(x) = C_F \left[ \frac{1 + x^2}{(1 - x)_+} + \frac{3}{2} \delta(1 - x) \right].$$

(81)
vector k, q, p, n, r, kT;
symbols eq, g, Cf, k2, kT2, ksi;
Indices alpha, beta, mu, nu, i;
Off statistics;
Local f = 2^-1*Cf * n(alpha)*n(beta) * eq^-2*g^-2 * (k.k)^-2 *
  (( -d_(mu,nu) + ( n(mu) * r(nu) + n(nu) * r(mu) ) * (n.r)^-1 ) *
   ( g_(1, beta, k, alpha, k, mu, p, nu, k) +
     g_(1, beta, q, alpha, k, mu, p, nu, k) ) );
Trace4,1;
Bracket eq,g k.k;
Print;
.sort
Repeat;
id k.k^-1= k2^-1;
id k.k = k2;
id r = p-k;
id k = ksi*p + (kT2+k2) * (2*ksi)^-1 * n + kT;
id n.n = 0;
id n.p = 1;
id p.p = 0;
id n.kT = 0;
id p.kT = 0;
id q.n = 0;
id kT.kT = -kT2;
id kT2 = -(1-ksi)*k2;
id n.r^-1 = (1-ksi)^-1;
EndRepeat
Print;
.sort
AntiBracket ksi;
Bracket eq, g, Cf,k2;
Print;
.end

Figure 6: FORM code deriving Eq. \([76]\)

Figure 7: Squared Feynman diagrams for virtual gluon emission contributing to deep inelastic scattering
where the ‘plus’ distribution is defined by

$$\int_0^1 dx \frac{f(x)}{(1 - x)_+} = \int_0^1 dx \frac{f(x) - f(1)}{(1 - x)} ,$$

so that its integral with any sufficiently smooth function is finite, and

$$\frac{1}{(1-x)_+} = \frac{1}{1-x} \quad \text{for} \quad 0 \leq x < 1 .$$

The one-gluon contribution to the longitudinal structure function, $F_L = \hat{F}_2 - 2x \hat{F}_1$, can be calculated in a similar way, by calculating

$$\frac{x^2}{\pi \nu^2} \eta^\alpha \eta^\beta \sum |\mathcal{M}|^2_{\alpha \beta} .$$

The leading contribution again comes from Fig. 5(a), but in this case there is no (logarithmic) divergence. The result for the longitudinal structure function is

$$\hat{F}_L (x, Q^2) = e_q^2 x \left[ 0 + \frac{\alpha_S}{2\pi} C_F 2x \right] .$$

In order to interpret the divergence in the structure function Eq. (78), we must first notice that the singularity arises at $k_T = 0$, i.e. when the gluon is emitted parallel to the quark. For that reason this divergence is often referred to as a collinear divergence. The limit $k_T^2 \to 0$ corresponds to a long-range part of the strong interaction, inaccessible to perturbation theory. Now, to get a proton structure function we must introduce a set of ‘bare’ quark distributions $q_0$, one for each type of quark, and sum over the different flavors, as we did in section 1.2 for the naive parton model. The chance that a proton contains a quark with momentum fraction $x$, is equal to the chance that it contains a ‘bare’ quark with momentum $\xi$, $x \leq \xi \leq 1$, which, in turn, contains a quark with a relative momentum fraction $0 \leq \frac{x}{\xi} \leq 1$. Or

$$F_2 (x, Q^2) = \sum_{q, \bar{q}} \int_0^1 d\xi \, q_0 (\xi) \hat{F}_2 \left( \frac{x}{\xi}, Q^2 \right)$$

$$= \sum_{q, \bar{q}} \int_0^1 d\xi e_q^2 \frac{x}{\xi} q_0 (\xi) \left[ \xi \delta (\xi - x) + \frac{\alpha_S}{2\pi} \left\{ P(x) \ln \frac{Q^2}{k^2} + C(x) \right\} + \mathcal{O} (\alpha^2_S) \right]$$

$$= x \sum_{q, \bar{q}} e_q^2 \left[ q_0 (x) + \frac{\alpha_S}{2\pi} \int_0^1 d\xi \, q_0 (\xi) \left\{ P(x) \ln \frac{Q^2}{k^2} + C(x) \right\} + \mathcal{O} (\alpha^2_S) \right] .$$

The introduction of a bare quark distribution is similar to the introduction of bare quantities in Lagrangians in the renormalization process of loop divergences. As in renormalization, $q_0(x)$ is considered an unmeasurable distribution. The singularities are absorbed into the bare distribution at the factorization scale $\mu$, which is the equivalent to the renormalization scale. A ‘renormalized’ distribution function $q \,(x, \mu^2)$ can now be defined by

$$q \,(x, \mu^2) = q_0 (x) + \frac{\alpha_S}{2\pi} \int_0^1 d\xi \, q_0 (\xi) \left[ P(x) \ln \frac{\mu^2}{k^2} + C(x) \right] + \mathcal{O} (\alpha^2_S) ,$$

in terms of which we have

$$F_2 (x, Q^2) = x \sum_{q, \bar{q}} e_q^2 \int_0^1 d\xi q (\xi, \mu^2) \left[ \delta \left( 1 - \frac{x}{\xi} \right) + \frac{\alpha_S}{2\pi} P(x) \ln \frac{Q^2}{\mu^2} + \mathcal{O} (\alpha^2_S) \right] .$$
This can be shown by substituting Eq. (87) into Eq. (88), leading to

\[
F_2^{\text{ren}} (x, Q^2) = x \sum_{q, \bar{q}} c_q^2 \int \frac{d\xi}{x} \left\{ q_\xi (\xi) + \frac{\alpha_s}{2\pi} \int \frac{d\xi'}{\xi} q_\xi (\xi') \left[ P \left( \frac{\xi}{\xi'} \right) \ln \frac{\mu^2}{\xi'^2} + C \left( \frac{\xi}{\xi'} \right) \right] \right\} \times 
\]

\[
\left\{ \delta \left( 1 - \frac{x}{\xi} \right) + \frac{\alpha_s}{2\pi} P \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{\mu^2} \right\}
\]

\[
= x \sum_{q, \bar{q}} c_q^2 \left\{ q_\xi (x) + \frac{\alpha_s}{2\pi} \int \frac{d\xi}{x} q_\xi (\xi) \left[ P \left( \frac{x}{\xi} \right) \ln \frac{\mu^2}{\xi^2} + C \left( \frac{\xi}{\xi'} \right) \right] \right\}
\]

\[
= x \sum_{q, \bar{q}} c_q^2 \int \frac{d\xi}{x} q (\xi, \mu^2) \left[ \delta \left( 1 - \frac{x}{\xi} \right) + \frac{\alpha_s}{2\pi} P \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{\mu^2} + O \left( \alpha_s^2 \right) \right]
\]

\[
= F_2 (x, Q^2),
\]

where, to get the final result, we changed the integration variable \( \xi' \) to \( \xi \). Hence we see that the long distance contribution to the structure function is factored into the 'renormalized' quark distribution function. Therefore this procedure is called factorization rather than renormalization. Notice, that at a factorization scale \( \mu^2 = Q^2 \), Eq. (88) becomes \( F_2 (x, Q^2) = x \sum c_q^2 q (x, Q^2) \), so despite the fact that, due to the inclusion of long-distance (i.e. non-perturbative) contributions, the distribution \( q (x, \mu^2) \) cannot be calculated using perturbation theory, it can still be determined experimentally. Factorization is possible up to all orders of \( \alpha_s \), but we will not prove that here. For more details, see ref [3].

In the factorized quark distribution, Eq. (87), besides the divergent terms, we have also included all of the finite terms. This is a choice however and we may choose to include an arbitrary part of \( C \). The way we handle the finite parts defines what is called the ‘factorization scheme’. The example above, where all of the finite contribution is absorbed in the distribution \( q (x, \mu^2) \), is called the DIS (Deep Inelastic Scattering) scheme. A more common choice is to include, besides the divergent part, only an always-present \( \ln (4\pi) - \gamma_E \) into the distribution. This is called the \( \overline{\text{MS}} \) scheme. In the \( \overline{\text{MS}} \) scheme we would have

\[
F_2 (x, Q^2) = x \sum_{q, \bar{q}} c_q^2 \int \frac{d\xi}{x} q (\xi, Q^2) \left[ \delta \left( 1 - \frac{x}{\xi} \right) + \frac{\alpha_s}{2\pi} C_{\overline{\text{MS}}} \left( \frac{x}{\xi} \right) + O \left( \alpha_s^2 \right) \right],
\]

where the \( \overline{\text{MS}} \) \( O(\alpha_s) \) coefficient function for \( F_2 \) is [9]

\[
C_{\overline{\text{MS}}} (z) = C_F \left[ 2 \left( \frac{\ln (1-z)}{1-z} \right) + \frac{3}{2} \left( \frac{1}{1-z} \right) + (1+z) \ln (1-z) - \frac{1}{1-z} \ln z + 3 + 2z - \frac{\pi^2}{3} + \frac{9}{2} \right] \delta (1-z).
\]

If we study Eq. (91) more closely, we see that the first term in the square brackets has a second order pole at \( z = 1 \). The Mellin transform of this term behaves as \( \ln^2 (N) \) for large \( N \), endangering the perturbative nature of the structure function. This can be remedied by means of resummation as will be discussed in section [4].
To get a complete result for the structure functions we must also consider the scattering of the photon off an initial gluon. At $O(\alpha_S)$ this is the process $g \rightarrow q\bar{q}$, for which the Feynman diagrams are shown in Fig. 8. The calculation of this contribution goes similar to the calculation from the $q \rightarrow qg$ contribution as done earlier in this section. The result is

$$\hat{F}_g^q(x, Q^2) = x \sum_{q, \bar{q}} e_q^2 \frac{\alpha_S}{2\pi} \left( P_{qq}(x) \ln \frac{Q^2}{\kappa^2} + C_g(x) \right),$$

(92)

where the splitting function now is

$$P_{qq}(x) = T_R \left[ x^2 + (1-x)^2 \right],$$

(93)

with $T_R$ the color factor $T_R = \frac{1}{2}$ and the sum in (92) is over all quark and antiquark flavors that contribute at this energy scale. Now in order to get the full physical structure function we have to add Eq. (92), again convoluted with a bare gluon distribution $g_0(x)$, to Eq. (86) to get

$$F_2(x, Q^2) = x \sum_q e_q^2 \left[ q_0(x) + \frac{\alpha_S}{2\pi} \int \frac{d\xi}{\xi} g_0(\xi) \left\{ P_{qq} \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{\kappa^2} + C_g \left( \frac{x}{\xi} \right) \right\} + \frac{\alpha_S}{2\pi} \int \frac{d\xi}{\xi} g_0(\xi) \left\{ P_{qg} \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{\kappa^2} + C_g \left( \frac{x}{\xi} \right) \right\} + O(\alpha_S^2) \right],$$

(94)

where we relabeled the original $P$ and $C$ from Eq. (86) to $P_{qq}$ and $C_g$ respectively. We can now redefine the factorized quark distribution function at the factorization scale $\mu$ from Eq. (87) as

$$q(x, \mu^2) = q_0(x) + \frac{\alpha_S}{2\pi} \int \frac{d\xi}{\xi} g_0(\xi) \left\{ P_{qq} \left( \frac{x}{\xi} \right) \ln \frac{\mu^2}{\kappa^2} + C_g \left( \frac{x}{\xi} \right) \right\} + \frac{\alpha_S}{2\pi} \int \frac{d\xi}{\xi} g_0(\xi) \left\{ P_{qg} \left( \frac{x}{\xi} \right) \ln \frac{\mu^2}{\kappa^2} + C_g \left( \frac{x}{\xi} \right) \right\} + O(\alpha_S^2).$$

(95)

By absorbing all of the gluon contribution into the quark distribution like this, following the DIS factorization scheme, we still have $F_2(x, Q^2) = x \sum e_q^2 q(x, Q^2)$. In the MS scheme however, only the divergent part of the gluon distribution is absorbed in the quark distribution, leaving us with
a gluon distribution function \( g(x, \mu^2) \). We then have

\[
F_2(x, Q^2) = \sum_{q, \bar{q}} e_q^2 \int_0^1 \frac{d\xi}{\xi} q(\xi, Q^2) \left[ \delta(1 - \frac{x}{\xi}) + \frac{\alpha_s}{2\pi} C_q^{\overline{\text{MS}}} \left( \frac{x}{\xi} \right) \right] +
\]

\[
\sum_{q, \bar{q}} e_q^2 \int_0^1 \frac{d\xi}{\xi} g(\xi, Q^2) \left[ \frac{\alpha_s}{2\pi} C_g^{\overline{\text{MS}}} \left( \frac{x}{\xi} \right) \right].
\]  

(96)

To study the evolution of the parton distribution functions (PDF’s), let us define \( t = \mu^2 \), and consider the structure function \( F_2(x, t) \) in the simplified case that there is only one type of quark, no antiquark, and no gluon contribution, i.e. Eq. (88) without the sum present. If we now take the \( \ln t \) derivative we get

\[
t \frac{\partial}{\partial t} F_2(x, Q^2) = 0 = t \frac{\partial}{\partial t} x e_q^2 \left( q(x, t) + \int_0^1 \frac{d\xi}{\xi} q(\xi, t) \frac{\alpha_s}{2\pi} P \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{t} \right)
\]

\[
\Rightarrow 0 = t \frac{\partial}{\partial t} q(x, t) - \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} P \left( \frac{x}{\xi} \right) q(\xi, t)
\]

\[
+ \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} \frac{\partial}{\partial t} q(\xi, t) P \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{t}.
\]  

(97)

We find

\[
t \frac{\partial}{\partial t} q(x, t) = \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} P \left( \frac{x}{\xi} \right) q(\xi, t) - \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} t \frac{\partial}{\partial t} q(\xi, t) P \left( \frac{x}{\xi} \right) \ln \frac{Q^2}{t},
\]  

(98)

but since

\[
t \frac{\partial}{\partial t} q(x, t) = \mathcal{O}(\alpha_s) \Rightarrow \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} \frac{\partial}{\partial t} q(\xi, t) = \mathcal{O}(\alpha_s^2).
\]  

(99)

So at \( \mathcal{O}(\alpha_s) \) we have

\[
t \frac{\partial}{\partial t} q(x, t) = \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} P \left( \frac{x}{\xi} \right) q(\xi, t).
\]  

(100)

This equation is known as the (Dokshitzer)-Gribov-Lipatov-Altarelli-Parisi ((D)GLAP) equation. Considering all quarks, antiquarks and the gluon the (D)GLAP equation is actually a set of \( 2n_f + 1 \) equations, which, in matrix form, is given by

\[
t \frac{\partial}{\partial t} \begin{pmatrix} q_i(x, t) \\ g(x, t) \end{pmatrix} = \frac{\alpha_s}{2\pi} \int_0^1 \frac{d\xi}{\xi} \begin{pmatrix} P_{q_i q_j} \left( \frac{x}{\xi} \right) & P_{q_i g} \left( \frac{x}{\xi} \right) \\ P_{g q_j} \left( \frac{x}{\xi} \right) & P_{gg} \left( \frac{x}{\xi} \right) \end{pmatrix} \begin{pmatrix} q_j(x, t) \\ g(x, t) \end{pmatrix},
\]  

(101)

where the sum over the repeated index \( j \) is implicit and runs over all \( n_f \) quark flavours as well as over the \( n_f \) antiquarks. In the next section we will (re)derive the splitting functions that appear in this equation.

### 1.5 Parton branching

In this section we will discuss parton branching of both incoming and outgoing particles, which will result in the derivation of the splitting functions used in the evolution of the PDF’s. Parton branching is a collinear enhancement to the matrix element of a QCD Feynman diagram of any
order of $\alpha_s$. An example of parton branching can be seen in Fig. 9. The shaded blob represents the rest of the diagram.

In Fig. 9 we see the kinematics and notation for the branching of an outgoing parton $a$ into $b$ and $c$, also called timelike branching. We assume that

$$p_b^2, p_c^2 \ll p_a^2 = t.$$  \hfill (102)

The angle between line $b$ and $c$ is $\theta = \theta_b + \theta_c$. Defining the energy fraction

$$z = \frac{E_b}{E_a} = 1 - \frac{E_c}{E_a},$$  \hfill (103)

we have

$$t = p_a^2 = (p_b + p_c)^2 = p_b^2 + p_c^2 + 2p_b p_c.$$  \hfill (104)

Ignoring the mass of the partons we get, for small angles $\theta$

$$t = 2p_b p_c = 2E_b E_c - 2\vec{p}_b \cdot \vec{p}_c = 2E_b E_c (1 - \cos \theta) = z(1 - z)E_a^2 \theta^2 + O(\theta^4).$$  \hfill (105)

Hence

$$\theta = \frac{1}{E_a} \sqrt{\frac{t}{z(1 - z)}},$$  \hfill (106)

or, using transverse momentum conservation

$$E_b \sin \theta_b = E_c \sin \theta_c$$
$$\Rightarrow z E_a \theta_b = (1 - z) E_a \theta_c$$
$$\Rightarrow \theta_b = \frac{1 - z}{z} \theta_c,$$  \hfill (107)

we have

$$\theta = \theta_b + \theta_c = \frac{\theta_b}{1 - z} = \frac{\theta_c}{z}.$$  \hfill (108)

We will first consider the case where all the partons are gluons. There will be a factor $\frac{1}{4}$ in the amplitude because of the propagator of gluon $a$, as well as the triple-gluon vertex

$$V_{ggg} = ig f^{ABC} \epsilon_a^\alpha \epsilon_b^\beta \epsilon_c^\gamma [g_{\alpha\beta}(p_a - p_b)_{\gamma} + g_{\beta\gamma}(p_b - p_c)_{\alpha} + g_{\gamma\alpha}(p_c - p_a)_{\beta}],$$  \hfill (109)

where $\epsilon_i^\mu$ is the polarization vector for gluon $i$. Note that all momenta are defined outgoing, hence $p_a = -p_b - p_c$. Using that $\epsilon_i \cdot p_i = 0$ we can write Eq. (109) as

$$V_{ggg} = ig f^{ABC} [(\epsilon_a \cdot \epsilon_b)\epsilon_c \cdot (-p_c - 2p_b) + (\epsilon_b \cdot \epsilon_c)\epsilon_a \cdot (2p_b - p_a) +$$
$$(\epsilon_c \cdot \epsilon_a)\epsilon_b \cdot (2p_c + p_b)]$$
$$= -2ig \left[ f^{ABC}(\epsilon_a \cdot \epsilon_b)(\epsilon_c \cdot p_b) - (\epsilon_b \cdot \epsilon_c)(\epsilon_a \cdot p_b) -$$
$$(\epsilon_c \cdot \epsilon_a)(\epsilon_b \cdot p_c) \right].$$  \hfill (110)
Since the gluons are almost on shell, we can assume their polarization vectors to be transverse. To describe the polarization vectors of parton $i$ we can choose the basis $\{\epsilon_i^{\text{in}}, \epsilon_i^{\text{out}}\}$, where $\epsilon_i^{\text{in}}$ is in the plane of branching and $\epsilon_i^{\text{out}}$ normal to the plane, so that for small $\theta$, ignoring terms of order $\theta^2$, we have for gluons $i$ and $j$

\[
\epsilon_i^{\text{in}} \cdot \epsilon_j^{\text{in}} = \epsilon_i^{\text{out}} \cdot \epsilon_j^{\text{out}} = -1
\]

and

\[
\epsilon_i^{\text{in}} \cdot p_b = -E_b \sin(\theta_b) = -E_b \theta_b = z(1-z)E_a \theta
\]

\[
\epsilon_i^{\text{out}} \cdot p_c = -E_c \cos(\theta + \frac{\pi}{2}) = E_c \sin(\theta) = E_c \theta = (1-z)E_a \theta
\]

\[
\epsilon_i^{\text{out}} \cdot p_c = -E_c \cos(\frac{\pi}{2} - \theta) = -E_b \sin(\theta) = -E_b \theta = -zE_a \theta,
\]

hence, all terms in the vertex are proportional to $E_a \theta = \sqrt{\frac{1}{z(1-z)}}$.

We can now write the matrix element squared for $n+1$ partons in terms of that for $n$ partons as

\[
|M_{n+1}|^2 \sim \frac{4g^2}{\bar{t}} C_A F(z; \epsilon_a, \epsilon_b, \epsilon_c) |M_n|^2,
\]

where the $\frac{1}{\bar{t}}$ factor comes from the $\frac{1}{t}$ in the propagator of parton $a$ in combination with the $\sqrt{t}$ factor in the vertex terms, and the color factor $C_A = 3$ comes from $f^{ABC} f^{ABC}$. The functions $F(z; \epsilon_a, \epsilon_b, \epsilon_c)$ come from the part of Eq. (110) in brackets, and can be found in Table 1. Any other configuration of the polarizations give zero in Eq. (110) and is therefore not allowed. Defining $\langle F \rangle$ by summing over the polarizations of partons $\bar{b}$ and $c$, and averaging with respect to the polarization of parton $a$, we get

\[
C_A \langle F \rangle = \hat{P}_{gg}(z) = C_A \left[ \frac{1-z}{z} + \frac{z}{1-z} + z(1-z) \right],
\]

where $\hat{P}_{gg}(z)$ is the unregularized $g \to g + g$ splitting function as mentioned in Section 1.1. We can see from Table 1 that the singularities $z \to 0$ ($b$ soft) and $z \to 1$ ($c$ soft) are associated with the emission of a soft gluon with polarization in the plane of branching.

Let us now look at another important polarization effect, namely the correlation between the plane of branching and the polarization of gluon $a$. If we take the polarization of gluon $a$ to be at an angle $\phi$ with the plane of branching, we have $\epsilon_a^\mu = \epsilon_a^{\text{in}} \cos \phi + \epsilon_a^{\text{out}} \sin \phi$, so

\[
F_\phi \propto \sum_{\epsilon_b, \epsilon_c} |M(\epsilon_a^{\text{in}}, \epsilon_b, \epsilon_c) \cos \phi + M(\epsilon_a^{\text{out}}, \epsilon_b, \epsilon_c) \sin \phi|^2
\]

\[
= \cos^2 \phi \left[ \frac{1-z}{z} + \frac{z}{1-z} + 2z(1-z) \right] + \sin^2 \phi \left[ \frac{1-z}{z} + \frac{z}{1-z} \right]
\]

\[
= \frac{1-z}{z} + \frac{z}{1-z} + z(1-z) + z(1-z) \cos 2\phi.
\]

Notice that there are no cross-terms since, according to Table 1, for any configuration of $\epsilon_b$ and $\epsilon_c$ at least one of the two terms inside the sum vanishes. The last term of Eq. (115) gives the
Figure 10: $g \rightarrow q\bar{q}$ branching

correlation, which indicates a preferred polarization in the plane of branching given by $\phi = 0$. The polarized term is maximal for $z = \frac{1}{2}$, giving a contribution of only $\frac{1}{2}$ of the unpolarized part. The polarization effect for gluon-gluon splitting is thus quite small.

We will next discuss the branching of a gluon into a quark-antiquark pair, $g \rightarrow q\bar{q}$, see Fig. 10. We will again find a $\frac{1}{2}$ factor in the amplitude because of the gluon propagator. The vertex factor, however, now becomes

$$V_{gq\bar{q}} = -ig t^a_{bc} \bar{u}^b \gamma_\mu \epsilon_{a}^i v^c$$

where $u^b$ and $v^c$ are the quark and antiquark spinors. We will keep the same gluon polarization conventions used in the $g \rightarrow gg$ case, but we will have to introduce the quark and antiquark helicity states. We will use the conventions of Bjorken and Drell for spinors and Dirac algebra, in which

$$\gamma^0 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \gamma^i = \begin{pmatrix} 0 & \sigma_i \\ -\sigma_i & 0 \end{pmatrix}$$

where the matrix elements are $2 \times 2$ matrices and the $\sigma_i$ are the Pauli spin matrices. Using this specific representation of the Dirac algebra, we will search for solutions of the Dirac equation that are eigenstates of the helicity operator. Since we consider the fermions to be massless, the Dirac equation for quark $b$ reads

$$\not{p}_b u(p_b) = 0$$

Choosing our coordinates as in Fig. 10 we have for small angles $\theta_b$

\[
p_b = p_0 \begin{pmatrix} 1 \\ \sin \theta_b \\ 0 \\ \cos \theta_b \end{pmatrix} = p_0 \begin{pmatrix} 1 \\ \theta_b \\ 0 \\ 1 \end{pmatrix},
\]

so that

\[
\not{p} = \gamma_\mu p_\mu = \left( \begin{array}{c} p_0^0 \\ p_0 \cdot \hat{\sigma} \\ -\vec{p} \cdot \hat{\sigma} \end{array} \right) = p_0 \begin{pmatrix} 1 & 0 & -1 & -\theta_b \\ 0 & 1 & -\theta_b & 1 \\ 1 & \theta_b & -1 & 0 \\ \theta_b & -1 & 0 & -1 \end{pmatrix}.
\]

The four components of $u(p_b)$ can be an arbitrary function of $\theta_b$, but since $\theta_b$ is very small we can write it as a Taylor series in $\theta_b$ and keep only the zeroth and first order term. Hence, we can write

$$u(p_b) = \begin{pmatrix} a_1 + a_2 \theta_b \\ b_1 + b_2 \theta_b \\ c_1 + c_2 \theta_b \\ d_1 + d_2 \theta_b \end{pmatrix}.$$
The Dirac equation, Eq. (118) now becomes a set of equations:

\[
\begin{align*}
    a_1 - c_1 + \theta_b(a_2 - c_2 - d_1) &= 0 \\
    b_1 + d_1 + \theta_b(b_2 - c_1 + d_2) &= 0 \\
    a_1 - c_1 + \theta_b(a_2 - c_2 + b_1) &= 0 \\
    -b_1 - d_1 + \theta_b(a_1 - b_2 - d_2) &= 0
\end{align*}
\]  
(122)

since Eq. (122) must hold for any \( \theta_b \). This gives us two independent sets of solutions:

\[
\begin{align*}
    u^b_+(p_b, \alpha_+) &= c_+(p^0_b) \begin{pmatrix} 1 \\ \frac{1}{2} \theta_b + \alpha_+ \theta_b \\ \frac{1}{2} \theta_b - \alpha_+ \theta_b \end{pmatrix}, \\
    u^b_-(p_b, \alpha_-) &= c_-(p^0_b) \begin{pmatrix} -\frac{1}{2} \theta_b + \alpha_- \theta_b \\ \frac{1}{2} \theta_b + \alpha_- \theta_b \\ -1 \end{pmatrix}
\end{align*}
\]  
(124)

where \( \alpha_+ \) and \( \alpha_- \) can take any value, and \( c_+ \) and \( c_- \) are normalization factors later to be determined. To find the helicity eigenstates we act on these states with the helicity operator

\[
h_b \equiv \hat{p}_b \cdot \vec{S} = \frac{1}{2} \hat{p}^i_b \begin{pmatrix} \sigma^i & 0 \\ 0 & \sigma^i \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & \theta_b & 0 & 0 \\ \theta_b & -1 & 0 & 0 \\ 0 & 0 & 1 & \theta_b \\ 0 & 0 & \theta_b & -1 \end{pmatrix},
\]  
(125)

so that we get

\[
\begin{align*}
    h u^b_+(p_b, \alpha_+) &= \frac{1}{2} c_+(p^0_b) \begin{pmatrix} 1 \\ \frac{1}{2} \theta_b - \alpha_+ \theta_b \\ \frac{1}{2} \theta_b + \alpha_+ \theta_b \end{pmatrix}, \\
    h u^b_-(p_b, \alpha_-) &= \frac{1}{2} c_-(p^0_b) \begin{pmatrix} \frac{1}{2} \theta_b + \alpha_- \theta_b \\ -1 \\ -\frac{1}{2} \theta_b + \alpha_- \theta_b \end{pmatrix}
\end{align*}
\]  
(126)

The eigenstates and eigenvalues of the helicity operator are now given by

\[
\begin{align*}
    h u^b_+ &= h u_+(p_b, 0) = \frac{1}{2} u^b_+ \\
    h u^b_- &= h u_-(p_b, 0) = -\frac{1}{2} u^b_-
\end{align*}
\]  
(127)

To find the values of the normalization factors \( c_+ \) and \( c_- \), we use the completeness identity

\[
\sum_\lambda u_\lambda(p) \bar{u}_\lambda(p) = \hat{p} + m \text{ as a requirement,}
\]

\[
p^0_b = p^0_b \begin{pmatrix} 1 & 0 & -1 & -\theta_b \\ 0 & 1 & 0 & 0 \\ 1 & \theta_b & -1 & 0 \\ \theta_b & -1 & 0 & -1 \end{pmatrix} = u^b_+ \bar{u}^b_+ + u^b_- \bar{u}^b_-
\]

\[
= c_+^2 \begin{pmatrix} 1 \\ \frac{1}{2} \theta_b \\ \frac{1}{2} \theta_b \end{pmatrix} (1, \frac{1}{2} \theta_b, \frac{1}{2} \theta_b) \gamma^0 + c_-^2 \begin{pmatrix} -\frac{1}{2} \theta_b \\ \frac{1}{2} \theta_b \\ -1 \end{pmatrix} (-\frac{1}{2} \theta_b, 1, \frac{1}{2} \theta_b, -1) \gamma^0
\]  
(128)
\[
\epsilon_a \quad \lambda_b \quad \lambda_c \quad F(z; \epsilon_a, \lambda_b, \lambda_c)
\]

<table>
<thead>
<tr>
<th>in (\pm)</th>
<th>(\mp)</th>
<th>((1 - 2z)^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>out (\pm)</td>
<td>(\mp)</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2: Polarization dependence of the branching \(g \to q\bar{q}\).

It is clear that this can only be true when
\[
c_+ = c_- = p_b^0 = \sqrt{E_b}.
\] (129)

Now that we have found the helicity spinors for the quark we can do something similar for the antiquark. We find, consistent with Bjorken and Drell, that to first order in the small angles, the helicity spinors are
\[
u^c_+ = i\sqrt{E_c} \begin{pmatrix}
-\frac{1}{2} \theta_c \\
1 \\
\frac{1}{2} \theta_c
\end{pmatrix},
\quad
\nu^c_- = i\sqrt{E_c} \begin{pmatrix}
-1 \\
\frac{1}{2} \theta_c \\
-1
\end{pmatrix}.
\] (130)

Now choosing the plane of branching as in Fig. 10 to be in the x-z plane, we get, for example, using Eq. (293) and Eq. (108)
\[
-ig \bar{u}_b^\lambda \gamma_\mu \epsilon^\mu_a v^c = i g(u_b^\lambda)^\dagger \gamma_\mu \gamma_0 (u_c^-)^\dagger v^c
\]
\[
= -g \sqrt{E_b E_c} \begin{pmatrix}
1, \frac{1}{2} \theta_b, 1, \frac{1}{2} \theta_b \\
0, 0, 1, 0 \\
0, 1, 0, 0 \\
1, 0, 0, 0
\end{pmatrix} \begin{pmatrix}
-\frac{1}{2} \theta_c \\
1 \\
\frac{1}{2} \theta_c
\end{pmatrix}
\]
\[
= g \sqrt{E_b E_c} \left( \theta_b - \theta_c \right) = g \sqrt{z E_a (1 - z) E_a (1 - z - \theta z)}
\]
\[
= g \sqrt{z(1 - z)(1 - 2z) E_a \theta} = g \sqrt{t(1 - 2z)}
\] (131)

The same can be done for any configuration of polarization and helicity, giving a total of 4 contributions to the amplitude
\[
-ig \bar{u}_b^\lambda \gamma_\mu \epsilon^\mu_a v^c = g \sqrt{t(1 - 2z)}
\]
\[
-ig \bar{u}_b^\lambda \gamma_\mu \epsilon^\mu_a v^c = g \sqrt{t(1 - 2z)}
\]
\[
-ig \bar{u}_b^\lambda \gamma_\mu \epsilon^\mu_a v^c = ig \sqrt{t}
\]
\[
-ig \bar{u}_b^\lambda \gamma_\mu \epsilon^\mu_a v^c = -ig \sqrt{t}.
\] (132)

Together with the \(\frac{1}{t}\) factor from the propagator, we get
\[
|\mathcal{M}_{n+1}|^2 \sim \frac{g^2}{t} T_R F(z; \epsilon_a, \lambda_b, \lambda_c) |\mathcal{M}_n|^2
\] (133)

where \(T_R\) is the color factor \(T_R = \text{Tr} \left( t^A A \right) / 8 = 1/2\), and the non-vanishing parts of \(F(z; \epsilon_a, \lambda_b, \lambda_c)\) are given in Table 2.

