Zeus Note 97-066

QCDNUM16: A fast QCD evolution program

M.A.J. Botje
NIKHEF
PO Box 41882, 1009DB Amsterdam, the Netherlands

January 19, 1998

(QCDNUM version 16.10 and 16.11)

Abstract

Qcdnum is a program which numerically evolves parton distributions using the Altarelli Parisi QCD evolution equations in LO or in NLO in the $\overline{\rm MS}$ scheme. From the evolved distributions the structure functions F_2, F_L and xF_3 can be calculated. The program can handle flavor thresholds at fixed values of Q^2 (light quark variable flavor number scheme) or, alternatively, evolve the three light quarks only and calculate the charm and bottom contributions to the structure functions from the photon-gluon fusion process (heavy quark fixed flavor number scheme). To study the scale uncertainties it is possible to independently vary the renormalization scale and the mass factorization scale. In this report we describe in detail how to use Qcdnum in a QCD analysis of (unpolarized) structure functions.

Contents

1 Introduction

Qcdnum is a fast QCD evolution program which provides:

- Calculation of the Q^2 evolution of α_s up to NLO.
- Q^2 evolution of the gluon, singlet and non-singlet distributions up to NLO.
- Calculation of F_2, F_L and xF_3 up to NLO.
- Calculation of the heavy quark contributions to F_2 and F_L up to NLO.
- Possibility to vary the mass factorization or renormalization scale.

In NLO the calculations are carried out in the $\overline{\mathrm{MS}}$ scheme.

Qcdnum has a long history[1]. The program was originally developed by members of the BCDMS collaboration [1] and was used, for instance, in the QCD analysis of SLAC/BCDMS F_2 structure functions [2]. Qcdnum was later on upgraded for use at low x by the NMC [3]. At this stage the program could only run on vectorized machines like the CRAY. A complete revision for the QCD analysis of ZEUS F_2 data [4] led to fast devectorized code which can run on any machine. During the 1995–1996 Hera workshop results from Qcdnum were compared to those from several other NLO evolution codes and found to be in agreement to within 0.05% [5].

The present version of Qcdnum can be regarded as a 'black box' which performs numerical calculations with reasonable speed and accuracy. A user interface allows to interact with the program. A QCD analysis is therefore not performed by Qcdnum itself but by user code instructing Qcdnum what actions to take.

This report is organized as follows: in section 2 we give a brief outline of the formalism underlying a QCD analysis of structure functions. Section 3 is devoted to the numerical method used by Qcdnum. Section 4 (user manual) describes how to use Qcdnum in a QCD analysis. Some details on how to control accuracy and speed are given in section 5 whereas section 6 (reference section) contains a short description of all subroutine calls.

2 The QCD analysis of structure functions

2.1 QCD evolution of α_s

The renormalization scale ($\mu^2 \equiv \mu_R^2$) dependence of the strong coupling constant α_s is governed by the renormalization group equation which reads in next-to-leading order (NLO):

$$\frac{\partial a_s(\mu^2)}{\partial \ln \mu^2} = -\beta_0 a_s^2(\mu^2) - \beta_1 a_s^3(\mu^2) \tag{1}$$

where $a_s \equiv \alpha_s/4\pi$ and the QCD beta functions are given by $\beta_0 = 11-2f/3$ and $\beta_1 = 102-38f/3$ with f the number of active quark flavors.

[1] The first entry in the Qcdnum history record dates from January 1987.

The solution of eq. (1) can be written as

$$\frac{1}{a_s(\mu^2)} + \frac{\beta_1}{\beta_0} \ln\left[\frac{\beta_1 a_s(\mu^2)}{\beta_0 + \beta_1 a_s(\mu^2)}\right] = \beta_0 \ln\left(\frac{\mu^2}{\Lambda^2}\right) \tag{2}$$

where we have introduced the QCD scale parameter Λ which is defined here as the scale where the left hand side of eq. (2) vanishes. There are however other definitions of Λ which are widely used: we prefer to remove this ambiguous scale parameter by writing eq. (2) in terms of the value of a_s at some input scale μ_0^2:

$$\frac{1}{a_s(\mu^2)} = \frac{1}{a_s(\mu_0^2)} + \beta_0 \ln\left(\frac{\mu^2}{\mu_0^2}\right) - \frac{\beta_1}{\beta_0} \ln\left\{\frac{a_s(\mu^2)[\beta_0 + \beta_1 a_s(\mu_0^2)]}{a_s(\mu_0^2)[\beta_0 + \beta_1 a_s(\mu^2)]}\right\}. \tag{3}$$

The beta functions depend on the number of flavors $f = (3,\ 4,\ 5)$. The flavor thresholds are usually set at $\mu_{c,b}^2 = m_{c,b}^2$. Thus evolving downwards from $\alpha_s(M_Z^2)$ for instance, Qcdnum calculates $\alpha_s(m_b^2)$ from eq. (3) with $f = 5$ flavors. This value of α_s is then taken as an input

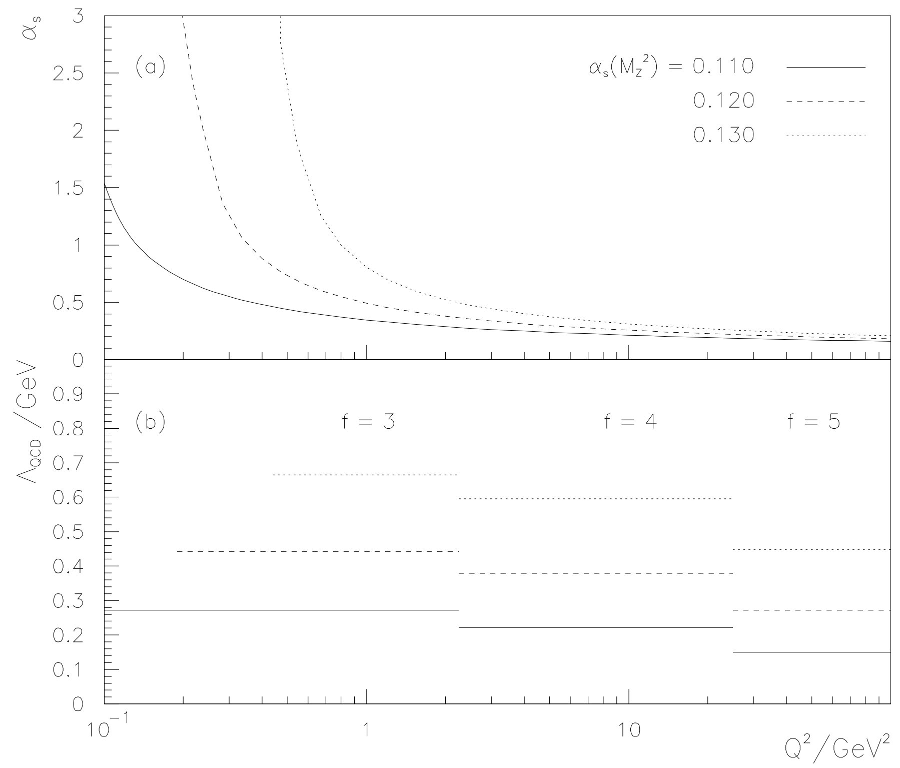

Figure 1: (a) The evolution of α_s in NLO in the $\overline{\mathrm{MS}}$ scheme starting from $\alpha_s(M_Z^2) = (0.110, 0.120, 0.130)$ with the quark mass thresholds taken to be $m_{c,b} = (1.5, 5)$ GeV. (b) The corresponding values of the QCD scale parameter Λ.

to evolve down to m_c^2 with $f = 4$ flavors and so on. In this way, α_s is continuous at the flavor thresholds but its derivative is not.

