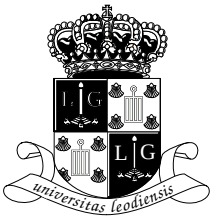
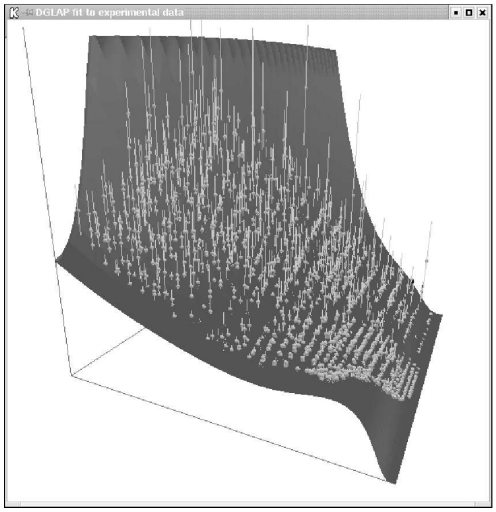


Université de Liège
Faculté des Sciences
Département de Physique

Année académique 2001-2002

The DGLAP Evolution Equation

Analytical properties and Numerical resolution



G. Soyez

Mémoire présenté en vue
de l'obtention du
DEA en Sciences Physiques

Chapter 1

Introduction

As soon as QCD was born [1–5], theoretical physicists faced the problem of computing physical quantities in strong interaction processes. Actually, the **asymptotic freedom** [4, 6] of QCD, telling that the coupling constants goes to zero as the momentum scale goes to infinity, implies a pole at small scale (of order 220 MeV), known as the **Landau pole**. This means that the perturbative theory breaks down for soft particles.

In this document, we will attach ourselves to the QCD implications in **Deep Inelastic Scattering** (DIS) i.e. $ep \to eX$ collisions. Experimental measurements of the F_2 structure function through these collisions[1] are available from various experiments around the world [7–18]. The most important source of F_2 values are certainly the HERA experiments (ZEUS [10,11] and H1 [7–9]). Their latest sets of experimental data reached a level of precision very useful for theoretical tests.

In this type of collision, when we compute the proton structure function in the QED framework, by a single photon exchange, we obtain the well known **Bjorken scaling** [19]: the structure functions are independent of the virtuality Q^2 of the exchanged photon. If we compute QCD corrections in the large Q^2 limit, we meet collinear divergences, appearing through $\log(Q^2/\mu^2)$ corrections [20]. This is known as **scaling violation**. Dealing with these divergences is a typical example where resummation of perturbation theory is to be considered.

The scaling violation implies that parton densities inside the proton become Q^2-dependent and the resummation of the collinear divergences in the framework of perturbative QCD leads to Q^2 evolution equations for these parton densities. These equations, called the DGLAP[2] equations [20], were found independently by Gribov-Lipatov in 1972, Altarelli-Parisi and Dokshitzer in 1977. Due to their very good agreement with experiment [21], the DGLAP equations are considered as one of the best successes of pQCD.

We will start this report by establishing the DGLAP equations. This will be done in two steps. The first one shows how to deal with collinear divergences and how resummation of the perturbative expansion and reabsorbtion of these divergences in physical

[1] Some experiments, like E665 [15], use muons instead of electrons.

[2] Some authors refer to these equations as the GLAP equations or the Altarelli-Parisi equations.

quantities, like in renormalization techniques, leads to scaling violation. The development we will show can be applied to any order of perturbation theory and is mainly the **factorisation theorem** [22]. In the second step, we will calculate the correction due to one gluon emission (OGE) and obtain the DGLAP evolution equations at leading order.

Once the evolution equations are established, we will study their analytical properties. This suggests to go to Mellin space where the structure of the equation become much simpler. Studying the solutions in Mellin space, we will link our results with analytical S-matrix theory. Precisely, we will show that at small x_B (large s), the DGLAP analytical solutions behave like an essential singularity, appearing at the factorisation scale, in Regge theory [23, 24]. We will cite some development, in QCD theory and phenomenological models, trying to solve that problem. We will conclude the chapter by a comment on eigenfunctions of the evolution kernel.

Since no analytical solution of the DGLAP equations exists in x-space for an arbitrary given initial condition , we must solve them numerically. The main task of this work consist in writing an algorithm to solve the DGLAP equations, from a given x behaviour at an initial scale, up to a final scale. Our algorithm works directly in x-space and we will explain the main tools, such as x-space discretization which we used. We show, by comparison with the GRV [28, 29] parton densities, that our algorithm works with a good precision.

Such an evolution algorithm can find a lot of application in systematic studies of the DGLAP equations. We chose to present one of them in this report. Since we have quite a lot of experimental measurements of the F_2 structure function, we can compare them with the DGLAP predictions. This task consists in starting with a parametric initial condition, evolving it with our evolution algorithm, compare prediction with the data, and find the best values for the parameters. We will describe how this has been implemented. Finally, we will give an example of fit to F_2 structure function, obtained with our application, using the MRST parametrization [21] for the initial condition.

Chapter 2

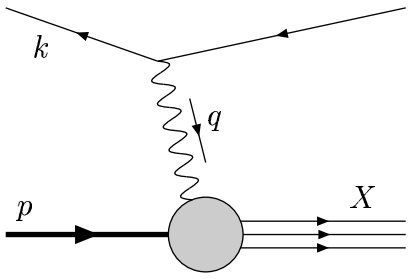
The DGLAP Equation

The DGLAP evolution equation and its implications on scaling violation are considered as one of the most famous success of QCD and, more precisely, perturbative QCD (pQCD). Its agreement with experimental measurements of structure functions has already been tested successfully at leading order (LO), next-to-leading order (NLO) and next-to-next-leading order (NNLO) in perturbation theory.

In this chapter, I will start with a short derivation of the DGLAP equation. Since this equation is based on the factorisation theorem, which is by itself a powerful result, I will firstly explain how to obtain that theorem. I will then extract the LO splitting by calculating the first order QCD corrections.

2.1 Framework

We will place ourselves in the framework of Deep Inelastic Scattering (DIS). We thus consider scattering of a positron on a proton by exchange of a virtual photon, as shown on the figure below.



$$\begin{aligned} q^2 &= -Q^2 < 0 \\ s &= (p+k)^2 \\ W^2 &= (p+q)^2 \\ \nu &= p.q \\ x &= \frac{Q^2}{2\nu} \\ y &= \frac{p.q}{p.k} \end{aligned}$$

The differential-cross section can be written

$$\frac{d^2\sigma}{dxdQ^2} = \frac{2\pi\alpha^2}{xQ^4}\left\{\left[1+(1-y)^2\right]F_2(x,Q^2) - y^2 F_L(x,Q^2)\right\} \tag{2.1}$$

At lowest order, the photon directly interacts with a quark from the proton and

$$F_2(x,Q^2) = x\sum_{q,\bar{q}} e_q^2\left[q(x)+\bar{q}(x)\right]. \tag{2.2}$$

F_2 is therefore scale-independent. This is known as Bjorken scaling [19].

However, when we try to compute QCD corrections, we face collinear divergences of the form $\alpha_s^n \ln^n(Q^2/\kappa^2)$, where κ^2 is an infra-red cut-off. This implies that we need to resum the perturbation theory expansion and $q(x)$ in (2.2) will also depend on Q^2. We will do this in the next pages, considering the regime where $Q^2 \to \infty$.

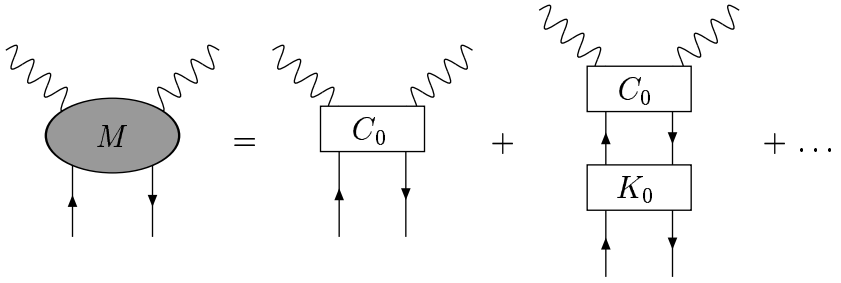
2.2 Factorisation theorem

2.2.1 Factorising the $\gamma^* q$ amplitude

We will only consider the non-singlet case [22]. Since the detailed proof of the factorisation theorem is quite tedious, we will only outline the salient points without entering much into the calculational details.

Our task is to extract the collinear singularities out of the $\gamma^* p$ interaction. These correspond to logarithmic divergences of the form $\ln(Q^2/p^2)$. Thus, the large Q^2 limit corresponds to the small p^2 one, and divergences are also called mass singularities.

First of all, let us consider the squared matrix element for $\gamma^* q$ interaction, denoted by M. We want to extract mass singularities of M in the axial gauge[1]. M can be expanded in terms of the 2-particle-irreducible (2PI) kernels K_0 and C_0 as follow



That ladder expansion gives

$$M = C_0 \otimes (\mathbb{1} + K_0 + K_0 \otimes K_0 + \ldots) = C_0 \otimes \frac{1}{\mathbb{1} - K_0} = C_0 \otimes \Gamma_0, \tag{2.3}$$

where convolution contains spin sums and phase space integrals.

Factorisation of M is then performed by introducing a projector $\mathbb{P} = \mathbb{P}_\varepsilon \times \mathbb{P}_n$ where

1. $\mathbb{P}_n$ decouples C_0 and Γ_0 in spinor indices

[1] Using the axial gauge $n.A = 0$ allows to keep the probabilistic interpretation when going from leading order to next-to-leading order, to simplify callculations and to compare results with the Operator Product Expansion techniques.

2. $\mathbb{P}_\varepsilon$ extract the collinear singularities out of the phase space integrations. These singularities behave like $1/\varepsilon$ and arise from the dk^2/k^2 integration in $4+\varepsilon$ dimensions.

We must use this projector to carry mass singularities out of the ladder expansion. By an easy recurrence, we find

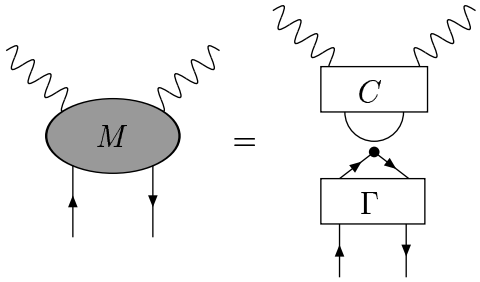
$$M = \left(C_0 \frac{1}{\mathbb{I} - (\mathbb{I} - \mathbb{P})K_0}\right)\left(\frac{1}{\mathbb{I} - \mathbb{P}K}\right) \tag{2.4}$$

where

$$K = K_0\left(\frac{1}{\mathbb{I} - (\mathbb{I} - \mathbb{P})K_0}\right) \tag{2.5}$$

$$\frac{1}{\mathbb{I} - \mathbb{P}K} = \mathbb{I} + \mathbb{P}K + (\mathbb{P}K)(\mathbb{P}K) + \ldots. \tag{2.6}$$

We thus have rewritten M



where $C = C_0 \frac{1}{1-(1-\mathbb{P})K_0}$ is free of mass singularity and $\Gamma = \frac{1}{\mathbb{I}-\mathbb{P}K}$ also includes Z_F, the residue of the pole of the full quark propagator.

Physically, $\Gamma(Q^2/\mu^2, x, \alpha, 1/\varepsilon)$ can be seen as a parton density in the parent quark, while $C(Q^2/\mu^2, x, \alpha)$ appears as a short-distance cross-section.

Before going into the consequences of this result, it is interesting to note that, in Mellin space[2], factorisation is written as a simple product instead of the convolution in momentum space

$$M(Q^2, j, \alpha, 1/\varepsilon) = C(Q^2/\mu^2, j, \alpha)\Gamma(Q^2, j, \alpha, 1/\varepsilon). \tag{2.7}$$

2.2.2 Renormalization group techniques

In another way, eq. (2.4), can be written $M(\mathbb{I} - \mathbb{P}K) = C$, which looks like a mass renormalization $M_{\text{bare}}Z = M_r$. The main difference in our case is that Γ, and thus $\mathbb{I}-\mathbb{P}K$,

[2]Mellin space corresponds to x moments.

$$f(j) = \int_0^1 dx\, x^{j-1} f(x), \qquad \text{and} \quad f(x) = \int_{c-i\infty}^{c+i\infty} dj\, x^{-j} f(j),$$

where the integration contour stands at the right of all f singularities.

seems to depend on Q^2/μ^2. One of the important result of the factorisation theorem is that Γ is actually independent of Q^2/μ^2. To see this, start from the fact that one can show that the 2PI kernel is finite. Since the projector $\mathbb{P}$ extract the $1/\varepsilon$ divergences, K_0 and $K_0\left[(\mathbb{1}-\mathbb{P})K_0\right]^n$ are finite when $\varepsilon \to 0$ before the last dk^2/k^2 integration. Thus

$$\Gamma(Q^2/\mu^2, x, 1/\varepsilon) = \delta(1-x) + \sum_i \frac{1}{\varepsilon^i}\Gamma_i(Q^2/\mu^2, x) = \delta(1-x) + \int_{-Q^2}^{0} \frac{dk^2}{k^2}\Phi(k^2/\mu^2, x, \varepsilon), \quad (2.8)$$

where Φ is finite when $\varepsilon \to 0$. Differentiating both sides with respect to Q^2 and comparing coefficients gives $\partial_{Q^2}\Gamma_i = 0$ for all i, and thus

$$\partial_{Q^2}\Gamma = 0. \quad (2.9)$$

as stated.

This proves that we can apply the well-known renormalization group techniques to our problem. Let $\mathcal{D}$ denote the total derivative with respect to μ^2

$$\mathcal{D} = \mu\partial_\mu + \left[\beta(g) + \frac{1}{2}g\varepsilon\right]\partial_g. \quad (2.10)$$

Being a physical quantity, M must be μ-independent. So, working in Mellin space, we have from (2.7)

$$\mathcal{D}\ln(M) = \mathcal{D}\ln(C) + \mathcal{D}\ln(\Gamma) = 0. \quad (2.11)$$

If we introduce

$$\gamma(j, \alpha, \varepsilon) = -\mathcal{D}\ln(\Gamma) = -\left[\beta(g) + \frac{1}{2}g\varepsilon\right]\partial_g \ln\left(\Gamma(j, \alpha, 1/\varepsilon)\right), \quad (2.12)$$

we have the following "renormalization group equation"

$$\left[\mathcal{D} - \gamma(j, \alpha)\right] C(Q^2/\mu^2, j, \alpha, \varepsilon) = 0. \quad (2.13)$$

The key point of this expression is the fact that, since both $\mathcal{D}$ and C don't contain poles in ε, γ is itself free of poles in ε. Expanding $\Gamma(j) = 1 + \sum_i \Gamma_k \varepsilon^{-k}$, we get $\gamma = -\alpha\partial_\alpha\Gamma_1$ which is useful to calculate γ.

Finally, if we integrate (2.12), we have

$$\Gamma(j, \alpha, 1/\varepsilon) = \exp\left[-\int_0^\alpha \frac{d\lambda}{\lambda}\frac{\gamma(j, \lambda)}{2\bar{\beta}(\lambda) + \varepsilon}\right], \quad (2.14)$$

with $\bar{\beta}(\alpha) = \frac{1}{g}\beta(g)$.