We can see from Table 2 that helicity conservation is automatically satisfied, since the quark and antiquark have opposite helicities. In contrary with the \(g \to gg\) case, there
are no $z \to 0$, $z \to 1$ singularities here, since these are associated only with soft gluon emission. Again averaging over the gluon’s polarization and summing over the helicities of the quarks we get
\[
\langle F \rangle = \frac{2(1 - 2z)^2 + 2}{2} = 2 - 4z + 4z^2 = 2 \left[ z^2 + (1 - z)^2 \right],
\]
so
\[
\tilde{P}_{qg} = T_R \langle F \rangle = z^2 + (1 - z)^2 \tag{134}
\]
From Table 2 we can see that for $z = \frac{1}{2}$ the amplitude vanishes when the gluon is polarized in the plane of branching. Apparently there is a strong anti-correlation between the polarization and the plane of branching. If again we write $\epsilon^a_\mu = \epsilon^a_{\text{in}} \cos \phi + \epsilon^a_{\text{out}} \sin \phi$, we get
\[
F_\phi \propto \sum_{\lambda_a, \lambda_c} \left| \cos \phi M(\epsilon^a_{\text{in}}, \lambda_a, \lambda_c) + \sin \phi M(\epsilon^a_{\text{out}}, \lambda_a, \lambda_c) \right|^2 \\
\propto \cos^2 \phi (1 - 2z)^2 + \sin^2 \phi + \sin \phi \cos \phi \{ -i(1 - 2z) + i(1 - 2z) \}
\]
\[
= 1 + 4(2 - z) \cos^2 \phi = 1 + 2(z - 1) \cos 2\phi
\]
\[
= z^2 + (1 - z)^2 - 2z(1 - z) \cos 2\phi. \tag{135}
\]
Finally we will discuss the branching of a (light) quark into a quark and a gluon. As before we will have a $\frac{1}{t}$ from the propagator. The vertex for this process is
\[
V_{qqg} = -igt^A_{\mu} v^\lambda c_\mu v^\lambda u^\lambda a. \tag{136}
\]
Here the polarization of gluon $c$, $\epsilon_c$, can again be considered being either in or out the plane of branching. However, since the polarization must be perpendicular to the momentum, $\epsilon^a_{\text{in}}$ is rotated slightly towards the $z$-direction, giving us for small angels
\[
\epsilon^a_{\text{in}} = (0, \cos \theta_c, 0, \sin \theta_c) = (0, 1, 0, \theta_c). \tag{137}
\]
So we get, for example
\[
-igt^a_{\mu} u^+_b \gamma^\mu u^+_a = igE_a \sqrt{z} \left( 1, \frac{1}{2} \theta_b, 1, \frac{1}{2} \theta_b \right) \left( \begin{array}{cccc}
0 & 0 & \theta_c & 1 \\
0 & 0 & 1 & -\theta_c \\
\theta_c & 1 & 0 & 0 \\
1 & -\theta_c & 0 & 0
\end{array} \right) \left( \begin{array}{c}
1 \\
0
\end{array} \right)
\]
\[
= igE_a \sqrt{z}(\theta_b + 2\theta_c) = igE_a \theta \sqrt{z}(1 + z) = igt \frac{1 + z}{\sqrt{1 - z}}. \tag{138}
\]
Calculating all configurations in this way we get
\[
|M_{n+1}|^2 \sim \frac{4g^2}{t} C_F F(z; \lambda_a, \lambda_b, \epsilon_c)
\]
where $C_F = \text{Tr}(t^A t^A)/3 = 4/3$ is the color factor, and $F(z; \lambda_a, \lambda_b, \epsilon_c)$ is given in Table 3. We see that helicity conservation ensures that the quark does not change its helicity during the branching. Summing over the spin of quark $b$ and polarization of gluon $c$, and averaging over the spin of quark $a$, we get
\[
\langle F \rangle = \frac{1}{2} \left[ \left( \frac{1 + z}{1 - z} \right)^2 + 1 - z \right] = \frac{1}{2} \left( \frac{(1 + z)^2 + (1 - z)^2}{1 - z} \right) = \frac{1 + z^2}{1 - z}. \tag{140}
\]

<table>
<thead>
<tr>
<th>$\lambda_a$</th>
<th>$\lambda_b$</th>
<th>$\epsilon_c$</th>
<th>$F(z; \lambda_a, \lambda_b, \epsilon_c)$</th>
<th>$(1 + z)^2/(1 - z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pm$</td>
<td>$\pm$</td>
<td>in</td>
<td></td>
<td>$(1 + z)^2/(1 - z)$</td>
</tr>
<tr>
<td>$\pm$</td>
<td>$\pm$</td>
<td>out</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For the type of branching discussed in this section, i.e. timelike branching, to go from the process $q \to qg$, we have directly found the $q \to g$ splitting function $P_{gq}$

$$P_{gq}(z) = P_{gq}(1-z) = C_F \cdot \frac{1 + (1-z)^2}{z}$$

(142)

From Table 3 we can see that there is a singularity at $z = 1$ only when the emitted gluon is polarized in the plane of branching. Therefore, we can look at the correlation between the amplitude and the polarization of the emitted gluon. Assuming gluon $c$ is at an angle $\phi$ to the plane of branching, we have

$$F_{\phi} \propto F(z; \lambda_a, \lambda_b, c_{\text{in}}) \cos^2 \phi + F(z; \lambda_a, \lambda_b, c_{\text{out}}) \sin^2 \phi$$

$$\propto \frac{(1+z)^2}{1-z} \cos^2 \phi + (1-z) \sin^2 \phi = (\cos^2 \phi + \sin^2 \phi)(1-z) + \frac{4z}{1-z} \cos^2 \phi$$

$$= \frac{(1-z)^2 + 2z}{1-z} + \frac{2z}{1-z} \cos 2\phi = \frac{1 + z^2}{1-z} + \frac{2z}{1-z} \cos 2\phi$$

(143)

In order to compute cross sections from the amplitude of the different branching types we need to consider the initial state flux $F$ and the $n$-parton phase space $d\Phi_n$

$$d\sigma_n = F |M_n|^2 d\Phi_n.$$  

(144)

For the type of branching discussed in this section, i.e. timelike branching, to go from the process where an outgoing parton $a$ is replaced by partons $b + c$ we have to replace

$$d\Phi_n = \ldots \frac{d^3p_a}{2(2\pi)^3 E_a}$$  

(145)

by

$$d\Phi_{n+1} = \ldots \frac{d^3p_b}{2(2\pi)^3 E_b} \frac{d^3p_c}{2(2\pi)^3 E_c}.$$  

(146)

Now since $p_c = p_a - p_b$ we have that $d^3p_c = d^3p_a$ for fixed $p_b$. Hence we can write

$$d\Phi_{n+1} = \ldots \frac{d^3p_b}{2(2\pi)^3 E_b} \frac{d^3p_c}{2(2\pi)^3 E_c} = \ldots \frac{d^3p_b}{2(2\pi)^3 E_b} \frac{d^3p_a}{2(2\pi)^3 E_a} E_c = \frac{d\Phi_n}{2(2\pi)^3 E_b} \frac{d^3p_a}{2(2\pi)^3 E_a} E_c$$

$$= \frac{d\Phi_n}{2(2\pi)^3} \delta(\theta_b,\theta_c) \delta(\phi_b,\phi_c) \int_0^\infty dt \delta(t - E_b E_c \theta^2) \int_0^1 dz \frac{dz}{1-z} \delta(z - E_b E_c).$$

(147)

Using Eq. (108) to rewrite the delta-function, keeping in mind that we integrate over strictly positive $\theta_b$, thus picking up only one zero of the delta-function, namely $\theta_b = (1-z)\sqrt{\frac{t}{E_b E_c}}$, we get

$$d\Phi_{n+1} = \frac{d\Phi_n}{2(2\pi)^3} \delta t \delta \phi \theta_b \theta_c \frac{(1-z)^2}{2E_b E_c \theta_b} \delta \left( \theta_b - (1-z)\sqrt{\frac{t}{E_b E_c}} \right) \int_0^\infty dt \frac{E_a E_b}{E_b - E_a} \delta(E_b - E_a z)$$

$$= \frac{d\Phi_n}{2(2\pi)^3} \delta t \delta \phi \frac{E_a (1-z)^2}{2E_c (1-z)} = \frac{d\Phi_n}{4(2\pi)^3} \delta t \delta \phi.$$

(148)

Inserting this into Eq. (144), as well as the appropriate function for $M_{n+1}$ we find that

$$d\sigma_{n+1} = \frac{dt}{t} \frac{d\phi}{2\pi} \frac{A g^2}{4(2\pi)^2} C_F \cdot F |M_n| d\Phi_n = d\sigma_n \frac{dt}{t} \frac{d\phi}{2\pi} \frac{\alpha_s}{2\pi} C_F$$

(149)
where $C$ and $F$ are respectively the appropriate color factor and the polarization dependent distribution functions derived in this section. If we don’t care about the $\phi$-dependence we can replace

$$\int \frac{d\phi}{2\pi} CF = \hat{P}_{ba}(z)$$

(150)

where $\hat{P}_{ba}$ is the appropriate splitting function. Then we get

$$d\sigma_{n+1} = d\sigma_n \frac{dt}{t} \frac{dz}{z} \alpha_s \frac{\hat{P}_{ba}(z)}{2\pi}.$$ \hspace{1cm} (151)

Let us now consider the branching of an incoming parton line. The kinematics for this process can be seen in Fig. 11. This branching process will replace the process of that with a parton $b$ incoming with that of parton $a$ incoming and $c$ emitted. Since we calculated the vertices in this section for on-shell partons, the only thing that changes going from timelike to spacelike branching are the kinematics. For this spacelike branching we have for small angles

$$t \equiv \left| p_b^2 \right| = \left| (p_a - p_c)^2 \right| = -2E_a E_c (1 - \cos \theta_c) = E_a E_c \theta_c^2.$$ \hspace{1cm} (152)

To calculate the $n + 1$ phase space $d\Phi_{n+1}$, we just need to include the phase space of the emitted parton $c$. Using that for fixed $p_c$, $dp_a = dp_b$, we have

$$d\Phi_{n+1} = d\Phi_n \frac{d^3 p_c}{2(2\pi)^3 E_c}$$

$$= \frac{d\Phi_n}{2(2\pi)^3} \int E_c dE_c \sin \theta_c d\theta_c d\phi dz dt \delta \left( t - E_a E_c \theta_c^2 \right) \delta \left( z - \frac{E_b}{E_a} \right)$$

$$= \frac{d\Phi_n}{4(2\pi)^3} \int E_c dE_c \delta \left( E_c - (1 - z) E_a \right) d\phi dz dt$$

Hence Eqs. (149) and (151) also hold for spacelike branching.

### 1.6 Coherent Branching

Up to now we have been discussing a parton branching formalism which takes account of collinear enhancements only. But, as we have already seen, there exist enhancements associated with soft
gluon emission. We encountered these singularities in our discussion of the small-angle parton splitting functions, but they are more general. Whenever an external line of a QCD Feynman diagram of momentum $p$ and mass $m$ (not necessarily small) emits a gluon of momentum $q$, the amplitude contains the propagator factor proportional to

$$\frac{1}{(p \pm q)^2 - m^2} = \frac{\pm 1}{2p \cdot q} = \frac{\pm 1}{2\omega E(1 - \frac{p^2}{E} \cos \theta)} = \frac{\pm 1}{2\omega E(1 - v \cos \theta)},$$

where $\omega$ is the energy of the gluon, $E$ and $v$ the energy and velocity of the parton emitting it, and $\theta$ the angle of emission. We can see that for light partons ($v \to 1$) we have the usual collinear divergence, i.e. when $\theta \to 0$. However, we can directly see that for any velocity $v$ and angle $\theta$ we have an enhancement as $\omega \to 0$, i.e. soft gluons.

The full factor in the amplitude for the soft enhancement on an initial state line is a color factor times a spin independent factor of

$$g F_{\text{soft}} u(p) \equiv g \frac{\gamma_\nu q_\mu}{2p \cdot q} u(p) \epsilon^\mu_* = g \frac{\gamma_\nu p_\nu + \gamma^\mu m}{2p \cdot q} u(p) \epsilon^\mu_* = g \frac{2g_\mu^\nu p_\nu - \gamma^\mu \not{p} + \gamma^\nu m}{2p \cdot q} \epsilon^\mu_* = g \frac{\not{p} \cdot \epsilon^*}{p \cdot q} u(p),$$

where in the last step we used the Dirac equation for the on-shell spinor $u(p)$. For the emission of a soft gluon from a final state line we would pick up an extra minus sign. Notice that there is no soft enhancement on any (off-shell) internal lines, since the denominator $(p \pm q)^2 - m^2 \to p^2 - m^2 \neq 0$ as $\omega \to 0$. An enhancement factor in the amplitude for each (single) external line implies that for the cross section we have to sum over all pairs of external lines $\{i, j\}$ to get the correct enhancement factor:

$$d\sigma_{n+1} = \frac{d\omega}{\omega} \frac{d\Omega}{2\pi} \frac{\alpha_s}{2\pi} \sum_{i,j} C_{ij} W_{ij},$$

where $d\Omega$ is the element of solid angle for the emitted gluon, $C_{ij}$ is a color factor to be computed and $W_{ij}$ is the radiation function. The first three terms of the right hand side of Eq. (156) are just the phase space of the gluon times $\frac{2\pi}{\omega^2}$:

$$\frac{g^2}{\omega^2} \frac{dq^3}{(2\pi)^3 2q^0} = \frac{g^2}{\omega^2} \frac{d\omega d\Omega}{2\pi^2 2\omega} = \frac{d\omega d\Omega}{\omega} \frac{\alpha_s}{2\pi}.$$

The radiation function $W_{ij}$ can be derived from the standard calculation of the amplitude squared. We get

$$W_{ij} = \frac{\omega^2 p_i \cdot p_j}{p_i \cdot q p_j \cdot q} = \frac{1 - v_i v_j \cos \theta_{ij}}{(1 - v_i \cos \theta_{iq})(1 - v_j \cos \theta_{jq})},$$

where $v_i$ is defined as in Eq. (154), $\theta_{ij}$ is the angle between line $i$ and $j$ and where the $\omega^2$ comes in to cancel the $\frac{1}{\omega^2}$ in Eq. (157).

The radiation function gives an interference contribution from lines $i$ and $j$. It is, however, possible to separate $W_{ij}$ into two parts containing the leading collinear singularities. If we, for simplicity, take the partons again to be massless, i.e. $v_i = v_j = 1$, we can write

$$W_{ij} = W_{ij}^{[i]} + W_{ij}^{[j]},$$

where

$$W_{ij}^{[i]} = \frac{1}{2} \left( W_{ij} + \frac{1}{1 - \cos \theta_{iq}} - \frac{1}{1 - \cos \theta_{jq}} \right).$$
As we will prove below, the function \( W_{ij}^{[i]} \) has a property known as *angular ordering*. This means that if we would write the angular integration in Eq. (156) in terms of polar and azimuthal angles with respect to the direction of line \( i \) and then carry out the azimuthal integration, we would find that

\[
\int_{0}^{2\pi} \frac{d\phi_{iq}}{2\pi} W_{ij}^{[i]} = \begin{cases} 
\frac{1}{1 - \cos \theta_{iq}} & \text{if } \theta_{iq} < \theta_{ij} \\
0 & \text{otherwise.}
\end{cases}
\]  

(161)

To prove this property, we first show that

\[
1 - \cos \theta_{jq} = a - b \cos \phi_{iq},
\]

(162)

where

\[
a = 1 - \cos \theta_{ij} \cos \theta_{iq}, \quad b = \sin \theta_{ij} \sin \theta_{iq}.
\]

(163)

Let \( \hat{i}, \hat{j} \) and \( \hat{q} \) be the unitary vectors along the \( i \), \( j \) and \( q \) lines. Now suppose we choose spherical coordinates with the polar and azimuthal angle with respect to direction \( \hat{i} \). Then \( \hat{i} = (1, 0, 0) \) together with \( \hat{k} \equiv (1, \frac{\pi}{2}, 0) \) and \( l \equiv (1, \frac{\pi}{2}, \frac{\pi}{2}) \) form an orthonormal Cartesian basis. Then

\[
\cos \theta_{jq} = \hat{j} \cdot \hat{q} = (\hat{j} \cdot \hat{i})(\hat{q} \cdot \hat{i}) + (\hat{j} \cdot \hat{k})(\hat{q} \cdot \hat{k}) + (\hat{j} \cdot \hat{l})(\hat{q} \cdot \hat{l}) = \cos \theta_{ij} \cos \theta_{iq} + \sin \theta_{ij} \cos \phi_{ij} \sin \theta_{iq} \cos \phi_{iq} + \sin \theta_{ij} \sin \phi_{ij} \sin \theta_{iq} \sin \phi_{iq}.
\]

(164)

Since, in the end, we are to do the full azimuthal integration, we can, without loss of generality, set \( \phi_{ij} \) to zero. Eq. (164) then becomes

\[
\cos \theta_{jq} = \cos \theta_{ij} \cos \theta_{iq} + \sin \theta_{ij} \sin \theta_{iq} \cos \phi_{iq}
\]

\[
\Rightarrow 1 - \cos \theta_{jq} = 1 - \cos \theta_{ij} \cos \theta_{iq} - \sin \theta_{ij} \sin \theta_{iq} \cos \phi_{iq} = a - b \cos \phi_{iq}.
\]

(165)

(166)

Defining \( z = \exp(i \phi_{iq}) \), we have

\[
I_{ij}^{[i]} = \int_{0}^{2\pi} \frac{d\phi_{iq}}{2\pi(1 - \cos \theta_{jq})} = \int_{0}^{2\pi} \frac{d\phi_{iq}}{2\pi(a - b \cos \phi_{iq})} = \frac{1}{2\pi} \int_{|z| = 1} \frac{dz}{iz - a - b(\frac{z + z^{-1}}{2})} = \frac{1}{i\pi b} \int_{|z| = 1} \frac{dz}{2a - z - z^{-1}}
\]

(167)

where

\[
z_{\pm} = \frac{a}{b} \pm \sqrt{\frac{a^2}{b^2} - 1}.
\]

(168)

Now since \( a \geq b \), and on the interval \( (0, \pi) \) both \( a \) and \( b \) are positive only the pole at \( z = z_- \) can lie inside the unit circle. To see that \( a \geq b \) we look at the extrema of the function

\[
f(\theta_{ij}, \theta_{iq}) = a - b = 1 - \cos \theta_{ij} \cos \theta_{iq} - \sin \theta_{ij} \sin \theta_{iq}
\]

in an arbitrary direction parametrized by \( t \). Now

\[
0 = \frac{df(\theta_{ij}, \theta_{iq})}{dt} = \frac{df}{d\theta_{ij}} \frac{d\theta_{ij}}{dt} + \frac{df}{d\theta_{iq}} \frac{d\theta_{iq}}{dt} = \left( \frac{d\theta_{ij}}{dt} - \frac{d\theta_{iq}}{dt} \right) \left( \sin \theta_{ij} \cos \theta_{iq} - \cos \theta_{ij} \sin \theta_{iq} \right)
\]

\[
\Rightarrow \tan \theta_{ij} = \tan \theta_{iq} \Rightarrow \theta_{ij} = \theta_{iq},
\]

(170)
since we are looking at the interval \((0, \pi)\). So there is one extremum at \(\theta_{ij} = \theta_{iq}\) where \(\frac{a}{b} = 1\). To see that it is a minimum we differentiate again in the same direction and evaluate it at \(\theta_{ij} = \theta_{iq}\):

\[
\frac{d^2 f(\theta_{ij}, \theta_{iq})}{dt^2} \bigg|_{\theta_{ij}=\theta_{iq}} = \left( \frac{d\theta_{ij}}{dt} - \frac{d\theta_{iq}}{dt} \right)^2 \left( \sin \theta_{ij} \sin \theta_{iq} + \cos \theta_{ij} \cos \theta_{iq} \right) \bigg|_{\theta_{ij}=\theta_{iq}}
\]

\[
= \left( \frac{d\theta_{ij}}{dt} - \frac{d\theta_{iq}}{dt} \right)^2 \bigg|_{\theta_{ij}=\theta_{iq}} \geq 0. \quad (171)
\]

Since the second derivative is positive at its single extremum, it is a minimum.

Now, since the pole at \(z = z_-\) lies in the unit circle we have

\[
I_{ij}^{[i]} = -\frac{2\pi i}{i\pi b} = \frac{2}{b} \frac{1}{\sqrt{a^2 - b^2}} = \frac{1}{\sqrt{a^2 - b^2}}
\]

\[
= \sqrt{1 - 2 \cos \theta_{ij} \cos \theta_{iq} + \cos^2 \theta_{ij} \cos^2 \theta_{iq} - \sin^2 \theta_{ij} \sin^2 \theta_{iq}}
\]

\[
= \frac{1}{\sqrt{\cos^2 \theta_{ij} + \cos^2 \theta_{iq} - 2 \cos \theta_{ij} \cos \theta_{iq}}}
\]

\[
= \frac{1}{\sqrt{(\cos \theta_{ij} - \cos \theta_{iq})^2}} = \frac{1}{|\cos \theta_{ij} - \cos \theta_{iq}|}. \quad (172)
\]

Hence

\[
\int_0^{2\pi} \frac{d\phi_{iq}}{2\pi} W_{ij}^{[i]} = \frac{1}{2} \int_0^{2\pi} \frac{d\phi_{iq}}{2\pi} \left[ \frac{1 - \cos \theta_{ij}}{(1 - \cos \theta_{iq})(1 - \cos \theta_{jq})} + \frac{1}{1 - \cos \theta_{iq}} - \frac{1}{1 - \cos \theta_{jq}} \right]
\]

\[
= \frac{1}{2} \left[ \frac{1}{1 - \cos \theta_{iq}} + I_{ij}^{[i]} \left( \frac{1 - \cos \theta_{ij}}{1 - \cos \theta_{iq}} - 1 \right) \right]
\]

\[
= \frac{1}{2(1 - \cos \theta_{iq})} \left[ 1 + \frac{\cos \theta_{iq} - \cos \theta_{ij}}{|\cos \theta_{iq} - \cos \theta_{ij}|} \right], \quad (173)
\]

and because \(\cos(x)\) is strictly decreasing for \(x \in (0, \pi)\) this gives Eq. (161).
2 Parton shower simulations

The main results of section 1 are the derivation of the DGLAP equations, Eq (100) in section 1.4, together with the calculation of the various splitting functions as done in section 1.5. Using these results, the QCD-improved parton model allows us to easily and accurately calculate single non-loop corrections to known hadron-hadron deep inelastic inclusive observables. However, actual hadron-hadron collisions at TeV scale usually generate showers of extra quarks and gluons. Analytically expanding these observables to all-order corrections is tedious, and it has proven to be more fruitful to compare the statistics of experiments with the statistics of Monte Carlo simulations instead.

Because of the simplicity of the concept of parton branching, see section 1.5, generating a parton shower can be done iteratively: \( 1 \to 2 + 3, \ 2 \to 4 + 5, \ 3 \to 6 + 7 \) and so on. The Markovian nature of this system makes it ideal for numerical simulations. In this section we will first introduce some Monte Carlo techniques before discussing the structure of the parton shower simulations written for this thesis.

2.1 The Veto Algorithm

The veto algorithm treats decay-like problems, which are very common in physics. Here we discuss its application to the parton shower. Suppose we have a variable \( t \) which may be thought of as some kind of time axis along which different events are ordered, and suppose we have a function \( f(t) \) which is the chance that the decay event happens at time \( t \). Since we are discussing decay-like processes this event can only happen if the decay didn’t already take place at a previous time \( t' < t \). The probability that nothing has happened up until time \( t \) is expressed by a function \( N(t) \), also called the Sudakov form factor, and hence the differential chance that a specific decay happens at time \( t \) is given by

\[
P(t) = -\frac{dN}{dt} = f(t)N(t).
\]

From this we can solve

\[
N(t) = N(0) \exp \left\{ - \int_0^t dt' f(t') \right\},
\]

hence, for simplicity setting \( N(0) = 1 \), we have

\[
P(t) = f(t) \exp \left\{ - \int_0^t dt' f(t') \right\}.
\]

\( P(t) \) is the real distribution of events in \( t \). If we now want to select a \( t \) according to this distribution when the event happens we can do that using the standard Monte Carlo procedure

\[
\int_0^t P(t') dt' = N(0) - N(t) = 1 - \exp \left\{ - \int_0^t dt' f(t') \right\}
\]

\[
= R \int_0^\infty P(t') dt' = RN(0) - RN(\infty) = R,
\]

where \( R \) is a random number in the interval \((0, 1)\). Now since \( R \) is a random number between 0 and 1, we might as well replace \( R \to 1 - R \) which gives us

\[
\exp \left\{ - \int_0^t dt' f(t') \right\} = R \implies F(0) - F(t) = \log R
\]

\[
\implies t = F^{-1}(F(0) - \log R).
\]
To test if our algorithm works we can calculate a large number of \( t \)-values and count what fraction \( r \) of them is smaller than some test value \( t_{\text{test}} \). We should find that

\[
 r \approx \int_{t_{\text{test}}}^{t_{\text{test}}} \mathcal{P}(t') dt' = \mathcal{N}(0) - \mathcal{N}(t_{\text{test}}) = 1 - \exp \left\{ - \int_0^{t_{\text{test}}} dt' f(t') \right\} .
\]  

(179)

### 2.1.1 The Standard Veto Algorithm

This method obviously only works when \( f(t) \) has primitive \( F(t) \) with a well defined inverse \( F^{-1} \). If this is not the case, we can try to find a better behaving function \( g(t) \) which does have these properties and for which \( g(t) \geq f(t) \) for all \( t \geq 0 \). If we find such a function we can now use the so-called veto algorithm:

1. start with \( i = 0 \) and \( t_0 = 0 \)
2. add 1 to \( i \) and select \( t_i \) according to distribution \( g(t) \), hence \( t_i = G^{-1}(G(t_{i-1}) - \log R) \), but with the constraint that \( t_i > t_{i-1} \). If the selected \( t_i \) fails this constraint, try again, i.e. pick a new \( R \) (but do not increase \( i \)).
3. compare a (new) random \( R \) with the ratio \( \frac{f(t_i)}{g(t_i)} \). If \( \frac{f(t_i)}{g(t_i)} \leq R \), return to step 2.
4. otherwise \( t_i \) is accepted as the final \( t \).

To see that this gives indeed the correct distribution \( \mathcal{P}(t) \) as in Eq. (176), let us consider the various ways the algorithm can yield a specific time \( t \). The probability that the first try works, i.e. no intermediate \( t \) values need to be rejected, and gives \( t \) is given by the chance to pick \( t \) with distribution \( \mathcal{P}^g(t) \), times the chance that this \( t \) is accepted, or

\[
\mathcal{P}_0(t) = \mathcal{P}^g(t) \frac{f(t)}{g(t)} = \exp \left\{ - \int_0^t dt' g(t') \right\} g(t) \frac{f(t)}{g(t)}
\]

\[
= f(t) \exp \left\{ - \int_0^t dt' g(t') \right\} ,
\]

(180)

where \( \mathcal{P}^g(t) \) is just Eq. (176) applied to \( g(t) \). Now consider the case where one intermediate time \( t_1 \) is rejected and \( t_2 = t \) is accepted. The chance for this happening is

\[
\mathcal{P}^g(t_1) \times \mathcal{P}(t_1 \text{ is rejected}) \times \mathcal{P}^g(t_2) \times \mathcal{P}(t \text{ is accepted}).
\]

(181)

Now \( t_1 \) can have any value between 0 and \( t \), so we have to integrate over \( t_1 \):

\[
\mathcal{P}_1(t) = \int_0^t dt_1 \exp \left\{ - \int_0^{t_1} dt' g(t') \right\} g(t_1) \left[ 1 - \frac{f(t_1)}{g(t_1)} \right] \times 
\]

\[
\exp \left\{ - \int_{t_1}^t dt' g(t') \right\} g(t) \frac{f(t)}{g(t)} .
\]

(182)

Combining the exponentials and working out the square brackets this becomes

\[
\mathcal{P}_1(t) = f(t) \exp \left\{ - \int_0^t dt' g(t') \right\} \int_0^t dt_1 [g(t_1) - f(t_1)]
\]

\[
= \mathcal{P}_0(t) \int_0^t dt_1 [g(t_1) - f(t_1)] .
\]

(183)
This generalizes. If we consider two intermediate times \(0 \leq t_1 \leq t_2 \leq t_3 = t\) we have

\[
\mathcal{P}_2(t) = \mathcal{P}_0(t) \int_0^t dt_1 [g(t_1) - f(t_1)] \int_{t_1}^t dt_2 [g(t_2) - f(t_2)]
\]

\[
= \mathcal{P}_0(t) \int_{0 \leq t_1 \leq t_2 \leq t} dt_1 dt_2 [g(t_1) - f(t_1)] [g(t_2) - f(t_2)]
\]

\[
= \frac{1}{2} \mathcal{P}_0(t) \left[ \int_{0 \leq t_1 \leq t_2 \leq t} dt_1 dt_2 [g(t_1) - f(t_1)] [g(t_2) - f(t_2)] \right]
\]

\[
= \frac{1}{2} \mathcal{P}_0(t) \left[ \int_0^t dt_1 [g(t_1) - f(t_1)] \int_{t_1}^t dt_2 [g(t_2) - f(t_2)] \right]
\]

\[
= \frac{1}{2} \mathcal{P}_0(t) \left( \int_0^t dt' [g(t') - f(t')] \right)^2,
\]

where, in the third line, we have used that changing the region of integration in this manner is just relabeling the indices, which does not change the value of the integral. Also we have used that

\[
\int_{0}^{t_1} dt_1 \int_{t_1}^{t_2} dt_2 = \int_{0}^{t_2} dt_2 \int_{0}^{t} dt_1.
\]

In general now, for \(\mathcal{P}_i\) there are \(i!\) ways to order the \(i\) intermediate times \(t_i\). Hence, the total probability to accept \(t\) in any step is

\[
\mathcal{P}_{\text{any}}(t) = \sum_{i=0}^{\infty} \mathcal{P}_i(t) = \mathcal{P}_0(t) \sum_{i=0}^{\infty} \frac{1}{i!} \left( \int_0^t dt' [g(t') - f(t')] \right)^i
\]

\[
= f(t) \exp \left\{ - \int_0^t dt' g(t') \right\} \exp \left\{ \int_0^t dt' [g(t') - f(t')] \right\}
\]

\[
= f(t) \exp \left\{ - \int_0^t dt' f(t') \right\},
\]

which is the desired distribution \(\mathcal{P}(t)\) as in Eq. (176).
2.1.2 Multivariable Veto algorithm

The veto algorithm as described above obviously fails when \( f(t) \) is an incalculable function. This is often the case when \( f(t) \) represents some (multidimensional) integral, i.e.

\[
f(t) = \int_{z_{i_{\text{min}}}^1}^{z_{i_{\text{max}}}^1} dz_1 \int_{z_{j_{\text{min}}}^2}^{z_{j_{\text{max}}}^2} dz_2 \ldots \int_{z_{n_{\text{min}}}^n}^{z_{n_{\text{max}}}^n} dz_n f(z_1, z_2, \ldots, z_n, t).
\]  

(187)

In this case, besides \( t \), one often wants to generate a complete set \( \{z_1, z_2, \ldots, z_n\} \) where, for a given \( t \) and \( z_1, \ldots, z_{i-1}, z_i \) is distributed according to

\[
P(z_i) = \int_{z_{i_{\text{min}}}^{i+1}}^{z_{i_{\text{max}}}^{i+1}} dz_{i+1} \ldots \int_{z_{n_{\text{min}}}^n}^{z_{n_{\text{max}}}^n} dz_n f(z_1, z_2, \ldots, z_n, t).
\]

(188)

The \( z \)-variables are of a different nature than \( t \); they are not evolution variables of the decay, but merely represent some configuration parameters of the decay event. The integration limits \( z_{i_{\text{min}}}^i \) and \( z_{i_{\text{max}}}^i \) can in general depend on \( t \), or even on \( z_j \) for \( j < i \), but this dependency is left out for readability.

One could of course try to find an overestimation of the \( z \)-integrals and perform the \( z \)-integrals numerically to decide if the selected \( t \) should be accepted or vetoed, but this is in general too time consuming to be implemented in a Monte Carlo simulation, even for \( n = 1 \). The solution is to find an overestimation \( g(z_1, z_2, \ldots, z_n, t) \geq f(z_1, z_2, \ldots, z_n, t) \) of the integrand for which all (sub)integrals are analytically calculable. Then, a \( t = t_0 \) can be selected in the regular way from the distribution \( g(t) \exp \left\{ - \int_0^t dt' g(t') \right\} \), where \( g(t) \) is defined in the same way as \( f(t) \). Furthermore, we also select all \( z_i \), in ascending order of \( i \), according to Eq. (188), for the selected \( t \) and \( z_j \) with \( j < i \). We now accept the set of \( \{t, z_1, \ldots, z_n\} \) when

\[
\frac{f(z_1, z_2, \ldots, z_n, t)}{g(z_1, z_2, \ldots, z_n, t)} \geq R,
\]

(189)

where \( R \) is a random number. When the selected set is not accepted, the set as a whole is rejected, and is again selected, with the restriction that the new \( t_1 > t_0 \). This can be continued until a \( t_i \) has been accepted or exceeds some \( t_{\text{max}} \).