Fig. 1a shows the evolution of α_s for three input values of $\alpha_s(M_Z^2) = 0.110$, 0.120 and 0.130. The quark mass thresholds were set to m_c $(m_b) = 1.5$ (5) GeV. The values of the QCD scale parameter Λ, as defined in eq. (2), are shown in fig. 1b. Notice that Λ is discontinuous at the flavor thresholds. Notice also from eq. (2) that α_s cannot be evolved to values of μ^2 at or below the scale Λ^2, for instance not below $\mu^2 = 0.44$ GeV^2 if $\alpha_s(M_Z^2) = 0.130$ and if the flavor thresholds are taken to be those given above.

2.2 QCD evolution of parton densities

The Altarelli Parisi evolution equations [6] can be written as

$$\frac{\partial h_i(x,\mu^2)}{\partial \ln \mu^2} = \frac{\alpha_s(\mu^2)}{2\pi} \sum_{j=-f}^{f} \int_x^1 \frac{dz}{z} P_{ij}\left(\frac{x}{z}\right) h_j(z,\mu^2). \tag{4}$$

Here $\mu^2 \equiv \mu_M^2$ is the mass factorization scale,[2] $h_i(x,\mu^2)$ are the parton density distributions, $P_{ij}(x)$ are the QCD splitting functions and f is the number of active flavors. When the index $i > 0$, h_i denotes the quark distribution $q_i(x,\mu^2)$ of flavor $i =$ d, u, s,... ; for $i < 0$ it denotes the corresponding anti-quark distribution $\bar{q}_i(x,\mu^2)$ and for $i = 0$ it denotes the gluon distribution $g(x,\mu^2)$. In the quark parton model, and also in leading order (LO) perturbative QCD, the parton density distributions are defined such that $h_i(x,\mu^2)dx$ is, at a given μ^2, the number of partons which carry a fraction of the nucleon momentum between x and $x+dx$. The distribution $xh_i(x,\mu^2)$ is then the (fractional) momentum density. Beyond LO QCD no such intuitive interpretation of parton distributions is possible: they become theoretical constructs and their definition depends on the renormalization and factorization scheme in which the calculations are carried out ($\overline{\text{MS}}$ in Qcdnum).

If the x dependences of the parton densities are known at some fixed value of $\mu^2 = \mu_0^2$, they can be evolved to any value of μ^2 using eq. (4) which consists of $2f + 1$ coupled integro-differential equations. Using symmetries in the splitting functions (see e.g. [7]) a much more simple set of equations can be written for the evolution of the flavor singlet and flavor non-singlet quark distributions. The singlet quark distribution is defined as the sum of all quark and anti-quark densities[3]

$$\Sigma(x) = \sum_{i=1}^{f} [q_i(x) + \bar{q}_i(x)] \tag{5}$$

and obeys an evolution equation which is coupled to that of the gluon:

$$\begin{aligned} \frac{\partial \Sigma(x)}{\partial \ln \mu^2} &= \frac{\alpha_s}{2\pi} \int_x^1 \frac{dz}{z} \left\{ P_{FF}\left(\frac{x}{z}\right) \Sigma(z) + P_{FG}\left(\frac{x}{z}\right) g(z) \right\} \\ \frac{dg(x)}{d \ln \mu^2} &= \frac{\alpha_s}{2\pi} \int_x^1 \frac{dz}{z} \left\{ P_{GF}\left(\frac{x}{z}\right) \Sigma(z) + P_{GG}\left(\frac{x}{z}\right) g(z) \right\}. \end{aligned} \tag{6}$$

There are two types of non-singlet distributions:

$$q_i^-(x) = q_i(x) - \bar{q}_i(x) \quad \text{(valence distribution)} \tag{7}$$

[2]In this section we set the renormalization scale μ_R^2 (see section 2.1) equal to μ_M^2.

[3]Here and in the following we will suppress the argument μ^2 and write α_s for $\alpha_s(\mu^2)$, $q(x)$ for $q(x,\mu^2)$, etc.

$$q_i^+(x) \;=\; q_i(x) + \bar{q}_i(x) - \frac{1}{f}\Sigma(x).$$

The evolution of these distributions does not depend on the gluon:

$$\frac{\partial q_i^{\pm}(x)}{\partial \ln \mu^2} = \frac{\alpha_s}{2\pi}\int_x^1 \frac{dz}{z} P_{\pm}\left(\frac{x}{z}\right) q_i^{\pm}(z). \tag{8}$$

This equation is linear in the quark density so that any linear combination of the q_i^+ (q_i^-) is again a non-singlet distribution of the type q^+ (q^-) which obeys eq. (8); in the following we denote by Δ_{ij} the non-singlet combination

$$\Delta_{ij}(x) = q_i^+(x) - q_j^+(x) = [q_i(x) + \bar{q}_i(x)] - [q_j(x) + \bar{q}_j(x)]\,. \tag{9}$$

Notice that the sum of the q^+ distributions vanishes: $\sum_f q_i^+ = 0$.

The splitting functions can be expanded in a perturbative series in α_s which presently is calculated up to next-to-leading order (NLO) [8, 9]:

$$\begin{aligned} P_{\pm}(x) &= P_{FF}^{(0)}(x) + \frac{\alpha_s}{2\pi} P_{\pm}^{(1)}(x) \\ P_{AB}(x) &= P_{AB}^{(0)}(x) + \frac{\alpha_s}{2\pi} P_{AB}^{(1)}(x) \end{aligned} \tag{10}$$

where AB stands for FF, FG, GF or GG. The expressions for the LO splitting functions $P^{(0)}$ are given in appendix A. Those for the NLO splitting functions $P^{(1)}$ are complicated: results calculated the $\overline{\mathrm{MS}}$ renormalization scheme are given in refs. [8] (non-singlet case) and [9] (singlet case).

Like for the beta functions introduced in section 2.1 the number of active flavors, f, enters as a parameter in several LO and NLO splitting functions. The parton distributions $h_i(x, \mu^2)$ are continuous functions of μ^2 but the slope $\partial h_i(x, \mu^2)/\partial \ln \mu^2$ changes when crossing a flavor threshold. This is also true for Σ, q^- and Δ but *not* for q^+: this distribution is – per definition – discontinuous, see eq. (7).

It is important to notice that Qcdnum can only evolve the singlet/gluon and non-singlet distributions. Only such distributions should therefore be passed as input to the program.[4] Any linear combination of quark and anti-quark distributions can be decomposed into a linear combination of the singlet density Σ and one or more non-singlet densities:

$$\sum_f (a_i q_i + b_i \bar{q}_i) = <c> \Sigma + \sum_f c_i q_i^+ + \sum_f d_i q_i^- \tag{11}$$

$$\text{with} \quad c_i = \frac{a_i + b_i}{2}, \quad d_i = \frac{a_i - b_i}{2} \quad \text{and} \quad <c> = \frac{1}{f}\sum_f c_i.$$

As an example we take the proton distribution in charged lepton scattering which can be written as

$$\begin{aligned} q^{\ell p}(x) &= \sum_f e_i^2 \left[q_i(x) + \bar{q}_i(x)\right] \\ &= <e^2> \Sigma(x) + q_{\ell p}^{ns}(x) \end{aligned} \tag{12}$$

[4] To be more precise: one should pass the parton *momentum* densities xg, $x\Sigma$, xq^+ etc. instead of the parton *number* densities g, $\Sigma, \ldots$ used in this and the next sections.

where e_i is the quark charge of flavor (i) in units of the electron charge and $<e^2>\equiv (1/f)\sum e_i^2$ is the average of the square of the quark charges: $<e^2> = (4/18,\ 5/18,\ 11/45)$ for $f = (3,\ 4,\ 5)$ flavors. The distribution $q^{ns}_{\ell p}$ is a pure q^+-type non-singlet:[5]

$$q^{ns}_{\ell p}(x) = \sum_f e_i^2 q_i^+(x) \tag{13}$$

Because $q^{\ell p}$ and Σ are continuous functions of μ^2 it follows that the change in $<e^2>$ must be compensated for by a discontinuity in $q^{ns}_{\ell p}$ when crossing a flavor threshold. We find

$$\begin{aligned} q^{ns}_{\ell p}(f=4) &= q^{ns}_{\ell p}(f=3) - \frac{1}{18}\Sigma \\ q^{ns}_{\ell p}(f=5) &= q^{ns}_{\ell p}(f=4) + \frac{1}{30}\Sigma. \end{aligned} \tag{14}$$

In this way the proton distribution for all μ^2 can be obtained from a singlet/gluon evolution together with that of only one q^+-type non-singlet, provided the threshold discontinuities in the latter are taken properly into account. See section 2.5 for further details on heavy flavor thresholds.