Using the μ-independence of M, we can write

$$M(Q^2, j, \alpha, 1/\varepsilon) = C\left(1, j, \alpha(Q^2)\right) \exp\left[-\int_0^{\alpha(Q^2)} \frac{d\lambda}{\lambda}\frac{\gamma(j, \lambda)}{2\bar{\beta}(\lambda) + \varepsilon}\right]. \quad (2.15)$$

2.2.3 The full $\gamma^* p$ case : parton densities

The last step in order to recover the full DIS properties is to consider an incoming proton instead of a single quark. The total $\gamma^* p$ square amplitude, will be the $\gamma^* q$ square amplitude we just calculated, convoluted with the quark *bare* density inside the proton.

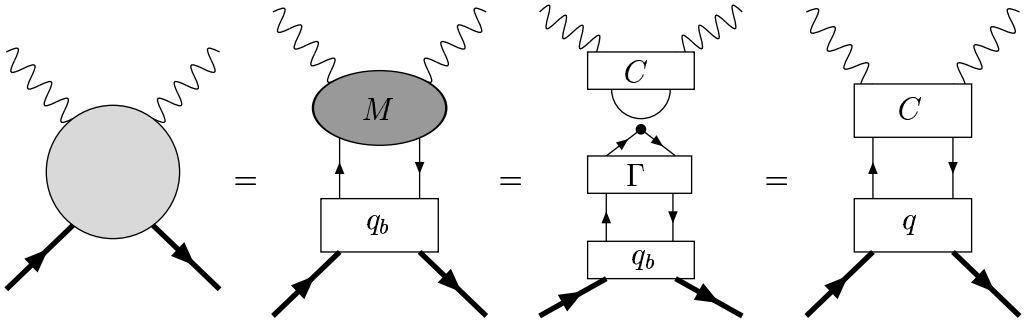
Let us call H the 2PI $p \to q$ kernel. If we assume that H is tempered i.e. $H(p^2, x) \leq C\,|p^2|^\eta$ when $\mid p^2 \mid \to \infty$ for some C and $\eta > 0$, the bare parton density becomes[3]

$$q_{\text{bare}}(x, \alpha, 1/\varepsilon) = \int_{-Q^2}^{0} \frac{dp^2}{p^2} H(p^2, x) = \int^{-\infty} \frac{dp^2}{p^2} H(p^2, x) + \mathcal{O}\left[\left(\frac{\mu^2}{Q^2}\right)^\eta\right]. \tag{2.16}$$

The lower limit in this integral will generate the mass divergences which, due to the KLN theorem [25], must cancel the ones from the $\gamma^* q$ interaction. This allows us to define the *dressed* density through

$$q(j, Q^2) = \exp\left[-\int_0^{\alpha(Q^2)} \frac{d\lambda}{\lambda} \frac{\gamma(j, \lambda)}{2\bar{\beta}(\lambda) + \varepsilon}\right] q_{\text{bare}}(j, Q^2). \tag{2.17}$$

Diagramatically, the process of factorisation for $\gamma^* p$ interaction can be summarise like this



2.2.4 Parton density evolution

With all the calculations performed since the beginning of this section, the DGLAP evolution equation is directly obtained by differentiating eq (2.17) with respect to Q^2. This gives[4]

$$\boxed{Q^2 \partial_{Q^2} q(j, Q^2) = -\frac{1}{2}\gamma\left(j, \alpha(Q^2)\right) q(j, Q^2)}, \tag{2.18}$$

with the solution

$$q(j, Q^2) = q(j, Q_0^2) \exp\left[-\int_{\alpha(Q_0^2)}^{\alpha(Q^2)} \frac{d\lambda}{\lambda} \frac{\gamma(j, \lambda)}{2\bar{\beta}(\lambda)}\right]. \tag{2.19}$$

[3] Contributions from the last term, going to 0 like a power of Q^2, are called a **higher twists** and are neglected here.

[4] The $-\frac{1}{2}$ factor in the RHS is often absorbed into γ.

2.3 First order corrections and LO DGLAP

If we go back to momentum space, the evolution equation (2.18) becomes

$$Q^2 \partial_{Q^2} q(x, Q^2) = \int_x^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}, \alpha_s(Q^2)\right) q(\xi, Q^2). \tag{2.20}$$

$P(x)$ is called the *splitting function*, and its Mellin transform

$$\gamma(j, \alpha_s) = \int_0^1 dx\, x^{j-1} P(x, \alpha_s), \tag{2.21}$$

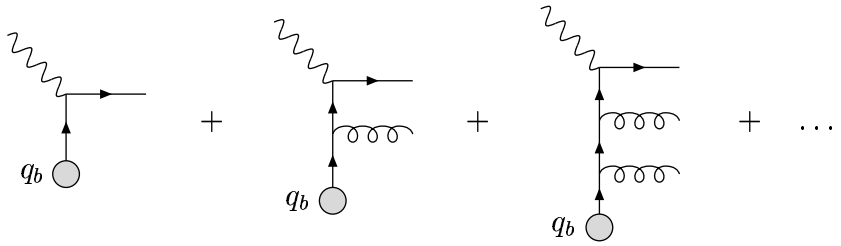
is known as the *anomalous dimension*.

From the point of view of perturbative QCD, we can expand $P(x, \alpha_s)$ as a series of the strong coupling constant α_s

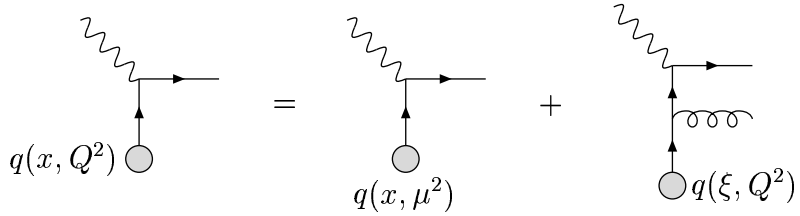
$$P(x, \alpha_s) = \frac{\alpha_s}{2\pi} P^{(0)}(x) + \left(\frac{\alpha_s}{2\pi}\right)^2 P^{(1)}(x) + \dots \tag{2.22}$$

In this section we will show how to obtain the non-singlet splitting function, at leading order, from QCD calculations.

The collinear divergences arise from gluons emissions and in the large Q^2 limit, we will have to resum contributions proportional to $\alpha_s^n \ln^n(Q^2/\mu^2)$. Starting from the bare density, $q_b(x)$, we want to calculate



If we use the dressed parton densities, the Q^2 corrections appear clearer



This means that we only need to calculate the one gluon emission (OGE) diagram, which is of order α_s. The diagrams corresponding to real gluon emission are shown in figure 2.1

Since we are only interested by the large Q^2 corrections, we can consider that the incoming quark is on its mass-shell ($p^2 = 0$) and neglect all the quark masses. Those will actually lead to subleading corrections or higher twists.

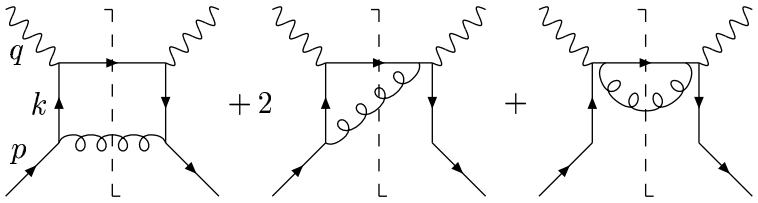


Figure 2.1: Diagrams for real gluon emission at order α_s.

To perform calculations, we will place ourselves in the light-cone frame. This is done by introducing a second light-like 4-vector n satisfying

$$n^2 = 0 \qquad \text{and} \qquad n.p = 1. \tag{2.23}$$

Using this new vector, the q and k impulsions can be written

$$q^\mu = \nu n^\mu + q_\perp^\mu, \tag{2.24}$$

$$k^\mu = \xi p^\mu + \frac{k_\perp^2 - |k^2|}{2\xi} n^\mu + k_\perp^\mu. \tag{2.25}$$

This parametrization for q satisfies $q^2 = -q_\perp^2 = -Q^2$ and $p.q = \nu$. One can easily check that the jacobian of such a parametrization gives

$$d^4k = \frac{d\xi}{4\xi} d\left|k^2\right| dk_\perp^2 d\theta. \tag{2.26}$$

Summing over polarisation states, the amplitude of the first Feynman graph is

$$|\mathcal{M}|^{\mu\nu} = \frac{1}{2} e_q^2 g^2 C_F \frac{1}{k^4} \sum_\lambda \mathrm{tr}\left(\not{p}\not{\varepsilon}_\lambda \not{k}\gamma^\nu (\not{k} + \not{q})\gamma^\mu \not{k}\not{\varepsilon}_\lambda^*\right) \tag{2.27}$$

We use the light-cone gauge $n.A = 0$. In this gauge, if we have a gluon of impulsion l, the sum over its polarisation states gives

$$\sum_\lambda \varepsilon_\lambda^\mu(l)\varepsilon_\lambda^{*\nu}(l) = -g^{\mu\nu} + \frac{n^\mu l^\nu + l^\mu n^\nu}{l.n}. \tag{2.28}$$

To fully calculate the contribution to F_2 from the OGE diagram we also need to integrate over phase space and project the result on the F_2 part of the cross-section. In our particular case, the phase space can be written

$$\begin{aligned} \phi &= \frac{d^4k}{(2\pi)^4}(2\pi)\delta^+\left((p-k)^2\right)(2\pi)\delta^+\left((k+q)^2\right) \\ &= \frac{1}{16\pi^2\nu} d\xi dk^2 dk_\perp^2 d\theta \delta\left(k_\perp^2 - (1-\xi)\left|k^2\right|\right)\delta\left(\xi - x - \frac{|k^2| + 2\vec{q}_\perp.\vec{k}_\perp}{2\nu}\right), \end{aligned} \tag{2.29}$$

while to project out the F_2 contribution, we have to multiply the result by an overall factor

$$\frac{1}{4\pi} n^\mu n^\nu \tag{2.30}$$

Inserting the relation (2.28) in the amplitude expression and using our parametrization for k, we can compute the projected amplitude. After a little bit of algebra, we find

$$\frac{n^\mu n^\nu}{4\pi} |\mathcal{M}|^2_{\mu\nu} = 8 e_q^2 \frac{\alpha_s}{|k^2|} \xi P(\xi), \tag{2.31}$$

with

$$P(x) = C_F \frac{1+x^2}{1-x}. \tag{2.32}$$

We are left with the phase space integration. The first step is to remove the $k_\perp^2$ and θ integrations using the on-shell conditions[5]. This leads to

$$F_2 = \frac{e_q^2 \alpha_s}{2\pi^2} \int_0^{2\nu} \frac{d|k^2|}{|k^2|} \int_{\xi_-}^{\xi_+} d\xi \frac{\xi P(\xi)}{\sqrt{(\xi_+ - \xi)(\xi - \xi_-)}} \tag{2.33}$$

where the limits on $|k^2|$ come from energy positivity and the limits on ξ, which are ($z = |k^2|/(2\nu)$)

$$\xi_\pm = x + z - 2xz \pm \sqrt{4x(1-x)z(1-z)}, \tag{2.34}$$

are obtained by imposing $-1 \leq \cos(\theta) \leq 1$.

Expanding the ξ integral in series of z, we find

$$\begin{aligned} \int_{\xi_-}^{\xi^+} d\xi \frac{\xi P(\xi)}{\sqrt{(\xi_+ - \xi)(\xi - \xi_-)}} &= \int_{\xi_-}^{\xi^+} d\xi \frac{x P(x)}{\sqrt{(\xi_+ - \xi)(\xi - \xi_-)}} + \mathcal{O}(z) \\ &= \pi x P(x) + \mathcal{O}\left(\frac{|k^2|}{2\nu}\right). \end{aligned} \tag{2.35}$$

We can keep only the first term since we are interested in the leading Q^2 contribution. The $|k^2|$ integral is divergent[6]. We will thus introduce a small infra-red cut-off κ^2 in order to make it finite. The contribution of the first diagram is then

$$F_2 = e_q^2 \frac{\alpha_s}{2\pi} x P(x) \ln\left(\frac{2\nu}{\kappa^2}\right). \tag{2.36}$$

The second diagram, contains a $1/(p+q)^2$ from the quark propagator instead of $1/k^2$, thus its contribution is proportional to 1 compared with $\ln(Q^2/\kappa^2)$. The third one leads to the same kind of subleading corrections. Note that, in the same order of idea, we can replace $\ln(2\nu/\kappa^2)$ by $\ln(Q^2/\kappa^2)$ and neglect the $\ln(1/x)$ contribution which is subleading too.

[5] $k_\perp^2$ is extracted out of the first δ condition and $\cos(\theta)$ out of the second one.

[6] This corresponds to the collinear divergences we met in the factorisation theorem.

We still need to compute the virtual diagrams. These are vertex and propagator corrections. However, we can do this without entering the details if we simply note that virtual corrections have the same behaviour in x than the 0th order contributions and are thus proportional to $\delta(1-x)$. Therefore, their effect is to replace $P(x)$ by $P(x)+K\delta(1-x)$. The constant K is fixed by asking that $P(x)$ integrate to 0 which is a consequence of quark number conservation. If we introduce the *plus* distribution defined through[7]

$$\frac{1}{(1-x)_+} = \frac{1}{1-x} \qquad \text{for} \quad 0 \leq x < 1, \tag{2.37}$$

$$\int_0^1 dx \, \frac{f(x)}{(1-x)_+} = \int_0^1 dx \, \frac{f(x)-f(1)}{1-x}, \tag{2.38}$$

the final splitting is

$$C_F \left[\frac{1+x^2}{(1-x)_+} + \frac{3}{2}\delta(1-x) \right]. \tag{2.39}$$

Taking into account the virtual corrections, we must replace the unrenormalized α_s by the running coupling[8].

Our last task in this section is to remove the infra-red cut-off. This is done by going from the bare to the dressed parton density. Actually, the computed correction has to be convoluted by the bare quark density. The contribution up to first order in α_s is

$$F_2(x,Q^2) = e_q^2 x \int_x^1 \frac{d\xi}{\xi} \left[\delta\left(1-\frac{x}{\xi}\right) + \frac{\alpha_s}{2\pi} \ln\left(\frac{Q^2}{\kappa^2}\right) P\left(\frac{x}{\xi}\right) \right] q_b(\xi). \tag{2.40}$$

We can introduce a *factorisation scale* μ and define a dressed density at that point

$$q(x,\mu^2) = q_b(x) + \frac{\alpha_s}{2\pi} \ln\left(\frac{\mu^2}{\kappa^2}\right) \int_x^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}\right) q_b(\xi). \tag{2.41}$$

If we write $F_2(x,Q^2) = e_q^2 x q(x,Q^2)$, we can remove the bare density dependence

$$\begin{aligned} q(x,Q^2) &= q_b(x) + \frac{\alpha_s}{2\pi} \ln\left(\frac{Q^2}{\kappa^2}\right) \int_x^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}\right) q_b(\xi) \\ &= q(x,\mu^2) + \frac{\alpha_s}{2\pi} \ln\left(\frac{Q^2}{\mu^2}\right) \int_x^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}\right) q(\xi,\mu^2) + \mathcal{O}(\alpha_s^2). \end{aligned} \tag{2.42}$$

The factorisation allows us to replace the divergent bare density by the finite dressed one, introducing a scale dependence.