To see that this approach gives the correct distribution in \( t \) we will follow the proof of the standard veto algorithm as shown in the previous section. The probability that the first try works and yields \( t \) is given by the product of the distribution in \( t \) using \( g(t) \) times the probability of accepting \( t \), or

\[
P_0(t) = P^g(t) \times P(t \text{ is accepted}).
\]

(190)

Again, \( P^g(t) \) is just Eq. (176) applied to \( g(t) \)

\[
P^g(t) = g(t) \exp \left\{ - \int_0^t dt' g(t') \right\},
\]

(191)

where

\[
g(t) = \int_{z_{i_{\text{min}}}^1}^{z_{i_{\text{max}}}^1} dz_1 \int_{z_{j_{\text{min}}}^2}^{z_{j_{\text{max}}}^2} dz_2 \ldots \int_{z_{n_{\text{min}}}^n}^{z_{n_{\text{max}}}^n} dz_n g(z_1, z_2, \ldots, z_n, t).
\]

(192)

\( P(t \text{ is accepted}) \) is, however, more complicated than in the standard algorithm:

\[
P(t \text{ is accepted}) = \int dz_1 P(z_1) \int dz_2 P(z_2) \ldots \int dz_n P(z_n) \frac{f(z_1, z_2, \ldots, z_n, t)}{g(z_1, z_2, \ldots, z_n, t)},
\]

(193)

where the (normalized) differential probability of selecting a specific \( z_i \),

\[
P(z_i) = \frac{P(z_i)}{\int dz_i P(z_i)} = \frac{\int \ldots \int dz_{i+1}^{'} \ldots dz_n^{'} g(z_1, \ldots, z_i, z_{i+1}^{’}, \ldots, z_n^{’}, t)}{\int \ldots \int dz_1^{'} \ldots dz_n^{'} g(z_1, \ldots, z_{i-1}^{’}, z_i^{’}, \ldots, z_n^{’}, t)},
\]

(194)
depends on the previously selected \( t \) and \( \{z_1, \ldots, z_{i-1}\} \). Substituting Eq. (194) into Eq. (193) we obtain

\[
P(t \text{ is accepted}) = \int dz_1 \left[ \frac{\int dz'_2 \cdots \int dz'_n g(z_1, z'_2, \ldots, z'_n, t)}{\int dz'_1 \cdots \int dz'_n g(z'_1, \ldots, z'_n, t)} \right] \times \\
\int dz_2 \left[ \frac{\int dz'_3 \cdots \int dz'_n g(z_1, z_2, z'_3, \ldots, z'_n, t)}{\int dz'_2 \cdots \int dz'_n g(z_1, z'_2, \ldots, z'_n, t)} \right] \\
\times \cdots \times \int dz_n \left[ \frac{g(z_1, \ldots, z_n, t)}{\int dz'_n g(z_1, \ldots, z_n-1, z'_n, t)} \right] \times \frac{f(z_1, z_2, \ldots, z_n, t)}{g(z_1, z_2, \ldots, z_n, t)}. \tag{195}
\]

Noticing that the \( i \)-th denominator cancels the \((i-1)\)-th numerator, we are left with

\[
P(t \text{ is accepted}) = \int dz_1 \cdots \int dz_n f(z_1, \ldots, z_n, t) = \frac{f(t)}{g(t)}. \tag{196}
\]

Thus, as in the standard veto algorithm we find that

\[
P_0(t) = P^g(t) \frac{f(t)}{g(t)} = \exp \left\{ - \int_0^t dt' g(t') \right\} g(t) \frac{f(t)}{g(t)} \\
= f(t) \exp \left\{ - \int_0^t dt' g(t') \right\}. \tag{197}
\]

Now using that

\[
P(t_1 \text{ is rejected}) = 1 - P(t_1 \text{ is accepted}) = 1 - \frac{f(t_1)}{g(t_1)}, \tag{198}
\]

the proof that the multivariable veto algorithm gives the correct distribution in \( t \) is identical to that of the standard veto algorithm, as done in the previous section (Eqs. (180)-(186)).

### 2.1.3 Numerical Test

In Fig. 12 we see the distribution of points acquired from the simple decay algorithm and from the veto algorithm plotted against the expected distribution. For both sets of points we used

\[
f(t) = t, \tag{199}
\]

![Figure 12: Randomly collected data points versus the expected curve \( e^{-\frac{1}{2}t^2} \)](image)
so the expected distribution is

\[ P(t) = f(t) \exp \left\{ - \int_0^t dt' f(t') \right\} = te^{-\frac{1}{2}t^2}. \]  

(200)

For the veto algorithm we used

\[ g(t) = t^2 + \frac{1}{2} \geq f(t) \text{ for } t \geq 0. \]  

(201)

Fig. 12, generated by the veto algorithm toy model \(<\text{veto\_toy.c}>\) as printed in section A.4.3, shows us that the results of the veto algorithm and the simple decay algorithm are essentially the same. The veto algorithm however becomes less efficient as the approximate function \(g(t)\) is increasingly different from the original function \(f(t)\). This inefficiency grows linearly with the relative surface between \(g(t)\) and \(f(t)\).

Besides these plots, two more tests of are implemented in the toy-model. First the program counts what fraction \(r\) of the generated \(t\) values are smaller than a random chosen \(t_{\text{test}}\) and compares that with what one would expect:

\[ r = \frac{\int_0^{t_{\text{test}}} dt f(t) \exp \left\{ - \int_0^t dt' f(t') \right\}}{\int_0^\infty dt f(t) \exp \left\{ - \int_0^t dt' f(t') \right\}} = 1 - e^{-\frac{1}{2}t_{\text{test}}^2}. \]  

(202)

The second test is to compare the theoretical standard deviation with the deviation of the measured average from the expected average. The expected average is

\[ \langle t \rangle = \int_0^\infty dt t P(t) = \int_0^\infty dt t^2 e^{-\frac{1}{2}t^2} = \sqrt{\frac{\pi}{2}}. \]  

(203)

The theoretical standard deviation for one \(t\) is

\[ \sigma = \sqrt{\langle t^2 \rangle - \langle t \rangle^2} = \sqrt{\left[ \int_0^\infty dt t^2 P(t) - \frac{\pi}{2} \right]^\frac{1}{2}} \]

\[ = \sqrt{\left[ \int_0^\infty dt t^3 e^{-\frac{1}{2}t^2} - \frac{\pi}{2} \right]^\frac{1}{2}} = \sqrt{2 - \frac{\pi}{2}}. \]  

(204)

Hence for \(N\) random values of \(t\), the standard deviation of the average is

\[ \sigma_N = \sqrt{\frac{2 - \frac{\pi}{2}}{N}} \]  

(205)

which is about \(2.072 \cdot 10^{-3}\) for \(N = 10^5\). Fig 13 shows a typical output of \(<\text{veto\_toy.c}>\). The fractions smaller than \(t_{\text{test}}\) are the same as the theoretical values \(\pm 1 \cdot 10^{-3}\). The deviation from the average is of the same order as the standard deviation.
t_test = 0.406910

Simple decay algorithm for g(t) = t
Simulation: left=0.078950, right=0.921050
Theoretical: left=0.079454, right=0.920546

Expected average: 1.253314,
actual average: 1.250746
Standard deviation: 0.002072,
deviation from average: 0.002568

Veto algorithm for t^2+a > t for t>0
Simulation: left=0.079170, right=0.920830
Theoretical: left=0.079454, right=0.920546

Expected average: 1.253314,
actual average: 1.254957
Standard deviation: 0.002072,
deviation from average: 0.001642

The next simulation gives a list of (t,z) branchings up until it reaches the scale = 4.0
parton is at (0.000000, 1.000000)
parton is at (1.068961, 0.885960)
parton is at (2.417160, 0.623417)
parton is at (2.635963, 0.329665)
parton is at (2.649448, 0.187860)
parton is at (2.738431, 0.172800)
parton is at (2.987053, 0.102602)
parton is at (3.486877, 0.079496)
parton is at (3.795159, 0.041836)
parton is at (3.955178, 0.039052)

Figure 13: Output of the veto toy model program <veto_toy.c>
2.2 Final state shower

This section discusses the theoretical grounds of the final state shower as programmed in `<fsshower.cpp>`. This program generates a parton shower from the annihilation of an electron and a positron via a virtual photon. The photon decays into a quark-antiquark pair and thus initiates the shower. The antiquark is put on mass shell and is not considered in the program. The quark obtains a virtuality $t_1 \leq Q^2$ by using the veto algorithm and, if $t_\text{min} \leq t_1$ emits a gluon.

In this program the gluons are not allowed to branch and are thus put on shell. After emitting a gluon, a new virtual mass $t_2 \leq t_1$ is generated, and the quark can emit another gluon. This continues until the virtuality drops below $t_\text{min}$, in which case the quark is put on shell. Since the evolution variable is the quark’s virtual mass squared, this is a virtuality-ordered shower. Other choices for the ordering parameter are also possible, such as the transverse momentum of the decaying products, or the branching angle, and although the latter automatically takes coherence effects into account, angular ordering can be implemented straightforwardly by vetoing emissions that violate it, as is done in SHERPA [10].

2.2.1 The Sudakov form factor

Together with the splitting variable $z_i$, $t_i$ defines the kinematics of the $i$th branching, up to an azimuthal angle $\phi_i$, which we choose to be isotropically distributed. $z$ gives the energy fraction of the daughter quark with respect to the mother, that is

$$z_i = \frac{E_{q,i+1}}{E_{q,i}}.$$ (206)

This implies that the energy fraction of the daughter gluon with respect to the mother is

$$\frac{E_{g,i+1}}{E_{q,i}} = 1 - z_i.$$ (207)

The differential chance for a parton of type $a$ to branch is given by

$$dP_a = \sum_{b,c} \frac{\alpha_s}{2\pi} P_{a \rightarrow bc}(z) \frac{dt}{t} dz.$$ (208)

To show that this follows from the evolution equations derived in section [1], we will rederive the DGLAP equation, Eq. (100), starting from Eq. (208). In our case, the only partons considered are (massless) quarks which can only branch into a quark and a gluon. Hence

$$dP = \frac{\alpha_s}{2\pi} P_{q \rightarrow qg}(z) \frac{dt}{t} dz,$$ (209)

where

$$P_{q \rightarrow qg}(z) = \frac{1 + z^2}{1 - z}.$$ (210)

Consider now a small increase in scale from $t \rightarrow t + \delta t$, and let $f(x,t)$ be the parton distribution or, in the case of only one parton type, the momentum fraction distribution at the given scale $t$. The net change in the parton distribution $\delta f(x,t)$, when considering only one type of parton, is just the number of partons with momentum fraction $x' = x/z > x$ that branched with branching variable $z$, $\delta f_{\text{in}}(x,t)$, minus the number of partons with momentum fraction $x$ that branched into lower momentum fractions $x' = zx$, $f_{\text{out}}(x,t)$. To find $\delta f_{\text{in}}(x,t)$, we must integrate the branching

\[ \text{(1)} \] Other distributions of $\phi$ may be chosen to account for polarization effects.
propability, Eq. (209), times the parton density, \( f(x', t) \), over all allowed \( x' \) values, i.e. \( x < x' \leq 1 \):

\[
\delta f_{\text{in}}(x, t) = \frac{\delta t}{t} \int_{x}^{1} dx' \int_{0}^{1} dz \frac{\alpha_s}{2\pi} \hat{P}(z) f(x', t) \delta(x - zx')
\]

\[
= \frac{\delta t}{t} \int_{0}^{1} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z, t),
\]

where \( \hat{P}(z) \) is the unregularized splitting function for the chosen parton. The lower limit of the \( z \)-integration can effectively be chosen \( x \) since the parton density for \( z < x \Rightarrow x' > 1 \) vanishes. To find \( f_{\text{out}}(x, t) \), we must instead integrate over all lower momentum fractions \( x' = zx \):

\[
f_{\text{out}}(x, t) = \frac{\delta t}{t} f(x, t) \int_{0}^{x} dx' \int_{0}^{1} dz \frac{\alpha_s}{2\pi} \hat{P}(z) \delta(x' - zx)
\]

\[
= \frac{\delta t}{t} f(x, t) \int_{0}^{1} dz \frac{\alpha_s}{2\pi} \hat{P}(z).
\]

We thus find that the total change in the parton distribution for a small increase in \( t \to t + \delta t \) is

\[
\delta f(x, t) = \delta f_{\text{in}}(x, t) - f_{\text{out}}(x, t) = \frac{\delta t}{t} \int_{0}^{1} dz \frac{\alpha_s}{2\pi} \hat{P}(z) \left[ \frac{1}{z} f(x/z, t) - f(x, t) \right].
\]

Using the ‘plus’ prescription as defined in Eq. (82), we can define the regularized splitting function as

\[
P(z) = \hat{P}(z)_+.
\]

Since the divergence in the unregularized splitting function \( \hat{P}(z) = \frac{1+z^2}{1-z} \) is at \( z = 1 \), and \( \frac{1}{z} f(x/z, t)|_{z=1} = f(x, t) \), in terms of the regularized splitting function, the change in \( f(x, t) \) becomes

\[
\delta f(x, t) = \frac{\delta t}{t} \int_{0}^{1} dz \frac{\alpha_s}{2\pi} P(z) f(x/z, t).
\]

From this we can rederive the Altarelli-Parisi evolution equation, Eq. (100), as

\[
t \frac{\partial}{\partial t} f(x, t) = \int_{x}^{1} \frac{dz}{z} P(z) f(x/z, t).
\]

The function \( f(x, t) \) is usually referred to as the parton distribution function, as explained above, representing the distribution in the parton’s momentum fraction of the hadron it is part of, at scale \( t \). In the final state shower however, the parton line is not (yet) part of a hadron. Since we will not do any hadronization here, we will follow the notation as introduced in section 1.5 where the splitting variable \( z \) is defined as the fraction of the energy the daughter parton obtains from its mother. Defining the first outgoing parton to have \( x_1 = 1, x_i \) then represents the fraction of the total energy the \( i \)-th parton has, \( f(x, t) \) is then more accurately described as the energy fraction distribution at scale \( t \).

Given the differential chance of branching, Eq. (209), following the reasoning of section 2.1 the probability that there is no branching between \( t_0 \) and \( t \), the Sudakov form factor, \( \Delta(t) \), is given by

\[
\Delta(t) = \exp \left\{ -\int_{t_0}^{t} \frac{dt'}{t'} \int_{z_0(t')}^{1} dz \frac{\alpha_s}{2\pi} \hat{P}_{q\to qq}(z) \right\},
\]

\[
\Delta(t) = \exp \left\{ -\int_{t_0}^{t} \frac{dt'}{t'} \int_{z_0(t')}^{1} dz \frac{\alpha_s}{2\pi} \hat{P}_{q\to qq}(z) \right\}.
\]
The cut-off value \( z_0 \) is present to reject divergent unresolvable soft gluon emissions, as well as to represent the phase space for a given \( t \), and is therefore a function of \( t \). We will discuss the precise form of \( z_0(t) \) in section 2.2.3. Using the Sudakov form factor as defined above, and the fact that

\[
\frac{\partial \Delta(t)}{\partial t} = -\frac{1}{t} \int \frac{dz}{2\pi} \frac{\alpha_s}{z} \hat{P}(z) \Delta(t)
\]

we can rewrite Eq. (213) as

\[
\delta f(x,t) = \frac{\delta t}{t} \int \frac{dz}{z} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z,t) + \frac{\delta t}{t} \frac{\partial \Delta(t)}{\partial t} 
\]

\[
\Rightarrow t \frac{\partial}{\partial t} f(x,t) = \int \frac{dz}{z} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z,t) + \frac{f(x,t)}{\Delta(t)} \frac{\partial \Delta(t)}{\partial t}.
\]

This in turn we can use to write

\[
t \frac{\partial}{\partial t} \left( \frac{f(x,t)}{\Delta(t)} \right) = \frac{t}{\Delta} \frac{\partial}{\partial t} \Delta - t f \frac{\partial \Delta}{\partial t} = \int \frac{dz}{z} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z,t) + \frac{\Delta}{\Delta} \frac{t \partial \Delta}{\partial t} - t f \frac{\partial \Delta}{\partial t}
\]

\[
= \frac{1}{\Delta} \int \frac{dz}{z} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z,t).
\]

Eq. (220) is similar in form as the Altarelli-Parisi equation, Eq. (215), but with the regularized splitting function replaced by the unregularized one, and with \( f \) replaced by \( f/\Delta \). The former replacement is especially useful when dealing with computer programs, which in general have problems dealing with plus distributions.

Integrating Eq. (220) from \( t_s \) to \( t \) we obtain an integral equation for \( f(x,t) \) in terms of the structure function at the initial value \( t = t_s \):

\[
\frac{f(x,t)}{\Delta(t_n)} \bigg|_{t_s}^t = \int_{t_s}^t \frac{dt'}{t'} \int \frac{dz}{z} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z,t')
\]

\[
\Rightarrow f(x,t) = \frac{\Delta(t)}{\Delta(t_s)} f(x,t_s) + \int_{t_s}^t \frac{dt' \Delta(t)}{t'} \int \frac{dz}{z} \frac{\alpha_s}{2\pi} \hat{P}(z) f(x/z,t').
\]  

(221)

The first term can be interpreted as the contribution from the partons that did not branch between \( t_s \) and \( t \), where the integrand in the second term represents all partons that had their last branching at scale \( t' \). The factor \( \Delta(t)/\Delta(t') \) represents the probability of evolving from \( t' \) to \( t \) without branching. If we now fill in Eq. (221) for \( f(x/z,t') \) in Eq. (221) for \( f(x,z,t') \), we get

\[
f(x,t) = \frac{\Delta(t)}{\Delta(t_s)} f(x,t_s) + \int_{t_s}^t \frac{dt_1}{t_1} \frac{\Delta(t)}{t_1} \int \frac{dz_1}{z_1} \frac{\alpha_s}{\Delta(t_1)} \hat{P}(z_1)
\]

\[
\times \left[ \frac{\Delta(t_1)}{\Delta(t_s)} f\left( \frac{x}{z_1}, t_s \right) + \int_{t_s}^{t_1} \frac{dt_2}{t_2} \frac{\Delta(t_1)}{t_2} \int \frac{dz_2}{z_2} \frac{\alpha_s}{\Delta(t_2)} \hat{P}(z_2) f\left( \frac{x}{z_1 z_2}, t_2 \right) \right].
\]

(222)

Recursively applying this technique we derive the expression:

\[
f(x,t) = f(x,t_s) \frac{\Delta(t)}{\Delta(t_s)} + \sum_{n=1}^{\infty} \int f(x_n,t_s)
\]

\[
\times \prod_{k=1}^{n} \left[ \frac{dz_k}{z_k} \frac{dt_k}{t_k} \frac{\Delta(t_k)}{\Delta(t_{k+1})} \frac{\alpha_s}{2\pi} \hat{P}(z_k) \right] \frac{\Delta(t)}{\Delta(t_1)}.
\]

(223)
Figure 14: Examples of $x' t'$-paths for $n = 0, 1, 2, 3$. All paths end in the same point $(x, t)$.

where $t_{n+1} = t$ and $x_n = x/z_1 \ldots z_n$. A nice, intuitive interpretation of Eq. (223) is that each term represents all possible paths to evolve from $t_s$ to $t$ with exactly $n \geq 1$ branchings, while the first term corresponds to no branchings. Examples of such paths in $(x, t)$ space are shown in Fig. 14. The quantity in the square brackets gives the probability distribution which allows us to, given $t_{k+1}$ and $x_k$, generate $t_k$ and $x_{k-1} = z_k x_k$. Since the selection of $(t_k, x_{k-1})$ depend only on the previous pair $(t_{k+1}, x_k)$, this selection process is a Markov process, making it ideal for computer simulations. This Markov process allows us to compute the distribution $f(x, t)$ starting from a given distribution $f(x, t_s)$.

The Sudakov form factor expresses the chance that there is no emission between the minimal virtuality $t_0$ and $t$. Since in the timelike final state shower considered here the virtuality is evolved downwards from some maximum virtuality scale $Q^2$, the probability that no branching occurs between $t_1$ and $t_2 < t_1$ is given by

$$
\frac{\Delta (t_1)}{\Delta (t_2)} = \exp \left\{ - \int_{t_2}^{t_1} \frac{dt'}{t'} \frac{\alpha_s}{2\pi} \int_{z_0(t')}^{1-z_0(t')/2} P_{q\rightarrow qg(z)} \right\}
$$

(224)

The generation of a $t$ with downward evolution is similar to the upward case discussed in section 2.1 with a few minor adjustments. If again $f(t)$ is the chance of emission at virtuality $t$, we now have

$$
P(t) = \frac{dN}{dt} = f(t)\dot{N}(t).
$$

(225)

Notice the absence of a minus sign in contrast to the upward case, Eq. (174). To select a $t$
according to the $\mathcal{P}(t)$ distribution we calculate

\[
\int_{t}^{t_{\text{max}}} \mathcal{P}(t') dt' = N(t_{\text{max}}) - N(t) = 1 - \exp \left\{- \int_{t}^{t_{\text{max}}} dt' f(t') \right\}
= R \int_{-\infty}^{t_{\text{max}}} \mathcal{P}(t') dt' = RN(t_{\text{max}}) - RN(-\infty) = R,
\]
where $R$ is a random number in the interval $(0, 1)$. Substituting again $R \rightarrow 1 - R$ yields

\[
\exp \left\{- \int_{t}^{t_{\text{max}}} dt' f(t') \right\} = R \implies F(t) - F(t_{\text{max}}) = \log R \\
\implies t = F^{-1}(F(t_{\text{max}}) + \log R).
\]

The $f(t)$ in our case is given by

\[
\frac{1}{t} \frac{1 - z_{0}(t)}{2\pi} \int_{z_{0}(t)}^{1} P_{q \rightarrow qg}(z) = \frac{1}{t} \frac{1}{2\pi} \left[ 3z_{0}(t) - \frac{3}{2} + \log \left( \frac{1}{z_{0}(t)} - 1 \right) \right],
\]

which, after deriving and substituting $z_{0}(t)$, turns out not to be 'nice' enough to be able to analytically invert its primitive. We therefore use the veto algorithm as described in section 2.1, where we use, as an overestimation of $f(t)$

\[g(t) = 0.07 \sqrt{\frac{t_{\text{max}}}{t \log t}}.\]

The primitive and its inverse are then given by

\[
G(t) = 0.07 \sqrt{t_{\text{max}} \log (t - t_{\text{min}})} ,
\]

\[
G^{-1}(x) = \exp \left( \frac{x^{2}}{2t_{\text{max}}} \right).
\]

After a $t$ has been generated, a $z$ is picked from the interval $(z_{0}(t), 1 - z_{0}(t))$, where the chance of picking a specific $z$ is proportional to $P_{q \rightarrow qg}(z)$. An azimuthal angle $\phi$ is chosen randomly between 0 and $2\pi$, and from there the kinematics can be reconstructed.

### 2.2.2 Kinematics

The final state parton shower is generated in the center of mass rest frame of the incoming $e^{-}e^{+}$ pair. The four momentum of the virtual photon is therefore given by

\[p_{\gamma} = (Q, 0, 0, 0).\]

Since the anti-quark from the $q\bar{q}$ pair, that was created from the virtual photon, is put on mass shell, the four momenta of the quark and anti-quark are

\[p_{q} = (E_{1}, \vec{p}_{1}) ,\]

\[p_{\bar{q}} = (|\vec{p}_{1}|, -\vec{p}_{1}) .\]

\footnote{There are two problems with this overestimation if we would like to change the energy scale. Firstly, it becomes too crude for $t_{\text{max}}$ large, reducing the program’s efficiency, and secondly, it is not dimensionless, in contrast to $f(t)$, and neither is the argument of the logarithm. For the energy scale used here however, the overestimation suffices.}
Figure 15. An example of a final state shower generated by <fsshower.cpp>.

where

\[ E_1 = \sqrt{|\vec{p}_1|^2 + t_1}. \]  

(235)

Solving \( p_\gamma = p_q + p_\bar{q} \) for \( |\vec{p}_1| \) leads to

\[ |\vec{p}_1| = \frac{Q^2 - t}{2Q}, \]  

(236)

\[ E_1 = \frac{Q^2 + t}{2Q}. \]  

(237)

which thus gives the energy and momentum of the first quark \( q_1 \) as a function of \( t \). The first quark is defined to travel in the \( z \)-direction, so no \( \theta_1 \) angle needs to be calculated.

Fig. 15 shows the kinematics of a final state shower. The energy of the \( i \)th quark and gluon are, for \( i > 1 \), given by

\[ E_{q_i} = z_{i-1} E_{i-1}, \]  

(238)

\[ E_{g_i} = (1 - z_{i-1}) E_{i-1}, \]  

(239)

and, after the quark's virtual mass \( t_i \) has been generated we find the corresponding momenta

\[ \|\vec{p}_{q_i}\| = \sqrt{E_{q_i}^2 - t_i}, \]  

(240)

\[ \|\vec{p}_{g_i}\| = E_{g_i}. \]  

(241)

To find the angle \( \theta_i \) between \( q_i \) and \( g_i \) we use

\[ p_{q_{i-1}}^2 = t_{i-1} = (p_{q_i} + p_{g_i})^2 = p_{q_i}^2 + p_{g_i}^2 + 2p_{q_i} \cdot p_{g_i}, \]

\[ = t_i + 2z_{i-1}(1 - z_{i-1})E_{i-1}^2 - 2 \cos \theta \|\vec{p}_{q_i}\| \|\vec{p}_{g_i}\| \]

\[ \implies \cos \theta = \frac{t_i - t_{i-1} + 2z_{i-1}(1 - z_{i-1})E_{i-1}^2}{2\|\vec{p}_{q_i}\| \|\vec{p}_{g_i}\|}. \]  

(242)

To be able to reconstruct the 4-momenta of the quark and gluon, we need to calculate the actual angles \( \theta_{q_i} \) and \( \theta_{g_i} \) between the quark or gluon respectively and the parent quark \( q_{i-1} \) (the angles between the parton and the dashed line in Fig. 15). Defining

\[ \theta_{q_i} = \xi \theta_i, \quad -\theta_{g_i} = (1 - \xi) \theta_i, \]  

(243)
we can use transverse momentum conservation to find
\[
\begin{align*}
\|\vec{p}_q\| \sin \theta_q, &= \|\vec{p}_g\| \sin \theta_g, \\
\Rightarrow 0 &= \|\vec{p}_q\| \sin \xi \theta_i - \|\vec{p}_g\| \sin(1 - \xi) \theta_i \\
&= \|\vec{p}_q\| \sin \xi \theta_i - \|\vec{p}_g\| (\sin \theta_i \cos \xi \theta_i - \sin \xi \theta_i \cos \theta_i) \\
\Rightarrow 0 &= \|\vec{p}_q\| \tan \xi \theta_i - \|\vec{p}_g\| (\sin \theta_i - \tan \xi \theta_i \cos \theta_i) \\
\Rightarrow \xi &= \frac{1}{\theta_i} \arctan \left( \frac{\|\vec{p}_q\| \sin \theta_i}{\|\vec{p}_q\| + \|\vec{p}_g\| \cos \theta_i} \right). \quad (244)
\end{align*}
\]

Every parton’s own frame is defined as the photon’s COM frame, with the z-axis aligned with the parton’s momentum. Hence initially
\[
\vec{p}_i = (0, 0, \|\vec{p}_i\|). \quad (245)
\]

To get the parton’s momentum in the first quark’s frame, we successively rotate the frame from daughter to mother to grandmother, etc. The transformation of a vector \((p_x, p_y, p_z)\) in frame \(i\) to frame \(i-1\) is
\[
\begin{align*}
p'_x &= \cos(\theta_{p_i}) \cos(\phi_i) p_x - \cos(\theta_{p_i}) \sin(\phi_i) p_y + \sin(\theta_{p_i}) \cos(\phi_i) p_z \\
p'_y &= \sin(\phi_i) p_x + \cos(\phi_i) p_y + \sin(\theta_{p_i}) \sin(\phi_i) p_z \\
p'_z &= -\sin(\theta_{p_i}) \cos(\phi_i) p_x + \sin(\theta_{p_i}) \sin(\phi_i) p_y + \cos(\theta_{p_i}) p_z,
\end{align*}
\]
where \(\theta_{p_i}\) stands for either \(\theta_{q_i}\) or \(\theta_{g_i}\). Note that throughout the program we remain in the COM frame of the virtual photon initiating the final state shower. The transformations applied here are for mere rotations, and do not include any boost. Using consecutive transformations to write all momenta in the first quark’s frame, we can now perform a momentum conservation check, and calculate the transverse momentum.

### 2.2.3 Evolution

During the evolution of a quark \(q_i\), its daughter partons are assumed to be on shell. Substituting \(t_i = 0\) in Eq. \((242)\) we now obtain a kinematical constraint on \(z\) by demanding that \(|\cos \theta| \leq 1\).

\[
\begin{align*}
\cos \theta &= \frac{-t_{i-1} + 2z_{i-1}(1 - z_{i-1}) E_{i-1}^2}{2\|\vec{p}_q\| \|\vec{p}_g\|} \Rightarrow \\
z_{i-1}(1 - z_{i-1}) &= \frac{t_{i-1}}{2E_{i-1}^2(\cos \theta - 1)} \\
\cos \theta - 1 &\pm \sqrt{(\cos \theta - 1)(\cos \theta - 1 + 2\frac{t_{i-1}}{E_{i-1}^2})} \\
\Rightarrow z_{i-1} &= \frac{1}{2} \left( 1 \pm \sqrt{1 - 2\frac{t_{i-1}/E_{i-1}^2}{1 - \cos \theta}} \right) \in \left[ z_{0,\text{kin}}, 1 - z_{0,\text{kin}} \right), \quad (247)
\end{align*}
\]

where
\[
\begin{align*}
z_{0,\text{kin}} &= \frac{1}{2} \left( 1 - \sqrt{1 - \frac{t_{i-1}}{E_{i-1}^2}} \right) = \frac{1}{2} \left( 1 - \sqrt{\frac{\|\vec{p}_{i-1}\|}{E_{i-1}^2}} \right).
\end{align*}
\]

This restriction on the mother’s splitting variable \(z\) again represents the allowed phase space and thus has to be implemented in the Sudakov form factor as done in Eq. \((217)\), for the generation of \(t\), such that virtual masses with little phase space available are less likely to be selected than virtual masses with much corresponding phase space. Also, when selecting \(z\) itself, we have to make sure that it does not fall out the allowed region, to prevent unphysical splitting angles from
being generated. The actual cut-off value \( z_0(t) \) chosen is the maximum of the kinetic restriction and the infra-red cut-off

\[
z_0(t) = \max (z_0^{\text{kin}}, z_0^{\text{IR}}),
\]

where \( z_0^{\text{IR}} \), usually depends on the minimum virtuality \( t_0 \) and some minimum required transverse momentum.

The first \( t \) is generated using \( t_{\text{max}} = Q^2 \). After a parent quark has been generated and its daughter quark obtains a non-zero virtual mass (and a splitting value \( z \)), the assumption in the generation of the first quark that its daughters are on shell, was incorrect, which has to be accounted for. This can be done according to Bengtsson and Sjostrand [11] by redefining the splitting variable \( z \) as

\[
\tilde{z}_i = \left( z_i - \frac{1}{2} \right) \frac{\lambda(t_{q_i}, t_{g_{i+1}}, t_{g_{i+1}})}{t_{q_i}} + \frac{t_{q_i} + t_{g_{i+1}} - t_{g_{i+1}}}{2t_{q_i}},
\]

where

\[
\lambda(a, b, c) = \sqrt{(a - b - c)^2 - 4bc}.
\]

The parent’s new, transformed \( \tilde{z} \) automatically falls in the parent’s new allowed \( z \)-region. However, since the daughter’s energy has been changed, with it, also the allowed \( z \)-region changes, so we have to check if the generated \( z \) is still allowed. If this is not the case, we reject the branching, and select a new, smaller \( t \), using the original energy. Otherwise the branching at the selected \( t \) and \( z \) is accepted, and the kinematics can be reconstructed, and rotated back to the initial frame. If the virtual mass is larger than \( t_0 \), a new quark (and on-shell gluon) is created with maximum virtuality

\[
t_{\text{max}} = \min ((zE)^2, t).
\]

If not, the quark is put on shell, and the shower is finished.

### 2.2.4 Results

The parton shower created in this section is very much simplified, since we allow only one type of branching, namely \( q \rightarrow q + g \), and the running coupling constant \( \alpha_s \) is chosen to be fixed (by default \( \alpha_s = 0.2 \)). Hence, there is no point in comparing generated showers with real data. The purpose of this shower was, not so much to simulate the actual physics, since there are numerous excellent parton showers available such as implemented in PYTHIA, Sherpa or HERWIG, but mainly to construct the correct structure and behavior of a parton shower simulation in order to be able to study possible enhancements not implemented in the present-day parton showers, such as the effects of threshold resummation (see section 4).