Below we give another decomposition of the proton quark distribution:

$$q^{\ell p}_{f=3} = \frac{4}{18}\Sigma + \frac{1}{6}\Delta_{ud} - \frac{1}{6}q_s^+ \tag{15}$$

$$q^{\ell p}_{f=4} = \frac{5}{18}\Sigma + \frac{1}{6}\Delta_{ud} - \frac{1}{6}q_s^+ + \frac{1}{6}q_c^+ \tag{16}$$

$$q^{\ell p}_{f=5} = \frac{11}{45}\Sigma + \frac{1}{6}\Delta_{ud} - \frac{1}{6}q_s^+ + \frac{1}{6}q_c^+ - \frac{1}{6}q_b^+. \tag{17}$$

Such a decomposition is not very economic since it requires up to four non-singlet evolutions. It is given here only because it will be used in examples later on.

2.3 Structure function evaluation

In NLO (in the $\overline{\mathrm{MS}}$ scheme) the relation between the singlet quark distribution and the structure functions

$$F_2^s \text{ and } F_L^s \equiv F_2^s - 2xF_1^s$$

is given by [7]:

$$F_k^s(x,Q^2) = \delta_{k2}\, x\Sigma(x,Q^2) + \frac{\alpha_s(Q^2)}{2\pi} x \int_x^1 \frac{dz}{z} \left\{ C^{(1)}_{k,q}\left(\frac{x}{z}\right)\Sigma(z,Q^2) + C^{(1)}_{k,g}\left(\frac{x}{z}\right) g(z,Q^2) \right\} \tag{18}$$

with $k = 2, L$. Here we have introduced the 'physical' scale Q^2 which in deep inelastic scattering is the negative square of the four-momentum transferred from the initial lepton to the target. In eq. (18) the renormalization scale μ_R^2 (see section 2.1), the mass factorization scale μ_M^2 (see section 2.2) and Q^2 are assumed to be equal: $\mu_R^2 = \mu_M^2 = Q^2$.[6]

[5] We have $a_i = b_i$ in eq. (11) so that the non-singlet valence distributions q_i^- do not contribute to $q^{\ell p}$.

[6] See section 2.4 for when they are chosen to be unequal.

For clarity we will drop the argument Q^2 in the following and write for the non-singlet contributions to F_k

$$F_k^{ns}(x) = \delta_{k2}\, xq^{ns}(x) + \frac{\alpha_s}{2\pi} x \int_x^1 \frac{dz}{z} C^{(1)}_{k,q}\left(\frac{x}{z}\right) q^{ns}(z) \tag{19}$$

where q^{ns} is a shorthand notation for q_i^+, q_i^- or any linear combination of these. These structure functions do not depend on the gluon distribution. The first term on the r.h.s of eq. (18) and (19) is the leading order contribution to F_k^s and F_k^{ns} respectively. From eqs. (18) and (19) the structure function for any linear combination

$$q^A = <c> \Sigma + q^{ns}$$

can be written as

$$F_k^A(x) = \delta_{k2}\, xq^A(x) + \frac{\alpha_s}{2\pi} x \int_x^1 \frac{dz}{z} \left\{ C^{(1)}_{k,q}\left(\frac{x}{z}\right) q^A(z) + <c> C^{(1)}_{k,g}\left(\frac{x}{z}\right) g(z) \right\}. \tag{20}$$

The structure function xF_3 is a pure non-singlet:

$$xF_3(x) = xq^{ns}(x) + \frac{\alpha_s}{2\pi} x \int_x^1 \frac{dz}{z} C^{(1)}_{3,q}\left(\frac{x}{z}\right) q^{ns}(z). \tag{21}$$

The coefficient functions $C^{(1)}_{k,q}(x)$, $C^{(1)}_{k,g}(x)$ and $C^{(1)}_{3,q}(x)$ are calculated in the $\overline{\text{MS}}$ scheme in [7] and are given in appendix A.

As an example we consider the proton $F_{2,L}$ structure functions in charged lepton scattering which can be calculated from eq. (20) using the quark distributions given in eqs. (12) and (13) in the previous section.[7] The neutron distribution is obtained by assuming isospin symmetry: q_u and q_d are interchanged in eq. (12). The deuteron quark density is usually defined as $q^d \equiv (q^p + q^n)/2$ so that

$$q^{\ell d}(x) = q^{\ell p}(x) - \frac{1}{6}\Delta_{ud}(x). \tag{22}$$

The distribution Δ_{ud} is thus constrained by the difference of proton and deuteron F_2 structure functions. To calculate $F^d_{2,L}$ we need to evolve one more non-singlet distribution (Δ_{ud}) in addition to $q^{ns}_{\ell p}$ defined in eq. (13).

The structure functions in neutrino scattering are usually given as the sums and differences of νN and $\bar{\nu} N$ results on isoscalar targets. Here the relevant quark distribution is $x\Sigma$ for $F_2^{\nu N}$ and

$$xq_v = x(u - \bar{u}) + x(d - \bar{d})$$

for $xF_3^{\nu N}$. To calculate the latter we need to evolve a third non-singlet parton distribution, xq_v.

2.4 Renormalization and factorization scale dependence

In the previous section we have assumed that the renormalization scale (μ_R^2) and the mass factorization scale (μ_M^2) are equal to Q^2. Here we discuss two cases where these scales are chosen to be different: (i) $\mu_R^2 = \mu_M^2 \neq Q^2$ and (ii) $\mu_R^2 \neq \mu_M^2 = Q^2$.

[7] We neglect here electroweak contributions from $\gamma^* Z^0$ interference and Z^0 exchange which become important above $Q^2 \sim 5000$ GeV^2. For a full description of charged and neutral current cross-sections and structure functions in deep inelastic scattering see [10].

Let us first vary the mass factorization scale with respect to Q^2 for instance $Q^2/4 < \mu_M^2 = \mu_R^2 < 4Q^2$. This variation affects the relation between the parton distributions and the structure functions through a modification of the NLO coefficient functions. The μ_M^2 dependence of a structure function F_k calculated at a given Q^2 is defined as

$$F_k^{(n)}(Q^2, \mu_M^2) = F_k^{(n)}(\mu_M^2) + \Delta F_k^{(n-1)}(Q^2, \mu_M^2) \tag{23}$$

where $F_k^{(n)}(\mu_M^2)$ is the structure function calculated up to order n at the scale μ_M^2 and $\Delta F_k^{(n-1)}$ is the change in F_k when moving from μ_M^2 to Q^2, this change being calculated at order $n-1$. Clearly if n is large enough so that $\Delta F_k^{(n-1)} \approx \Delta F_k^{(n)}$ we have

$$F_k^{(n)}(Q^2, \mu_M^2) \approx F_k^{(n)}(\mu_M^2) + \Delta F_k^{(n)}(Q^2, \mu_M^2) \equiv F_k^{(n)}(Q^2)$$

so that the μ_M^2 dependence vanishes for large n.