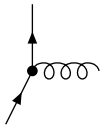
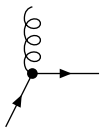
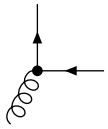
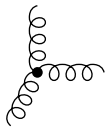
Since $q(x,Q^2)$ is a physical quantity, it may not depend on the factorisation scale μ^2. The derivative of equation (2.42) with respect to μ^2 has thus to vanish. This gives the well-known DGLAP evolution equation

[7] We will discuss briefly in the next section how it appears from regularization of the divergences

[8] Since we consider first order corrections, we must use the first order running coupling.

$$\boxed{Q^2\partial_{Q^2}q(x,Q^2) = \frac{\alpha_s(Q^2)}{2\pi}\int_x^1 \frac{d\xi}{\xi}P\left(\frac{x}{\xi}\right)q(\xi,Q^2)}. \tag{2.43}$$

In all these calculations, we have limited ourselves to the case of gluon emission by a quark coming out the proton. In that case, $P(x) \equiv P_{qq}(x)$ describe the probability density to find a quark of impulsion xp into a quark of impulsion p, emitting a gluon of impulsion $(1-x)p$. This can be generalised to the probability density to find a parton a (quark or gluon) into a parton b by emission of a third parton. The four cases are summarised in the table below [26]

Diagram	Splitting
	$P_{qq} = C_F\left[\frac{1+x^2}{(1-x)_+} + \frac{3}{2}\delta(1-x)\right]$
	$P_{gq} = C_F\left[\frac{1+(1-x)^2}{x}\right]$
	$P_{qg} = T_R\left[x^2+(1-x)^2\right]$
	$P_{gg} = 2C_A\left[\frac{x}{(1-x)_+} + (1-x)\left(x+\frac{1}{x}\right)\right] + \frac{11C_A-4n_fT_R}{6}\delta(1-x)$

$$P_{q_iq_j} = P_{\bar{q}_i\bar{q}_j} = P_{qq}\delta_{ij}. \tag{2.44}$$
$$P_{q\bar{q}} = 0, \tag{2.45}$$
$$P_{\bar{q}g} = P_{qg}, \tag{2.46}$$
$$P_{g\bar{q}} = P_{gq}, \tag{2.47}$$

The quark, antiquark and gluon densities are then coupled and the full set of DGLAP equations is

$$Q^2\partial_{Q^2}\begin{pmatrix} q_i(x,Q^2) \\ \bar{q}_i(x,Q^2) \\ g(x,Q^2)\end{pmatrix} = \frac{\alpha_s(Q^2)}{2\pi}\int_x^1\frac{d\xi}{\xi}\begin{pmatrix} P_{q_iq_j}\left(\frac{x}{\xi}\right) & 0 & P_{q_ig}\left(\frac{x}{\xi}\right) \\ 0 & P_{q_iq_j}\left(\frac{x}{\xi}\right) & P_{q_ig}\left(\frac{x}{\xi}\right) \\ P_{gq}\left(\frac{x}{\xi}\right) & P_{gq}\left(\frac{x}{\xi}\right) & P_{gg}\left(\frac{x}{\xi}\right)\end{pmatrix}\begin{pmatrix} q_j(x,Q^2) \\ \bar{q}_j(x,Q^2) \\ g(x,Q^2)\end{pmatrix}. \tag{2.48}$$

Beyond leading order, due to effects like

the splitting $P_{q\bar{q}}$ does not vanish anymore. However, if we introduce

$$q^{\pm}(x,Q^2) = q(x,Q^2) \pm \bar{q}(x,Q^2), \tag{2.49}$$

using flavour symmetry, we can construct 11 non-singlet densities : $V^i \equiv q_i^-$ and[9]

$$\begin{aligned} T_3 &= u^+ - d^+, \\ T_8 &= u^+ + d^+ - 2s^+, \\ T_{15} &= u^+ + d^+ + s^+ - 3c^+, \\ T_{24} &= u^+ + d^+ + s^+ + c^+ - 4b^+, \\ T_{35} &= u^+ + d^+ + s^+ + c^+ + b^+ - 5t^+. \end{aligned} \tag{2.50}$$

All these quantities evolve independently from one another.

The gluon distribution is part of the singlet densities and is coupled with the *flavor singlet quark density*

$$\Sigma(x,Q^2) = \sum_q \left[q(x,Q^2) + \bar{q}(x,Q^2) \right]. \tag{2.51}$$

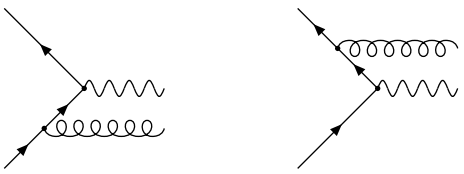
Besides the 11 uncoupled equations, we have the evolution equation for the flavor singlet

$$Q^2 \partial_{Q^2} \begin{pmatrix} \Sigma(x,Q^2) \\ g(x,Q^2) \end{pmatrix} = \frac{\alpha_s}{2\pi} \int_x^1 \frac{d\xi}{\xi} \begin{pmatrix} P_{qq}\left(\frac{x}{\xi}\right) & 2n_f P_{qg}\left(\frac{x}{\xi}\right) \\ P_{gq}\left(\frac{x}{\xi}\right) & P_{gg}\left(\frac{x}{\xi}\right) \end{pmatrix} \begin{pmatrix} \Sigma(\xi,Q^2) \\ g(x,Q^2) \end{pmatrix}. \tag{2.52}$$

2.4 Divergences and the plus distribution

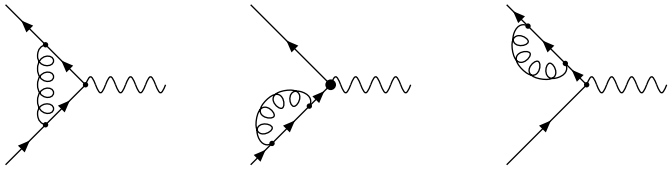
To conclude this chapter, we will shortly explain where the divergences comes from in virtual and emission diagrams, and how the plus distribution arises from the calculations. Since our aim is simply to tell what kind of divergences appear in the diagrams we met and to underline, how the plus distribution comes out of the divergences cancellation, we won't enter into the details of calculations in dimensional regularization.

Let us take, for example, the case of $q\bar{q} \to W^+$. If we want to compute the first order QCD corrections, we have two diagrams for real gluon emission



[9] indices are the order of the representation of the flavour symmetry group.

and three for virtual corrections



One can easily check, by power counting for example, that these diagrams are divergent. The real emissions contains infra-red and collinear divergences, corresponding to soft gluon emission and to emission parallel to the initial quark; while virtual correction are both infra-red and ultra-violet (hard gluon) divergent. If we use dimensional regularization to deal with them, we first look at the UV divergences in the virtual diagrams. Working with $d = 4 - 2\varepsilon_{UV}$, one can show that divergences cancel between propagator and vertex corrections.

One can then go to dimension $4 + 2\varepsilon_{IR}$ and deal with the soft divergences. The KLN (Kinoshita-Lee-Nauenberg) theorem [25] tells us that these cancel when we add virtual corrections to real emission diagrams.

We are then left with the collinear divergences in the real gluon emissions. If p denotes the initial quark impulsion and xp its impulsion after emission, the gluon emission amplitude contains terms proportional to $(1-x)^{1-2\varepsilon}$, which leads to both collinear and soft divergences. To isolate the pole in this expression, one can write

$$\begin{aligned}
\frac{1}{(1-x)^{1+2\varepsilon}} &= \frac{1}{(1-x)^{1+2\varepsilon}} \\
&\quad - \left[\delta(1-x)\int_0^1 \frac{dz}{(1-z)^{1+2\varepsilon}} + \delta(1-x)\frac{1}{2\varepsilon}\right] \\
&= \left[\frac{1}{(1-x)^{1+2\varepsilon}}\right]_+ - \frac{1}{2\varepsilon}\delta(1-x) \\
&= \left(\frac{1}{1-x}\right)_+ - 2\varepsilon\left[\frac{\ln(1-x)}{1-x}\right]_+ - \frac{1}{2\varepsilon}\delta(1-x).
\end{aligned}$$

The last divergent term corresponds to IR divergences and is cancelled by virtual corrections. We are thus left with collinear divergences

$$\sigma^{(1)} \sim \frac{1}{\varepsilon}\left(\frac{1+x^2}{1-x}\right)_+ + \text{finite.} \tag{2.53}$$

This clearly shown how the plus distribution allows us to isolate the collinear divergences in gluon emission diagram from the soft singularities, cancelled by virtual diagram.

Chapter 3

Analytical Properties

This chapter will be devoted to the study of the main analytical properties of the DGLAP evolution equation. Basically, we will look at the analytical behavior of the solutions of the equation, particularly at small x.

3.1 Solution in Mellin space

When dealing with the DGLAP equation, it is often useful to work in Mellin space. In that case, the x-convolution in the RHS of the equation simply become an algebraic product

$$Q^2 \partial_{Q^2} q(j, Q^2) = \frac{\alpha_s(Q^2)}{2\pi} \gamma(j) q(j, Q^2). \tag{3.1}$$

The Mellin transforms of the splitting functions, called *anomalous dimensions*, are

$$\gamma_{qq} = C_F \left[\frac{3}{2} - I(j-1) - I(j+1) \right] \tag{3.2}$$

$$\gamma_{qg} = T_R \left[\frac{2 + j + j^2}{j(j+1)(j+2)} \right] \tag{3.3}$$

$$\gamma_{gq} = C_F \left[\frac{2 + j + j^2}{j(j-1)(j+1)} \right] \tag{3.4}$$

$$\gamma_{gg} = \frac{11 C_A - 4 n_f T_R}{6} + 2 C_A \left[\frac{1}{j-1} - \frac{1}{j} + \frac{1}{j+1} - \frac{1}{j+2} - I(j) \right] \tag{3.5}$$

where[1]

$$I(\lambda) = \int dx \frac{1 - x^\lambda}{1 - x} = \gamma_E + \psi(\lambda + 1) \approx \sum_{j=1}^{\lambda} \frac{1}{j}. \tag{3.6}$$

They are shown in fig. 3.1.

[1] γ_E is the Euler constant and $\psi(z)$ is the *poly-gamma* function, which is the logarithmic derivative of the Euler gamma function.

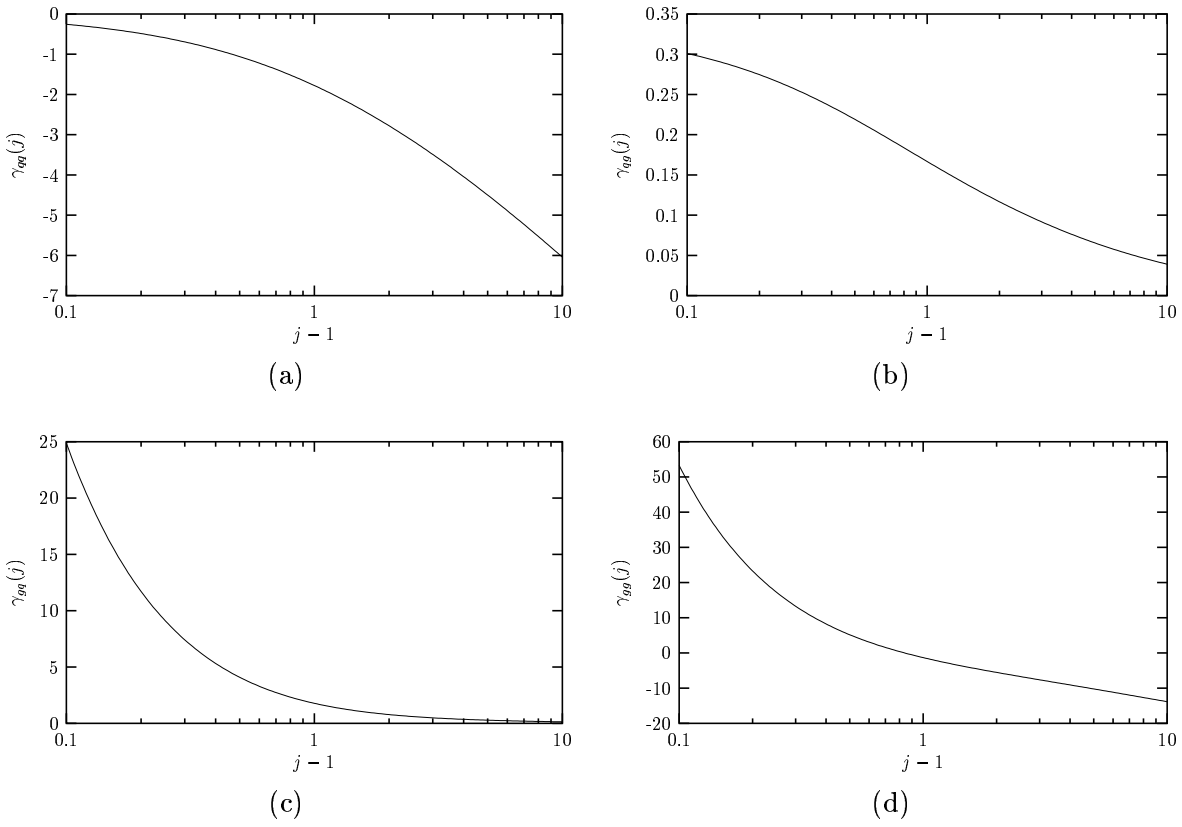


Figure 3.1: Anomalous dimensions for : (a) $q \to q$, (b) $g \to q$, (c) $q \to g$, (d) $g \to g$.

Using $\alpha_s(Q^2) = \frac{12\pi}{(33-2n_f)\ln(Q^2/\Lambda^2_{QCD})}$, we can easily check that the solution of (3.1) is

$$q(j,Q^2) = q(j,Q_0^2)\left[\frac{\alpha_s(Q_0^2)}{\alpha_s(Q^2)}\right]^{\frac{6}{33-2n_f}\gamma(j)} . \tag{3.7}$$

For the non-singlet case, the Mellin space expression still contains a matrix product. We first diagonalize the anomalous dimensions matrix and then solve it like in the singlet case.

3.2 Analytical behavior

It is of prime importance to study the analytical singularity structure of the DGLAP solutions, particularly for $x \ll 1$ where the Regge theory applies. At small x, the rightmost singularities in the Mellin plane dominates. So, as one can directly see from anomalous dimensions expressions, the dominant contribution at small x will come from the $\frac{1}{j-1}$ terms. Therefore

$$q(j,Q^2) \sim q(j,Q_0^2)\exp\left(\frac{C}{j-1}\right), \tag{3.8}$$

which suffers from two problems

1. it is an essential singularity, which is not the kind of singularity we expect from Regge theory,
2. it appears at the scale Q_0^2. A singularity appearing at the factorization scale, i.e. at an arbitrary and non-physical scale is far from being a good analytical behavior.

The first problem can be solved by resumming the perturbation theory in the large s limit. Although it suffers infra-red problems, this is done by the BFKL equation. That equation predicts that

$$F_2 \sim \frac{x^{-\omega}}{\sqrt{1/x)}} \qquad \text{with} \quad \omega = \frac{12\alpha_s \ln(2)}{\pi}. \tag{3.9}$$

Unfortunately, the BFKL prediction for structure function is unable to reproduce the data and its NLO corrections even may lead to negative cross-sections.

To avoid the second problem, one should try to modify the Q^2 dependence, which is only valid in the large Q^2 limit, in such a way that the singularity appears over the whole Q^2 range.