Fig. 16 shows the normalized transverse momentum squared distribution of the emitted gluons for \( Q^2 = 100 \text{ GeV}^2 \), \( \alpha_s = 0.2 \) and a sample size of \( 10^5 \) showers. The \( z \)-axis, i.e. the axis to which the transverse momentum is transverse, is directed along the momentum of the initial quark. As expected, gluons with little transverse momentum, i.e. soft collinear gluons, are favored greatly over gluons with much transverse momentum, i.e. hard emissions. Furthermore we know that the transverse momentum is maximal when

\[
p_{q_i} = (|\vec{p}_{T}|, -\vec{p}_{T})
\]

\[
p_{g_i} = (|\vec{p}_{T}|, \vec{p}_{T})
\]

such that

\[
(p_{q_i} + p_{g_i})^2 = 4 |p_T|^2 = t_{i-1} < Q^2
\]

\[
\Rightarrow |p_T|^2 < \frac{1}{4} Q^2,
\]

which is clearly seen in Fig. 16.
Figure 16: The normalized distribution in transverse momentum of the emitted gluons. The fluctuations on the right-hand side of the plot are due to the low statistics at high $p_T^2$. Note that the upper limit of $p_T^2$ is indeed $\frac{1}{4}Q^2 = 25$.

Figure 17: The number of gluons emitted with transverse momentum squared smaller or equal to $p_T^2$, for $10^5$ showers and different values of $Q^2$. 
Fig. 17 shows, cumulatively, the distribution in transverse momentum of the emitted gluons for different values of $Q^2$, with $\alpha_s = 0.2$. Each curve represents $10^5$ showers. Clearly, and as expected, a higher value of $Q^2$ yields more emitted gluons. Since the total amount of transverse momentum available to the shower is proportional to the virtuality of the first quark, a higher $Q^2$ allows higher virtualities generating more available transverse momentum. We would thus expect more hard gluons as $Q^2$ increases which is indeed the case. We would also expect more soft collinear gluons, since after the first branching, the quark takes on a virtual mass $t_2 < t_1$ which is equivalent to generating a new shower with $Q^2 = t_1$. This obviously generates the same amount of soft collinear gluons as a shower that started at a hard scattering scale $Q^2$. In addition, the gluon emitted at $t = Q^2$ can also be soft and collinear, thus strictly increasing the number of soft collinear gluons when increasing the hard scattering scale from $Q^2$ to $Q^2$. As it turns out, for higher $Q^2$ the distribution gives relatively even more weight to the small transverse momentum region than for lower $Q^2$, as can be seen in Fig. 17 from the increasing steepness of the graphs at $p_T^2 = 0$.

Fig. 18 shows the cumulative transverse momentum distributions of the emitted gluons for different values of $\alpha_s$, with $Q^2 = 1000 \text{ GeV}^2$. Again we see, as is to be expected, that the number of gluons emitted increases when we increase $\alpha_s$. This number is, however, not proportional to $\alpha_s$, but rather it seems to increase logarithmically with $\alpha_s$.

Fig. 19 shows the ratio of the number of emissions at $\alpha_s = 0.4$ over the number of emissions at $\alpha_s = 0.2$ as a function of the transverse momentum, for $Q^2 = 1000 \text{ GeV}^2$. As can be clearly seen, the number of hard emissions are more or less proportional with $\alpha_s$, whereas the soft emissions also increase with $\alpha_s$ but less than proportionally. To understand this we must realize that almost all of the hard emissions are the first emissions in the shower. This is because for a gluon to have high transverse momentum, the parent quark must have a large virtual mass. Now for the second gluon to be the hard emission, both the first and the second quark should have high virtual masses, since the virtual mass $t$ is strictly decreasing. But since the Sudakov form factor suppresses high virtual masses, as can be seen from Fig. 16, the probability that both the first and the second
quark have large $t$ is very small.

The transverse momentum of an emitted gluon is roughly proportional to the difference in virtual mass of its parent quark and its sister quark as can be seen from Eq. (242). When a hard gluon is emitted, this means that the shower evolution has taken a big step in virtuality $\Delta t$. In this same part of $t$-space multiple soft gluons could have been emitted. This means that when the number of hard emissions increases, the number of soft gluons decreases. This is however somewhat compensated by the increased number of soft emissions at low $t$, because of the increase in $\alpha_s$, resulting in a net increase, be it less than proportional, of the number of soft emissions.

2.3 Initial state shower

For the initial state shower we consider the spacelike branching off two incoming parton lines in a Drell-Yan process, starting at virtualities near their mass-shell, up to the hard scattering scale (see Fig. 20). Eq. (223) describes the forward evolution Markov chain of one such parton line, starting from $t_s$ up to $t$. Both the theory and the implementation of the initial state shower are more complicated than the final state shower. The main difference leading to this complication is that in the final state shower the starting configuration of the shower is known exactly and we fully inclusively accept all generated end products, whereas in the initial state shower, apart from extra jets, we require a very specific end product, namely the incoming lines in the hard scattering process, with given four-momenta. This makes forward evolution, as is done in the final state shower, very inefficient for the initial state shower [18], since we would have to reject all generated showers that do not yield the required end products.

There are several additional problems with the principle of forward evolution for the initial state shower. First of all, the hard scale $Q^2$, e.g. the COM energy of the produced leptons in Fig. 20, is determined by the $2 \to 2$ hard process where the partons collide. However, this process cannot be constructed until the parton lines have been evolved up to the assumed value of $Q^2$. Forward evolution of initial state showering often results in a final $\hat{s}$ smaller than (and thus inconsistent with) the value of $Q^2$. This $\hat{s} < Q^2$ problem leads to inefficient code, e.g. [12].
or requires some sort of pretabulation scheme prior to actual event generation as is, for example, done in [13].

A second problem with a forward evolution scheme for initial state showers is that, in order to get a valid representation of the parton distributions throughout the shower, at each branching both daughter partons should be candidates for being part of the spacelike line leading to the hard scattering. In the forward evolution algorithm, however, we need to choose which parton is going to be spacelike right away, in order to proceed to the next step in the algorithm.

The lack of information about the kinematics of the hard scattering until after the shower is evolved, together with the inability to determine if a parton is spacelike or timelike, makes it impossible to enforce precise kinematic constraints during shower generation, and one would have to use approximate physical regions [13]. This absence of exact constraints on the kinematics is considered the most serious shortcoming of the forward evolution formalism for initial state showering [18].

To avoid the problems associated with forward evolution of the initial state showering, we choose instead a backwards evolution scheme. In the backwards evolution scheme, the reconstruction is started with the quarks that partook in the hard scattering, at the $Q^2$ scale, and is then evolved backwards to the quarks where the final quarks originated from, with smaller $t$, and so on, all the way back to the parton that initiated the shower, at $t < t_0$, where $t = -m^2$ with $m$ the virtual mass. Where in the forward evolution scheme the correct distribution of partons is generated automatically via the splitting functions, in the backward evolution scheme the parton distribution functions (PDF’s) have to be implemented manually. For this goal we use the MSTW 2008 PDF’s [13].

2.3.1 Evolution

This section describes the initial state parton shower as programmed in <isshower.cpp>. The program simulates initial state gluon emissions off the two quarks that take part in the hard scattering of a Drell-Yan process (see Fig. 20). As was the case with the final state shower, the emitted gluons are taken to be on shell for the sake of simplicity, although we do calculate the maximum virtual mass available to the gluons to initiate timelike final state showers. The DGLAP equations Eq. [101] express that, during a small increase $dt$, a parton $a$ with momentum fraction $x’$ can resolve in a parton $b$ with momentum fraction $x = z x’$, and a parton $c$ with momentum fraction $x’ - x = (1 - z)x’$. Correspondingly, for backwards evolution, a parton $b$ can unresolve
into a parton \(a\) during a small decrease \(dt\). The relative probability \(dP_b\) for this to happen is \(df_b/f_b\) where \(f_b\) denotes the relevant PDF. Using the DGLAP equations Eq. (101) we find

\[
dP_b = \frac{df_b(x,t)}{f_b(x,t)} = \frac{dt}{t} \frac{\alpha_s}{2\pi} \int \frac{dx' f_a(x',t)}{x' f_b(x,t)} P_{a \rightarrow bc}\left(\frac{x}{x'}\right). \tag{254}\]

To construct the relevant Sudakov form factor, we sum the effect of many such small changes \(dt\), which exponentiates \([15]\), and find

\[
\Delta(x, t_{\text{max}}, t) = \exp\left\{-\alpha_s \frac{t_{\text{max}}}{2\pi} \int \frac{dt'}{t'} \int \frac{dx' f_a(x', t')}{x' f_b(x, t')} P_{a \rightarrow bc}\left(\frac{x}{x'}\right)\right\} = \exp\left\{-\alpha_s \frac{t_{\text{max}}}{2\pi} \int \frac{dt'}{t'} \int \frac{dz f_a(x/z, t')}{z f_b(x, t')} P_{q 
\to q'}(z)\right\}. \tag{255}\]

In contrast to forward evolution scheme, a new problem arises in the fact that the PDF’s, present in the form factor, are in general not known in closed form, and hence it is not possible to perform the \(z\)-integral analytically. Since numerical integration is too time consuming we use the multivariable veto algorithm as described in section 2.1.2. To find an overestimation of the \(z\)-integrand, we use that the function \(\sqrt{x' f_a(x', t')}\) in practice is a decreasing function of \(x' [16]\), for any parton distribution function \(f_a\), including \(u(x', t')\). Now since \(x' > x\),

\[
\frac{1}{z} \frac{u(x', t')}{u(x, t')} \leq \frac{1}{z} \frac{u(x', t')}{u(x, t')} \sqrt{\frac{x}{x'}} = \frac{1}{\sqrt{z}}. \tag{260}\]
Combining this result with
\[
P_{q\to qg}(z) = \frac{1 + z^2}{1 - z} \leq \frac{1 + z_{\text{max}}^2}{1 - z},
\]
we get, as an overestimation of the $z$-integrand,
\[
\frac{1 + z_{\text{max}}^2}{\sqrt{2}(1 - z)}.
\]
(262)

Since the splitting function diverges at $z = 1$, the soft gluon singularity, we have to introduce a upper cut-off value $z_{\text{max}} = x/(x + x_e)$, where $x_e$ is a small number, chosen such that the energy of the gluon is larger than $E_g^{\text{min}} = 2$ GeV when expressed the COM frame of the two quarks that partook in the hard scattering. The gluon energy, when assumed to be on shell, in the COM frame of the two colliding beams is
\[
E_g = |\vec{p}_g| = x_g|\vec{p}_p| = x_g\sqrt{s}/2 \geq x_e\sqrt{s}/2 = E_g^{\text{min}}/\gamma = 2\text{GeV}/\gamma,
\]
(263)
from which we solve
\[
z_{\text{max}} = \frac{x}{x + x_e} = \frac{x}{x + 4\text{GeV}/\gamma\sqrt{s}},
\]
(264)
where $\gamma$ is the Lorentz boost factor of the hard scattering. Whenever we switch from evaluating a parton to evaluating its mother parton, the system is boosted and rotated, as will be explained in the next section. In order to find the correct cut-off $z_{\text{max}}$, as well as to be able to reconstruct the shower, these rotations and boosts are stored in a $4 \times 4$ Lorentz transformation matrix, which at any time in the evolution process gives the Lorentz transformation to boost and rotate the system back to the photon’s COM frame and is essentially the inverse Lorentz transformation of all previously performed transformations. In this way the boost factor $\gamma$ is easily extracted to calculate $z_{\text{max}}$.

Since the cut-off $z_{\text{max}}$ does not depend on $t$, and has a relatively easy form, the integration is straightforward and can be done analytically. The upper limit was discussed above, and the lower limit follows straightforwardly from the upper limit of the momentum fraction of the parent parton
\[
x' \leq 1 \implies z = \frac{x}{x'} \geq x.
\]
(265)
The $z$-integral is now given by
\[
A = \int_{x}^{x_{\text{max}}} dz \frac{1 + z_{\text{max}}^2}{\sqrt{2}(1 - z)} = (1 + z_{\text{max}}) \left(\tanh^{-1}\left(\sqrt{z_{\text{max}}}\right) - \tanh^{-1}\left(\sqrt{x}\right)\right),
\]
(266)
and hence we find as an overestimation for the $t$ integrand
\[
g(t) = \frac{1}{t} \frac{\alpha_s}{2\pi} A,
\]
(267)
whose primitive, $G(t) = A \log t$, is invertible
\[
G^{-1}(y) = e^y.
\]
(268)
According to the multivariable veto algorithm (see section 2.1.2), we now select a $t < t_{\text{max}}$
\[
t = G^{-1}(\log R + G(t_{\text{max}})),
\]
(269)
and using that $t$, we select $z$ such that
\[
\int_{x}^{z} dz' \frac{1 + z_{\text{max}}^2}{\sqrt{2}(1 - z')} = R'A \implies z = \tanh^2\left[(1 - R')\tanh^{-1}\left(\sqrt{x}\right) + R'\tanh^{-1}\left(\sqrt{z_{\text{max}}}\right)\right],
\]
(270)
with $R'$ another random number between 0 and 1. We accept the selected set $(t, z)$ with a probability equal to the ratio of the real $z$-integrand to the overestimation of the $z$-integrand, i.e. we accept if

$$\frac{1 + z^2}{1 + z_{\text{max}}^2} \frac{\sqrt{x} u(x/z, t)}{u(x, t)} \geq R'',$$

where $R''$ is yet another random number between 0 and 1. If the selected set $(t, z)$ is rejected, we set $t_{\text{max}} = t$ and generate a new set $(t, z)$ until we either accept a set, which is then followed by the generation of another parton, or a selected $t$ drops below the minimal virtuality $t_0$, in which case the parton is put on mass shell, and the shower is terminated.

### 2.3.2 Kinematics

From the previous section, we have seen that we are able to construct $t$’s and $x$’s in an initial state shower from the hard scattering down to the cut-off virtuality $t_0$, using backwards evolution. Essential, however, to the reconstruction of the kinematics of the initial state shower is the understanding of the momentum fraction $x$. It turns out that the kinematical interpretation of this variable is not unique, since, especially close to the hard scattering, i.e. at large $t$, the partons can pick up a lot of momentum transverse to the proton beam. One possible choice is to use lightcone variables, and interpret $x$ as the $E + p_z$ fraction for the parton line developing along the $+z$ direction, and the $E - p_z$ fraction for the parton line along the $-z$ direction. The advantage of this choice is that both lines can be evaluated separately, potentially simplifying the structure of the code. However, this approach has some severe impracticalities [16], chiefly the fact that the virtualities would not be strictly ordered, and the problem that we would not know the hard scattering cross section until the showers are completed evolved. Alternatively, one may choose to require that, at any stage of evolution, we have $\hat{s} = x_1 x_2 \hat{s}$, where $\hat{s}$ is the center of mass energy of the shower so far generated, $s$ being the center of mass energy of the two incoming protons, and $x_1$ and $x_2$ the ‘momentum fraction’ of respectively the first and the second line currently being evolved. The result of this requirement is that at a branching with the splitting variable $z$, the total $\hat{s}$ has to be increased by a factor of $1/z$. It has been shown that in first-order QED [17], as well as in some simple QCD toy models [18] this so-called ‘$\hat{s}$ approach’ is the best choice. The kinematical reconstruction of the shower goes along the lines done in [20] for PYTHIA, with the simplification that the emitted gluons are not allowed to branch and are put on shell.

For a complete reconstruction of the kinematics we start with the hard scattering for which $\hat{s}$ has been chosen. Using the veto algorithm we generate the virtualities $t_1 = -m_1^2$ and $t_{\bar{q}_1} = -m_{\bar{q}_1}^2$ of respectively the quark and anti-quark that are involved in the hard scattering. In the rest of this section we replace $q_1$ and $\bar{q}_1$ by 1 and 2 respectively, for readability. We start off with the quark and the antiquark in their common COM frame, with the quark’s momentum directed along the positive, and the antiquark’s momentum directed along the negative $z$-axis, $\hat{p}_1 = -\hat{p}_2$. We then find for the four momenta of the quark-antiquark pair

$$\hat{s} = (p_1 + p_2)^2 = \left(\sqrt{p_1^2 - t_1} + \sqrt{p_2^2 - t_2}\right)^2$$

$$= 2 \left(p_1^4 - (t_1 + t_2) + 2 \sqrt{p_1^2 - t_1} \sqrt{p_2^2 - t_2}\right) \Rightarrow$$

$$\left(\hat{s} - 2 p_1^2 + t_1 + t_2\right)^2 = 4 p_1^4 - p_1^2 \left(t_1 + t_2\right) + t_1 t_2 \Rightarrow$$

$$p_1^4 = \hat{p}_1^2 = \sqrt{\left(t_1 + t_2\right)^2 - 4t_1 t_2}, \quad \left(272\right)$$

$$E_{q_1, \bar{q}_1} = \frac{\hat{s} \pm \left(t_2 - t_1\right)}{2\sqrt{\hat{s}}}, \quad \left(273\right)$$

\(^{3}\)A problem with this $\hat{s}$ interpretation however is that it is not quite equivalent with an $x$ definition of parton densities [19], or any other standard definition. The effects of this mismatch should however not be large [20].
where, in Eq. 273 we used that $E_3^2 = p_1^0^2 - t_1$, and $E_4^2 = p_1^0^2 - t_2$.

With Eqs. 272, 273 the kinematics of the subsystem consisting of $q_1$ and $\bar{q}_1$ are fully constructed. The next step in the backwards evolution scheme is to construct the kinematics of the branching closest to the hard scattering, i.e. to evolve the parton line with the largest virtuality. If, say, $t_1 > t_2$ then the branching that produced the quark $q_1$ is the closest to the hard scattering and is thus to be reconstructed first. If the quark 1 originated from a quark 3 which branched into 1 and a gluon 4, $3 \rightarrow 1 + 4$, we have four kinematical degrees of freedom, because, if $p_3$ is known, $p_4 = p_3 - p_1$ is given. One degree of freedom is just an arbitrary azimuthal angle around the $z$-axis which we will pick randomly from the interval $(0, 2\pi)$. Along with the generation of $t_1$, the splitting variable $z$ has been chosen, which eliminates another degree of freedom by posing the constraint

$$ (p_3 + p_2)^2 = \frac{\hat{s}}{z}, $$

where $\hat{s}$ is the invariant mass of the $1 - 2$ subsystem. The virtuality $t_3$ of parton 3, which is selected by the backwards evolution scheme, eliminates a third degree of freedom.

One final degree of freedom remains, and is related to the possibility of the gluon 4 to acquire a positive virtual mass and start a time-like parton shower. The maximum allowed squared mass $m_{\text{max}, 4}^2$ is found for collinear branching, $3 \rightarrow 1 + 4$. Defining

$$ q_1 = \frac{\hat{s} + t_2 + t_1}{z}, $$
$$ q_3 = \frac{\hat{s}}{z} + t_2 + t_3, $$
$$ r_1 = \sqrt{q_1^2 - 4t_2t_1}, $$
$$ r_3 = \sqrt{q_3^2 - 4t_2t_3}, $$

one finds [16]

$$ m_{\text{max}, 4}^2 = \frac{q_1q_3 - r_1r_3}{2t_2} - t_1 - t_3, $$

which, for the special case that $t_2 = 0$, reduces to

$$ m_{\text{max}, 4}^2 = \left\{ \frac{t_1}{z} - t_3 \right\} \left\{ \frac{\hat{s}}{\hat{s} + t_1} - \frac{\hat{s}}{\hat{s} + t_3} \right\}. $$

Using forward evolution the mass $m_4 \leq m_{\text{max}, 4}$ may be found, which dictates completely the kinematics. In our program we do not allow the gluon to branch, and thus, after calculating $m_{\text{max}, 4}$, we put the gluon on shell. We then find [16]

$$ E_3 = \frac{1}{2\sqrt{s}} \left( \frac{\hat{s} + t_2 - t_1}{z} \right), $$
$$ p_3^z = \frac{1}{2p_1^+} (q_3 - 2E_2E_3), $$
$$ p_3^+ = m_{\text{max}, 4} \frac{(s_1s_3 + r_1r_3)/2 - t_2(t_1 + t_3)}{r_1^2}. $$

The requirement that gluon’s maximum available virtual mass $m_{\text{max}, 4}$ is positive, puts an additional constraint on the allowed $z$-values. This constraint cannot be implemented in the selection process of $t_1$ where it logically belongs, since it also depends on $t_2$ and $t_3$, which is yet to be generated at this point. When a disallowed $z$ is generated, which happens in about 10%-40% of all events, depending on $Q^2$ and $\alpha_s$ it is almost always the $z$-value of the first branching (closest to the hard scattering), $z_1$. Therefore, a reasonably efficient way of dealing with this problem is to completely redo any shower that includes a disallowed $z$.

The $3 \rightarrow 1 + 4$ vertex is now completely reconstructed, and to continue, we boost the system to the COM frame of partons 3 and 2. To reconstruct the next vertex, either the one that produces
Figure 21: The normalized distribution in transverse momentum of the emitted gluons. Note that in contrary to the final state shower (see Fig. 16), the initial state shower does not have a clearly defined upper limit on the transverse momenta of the gluons since it also scales with the beam energy $S$.

parton 3, or the one that produces parton 2, the $3-2$ subsystem will play the role of the $1-2$ system in the previous reconstruction. The internal structure of the vertex $3 \rightarrow 1 + 4$ will appear nowhere in the continued description, but has become ‘unresolved’, i.e. none of the components of the four-vectors $p_1$ and $p_4$ will be used in any further calculations. All the relevant information of $1$ and $4$ is stored in $3$, which is just their sum.

2.3.3 Results

In this section we will discuss some results and distributions of the emitted gluons generated by the initial state shower. As was the case for the final state shower, the initial state shower is very much simplified. We only consider up and anti-up quarks emitting gluons while evolving towards the hard scattering, and no other quark flavors are allowed. We also again choose the coupling constant $\alpha_s$ to be fixed instead of running. Because of these simplifications, it again won’t be useful to compare the results with actual measurements, and we will limit ourselves to discussing the general behavior of the shower. Unless stated differently, the parameters with which the plots are generated:

\[
\begin{aligned}
\alpha_s &= 0.2, \\
\text{s}_{\text{tot}} &= 10^8 \text{ GeV}^2, \\
Q^2 (\text{or } \hat{s}) &= 10^4 \text{ GeV}^2, \\
\# \text{ of showers} &= 10^5,
\end{aligned}
\]

and the quarks from the colliding proton and antiproton are up, respectively anti-up quarks, which we represent using the NLO fit in the MSTW 2008 PDF’s [14].

Fig. 21 shows the transverse momentum distribution of the emitted gluons for a beam COM energy $s_{\text{tot}} = 10^8 \text{ GeV}^2$ and a COM energy of the two colliding quarks $\hat{s} = 10^4 \text{ GeV}^2$. The $z$-
Figure 22: Normalized distribution in momentum fraction of the first quarks in the shower for different values of $\alpha_s$. Direction is the direction of the proton-antiproton beam in their common COM frame. As can be clearly seen, soft collinear gluons are favored greatly over hard gluons, as was the case with the final state shower. This is due to the same Sudakov suppression we had in the final state shower, combined with the suppression of high momentum fractions $x$ by the pdf’s that appear in the Sudakov form factor, since a hard gluon requires a parent quark with large momentum, i.e. large $x$. However, since the kinematics are quite different from the kinematics in the final state shower, the initial state shower does not have the same upper limit on the transverse momenta as the final state shower (Eq. (253)), and consecutive branchings can build up a transverse momentum larger even than the hard scattering scale, which can be seen from Eq. (280) and in Fig. 21.

Fig. 22 shows the normalized momentum fraction distribution of the initial quarks in the shower, at the scale $t = t_{\text{min}} = 1 \text{GeV}^2$, for different values of $\alpha_s$. Effectively, this is the momentum fraction distribution one would get when evolving a delta-like distribution $\delta(x)$ at scale $t_{\text{max}}$ down to scale $t_{\text{min}}$ using the DGLAP equations. Alternatively this can be considered as the distribution at scale $t_{\text{min}}$ in the initial quarks that, when evolved up to scale $t_{\text{max}}$ would yield the correct momentum fraction $x_f = \sqrt{\frac{2}{S_{\text{tot}}}}$. Note that this distribution does not have the same shape as the parton distribution function for the up-quark at scale $t_{\text{min}}$ since, although the probability that the momentum fraction of the first quark $x_1$ equals $x$ is proportional to the parton distribution function $u(x)$, it is also weighted with the probability that, after showering, it evolves in a parton with momentum fraction $x_f$.

As discussed above, quarks with a large momentum fraction are heavily suppressed due to the properties of the implemented parton distribution functions. Since the momentum fractions of the quarks that partook in the hard scattering are fixed at $x_1 = x_2 = \sqrt{\frac{2Q}{S_{\text{tot}}}} = 10^{-2}$, the distribution peaks at $\log(x) = -2$. This peak actually represents the quark lines that did not branch at all during their evolution towards the hard scattering. Just to the right of the peak there is a narrow gap in the distribution that is caused by the cutoff value of the gluon COM energy. For the final quark to backwards evolve to an initial quark whose momentum fraction is only so little more
than the final quark, it has to emit a gluon with a C.O.M energy below the cutoff-value of 2 GeV.

As can be seen from Fig. 22 higher values of $\alpha_s$, on average, generate larger initial momentum fractions. Increasing the coupling constant will increase the total number of emissions and, since on each (backwards evolved) branching the momentum fraction becomes larger, the momentum fraction of the initial quarks will on average be larger. Increasing $\alpha_s$, however, will also shift the normalized $p_T^2$ distribution even more to the soft collinear region, so as to slow down the increase in momentum fraction, since large $x$ are suppressed by the Sudakov form factor. This effect can be seen in Fig. 23, where we have plotted the ratio of the $p_T^2$ distributions for $\alpha_s = 0.4$ versus $\alpha_s = 0.2$. This behavior is quite different from the behavior of the final state shower, Fig. 19.

In contrast to the initial state shower, for the final state shower to emit a hard gluon, the loss in virtual mass of the quark line due to the emission of the gluon, is about the same size as the $p_T^2$ of the emitted gluon, where in the initial state shower, these two quantities are not that directly correlated. Hence, emission of a hard gluon in the initial state shower can still leave enough virtuality for more (soft) branchings. As can be seen from Fig. 24, this causes the total number of emitted gluons to increase more or less proportional with $\alpha_s$, where for the final state shower, it grows only logarithmically (Fig. 18).

Varying $Q^2$ for the initial state shower yields similar results as for the final state shower, so we refrain from discussing this dependance further. There is however a parameter that does not appear in the final state shower, namely the proton-proton COM energy, $s_{tot}$. Figure 25 shows, for different values of $s$, the total number of gluons emitted as a function of $s_{tot}$ when we generate $10^4$ showers. Interesting to see is that increasing the beam energy does not per se mean that the number of emissions increases. With $s_{tot}$, the number of emissions grow rapidly up to a certain maximum, from where they decrease slowly. The location of the maxima seems to grow proportionally with $Q^2$, which indicates that there is a more or less fixed starting $x_1 = x_2 = x_{max}$ for which the number of emissions is maximal. This $x_{max}$ will shift somewhat with the scale $Q^2$ since the parton distribution functions also change with $Q^2$, but because the general shape of the PDF's doesn't change, the behavior is the same on all scales.

To understand the existence of a finite $s_{tot}$ that maximizes the number of emissions, we must
Figure 24: The number of gluons emitted with transverse momentum squared smaller or equal to $p_T^2$, for $10^5$ showers and different values of $\alpha_s$.

Figure 25: The total number of emitted gluons as a function of $s_{tot}$ for different values of $Q^2$, for $10^4$ showers per combination.
realize that the exponent of the Sudakov form factor, Eq. (256), is proportional to the ratio $-\frac{x^u(x_x)}{x^u(x)}$, where $f_u$ is the parton distribution function of the up quark, $x'$ the momentum fraction of the quark line on the proton side, and $x$ that of the quark line on the hard scattering side of the branching. Since the Sudakov form factor represents the probability of no branching, increasing the ratio $\frac{x^u(x_x)}{x^u(x)}$ increases the probability of branching, and hence the number of emitted gluons. Figure 26 shows the ratio $\frac{x^u(x_x)}{x^u(x)}$, using the MSTW 2008 NLO PDF's [14], where $x'$ is chosen to be $x' = 2x$. All three curves have a maximum near $x = 10^{-1.5}$, where for higher $Q^2$, the curve shifts somewhat to the left. Because to the right of the maximum the curves drop very fast, starting at the maximum would make the first branching easy, but any later branchings, which are at larger $x$, will be increasingly hard. Therefore, $x_{\text{max}}$ lies to the left of the peaks of the curves in Fig. 26. However, since the cut-off value of the COM energy of the gluons is fixed and does not scale with $Q^2$, the change in momentum fraction due to the emission of a soft gluon becomes smaller when we increase $Q^2$, thus allowing $x_{\text{max}}$ to get closer to the maximum of the curves in Fig. 26.
3 MC@NLO toy model with log-term

In the previous section we developed showering simulations for both timelike final state quarks, as well as spacelike initial state quarks. In the next section, section 4, we will present a modified version of the initial state shower with which we will attempt to better simulate the spacelike branching off the initial quark lines in a Drell-Yan process, near threshold. As we will show in section 4, the Drell-Yan cross section near threshold has a double logarithm. To get a feel for the effects of a double logarithm for a parton shower, we will construct a toy model similar to the MC@NLO toy model by Frxione and Webb [21], but with an additional logarithmic contribution to the cross section.

3.1 MC@NLO

Ref. [21] presents a toy model for a method for matching the next-to-leading order (NLO) calculation of a given QCD process with a parton shower Monte Carlo (MC) simulation. We won’t discuss this toy model in full detail, but we will go through it to some extent to be able to appreciate the similarities and differences with the "extended" model presented in section 3.2.

3.1.1 NLO

The toy model describes a system that can radiate massless particles that only have energy and no momentum, which we’ll call photons, whose energy is denoted by \( x \), with \( 0 \leq x \leq x_s \leq 1 \), where \( x_s \) is the energy of the system before radiation. After the radiation, \( x_s \to x_s - x \) and the system can emit another photon, but the emitted photons cannot themselves split further. The model considers only leading order (LO), or Born, and NLO contributions in perturbation theory, so at most one photon is emitted via this mechanism, which can either be virtual or real.

The corresponding contributions to the cross section are written as:

\[
\left( \frac{d\sigma}{dx} \right)_B = B\delta(x) \tag{281}
\]

\[
\left( \frac{d\sigma}{dx} \right)_V = a\left( \frac{B}{2\epsilon} + V \right)\delta(x) \tag{282}
\]

\[
\left( \frac{d\sigma}{dx} \right)_R = \frac{aR(x)}{x} \tag{283}
\]

for the Born, virtual, and real contributions respectively. Here \( \epsilon \) is the parameter entering dimensional regularization in \( 4 - 2\epsilon \) dimensions, \( a \) represents some coupling constant and

\[
\lim_{x \to 0} R(x) = B. \tag{284}
\]

If we then would want to calculate the expectation value of an (infrared-safe) observable to NLO accuracy, e.g. the average energy \( \bar{x} = \int dx x \frac{d\sigma}{dx} \), we would have to calculate

\[
\langle O \rangle = \lim_{\epsilon \to 0} \int_0^1 dx x^{-2\epsilon} O(x) \left[ \left( \frac{d\sigma}{dx} \right)_B + \left( \frac{d\sigma}{dx} \right)_V + \left( \frac{d\sigma}{dx} \right)_R \right], \tag{285}
\]

where \( O(x) \) is the observable as a function of \( x \). As can be seen in [21], using something they call the subtraction method, this can be written as

\[
\langle O \rangle_{\text{sub}} = \int_0^1 \left[ O(x) \frac{aR(x)}{x} + O(0) \left( B + aV - \frac{aB}{x} \right) \right]. \tag{286}
\]
3.1.2 MC

Typically a Monte Carlo procedure is used to calculate radiative corrections to a cross section (or another observable) at leading order. In the MC mechanism the system can emit a particle, in our toy model a photon, with some energy \( x \), where again \( 0 \leq x \leq x_s \leq 1 \). However this photon can then itself emit another photon with energy \( x' \), where \( 0 \leq x' \leq x \). This second photon can then again emit a photon and so on, until the remaining energy drops below some cut-off value \( x_0 \). The probability of emission, or branchings, is controlled by the Sudakov form factor \( \Delta (x_1, x_2) \), which we already encountered in Eq. (175). For the toy model in [21] the authors chose

\[
\Delta (x_1, x_2) = \exp \left( -a \int_{x_1}^{x_2} dz \frac{Q(z)}{z} \right),
\]

where \( Q(z) \) is some monotonic function such that

\[
0 \leq Q(z) \leq 1, \quad \lim_{z \to 0} Q(z) = 1.
\]

If \( x_M \) is the energy available to a new photon at a specific point in the MC evolution, \( \Delta(x, x_M) \) is the probability no photon of energy \( z \) is emitted such that \( x \leq z \leq x_M \). The procedure of selecting the photon energies is given in section 3.2 of [21]. It basically uses the veto algorithm as described in section 2.1 with

\[
f(z) = \frac{Q(z)}{z}, \quad g(z) = \frac{1}{z}.
\]

3.1.3 Matching NLO and MC

The general idea of matching the NLO and the MC predictions is to replace \( O(x) \) in Eq. (286) by a so-called interface-to-MC, \( I_{MC} (O, x_M(x)) \), which stands symbolically for the distribution in the observable \( O \) as obtained by running the MC with a maximum energy \( x_M \) available to the first photon, normalized to the Born value \( B \). Eq. (286) then becomes

\[
\left( \frac{d\sigma}{dO} \right)_{\text{naive}} = \int_0^1 \left[ I_{MC} (O, x_M(x)) \frac{aR(x)}{x} + I_{MC} (O, 1) \left( B + aV - \frac{aB}{x} \right) \right] d\theta.
\]

This would suggest that to generate an event, we would pick a random \( x \in (0, 1) \) representing the NLO emission, and then run two separate MC evolutions, one with maximum available energy equal to \( x_M(x) \), and one with maximum available energy 1, and with weights \( \frac{aR(x)}{x} \) and \( B + aV - \frac{aB}{x} \) respectively. However, both weights diverge as \( x \to 0 \), and although the divergencies cancel out, it is numerically unstable. Furthermore this method suffers from "double counting", meaning that the prediction of \( O(x) \) at \( O(a) \) is not equal to the NLO prediction.