From eq. (23) we obtain for the scale dependence of the structure functions in LO (see eqs. (18)–(21))

$$\begin{aligned} F_{2,3}^{(1)}(Q^2, \mu_M^2) &= F_{2,3}^{(1)}(\mu_M^2) = xq(\mu_M^2). \\ F_L^{(1)}(Q^2, \mu_M^2) &= 0. \end{aligned} \tag{24}$$

Thus in LO one takes for the structure functions F_2 and xF_3 the corresponding quark distribution at μ_M^2 instead of at Q^2. For the scale dependence in NLO we get[8]

$$\begin{aligned} F_{2,3}^{(2)}(Q^2, \mu_M^2) &= xq(\mu_M^2) + \frac{\alpha_s(\mu_M^2)}{2\pi}\left[C_{k,q}^{(1)} + P^{(0)} \ln\left(\frac{Q^2}{\mu_M^2}\right)\right] \otimes xq(\mu_M^2) \\ F_L^{(2)}(Q^2, \mu_M^2) &= F_L^{(2)}(\mu_M^2). \end{aligned} \tag{25}$$

Notice that F_L when calculated in NLO still has a large LO type scale uncertainty.

Next, we set $\mu_M^2 = Q^2$ and vary μ_R^2. This leads to a change in the value of α_s (being taken at μ_R^2 instead of Q^2) and to a modification of the NLO splitting functions:

$$\frac{\partial q(Q^2, \mu_R^2)}{\partial \ln Q^2} = \frac{\alpha_s(\mu_R^2)}{2\pi}\left\{P^{(0)} + \frac{\alpha_s(\mu_R^2)}{2\pi}\left[P^{(1)} + \frac{\beta_0}{2}P^{(0)} \ln\left(\frac{\mu_R^2}{Q^2}\right)\right]\right\} \otimes q(Q^2, \mu_R^2) \tag{26}$$

The structure functions are calculated from the evolved parton distributions with

$$F_k(Q^2, \mu_R^2) = (1 - \delta_{kL})xq(Q^2, \mu_R^2) + \frac{\alpha_s(\mu_R^2)}{2\pi} C_{k,q}^{(1)} \otimes xq(Q^2, \mu_R^2). \tag{27}$$

The scale uncertainties are usually assigned to α_s and the parton distributions. These uncertainties are obtained by varying the scale and re-determining α_s and the parton densities in a fit where the structure functions are kept fixed. For determinations of the scale errors on α_s see [2], [11] and [12].

[8] To keep the notation simple we give in this section only the expressions for non-singlet structure functions and quark densities. Convolution integrals are indicated by the symbol '$\otimes$'.

2.5 Heavy flavor contributions

In Qcdnum there are two ways in which heavy flavors can be taken into account:

1. Treat the heavy quarks ($h =$ c, b) as light quarks but allow them to contribute only above a given threshold in Q^2. In the following we will denote these thresholds by Q_c^2 and Q_b^2 for charmed and bottom quarks respectively. This prescription is known as the 'light quark variable flavor number scheme'.

2. Do not introduce any thresholds but consider only three light flavors: the parton distributions are evolved with $f = 3$ and the structure functions F_2 or F_L are calculated from eq. (20). The heavy quark contributions F_2^h or F_L^h are calculated from the photon-gluon fusion process, including NLO corrections (see below). This prescription is known as the 'heavy quark fixed flavor number scheme'.

In the following we will describe both schemes in more detail and point out a few Qcdnum do's and dont's.

To describe the first approach let us assume that we want to calculate the F_2 structure function in charged lepton–proton scattering from parton distributions given at an input scale Q_0^2. We choose for the singlet/non-singlet decomposition of the proton quark distribution those given in section 2.2 eqs. (15, 16, 17) for $f = (3, 4, 5)$ flavors. The range of the evolution and the input scale Q_0^2 are set such that

$$Q^2_{min} < Q_c^2 < Q_0^2 < Q_b^2 < Q^2_{max}.$$

Input to the calculation are the parton distributions

$$g(x, Q_0^2),\ \Sigma(x, Q_0^2),\ \Delta_{ud}(x, Q_0^2) \text{ and } q_s^+(x, Q_0^2).$$

Furthermore since $q_c = 0$ for $Q^2 \leq Q_c^2$ and $q_b = 0$ for $Q^2 \leq Q_b^2$ we set the starting values of q_c^+ and q_b^+ at

$$q_c^+(x, Q_c^2) = -\frac{1}{4}\Sigma(x, Q_c^2) \text{ and } q_b^+(x, Q_b^2) = -\frac{1}{5}\Sigma(x, Q_b^2).$$

The analysis now proceeds as follows:

- Evolve the singlet/gluon distribution from Q_0^2 down to Q^2_{min} and up to Q^2_{max}. In the upward (downward) evolution the number of flavors changes from 4 to 5 (3) at Q_b^2 (Q_c^2). Qcdnum takes automatically care of this.

- Evolve Δ_{ud}: here also Qcdnum takes care of the number of flavors.

- Evolve q_s^+: this is more complicated because q_s^+ is discontinuous at the thresholds. We therefore evolve from Q_0^2 up to Q_b^2, add $(1/4 - 1/5)\Sigma(x, Q_b^2)$ to $q_s^+(x, Q_b^2)$ and continue the evolution from Q_b^2 up to Q^2_{max}. Likewise for downward evolution where we add $(1/4 - 1/3)\Sigma(x, Q_c^2)$ at the threshold Q_c^2.

- Evolve q_c^+: here we evolve from Q_c^2 upwards to Q^2_{max}, taking into account the discontinuity at Q_b^2 as described above. Downward evolution does not make sense because we have assumed that charm does not contribute below the threshold Q_c^2.[9] Thus you should either take care that q_c^+ below the charm threshold is not used in subsequent calculations or you have to set it explicitly to

$$q_c^+(x,Q^2) = -\frac{1}{3}\Sigma(x,Q^2), \quad Q^2 < Q_c^2.$$

- Evolve q_b^+: similar to q_c^+ except that we evolve upwards starting from Q_b^2.
- Calculate the F_2 structure function (for any x and Q^2 within the evolution limits). First we have to assemble the proton quark distribution using eqs. (15), (16) and (17) for the regions $Q^2 < Q_c^2$, $Q_c^2 < Q^2 < Q_b^2$ and $Q_b^2 < Q^2$ respectively. F_2 is then calculated from eq. (20). Qcdnum provides a mechanism to define objects like the 'proton' as linear combinations of quark distributions for $f = (3, 4, 5)$ flavors.[10]

The outline given above should be considered as an example to illustrate the handling of the flavor thresholds: in practice it would be easier (and more efficient) to use the proton non-singlet distribution defined in eq. (13) in section 2.2 instead of the Δ_{ud}, q_s^+, q_c^+ and q_b^+ used above.

An alternative treatment of heavy flavor threshold effects in charged lepton scattering is based on the calculation of the heavy flavor contributions to F_2 or F_L through, in NLO [13]:

$$F_k^h(x,Q^2) = \frac{\alpha_s}{2\pi}\left\{ e_h^2\, g\otimes\mathcal{C}_{k,g}^{(0)} + \frac{\alpha_s}{2\pi}\left(e_h^2\, g\otimes\mathcal{C}_{k,g}^{(1)} + e_h^2\,\Sigma\otimes\mathcal{C}_{k,q}^{(1)} + q^{\ell p}\otimes\mathcal{D}_{k,q}^{(1)}\right)\right\} \tag{28}$$

where e_h is the charge of the heavy quark (in units of the electron charge) and $q^{\ell p}$ denotes the proton quark distribution (for $f = 3$ light flavors, see for instance eq. (15) in section 2.2).