3.3 Eigenfunctions

We would like to find eigenfunctions of the DGLAP equation, i.e. particular solutions satisfying

$$\int_x^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}\right) f(\xi) = \lambda f(x). \tag{3.10}$$

We will give two different proofs that this equation has no non-trivial solution.

3.3.1 Volterra equations

Let $u = \ln(1/x)$, this equation can be rewritten

$$\int_0^u dv P(u-v) f(v) = \lambda f(u). \tag{3.11}$$

If we look in mathematical books reference, we find that equations of the form

$$\phi(x) - \lambda \int_0^x dy K(x,y)\phi(y) = f(x), \tag{3.12}$$

are *Volterra integral equations of the second kind.* If $f(x) = 0$, we call it *homogeneous.* For those integral equations, we have the general result [27]

Theorem 3.1. *The Volterra integral equation of the second kind*

$$\phi(x) - \lambda \int_0^x dy K(x,y)\phi(y) = f(x),$$

where the kernel $K(x,y)$ and the function $f(x)$ belong to the class L_2 has one and essentially one (up to a function vanishing almost everywhere) solution in the same class L_2.

This theorem admits the following corollary

Corrollary 3.2. *The homogeneous Volterra integral equation of the second kind*

$$\phi(x) - \lambda \int_0^x dy K(x,y)\phi(y) = 0,$$

where the Kernel $K(x,y)$ belong to the class L_2, admits only almost everywhere vanishing solutions.

Since the DGLAP equation is obviously a homogeneous Volterra integral equation of the second kind, we deduce from that corollary that it has no non-trivial eigenfunctions.

3.3.2 Mellin space

For those who are not satisfied with the requirement that $K(x,y)$ must belong to the L_2 class, we shall give another proof that (3.10) has no non-trivial solution. If we go to Mellin space, (3.10) becomes

$$\gamma(j)f(j) = \lambda f(j). \tag{3.13}$$

If we want a non-vanishing solution $f(x)$, we will also have a non-vanishing solution $f(j)$ and thus

$$\gamma(j) = \lambda \equiv \mathrm{c}^{\mathrm{st}}. \tag{3.14}$$

Converting this back into x-space leads to

$$P(x) = \lambda\delta(1-x). \tag{3.15}$$

This is clearly not the case for the physical splitting functions, hence the expected result.

To conclude this section, we must make the important remark that this result is not restricted to the LO DGLAP equation. It is valid with the full splittings. Actually, we can write an equation like (3.10) for each Q^2. The parton distributions have thus to vanish for each Q^2, whatever the Q^2 dependence of $P(x,Q^2)$ is.

Chapter 4

Numerical Resolution

An interesting thing to do with the DGLAP evolution equations, is to compare their predictions on the F_2 values with the experimental measurements. This is done by starting from a given distribution at an initial scale Q_0^2 (for every x) and evolving up to higher Q^2 using DGLAP. This task is however impossible analytically, and we need to process evolution numerically. To achieve this, two kinds of tools are needed:

1. a numerical resolver for the DGLAP equation,
2. a comparison of the evolution result with the experimental data.

This chapter will describe both these subjects, explaining the programs we wrote. Some important parts of the source code are joined in appendix and the full application is available on the Internet at the address

`http://virtual.theo.phys.ulg.ac.be/physics/dglap`

4.1 Numerical DGLAP resolution

4.1.1 Generalities

Defining

$$\begin{aligned} q^+(x,Q^2) &= q(x,Q^2)+\bar{q}(x,Q^2), \\ T(x,Q^2) &= x\left(u^+(x,Q^2)-d^+(x,Q^2)-s^+(x,Q^2)+c^+(x,Q^2)\right), \\ S(x,Q^2) &= x\left(u^+(x,Q^2)+d^+(x,Q^2)+s^+(x,Q^2)+c^+(x,Q^2)\right), \\ G(x,Q^2) &= xg(x,Q^2), \end{aligned} \tag{4.1}$$

the proton structure function becomes

$$F_2(x,Q^2)=\frac{3T+5S}{18}. \tag{4.2}$$

The DGLAP evolution equation for T and S can be written

$$\begin{aligned}\partial_t T(x,t) &= \frac{\alpha_s}{2\pi}\int_x^1 \frac{d\xi}{\xi}\Pi_{qq}\left(\frac{x}{\xi}\right)T(\xi,t)\\ \partial_t\begin{pmatrix}S(x,t)\\ G(x,t)\end{pmatrix} &= \frac{\alpha_s}{2\pi}\int_x^1 \frac{d\xi}{\xi}\begin{pmatrix}\Pi_{qq}\left(\frac{x}{\xi}\right) & \Pi_{qg}\left(\frac{x}{\xi}\right)\\ \Pi_{gq}\left(\frac{x}{\xi}\right) & \Pi_{gg}\left(\frac{x}{\xi}\right)\end{pmatrix}\begin{pmatrix}S(\xi,t)\\ G(\xi,t)\end{pmatrix}\end{aligned}$$

where $t = \ln(Q^2)$ and $\Pi_{ab}(x) = xP_{ab}(x)$.

We thus have implemented two DGLAP modules, for single DGLAP evolution and for two coupled DGLAP equations. We solved directly the DGLAP equation in x-space by estimating, as precisely as possible, the integral in the RHS and considering the whole equation as a set of coupled differential equations. The values of T, S and G are calculated for $x_{\min} \leq x \leq 1$ and ${Q^2}_{\min} \leq Q^2 \leq {Q^2}_{\max}$. More precisely, the points where evolution is calculated form a grid with `NB_X` values in x ans `NB_Q2` values in Q^2. The values of x and Q^2 are chosen equally distributed respectively in $\ln(x)$ and $\ln(Q^2)$.

4.1.2 Splittings

Since $T, G \sim xq$ and $G \sim xg$, the splitting used in the evolution equations are also multiplied by x, compared with the usual DGLAP splittings.

Practically, each splitting is given by P_R, P_+, P_δ and DP_1 defined through

$$P = P_R + \frac{P_+}{(1-x)_+} + P_\delta\delta(1-x), \tag{4.3}$$

and $DP_1 = \partial P_+|_1$.

4.1.3 Integration

Due to the fact that we work with a grid, we have to discretize the integral in the RHS of the DGLAP equations. In that integral, only the densities depend on Q^2; this allow us to cast it in the following discretized form

$$\int_{x_i}^1 \frac{d\xi}{\xi}P\left(\frac{x}{\xi}\right)q(\xi,Q^2) \approx \sum_{j=i}^{n_x}\mathcal{K}_{ij}q(x_j,Q^2). \tag{4.4}$$

The main advantage of that expression is that we can compute the integration kernel $\mathcal{K}$, which is Q^2-independent, once for all before starting the evolution itself. The estimation of the RHS term during the differential equation resolution will therefore be limited to a matricial product instead of the full integration calculation.

Evaluation of the integral in the evolution equations has been performed by the Bode algorithm [36] when it is allowed. When the number of points over which integration is performed is too small to apply the Bode method (i.e. at large x), we use Simpson or trapezoidal rules.

To enter a little bit more in details, we explain the main steps of discretization of the integral in the case of trapezoidal integration. Generalisation to more precise algorithms follows directly. Let

$$I(x,t) = \frac{\alpha_s(t)}{2\pi t}\int_x^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}\right) f(\xi,t). \tag{4.5}$$

We consecutively have

$$\begin{aligned} I_i &= \int_{x_i}^1 \frac{d\xi}{\xi} P\left(\frac{x}{\xi}\right) f(\xi,t). \\ &\approx \int_{x_i}^1 \frac{d\xi}{\xi} P_\delta \delta\left(1-\frac{x}{\xi}\right) f(\xi,t) + \sum_{j=i}^{N-2} \frac{x_{j+1}-x_j}{2}\left[\frac{P_R(x_i/x_j)}{x_j} f_j + \frac{P_R(x_i/x_{j+1})}{x_{j+1}} f_{j+1}\right] + I_i^+ \\ &\approx P_\delta f_i + \sum_{j=i}^{N-2} \frac{x_{j+1}-x_j}{2}\left[\frac{P_R(x_i/x_j)}{x_j} f_j + \frac{P_R(x_i/x_{j+1})}{x_{j+1}} f_{j+1}\right] + I_i^+. \end{aligned} \tag{4.6}$$

The remaining part I_i^+, constituting the plus prescription part, can be simplified in the following way [1]

$$\begin{aligned} \int_x^1 \frac{d\xi}{\xi}\frac{f(\xi)}{(1-\frac{x}{\xi})_+} &= \int_x^1 \frac{du}{u}\frac{f(\frac{x}{u})}{(1-u)_+} \\ &= \int_x^1 \frac{du}{u}\frac{f(\frac{x}{u})-uf(x)}{1-u} + f(x)\ln(1-x) \\ &= \int_x^1 \frac{d\xi}{\xi}\frac{f(\xi)-\frac{x}{\xi}f(x)}{1-\frac{x}{\xi}} + f(x)\ln(1-x) \\ &= \int_x^1 d\xi\frac{f(\xi)-\frac{x}{\xi}f(x)}{\xi-x} + f(x)\ln(1-x), \end{aligned}$$

which can be directly inserted into (4.6) except for $\xi = x$ where we have

$$\lim_{\xi\to x}\frac{f(\xi)-\frac{x}{\xi}f(x)}{\xi-x} = \frac{1}{x}\left.\frac{\partial \xi f(\xi)}{\partial \xi}\right|_{\xi=x}$$

In our case, $f(\xi) \equiv P(x/\xi)f(\xi,t)$ so

$$\lim_{\xi\to x}\frac{f(\xi)-\frac{x}{\xi}f(x)}{\xi-x} = \frac{P_+(1)f_i}{x_i} + P_+(1)\frac{f_{i+1}-f_i}{x_{i+1}-x_i} - \frac{1}{x_i}\left.\frac{\partial P_+}{\partial x}\right|_{x=1} f_i.$$

[1] We have

$$\begin{aligned} \int_x^1 dx\frac{f(x)}{(1-x)_+} &= \left(\int_0^1 - \int_0^x\right) dx\frac{f(x)}{(1-x)_+} = \int_0^1 dx\frac{f(x)-f(1)}{1-x} - \int_0^x dx\frac{f(x)}{1-x} \\ &= \int_x^1 dx\frac{f(x)-f(1)}{1-x} - \int_0^x dx\frac{f(1)}{1-x} = \int_x^1 dx\frac{f(x)-f(1)}{1-x} + f(1)\ln(1-x). \end{aligned}$$

Finally,

$$
\begin{aligned}
I_i^+ &= \sum_{j=i+1}^{N-2} \frac{x_{j+1}-x_j}{2}\left[\frac{P_+(x_i/x_j)f_j - \frac{x_i}{x_j}P_+(1)f_i}{x_j-x_i} + \frac{P_+(x_i/x_{j+1})f_{j+1} - \frac{x_i}{x_{j+1}}P_+(1)f_i}{x_{j+1}-x_i}\right] \\
&+ \frac{x_{i+1}-x_i}{2}\left[\frac{P_+(1)f_i}{x_i} + P_+(1)\frac{f_{i+1}-f_i}{x_{i+1}-x_i} - \frac{1}{x_i}DP_1f_i\right] \\
&+ \frac{P(x_i/x_{i+1})f_{i+1} - \frac{x_i}{x_{i+1}}P_+(1)f_i}{2} + \frac{1}{2}f_iP_+(1)\ln(1-x_i).
\end{aligned}
$$

4.1.4 Differential equation

A differential equation solver contains 3 parts :

- an algorithm which evolves from a given initial value up to a given upper value.
- a stepper, which estimates errors of the previous algorithm and calculates the steps to pass to it, in order to minimise errors.
- a general function, dealing with the stepper, which really performs the evolution at the values you want it to. This is called the driver.

For our purposes, we used a 4th order Runge-Kutta method as evolution algorithm, the Bulirsch-Stoer stepper and the ODEInt driver, all three are described in from Numerical Recipes [36].

4.2 Testing the evolution

In order to test the efficiency of our evolution algorithm, we compared its results with the GRV [28] leading order evolution. We started et $Q^2 = 20\,\mathrm{GeV}^2$ with the GRV parton densities[2] and let them evolve with our algorithm. We then compared our results with the GRV one at various Q^2 up to 10 000 GeV^2. The Q^2 range was chosen to have a constant number of active flavours. We used $n_f = 5$, and $\Lambda_{QCD} = 132$ MeV. The densities we obtained using our evolution algorithm are shown in figure 4.1. The relative error with the GRV results is presented in fugure 4.2.

[2]We used the GRV98 densities [29].

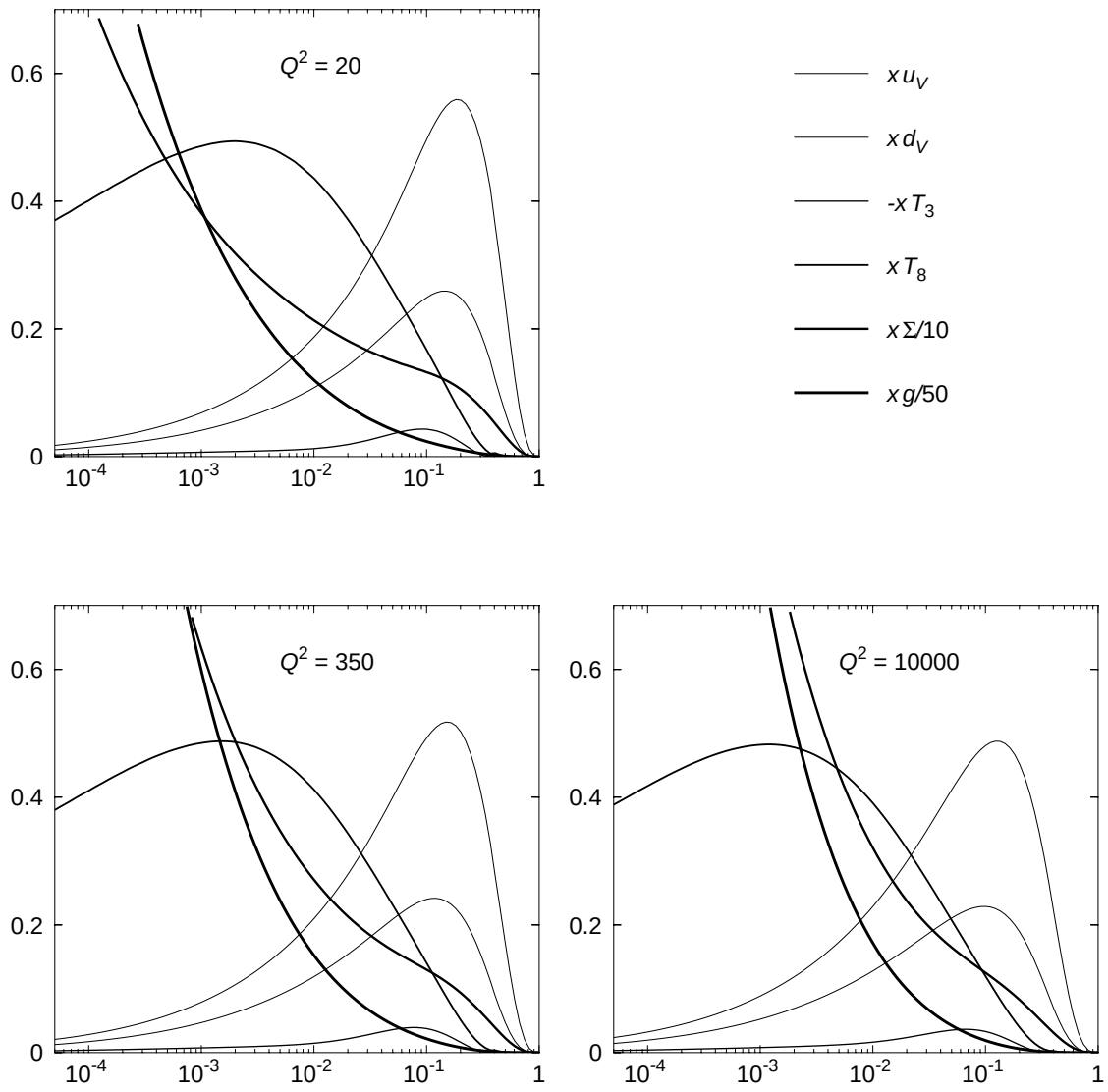


Figure 4.1: Parton densities obtained with our evolution algorithm starting at $Q^2 = 20\,\mathrm{Ge}V^2$ with the GRV densities.