To overcome these problems, Ref. [21] introduced a modified version of the subtraction method

\[
\left( \frac{d\sigma}{dO} \right)_{\text{msub}} = \int_0^1 \left[ I_{MC} (O, x_M(x)) \frac{a [R(x) - BQ(x)]}{x} + I_{MC} (O, 1) \left( B + aV + \frac{aB [Q(x) - 1]}{x} \right) \right] d\theta.
\]

where \( Q(x) \) is the same function that appears in the Sudakov form factor, Eq. (287). Eq. (291) does not have the problems of Eq. (290), and is used in [21] as the master equation for the toy model.
3.1.4 Results

The original toy model studies two observables. As an exclusive observable they study the quantity

\[ y = \max(x_1, \ldots, x_n), \]  

where \( n \) denotes the number of photons emitted in one event. As an inclusive observable they study the fully inclusive distribution of the photon energies

\[ z = \{x_1, \ldots, x_n\}, \]  

thus, including the energies of all the emitted photons. Furthermore they choose

\[ a = 0.3, \quad B = 2, \quad V = 1, \]  

and

\[ R(x) = B + x(1 + x/2 + 20x^2) \]  

so that

\[ \sigma_{\text{tot}} = B + a \left( V + \int_0^1 (1 + x/2 + 20x^2) \right) = 4.675. \]  

Since the toy MC code used implements ordered emissions in the photon energy, the first emitted photon always has the highest energy, so we expect to find for \( Q(y) = 1 \)

\[
\left( \frac{d\sigma}{dy} \right)_{\text{MC}} = a \sigma_{\text{tot}} \frac{Q(y)}{y} \Delta(y, 1) = \frac{a \sigma_{\text{tot}}}{y} \exp\{-a \int_y^1 \frac{1}{x} dx\}
\]

\[
= \frac{a \sigma_{\text{tot}}}{y} \exp\{a \log y\} = \frac{1.4025}{y^{0.7}},
\]  

where the \( \sigma_{\text{tot}} \) enter the equation because the MC procedure is normalized to \( \sigma_{\text{tot}} \) instead of \( B \). The NLO prediction is just

\[
\left( \frac{d\sigma}{dy} \right)_{\text{MC}} = \frac{a R(y)}{y}.
\]  

Fig. 27(a) shows the MC and NLO histograms for the quantity \( y \). Fig. 27(b) shows the MC results for different implementations of a dead zone for \( x \geq x_{\text{dead}} = 0.6 \). This corresponds to an MC that cannot emit photons with energy \( x > x_{\text{dead}}, \) which is easily introduced by choosing \( Q \) in Eq. (287) accordingly. Fig. 28 shows the histograms for the quantities \( y \) and \( z \) for different implementations of the dead zone. For all plots \( 10^5 \) events were generated.

3.2 The log model

We will now construct a similar toy model, but with an additional logarithmic contribution whose motivation we discussed in section 4. In the original MC@NLO toy model we had

\[
\sigma = \int_0^1 dx x^{-2\epsilon} \left[ B \delta(x) + a \left( \frac{B}{2\epsilon} + V \right) \delta(x) + a \frac{R(x)}{x} \right]
\]

\[
= B + a \left( \frac{B}{2\epsilon} + V \right) + a \int dx x^{-2\epsilon} \frac{1}{x} R(x)
\]

\[
= B + a \left( \frac{B}{2\epsilon} + V \right) + a \int dx \frac{e^{-2\epsilon \ln(x)}}{x} R(x) + a \int dx x^{1+2\epsilon}.
\]  

\[ \]
Figure 27: MC and NLO results for $y = \max(x_1, \ldots, x_n)$.

Figure 28: MC@NLO results for $y$ and $z$. 

MC, Q(x)=eq.(3.52)
MC, Q(x)=eq.(3.53)
MC, Q(x)=eq.(3.54)
MC@NLO, Q(x)=eq.(3.52)
MC@NLO, Q(x)=eq.(3.53)
MC@NLO, Q(x)=eq.(3.54)
NLO
NLO

(a) MC and NLO for $Q(x) = 1$

(b) MC with dead zone $x_{\text{dead}} = 0.6$. The equation numbers refer to equations in [21]. The first bin is normalized to unity.
Choosing $\epsilon < 0$ we get

$$
s(\epsilon) = B + a \left( \frac{B}{2\epsilon} + V \right) + a \int dx \left( \frac{1}{x^2} - 2\epsilon \left[ \frac{\ln(x)}{x} \right] + \frac{1}{2\epsilon} \delta(x) \right) R(x)
$$

$$
= B + aV + a \int dx \left( \frac{R(x) - B}{x} - 2\epsilon \left[ \frac{\ln(x)}{x} \right] + \frac{1}{2\epsilon} \delta(x) \right) .
$$

If we now would set $\epsilon \to 0$ the $\frac{\ln(x)}{x}$ vanishes. In realistic calculations double poles, and therefore double logs, do appear as seen for example in DIS (section 1.4) as well as in the next section. To keep a $\frac{\ln(x)}{x}$ present we adjust the contributions to the cross section as follows:

$$
\left( \frac{d\sigma}{dx} \right)_B = B\delta(x) 
$$

$$
\left( \frac{d\sigma}{dx} \right)_V = a \left( \frac{B}{2\epsilon} + \frac{B'}{2\epsilon^2} + V \right) \delta(x)
$$

$$
\left( \frac{d\sigma}{dx} \right)_R = a \left( \frac{R(x)}{x} + \frac{1}{\epsilon} \frac{R'(x)}{x} \right),
$$

where $R$ and $R' \to B$ and $B'$ respectively for $x \to 0$. We then get

$$
\sigma = \int_0^1 dxx^{-2\epsilon} \left[ B\delta(x) + a \left( \frac{B}{2\epsilon} + \frac{B'}{2\epsilon^2} + V \right) \delta(x) 
$$

$$
+ a \left( \frac{R(x)}{x} + \frac{1}{\epsilon} \frac{R'(x)}{x} \right) \right]
$$

$$
= B + a \left( \frac{B}{2\epsilon} + \frac{B'}{2\epsilon^2} + V \right) + a \int dx \frac{x^{-2\epsilon}}{x} \left( R(x) + \frac{1}{\epsilon} R'(x) \right)
$$

$$
= B + a \left( \frac{B}{2\epsilon} + \frac{B'}{2\epsilon^2} + V \right)
$$

$$
+ a \int dx \left( \frac{1}{x} - 2\epsilon \left[ \frac{\ln(x)}{x} \right] + \frac{1}{2\epsilon} \delta(x) \right) R(x) + \frac{1}{\epsilon} R'(x)
$$

$$
= B + aV + a \int dx \left( \frac{R(x) - B}{x} + \frac{1}{\epsilon} a \int dx \frac{R'(x) - B'}{x} \right)
$$

$$
+ a \int dx \left( \frac{\ln(x)}{x} \frac{R'(x) - B'}{x} \right) .
$$

So we are left with the original expression plus a term proportional to $\frac{\ln(x)}{x}$, which was what we wanted. We also pick up a singular term which we will ignore. \[^4\]

Following the procedure of section 3.1 (or actually the procedure described in \[^21\]) we apply

\[^4\]This term is in practice cancelled by a so-called mass factorization procedure, which is akin to renormalization.
the subtraction method to get
\[
\langle O \rangle_{\text{sub}} = BO(0) + a \left[ VO(0) + \int_0^1 dx \frac{O(x)R(x) - BO(0)}{x} \right. \\
\left. + \int_0^1 dx \ln x \frac{O(x)R'(x) - B'O(0)}{x} \right]
\]
\[
= \int_0^1 dx \left[ O(x) \left( \frac{aR(x)}{x} + a \ln x \frac{R'(x)}{x} \right) \\
+ O(0) \left( B + aV - \frac{aB}{x} - \ln x \frac{aB'}{x} \right) \right].
\]

(305)

Now, to match MC and NLO we get
\[
\left( \frac{d\sigma}{dO} \right)_{\text{naive}} = \int_0^1 dx \left[ I_{MC}(O, x_M) \left( \frac{aR(x)}{x} + a \ln x \frac{R'(x)}{x} \right) \\
+ I_{MC}(O, 1) \left( B + aV - \frac{aB}{x} - \ln x \frac{aB'}{x} \right) \right],
\]

(306)

and, switching to the modified subtraction method
\[
\left( \frac{d\sigma}{dO} \right)_{\text{msub}} = \int_0^1 dx \left[ I_{MC}(O, x_M) \left( \frac{a[R(x) - BQ(x)]}{x} + \ln x \frac{a[R'(x) - B'Q(x)]}{x} \right) \\
+ I_{MC}(O, 1) \left( B + aV + \frac{aB[Q(x) - 1]}{x} + \ln x \frac{aB'[Q(x) - 1]}{x} \right) \right].
\]

(307)

A convenient choice now is
\[
R' = -\alpha R \\
B' = -\alpha B,
\]

(308)

(309)

where \(\alpha\) is just a scaling parameter enabling us to vary the influence of the logarithmic term. Eq. (307) now becomes
\[
\left( \frac{d\sigma}{dO} \right)_{\text{msub}} = \int_0^1 dx \left[ I_{MC}(O, x_M) \left( \frac{a[R(x) - BQ(x)]}{x} (1 - \alpha \ln x) + \right. \\
I_{MC}(O, 1) \left( B + aV + \frac{aB[Q(x) - 1]}{x} (1 - \alpha \ln x) \right) \right].
\]

(310)

Effectively we have made the replacement
\[
\frac{1}{x} \rightarrow \frac{1 - \alpha \ln x}{x}.
\]

(311)

Since this doesn’t diverge faster then for \(\alpha = 0\), which is the original MC@NLO toy model, the modified subtraction method is still valid, and both terms are still finite for \(x \to 0\). The total cross section now becomes
\[
\sigma_{\text{tot}} = B + aV + a \int_0^1 dx \left( 1 - \alpha \ln x \right) [R(x) - B] = 4.675 + 1.00417\alpha,
\]

(312)

which is of the same order as in the original model (see Eq. 296) when \(\alpha\) is of order \(O(1)\). Hence the replacement in Eq. (311) can be considered a small adjustment to the original model.
3.2.1 Exclusive observable

If we now consider the exclusive quantity \( y = \max(x_1, \ldots, x_n) \) again, we expect

\[
\left( \frac{d\sigma}{dy} \right)_{\text{NLO}} = a \frac{R(y)}{y} (1 - \alpha \ln(y)) \tag{313}
\]

\[
\left( \frac{d\sigma}{dy} \right)_{\text{MC}} = a B Q(y) y \Delta(y, 1) \tag{314}
\]

Note that Eq. (314) is the same as in the original model, since we did not modify the Sudakov form factor. To see where the hardest photon came from, we have to consider the quantities \( I_{\text{MC}}(y, 1) \) and \( I_{\text{MC}}(y, x_M(x)) \) separately. For \( I_{\text{MC}}(y, 1) \) there is no NLO emission so the hardest photon comes from the MC giving

\[
I_{\text{MC}}(y, 1) = a Q(y) y \Delta(y, 1). \tag{315}
\]

For \( I_{\text{MC}}(y, x_M(x)) \), the hardest photon can come either from the NLO or the MC contribution, so

\[
I_{\text{MC}}(y, x_M(x)) = \Delta (\min \{x, x_M(x)\}, x_M(x)) \delta(y - x) + a Q(y) y \Delta(y, x_M(x)) \Theta(y - x) \Theta(x_M(x) - y), \tag{316}
\]

as derived in [21].

We will use the same shorthand notation as Frixione and Webber by denoting the contributions to \( d\sigma/dy \) from \( I_{\text{MC}}(y, 1) \) and \( I_{\text{MC}}(y, x_M(x)) \) by \( d\sigma_S/dy \) and \( d\sigma_H/dy \) respectively, corresponding to the Standard MC evolution (i.e. with no prior NLO emission), and to a hard MC evolution (whose initial conditions contained one NLO emission). From Eq. (306) we find that

\[
\left( \frac{d\sigma_S}{dy} \right)_{\text{naive}} = \frac{a Q(y)}{y} \Delta(y, 1) \left[ B + a V - a B \int_0^1 dx \frac{1}{x} \frac{1 - \alpha \ln x}{x} \right]. \tag{317}
\]

The contribution from \( d\sigma_H/dy \) depend on the function \( x_M(x) \) defined in section 3.1.3. If we define \( x_c \) to be the solution of \( x = x_M(x) \) and assume \( x_M(x) \) to be monotonically decreasing, we find from Eq. (316) that

- for \( y < x_c \):

\[
\left( \frac{d\sigma_{\text{naive}}}{dy} \right) = a \frac{R(y)}{y} (1 - \alpha \ln y) \Delta(y, x_M(y)) + a^2 \frac{Q(y)}{y} \int_0^y dx \frac{R(x)}{x} (1 - \alpha \ln(x)) \Delta(y, x_M(x)); \tag{318}
\]

- and for \( y > x_c \):

\[
\left( \frac{d\sigma_{\text{naive}}}{dy} \right) = a \frac{R(y)}{y} (1 - \alpha \ln y) + a^2 \frac{Q(y)}{y} \int_0^{x_M(y)} dx \frac{R(x)}{x} (1 - \alpha \ln(x)) \Delta(y, x_M(x)) \tag{319}
\]

Expanding in the coupling constant \( a \), we find that

\[
\left( \frac{d\sigma}{dy} \right)_{\text{naive}} = \left( \frac{d\sigma_{\text{naive}}}{dy} \right) + \left( \frac{d\sigma_{\text{naive}}}{dy} \right)_{\text{naive}} = a \frac{R(y)(1 - \alpha \ln y) + BQ(y)}{y} + O(a^2), \tag{320}
\]

63
which is, at first order in $a$, not equal to the NLO prediction. Hence, as in the original model, the naive subtraction method suffers from double counting.

If we now switch to the modified subtraction method in an attempt to solve these problems, we find

$$
\left( \frac{d\sigma}{dy} \right)_{\text{msub}} = \frac{a Q(y)}{y} \Delta(y, 1) \left[ B + aV + aB \int_0^1 \frac{Q(x) - 1}{x} \left( 1 - \alpha \ln x \right) dx \right]
$$

while

- for $y < x_c$:
  $$
  \left( \frac{d\sigma_{II}}{dy} \right)_{\text{msub}} = a \frac{R(y) - BQ(y)}{y} \left( 1 - \alpha \ln y \right) \Delta(y, x_M(y)) + a^2 \frac{Q(y)}{y} \int_0^y dx \frac{R(x) - BQ(x)}{x} \left( 1 - \alpha \ln x \right) \Delta(y, x_M(x)),
  $$

- and for $y > x_c$:
  $$
  \left( \frac{d\sigma_{II}}{dy} \right)_{\text{msub}} = a \frac{R(y) - BQ(y)}{y} \left( 1 - \alpha \ln y \right) \Delta(y, x_M(y)) + a^2 \frac{Q(y)}{y} \int_0^y dx \frac{R(x) - BQ(x)}{x} \left( 1 - \alpha \ln x \right) \Delta(y, x_M(x)).
  $$

Expanding in $a$ we now get

$$
\left( \frac{d\sigma}{dy} \right)_{\text{msub}} = a \frac{R(y) - BQ(y)}{y} \left( 1 - \alpha \ln y \right) \Delta(y, x_M(y)) + aB \frac{Q(y)}{y} + O\left(a^2\right)
$$

$$
= \left( \frac{d\sigma}{dy} \right)_{\text{NLO}} + aB \frac{Q(y)}{y} + O\left(a^2\right).
$$

So here the modified subtraction method as is, also suffers from double counting.

To cure this we have to redefine the Sudakov form factor from Eq. (287) as

$$
\Delta(x, y) = \exp \left[ -a \int \frac{dz}{z} \frac{Q(z)}{z} \left( 1 - \alpha \ln z \right) \right].
$$

We would then instead get

$$
\left( \frac{d\sigma_S}{dy} \right)_{\text{msub}} = \frac{a Q(y)}{y} \left( 1 - \alpha \ln y \right) \Delta(y, 1) \times
$$

$$
\left[ B + aV + aB \int_0^1 \frac{Q(x) - 1}{x} \left( 1 - \alpha \ln x \right) dx \right],
$$

leading to

$$
\left( \frac{d\sigma}{dy} \right)_{\text{msub}} = a \frac{R(y) - BQ(y)}{y} \left( 1 - \alpha \ln y \right) \Delta(y, x_M(y)) + O\left(a^2\right)
$$

$$
= a \frac{R(y)}{y} \left( 1 - \alpha \ln y \right) + O\left(a^2\right) = \left( \frac{d\sigma}{dy} \right)_{\text{NLO}} + O\left(a^2\right),
$$

so there is no longer double counting.
3.2.2 Inclusive observable

When we now accept Eq. (325), as our Sudakov form factor, and again consider as an inclusive quantity the fully inclusive distribution of the photon energies, \( z = \{x_1, \ldots, x_n\} \), we find that for the NLO and MC results we have

\[
\left(\frac{d\sigma}{dz}\right)_{\text{NLO}} = a \frac{R(z)}{z} (1 - \alpha \ln z),
\]

\[
\left(\frac{d\sigma}{dz}\right)_{\text{MC}} = aB \frac{Q(z)}{z} (1 - \alpha \ln z).
\]

In contrary to Eq. (314), Eq. (329) does not contain a factor \( \Delta(z, 1) \), since all photons contribute. To find the predictions of the MC@NLO, we have to consider the quantities \( I_{\text{MC}}(z, 1) \) and \( I_{\text{MC}}(z, x_M(x)) \) again, this time for the variable \( z \). These are:

\[
I_{\text{MC}}(z, 1) = aQ \left(\frac{z}{z} \right) (1 - \alpha \ln z),
\]

\[
I_{\text{MC}}(z, x_M(x)) = \delta(z - x) + a \frac{Q(z)}{z} (1 - \alpha \ln z) \Theta(x_M(x) - z).
\]

Here the \( \delta(z - x) \) represents the NLO emission, the other terms the MC emissions.

To check the \( z \) distribution of the naive subtraction method for double counting we find from Eq. (306)

\[
\left(\frac{d\sigma}{dz}\right)_{\text{naive}} = a \left(1 - \frac{\alpha \ln z}{z}\right) \left[ R(z) + Q(z) \left( B + aV - aB \int_0^1 dx \frac{1 - \alpha \ln x}{x} + a \int_0^1 dxR(x) \frac{1 - \alpha \ln x}{x} \right) \right]
\]

\[
= a \left(1 - \frac{\alpha \ln z}{z}\right) \left[ R(z) + Q(z) \left( \sigma_{\text{tot}} - a \int_{x_M^{-1}(z)}^1 dxR(x) \frac{1 - \alpha \ln x}{x} \right) \right],
\]

where \( \sigma_{\text{tot}} \) is given in Eq. (312). If we now expand again to order \( a \), we find

\[
\left(\frac{d\sigma}{dz}\right)_{\text{naive}} = a \left(1 - \frac{\alpha \ln z}{z}\right) \left[ R(z) + Q(z) \left( \sigma_{\text{tot}} - a \int_{x_M^{-1}(z)}^1 dxR(x) \frac{1 - \alpha \ln x}{x} \right) \right] + O(a^2)
\]

\[
= \left(\frac{d\sigma}{dz}\right)_{\text{NLO}} + aB \frac{Q(z)}{z} (1 - \alpha \ln z).
\]

So, as expected, we again find double counting in the naive subtraction method.

For the modified subtraction method we insert Eq. (330) and Eq. (331) into Eq. (310) to get

\[
\left(\frac{d\sigma}{dz}\right)_{\text{msub}} = a \left(1 - \frac{\alpha \ln z}{z}\right) \left[ R(z) - BQ(z) + Q(z) \left( \sigma_{\text{tot}} - a \int_{x_M^{-1}(z)}^1 dxR(x) - BQ(x) \frac{1 - \alpha \ln x}{x} \right) \right]
\]

\[
= a \left(1 - \frac{\alpha \ln z}{z}\right) R(z) + O(a^2) = \left(\frac{d\sigma}{dz}\right)_{\text{NLO}} + O(a^2).
\]

Hence, \( z \) is also free of double counting in the modified subtraction method, when \( \Delta(x, y) \) is chosen as in Eq. (325).
3.2.3 Results

To generate a MC parton shower with the Sudakov form factor as in Eq. (325), we again use the veto algorithm described in 2.1, but now with

\[ f(z) = Q(z) \frac{1 - \alpha \ln z}{z}, \quad g(z) = \frac{1 - \alpha \ln z}{z}. \]  

(334)

Now to choose a \( z \) according to the distribution \( 1 - \alpha \ln z \), we have to pick a random \( R \) and solve for \( z \) the equation

\[ \exp \left[ -a \int_{z}^{M} dz' \frac{1 - \alpha \ln z'}{z'} \right] = \exp \left[ -a \left( \ln \frac{x_M}{z} + \frac{\alpha}{2} (\ln^2 z - \ln^2 x_M) \right) \right] = R \]  

(335)

However, Eq. (335) has no analytical solution, so we use Newton’s method to numerically solve it. This, however, is very slow and it might be a good idea to adjust the root-finding such that it best fits this specific case, or to search for a different \( g(z) \) that has a nice invertible primitive, and is close enough to \( f(z) \) to make it efficient. We made no further attempts however in this direction.

To illustrate the properties of the extended model best, we choose \( \alpha = 2 \). Fig. 29 and 30 show the graphs for the extended model corresponding to Fig. 27 and 28 respectively. The analytic functions in Fig. 29(a), for \( Q(y) = 1 \) and the MC normalized at \( \sigma_{\text{tot}} \) given by Eq. (312), are given by

\[ \frac{d\sigma}{dy}_{\text{NLO}} = a R(y) \frac{1 - \alpha \ln y}{y}, \]  

(336)

\[ \frac{d\sigma}{dy}_{\text{MC}} = \alpha \sigma_{\text{tot}} Q(y) \frac{1 - \alpha \ln y}{y} \exp \left[ -a \int_{y}^{\frac{1}{y}} dy' \frac{1 - \alpha \ln y'}{y'} \right] \]

\[ = \alpha \sigma_{\text{tot}} Q(y) \frac{1 - \alpha \ln y}{y} \exp \left[ -a \left( \frac{\alpha}{2} \ln^2 y - \ln y \right) \right]. \]  

(337)
From Fig. 29(a) we see that the MC no longer diverges but is actually suppressed for \( y \to 0 \). This can also be seen directly from Eq. (337). If we take the limit \( y \to 0 \) the \( \ln^2 y \) term dominates the \( \ln y \) term in the exponent so

\[
\lim_{y \to 0} \exp \left[ -a \left( \frac{\alpha}{2} \ln^2 y - \ln y \right) \right] = \exp \left[ -\infty \right] = 0 .
\]

(338)

Since this suppression by the Sudakov form factor is exponential the divergent factor \( \frac{1-\alpha \ln y}{y} \) cannot overcome this suppression. This was not the case in the original model where the Sudakov form factor went to zero as \( y^a \), which was not enough to overcome the divergent \( \frac{1}{y} \) behavior. This behavior near the elastic limit is more realistic than the divergent behavior of the original toy model (Fig. 27(a)), since a small \( y \) means that all other emitted photons have to be even softer, which now becomes less likely when \( y \) decreases.

If we compare Fig. 30 with Fig. 28, we see that for small \( y \), the extended MC@NLO (Fig. 30(a)) differs very much from the original (Fig. 28(a)), as well as from the NLO. This is because for small values of \( y \) the MC is modified the most, since as discussed above, the asymptotic behavior for \( y \to 0 \) changes drastically going from the original to the extended model, and the MC is the dominant contribution at small \( y \) in the MC@NLO.

The latter can be easily seen from Eq. (310). Since \( R(x) - B \in \mathcal{O}(x) \), the first term has no \( \frac{1}{2} \) behavior, in contrary to the second (counter)term. Since this counterterm represents MC emissions without a previous NLO emission, the MC dominates for small \( y \). The dominance of the MC over the NLO for small \( y \) can also be clearly seen in Fig. 31. Here, the MC contribution includes both the counter term as the normal MC term.

As discussed above, the results for extended and the original MC@NLO for the exclusive observable \( y \) were very different. The results for the fully inclusive quantity \( z \), however, appear very similar. If we compare Fig. 30(b) with Fig. 28(b), we see that the shape of the two graphs are very similar. This emphasizes the earlier stated fact that the extended MC@NLO toy model can be considered a small adjustment to the original. However, if we look at the ratio of the fully inclusive cross sections of the extended and original model (see Fig. 32) we see that this has the shape of 1 minus a logarithm and becomes quite large for small \( z \). We can derive this behavior by comparing \( \left( \frac{d\sigma}{dz} \right)_{\text{meas}} \) for the original and the extended model:

\[
\frac{\left( \frac{d\sigma}{dz} \right)_{\text{meas}} \text{extended}}{\left( \frac{d\sigma}{dz} \right)_{\text{meas}} \text{original}} = \frac{a^{1-\alpha \ln z} R(z) + \mathcal{O} \left( a^2 \right)}{a^{R(z) \frac{1}{z} + \mathcal{O} \left( a^2 \right)}} \approx 1 - \alpha \ln z ,
\]

(339)
Figure 31: The contributions of the NLO and the MC (with or without prior NLO emission) to the total MC@NLO cross section for the exclusive observable $y$.

Figure 32: The ratio of the extended model over the original for the fully inclusive variable $z$. 
which corresponds well with Fig. [32]. We thus see that the extended model emits more photons in general, but especially more in the lower energy region.

Comparing the results for the variables $y$ and $z$, we can conclude that, although the extended model emits a lot more low-energy photons than the original model, the chance that such a low-energy photon is the hardest photon emitted has actually decreased. This is a result of the fact that the number of photons emitted in the middle energy region has also increased, albeit not as much as the lower energy region. Hence, since there are more photons with higher energy than the low-energy photon emitted than in the original model, the chance that a photon with higher energy was emitted is larger, i.e. the chance that the low-energy photon was the hardest is smaller. Hence the suppression by the Sudakov form factor, which is just this chance, overcomes the increase in emitted low-energy photons as shown by the $z$-distribution.
4 Threshold Resummation

In this section we will extend the initial state shower described in section 2.3 with an additional logarithm to account for the behavior of the Drell-Yan process near threshold, as already announced in the introduction of section 3, where we also created doubly logarithmic extension of the MC@NLO toy-model. We will first discuss some of the underlying theory of threshold resummation before continuing to the discussion of the results of the threshold model.

In perturbation theory one expresses observables as a power series in a sufficiently small parameter, usually the coupling constant, and when calculating that observable one keeps only the first few terms. Often however, especially in QCD, the relevant expansion parameter actually is $\alpha_s L$, where $L$ is some, potentially large, logarithm such that, even for small $\alpha_s$, the power series might not converge. Below, we shall see that the effective expansion parameter can even be $\alpha_s L^2$.

4.1 Theory

Define such an observable whose effective expansion parameter is $\alpha_s L$:

$$O = C_0 + \alpha_s(C_{11}L + C_{10}) + \alpha_s^2(C_{22}L^2 + C_{21}L + C_{20}) + \ldots. \quad (340)$$

Usually, when encountering such a logarithmic behavior, the expression can be used only in the perturbative range where $\alpha_s L$ is sufficiently small. For example, the factorized deep-inelastic proton structure function Eq. (88), we encountered a log

$$L = \ln \left(\frac{Q^2}{Q_0^2}\right). \quad (341)$$

Here one would argue that calculations only have predictive power for small enough $Q^2$ where perturbation theory still holds.

Another possible solution is resummation of the problematic terms into an analytic function, such that the observable becomes of the form:

$$O = f(\alpha_s, L)(1 + C_1' \alpha_s + C_2' \alpha_s^2 + \ldots), \quad (342)$$

where $f(\alpha_s, L)$ is an analytic function. As an example, consider again the deep-inelastic proton structure function $F_2$, assuming for simplicity only one quark type $q$ contributes, in moment space

$$F_2(N, Q) \equiv \int_0^1 dx x^{N-1} F_2(x, Q) = C(N, Q/\mu) q(N, \mu). \quad (343)$$

This is just the Mellin transform of Eq. (88), where the factor in square brackets, here replaced by $C(x, Q/\mu)$, is an infrared-safe coefficient function, and $q$ the distribution function for the quark in the proton. We can now rederive the DGLAP evolution equation as is done in Eq. (100), but in moment space. Since

$$\mu \frac{d}{d\mu} \ln F_2(N, Q) = 0, \quad (344)$$

we have

$$\mu \frac{d}{d\mu} \ln q(N, \alpha_s(\mu)) = -\mu \frac{d}{d\mu} \ln C(N, Q/\mu, \alpha_s(\mu)) \equiv \gamma, \quad (345)$$

where $\gamma = \gamma(N, \alpha_s(\mu))$ since $N$ and $\alpha_s(\mu)$ are the only common variables of $q$ and $C$. Integrating over $\mu$ from $Q_0$ to $Q$, we find the resummed expression

$$q(N, Q) = q(N, Q_0) \exp \left[ \frac{\int_{Q_0}^Q \frac{d\mu}{\mu} \gamma(N, \alpha_s(\mu))}{Q_0} \right]. \quad (346)$$
This was an example of the resummation of single logarithms (one logarithm per order of $\alpha_s$), since the exponent in square brackets, when approximating $\gamma(N, \alpha_s) \approx \alpha_s \gamma^{(1)}(N)$ reads

$$\alpha_s \gamma^{(1)}(N) \ln(Q/Q_0).$$

(347)

As stated above, the effective expansion parameter can even be $\alpha_s L^2$ with $L$ the potentially large logarithm. Such an observable would have a general form:

$$O = C_0 + \alpha_s (C_{12} L^2 + C_{11} L + C_{10}) + \alpha_s^2 (C_{24} L^4 + C_{23} L^3 + C_{22} L^2 + C_{21} L + C_{20}) \ldots$$

(348)

An example of such a doubly logarithmic behavior is found when considering the recoil of $Z$-bosons. For small transverse momenta one encounters

$$L^2 = \ln^2 \left( \frac{p_T}{M_Z} \right),$$

(349)

which blows up when $p_T$ becomes small.

As another example of a double logarithm we have seen the Mellin transform of the coefficient function Eq. (91) of the structure function $F_2$ in the MS factorization scheme, as encountered in section 1.4. It differs from the recoil double logarithm in the way that the logarithmic divergence now only appears in an integral (when we perform the inverse Mellin transform), whereas in the previous case the logarithmic (ultraviolet) divergence was manifestly there. This “invisible” doubly logarithmic behavior is often seen at systems near a certain threshold, and the resummation of those logarithms is therefore called threshold resummation.