The first term in eq. (28) is the LO contribution from the photon-gluon fusion process $\gamma^* g \to h\bar{h}$. The last three terms correspond to the NLO subprocess $\gamma^* g \to h\bar{h}g$ and $\gamma^* q \to h\bar{h}q$.[11]

In eq. (28) we have introduced the shorthand notation

$$\frac{\alpha_s}{2\pi}\, f\otimes\mathcal{C} = \frac{\alpha_s(\mu_M^2)}{2\pi}\int_{ax}^{1}\frac{dz}{z}\; zf(z,\mu_M^2)\,\mathcal{C}(x/z,Q^2,\mu_M^2,m_h^2) \tag{29}$$

where $a = 1 + 4m_h^2/Q^2$ and μ_M^2 is the factorization (equals renormalization) scale which is usually set to $\mu_M^2 = Q^2$ or $\mu_M^2 = Q^2 + 4m_h^2$. The kinematic domain where the heavy quarks contribute is restricted by the requirement that the square of the γ^*p center of mass energy must be sufficient to produce the $h\bar{h}$ pair: $W^2 = M^2 + Q^2(1-x)/x \geq M^2 + 4m_h^2$ i.e, the lower integration limit $ax \leq 1$ in eq. (29). For the heavy quark coefficient functions $\mathcal{C}$ and $\mathcal{D}$ in eq. (28) we refer to [13].[12]

---

[9] Qcdnum however will continue to evolve q_c^+ as a non-singlet q^+ distribution below the charm threshold if you ask the program to do so.

[10] Notice that the proton distribution is continuous at the flavor thresholds although it is build up from distributions some of which are discontinuous. In fact, a powerful check on the consistency of a Qcdnum based analysis is provided by verifying that no discontinuities show up in distributions which should be continuous.

[11] In the LO and the first two NLO terms the virtual photon couples to the heavy quark, hence the factor e_h^2 in eq. (28). The last NLO term describes the process where the virtual photon couples to a light quark which subsequently branches into a $h\bar{h}$ pair via an intermediate gluon: hence the appearance of the charge weighted sum, $q^{\ell p}$, of light quark distributions.

[12] Some of these coefficient functions are given as interpolation tables since they are too complex to be cast into analytical form. All heavy quark coefficient functions in Qcdnum are taken from code provided by S. Riemersma.

A QCD analysis of charged lepton structure functions now might proceed as follows: the parton distributions, given at some scale $Q^2 = Q_0^2$, are evolved in Q^2 with $f = 3$ flavors. The light quark contribution (u, d, s) to the proton quark density is assembled from the evolved distributions and the structure functions $F_{2,L}$ are calculated from eq. (20) (with $f = 3$ flavors). To these are added the heavy flavor contributions $F_{2,L}^c$ and $F_{2,L}^b$ as calculated from eq. (28) above. The resulting structure functions are then compared to data. Notice that such heavy flavor corrections are process dependent: the method described above applies to charged lepton deep inelastic scattering only.

Finally we remark that Qcdnum will calculate F_k^h from *any* quark distribution (e.g. singlet, non-singlet) but that the result only makes sense when obtained for the proton, deuteron or neutron distributions as defined in section 2.3, for $f = 3$ flavors.

3 Numerical method

3.1 Overview

The Qcdnum program calculates the Q^2 evolution of parton momentum densities (in LO or in NLO in the $\overline{\mathrm{MS}}$ scheme) on a user defined grid in x and Q^2. The value of these parton densities at fixed $Q^2 = Q_0^2$ has to be specified for each grid point in x. This implies that Qcdnum does not use, nor cares about, parameterizations describing the x dependence of the input distributions.

The calculation of the logarithmic slopes in Q^2 is based on the computation of convolution integrals, see eqs. (6) and (8). These are calculated with the assumption that the parton densities can be linearly interpolated (at all Q^2) from one x-grid point to the next. With this assumption, the convolution integrals can be evaluated as weighted sums. The weights, which are essentially integrals of the QCD splitting functions, are calculated to high precision at program initialization.

From the value of a given input parton distribution and the calculated slopes at Q_0^2, the distribution can be evaluated at the next grid point $Q_1^2 > Q_0^2$ (or $Q_1^2 < Q_0^2$). This distribution then serves as an input to calculate the slopes at Q_1^2 and so on. In this way the evolution can be continued over the whole x-Q^2 grid. In Qcdnum, the evolution is based on a second order spline interpolation in $\ln Q^2$. In earlier versions of the program [1] this quadratic interpolation was achieved iteratively, starting from a linear evolution in $\ln Q^2$ from one Q^2 grid point to the next. In the present program this is achieved without iteration by solving linear equations at each evolution step.

The number of grid points in x and Q^2 has to be chosen with some care. A larger number of grid points improves the numerical accuracy (the interpolations underlying the evolution algorithm become better approximations) but cannot be taken too large because the CPU time increases quadratically (linearly) with the number of points in x (Q^2). In practice, good balance between accuracy and speed has been achieved on a 100×40 x–Q^2 grid covering a large kinematic range down to $x = 10^{-5}$ and $4 < Q^2 < 10000$ GeV^2, see section 5.

It is important to notice that the Qcdnum program evolves parton *momentum* densities in contrast to the formalism presented in the previous sections which is given in terms of parton *number* densities. The user has thus to supply starting values for $xg(x)$, $x\Sigma(x), \ldots$ instead of $g(x)$, $\Sigma(x)$ etc.

3.2 Evaluation of convolution integrals

When written in terms of the parton momentum density $H(x) = xh(x)$ the convolution integrals in eqs. (6), (8) and (18)–(21) take the following form:

$$I(x_0) = \int_{x_0}^{1} \frac{x_0 dz}{z^2} P\left(\frac{x_0}{z}\right) H(z) \tag{30}$$

where $P(x)$ is a QCD splitting or coefficient function. The distribution $H(x)$ is sampled on a grid

$$x_0 < x_1 < \cdots < x_n < x_{n+1} \equiv 1 \text{ with } \; H(x_{n+1}) = H(1) \equiv 0.$$

This distribution is linearly interpolated between grid points:

$$\begin{aligned} H(x_i < x < x_{i+1}) &= (1-t)H(x_i) + tH(x_{i+1}) \\ t &= (x - x_i)/(x_{i+1} - x_i). \end{aligned} \tag{31}$$

Inserting eq. (31) into eq. (30) the convolution integral can be written as a weighted sum:

$$I(x_0) = \sum_{i=0}^{n} w(x_i, x_0) H(x_i) \tag{32}$$

with

$$\begin{aligned} w(x_0, x_0) &= S_1(s_1, s_0) \\ w(x_i, x_0) &= S_1(s_{i+1}, s_i) - S_2(s_i, s_{i-1}) \end{aligned} \tag{33}$$

where $s_i \equiv x_0/x_i$ and

$$\begin{aligned} S_1(u, v) &= \frac{v}{v-u} \int_u^v (z-u) P(z) \frac{dz}{z} \\ S_2(u, v) &= \frac{u}{v-u} \int_u^v (z-v) P(z) \frac{dz}{z}. \end{aligned} \tag{34}$$

In Qcdnum the weights are calculated at initialization from Gauss integration of eq. (34) for all LO and NLO QCD splitting and light quark coefficient functions and are stored in 2-dimensional tables. In addition tables based on analytical expressions for the LO weights are available: they may serve as a check on the accuracy of the numerical integration.

Inserting eq. (32) into eqs. (6) and (8) the evolution equations for the parton momentum densities can be written as weighted sums, e.g. for the the singlet quark momentum density we have:

$$\begin{aligned} \frac{dx_0\Sigma(x_0)}{d\ln Q^2} = \frac{\alpha_s}{2\pi} \sum_{i=0}^{n} & \left[\; w_{FF}^{(0)}(x_i, x_0) + \frac{\alpha_s}{2\pi} w_{FF}^{(1)}(x_i, x_0) \; \right] x_i\Sigma(x_i) \; + \\ & \left[\; w_{FG}^{(0)}(x_i, x_0) + \frac{\alpha_s}{2\pi} w_{FG}^{(1)}(x_i, x_0) \; \right] x_i g(x_i) \end{aligned} \tag{35}$$

Likewise the expressions for the structure functions F_2, F_L and xF_3 in NLO (see eqs. (18)–(21)) can be written as weighted sums.