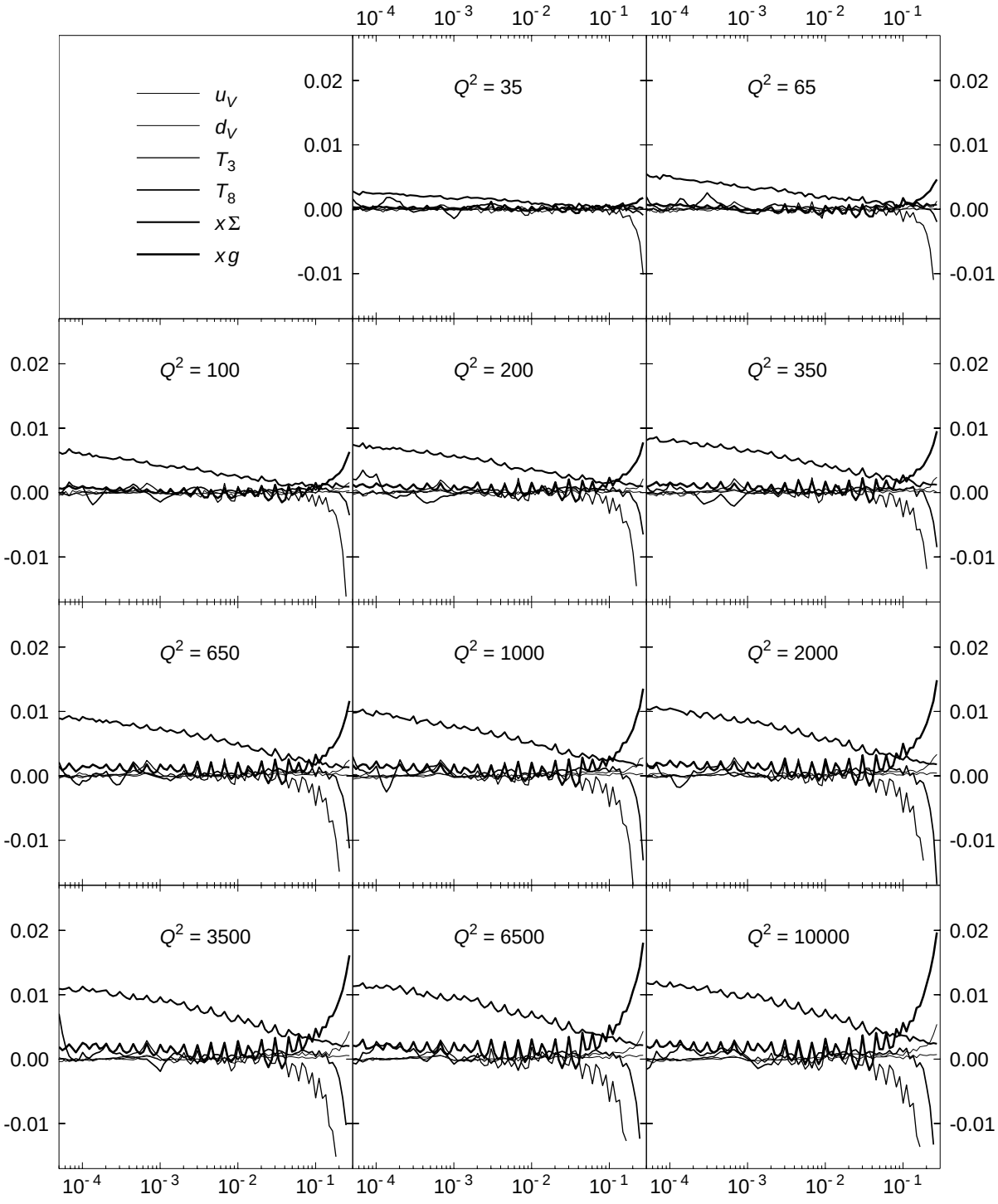


Figure 4.2: Relative error between our evolution and the GRV98 one (at leading order)

4.3 Comparison with experiment

Once we have an algorithm to evolve DGLAP numerically, one of the interesting things to do is to compare evolution results with experimental measurements[3] of F_2. This is done by choosing an initial density $q(x, Q_0^2)$ as a function of x depending on some parameters $a_0, a_1, \ldots, a_{n-1}$. For each value of the parameters, we can evolve and compare with the experimental data. We then have to fit the parameters a_i to produce a good χ^2.

The only tool we need, in addition to the ones we used in DGLAP resolution, is a minimization algorithm. In our program, we used the MINUIT program from the CERN library. It is used in batch mode and we refer to the CERN documentation [37] for more information about MINUIT and its usage.

4.4 Application to F_2 fit

4.4.1 Description

Basically, we applied our DGLAP [20] evolution equations solver to fit the F_2 structure function. Starting from some given parametrization at an initial scale, we evolve the proton distribution functions using DGLAP and compare with experimental values of F_2.

4.4.2 Installation

Requirements

This program requires

- a C++ compiler like gcc [30],
- the PACKLIB library from CERNLIB [31],
- OpenGL libraries (if you want to use the GL [32] extension) and the `glut` toolkit from Mesa [33]

Compilation

The CERNLIB libraries are supposed to be installed in the following directories

- libraries in `/usr/local/cern/lib`,
- C headers in `/usr/local/cern/include`.

If it is not the case, you can change the library path through the `CERNLIB` variable. It is defined at the beginning of the file `/src/Makefile`. The header directory is controlled by `CERN_INCDIR` in the `src/base` Makefile.

Once the path is correctly configured, simply type `make` in the main directory. This will output the binary file `Solve` in the main directory.

[3] See a little bit below for experiments involved.

Platforms

I have tested installation for the following platforms

- Linux RedHat 7.1 [34]
- Linux Debian 2.2 [35]

The OpenGL module have been tested both with the Mesa library and GL library for NVIDIA GeForce2.

Please let me know if you have any comment for other platforms.

4.4.3 Execution and results

Execution

Executing `Solve` should launch Minuit in batch mode. If one enabled the OpenGL Applet, a new window should appear with the fit graphic. One can close it at the end of the program by pressing the ESC key.

Output

Once the program execution is finished, it produces 2 outputs:

- The fit result is in the file `Fit.dat`,
- The fit result at experimental data points is saved in `ResAtExp.dat`,
- The final value of the parameters is in `param.dat`.

You can change these filenames in `setup.h`.

4.4.4 Data

The experimental points should be located in the `ExpData` directory. The `F2p.dat` file contains 5 columns which are x, $Q^2(GeV^2)$, F_2, ΔF_2, and an index corresponding to the experiment as stated in the following table [38]

Index	Experiment
1	BCDMS
2	E665
3	EMC
4	H1
5	NMC
6	SLAC
7	WA25
8	ZEUS

4.4.5 OpenGL applet

In order to visualise graphically the fit evolution, a 3D-plot of the fit is shown in real-time. Since it depends on the OpenGL library, it is available optionally. If you want to enable (disable) the visualisation plug-in, you simply have to set `USE_OPENGL = 1` (`USE_OPENGL = 0`) in the file `src/Makefile`. By default, the plug-in is unabled. To compile the OpenGL files, you should also need to specify the X-Windows and OpenGL libraries location. If they are in unusual directories, you can set this through the `GLLIBS` variable in the same file.

Basically, the applet consists of a second window showing the current state of the fit through a surface in blue and the experimental points with error bars in red. When the windows appear, the vertical axis shows the F_2 value. It has been bounded between 0 and 2 to avoid divergences. The two horizontal axis are $\ln(x)$ and $\ln(Q^2)$. The point closest from the viewpoint corresponds to $x = 1$ and $Q^2 = Q_0^2$. The axis on the left corresponds to $\ln(x)$ and the one on the right to $\ln(Q^2)$.

The applet allows you to move around the plot. When the window is active, the keys are the following

Key	movement	description
7	positive y translation	move up
1	negative y translation	move down
4	negative x translation	move left
6	positive x translation	move right
8	positive z translation	move forward
5	negative z translation	move backward
$\rightarrow$	positive y rotation	turn right
$\leftarrow$	negative y rotation	turn left
$\uparrow$	positive x rotation	lower head
$\downarrow$	negative x rotation	raise head
a	positive z rotation	balance left
z	negative z rotation	balance right

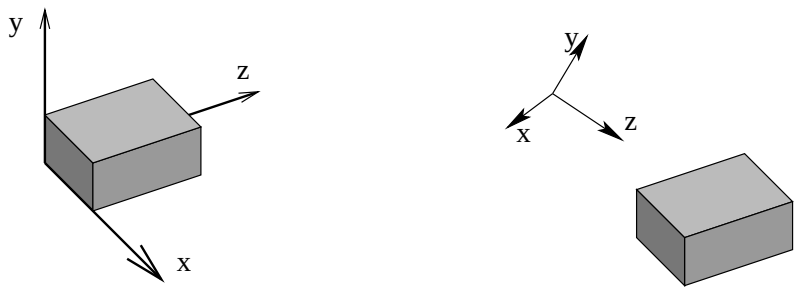
To quit the plug-in, simply click the `Esc` key. As usual in 3D rendering programs, for the observer reference frame, x-axis points to the right of the observer, y-axis to his head and z-axis in front of him as shown in the figure 4.3[4]

The source code for the OpenGL module can be found in the `src/gl` directory.

4.4.6 Personalization

Although some default parameters are set, you can widely modify the existing program. I will shortly describe here the main parameters you can easily change. Most of the constants on which the fit depend are defined in `src/base/setup.h`.

[4] This is a left-handed reference frame. Therefore, a positive z rotation, putting x onto y, is opposite to the usual interpretation.



(a) World (b) Observer

Figure 4.3: Reference frames used in OpenGL application.

Grid

The number of points in the (x, Q^2) grid are defined through `NB_X` and `NB_Q2` in `src/base/setup.h`. Default values are 104 and 32.

The minimal and maximal values of x and Q^2 are defined by `X_MIN`, `Q2_MIN` and `Q2_MAX` in the same file. Default values are 10^{-4}, 5 GeV^2 and 30000 GeV^2.

QCD

The QCD parameters such as the number of flavors (N_f) and the QCD scale Λ_{QCD} are defined in `src/dglap/QCD.h`.

Splittings

The DGLAP evolution splittings are defined in `src/dglap/Splitting.cpp`. They are cast on the form described by eq (4.3).

Parametrization

In order to apply DGLAP evolution, you need an initial condition. This is given by the distributions, as functions of x, at the initial scale Q^2_{min}.

These three functions are called `TDistrib`, `SDistrib` and `GDistrib` and are defined in the source file `src/base/param.cpp`. They take as argument the x value and the parameter values (as a zero-based indexed array).

The number of parameters is `NB_PARAM`, defined in `setup.h`.

Data set

The functions reading the experimental data file are grouped in the `src/base/readdata.cpp` file. The actual configuration accept the file format described before and only read the points inside the (x, Q^2) domain.

4.4.7 Future

Some tasks are planned for next release of this program. As example, we can consider

- using parametrizations using Dynamic Libraries,
- allowing a formula interpreter for parametrization,
- changing the grid size without recompiling,
- adding a DGLAP module to deal with an arbitrary number of equations,
- adding a minimization module independent from MINUIT,
- write a configuration script,
- NLO corrections to the DGLAP equations,
- adding an automatic backup of parameters value during the fit,
- considering Λ_{QCD} as a free parameter for the fit.

4.4.8 Contact

Please send Bugs or Reports at

g.soyez@ulg.ac.be

4.4.9 License

This program is under GPL version 2 [39]. A copy of the license should be given with the source code.

4.5 Example

To conclude this chapter, we will give an example of a fit produced by our application. We will produce an F_2^p fit using the MRST [21] initial parametrization. This means that we adopt the following initial condition

$$\begin{aligned} xu_V &= A_u(1+B_u\sqrt{x}+C_u x)x^{\delta_u}(1-x)^{\eta_u}, \\ xd_V &= A_d(1+B_d\sqrt{x}+C_d x)x^{\delta_d}(1-x)^{\eta_d}, \\ xS &= A_S(1+B_S\sqrt{x}+C_S x)x^{\delta_S}(1-x)^{\eta_S}, \\ xg &= A_g(1+B_g\sqrt{x}+C_g x)x^{\delta_g}(1-x)^{\eta_g} - A'_g x^{\delta'_g}(1-x)^{\eta'_g}, \\ x\Delta &= A_\Delta(1+B_\Delta\sqrt{x}+C_\Delta x)x^{\delta_\Delta}(1-x)^{\eta_\Delta}, \end{aligned} \tag{4.7}$$

with

$$\begin{aligned} 2\bar{u} &= \frac{2}{5}S - \Delta, \\ 2\bar{d} &= \frac{2}{5}S + \Delta, \\ s = \bar{s} &= \frac{1}{10}S. \end{aligned}$$

For the QCD parameters, we used 4 flavors and $\Lambda_{QCD} = 220\,\mathrm{MeV}$.

Using the $q^+(x,Q^2)$ densities, one can easily show that

$$d^+(x,Q^2) - u^+(x,Q^2) = d_V(x,Q^2) - u_V(x,Q^2) + 2\Delta(x,Q^2).$$

Since both $d^+ - u^+ \equiv -T_3$, d_V and u_V evolve like flavor non-singlet densities, i.e. non-coupled to the gluon density, the sea asymmetry $\Delta(x,Q^2)$ also evolves as a non-singlet quantity. Using these MRST definitions, the singlet density becomes

$$\Sigma = x(u_V + d_V + S),$$

and if we introduce

$$T = x(3u_V - 2d_V - 5\Delta)$$

which evolves like a non-singlet density, the proton structure function can be written

$$F_2(x,Q^2) = \frac{11\Sigma + 3T}{45}. \tag{4.8}$$

We started with the initial parametrization (4.7) at $Q^2 = 1\,\mathrm{GeV}^2$ and evolved using our algorithm[5]. The grid had 36 points in Q^2 going from 1 to 30 000 GeV^2 and 118 points in x going from $2.0\,10^{-5}$ to 1. Like for the MRST parametrization, we have only fitted the experimental points in the region

$$Q^2 > 2\,\mathrm{GeV}^2 \qquad \text{and} \qquad W^2 > 12.5\,\mathrm{GeV}^2.$$

This gives us 2054 experimental points.