In section 2.3 we discussed the spacelike showering off the initial state quark lines of a Drell-Yan process as seen in Fig. 33. For the cross section of this process we have

$$\frac{d\sigma_{\text{DY}}}{dQ^2} \propto \ln^2 \left( 1 - \frac{Q^2}{\hat{s}} \right).$$

(350)

When $Q^2$ approaches $S$, the $\ln^2 \left( 1 - \frac{Q^2}{\hat{s}} \right)$ factor becomes very large, and resummation is needed.
The threshold-resummed cross section for the Drell-Yan process reads\cite{23}

\[
\frac{d\sigma_{p\bar{p}}}{dQ^2} = \sigma_0(Q^2) \sum_{q\bar{q}} \int dN \frac{\tau^N}{2\pi^2} \phi_{q/p}(N-1, \mu_F) e^{E(N,Q,\mu_F)},
\]

where we approximate the exponent $E(N,Q,\mu_F)$ by a form that is accurate to next-to-leading logarithm (NLL) in $N$:

\[
E^{\text{eik}}(N,Q,\mu_F) = 2C_F \int_{Q/\tilde{N}}^Q \frac{dk_T}{k_T^2} \frac{\alpha_s}{\pi} \ln \left( \frac{k_T}{Q} \right) - 2 \ln(\tilde{N}) \int_{\mu_F}^{Q/\tilde{N}} \frac{dk_T}{k_T^2} \frac{\alpha_s}{\pi},
\]

where $\tilde{N} = Ne^{-\gamma_E}$, with $\gamma_E$ the Euler constant. Note that this is the result of a resummation of a double log, since Eq. (352) contains both an explicit log, and an implicit log by means of the $\frac{dk_T^2}{k_T^2}$. We can now rewrite and regroup Eq. (352) as

\[
E^{\text{eik}}(N,Q,\mu_F) = 4C_F \int_{Q/\tilde{N}}^Q \frac{dk_T}{k_T^2} \frac{\alpha_s}{\pi} \ln \left( \frac{k_T}{Q} \right) - 4C_F \ln(\tilde{N}) \int_{\mu_F}^{Q/\tilde{N}} \frac{dk_T}{k_T^2} \frac{\alpha_s}{\pi},
\]

\[
\approx -4C_F \int_{Q/\tilde{N}}^Q \left[ \frac{dk_T}{k_T^2} \frac{\alpha_s}{\pi} \ln \left( \frac{Q}{k_T} \right) + C \right]
\]

\[
+ C_F \int_{\mu_F}^{Q/\tilde{N}} \frac{dk_T}{k_T^2} \left[ -4 \frac{\alpha_s}{\pi} \ln(\tilde{N}) - C \right],
\]

where $C_F$ is the color factor.

After labeling

\[
E^{\text{thr}}(N,Q,\mu_F) = -4C_F \int_{Q/\tilde{N}}^Q \left[ \frac{dk_T}{k_T^2} \frac{\alpha_s}{\pi} \ln \left( \frac{Q}{k_T} \right) + C \right],
\]

\[
E^{\text{ev}}(N,Q,\mu_F) = C_F \int_{\mu_F}^{Q/\tilde{N}} \frac{dk_T}{k_T^2} \left[ -2 \frac{\alpha_s}{\pi} \ln(\tilde{N}) - C \right],
\]

such that Eq. (354) becomes

\[
E^{\text{eik}}(N,Q,\mu_F) = E^{\text{thr}}(N,Q,\mu_F) + 2E^{\text{ev}}(N,Q,\mu_F),
\]

we can substitute $E^{\text{eik}}$ into Eq. (351) and distribute the two factors $e^{E^{\text{ev}}}$ among the pdf’s to obtain

\[
\frac{d\sigma_{p\bar{p}}}{dQ^2} = \sigma_0(Q^2) \sum_{q\bar{q}} \int dN \frac{\tau^{-N}}{2\pi^2} \phi_{q/p}(N-1, \mu_F) e^{E^{\text{ev}}(N,Q,\mu_F)} \phi_{q/p} e^{E^{\text{thr}}(N,Q,\mu_F)} (N-1, \mu_F) e^{E^{\text{thr}}(N,Q,\mu_F)}.
\]

When ignoring any constant terms and factors, $E^{\text{ev}}$ is to first order in $\alpha_s$ equal to the exponent in Eq. (346), because in Eq. (347) $(1)_{(N)} = -\ln(\tilde{N})$. Hence, $E^{\text{ev}}$ evolves the pdf’s from $\mu_F$ to $Q/\tilde{N}$. Eq. (358) then simply becomes

\[
\frac{d\sigma_{p\bar{p}}}{dQ^2} = \sigma_0(Q^2) \sum_{q\bar{q}} \int dN \frac{\tau^{-N}}{2\pi^2} \phi_{q/p}(N-1, Q/\tilde{N}) \phi_{q/p}(N-1, Q/\tilde{N}) e^{E^{\text{thr}}(N,Q,\mu_F)}.
\]
The remaining exponent $E^{\text{thr}}$ (Eq. (355)) has the typical form of a Sudakov suppression, and it is this part that we will use to simulate radiation near threshold. The term $C$ should be fixed to account for the difference between the eikonal approximation and the full partonic cross section in the threshold region [24].

When considering the Drell-Yan system near threshold we have that $\tau = \frac{Q^2}{s}$ is almost 1, and only large moments $N$ contribute. We will furthermore replace all appearances of $\tilde{N}$ by $N$, since this differs only a constant factor. From [25] we see that the $\ln^2 N$ term that appears in $E^{\text{thr}}$ after doing the integral in Eq. (355), is the leading log of the Mellin transform of

$$\frac{1}{2} \ln^2 N \approx \int_0^1 dz \left( z^{N-1} - \frac{\ln(1-z)}{1-z} \right).$$

Since we also have

$$- \int_0^{1-\frac{1}{N}} \frac{\ln(1-z)}{1-z} + \int_0^1 \frac{\ln(1-z)}{1-z} = - \int_0^1 \frac{\ln(1-z)}{1-z} \bigg|_{(1-z)=\frac{1}{N}} = \frac{1}{2} \ln^2 N,$$

we find that $z \approx 1 - \frac{1}{N}$, so that when performing the inverse Mellin transform, we effectively have to make the substitution

$$N \rightarrow \frac{1}{1-z}.$$

Making this substitution we find

$$\frac{d\sigma}{dQ^2} = \Phi(z,(1-z)Q) \exp \left[ -4 \int \frac{d\mu}{(1-z)Q} \frac{\alpha_s}{\pi} \ln \left( \frac{Q}{\mu} \right) \right],$$

where $\Phi$ is the “parton flux” which we will define below, $z$ is defined as

$$z = \frac{Q^2}{s} = \frac{\tau}{x_1 x_2},$$

and the exponent represents the probability that no branching occurred between scale $Q$ and $(1-z)Q$, and thus provides us with a way to generate a $z$.

To define the parton flux, consider the general form of an inclusive hadronic cross section from a $p\bar{p}$-collision, allowing only one quark flavor and color:

$$\sigma_{p\bar{p}} = \int_0^1 \frac{dx_1}{x_1} \int_{x_1}^1 \frac{dx_2}{x_2} \phi(x_1) \phi(x_2) \hat{\sigma}(Q^2 = x_1 x_2 S),$$

where $\phi$ is the parton distribution function of the (anti-)quark in the (anti-)proton, $x_1$ and $x_2$ the momentum fractions of the quark and anti-quark, and $\hat{\sigma}$ the partonic cross section of the process. Since $\phi(x)$ vanishes for $x < 0$ & $x > 1$, and $x_1$ and $x_2$ are related by $x_1 x_2 S = Q^2$, we can set the lower integration limits to zero, since the added integration range does not contribute anyway. We can insert the identity

$$1 = \int_0^1 dz \delta \left( z - \frac{\tau}{x_1 x_2} \right)$$

73
to obtain
\[
\sigma_{p\bar{p}} = \int_0^\tau \int_0^{x_1} \int_0^{x_2} \frac{d\tau}{x_{1x_2}} \delta \left( z - \frac{\tau}{x_{1x_2}} \right) \phi(x_1) \phi(x_2) \tilde{\sigma}(x_1x_2s)
\]
\[
= \int_0^1 \frac{dz}{z} \int_0^1 dx_1 \int_0^1 dx_2 \left( \frac{\tau}{z} - x_1x_2 \right) \phi(x_1) \phi(x_2) \tilde{\sigma}'(zS)
\]
\[
\equiv \int_0^1 \frac{dz}{z} \Phi \left( \frac{\tau}{z} \right) \tilde{\sigma}'(zS), \quad (367)
\]
where we thus defined the parton flux \( \Phi \) as
\[
\Phi(x) = \int_0^1 dx_1 \int_0^1 dx_2 \delta \left( x - x_1x_2 \right) \phi(x_1) \phi(x_2). \quad (368)
\]

4.2 Parton shower implementation

To try and simulate the emission of gluons for the Drell-Yan process near threshold, we use Eq. (363) as our main formula, since this already resembles the structure of a parton shower. The exponential in Eq. (363), which can be interpreted as the probability that no resolvable gluons are emitted between \((1-z)Q\) and \(Q\), provides us with a probability distribution in \(z\), which we can use to generate \(z\) by means of hit-and-miss. However, once one selects a \(z\), it is not obvious how to proceed. The problem is that \(z\) has a physical meaning: \((1-z)s\) is the amount of energy radiated via gluon emissions, i.e.
\[
\sum_i E_{g_i} = (1-z)s. \quad (369)
\]
This is an overall condition on the total set of emitted gluons. Any attempt to fulfill it would therefore break the Markovian nature of the parton shower. Considering the fact that the main reason that parton showers are so successful, efficient and relatively easily implemented is that they are, in fact, Markov processes, indicates that this might be unwanted behavior.

To avoid the problem of the system not being Markovian, we chose to ignore the physical meaning of \(z\). The first factor, the parton flux, is then simulated by our original initial state parton shower, as discussed in section 2.3, but evolved down from a scale \((1-z)^2Q^2\) instead of the hard scattering scale \(Q^2\). In this section we will discuss the behavior of this model, and compare it to the standard initial state shower from section 2.3. Figure 34 shows the distribution in \(z\) that follows from the exponential in Eq. (363) which we will implement using a simple hit-and-miss algorithm. As can be seen the distribution favors small \(z\), which means that, on average, the difference between the threshold shower and the normal initial state shower, is that the former has only a small piece of \(t\)-space missing at the high-\(t\) side of the shower. This difference will turn out to be indeed very small, and for some observables hardly noticeable.

Fig. 35 shows the ratio of the transverse momentum distribution of the threshold shower over the normal initial state shower, for:
\[
\# \text{ of showers} = 10^5, \quad \alpha_s = 0.2, \quad Q^2 = 10^4 \text{GeV}^2, \quad S = 10^6 \text{GeV}^2, \quad E_{g_{\text{min}}} = 2 \text{GeV}, \quad (370)
\]
where \(E_{g_{\text{min}}}^{\text{min}}\) is the gluon cutoff energy, as introduced in Eq. (263), below which gluons are considered to be unresolvable. Note that this system is here chosen not to be near threshold, since
Figure 34: Normalized distribution in $z$ according to the exponential in Eq. \ref{eq:363}.

Figure 35: Ratio of the number of gluon emissions as a function of $p_T^2$ for the threshold shower over the standard initial state shower.
Figure 36: Normalized distribution of the hardest emission (energy-wise) of the threshold showers. Note that this observable is the same as the exclusive observable $y$ in the MC@NLO toy model, section 3.

$S \gg Q^2$, to better illustrate the differences between the normal and the threshold shower. It is clearly shown that the decrease in the number of gluons becomes more severe for higher $p_T^2$. To understand this, we must realize that the switch from the standard initial state shower to the threshold shower effectively is a more or less random cut-off on the high-$t$ side of the shower. Since, as discussed before, the harder emissions tend to be in the high virtuality region, whereas the low virtuality region is dominated completely by soft collinear emissions, cutting off a part of the high-$t$ range decreases mostly the number of hard emissions.

When we now choose the system to be near threshold, for example we choose

$$Q^2 = 5 \cdot 10^3 \text{GeV}^2$$
$$S = 10^4 \text{GeV}^2,$$

we find that both the normal as the threshold shower hardly branch at all, and when they branch it is almost always only one emission. In that case, the only effect that the switch from the normal initial state shower to the threshold shower has, is that the total number of emissions increases slightly. The statistics of these emissions, however, are essentially indistinguishable. To understand this, we must realize that the initial state shower favors low virtualities over high virtualities. Hence, when there is only one emission from the normal initial state shower, the probability that its virtuality falls below the $(1-z)Q$ threshold cutoff is very high, in which case there is no difference between the standard and the threshold shower. In order to stimulate the number of emissions while remaining near threshold, we increase the coupling constant, $\alpha_s \to 0.8$, and decrease the gluon energy cutoff, $E_{g\min} \to 0.25$.

Fig. 36 shows the normalized distribution in the exclusive observable $y$, as defined in Eq. (292) for the MC@NLO toy model, which, for one shower, is the energy of the gluon with the highest energy. Recall the behavior of the MC in the MC@NLO toy model, shown in Fig. 27(a), and the MC in the extended model, Fig. 29(a), where we introduced an extra logarithm in the cross section. Through simulations we have seen that for the extended, doubly logarithmic, model the
Figure 37: The ratio of the distributions in the $y$ observable for the threshold shower over the normal initial state shower.

distribution did no longer diverge for $y \to 0$, and that below a certain turnover value of $y$ the distribution actually drops. The same behavior can be seen in Fig. 36, which indicates that the system indeed behaves double logarithmically. However, the same plot for the normal initial state shower, which is not shown here, is essentially identical to that of the threshold shower, and has a turnover as well. This can be understood if we realize that the initial state Sudakov form factor, Eq. (256), already has two implicit logarithms, namely $\frac{dt}{T}$ and $\frac{dz}{z}$. To evaluate the differences between the threshold and the normal shower, we therefore look at the ratio of the two, which can be found in Fig. 37. Since in the threshold shower, a piece of the high-$t$ range is cut off, we expect that the hardest gluon is, on average, softer than for the standard initial state shower, because most of the hard emissions happen close to the hard scattering. We indeed see that for relatively high energies $y$, the ratio is on average significantly below 1, meaning that this happens more often for the standard shower than for the threshold shower. We also see that for relatively low energies $y$ the ratio is above 1, meaning that this happens more often for the threshold shower. The latter statement is a direct consequence of the former since, unless no gluons are emitted, some gluon must have the highest energy. Hence, if for the threshold shower it happens less often that the hardest gluon is from the say $y > 2$ GeV region, it must happen more often that the hardest gluon is from the $y < 2$ GeV region.

Fig. 38 shows the ratio of the distributions for the threshold shower over the normal shower in the observable $z$, as defined in Eq. (293), which is the fully inclusive set of gluon energies of a shower. We can see that the threshold shower emits less gluons along the whole energy spectrum, and especially at higher energies. This is, again, because of the cut on the high-$t$ side of the shower where, as discussed before, almost all of the hard gluons are generated. Fig. 38 shows that the threshold shower also emits less soft gluons, because the initial state shower can also emit soft gluons at high $t$. 
Figure 38: The ratio of the distributions in the $z$ observable for the threshold shower over the normal initial state shower.
A  Code

This appendix shows the code, used for the various simulations, that was created during the project. All files shown here, as well as the makefiles, configuration templates, and code obtained from other sources can be found at [http://www.nikhef.nl/~bveroude/thesis/code/](http://www.nikhef.nl/~bveroude/thesis/code/). If for any reason the files are not accessible, they can be requested by sending an email to bartverouden@gmail.com.

A.1  Code Final State Shower

The final state shower uses a slightly modified version of the file `<vec4d.cpp>`, and `<bindat.cpp>` from section A.4, as well as a pseudo random number generator from [27].

A.1.1  shower.h

```c
/*
 * Name: shower.h (header file for shower.cpp)
 * Author: Bart Verouden
 * Version: 1.0 - 14 October 2009
 *
 */

#ifndef _SHOWER_
#define _SHOWER_
#include "partons.h"

double A(double z0);
double P(double z);
double z0(double t, double E);
double a_s = 0.2;

#endif
```


This program generates final state showers. The program creates an instance of the quark-class as defined in partons.h which generates a gluon and another quark, the latter creating yet another quark-gluon pair etc., until the virtuality of a new quark drops below the cutoff value $t_0$. The new quark is then put on mass shell, and the result is a generated (simple) parton shower. The program keeps doing this until it generates a shower with at least 2 branchings. At the moment the generated gluons are not allowed to branch, and are put on mass shell. After a shower with at least 2 branchings has been generated its particle’s 4-momenta and other properties are printed, and a momentum conservation check is done.

Name: shower.cpp
Author: Bart Verouden
Version: 1.01 – 20 October 2009
Uses: partons.cpp (and header files)

Includes:
- `int main()`: Creates the parton showers.
- `double P(double z)`: The splitting function.
- `double A(double z0)`: $(a_s/2\pi)$ times The integrand over $z$ of the splitting function from $z_0$ to $1-z_0$.
- `double z0(double t, double E)`: The cutoff value of $z$ as a function of the virtual mass squared and the energy

#include <stdio.h>
#include <math.h>
#include "shower.h"
#include "partons.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
using namespace std;

bool debug2 = false;

#define PI 3.1415926535897932385

/*
 * Global variables used as external variables in partons.cpp.
 */

double tmin = 1.01, Q2 = 1000., binsize = 0.01;
int nobmax=0, quark::nob=0;
inbins = (int)(0.25*Q2/binsize)+1;

int main () {


quark * godfather = NULL; // The initial quark.
int nsim = 100000;

cout << "Q2 = " << Q2 << ", nsim = " << nsim << endl;

DynBinDat *pt2 = new DynBinDat((int)Q2);
pt2->add(Q2/4);

for(int n=0; n<nsim ; n++)
{
    if((n*100)%nsim==0)
        cout <<(n*100)/nsim<<"%\n";
    delete godfather;
    godfather = new quark(Q2,NULL);
    godfather->trackpt2(pt2);
}

stringstream ss, ss2;
ss<<"pt2_cum_as="<<a_s;
pt2->plotCum(ss.str());
ss2<<"pt2_as="<<a_s;
pt2->plotSec(ss2.str());

// code for generating mathematica 3d plot of the shower.
// godfather->graphplot(1);

/*
 * Call the print function of the initial quark, which in turn calls
 * the print function of the children, hence, effectively, printing the
 * shower.
 */
godfather->print();

return 0;
}

/*
 * The q->qg splitting function
 */
double P(double z) {
    return (1+z*z)/(1-z);
}

/*
 * A= (a_s/2*PI)Integrate [P(z),\{z,z0,1-z0\}] 
 */
double A(double z0) {
    return (a_s/(2.*PI))*( -3./2. + 3.*z0 + 2.*log(-1.+1./z0));
}

/*
 * z0(t) gives the cutoff value of z as a function of the virtual mass squared,
 */
* and the energy.

```c
double z0(double t, double E) {
    return fmax(0.5*(1.0 - sqrt(1.0 - t/(E*E))), 1.0e-6);
}
```
A.1.3 parton.h

/*
 * The class parton is the abstract base class of the subclasses quark and
 * gluon. It contains the pure virtual function print(), making it an abstract
 * class. These classes are made to generate a final state parton shower.
 * Name: partons.h (header file of partons.cpp)
 * Author: Bart Verouden
 * Version: 1.01 – 20 October 2009
 */

#ifndef PARTON
#define PARTON
#define PI 3.1415926535897932385
#include "vec4d.h"
#include "bindat.h"

double g(double t, double tmax);
double G(double t, double tmax);
double Ginv(double x, double tmax);

class parton
{
protected:
    double phi;       // The azimuthal angle w.r.t. the parent frame
    double theta;     // The polar angle w.r.t. the parent frame
    double t;         // The virtuality of the parton
    double z;

    ...

    /*
    * Pointers to child and parent partons. These are NULL if nonexisting.
    */
    class parton *qchild;
    class parton *gchild;
    class parton *parent;

public:
    vec4d p;           // the 4-vector in the COM-frame
    /
    * Constructors and destructors
    */
    parton(double tmax, parton *parent);
    virtual ~parton(){};

    /*
    * General public functions
    */
void transformcoordinates(double *px, double *py, double *pz);
void transformcoordinates(void){ transformcoordinates(&p.x, &p.y,&p.z );}
double transformz(double td);
virtual void print(void) =0;
double f(double t, double tmax);
double pt2(void);
void graphplot(int i);
virtual void trackpt2(DynBinDat *pt){};

/*
 * Getters
*/
double gett(void){ return this->t;}
double getE(void){ return p.E;}
double getz(void){ return z;}
parton *getchild(){ return gchild;}
parton *getqchild(){ return qchild;}
virtual parton *getsister()=0;

/*
 * Setters
*/
void setphi(double phi){ this->phi=phi;}
void setz(double z){ this->z=z;}
void settheta(double theta){ this->theta=theta;}
void setE(double E){ p.E=E;}
void setpz(double pz){ p.z=pz;}

};
class quark : public parton
{
public :
static int nob; // nob is the number of branches in the shower.

/*
 * Constructors and destructors.
*/
quark(double tmax, parton *parent);
~quark();

/*
 * Implementations of virtual functions inherited from parton class.
*/
parton *getsister();
void print(void);
void trackpt2(DynBinDat *pt);

/*
 * General public functions.
*/
bool checkz(void);
};
class gluon : public parton
{
public :
  /*
   * Constructors and destructors.
   */
gluon(double tmax, parton *parent);
~gluon();

  /*
   * Implementations of virtual functions inherited from parton class.
   */
  parton *getsister();
  void print(void);
  void trackpt2(DynBinDat *pt);
};

#endif // _PARTON_
A.1.4 parton.cpp

/*
 * This file contains the constructors and functions for the parton class and
 * its subclasses.
 * Name: partons.cpp
 * Author: Bart Verouden
 * Version: 1.01–20 October 2009
 * Uses: rng.cpp (and header file)
 *
 * Includes:
 *   - class parton
 *   - class quark (subclass of parton)
 *   - class gluon (subclass of parton)
 *
 */
#include "partons.h"
#include "rng.h"
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <vector>
#include "vec4d.h"
using namespace std;

#define PI 3.1415926535897932385
#define DEFAULTSEED 123456789L

// set to true to enable debugging
bool debug=false;

ofstream graph("shower.nb");

/*
 * Constructors and destructors
 *
 */

/*
 * The parton constructor requires as input the maximum virtuality available to
 * the parton, as well as a pointer to the parent parton, which must be set NULL
 * if the parton is an initial parton. The constructor initializes all variables
 * to zero (except for t), sets the pointer to parent, and initializes the RNG if
 * not done so already.
 *
 */
parton::parton(double tmax, parton *parent)
{
    long testseed;

    /* Initialize variables.*/
    phi=theta=z=0.0;
p = new vec4d();

/* Set parent if the parton isn't an initial parton */
if (parent) this->parent = parent;
else {
  this->parent = NULL; /* Initialize random number generator if not initialized yet */
  GetSeed(&testseed);
  if (testseed == DEFAULTSEED) {
    if (debug) cout << "initializing RNG" << endl;
    PutSeed(-1); // The seed -1 generates a seed depending on the time.
  }
}

/* initializing child pointers to NULL */
qchild = NULL;
gchild = NULL;

/*
 * The gluon constructor calls the parton constructor with the same values. No further branching from a gluon is allowed at this time, hence the values of the pointers to the children are set to NULL
 */

 gluon::gluon(double tmax, parton *parent) :
  parton(tmax, parent) {
    gchild = qchild = NULL; // Gluons are not allowed to branch
    t = 0.; // and are therefore chosen to be on shell
  }

/*
 * The quark constructor is the core of the final state shower and calls the parton constructor with the same values. A virtual mass t<tmax and the splitting variable z: z0<z<1−z0 are generated by veto techniques. After that, the kinematic variables are calculated, as well as the gluon sisters and if t>tmin, a new quark−gluon pair is created.
 */

 quark::quark(double tmax, parton *parent) :
  parton(tmax, parent) {
    extern double A(double z0), z0(double t, double E), P(double z);
    extern double tmin, Q2;
    double told, E, z00;
    int snob;

    /* Keep track of number of branches in the shower. */
    if (parent) {
      snob = nob;
      nob = snob + 1; // Increase number of branches
    }
else

    nob = 0; // Reset number of branches

if(debug)
    cout << "Generating new quark\n";

/* Start the Veto Algorithm */
double scale = (parent ? parent->getz() * parent->getE() : tmax);
t = tmax;
do{
    if(tmax<tmin){ //This happens if the particles energy is less then tmin.
        t = 0.;
        break;
    }
    told = t; // Set the (new) upper limit for the allowed t.
do t = Ginv(log(Random()) + G(told, scale), scale); // Pick a t......
while(t >= told); // .......<told.
if(t < tmin) break;
/* Calculate energy (corresponding to the selected t) */
if(!parent)
    E = (Q2 + t) / (2. * sqrt(Q2));
else
    {E = parent->getz() * parent->getE();}
/*
 * Calculate splitting variable z in the interval (z0,1−z0)
 * corresponding to the selected t and E. z is randomly choosen, and
 * accepted with a chance proportional to the splitting function P(z).
 */
z00 = z0(t,E);
do
    z = z00 + Random() * (1. - 2. * z00);
    while (P(z) / P(1.0 - z00) < Random());
/* Veto if f/g < R, or if the parent’s z falls out of its new boundaries. */
} while(f(t, tmax) / g(t, scale) < Random() || (parent ? !checkz() : false));

if(debug) cout << "generated t = " << t << endl;

if(t < tmin)
    t = 0.; // put the quark on shell if the virtuality is to low.
/* Adjust parent’s splitting variable if the daughter quark got a mass. */
else if(parent){
    if(debug) cout << "z: " << parent->getz();
    parent->setz(parent->transformz(t));
    if(debug)
        cout << " \rightarrow " << parent->getz() << ", " << "z0 = " << z0(t,E) << endl;
}

/* Set Kinematics */
if(parent) {
    double zeff = parent->getz(); // Get parent’s final z.
    p.E = zeff * parent->getE(); // Set quark’s final E.
    p.z = sqrt(p.E*p.E-t); // Set quarks 3 momentum in it’s own frame.
}
double Es = (1.0 - zeff) * parent->getE(); // Sister gluon's Energy.
double ps = Es; // Sister gluon's |p|.

phi = 2.0*PI*Random(); // Pick phi random from flat distribution.

getsister()->setphi(phi); // Set sister's phi the same as this phi.

/*
 * Calculate the angle between the new quark and gluon from 4(or 3)-
 * momentum conservation, in order to calculate the quark's theta
 * and the gluon's theta from transverse momentum conservation.
 */

double costhn = 2.0*p.E*Es-parent->gett()+t;
double costhd = 2.0*p.z*ps;
double costh = costhn/costhd;

if(debug){
    cout <<" costh = " << costhn <<" / " << costhd <<" = " ;
    cout <<costh<<endl;
}

double sinth = sqrt(1.0-costh*costh);
double xi=(1.0/acos(costh))*atan(ps*sinth/(p.z+ps*costh));

/* Set the quark and the sister gluon's theta */
theta = fabs(xi+acos(costh));
getsister()->settheta(-(1.0-xi)*acos(costh));
getsister()->setE(Es); // Set the sister's energy.
getsister()->setpz(ps);// Set the sister's 3 momentum in it's own frame.
}
else {
    p.E = (Q2+t)/(2.*sqrt(Q2));
p.z = (Q2-t)/(2.*sqrt(Q2));
theta = 0.0;
}

/* Transform the 3-momenta to the COM frame. */
transformcoordinates();

if(parent)
    getsister()->transformcoordinates();

/* Create a new quark-gluon pair if t>tmin */
if (t>tmin) {
    gchild = new gluon(fmin(t,(1.-z)*(1.-z)*E*E), this);
    qchild = new quark(fmin(t,z*z*E*E), this);
}
else {
    qchild = NULL;
    gchild = NULL;
}
}

quark::~quark() {
    delete qchild;
    delete gchild;
}

gluon::~gluon() {}

The implementations of the parton's virtual getsister() function that returns its sister parton, which is a quark for the gluon and vica versa.

```c
parton *quark::getsister(void) {
    return parent->getgchild();
}
parton *gluon::getsister(void) {
    return parent->getqchild();
}
```

The implementations of the parton's overloaded virtual trackpt2 function which tracks the distribution of the transverse momentum of the emitted gluons either in an external vector, or in a DynBinDat object that is passed to the function by reference.

```c
void quark::trackpt2(DynBinDat *pt) {
    if (qchild) {
        qchild->trackpt2(pt);
        gchild->trackpt2(pt);
    }
}
void gluon::trackpt2(DynBinDat *pt) {
    pt->add(pt2());
}
```

The implementations of the partons virtual print() function. It prints information about the quark/gluon and calls the print function of it's daughters.

```c
void quark::print (void){
    cout <<(parent?(t>0?"A virtual ":"A final "):"The initial ");
    if (qchild) gchild->print();
    if (qchild) qchild->print();
}
void gluon::print (void){
    printf("A gluon was emitted with: E=%lf, phi=%f, theta=%f\n",p.E, phi, theta);
    printf("Momentum: ( %9f, %9f, %9f )\n", p.x,p.y,p.z);
    if (gchild) gchild->print();
    if (qchild) qchild->print();
}
```
/ * The parton::transformcoordinates function transforms coordinates (px, py, pz) * in its own frame to its parents frame, an then calls the * transformcoordinates function of the parent, using the new coordinates as * input. So effectively, transformcoordinates transforms the coordinates to the * initial (C.O.M.) frame.
 */
void parton::transformcoordinates(double *px, double *py, double *pz)
{
    double xt, yt, zt;

    xt= cos(theta)*cos(phi)*px-cos(theta)*sin(phi)*py+sin(theta)*cos(phi)*pz;
    yt= sin(phi)*px + cos(phi)*py + sin(theta)*sin(phi)*pz;
    zt= -sin(theta)*cos(phi)*px + sin(theta)*sin(phi)*py + cos(theta)*pz;

    *px=xt;
    *py=yt;
    *pz=zt;

    if (parent) parent->transformcoordinates(px,py,pz);
}

/*
 * The double parton::transformz function returns the transformed z-value where * the masses of its daughters have been taken into account.
 */

double parton::transformz(double td)
{
    return (z-0.5)*(t-td)/(t+ttd)/(2.*t);
}

/*
 * The bool quark::checkz function checks if for a generated t, the generated z * still lies between z0 and 1-z0 which are dependent on the energy, which * gets changed when the quark gets a mass.
 */
bool quark::checkz(void)
{
    extern double z0(double t, double E);
    double z00 = z0(t,parent->transformz(t)*parent->getE());
    return(z >= z00 && z<=1.-z00);
}

/*
 * Returns the transverse momentum squared of the sub-shower initiated by the * parton, where the z-axis aligns with the direction of the first parton.
 */

double parton::pt2(void)
double parton::f(double t, double tmax)
{
extern double z0(double t, double E), A(double z0);
if (parent)
    return (A(z0(t, parent->getz())*parent->getE())) / t);
else
    return(A(t / (tmax+t)) / t);
}

double g(double t, double tmax)
{
    return(0.07*sqrt(tmax)/(t*log(t)));
}

double G(double t, double tmax)
{
    extern double tmin;
    return(0.07*sqrt(tmax)*log(log(t)/log(tmin)));
}

double Ginv(double x, double tmax)
{
    extern double tmin;
    return pow(tmin, exp(x/(0.07*sqrt(tmax))));
}
/ * Generate Mathematica code that creates a 3d representation of the shower */

void parton::graphplot(int i)
{
    extern int nobmax;
    if (!parent) {
        i = 1;
        graph << "q0={0,0,0}\n";
        graph << "q1={" << p.x << "," << p.y << "," << p.z << "}\n";
    } else {
        if (this == parent->qchild) {
            graph << "q" << i << ";n";
            graph << "q" << i - 1 << "+n" << "{" << p.x << "," << p.y << "," << p.z << "}\n";
        } else {
            graph << "g" << i << ";n";
            graph << "q" << i - 1 << "+n" << "{" << (nobmax - i + 2) * p.x << "," << (nobmax - i + 2) * p.y << "}\n";
            graph << "," << (nobmax - i + 2) * p.z << "}\n";
        }
    }

    if (qchild) {
        qchild->graphplot(i + 1);
        qchild->graphplot(i + 1);
    } else if (!parent || this == parent->qchild) {
        graph << "tree={a0->a1}";
        for (int j = 1; j < i; j++)
            graph << ",a" << j << "$b" << j + 1 << ",a" << j + 1 << "$a" << j + 1;
        graph << "}\n";
        graph << "GraphPlot3D[tree, VertexCoordinateRules ->\n{a0->q0, a1->q1};
        for (int j = 2; j < i; j++)
            graph << ",a" << j << "$q" << j << ",b" << j << "$g" << j;
        graph << "}\n";
        graph. close();
    }
}
A.2 Code Initial State Shower

The initial state shower uses a slightly modified version of the file <vec4.cpp> and its header from Pythia 8.1 [26], as well as a pseudo random number generator [27]. Additionally it uses the mstw2008 pdf representation <mstwpdf.cc> from [13] and the file <bindat.cpp> which can be found in section A.4. Furthermore it uses the file <ConfigFile.cpp> from [28] to optionally load a template for the configuration of the various parameters such as $\alpha_s$, $Q^2$ and $S_{had}$ and others. Such a template, or configuration file, can be called in the command line by passing the filename as an argument to the program. For example running .\isshower threshold.conf runs both the normal and the threshold shower simulations for the parameters as defined in section 4.2.