The weight tables for the heavy quark coefficient functions depend on Q^2 which makes them 3-dimensional. Their size is however much reduced by calculating the heavy quark structure functions on an equidistant logarithmic grid in x (this grid is automatically maintained by Qcdnum). For such a grid the weights depend only on the difference $(x_i - x_0)$ thus saving a factor $\sim n_x/2$ in storage.[13]

In this way, fast calculation of the logarithmic slopes (and of structure functions) is achieved entirely based on lookup tables which can be pre-calculated with high (and controlled) accuracy. The x-grid must of course be dense enough so that linear interpolation in x, eq. (31), is a good approximation. Since parton densities are smoothly varying functions of x this can be achieved with relatively few grid points (typically $n_x \approx 100$).

3.3 Evolution in $\mathbf{Q^2}$

In this section we describe the calculation of the Q^2 evolution of *non-singlet* quark momentum densities $H(x,Q^2) = xq^{\pm}(x,Q^2)$. Extension of the method to singlet/gluon evolution is straight forward. For convenience we introduce the variable $t = \ln Q^2$ and write for a given grid point $\{x_i, t_j\}$ the evolution equation for H as, cf. eq. (35):

$$H'(x_i,t_j) \;=\; \frac{\alpha_s(t_j)}{2\pi} \sum_{k=i}^{n} \left[\; w_{\pm}^{(0)}(x_k,x_i) + \frac{\alpha_s(t_j)}{2\pi}\, w_{\pm}^{(1)}(x_k,x_i) \;\right]\; H(x_k,t_j) \tag{36}$$

with $H' = \partial H/\partial t$. This slope cannot be calculated directly because the first term in the sum on the r.h.s. of eq. (36) contains $H(x_i,t_j)$ which is *a priori* unknown. Separating this term from the sum and denoting the remaining sum (which runs from $k = i+1$ to n) by S we write eq. (36) in shorthand notation as:

$$H'_j = wH_j + S \tag{37}$$

Next, we assume that the following condition is fulfilled:

> Condition (i): In previous steps of the evolution $H(x_k,t_j)$ has been calculated for all values of $x_k > x_i$, i.e. for $k > i$.

With this assumption the sum S can be evaluated and, as a consequence, eq. (37) becomes a linear equation with two unknowns H'_j and H_j.

A second linear equation is obtained as follows: the density $H(x_i,t)$ is quadratically interpolated between the grid points t_{j-1} and t_j :

$$H(x_i, t_{j-1} < t < t_j) = at^2 + bt + c.$$

With this interpolation $H(x_i,t_j)$ is related to $H(x_i,t_{j-1})$, $H'(x_i,t_j)$ and $H'(x_i,t_{j-1})$ through

$$H(x_i,t_j) = H(x_i,t_{j-1}) + \frac{1}{2}\left[H'(x_i,t_j) + H'(x_i,t_{j-1}\right]\,\Delta t_j \tag{38}$$

with $\Delta t_j = t_j - t_{j-1}$. If we assume that the following condition is satisfied:

[13]The weight tables for the splitting and light quark coefficient functions can of course also be reduced in size but we prefer to give complete freedom to choose the grid density at both low and high x by allowing a non-equidistant grid in $\ln x$. An equidistant logarithmic x–grid with larger spacing at high x does not much affect the heavy quark structure functions since these are strongly suppressed by threshold effects at large x.

Condition (ii): In previous steps of the evolution both $H(x_i, t_{j-1})$ and $H'(x_i, t_{j-1})$ have been calculated for $t_{j-1} < t_j$.

then also eq. (38) is a linear equation with two unknowns H_j and H'_j; in shorthand notation:

$$H_j = H_{j-1} + \frac{1}{2}(H'_{j-1} + H'_j)\,\Delta_j. \tag{39}$$

Solving eqs. (37) and (39) for H_j we get:

$$H_j = \frac{2H_{j-1} + (H'_{j-1} + S)\,\Delta_j}{2 - w\,\Delta_j}. \tag{40}$$

Because of conditions (i) and (ii) above all quantities on the r.h.s. of eq. (40) are known. The value of H'_j is found by substituting eq. (40) back into eq. (37). In this way both $H(x_i, t_j)$ and $H'(x_i, t_j)$ are calculated for a given grid point $\{x_i, t_j\}$ by solving two linear equations with two unknowns.

Next, we show that it is possible to define, starting from a suitable grid point, a progression in x and t such that the conditions (i) and (ii) mentioned above are satisfied at each step of the evolution. First, condition (i) is satisfied for all t at the highest grid point x_n since we have defined $x_{n+1} = 1$ and $H(x_{n+1}, t) = H(1, t) = 0$ (see section 3.2). Second, at the starting value of the evolution t_0, $H(x, t_0)$ is known (input by the user) and $H'(x, t_0)$ can be calculated directly from the QCD evolution eq. (36). At the next grid point $t_1 > t_0$ condition (ii) is thus satisfied for all values of x. It follows that both conditions (i) and (ii) are met at the grid point $\{x_n, t_1\}$ which serves as the starting point of the evolution. It is easy to see that one can subsequently progress to larger t at fixed values of x (or to smaller x at fixed values of t) and continue the evolution over the whole x–t grid. Here we have tacitly assumed upward evolution in $t > t_0$; the method applies to downward evolution as well provided t_{j-1} is replaced by t_{j+1} in the above.

It is straight forward to extend the evolution algorithm to the singlet/gluon case by solving four linear equations with four unknowns at each evolution step.

Finally, we remark that the method outlined in this and the previous section yields parton distributions which are represented by second order splines in t at fixed values of x and by first order splines in x at fixed values of t. The evolution is fully numerical in the sense that no use is made of specific parametric forms of the parton densities.

4 Qcdnum user guide

The Qcdnum program consists of a set subroutines which perform the QCD evolution of parton distributions (and α_s) and which calculate structure functions from the evolved distributions. Interface routines allow to interact with the program which, from a user point of view, can be regarded as a black box. A QCD analysis is performed by an application program which in addition to Qcdnum might use Pdflib (for parton distribution input) or Minuit (for QCD fits).

4.1 The Qcdnum program

The Qcdnum code is available as a patchy `.car` file. The patches to be selected in the patchy run are:

```
+USE,DOUBLE.                    Select double precision version
+USE,QCDCOM,T=EXE.              Qcdnum common blocks
+USE,QCDNUM,T=EXE.              Qcdnum code (light quarks)
+USE,QCUTIL,T=EXE.              Utility routines
+USE,QHEAVY,T=EXE.              Qcdnum code (heavy quarks)
```

The switch `+use,double` selects the double precision version of Qcdnum. To obtain the single precision version you simply omit the line `+use,double` or comment it out: `c+use,double`. It is recommended to always compile the program in double precision except on machines with 60 bit arithmetic like the CRAY. When the double precision version is used:

- floating point variables in Qcdnum subroutine calls and Qcdnum floating point functions should be declared `double precision` in the calling routine e.g:

  ```
  double precision val, x, q2, QPDFXQ
       ..
  val = QPDFXQ ( 'name', x, q2, iflag )
       ..
  ```

  The easiest is of course to simply declare all floating point variables as double precision in the calling routine:

  ```
  ----------------
  subroutine MYSUB
  ----------------
  implicit double precision (A-H,O-Z)
       ..
  ```

- Actual values passed to subroutines *must* be written in double precision format e.g. `1.3D0` instead of `1.3` in for instance:

  ```
  call QNRSET('MCSTF',1.3D0)
  ```

In the remainder of this writeup we assume that Qcdnum is run in double precision mode.