After fitting the parameters of the initial distributions, we obtain the following χ^2

Experiment	χ^2	Nb points	χ^2/N
H1	435.734069	567	0.768490
ZEUS	634.954397	564	1.125806
BCDMS	227.721101	166	1.371814
E665	60.071223	53	1.133419
NMC	524.933698	494	1.062619
SLAC	181.578714	210	0.864661
Total	2064.993202	2054	1.005352

[5]We extended the evolution, which by default starts at 5 GeV^2 down to 1 GeV^2.

This fit is realized with the following parameter values

Parameter	Value	Error
A_u	1.4794e+00	9.7348e-03
B_u	8.9827e-07	1.6634e-01
C_u	1.5015e-01	2.2200e-02
δ_u	1.8443e-01	2.2454e-03
η_u	1.6766e+00	1.4843e-02
A_d	4.7855e+00	2.1556e-02
B_d	1.9998e+03	2.0490e+02
C_d	4.3475e+04	2.2252e+02
δ_d	2.3942e+00	1.8802e-03
η_d	2.7464e+01	3.1999e-02
A_S	8.1010e-01	3.5729e-02
B_S	-1.8385e+01	9.2034e-01
C_S	2.9507e+02	2.2296e+01
δ_S	-1.2304e-01	4.7450e-03
η_S	2.9502e+02	1.3852e+01
A_g	5.0429e+00	1.8724e-01
B_g	8.2058e+01	1.6733e-01
C_g	-9.7547e+01	2.0025e-01
δ_g	3.4205e+00	3.5513e-02
η_g	1.1812e+00	2.5991e-02
A'_g	-4.4882e-01	2.3079e-02
δ'_g	-1.8167e-01	1.0538e-02
η'_g	4.9056e+00	7.2829e-01
A_Δ	5.0000e+04	1.7502e+03
B_Δ	4.0808e+00	5.3084e-02
C_Δ	-2.6877e+00	3.0569e-01
δ_Δ	3.1073e+00	1.8243e-03
η_Δ	2.7911e+01	3.1682e-02

The figures of the next pages show the experimental points and the fit for F_2 as a function of x for several values of Q^2. It is important to underline the fact that this is just an illustration of our fit application, a simple example whose only objective is to show that we can effectively fit F_2 experimental data with an initial condition and the DGLAP evolution. The exact value of the fitted parameters is totally irrelevant since the contributions of the different quark densities can't be precise if we do not fit the neutron and deuteron structure functions, as done by MRST. These experiments, actually, force to consider the u_V, d_V and Δ distributions independently and thus constraints deeply their relative contributions to the T distribution.

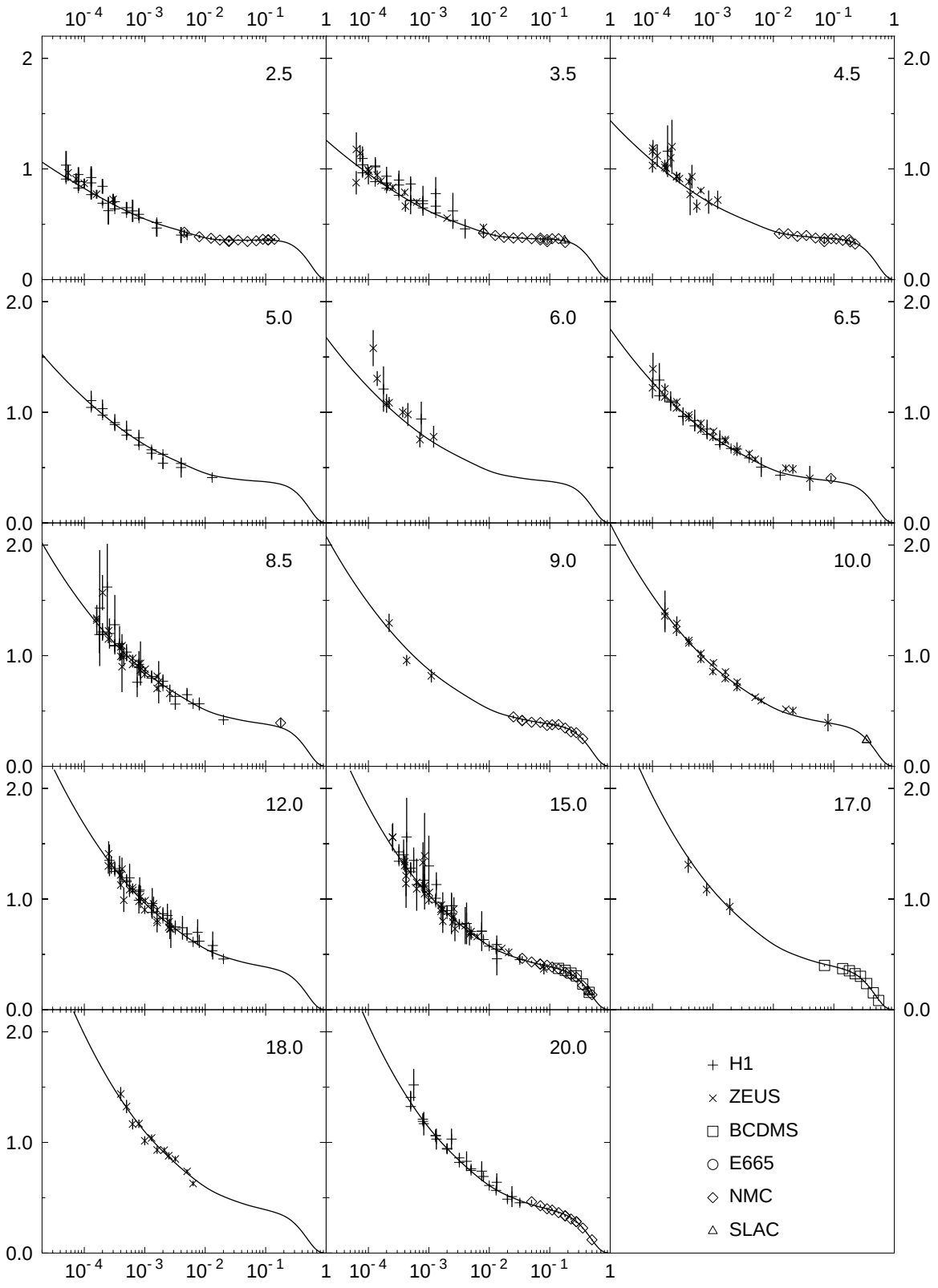


Figure 4.4: DGLAP fit with MRST parametrization. This shows F_2 as a function of x for low Q^2 values

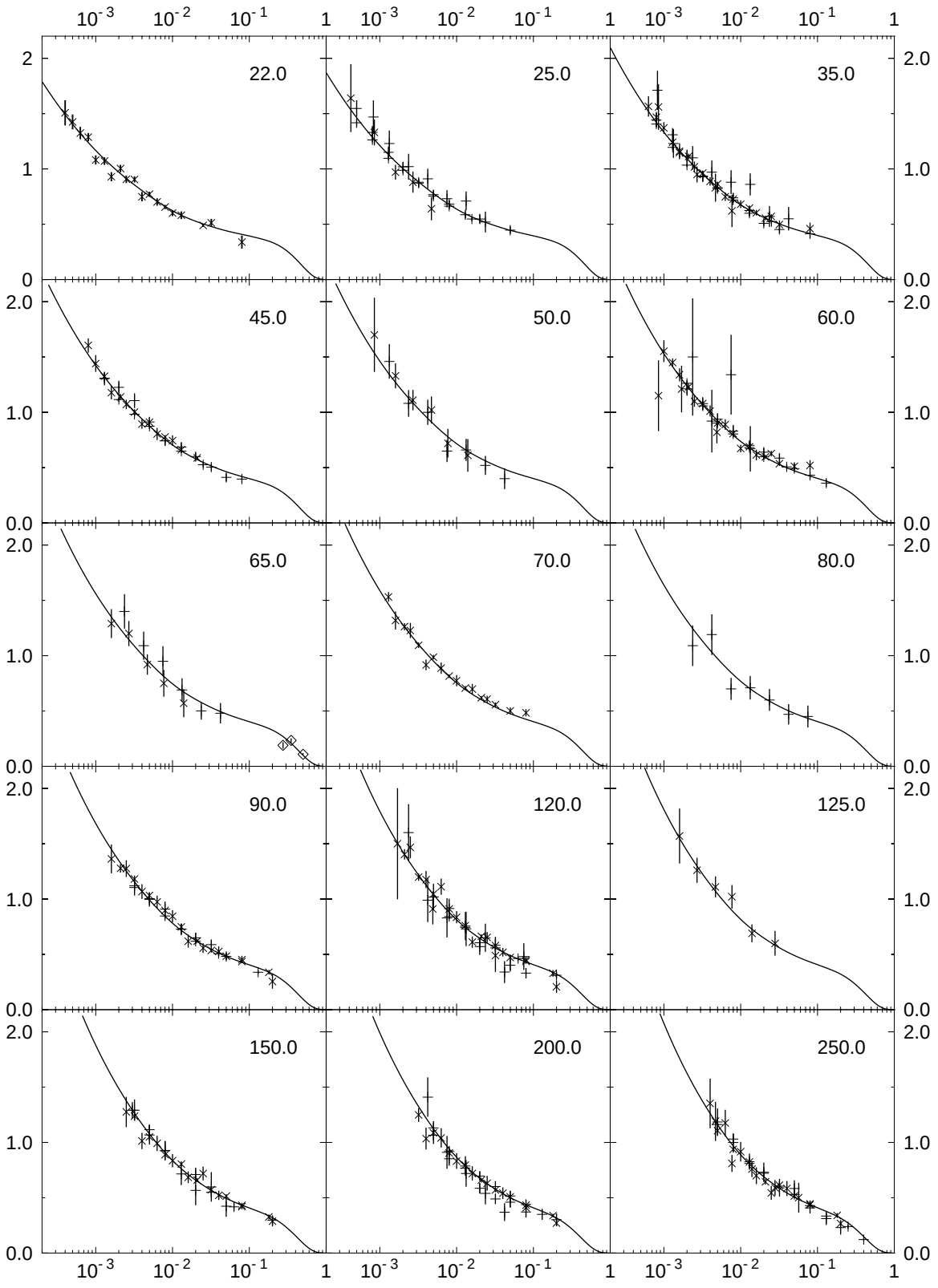


Figure 4.5: DGLAP fit with MRST parametrization. This shows F_2 as a function of x for middle Q^2 values

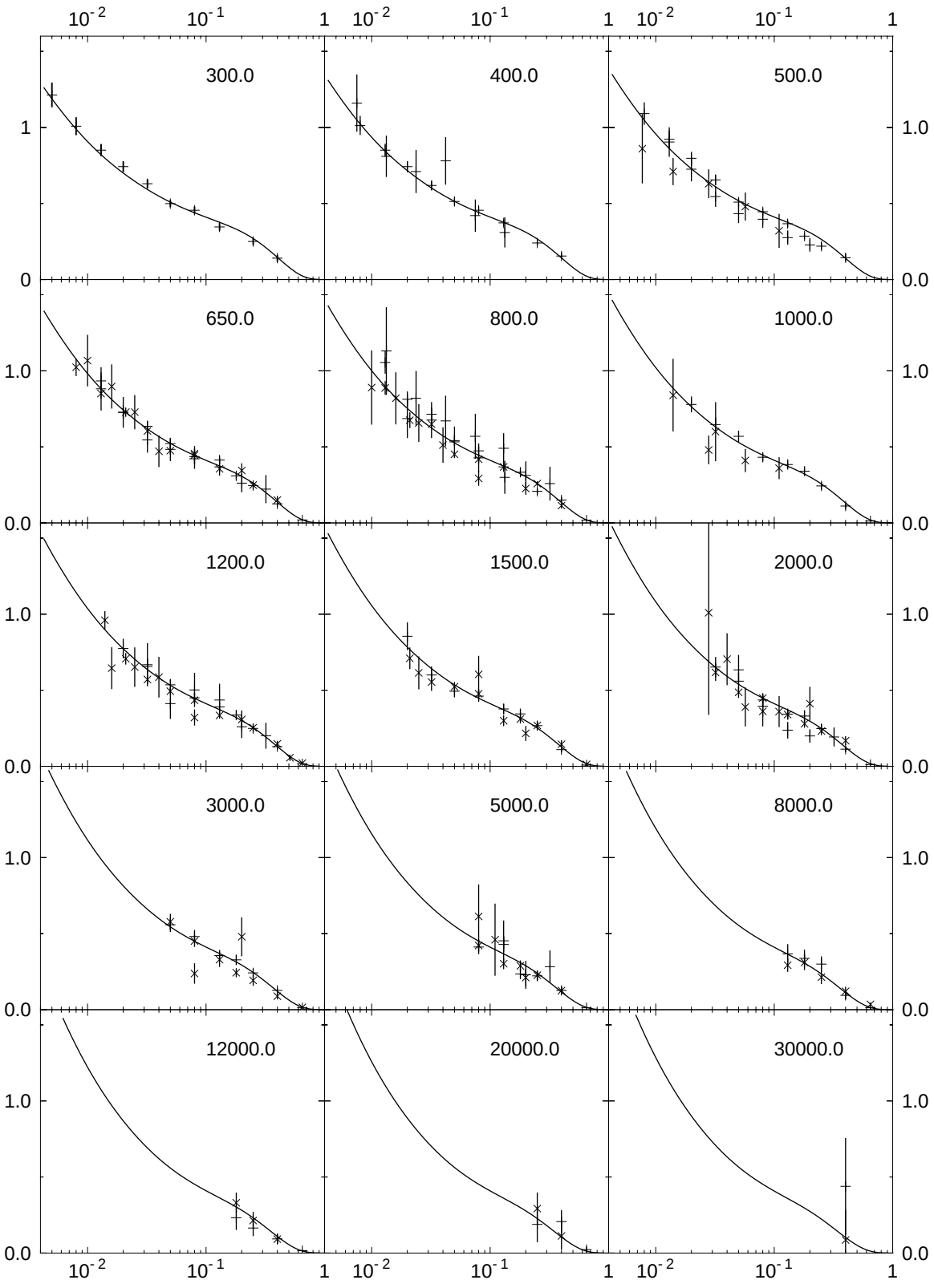


Figure 4.6: DGLAP fit with MRST parametrization. This shows F_2 as a function of x for high Q^2 values

Chapter 5

Conclusion

This report has presented an overview of all the main features of the DGLAP evolution equations.

After placing these equations in the context of perturbative QCD theory and of Deep Inelastic Scattering, we gave their derivation. We started our approach by an overview of the factorization theorem. This gives us a typical example where resummation of the perturbative expansion is required. The large Q^2 limit of the $\gamma^* q$ interaction gave rise to collinear divergences. By resumming these divergences, we could factorize the amplitude in a finite and a divergent part. Introducing a factorization scale in the divergent part, we have shown that renormalization group techniques were applicable to the factorized amplitude. The $\gamma^* p$ case was obtained by convoluting the $\gamma^* q$ with the bare parton densities. Reabsorption of the $\gamma^* q$ divergences into quark densities introduced the dressed densities and the anomalous dimensions. Finally, the requirement that the physical $\gamma^* p$ amplitude does not depend on the factorization scale directly lead to the DGLAP evolution equations for parton densities. These results are applicable at any order in perturbation theory. As an example of QCD calculation, we found the qq splitting functions at leading order. This was done by calculating the first order corrections to the $\gamma^* q$ amplitude, corresponding to the one-gluon-emission diagrams. We also gave the full set of splittings at leading order and introduced the non-singlet densities evolving alone, and the singlet density, evolving coupled with the gluon density. We ended that chapter by showing how the plus distribution is useful to separate the infra-red divergences, cancelled by virtual diagrams, from the collinear divergences.