A.2.1 isshower.h

/*
* The class isshower is the base of the initial state showering algorithm for
* p–pbar collisions.
* Name: isshower.h (header file of isshower.cpp)
* Author: Bart Verouden
* Version: 1.1 – 28 May 2010
* Uses: vec4.h (four–vector handling from Pythia)
*        bindat.h (binning tool)
*        isparton.h (contains initial state partons)
* Includes:
* - class isparton
* - class isquark (subclass of isparton)
* - class isgluon (subclass of isparton)
* /

#ifndef _ISShower_
#define _ISShower_

#include "isparton.h"
#include "bindat.h"
#include "vec4.h"
#include <iostream>

using namespace std;

class isshower
{
    protected:
    Vec4 K;  // Sum of all emitted gluon four vectors.

    public:
    /* Constructors and destructors */
    isshower(double tmax, double x1, double x2);
    ~isshower();

    /* General public functions */
    void ptdist(DynBinDat *dat);
    void Edist(DynBinDat *z, DynBinDat *y);
    Vec4 photon(void);

    isquark *q1, *q2;  // The two colliding parton lines.

*/
/ * Static variables */
  static bool kinbreak;

  /* Friends */
  friend ostream& operator<<(ostream&, const isshower&);
};
#endif // _ISSHOWER_

double selectz(void);
void outconf(void);
This file generates initial state (spacelike) showers by the use of the isparton class and its subclasses. The variables defined in the section Configurable variables can be changed by creating a configuration file and passing its path as an argument when executing the program. These values need to be assigned. Also an global pointer to a c_mstwpdf instance named pdf must be declared and initialized here, as well as all static members of isparton and its subclasses.

This specific file uses a file bindat.h, which contains the classes BinDat and DynBinDat that I created, and enables the user to quickly and easily distribute data over bins and export and/or plot the distributions.

Name: isshower.cpp
Author: Bart Verouden
Version: 1.1 - 28 May 2010
Uses:
- isparton.h (contains the parton classes)
- mstwpdf.h (mstw parton distribution functions)
- bindat.h (bin-tool)
- ConfigFile.h (handles configuration files)

#include <cmath>
#include <sstream>
#include <ctime>
#include <fstream>
#include "isparton.h"
#include "isshower.h"
#include "mstwpdf.h"
#include "bindat.h"
#include "vec4.h"
#include "rng.h"
#include "ConfigFile.h"

using namespace std;

/* Configurable variables (Default values entered) */
double as=0.2; // The (fixed) strong coupling constant
double tmin = 1.; // The cutoff virtuality
double stot=1e8; // Beam center of mass energy squared in GeV^2
double s_hat=1e4; // Center of mass energy of the interacting partons.
double res = 2.; // The minimum energy for emitted gluons to have to be considered resolvable.
bool do_iss = true; // simulate the initial state shower.
bool do_thr = false; // simulate the threshold shower.
int nos = 100000; // number of showers to simulate.
bool trackpt = true; // track the transverse momenta of the gluons.
bool trackE = false; // track the energy of the gluons in the lab frame.
string output = "out"; // set the output folder for plots and data.

/* Global variables */
double maxfactor = 0; // Track the correctness of the overestimation for the
// Veto algorithm
c_mstwpdf *pdf; // MSTW2008 parton distribution functions
double maxE; // The maximum gluon–energy emitted by the shower.

/* Declaration of static class members */
int isquark::nob=0;
double isquark::x2;
Vec4 isparton::p_1;
Vec4 isparton::p_2;
bool isparton::kinbreak;
bool isshower::kinbreak=false;
RotBstMatrix isparton::history;
int isparton::gluoncount = 0;

imizer

/* Main function – As an argument the user can specify the path to the
* configuration he wants to use.
* */

int main(int argc, char* argv[])
{
    extern double sfty;
    bool redone = false;

    if(argc==1){
        cout<<"No configuration file specified. Using default values:\n";
        outconf();
    } else {
        try{
            ConfigFile input(argv[1]);
            cout << "Using \"" << argv[1] << "\" as configuration file.\n";
            input.readInto(as, "as");
            input.readInto(tmin, "tmin");
            input.readInto(stot, "stot");
            input.readInto(s_hat, "s_hat");
            input.readInto(res, "res");
            input.readInto(do_iss, "do_iss");
            input.readInto(do_thr, "do_thr");
            input.readInto(nos, "nos");
            input.readInto(trackpt, "trackpt");
            input.readInto(trackE, "trackE");
            input.readInto(output, "output");
        }
        catch(ConfigFile::file_not_found) {
            cout<< "Error: Config file not found at " << argv[1] << "\n";
        }
    }
}
cout << "Using default configuration:\n";
outconf();
}

/* Initialize the parton distribution functions */
char filename[] = "mstw2008nlo.00.dat";
pdf = new c_mstwpdf(filename);

isshower *iss = NULL;
isshower *news = NULL;

DynBinDat *pt = NULL;
DynBinDat *ptn = NULL;
DynBinDat *y_nor = NULL;
DynBinDat *y_thr = NULL;
DynBinDat *z_nor = NULL;
DynBinDat *z_thr = NULL;

double x1, x2, z;
x1 = x2 = sqrt(s_hat / stot);

do{
    if (do_iss) {
        if (trackpt) 
            pt = new DynBinDat(1000);
        if (trackE) {
            y_nor = new DynBinDat(100);
            z_nor = new DynBinDat(100);
        }
    }
    if (do_thr) {
        if (trackpt) 
            ptn = new DynBinDat(1000);
        if (trackE) {
            y_thr = new DynBinDat(100);
            z_thr = new DynBinDat(100);
        }
    }
}

maxfactor = 0;

cout << endl;
for (int i = 0; i < nos; i++) {
    if ((i * 100) % nos == 0) {
        cout << endl;
    }
    if (do_iss) {
        /* Do a 'normal' initial state shower. */
        do{
            delete iss;
            iss = new isshower(s_hat, x1, x2);
        } while (isshower::kinbreak);
    }
if(trackpt)  
iss->ptdist(pt);
if(trackE)  
iss->Edist(z_nor,y_nor);
if(i == nos-1){  
cout << "\nInitial state shower\n\n";
cout <<endl<< *iss << endl<<endl;
}  
delete iss;
iss = NULL;
}  

if(do_thr) {  
/* Do a threshold adapted initial state shower. */  
do{
   delete news;
   news = NULL;
z=selectz();
   news = new ishower(s_hat*pow2(1-z), x1, x2);
}  
while(ishower::kinbreak);
if(trackpt)  
news->ptdist(ptn);
if(trackE)  
news->Edist(z_thr, y_thr);
if(i == nos-1){  
cout << "\nThreshold shower:\n\n";
cout <<endl<< *news<<endl;
}  
delete news;
news = NULL;
}  

if(maxfactor>1) {  
if(redone) {  
cout<<"Encountering too large momentum fractions. Settings out";
cout<<" of scope of program.\n";
cout<<"Exiting program. Results may be unreliable because the ";
cout<<"overestimation in veto-\nalgorithm is not valid at all ";
cout<<"times.\n\n";
goto prints;
}  
delete pt;
pt = NULL;
delete ptn;
ptn = NULL;
cout << "Safety factor to low, increasing sfty from " << sfty;
sfty *= (4./3. * maxfactor - 1./3.);
cout << ", to " << sfty << endl;
redone = true;
}  
} while(maxfactor>1);
/* output statements */

// Switching working dir to output folder
stringstream ss;
ss << "mkdir " << output;
if(chdir(output.c_str())!=0) {
    system(ss.str().c_str());
    chdir(output.c_str());
}

// Generating output
if(do_iss && do_thr) {
    /*
     * Set the sizes of the bins for the data of the normal and the initial
     * state shower equal, so they can be compared straightforwardly.
     */
    if(trackpt)
        eqRange(pt, ptn);
    if(trackE) {
        eqRange(y_nor, y_thr);
        eqRange(z_nor, z_thr);
    }
}
if(do_iss) {
    if(trackpt)
        pt->plotSec("pt_nor");
    if(trackE) {
        y_nor->plotSec("y_nor");
        z_nor->plotSec("z_nor");
    }
}
if(do_thr) {
    if(trackpt)
        ptn->plotSec("pt_thr");
    if(trackE) {
        y_thr->plotSec("y_thr");
        z_thr->plotSec("z_thr");
    }
}

delete pt;
pt = NULL;
delete ptn;
ptn = NULL;

delete pdf;
pdf = NULL;
}

/*
 * Selectz() generates a 'z' according to the appropriate threshold resummed
 * distribution.
 *
 */
```cpp
double selectz() {
    long testseed;
    GetSeed(&testseed);
    if(testseed == DEFAULT)
        PutSeed(-1); // The seed -1 generates a seed depending on the time.

double z;
do
    z = Random();
    while(exp(-0.5*exp2(exp(log(1-z)))) < Random());
return z;
}

/*
* Print the current configuration to standard output.
*/
void outconf(void)
{
cout << endl;
cout << "a_s = " << as << endl;
cout << "tmin = " << tmin << endl;
cout << "s_hadron = " << stot << endl;
cout << "Q^2 = " << s_hat << endl;
cout << "generating " << nos << " events\n";
cout << "resolution = " << res << endl;
if(trackpt)
    cout << "tracking transverse momenta\n";
if(trackE)
    cout << "tracking energies\n";
if(do ISS && do thr)
    cout << "simulating normal and threshold shower\n";
else {
    if(do ISS)
        cout<< "simulating initial state shower\n";
    if(do thr)
        cout<< "simulating threshold shower\n";
}
cout << endl;
}

/*
The isshower constructor backwardly generates an initial state shower of
quark – antiquark pair coming from a proton – antiproton collision. Both
quark lines are evolved separately from scale tmax down to tmin.
The kinematics, however, are filled in later, where the order of kinematical
evaluation is equal to the decreasing order in virtuality.
*/
isshower::isshower(double tmax, double x1, double x2)
{
    isshower::kinbreak = false; // Track kinematical breakdown.
```
K = 0;

/*
 * Generate sequences of virtualities at which branchings should take place
 * for both quark lines.
 */
q1 = new isquark(tmax, x1, NULL);
q2 = new isquark(tmax, x2, NULL);

// Construct the kinematics for the two quarks entering the hard scattering
q1->construct(q2);
q2->construct(q1);

// Reset the history of rotations and boosts.
isparton::history.reset();

double t1, t2;
t1 = q1->current()->gett();
t2 = q2->current()->gett();

/*
 * Pick the (next) branching that is closest to the hard scattering and
 * construct the kinematics.
 */
while(t1 > 0 || t2 > 0) {
    if(t1 > t2) {
        if(q1->current()->mother->construct(q2->current())) {
            isshower::kinbreak = true; // Shower was kinematically incorrect,
            return; // abort.
        }
    }
    else {
        if(q2->current()->mother->construct(q1->current())) {
            isshower::kinbreak = true; // Shower was kinematically incorrect.
            return; // abort.
        }
    }

    /*
    * After constructing the kinematics of a branching, the system is
    * boosted and rotated to the COM-frame of the quarks in both quark
    * lines that are to become unresolved. This is done using the
    * RotBstMatrix class, as defined in vec4.h, from PYTHIA 8.1, and
    * applying that matrix to all 4-vectors.
    *
    */
    RotBstMatrix M; // declare matrix.
    // initialize matrix
    M.toCMframe(q1->current()->getp(), q2->current()->getp());
    // apply matrix to all other 4-momenta.
    RotnBoost(q1->current(), NULL, M);
    RotnBoost(q2->current(), NULL, M);
    // Update the history of boosts and rotations.
    isparton::history.robst(M);
/**
 * After the shower is fully unresolved, boost back to the COM frame of the
 * two colliding protons. This will mean that the orientation of the two
 * colliding partons, and thus that of the outgoing photon, will not be
 * trivial, and will not be oriented along the z axis.
 */

// Create the four vector of the protons
Vec4 pp1 = q1->current()->getp() / q1->current()->getx();
Vec4 pp2 = q2->current()->getp() / q2->current()->getx();
RotBstMatrix M; // declare boost matrix
M.toCMframe(pp1, pp2); // initialize boost matrix
RotnBoost(q1->current(), NULL, M); // apply matrix to all other 4-momenta.
RotnBoost(q2->current(), NULL, M);
isparton::history.rotbst(M); // Update the history of boosts and rotations.
}

/**
 * isshower destructor.
 */

isshower::~isshower()
{
    delete q1;
    delete q2;
    q1=q2=NULL;
}

/**
 * Returns the four momentum of the outgoing photon.
 */

Vec4 isshower::photon(void)
{
    return q1->getp()+q2->getp();
}

/**
 * Get the transverse momentum squared of all gluons in the shower and add
 * them to the supplied DynbinDat object to generate a distribution.
 */

void isshower::ptdist(DynBinDat *dat)
{
    q1->current()->ptdist(dat);
    q2->current()->ptdist(dat);
}

void isshower::Edist(DynBinDat *z, DynBinDat *y)
{
double maxE=0;
q1->current()->Edist(z, maxE);
q2->current()->Edist(z, maxE);

if (maxE>0)
    y->add(maxE);

/*
 * Print a textual representation of the initial state shower to the ostream os.
 */
ostream& operator << (ostream& os, const isshower& iss)
{
    isquark *temp = iss.q1->current();
    os << "Quark from proton\n";
    do{
        os << *temp << endl;
        if (temp->daughter != NULL)
            os << *temp->gdaughter << endl;
    } while ((temp = temp->daughter) != NULL);
    os<<"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n";
    os<<" Hard Scattering ........................................\n"
    os<<"Anti–quark from anti–proton\n";
    os<<"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"
    temp = iss.q2->current();
    do{
        os << *temp << endl;
        if (temp->daughter != NULL)
            os << *temp->gdaughter << endl;
    } while ((temp = temp->daughter) != NULL);
    return os;
}
A.2.3 isparton.h

/*
 * The class isparton is the base class of the subclasses isquark and isgluon.
 * These classes are made to generate an initial state parton shower.
 * Name: ispartons.h (header file of ispartons.cpp)
 * Author: Bart Verouden
 * Version: 1.0 – 21 May 2010
 * Uses: vec4.h (four–vector handling from Pythia)
 * bindat.h (binning tool)
 * Includes:
 * -- class isparton
 * -- class isquark (subclass of isparton)
 * -- class isgluon (subclass of isparton)
 */
#endif _ISPARTON_
#define _ISPARTON_

// Define pi if not yet done.
#ifdef M_PI
#define M_PI 3.1415926535897932385
#endif //M_PI

#include "isshower.h"
#include "vec4.h"
#include "bindat.h"
#include <iostream>
#include <vector>
using namespace std;

class isquark; // Reference to the isquark class needed for isparton.
class isgluon; // Reference to the isgluon class needed for isparton.

class isparton
{
    protected:
    Vec4 p; // The parton’s four momentum (px, py, pz, E)
    isquark *mother; // Pointer to the parton’s mother quark.

    static Vec4 p_1; // The 4–momentum of the final quark.
    static Vec4 p_2; // The momentum of the 'other' quark in the hard scattering.

    public:
    static bool kinbreak; // Boolean representing the kinematical breakdown.
    static RotBstMatrix history; // RotBstMatrix tracking the history of
    // the rotations and boosts made.
    static int gluoncount; // Keeps track of the total number of of gluons
    // emitted.

    /* Getters and setters */
Vec4 getp(void){\textbf{return} p;}
void setp(Vec4 pp){p=pp;}
Vec4 getp_2(void){\textbf{return} p_2;}

/\* Friends */
friend void RotnBoost(isquark *q, isgluon *g, RotBstMatrix M);
frend class isshower;
}

class isgluon : public isparton{
  private :
double tmax;  // The maximum virtual mass available for timelike shower.
  // (timelike showering off gluons not implemented)

public :
  /* Constructors */
isgluon(double tmax, isquark *mam);

  /* Friends */
friend ostream& operator<<(ostream&, \textbf{const} isgluon&);
}

class isquark : public isparton{
  private :
double x;  // The fractional COM energy (x*x_2*s=s^hat)
double z;  // The splitting variable
double t;  // t=-M^2>=0
double s;  // the COM energy of the subsystem (s^hat)

  /* private functions */
double selectt(double tmax);
void restart(void);
double zmax(void);
int construct(isquark *q2);

public :
  bool isresolved;
static int nob;  // The number of branches in the shower.
static double x2;  // the x variable of the 'other' hard parton in the hard scattering
isquark *daughter;  // The quarks daughter quark
isgluon *gdaughter;  // The quarks daughter gluon

static int noq;

  /* Constructors */
isquark(double tmax, double xx, isquark *d);
~isquark();

  /* Getters */
double gett(void){return t;}
double getz(void){return z;}
double getx(void){return x;}
double gets(void){return s;}

/* Setters */
void sett(double tt){t=tt;}
void setz(double zz){z=zz;}
void setx(double xx){x=xx;}
void sets(double ss){s=ss;}

/* Public functions */
void ptdist(DynBinDat *dat);
void Edist(DynBinDat *z, double &maxE);
isquark * current(void);
double pt2max(void);

/* Friends */
//friend ostream& operator<<(ostream&, const isshower&);
friend ostream& operator<<(ostream&, const isquark&);
friend class isshower;
};

#endif // _ISPARTON_
/ * This file contains the constructors and functions for the isparton class and
 * its subclasses.
 * Name: ispartons.cpp
 * Author: Bart Verouden
 * Version: 0.2 - 4 December 2009
 * Uses: isparton.h (header file)
 *       mstwpdf.h (mstw parton distribution functions)
 *       rng.h (random number generator)
 * Includes:
 *   - class isparton
 *   - class isquark (subclass of isparton)
 *   - class islgluon (subclass of isparton)
 *   - double A(double x, double zmax)
 *     : A represents the z-integral of the overestimation from x to zmax.
 *   - double G(double t, double x, double zmax)
 *     : G is the primitive of the overestimation of the t-integrand.
 *   - double Ginv(double y, double x, double zmax)
 *     : Ginv is the inverse of G.
 *
#include "isparton.h"
#include "mstwpdf.h"
#include "rng.h"
#include <cmath>
#include <vector>

double sfty = 1; // A safety factor to ensure that the overestimation of the
                 // exponent in the Sudakov form factor is indeed an
                 // overestimation. Should be adjusted according to s_hat and
                 // stot.

/*
 * Declaration of functions used in the veto-algorithm
 */
double A(double x, double zmax);
// double P(double z);
// double g(double t, double x, double zmax);
double G(double t, double x, double zmax);
double Ginv(double y, double x, double zmax);

/*
 * Constructors
 *
*/
/*
 * The isquark constructor requires as input the maximum virtuality, the x value
 * and a pointer to it's daughter quark (NULL if absent), and generates a quark
 * with t and z, 0<=-m^2<=t<zmax and x<z<zmax, by means of a veto-algorithm. If a
 * daughter quark was present, a (timelike) daughter gluon must be created as
 */
well with virtuality \( t_4 = m^2 - t_4 \text{ max} \). The gluon virtuality is set to zero here (so no timelike sideshowering is generated), but the maximum available virtuality is stored to check for kinematical consistency.

If the quarks generated (and accepted) \( t \) is non-zero, the (backward) showering continues, and a mother quark is created, following the same procedure. Hence the isquark constructor is the kernel of the backwards showering.

```c
isquark::isquark(double tmax, double xx, isquark *d) {
  extern c_mstwpdf *pdf;

  extern double maxfactor;

  z = 1.234;
  x = xx;
  t = tmax;
  daughter = d;
  s = 0;
  mother = NULL;
  gdaughter = NULL;
  isresolved = true;

  /* Keep track of number of branches in the shower. */
  int snob;
  if(daughter) {
    snob = nob;
    nob = snob + 1;
  } else {
    nob = 0; // (re)set the number of branches to zero.
    isparton::kinbreak = false; // reset kinematical breakdown boolean.

    /* Initialize the random number generator if this is the first shower generated. */
    long testseed;
    GetSeed(&testseed);
    if(testseed == DEFAULT)
      PutSeed(-1); // The seed -1 generates a seed depending on the time.
  }

  /* Veto part */
  double z_max = zmax();
  double test;
  do {
    t = selectt(t); // Pick a t.
    if(t == 0.) break; // No further showering allowed when t=0.
    double R = Random();
    /* We solve
```
\[ \int \frac{(as/2 \times P1)(1 + z_{\text{max}}^2)}{\sqrt{z'}} \times \frac{1 + z'}{1 - z'} \], \{ z', x, z \} = R \times A(x, z_{\text{max}}) \]

// Pick a z.
\[
z = \text{pow}^2(\tanh((1 - R) \times \text{atanh}(\sqrt{x}) + R \times \text{atanh}(\sqrt{z_{\text{max}}})));
\]

/*
* Veto if the actual z-integrand is smaller than a random fraction R of
* the overestimation.
*/
\[
test = (1 / \text{sfty}) \times ((1 + \text{pow}^2(z)) / (1 + \text{pow}^2(z_{\text{max}}))) \times \sqrt{z} \times \left( \frac{\text{pdf} \rightarrow \text{parton}(2, x / z, t)}{\text{pdf} \rightarrow \text{parton}(2, x, t)} \right);
\]
\[\text{maxfactor} = \text{fmax}(\text{maxfactor}, test);\]

} while (test < \text{Random}());

if (t > 0)
    mother = \text{new isquark}(t, x / z, \text{this}); // Create a new mother quark.
}

/*
* The destructor of the initial state quark.
*/

isquark::~isquark()
{
    delete mother;
    delete gdaughter;
    mother = NULL;
    gdaughter = NULL;
}

/*
* The isgluon constructor needs as input the maximum virtuality available for a
* timelike shower initiated for the gluon (which we won't do, but we store the
* virtuality available), and a pointer to the mother quark.
* The 4-momentum is set by the mother quark, so nothing needs to be done here
* except for storing the two input values.
*/

isgluon::isgluon(double tm, isquark *m)
{
    tmax = tm;
    mother = m;
}

/*
* The isquark::construct(isquark *q2) function fills in the kinematics of the
* isquark instance it is a member from. The isquark q2, passed by reference, is
* the quark from the other quark line that is still resolved with the highest
* virtuality. The system is thus in the COM frame of q2 and the daughter quark
* of the quark currently being constructed.
*/

int isquark::construct(isquark *q2)
extern double stot;

double t2 = q2->gett();
double x2 = q2->getx();
s = x * x2 * stot;

/* Fill in the kinematics */
double t4_max;
if(daughter == NULL) {
    // This is the situation where the quark partook in the hard scattering
    double pz = 0.5 * sqrt((pow2(s + t + t2) - 4 * t * t2) / s);
    double E = 0.5 * (s + t2 - t) / sqrt(s);
    if(!((q2->isresolved))
        pz = -pz;
    p=Vec4(0.,0.,pz,E);
} else {
    // Variables used in kinematical reconstruction
    double t1 = daughter->t;
    double s1 = daughter->z*s + t1 + t2;
    double s3 = s + t + t2;
    double r1 = sqrt(s1 * s1 - 4 * t2 * t1);
    double r3 = sqrt(s3 * s3 - 4 * t2 * t);

    // calculate maximum virtuality gluon.
    if(t2 == 0.) {
        t4_max = ((t1 / daughter->z) - t) * daughter->z*s *
            (1. / (daughter->z*s + t1) - 1. / (s + t));
    } else {
        t4_max = (s1 * s3 - r1 * r3) / (2 * t2) - t1 - t;
    }
    if(t4_max < 0.) {    // Kinematical breakdown. Note, and abort shower.
        isparton::kinbreak = true;
        return 1;    // Return value for kinematical failure.
    }

double E = 0.5 * (1. / sqrt((daughter->z*s))) * (s - t1 + t2);
    double pz = 0.5 * (1. / daughter->p.pz()) * (s3 - 2 * q2->getp().e()*E);
    double pt = sqrt(t4_max * (0.5 * (s1*s3 + r1*r3) - t2 * (t1 + t))
            / (r1*r1));
    double phi = 2.*M_PI*Random();    // Generate a random azimuthal angle phi.

    p = Vec4(pt * sin(phi), pt * cos(phi), pz, E);    // Set 4-momentum
    gdyughter = new isgluon(t4_max, this);    // Create new gluon
    gdyughter->setp(p - daughter->getp());    // Set it's 4-momentum
}
    isresolved = false;
    return 0;    // Return value for succes.
}

/*
  * isquark::select(tmax) returns a virtuality t<tmax, with probability distribu-
  * tion g(t)*Integrate(g(t'), {t', t, zmax})
*/
double isquark::selectt(double tmax)
{
    extern double tmin;
    double t = tmax, told, z_max = zmax();

    if(z_max < x)
        return 0;

    told = t; // Set the (new) upper limit for the allowed t
    do {
        t = Ginv(log(Random())+G(told, x, z_max), x, z_max); // Pick a t ....
    } while(t >= told);
    if(t < tmin) {
        return 0.;
    }
    return t;
}

/*
 * The isquark::current() function returns a pointer to the quark in the shower
 * closest to the hard scattering, that is still resolved.
 *
 */

isquark * isquark::current(void)
{
    if(isresolved)
        return daughter;
    else if(t > 0)
        return mother->current();
    else return this;
}

/*
 * isquark::zmax() returns the maximum value of z such that if z<zmax, the
 * energy of the gluon is larger than the resolution, res, in the C.O.M. frame
 * of the two quarks that partook in the hard scattering.
 *
 */

double isquark::zmax(void)
{
    extern double stot, res;
    double gamma = history.gamma();
    double xe = 2.*res/(sqrt(stot) * gamma);
    return(x / (x + xe));
}

/*
 * The isquark::ptdist(*dat) adds the absolute value squared of the transverse
 * momenta of the gluons in the shower to the DynBinDat object dat.
 *
 */
**void** isquark::ptdist(DynBinDat *dat)
{
    if (this->daughter != NULL) {
        dat->add(gdaughter->getp().pT2());
        daughter->ptdist(dat);
        ++gluoncount;
    }
    return;
}

/
* The isquark::Edist(*dat, &maxE) adds the absolute value squared of the
  energy of the gluons in the shower to the DynBinDat object z, and updates the
  maximum emitted energy maxE.
*/
**void** isquark::Edist(DynBinDat *dat, double &maxE)
{
    if (this->daughter != NULL) {
        maxE = fmax(maxE, gdaughter->getp().e());
        dat->add(gdaughter->getp().e());
        daughter->Edist(dat, maxE);
    }
    return;
}

/
* isquark::pt2max() returns the pt^2 of the gluon with the highest transverse
  momentum in this quark line.
*/
**double** isquark::pt2max(void)
{
    if (this->daughter != NULL) {
        return(fmax(gdaughter->getp().pT2(), daughter->pt2max()));
    } else
    return 0.0;
}

/
* RotnBoost(q, g, M) applies the matrix M to q and g, and to all q’s offspring.
*/
**void** RotnBoost(isquark *q, isgluon *g, RotBstMatrix M)
{
    q->p.rotbst(M);
    if (g != NULL)
        g->p.rotbst(M);
    if (q->daughter != NULL) {
        RotnBoost(q->daughter, q->gdaughter, M);
    }
Quark and gluon's output stream operators.

ostream& operator << (ostream& os, const isquark& q) {
os << "q: p = " << q.p << ", x = " << q.x << ", t = " << q.t << endl;
    return os;
}

ostream& operator << (ostream& os, const isgluon& g) {
os << "g: p = " << g.p << ", pt^2 = " << g.p.pT2() << endl;
    return os;
}

Functions needed for the veto–algorithm.

We overestimate the z–integrand appearing in the Sudakov form–factor:
\[ \frac{(1+z^2)/(1-z)}{(1/z)^{\frac{1}{2}}} \frac{pdf(x',t)}{pdf(x,t)} \] by
\[ \frac{(1+z_{\text{max}}^2)/(1-z)}{(1/z)^{\frac{1}{2}}} \frac{1}{\sqrt{z}} \].

"A" represents the z–integral of this overestimation from x to z_{max}.

double A(double x, double zmax)
{
    extern double as;
    return (sfty * (as / M_PI) * (1 + pow2(zmax)) * (atanh(sqrt(zmax)) -
    atanh(sqrt(x))));
}

P is the splitting function from the DGLAP equations.

double P(double z) {
    return ((1 + z * z) / (1 - z));
}

g is the overestimation of the t–integrand in the Sudakov form factor.

double g(double t, double x, double zmax)
{
    return ((1 / t) * A(x, zmax));
}

G is the primitive of the overestimation of the t–integrand, g.
double G(double t, double x, double zmax)
{
    return(A(x, zmax) * log(t));
}

double Ginv(double y, double x, double zmax)
{
    return(exp(y / A(x, zmax)));
}
A.3 Code MC@NLO toy model

The MC@NLO toy model texttt<MCNLO.c> and the extended model texttt<logterm.c> both use a pseudo random number generator \cite{27}. The extended toy model also uses the file texttt<rtsafe.c> from \cite{28}. Because the extended and the standard toy model are essentially the same we will only show the code of the extended model here.