The patchy directive `+use,qheavy,t=exe` makes the heavy quark code available. This directive may be omitted (or commented out) producing a smaller executable but then the heavy quark structure functions cannot be calculated anymore.[14]

The Qcdnum code is written in standard Fortran77 and uses only a few CERN library routines namely `DGAUSS` (D103), `DDILOG` (C304), `FLPSOR` (M103) and, for character string manipulation, `LENOCC` (M507) and `CHPACK` (M432). Qcdnum does not use a dynamic memory manager so that all common blocks are defined at compilation time. The size of these common blocks is governed by the two parameters `mxx` and `mq2`: `mxx-1` and `mq2-1` are the maximum number of grid points in x and Q^2 the user can define. This can be set directly in the source code or through the patchy directive (see table 1) e.g:

[14] An attempt to do so will cause Qcdnum to abend with an error message.

```
+-----------------------------------------------------------------+
|                                                                 |
|  #!/bin/csh -fx                                                 |
|  #                                                              |
|                                                                 |
|  /cern/pro/bin/ypatchy - qcdprog tty qcdprog.list .go << //     |
|                                                                 |
|  +OPTION,MAPASM,UREF.                                           |
|                                                                 |
|  +USE,DOUBLE.                                                   |
|                                                                 |
|  +USE,P=QUCOMM,QUPROG,T=EXE................user program          |
|  +USE,QCDCOM,QCDNUM,QCUTIL,T=EXE...........Qcdnum code          |
|  +USE,QHEAVY,T=EXE.........................Heavy quark code     |
|                                                                 |
|  +REPL,QCDCOM,QCDCOM,6-7...................Max # grid points    |
|        PARAMETER ( MXX = 100 )                                  |
|        PARAMETER ( MQ2 =  40 )                                  |
|                                                                 |
|  +EXE,P=CRA*                                                    |
|                                                                 |
|  +PAM, 11, T=ATTACH, T=CARDS.       qcduser.car                 |
|  +PAM, 12, T=ATTACH, T=CARDS.       qcdnum16.car                |
|                                                                 |
|  +QUIT.                                                         |
|                                                                 |
|  //                                                             |
|                                                                 |
|  f77 $f77ldflags -O qcdprog.f -o qcdprog -L/cern/pro/lib \      |
|                               -lgenlib -lpacklib -lkernlib      |
|                                                                 |
|  qcdprog > qcdprog.log << EOF                                   |
|                                                                 |
|  (User datacards, e.g. MINUIT, if any)                          |
|                                                                 |
|  EOF                                                            |
|                                                                 |
+-----------------------------------------------------------------+
```

Table 1: Example of a unix script running Qcdnum

```
+REPL,QCDCOM,QCDCOM,6-7.
      PARAMETER ( MXX = 150 )
      PARAMETER ( MQ2 =  60 )
```

See section 5 for further details on program size, accuracy and speed.

A unix script which performs a patchy run, compiles the resulting fortran code, makes the executable and runs it might look like that shown in table 1. In this example the Qcdnum code is stored in the file `qcdnum16.car` and user code in `qcduser.car`. The user code contains two patches: `qucomm` and `quprog`. An example of such code is given in section 4.3.

4.2 User program

An analysis program based on Qcdnum should contain the following steps:

1. Initialization.
2. Definition of the x–Q^2 grid.
3. Calculation of weights. The weight calculations depend on the grid definition which therefore cannot be changed afterwards. By default, weight tables are computed for the LO and NLO splitting functions as well as for the NLO F_2, F_L and xF_3 coefficient functions. Tables which depend on the number of flavors are generated for $f = 3$, 4 and 5. Optionally weight tables are calculated for the heavy quark coefficient functions.
4. Definition (booking) of the parton distributions. Internally Qcdnum reserves space for the gluon and the quark singlet momentum density and for up to nine user defined non-singlet densities. Furthermore up to 20 linear combinations of the quark densities can be defined.
5. Setting of options/parameters e.g:
   - input of the flavor thresholds.
   - input value for α_s.
   - LO or NLO calculations.
   - etc.
6. Input of the parton densities for each grid point in x at some fixed value of $Q^2 = Q_0^2$ followed by the QCD evolution of these distributions. Routines are provided for the singlet/gluon evolution and for the evolution of non-singlet quark densities xq^+, $x\Delta$ and xq^-.
7. Evaluation of structure functions and, in fitting applications, the calculation of the χ^2. Qcdnum provides a routine for the calculation of F_2, F_L and xF_3. The contribution to F_2 and F_L from charm and bottom can also be calculated.

In the following subsections we give a detailed description of the steps outlined above. For quick reference a list of Qcdnum subroutines is given in table 2. In the remainder of this write-up we use the following notation:

Subroutine or function	Description
`QNINIT`	Initialization
`QNVERS ( 'vers' Ldoubl, nxmax, nqmax )`	Get info about current version
`QNTIME ( 'option' )`	Start/stop CPU timelog
`QNiSET ( 'var', ival ), i = L, I, R`	Set Qcdnum variable
`QNiGET ( 'var', ival ), i = L, I, R`	Get Qcdnum variable
`GRXDEF ( nx, xmi )`	Define the x grid
`GRQDEF ( nq, qmi, qma )`	Define the Q^2 grid
`GRXINP ( xarray, nx )`	Add x grid points
`GRQINP ( qarray, nq )`	Add Q^2 grid points
`GRXNUL, GRQNUL`	Clear the x, Q^2 grid
`GRMXMQ ( nxmax, nqmax )`	Give maximum number of grid points
`GRGIVE ( nx, xmi, xma, nq, qmi, qma )`	Get definition of the current grid
`GRXOUT ( xgrid ), GRQOUT ( qgrid )`	Copy x, Q^2 grid to local array
`xx = XFROMIX ( ix ), q2 = QFROMIQ ( iq )`	Get x, Q^2 for given grid index
`ix = IXFROMX ( xx ), iq = IQFROMQ ( q2 )`	Get grid index for given x, Q^2
`ix = IXNEARX ( xx ), iq = IQNEARQ ( q2 )`	Get grid index for given x, Q^2
`GRCUTS ( xmi, qmi, qma, roots )`	Set cuts
`istat = IFAILIJ ( ix, iq )`	`ix`, `iq` fails/passes cuts
`istat = IFAILXQ ( x, q2 )`	x, Q^2 fails/passes cuts
`QNFILW( 0, 0 )`	Calculate weight tables
`QNDUMP( lun )`	Write weight tables to disk
`QNREAD( lun, istop, ierr )`	Read weight tables from disk
`QNBOOK ( id, 'name' )`	Book non-singlet distribution
`QNLINC ( id, 'name', nf, factors )`	Define linear combination
`id = IPDFID ( 'name' )`	Get `id` of quark distribution
`QTHRES ( t34, t45 )`	Set flavor thresholds
`nf = NFLGET ( iq )`	Get number of flavors
`as = QALFAS ( q2 ,qlam, nf, ierr)`	Get $\alpha_s(Q^2)$
`QNPSET ( 'name', ix, iq, value )`	Set value of parton density
`QADDSI ( 'name', iq, factor )`	Add `factor` times singlet
`QNPNUL ( 'name' )`	Set parton distribution to zero
`EVOLSG ( iq0, iqmin, iqmax )`	Singlet/gluon evolution
`EVOLNP ( 'name', iq0, iqmin, iqmax )`	Non-singlet evolution ($x\Delta$)
`EVPLUS ( 'name', iq0, iqmin, iqmax )`	Non-singlet evolution (xq^+)
`EVOLNM ( 'name', iq0, iqmin, iqmax )`	Non-singlet evolution (xq^-)
`val = QPDFIJ ( 'name', ix, iq, ifl )`	Get value of parton density
`val = QPDFXQ ( 'name', x, q2, ifl )`	Get value of parton density
`val = QSTFIJ ( 'opt', 'name', ix, iq, ifl )`	Calculate F_2, F_L or xF_3
`val = QSTFXQ ( 'opt', 'name', x, q2, ifl )`	Calculate F_2, F_L or xF_3
`QFMARK ( x, q2 )`	Mark for fast F_i calculation
`QFMNUL`	Clear marks
`STFAST ( 'opt', 'name' )`	Fast F_i calculation
`STFCLR`	Clear memory allocation
`QPRINT ( lun, 'opt' )`	Print Qcdnum info

Table 2: Subroutine and function calls in Qcdnum

- A variable with a name starting with the letter `L` is of type logical.
- A variable with a name starting with the letter `I`–`N` (excluding `L`) is of type integer.
- We denote character variables by giving the name in quotes, e.g. `'name'`. Significant characters of input variables are given in capitals as in e.g:

```
call QPRINT(6,'Timelog')
```

  Qcdnum is case insensitive so that `'T'`, `'t'`, `'time'` and `'Timelog'` are all valid inputs.
- All other variables are either of type real or of type double precision depending on the Qcdnum version which is used.