In the next chapter, we discussed briefly the analytical properties of the DGLAP equations. This discussion was composed of two salient points. The analytical solution in Mellin space clearly highlighted the presence of an essential singularity above the factorization scale, in the small x limit. We then proved, in two different ways that the DGLAP kernel had no eigenfunctions.

Since, given a general initial condition in x space, we cannot find the analytical solution of the DGLAP equations at every Q^2, we developped an algorithm to perform that evolution numerically. After having explained the techniques used to evolve parton densities in x space, we have checked that our algorithm was correct and precise. This verification

was completed by starting the evolution at $Q^2 = 1\,\mathrm{GeV}^2$ with the GRV densities, and by comparing the densities obtained from their evolution to the one obtained from our evolution. The result shows a good agreement over the physical region considered.

With that algorithm at hand, we decided to apply DGLAP evolution to fit the experimental measurements of the F_2 structure function. We described in details how this application was written. We explained how to install, use and personalise it, and how to follow in real-time the evolution of the fit through an OpenGL applet. We also gave a list of the possible extensions of both our evolution algorithm and the fit application. As an example, we shown that, using the MRST initial parametrization at $Q^2 = 1\mathrm{GeV}^2$, we were able to fit the F_2 experimental data with a very good accuracy.

ACKNOWLEDGMENTS

This work was supported by National Fund for Scientific Research, Belgium. I would like to thank J.R. Cudell for a lot of fruitful discussions as well as for correcting some english mistakes in this document. I also thank R.S. Thorne for interesting information about DGLAP fits to experimental data, and G. Sterman for discussion about the factorization theorem.

Appendix A

Code for Evolution

We present here the interesting parts of the C code that performs the DGLAP evolution resolution.

A.1 QCD constants

This is simply a header file defining QCD usefull constants.

```
//--------------------------------------------------------------------//
//  File: QCD.h                                                       //
//  Description: QCD constants definition                             //
//  Version: 2.0                                                      //
//  Author: G. Soyez                                                  //
//  e-mail: G.Soyez@ulg.ac.be                                         //
//                                                                    //
// Copyright (C) 2002 G. Soyez                                        //
//--------------------------------------------------------------------//

#ifndef __QCD_H__
#define __QCD_H__

#include <math.h>

#define Nf 4
#define CA 3.0
#define CF 4.0/3.0
#define TR 0.5

#define LAMBDA_QCD 0.1763

inline double AlphaS(double Q2){
```

```
  return 12.0*M_PI/(33.0-2.0*Nf)/(Q2-log(LAMBDA_QCD*LAMBDA_QCD));
}

#endif
```

A.2 Splittings declaration

The C-source file implements the `CSplitting` object and the DGLAP LO Splittings.

```
//--------------------------------------------------------------------//
//  File: Splitting.cpp                                               //
//  Description: DGLAP splittings definition                          //
//  Version: 2.0                                                      //
//  Author: G. Soyez                                                  //
//  e-mail: G.Soyez@ulg.ac.be                                         //
//                                                                    //
// Copyright (C) 2002 G. Soyez                                        //
//--------------------------------------------------------------------//

#include "Splitting.h"
#include "QCD.h"

//--------------------------------//
// CSplitting Class implementation //
//  See CSplitting.h for details   //
//--------------------------------//

CSplitting::CSplitting(){
  IsInit=0;
}

CSplitting::CSplitting(ptrfunc InitRegul, ptrfunc InitDPlus, double InitDelta,
                       double InitDerP1){
  IsInit=Init(InitRegul, InitDPlus, InitDelta, InitDerP1);
}

CSplitting::~CSplitting(){

}

int CSplitting::Init(ptrfunc InitRegul, ptrfunc InitDPlus, double InitDelta,
                     double InitDerP1){
  Regul=InitRegul;
```

```
  DPlus=InitDPlus;
  Delta=InitDelta;
  DerP1=InitDerP1;
  return 1;
}


//-------------------------------------------//
// Declaration of all splitting functions used //
// !! WARNING !! we calculate x q(x,t), so we  //
//               use x P(x) as Splitting       //
//-------------------------------------------//

// Leading Log
//============

// The nonsinglet-nonsinglet contribution (non-singlet is alone in the DGLAP
// equation) !!!

double PTT_Regul(double x){ return 0.0; }
double PTT_DPlus(double x){ return CF*(1.0+x*x)*x; }

double PTT_Delta(){ return 1.5*CF; }
double PTT_DerP1(){ return 4.0*CF; }

// The singlet contribution (two coupled equations)

// singlet-singlet
double PSS_Regul(double x){ return 0.0; }
double PSS_DPlus(double x){ return CF*(1.0+x*x)*x; }

double PSS_Delta(){ return 1.5*CF; }
double PSS_DerP1(){ return 4.0*CF; }

// singlet-gluon
double PSG_Regul(double x){ return 2.0*Nf*TR*(2.0*x*(x-1.0)+1.0)*x; }
double PSG_DPlus(double x){ return 0.0; }

double PSG_Delta(){ return 0.0; }
double PSG_DerP1(){ return 0.0; }

// gluon-singlet
double PGS_Regul(double x){ return CF*(x*(x-2.0)+2.0); }
```

```
double PGS_DPlus(double x){ return 0.0; }

double PGS_Delta(){ return 0.0; }
double PGS_DerP1(){ return 0.0; }

// gluon-gluon
double PGG_Regul(double x){ return 2.0*CA*(1.0-x)*(1.0+x*x); }
double PGG_DPlus(double x){ return 2.0*CA*x*x; }

double PGG_Delta(){ return (11.0*CA-4.0*Nf*TR)/6.0; }
double PGG_DerP1(){ return 4.0*CA; }

// Nul distrib

double P0_Regul(double x){ return 0.0; }
double P0_DPlus(double x){ return 0.0; }

double P0_Delta(){ return 0.0; }
double P0_DerP1(){ return 0.0; }
```

A.3 ODE integration

The file `ODEint.cpp` contains the code needed to integrate the sets of ordinary differential equations. Since they are extremely close to the ones found in the Numerical Recipes manual, we won't give them explicitely here.

A.4 DGLAP evolution main file

A.4.1 Single equation

The 'public' functions really performing the DGLAP evolution, which are the ones you need to call if you want to use the evolution in your own code, are defined in the header file `evolve.h`.

```
//--------------------------------------------------------------------//
//  File: evolve.h                                                    //
//  Description: DGLAP evolution for a single equation                //
//  Version: 2.0                                                      //
//  Author: G. Soyez                                                  //
//  e-mail: G.Soyez@ulg.ac.be                                         //
//                                                                    //
```

```
// Copyright (C) 2002 G. Soyez                                          //
//----------------------------------------------------------------------//

#ifndef __EVOLVE_H__
#define __EVOLVE_H__

#include "Splitting.h"

typedef struct{
  CSplitting *P;     // The Splitting function for the DGLAP equation
  int Nb_x, Nb_Q2;   // number of x's, Q2's
  double Deltax;     // Space between ln(x) values
  double *x;         // Set of  x values
  double *Q2;        // set of Q2 values
  double **Res;      // result

  // integral variables
  double **KerR;
  double **KerP;

  // differential equation variables
  int iQ;            // index of Q2 while solving EqDiff
  double *yODE;

} DGLAP_Evolution_1;

// initialize splittings
int InitSplitting(ptrfunc, ptrfunc, double, double, DGLAP_Evolution_1*);

// initialize evolution
// you must allocate arrays and Nb* before calling this function
int InitEvolution(DGLAP_Evolution_1*);

// perform evolution
int Evolve(DGLAP_Evolution_1*);

// close evolution
int EndEvolve(DGLAP_Evolution_1*);

#endif
```

The C-source file `evolve.cpp` implements these functions

```
//-------------------------------------------------------------------//
//  File: evolve.cpp                                                 //
//  Description: DGLAP evolution for a single equation               //
//  Version: 2.0                                                     //
//  Author: G. Soyez                                                 //
//  e-mail: G.Soyez@ulg.ac.be                                        //
//                                                                   //
// Copyright (C) 2002 G. Soyez                                       //
//-------------------------------------------------------------------//

#include "evolve.h"
#include "QCD.h"
#include "ODEInt.h"
#include <stdio.h>

#define EPS_ODE   1e-5

// Global variables for evolution
DGLAP_Evolution_1 *Current_Evolution;

int InitKernel(DGLAP_Evolution_1 *e){
  int i,j;
  double deltax, weight;

  CSplitting *P;
  double *x, **KerP, **KerR;
  double Deltax;
  int Nb_x;

  Deltax=e->Deltax;
  Nb_x=e->Nb_x;
  x=e->x;
  P=e->P;

  KerR=new (double*)[Nb_x];
  KerP=new (double*)[Nb_x];
  for (i=0;i<Nb_x;i++){
    KerR[i]=new double[Nb_x];
    KerP[i]=new double[Nb_x];
    for (j=0;j<Nb_x;j++){
      KerR[i][j]=0.0;
```

```
    KerP[i][j]=0.0;
  }
}

/*
  We use Simpson 3/8 integration.
 */

for(i=0;i<Nb_x;i++){
  // Delta contribution
  KerR[i][i]+=P->Delta;

  if (i!=Nb_x-1){
    // inferior bound contribution
    deltax=x[i+1]-x[i];

    // weight coefficient at x[i] and x[N-1]
    if (i==Nb_x-2) weight=0.5;             // trapezoidal rule
    else if (i==Nb_x-3) weight=1.0/3.0;    // Simpson
    else if (i==Nb_x-4) weight=0.375;      // Simpson 3/8
    else if (i==Nb_x-5) weight=14.0/45.0; // Bode
    else weight=0.375;                     // Simpson 3/8


    // P_Reg contribution at x[i]
    KerR[i][i]+=0.5*P->Regul(1.0)*(deltax/x[i]);

    // P_Reg contribution at x[i+1]
    KerR[i][i+1]+=0.5*P->Regul(x[i]/x[i+1])*(deltax/x[i+1]);

    // P_+ contribution at x[i]
    KerP[i][i]+=Deltax*(P->DPlus(1.0)*(x[i+1]-2.0*x[i])/(x[i+1]-x[i])
                        -P->DerP1)*weight;
    KerP[i][i+1]+=Deltax*P->DPlus(1.0)*x[i]/(x[i+1]-x[i])*weight;

    // weight coefficient at x[N-1]
    if (i==Nb_x-2) weight=0.5;
    else if (i==Nb_x-4) weight=0.375;  // exactly one Simpson's 3/8 rule
    else weight=1.0/3.0;

    // P_+ contribution at x[N-1]
    KerP[i][Nb_x-1]+=Deltax*P->DPlus(x[i])/(1.0-x[i])*weight;
    KerP[Nb_x-1][i]-=Deltax*P->DPlus(1.0)*(x[i]/(1.0-x[i]))*weight;
```

```
      // extra P_+ contribution
      KerP[i][i]+=P->DPlus(1.0)*log(1.0-x[i]);

      // the remaining integral
      for (j=i+1;j<Nb_x-1;j++){
        deltax=x[j+1]-x[j];

        // weight coefficient at x[j]
        if (i==Nb_x-3) weight=4.0/3.0;   // Simpson
        else if (i==Nb_x-4) weight=1.125;     // Simpson 3/8
        else if (i==Nb_x-5){
          if (j==i+2) weight=24.0/45.0; else weight=64.0/45.0; // Bode
        } else {  // Simpson 3/8
          if ((j==i+1) || (j==Nb_x-2)) weight=7.0/6.0;
          else if ((j==i+2) || (j==Nb_x-3)) weight=23.0/24.0;
          else weight=1.0;
        }

        // P_Reg contribution at x[j]
        KerR[i][j]+=0.5*P->Regul(x[i]/x[j])*(deltax/x[j]);

        // P_+ contribution at x[j]
        KerP[i][j]+=Deltax*P->DPlus(x[i]/x[j])*(x[j]/(x[j]-x[i]))*weight;
        KerP[j][i]-=Deltax*P->DPlus(1.0)*(x[i]/(x[j]-x[i]))*weight;

        // P_Reg contribution at x[j]
        KerR[i][j+1]+=0.5*P->Regul(x[i]/x[j+1])*(deltax/x[j+1]);

      }
    }
  }

  e->KerP=KerP;
  e->KerR=KerR;

  return 0;
}

int InitSplitting(ptrfunc Regul, ptrfunc DPlus, double Delta, double DerP1,
                  DGLAP_Evolution_1 *e){
  e->P=new CSplitting();
  return e->P->Init(Regul, DPlus, Delta, DerP1);
```

```
}

int InitEvolution(DGLAP_Evolution_1 *e){
  e->Deltax = log(1.0/e->x[0])/(e->Nb_x-1.0);
  e->yODE=new double[e->Nb_x];
  InitKernel(e);
  e->iQ=0;

  return 0;
}

// this function uses Current_Evolution as evolution structure
int Integre(double Q2, double *f, double *out){
  int i,j,nmax;
  double **KerR, **KerP;

  KerR=Current_Evolution->KerR;
  KerP=Current_Evolution->KerP;

  nmax = Current_Evolution->Nb_x;
  for(i=0;i<Current_Evolution->Nb_x;i++){
    out[i]=(KerR[i][i]+KerP[i][i])*f[i];
    for (j=nmax-1;j>i;j--)
      out[i]+=(KerR[i][j]*f[j])+(KerP[i][j]*f[j]+KerP[j][i]*f[i]);
    out[i]*=AlphaS(Q2)/2.0/M_PI;
  }

  return 1;
}

int Diff(double Step, DGLAP_Evolution_1 *e){

  static int i, kount, nok, nbad;

  for (i=0;i<e->Nb_x;i++)
    e->yODE[i]=e->Res[e->iQ-1][i];

  Current_Evolution=e;
  if (!odeint(e->yODE, e->Nb_x, e->Q2[e->iQ-1], e->Q2[e->iQ], EPS_ODE, Step,
      0.0, &nok, &nbad, Integre, -1, &kount, NULL, NULL, 0.0))
    return 1;

  for (i=0;i<e->Nb_x;i++)
```

```
    e->Res[e->iQ][i]=e->yODE[i];

  return 0;
}

int Evolve(DGLAP_Evolution_1 *e){
  int i;

  for (i=1;i<e->Nb_Q2;i++){
    e->iQ=i;
    Diff(e->Q2[i]-e->Q2[i-1], e);
  }

  return 0;
}

int EndEvolve(DGLAP_Evolution_1 *e){
  int i;

  delete e->P;
  delete[] e->yODE;
  for (i=0;i<e->Nb_x;i++){
    delete[] e->KerR[i];
    delete[] e->KerP[i];
  }
  delete[] e->KerR;
  delete[] e->KerP;
  return 0;
}
```