A.3.1 MCNLO.h

```c
/* ----------------------------------------------------------
   * Header file for MCNLO.c and logterm.c
   * ----------------------------------------------------------
   */

// Arrays for data storage.
double NLO[50][2], MC[4][50][2], EXCL[4][50][2], INCL[4][50][2];
double UNWZ1[4][50][2], UNWZ2[4][50][2], UNWY1[4][50][2], UNWY2[4][50][2];
double YM[4][50][2], YN[4][50][2], YC[4][50][2];
double COUNTMC[4][50][2], COUNTINLO[4][50][2];

double Q(double x);
double R(double x);
double xmd(double x);
double G(double x, int nmbr);
double F1(double x);
double F2(double x);
void doMCy(double xmd, double *y);
void doMCz(double xmd, double *z);
void f(double z, double R, double xmd, double *f, double *df);
double rtsafe(void (*funcd)(double, double, double, double *, double *),
              double x1, double x2, double xacc, double R);
```
A.3.2 logterm.c

/*
 * This is essentially a copy of the file MCNLO.c but with 1/x replaced by 
 * (1-T*e log(x))/x as to include a log term. sigma is adjusted accordingly.
 * 
 * This program generates the data for the extended MC@NLO toy model.
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "rng.h"
#include "MCNLO.h"

double d=2, a =0.3, B =2, V =1, x0 =0.02, xdead =0.6, binsize , sigma;
int nbin = 50, nsim = 100000, j;

int main(void){
    int i,k,p, count1, count2, procrn=1;
    double x, y, z[20], temp;

    FILE *monte[4], *exc1[4], *incl[4], *unwy[4], *unwz[4], *ymc[4], *ynlo[4],
         *yct[4], *countmc[4], *countnlo[4], *nlo = fopen("nlo.dat", "wt");

    /*
    * Initialisation
    */
    sigma = B+(1./72.)*a*(570+241*d)*a*V;
binsize = (1-x0)/nbin;

    /* initializing files */
    monte[0] = fopen("monte1.dat", "wt");
    monte[1] = fopen("monte2.dat", "wt");

    exc1[0] = fopen("exc1.dat", "wt");
    exc1[1] = fopen("exc2.dat", "wt");

    incl[0] = fopen("incl1.dat", "wt");
    incl[1] = fopen("incl2.dat", "wt");

    unwy[0] = fopen("unwy1.dat", "wt");
    unwy[1] = fopen("unwy2.dat", "wt");

    unwz[0] = fopen("unwz1.dat", "wt");

    return 0;
}
unwz[1] = fopen("unwz2.dat", "wt");

ymc[0] = fopen("ymc1.dat", "wt");
ymc[1] = fopen("ymc2.dat", "wt");

ynlo[0] = fopen("ynlo1.dat", "wt");
ynlo[1] = fopen("ynlo2.dat", "wt");

yc[0] = fopen("yct1.dat", "wt");
cy[1] = fopen("yct2.dat", "wt");

countmc[0] = fopen("countmc1.dat", "wt");
countmc[1] = fopen("countmc2.dat", "wt");

countnlo[0] = fopen("countnlo1.dat", "wt");
countnlo[1] = fopen("countnlo2.dat", "wt");

/* initialize random generator */
PutSeed(-1);

/* initializing arrays */
for(j=0; j<4; j++)
    for(i=0; i<nbin; i++)
        NLO[i][j][1] = MC[i][j][1] = EXCL[i][j][1] = INCL[i][j][1] =
            UNWY1[i][j][1] = UNWY2[i][j][1] = UNWZ1[i][j][1] =
            UNWZ2[i][j][1] = YMC[i][j][1] = YMLO[i][j][1] =
            YCT[i][j][1] = COUNTMC[i][j][1] = COUNTNLO[i][j][1] = 0;
        NLO[i][0] = MC[i][0] = EXCL[i][0] = INCL[i][0] =
            UNWY1[i][0] = UNWY2[i][0] = UNWZ1[i][0] =
            UNWZ2[i][0] = YMC[i][0] = YMLO[i][0] =
            YCT[i][0] = COUNTMC[i][0] = COUNTNLO[i][0] =
            x0+(i+0.5)*binsize;

/* Monte Carlo 1 (Q(x) = 1) */
printf("simulation %d/11\n", procnr++);

for(i=0; i<nsim; i++)
    doMCy(1, &y);
    if(y>x0)
MC[0][(int)((y-x0)/binsize)][1]++;

/*
 * NLO hit and miss. Not necessary. We might as well just plot it. Hit and
 * miss was already very inefficient for R(y)/(y) because of its singularity
 * at y=0, but it gets even worse for (1-log(x))*R(x)/x.
 * printf("simulation %d/11\n", procnr++);
 * for (i=0; i<nsim; i++) {
 *   do {
 *     while ((y=Random()) < x0) {} } while (R(y)*(1-log(y))/(y*(1-log(x0))*(R(x0)/x0) <= Random ());
 *   NLO[(int)((y-x0)/binsize)][1]++;
 *}
 */

for (i=0; i<nbin; i++){
  y=NLO[i][0];
  NLO[i][1] = a*R(y)*(1-d*log(y))/y;
}

/* Renormalising MC and writing to files */
for(k=0; k<nbin; k++) {
  MC[0][k][1] = (sigma*nbin/nsim);
  fprintf(monte[0], "%f %f\n", MC[0][k][0], MC[0][k][1]);
  fprintf(nlo, "%f %f\n", NLO[k][0], NLO[k][1]);
}

/*
 * MC for all Q(x) with dead-zone at 0.6
 */
for(j=1; j<4; j++) {
  printf("simulation %d/11\n", procnr++);
  for (i=0; i<nsim; i++) {
    doMCy(1,&y);
    if(y > x0)
      MC[j][(int)((y-x0)/binsize)][1]++;
  }
  // renormalising, setting bin 0 to unity.
  for(k=nbin-1; k>=0; k--)
    MC[j][k][1] /= MC[j][0][1];
  // writing data to file
  for(k=0; k<nbin; k++)
    fprintf(monte[j], "%f %f\n", MC[j][k][0], MC[j][k][1]);
}

/*
 * MC@NLO for the exclusive variable y = max{ x1, x2, ..., xn } where x1...xn
 * are the emitted photons including the NLO photon.
 */
for(j=1; j<4; j++) {
  printf("simulation %d/11\n", procnr++);
  for (i=0; i<nsim; i++) {
    doMCy(1,&y);
    if(y > x0)
      MC[j][(int)((y-x0)/binsize)][1]++;
  }
  // renormalising, setting bin 0 to unity.
  for(k=nbin-1; k>=0; k--)
    MC[j][k][1] /= MC[j][0][1];
  // writing data to file
  for(k=0; k<nbin; k++)
    fprintf(monte[j], "%f %f\n", MC[j][k][0], MC[j][k][1]);
}
\[ x = \text{Random}(); \quad \text{// represents the NLO-photon} \]
\[ \text{doMCy} (xm(x), &\text{temp}); \]
\[ y = \text{max}(x, \text{temp}); \]
\[ \text{if}(x > x0 \&\& y > x0) \{ \quad \text{// add weight} \]
\[ \text{EXCL}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1] += F1(x); \]
\[ \text{if}(x=y) \{ \]
\[ \text{YNLO}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1] += F1(x); \]
\[ \text{COUNTNLO}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1]++; \]
\[ \} \quad \text{else} \{ \]
\[ \text{YMC}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1] += F1(x); \]
\[ \text{COUNTMC}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1]++; \]
\[ \} \]
\[ \text{doMCy}(1, &y); \quad \text{// counter term} \]
\[ \text{if}(x > x0 \&\& y > x0) \{ \quad \text{// add counter-weight} \]
\[ \text{EXCL}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1] += F2(x); \]
\[ \text{YCT}[j][\lfloor \text{int} \rfloor((y-x0)/\text{binsize})][1] += F2(x); \]
\[ \} \]
\[ \} \]
\[ \} \]
\[ \} \quad \text{// writing data to files} \]
\[ \text{for}(k=0; k<n\text{bin}; k++) \{ \]
\[ \text{EXCL}[j][k][1] += (\text{double})n\text{bin}/n\text{sim}; \]
\[ \text{YMC}[j][k][1] += (\text{double})n\text{bin}/n\text{sim}; \]
\[ \text{YNLO}[j][k][1] += (\text{double})n\text{bin}/n\text{sim}; \]
\[ \text{YCT}[j][k][1] += (\text{double})n\text{bin}/n\text{sim}; \]
\[ \}
\[ \]
if(x>x0) { 
  // add the NLO photon weighted to the data.
  INCL[j][(int)((x-x0)/binsize)][1] += F1(x);
}
doMCz(1,z);
k=0;
while(z[k]>x0) { 
  // add the MC counter-term weighted to the data.
  INCL[j][(int)((z[k]-x0)/binsize)][1] += F2(x);
  k++;
}
/* writing data to files */
for(k=0; k<nbin; k++) {
  INCL[j][k][1] *= (double)nbin/nsim;
  fprintf(incl[j], "%f %f\n", INCL[j][k][0], INCL[j][k][1]);
}
return 0;

/* Q characterizes the sudakov form-factor in the MC generator. */

double Q(double x) {
  switch (j) { // j should be between 0 and 3 to distinguish the different
    // choices for Q(x). j is defined globally.
    case 0:
      return 1;
    case 1:
      if(xdead - x > 0) return 1;
    case 2:
      if(xdead - x > 0) return(G(x/xdead,1));
    case 3:
      if(xdead - x > 0) return(G(x/xdead,2));
    }
  return 0;
}

double R(double x) {
  return (B + x * (1 + .5*x + 20*x*x));
}

double xn(double x) {
  return (1-x);
}

double G(double x, int nmb) {
  if(nmb == 1)
    return(pow(1-x, 2)/(x*x + pow(1-x, 2)));
  return(64*pow(1-x, 2)/(pow(x, 4) + 64*pow(1-x, 2)));
}

double F1(double x)
{  return (a*(R(x)-B*Q(x))*(1-d*log(x))/x); }

double F2(double x)
{  return (B+a*(V+B*(Q(x)-1)*(1-d*log(x))/x)); }

/**
 * MC simulation for the variable y. Since the MC photon's are strictly
 * decreasing in energy, only the first photon is a candidate to be the hardest
 * photon. Hence we stop after one branching. Uses newton's method rtsafe to find
 * root.
 */
void doMCy(double xm, double *y) {
  while(Q(*y = rtsafe(f, x0,xm, 1E-4,Random())) < Random())
    xm = *y;
  return;
}

/**
 * MC simulation for the variable z. Here we need the full distribution, so we
 * let the system to keep on branching until it generates a photon with energy
 * less than x0. Uses newton's method rtsafe to find root.
 */
void doMCz(double xm, double *z) {
  int i=0;
  while((xm > x0 && i<19) {  // we accept a maximum of 19 branchings.
    while(Q(z[i] = rtsafe(f, x0,xm, 1E-4,Random())) < Random())
      xm = z[i];
    xm=z[i];
    i++;}
  // resetting all unused positions in the array, so that no old data is used.
  while(i<19 && z[i] != 0) {
    z[i]=0;
    i++;
  }
  return;
}

void f(double z, double R, double xm, double *f, double *df) {
  *f = exp(-a*(-log(xm/z) + .5*d*(-pow(log(xm),2)+pow(log(z),2)))) - R;
  *df = (*f)*a*(1-d*log(z))/z;
  return;
}
A.4 Additional Code

A.4.1 bindat.h

/*
 * The BinDat and DynBinDat class provides a tool to automatically and easily
 * bin and manipulate generated data. The BinDat constructor requires as input
 * a minimum and maximum value, as well as the number of required bins, whereas
 * the DynBinDat constructor only requires the number of bins as an input. Since
 * DynBinDat rebins the data when an input value falls out of the current
 * minimum-maximum range, the dynamic version is somewhat slower, which might
 * become a problem for large sets of data.
 *
 * BinDat allows you to
 * -- add data without actively having to bin it. (add)
 * -- get the vector of bins as an output (get)
 * -- generate a text file with frequencies (out)
 * -- generate and run a gnuplot file of the data (plot)
 * All output methods (get, out and plot) can be called returning either the
 * plain frequencies, or a normalized distribution by adding the "Norm" resp.
 * "Norm" suffix.
 *
 * Name: BinDat.h
 * Author: Bart Verouden
 * Version: 1.3 – 28 April 2010
 *
 * Includes:
 * -- class BinDat
 * -- class DynBinDat (subclass of BinDat)
 *
 */

#ifndef BINDAT
#define BINDAT
#include <vector>
#include <string>
#include <fstream>
using namespace std;

class BinDat
{
    protected:
        vector<int> data;
        double binsize;
        int nentries;
        double xmin, xmax;
        int nbins;
        int missed;

        // Default constructor – Should not manually be called, hence protected.
        BinDat(void){nentries = 0; xmin=xmax=0;}

    private:
        ofstream output;
};
public:
   // Constructor
   BinDat(int nbins, double xmi, double xma);

   /* public functions */
   void add(double x);
   void outSec(string filename);
   void outNorm(string filename);
   void outCum(string filename);
   void outWeighted(string filename, double (*weight)(double));
   void outSmooth(string filename, int n);
   vector<int> getSec(void);
   vector<double> getNorm(void);
   vector<int> getCum(void);
   vector<double> getWeighted(double (*weight)(double));
   vector<double> getSmooth(int n);
   vector<double> getEntries(void){return ddata; } /* Return the original input as a vector. */
   void rescale(int nbins, double xmi, double xma);

   /* Friends */
   friend void eqRange(DynBinDat *, DynBinDat *);
};
A.4.2  bindat.cpp

/*
 * This file contains the constructors and functions for the BinDat class and
 * its subclass DynBinDat. For more info see the header file.
 * Name: BinDat.cpp
 * Author: Bart Verouden
 * Version: 1.3 – 28 April 2010
 * Uses: header file
 *
 * Includes:
 * – class BinDat
 * – class DynBinDat (subclass of BinDat)
 */
#include "bindat.h"
#include <iostream>
#include <sstream>
#include <cmath>

/*
 * Constructor – The BinDat constructor requires a range of of x-values, and
 * the number of bins
 */
BinDat::BinDat(int nbins, double xmi, double xma)
{
    xmin=xmi; xmax=xma;
    binsize = (xmax-xmin)/nbins;
    data.assign(nbins,0);
    nentries=0;
    missed = 0;
}

/*
 * Add 1 to the bin where x falls in.
 */
void BinDat::add(double x)
{
    if(x>xmax || x<xmin) {
        ++missed;
        return;
    }
    if(x==xmax)
        data.at(nbins-1)++;
    else
        data.at((int)((x-xmin)/binsize))++;
    nentries++;
}

/*
 * Write the data to the file <filename>
 */

void BinDat::outSec(string filename)
{
    output.open(filename.c_str());
    if(!output.is_open()) {
        cout << "Could not open file: " << filename;
        cout << " , check for write protection.\n";    }
    else
        for(int i=0;i<(int)data.size();i++) {
            output <<(0.5+i)*binsize+xmin<<" \n";
            output << data.at(i)<<endl;
        }
    output.close();
}

/*
* Normalize the data such that (number of entries) x binsize = 1, and write the
* normalized data to the file <filename>
*
*/

void BinDat::outNorm(string filename)
{
    output.open(filename.c_str());
    if(!output.is_open()){
        cout << "Could not open file: " << filename;
        cout << " , check for write protection.\n";    }
    else {
        int temp =0;
        for(int i=0;i<(int)data.size();i++) {
            output <<(0.5+i)*binsize+xmin<<" \n";
            output <<(double)(data.at(i)/((double)entries*binsize))<<endl;
        }
    }
    output.close();
}

/*
* Write the cumulative data to the file <filename>
*
*/

void BinDat::outCum(string filename)
{
    output.open(filename.c_str());
    if(!output.is_open()){
        cout << "Could not open file: " << filename;
        cout << " , check for write protection.\n";    }
    else {
        int temp =0;
        for(int i=0;i<(int)data.size();i++) {
            output <<(0.5+i)*binsize+xmin<<" \n";
            output <<(temp+=data.at(i))<<endl;
        }
    }
    output.close();
}
* Write the weighed data to the file <filename>
*
*/
void BinDat::outWeighted(string filename, double (*weight)(double))
{
    output.open(filename.c_str());
    if(!output.is_open()){
        cout << "Could not open file: " << filename;
        cout << " , check for write protection.\n";
    } else {
        for(int i=0; i<(int)data.size(); i++) {
            double x = (0.5+i)*binsize+xmin;
            output << x << " " << data.at(i) * weight(x) << endl;
        }
    }
    output.close();
}

/*
* Write the data to the file <filename> where each bin is averaged with its n neighbors on either side.
*
*/
void BinDat::outSmooth(string filename, int n)
{
    output.open(filename.c_str());
    if(!output.is_open()){
        cout << "Could not open file: " << filename;
        cout << " , check for write protection.\n";
    } else {
        for(int i=0; i<(int)data.size(); i++) {
            double x = (0.5+i)*binsize+xmin;
            output << x << " " << output/d << endl;
        }
    }
    output.close();
}

/*
* Return the data as a vector.
*
*/
vector <int> BinDat::getSec( void ) { return data; }

/*
* Normalize the data such that (number of entries) x binsize = 1, and return * the normalized data as a vector.
* 
*/
vector <double> BinDat::getNorm(void)
{
    vector <double> result;

    for(int i=0; i<(int)data.size(); ++i)
        result.push_back((double)(data.at(i)/((double)nentries*binsize)));
    return result;
}
/
* Return the cumulative data as a vector.
* 
*/
vector <int> BinDat::getCum(void)
{
    vector <int> result;
    int temp=0;
    for(int i=0; i<(int)data.size(); ++i)
        result.push_back((temp+=data.at(i)));
    return result;
}
/
* Return the weighted data as a vector.
* 
*/
vector <double> BinDat::getWeighted(double (*weight)(double))
{
    vector <double> result;
    for(int i=0; i<(int)data.size(); ++i) {
        double x = (0.5+i)*binsize+xmin;
        result.push_back(data.at(i) * weight(x));
    }
    return result;
}
/
* Return the data as a vector, where each bin is averaged with its n * neighbors on either side.
* 
*/
vector <double> BinDat::getSmooth(int n)
{
    vector <double> result;

    for(int i=0; i<(int)data.size(); ++i) {
        double out=0;
        int d=0;
        for(int j=-n; j<=n; ++j) {
            if(i+j>=0 && i+j<(int)data.size()) {
                

128
```cpp
out += data.at(i+j);
++d;
}
result.push_back(out/d);
}
return result;
}

/*
* Write the data to the file <name>.dat, generate a gnuplot file <name>.plot
* and run gnuplot on that file creating a plot <name>.eps
*/

void BinDat::plotSec(string name)
{
    stringstream datfile, plotfile, command;

datfile << name << " . dat " ;
plotfile << name << " . plot " ;
command << " gnuplot " << plotfile.str();

outSec(datfile.str().c_str());
ofstream plot(plotfile.str().c_str());
if(plot.is_open()){
    plot << " set style data histeps \n " ;
    plot << " set term postscript eps enhanced color \n " ;
    plot << " set output " " " " " " << name << " . eps \n " ;
    plot << " plot " " " " << datfile.str() " " " " w lines \n " ;
    plot.close();

    system(command.str().c_str());
} else {
    cout << " Could not open file : " << plotfile.str().c_str();
    cout << " , check for write protection \n " ;
}
}

/*
* Normalize the data such that (number of entries) x binsize = 1, write the
* normalized data to the file <name>.dat, generate a gnuplot file <name>.plot
* and run gnuplot on that file creating a plot <name>.eps
*/

void BinDat::plotNorm(string name)
{
    stringstream datfile, plotfile, command;

datfile << name << " . dat " ;
plotfile << name << " . plot " ;
command << " gnuplot " << plotfile.str();

outNorm(datfile.str().c_str());
ofstream plot(plotfile.str().c_str());
```
if (plot.is_open()) {
    plot << "set style data histeps\n";
    plot << "set term postscript eps enhanced color\n";
    plot << "set output " << name << ".eps\n";
    plot << "plot " << datfile.str() << " w lines\n";
    plot.close();
    system(command.str().c_str());
} else {
    cout << "Could not open file: " << plotfile.str().c_str();
    cout << ", check for write protection.\n";
}

/*
 * Write the cumulative data to the file <name>.dat, generate a gnuplot file
 * <name>.plot and run gnuplot on that file creating a plot <name>.eps
 *
*/
void BinDat::plotCum(string name) {
    stringstream datfile, plotfile, command;
    datfile << name << ".dat";
    plotfile << name << ".plot";
    command << "gnuplot " << plotfile.str();
    outCum(datfile.str().c_str());
    ofstream plot(plotfile.str().c_str());
    if (plot.is_open()) {
        plot << "set style data histeps\n";
        plot << "set term postscript eps enhanced color\n";
        plot << "set output " << name << ".eps\n";
        plot << "plot " << datfile.str() << " w lines\n";
        plot.close();
        system(command.str().c_str());
    } else {
        cout << "Could not open file: " << plotfile.str().c_str();
        cout << ", check for write protection.\n";
    }
}

/*
 * Write the weighted data to the file <name>.dat, generate a gnuplot file
 * <name>.plot and run gnuplot on that file creating a plot <name>.eps
 *
*/
void BinDat::plotWeighted(string name, double (*weight)(double)) {
    stringstream datfile, plotfile, command;
    datfile << name << ".dat";
    plotfile << name << ".plot";
    command << "gnuplot " << plotfile.str();
outWeighted(datfile.str().c_str(), weight);
ofstream plot(plotfile.str().c_str());
if(plot.is_open()){
    plot<<"set style data histeps n";
    plot<<"set term postscript eps enhanced color n";
    plot<<"set output "<<name<<".eps n";
    plot<<"plot """<<datfile.str()<<" w lines n";
    plot.close();
}
else {
    cout << "Could not open file: " << plotfile.str().c_str();
    cout <<", check for write protection.\n";
}
}

/*
 * Write the data to the file <name>.dat, where each bin is averaged with its n
 * neighbors on either side, generate a gnuplot file <name>.plot and run gnuplot
 * on that file creating a plot <name>.eps.
 */

void BinDat::plotSmooth(string name, int d)
{
    stringstream datfile, plotfile, command;

    datfile <<name<<".dat";
    plotfile <<name<<".plot";
    command <<"gnuplot """<<plotfile.str();

    outSmooth(datfile.str().c_str(), d);
ofstream plot(plotfile.str().c_str());
    if(plot.is_open()){
        plot<<"set style data histeps n";
        plot<<"set term postscript eps enhanced color n";
        plot<<"set output """<<name<<".eps n";
        plot<<"plot """<<datfile.str()<<" w lines n";
        plot.close();
    }
    else {
        cout << "Could not open file: " << plotfile.str().c_str();
        cout <<", check for write protection.\n";
    }
}

/*
 * Functions for the dynamic subclass DynBinDat
 */

/*
 * Add one to the bin where x belongs. If x is out of the current range
 */
* $(x_{\text{min}}, x_{\text{max}})$, adjust range, and update current bin-vector data, before adding $x$.
* 
*/

```cpp
void DyBinDat::add(double x)
{
    if (ddata.size() == 0) {
        xmin = xmax = x;
        ddata.push_back(x);
    } else if (x == xmin && xmin == xmax)
        ddata.push_back(x);
    else {
        if (x < xmin)
            xmin = x;
        binsize = (xmax - xmin) / nbins;
        refill();
    } else if (x > xmax)
        xmax = x;
    binsize = (xmax - xmin) / nbins;
    refill();
    this->BinDat::add(x);
    ddata.push_back(x);
}
entries = ddata.size();
}
```

/* Bin all elements from the input history ddata into data. This is a private
 * function which is meant to be called after a rescaling of the bin-range.
 * */

```cpp
void DyBinDat::refill(void)
{
    vector<double>::iterator it;
    data.assign(nbins, 0);

    for (it = ddata.begin(); it < ddata.end(); it++) {
        if (*it == xmax)
            data.at(nbins - 1)++;
        else
            data.at((int)((*it - xmin) / binsize))++;
    }
}
```

/* Rescale the DyBinDat object to the new values of $x_{\text{min}}$, $x_{\text{max}}$ and nbins. The
 * new range must enclose the old range to prevent existing data to fall out of
 * the new range.
 * */

```cpp
void DyBinDat::rescale(int nb, double xmi, double xma)
{
    if (xmi > xmin || xma < xmax) {
```
```
cout << "Incorrect rescaling: Data out of new bounds.\n";
    return;
}
if (nb<=0) {
    cout << "Incorrect rescaling: Invalid new number of bins\n";
    return;
}
nbins = nb;
xmin = xmi;
xmax = xma;

binSize=(xmax - xmin) / nbins;

refill();
}

/*
 * eqRange(DynBinDat *a, DynBinDat *b) rescales the two DynBinDat objects such
 * that they have the same range and number of bins. The maximum of the two
 * number of bins is taken for the rescaled versions, and no data may fall out
 * of the new ranges, strictly increasing the ranges.
 */
void eqRange(DynBinDat *a, DynBinDat *b) {
    double xmin = fmin(a->xmin, b->xmin);
    double xmax = fmax(a->xmax, b->xmax);
    int nbins = max(a->nbins, b->nbins);
    a->rescale(nbins, xmin, xmax);
    b->rescale(nbins, xmin, xmax);
}
A.4.3 vec4d.h

/*
 * Name: vec4d.h (header file of vec4d.cpp)
 * Author: Bart Verouden
 * Version: 1.0 - 14 October 2009
 *
 * Includes:
 * - class vec4d
 *
*/

#ifndef VEC4D_H
#define VEC4D_H

class vec4d{
public:
    double E;
    double x;
    double y;
    double z;

    /*
     * Constructors and destructors
     */
    vec4d();
    vec4d(double E, double p);
    vec4d(double E, double px, double py, double pz);
    ~vec4d();

    /*
     * Operators
     */
    vec4d & operator=(const vec4d &p);
    const vec4d operator+( const vec4d & p);
    const vec4d operator-( const vec4d & p);
    const vec4d operator*(const double r);
    const double operator*(const vec4d p);

    /*
     * General public functions
     */
    double abs2(void){ return (*this)*(*this); }
};

vec4d operator*(double r, vec4d & p);

#endif
A.4.4 vec4d.cpp

/*
 * This file contains the constructors and functions for the vec4d class.
 * Name: vec4d.cpp
 * Author: Bart Verouden
 * Version: 1.0 - 14 October 2009
 * Uses: vec4d.h
 *
 * Includes:
 * - class vec4d
 *
*/
#include "vec4d.h"

/*
 * Constructors and destructors
 *
*/

/*
 * The default constructor initializes all values at zero.
 */
vec4d::vec4d()
{
    E=0.;
    x=0.;
    y=0.;
    z=0.;
}

/*
 * The vec4d(E, pz) constructor creates the 4-vector (E,0,0,p).
 */
vec4d::vec4d(double E, double pz)
{
    this->E=E;
    x=0.;
    y=0.;
    z=pz;
}

/*
 * The vec4d(E, px, py, pz) constructor creates the 4-vector (E,px,py,pz).
 */
vec4d::vec4d(double E, double px, double py, double pz)
{
    this->E=E;
    x=px;
    y=py;
    z=pz;
}
The destructor.

```cpp
vec4d::~vec4d()
{
}
```

Operators

* The '=' operator sets all elements of the lhs 4-vector equal to the elements of the rhs 4-vector.

```cpp
vec4d & vec4d::operator=(const vec4d &p)
{
    if (this != &p) {
        E = p.E;
        x = p.x;
        y = p.y;
        z = p.z;
    }
    return *this;
}
```

* The '+' operator.

```cpp
const vec4d vec4d::operator+(const vec4d &p)
{
    return vec4d(E + p.E, x + p.x, y + p.y, z + p.z);
}
```

* The '-' operator.

```cpp
const vec4d vec4d::operator-(const vec4d &p)
{
    return vec4d(E - p.E, x - p.x, y - p.y, z - p.z);
}
```

* The '×' operator returns a 4-vector of which all elements are multiplied by r.

```cpp
const vec4d vec4d::operator*(const double r)
{
    return vec4d(r * E, r * x, r * y, r * z);
}
```
*vec4d p * vec4d q returns the Minkovskian dot product of 4-vectors p and q.*

```cpp
const double vec4d::operator*(vec4d p)
{
    return E*p.E -x*p.x -y*p.y -z*p.z;
}

/*
* double r * vec4d p = p * r.
*/
vec4d operator*(double r, vec4d& p)
{
    return p*r;
}
```
A.4.5  veto_toy.c

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "veto_toy.h"
#include "rng.h"
#define PI 3.1415926535897932385

/*
 * This program gives a toy model for the Veto Algorithm as described in Pythia 6.4
 * Physics and Manual.
 */
double tmmin=0, tol, tmax=1, t, tot=0, ttest, scale=4, z, zold, n;
int nhits=100000, nbins=100;

int main(void)
{
    int i, j, k, dat[101];
    double d, R, r, test, a[20];
    FILE *data1= fopen("data1.dat", "wt"), *data2= fopen("data2.dat", "wt"),
         *data3= fopen("data3.dat", "wt");

    //initializing the data
    for(i=0; i<nbins; i++)
    {
        dat[i]=0;
    }

    //for (i=0; i<7; i++) a[i]=pow(2, i-1);
    for(i=0; i<20; i++) a[i]=.25+.5*(i);
    j=k=0;
    /* initialize random generator */
    PutSeed(-1);

    /*
     * First we are going to do a simple decay algorithm for a function g(t)
     * which has a known primitive that has a well defined inverse. To test that
     * this algorithm works we are going to do nhits=100000 calculations and see
     * what fraction 'r' of the resulting t-values are smaller than ttest. If
     * our code is correct we should see that r is to a good approximation equal
     * to the theoretical value 1-exp(-ttest*ttest/2).
     *
     * As an additional test we are going to see what the average is of all
     * generated numbers (we expect this to be sqrt(0.5*pi)) and see if the
     * deviation is of the order of the standard deviation sqrt((2-0.5*pi)/N).
     *
     * A more visual test is to devide our region of t's into bins and count
     * how many of the generated t's belong in each bin. If we normalize and
     * plot this we can compare the graph with the expected distribution.
     */
    ttest = Random();
    printf("t\_test = %f
\n", ttest);
printf("\nSimple decay algorithm for g(t) = t\n");

test = 0;
for(i=0;i<nhits;i++){
  R=Random();
  t=Ginv(G(tmin)−log(R));
  if(t<scale) //refusing t larger then our set scale
    dat[(int)(nbins/scale*t)]++; //save data
  if(t<test)
    k++;
  test = test+t/nhits;
}

/*
 * Writing data to file. To normalize we must first devide by nhits*dt =
 * nhits*scale/nbins
 */
for(i=0;i<=nbins;i++){
  fprintf(data1,"%f %f\n",(double)i*scale/nbins,
            (double)dat[i]*nbins/(nhits*scale));
  dat[i]=0;
}

r=(float)k/nhits;
printf("Simulation: left=%f, right=%f \n",r,1−r);
printf("Theoretical: left=%f, right=%f \n", 1−exp(−test*test/2),
         exp(−test*test/2));
printf("\nExpected average: %f, actual average: %f\n",sqrt(0.5*PI),test);
printf("Standard deviation: %f, deviation from average: %f\n",
         sqrt((2−0.5*PI)/nhits), fabs(sqrt(0.5*PI)−test));

k=0;
test=0;

/*
 * Now suppose g(t) was not nice enough, i.e. it has no primitive that has
 * a well-defined inverse. Now we can try to find a function h(t) that has a
 * primitive with a well-defined inverse, such that h(t) >= g(t) for t >= 0.
 * If we find such a function we can use the veto algorithm. First we are
 * again going to calculate 100.000 t-values and see if the ratio r matches
 * the theory again. Then we will again look at the standard deviation of
 * the data. At last we will make another plot. We are going to pretend that
 * g(t) = t is not nice enough, but h(t) = t^2 + a is, and since h(t)>=g(t)
 * if a>1/4 we can use the veto algorithm.
 */
printf("\nVeto algorithm for t^2+a > t for t>0\n");
for(i=0;i<nhits;++i){
  t=j=0;
  do {
    told = t;
    j++;
    do {
      t = Hinv(H(told,a[0])−log(Random()),a[0]);
    } while (t <= told);
  } while (g(t)/h(t,a[0])<= Random());
  if(t<scale) //refusing t larger then our set scale
    dat[(int)(nbins/scale*t)]++; //save data
  if(t<test)
    k++;
}

test = 0;

/* Now suppose g(t) was not nice enough, i.e. it has no primitive that has
 * a well-defined inverse. Now we can try to find a function h(t) that has a
 * primitive with a well-defined inverse, such that h(t) >= g(t) for t >= 0.
 * If we find such a function we can use the veto algorithm. First we are
 * again going to calculate 100.000 t-values and see if the ratio r matches
 * the theory again. Then we will again look at the standard deviation of
 * the data. At last we will make another plot. We are going to pretend that
 * g(t) = t is not nice enough, but h(t) = t^2 + a is, and since h(t)>=g(t)
 * if a>1/4 we can use the veto algorithm.
 */
printf("\nVeto algorithm for t^2+a > t for t>0\n");
k++;  
test = test + t/nhits;  
}  

/*  
 * Writing data to file. To normalize we must first divide by nhits*dt =  
 * nhits*scale/nbins  
 */  
for(i=0;i<=nbins;i++){  
    fprintf(data2,"%f %f \n",(double)i*scale/nbins,  
            (double)dat[i]*nbins/(nhits*scale));  
    dat[i]=0;  
}  

printf("Simulation: left= %f, right= %f \n",(float)k/nhits,1-(float)k/nhits);  
printf("Theoretical: left= %f, right= %f \n",1-exp(-ttest*ttest/2),  
       exp(-ttest*ttest/2));  
printf("\nExpected average: %f, actual average: %f \n",sqrt(0.5*PI),test);  
printf("Standard deviation: %f, deviation from average: %f \n",  
       sqrt((2-0.5*PI)/nhits),fabs(sqrt(0.5*PI)-test));  

/*  
 * Now we are going to look at how the efficiency changes as h(t) moves  
 * further away from g(t). As a measure of the efficiency we are going to  
 * use the number of calls made to Random()  
 *  
 * This test is relatively time consuming, and hence commented out. The test  
 * results in a linear relation between the number of calls and a, for  
 * large a. More interesting results might be found for real singular  
 * functions when h is very close to g.  
 *  
 * printf("\nEfficiency test \n");  
 */  
for(j=0;j<20;j++){  
    printf("\n");  
    n=0;  
    for(i=0;i<nhits;++i){  
        t=0;  
        do {  
            n++;  
            told = t;  
            do {  
                t = Hinv(H(told,a[j])-log(Random()),a[j]);  
                n++;  
            } while (t <= told);  
        } while (g(t)/h(t,a[j])<= Random());  
    }  
    fprintf(data3,"%f %e \n",a[j], n);  
}  

/*  
 * Now we do a simple particle branching simulation where the distribution  
 * of z is x^2 on the interval (0,1)  
 */  
printf("\n\nThe next simulation gives a list of (t,z) branchings up until\n");  
fpprintf("1\n\nit reaches the scale = %.1f\n\n", scale);
t = told = 0;
z = 1;
while (t <= scale)
{  
    printf("parton is at (%f, %f)\n", t, z);
    do
    {  
        told = t;
        do
        {  
            t = Hinv(H(told,1) - log(Random()), 1);
        } while (t <= told);
        do
        {  
            t = Hinv(H(told,1) - log(Random()), 1);
        } while (g(t)/h(t,1) <= Random());
        z = pow(3*(Random()*(1./3.)), 1./3.)*z;  // Some arbitrary distribution to // choose a new z(<z_old))
    }  
    return 0;
}

/*  
* g(t) original decay probability  
*/
double g(double t) {
    return t;
}

/*  
* the primitive of g(t)  
*/
double G(double t) {
    return t*t/2;
}

/*  
* the inverse of the primitive of g(t)  
*/
double Ginv(double x) {
    return pow(2*x, 1./2.);
}

/*  
* h(t) = t^2+a >= g(t) for a >= 1/4  (t>0)  
*/
double h(double t, double a) {
    return(t*t+a);
}

/*  
* the primitive of h(t)  
*/
double H(double t, double a) {
    return(t*t*t/3 + a*t);
}

/*  
* the inverse of the primitive of h(t)  
*/
double Hinv(double x, double a) {
    return (-pow(2*a*a*a/(3*x + sqrt(4*a*a*a +9*x*x)), 1./3.)
            + pow((3*x+sqrt(4*a*a*a+9*x*x))/2, 1./3.));
}
References


[26] PHYTIA 8.1
Source code available at http://home.thep.lu.se/~torbjorn/Pythia.html

[27] S. Park and D. Greyer, Pseudo random number generator.

http://www.nrbook.com/a/bookcpdf.php

[29] R. Wagnerr, “Configuration File Reader for C++”
http://www-personal.umich.edu/~wagnerr/ConfigFile.html