A.4.2 Two coupled equations

This is the generalization of the previous files to a set of two coupled DGLAP equations. Further extensions to more than two equations can easily be performed from these two files. Here are the functions declaration in `evolve2.h`

```
//-------------------------------------------------------------------//
//  File: evolve2.h                                                  //
//  Description: DGLAP evolution for 2 coupled equations             //
//  Version: 2.0                                                     //
//  Author: G. Soyez                                                 //
//  e-mail: G.Soyez@ulg.ac.be                                        //
```

```
//                                                                      //
// Copyright (C) 2002 G. Soyez                                          //
//----------------------------------------------------------------------//

#ifndef __EVOLVE2_H__
#define __EVOLVE2_H__
#include "Splitting.h"

typedef struct{
  CSplitting *P[2][2];   // The Splitting function for the DGLAP equation
  int Nb_x, Nb_Q2;       // number of x's, Q2's
  double Deltax;         // Space between ln(x) values
  double *x;             // Set of  x values
  double *Q2;            // set of Q2 values
  double **Res[2];       // result

  // integral variables
  double **KerR[2][2];
  double **KerP[2][2];

  // differential equation variables
  int iQ;            // index of Q2 while solving EqDiff
  double *yODE;
} DGLAP_Evolution_2;

// initialize splittings
int InitSplitting2(ptrfunc, ptrfunc, double, double,
   ptrfunc, ptrfunc, double, double,
   ptrfunc, ptrfunc, double, double,
   ptrfunc, ptrfunc, double, double,
   DGLAP_Evolution_2*);

// initialize evolution
// you must allocate arrays and Nb* before calling this function
int InitEvolution2(DGLAP_Evolution_2*);

// perform evolution
int Evolve2(DGLAP_Evolution_2*);

// close evolution
int EndEvolve2(DGLAP_Evolution_2*);

#endif
```

The C-source file `evolve2.cpp` implements these functions

```
//-------------------------------------------------------------------//
//  File: evolve2.cpp                                                //
//  Description: DGLAP evolution for 2 coupled equations             //
//  Version: 2.0                                                     //
//  Author: G. Soyez                                                 //
//  e-mail: G.Soyez@ulg.ac.be                                        //
//                                                                   //
// Copyright (C) 2002 G. Soyez                                       //
//-------------------------------------------------------------------//

#include "evolve2.h"
#include "QCD.h"
#include "ODEInt.h"
#include <stdio.h>

#define EPS_ODE   1e-5

// Global variables for evolution
DGLAP_Evolution_2 *Current_Evolution2;

int InitKernel2(DGLAP_Evolution_2 *e){
  int i,j;
  int qi, qj;
  double deltax, weight;

  CSplitting *P;
  double *x, **KerP[2][2], **KerR[2][2];
  double Deltax;
  int Nb_x;

  Deltax=e->Deltax;
  Nb_x=e->Nb_x;
  x=e->x;

  for (qi=0;qi<2;qi++){
    for (qj=0;qj<2;qj++){
      P=e->P[qi][qj];

      KerR[qi][qj]=new (double*)[Nb_x];
      KerP[qi][qj]=new (double*)[Nb_x];
      for (i=0;i<Nb_x;i++){
```

```
  KerR[qi][qj][i]=new double[Nb_x];
  KerP[qi][qj][i]=new double[Nb_x];
  for (j=0;j<Nb_x;j++){
    KerR[qi][qj][i][j]=0.0;
    KerP[qi][qj][i][j]=0.0;
  }
}

/*
  We use Simpson 3/8 integration.
*/

for(i=0;i<Nb_x;i++){
  // Delta contribution
  KerR[qi][qj][i][i]+=P->Delta;

  if (i!=Nb_x-1){
    // inferior bound contribution
    deltax=x[i+1]-x[i];

    // weight coefficient at x[i] and x[N-1]
    if (i==Nb_x-2) weight=0.5;            // trapezoidal rule
    else if (i==Nb_x-3) weight=1.0/3.0;   // Simpson
    else if (i==Nb_x-4) weight=0.375;     // Simpson 3/8
    else if (i==Nb_x-5) weight=14.0/45.0; // Bode
    else weight=0.375;                    // Simpson 3/8


    // P_Reg contribution at x[i]
    KerR[qi][qj][i][i]+=0.5*P->Regul(1.0)*(deltax/x[i]);

    // P_Reg contribution at x[i+1]
    KerR[qi][qj][i][i+1]+=0.5*P->Regul(x[i]/x[i+1])*(deltax/x[i+1]);

    // P_+ contribution at x[i]
    KerP[qi][qj][i][i]+=Deltax*(P->DPlus(1.0)*(x[i+1]-2.0*x[i])
                        /(x[i+1]-x[i])-P->DerP1)*weight;
    KerP[qi][qj][i][i+1]+=Deltax*P->DPlus(1.0)*x[i]
                          /(x[i+1]-x[i])*weight;

    // weight coefficient at x[N-1]
    if (i==Nb_x-2) weight=0.5;
    else if (i==Nb_x-4) weight=0.375;  // exactly one Simpson's 3/8 rule
```

```
      else weight=1.0/3.0;

      // P_+ contribution at x[N-1]
      KerP[qi][qj][i][Nb_x-1]+=Deltax*P->DPlus(x[i])/(1.0-x[i])*weight;
      KerP[qi][qj][Nb_x-1][i]-=Deltax*P->DPlus(1.0)
                              *(x[i]/(1.0-x[i]))*weight;

      // extra P_+ contribution
      KerP[qi][qj][i][i]+=P->DPlus(1.0)*log(1.0-x[i]);

      // the remaining integral
      for (j=i+1;j<Nb_x-1;j++){
        deltax=x[j+1]-x[j];

        // weight coefficient at x[j]
        if (i==Nb_x-3) weight=4.0/3.0;   // Simpson
        else if (i==Nb_x-4) weight=1.125;     // Simpson 3/8
        else if (i==Nb_x-5){
          if (j==i+2) weight=24.0/45.0; else weight=64.0/45.0; // Bode
        } else {  // Simpson 3/8
          if ((j==i+1) || (j==Nb_x-2)) weight=7.0/6.0;
          else if ((j==i+2) || (j==Nb_x-3)) weight=23.0/24.0;
          else weight=1.0;
        }

        // P_Reg contribution at x[j]
        KerR[qi][qj][i][j]+=0.5*P->Regul(x[i]/x[j])*(deltax/x[j]);

        // P_+ contribution at x[j]
        KerP[qi][qj][i][j]+=Deltax*P->DPlus(x[i]/x[j])
                            *(x[j]/(x[j]-x[i]))*weight;
        KerP[qi][qj][j][i]-=Deltax*P->DPlus(1.0)
                            *(x[i]/(x[j]-x[i]))*weight;

        // P_Reg contribution at x[j]
        KerR[qi][qj][i][j+1]+=0.5*P->Regul(x[i]/x[j+1])*(deltax/x[j+1]);

      }
    }
  }

  e->KerP[qi][qj]=KerP[qi][qj];
  e->KerR[qi][qj]=KerR[qi][qj];
```

```
    }
  }

  return 0;
}

int InitSplitting2(ptrfunc Regul11, ptrfunc DPlus11,
                   double Delta11, double DerP111,
                   ptrfunc Regul12, ptrfunc DPlus12,
                   double Delta12, double DerP112,
                   ptrfunc Regul21, ptrfunc DPlus21,
                   double Delta21, double DerP121,
                   ptrfunc Regul22, ptrfunc DPlus22,
                   double Delta22, double DerP122,
                   DGLAP_Evolution_2 *e){
  e->P[0][0]=new CSplitting();
  e->P[0][1]=new CSplitting();
  e->P[1][0]=new CSplitting();
  e->P[1][1]=new CSplitting();
  e->P[0][0]->Init(Regul11, DPlus11, Delta11, DerP111);
  e->P[0][1]->Init(Regul12, DPlus12, Delta12, DerP112);
  e->P[1][0]->Init(Regul21, DPlus21, Delta21, DerP121);
  e->P[1][1]->Init(Regul22, DPlus22, Delta22, DerP122);
  return 0;
}

int InitEvolution2(DGLAP_Evolution_2 *e){

  e->Deltax = log(1.0/e->x[0])/(e->Nb_x-1.0);
  e->yODE=new double[2*e->Nb_x];
  InitKernel2(e);
  e->iQ=0;

  return 0;
}

// this function uses Current_Evolution as evolution structure
int Integre2(double Q2, double *f, double *out){
  int i,j,nmax;
  int qi, qj;
  double **KerR[2][2], **KerP[2][2];

  for (qi=0;qi<2;qi++){
```

```
    for (qj=0;qj<2;qj++){
      KerR[qi][qj]=Current_Evolution2->KerR[qi][qj];
      KerP[qi][qj]=Current_Evolution2->KerP[qi][qj];
    }
  }

  for (qi=0;qi<2;qi++)
    for(i=0;i<Current_Evolution2->Nb_x;i++)
      out[i+qi*Current_Evolution2->Nb_x]=0.0;

  nmax = Current_Evolution2->Nb_x;
  for (qi=0;qi<2;qi++){
    for(i=0;i<Current_Evolution2->Nb_x;i++){
      for (qj=0;qj<2;qj++){
        out[i+qi*Current_Evolution2->Nb_x]+=
            (KerR[qi][qj][i][i]+KerP[qi][qj][i][i])
            *f[i+qj*Current_Evolution2->Nb_x];
        for (j=nmax-1;j>i;j--)
          out[i+qi*Current_Evolution2->Nb_x]+=
             (KerR[qi][qj][i][j]*f[j+qj*Current_Evolution2->Nb_x])
            +(KerP[qi][qj][i][j]*f[j+qj*Current_Evolution2->Nb_x]
            +KerP[qi][qj][j][i]*f[i+qj*Current_Evolution2->Nb_x]);
      }
      out[i+qi*Current_Evolution2->Nb_x]*=AlphaS(Q2)/2.0/M_PI;
    }
  }

  return 1;
}

int Diff2(double Step, DGLAP_Evolution_2 *e){

  static int i, qi, kount, nok, nbad;

  for (qi=0;qi<2;qi++)
    for (i=0;i<e->Nb_x;i++)
      e->yODE[i+qi*e->Nb_x]=e->Res[qi][e->iQ-1][i];

  Current_Evolution2=e;
  if (!odeint(e->yODE, 2*e->Nb_x, e->Q2[e->iQ-1], e->Q2[e->iQ], EPS_ODE, Step,
              0.0, &nok, &nbad, Integre2, -1, &kount, NULL, NULL, 0.0))
    return 1;
```

```
  for (qi=0;qi<2;qi++)
    for (i=0;i<e->Nb_x;i++)
      e->Res[qi][e->iQ][i]=e->yODE[i+qi*e->Nb_x];

  return 0;
}

int Evolve2(DGLAP_Evolution_2 *e){
  int i;

  for (i=1;i<e->Nb_Q2;i++){
    e->iQ=i;
    Diff2(e->Q2[i]-e->Q2[i-1], e);
  }

  return 0;
}

int EndEvolve2(DGLAP_Evolution_2 *e){
  int i, qi, qj;

  for (qi=0;qi<2;qi++){
    for (qj=0;qj<2;qj++){
      delete e->P[qi][qj];
      for (i=0;i<e->Nb_x;i++){
        delete[] e->KerR[qi][qj][i];
        delete[] e->KerP[qi][qj][i];
      }
      delete[] e->KerR[qi][qj];
      delete[] e->KerP[qi][qj];
    }
  }

  delete[] e->yODE;
  return 0;
}
```

Bibliography

[1] N. N. Bogolyubov, B. V. Struminsky, and A. N. Tavkhelidze, preprint JINR-D-1968 (Dubna 1965).

[2] M. Gell-Mann, Phys. Lett. **8** (1964) 214.

[3] S. Weinberg, Phys. Rev. Lett. **31** (1973) 494.

[4] D.J. Gross, F. Wilczek, Phys. Rev. **D8** (1973) 3633.

[5] H. Fritzsch, M. Gell-Mann and H. Leutwyler, Phys. Lett. B **47** (1973) 365.

[6] F. Wilczek, in *C76-10-06.10* Print-76-1054 (PRINCETON) *Presented at Meeting of APS Div. of Particles and Fields, Brookhaven National Lab., Upton, N.Y., Oct 6-8, 1976.*

[7] H1 Collaboration: C. Adloff *et al.*, Eur. Phys. J. **C13** (2000) 609.

[8] H1 Collaboration: C. Adloff *et al.*, Eur. Phys. J. **C19** (2001) 269.

[9] H1 Collaboration: C. Adloff *et al.*, Eur. Phys. J. **C21** (2001) 33.

[10] ZEUS Collaboration: J. Breitweg *et al.*, Eur. Phys. J. **C12** (2000) 35.

[11] ZEUS Collaboration: S. Chekanov *et al.*, Eur. Phys. J. **C21** (2001) 443.

[12] BCDMS Collaboration: A. C. Benvenuti *et al.*, Phys. Lett. B **223** (1989) 485.

[13] EMC Collaboration: Aubert *et al.*, Nucl. Phys. **B259** (1985) 189.

[14] EMC Collaboration: Bazizi *et al.*, CERN-PPE/90-175 (1990).

[15] E665 Collaboration: Adams *et al.*, Phys. Rev. **D54** (96) 3006.

[16] NMC Collaboration: M. Arneodo *et al.*, Nucl. Phys. **B483** (1997) 3; Nucl. Phys. **B487** (1997) 3

[17] SLAC Experiments: Whitlow, PL **B282** (92) 475 and SLAC-357 (1990).

[18] WA25 Collaboration: Allasia *et al.*, Zeit. Phys. **C28** (1991) 321.

[19] J. D. Bjorken and E. A. Paschos, Phys. Rev. **185** (1969) 1975.

[20] V.N. Gribov and L.N. Lipatov, *Sov. J. Nucl. Phys.* **15** (1972) 438.
G. Altarelli and G. Parisi, *Nucl. Phys.* **B126** (1977) 298.
Yu.L. Dokshitzer, *Sov. Phys. JETP* **46** (1977) 641.

[21] A. D. Martin, R. G. Roberts, W. J. Stirling and R. S. Thorne, arXiv:hep-ph/0201127.

[22] G. Curci, W. Furmanski and R. Petronzio, Nucl. Phys. B **175** (1980) 27.

[23] T. Regge, Nuovo Cim. **14** (1959) 951.

[24] T. Regge, Nuovo Cim. **18** (1960) 947.

[25] T. Kinoshita, J. Math. Phys. **3**, 650 (1962).

[26] R.K. Ellis, W.J. Stirling and B.R. Webber, QCD and Collider Physics, (Cambridge University Press, Cambridge, 1996), p. 109.

[27] F.G. Tricomi, Pure and Applied Mathematics, Vol V, Integral Equations, (Interscience Publishers, 1967), pp. 5-15.

[28] M. Gluck, E. Reya and A. Vogt, Eur. Phys. J. C **5** (1998) 461 [arXiv:hep-ph/9806404].

[29] http://cpt19.dur.ac.uk/hepdata/grv.html.

[30] http://gcc.gnu.org.

[31] http://wwwinfo.cern.ch/asd/cernlib.

[32] http://www.sgi.com/software/opengl or http://www.opengl.org.

[33] http://www.mesa3d.org.

[34] http://www.redhat.com.

[35] http://www.debian.org.

[36] http://www.nr.com.

[37] http://wwwinfo.cern.ch/asdoc/minuit (or http://wwwinfo.cern.ch/asdoc if you want to download the manual in Postscript).

[38] http://www-spires.dur.ac.uk/hepdata/online/f2.

[39] http://www.gnu.org/copyleft/gpl.html.

Contents