4.3 Example program

In this section we give an example of a simple application program. The program evolves the singlet quark and gluon distribution given at a starting value of $Q_0^2 = 7$ GeV^2 in NLO (NLO is Qcdnum default) on a 80×60 x–Q^2 grid. The grid covers the kinematic range $4 \times 10^{-4} < x < 1$ and $4 < Q^2 < 5000$ GeV^2. The flavor thresholds are set to Q_c^2 (Q_b^2) $= 4$ (25) GeV^2. The code uses the predefined names `'singlet'` (`'gluon'`) to access the singlet quark (gluon) distribution. The quark and gluon distributions at Q_0^2 are given by the user defined functions `quarks(x)` and `gluons(x)`. The singlet F_2 structure function is calculated from the evolved distributions.

```
+patch,QUCOMM.

           (user defined sequences, if any)

+patch,QUPROG.

*      --------------
       program QUPROG
*      --------------

       implicit double precision (A-H,O-Z)

*-1-   Initialization
       call QNINIT
       call QNTIME('start')

*-2-   Setup grid in x and Q2
       q0 = 7.
       call GRXDEF(80,4.D-4)
       call GRQDEF(59,4.D0,5000.D0)
       call GRQINP(q0,1)
       call GRGIVE(nx,xmi,xma,nq,qmi,qma)
```

```
*-3-  Create weight tables
      call QNFILW(0,0)

*-4-  Set charm (bottom) threshold at Q2 = 4 (25) GeV2
      call QTHRES(4.D0,25.D0)

*-5-  Set input value for alphas
      Mz = 91.2
      call QNRSET('alfas',0.118D0)
      call QNRSET('alfQ0',Mz*Mz)

*-6-  Input singlet and gluon at Q2 = 7 GeV2
*     quarks(x) and gluons(x) are user defined functions
      iq0 = IQFROMQ(q0)
      do ix = 1,nx
        x = XFROMIX(ix)
        call QNPSET('singlet',ix,iq0,quarks(x))
        call QNPSET('gluon'  ,ix,iq0,gluons(x))
      enddo

*-7-  Singlet/gluon evolution
      call EVOLSG(iq0,1,nq)

*-8-  Get singlet, gluon, F2 etc. at any x and Q2
      x    = 0.005
      q2   = 20.
      qval = QPDFXQ(     'singlet',x,q2,iflag)
      gval = QPDFXQ(     'gluon'  ,x,q2,iflag)
      fval = QSTFXQ('F2','singlet',x,q2,iflag)

*-9-  Some printout
      call QPRINT(6,'all')

      end
```

The user defined function `gluons(x)` for instance, might look as follows:

```
*     -----------------------------------
      double precision function GLUONS(x)
*     -----------------------------------

      implicit double precision (A-H,O-Z)

      dimension a(3)
      data a /1.01,-0.35,5.22/

      GLUONS = a(1) * x**a(2) * (1.-x)**a(3)
```

```
      return
      end
```

4.4 Initialization

Qcdnum is initialized by

```
      call QNINIT
```

This routine, which *must* be called before anything else, initializes a large amount of constants and sets reasonable defaults.

```
      call QNVERS ( 'version', Ldouble, nxmax, nqmax )
```

returns the version number in the `character*8` variable `'version'` and sets the logical variable `Ldouble` to `true` if the current program version is in double precision and to `false` if it is in single precision. The variables `nxmax` and `nqmax` are set to the maximum number of grid points in x and Q^2 the user can define.[15]

A timelog of Qcdnum subroutine calls is started by

```
      call QNTIME ( 'Start' )
```

At any moment you can interrupt the logging by a call to `QNTIME('Hold')` and resume it again by calling `QNTIME('Continue')`.

4.5 Qcdnum variables and options

A list of Qcdnum variables/options, is shown in table 3. The meaning of these variables is as follows:

- `W1ANA`, ..., `WTFLB`: these eight logical options steer the weight calculations as described in section 4.7.
- `LIMCK` if set to `true` one cannot access parton distributions or structure functions outside the grid boundaries or cuts. See section 4.12.
- `CLOWQ` if set to `true` one can access heavy quark structure functions only for $Q^2 > 1.5$ GeV^2. If set to `false` no such check is made. See section 4.12.
- `ORDER` = 1 (2) selects LO (NLO) calculations. Its value can be set at any time. Default: NLO.
- `SCAX0` and `SCAQ0` are related to the logarithmic/linear scale of the x and Q^2 grid, see section 4.6.

[15] See section 4.1 on how to change this.

var	typ	deflt	value	description
W1ANA	L	T	T	Analytical LO weight calculation
W1NUM	L	F	F	Numerical LO weight calculation
W2NUM	L	T	T	Numerical NLO weight calculation
W2STF	L	T	T	Structure function NLO weights
WTF2C	L	F	T	F2_charm weight calculation
WTF2B	L	F	T	F2_bottom weight calculation
WTFLC	L	F	T	FL_charm weight calculation
WTFLB	L	F	T	FL_bottom weight calculation
LIMCK	L	T	T	Check x, Q2 limits and cuts
CLOWQ	L	T	T	Heavy F2,FL only for Q2 > 1.5 GeV2
ORDER	I	2	2	LO (1) or NLO (2) calculations
SCAX0	R	0.20	0.20000E+00	x-grid scale from log --> linear
SCAQ0	R	+inf	0.10000E+11	Q2-grid scale from log --> linear
MCSTF	R	1.5	0.15000E+01	C mass for F2c, FLc (GeV)
MBSTF	R	5.0	0.50000E+01	B mass for F2b, FLb (GeV)
MCALF	R	1.5	0.15000E+01	C mass for alpha_s evolution (GeV)
MBALF	R	5.0	0.50000E+01	B mass for alpha_s evolution (GeV)
MTALF	R	188.	0.18800E+03	T mass for alpha_s evolution (GeV)
ALFAS	R	0.180	0.13000E+00	Value of alpha_s
ALFQ0	R	50.	0.83152E+04	Q2 where alpha_s is given (GeV2)
AAAR2	R	1.0	0.10000E+01	R2 = A*M2 + B (ren. scale)
BBBR2	R	0.0	0.00000E+00	R2 = A*M2 + B (ren. scale)
AAM2L	R	1.0	0.10000E+01	M2 = A*Q2 + B (light fact. scale)
BBM2L	R	0.0	0.00000E+00	M2 = A*Q2 + B (light fact. scale)
AAM2H	R	1.0	0.10000E+01	M2 = A*Q2 + B (heavy fact. scale)
BBM2H	R	0.0	0.00000E+00	M2 = A*Q2 + B (heavy fact. scale)
TCHRM	R	-inf	-0.10000E+11	Charm threshold
TBOTT	R	+inf	0.10000E+11	Bottom threshold
XMINC	R	0.0	-0.10000E+01	Xmin cut (.le.0 = no cut)
QMINC	R	0.0	-0.10000E+01	Qmin cut (.le.0 = no cut)
QMAXC	R	0.0	-0.10000E+01	Qmax cut (.le.0 = no cut)
ROOTS	R	0.0	-0.10000E+01	Roots cut (.le.0 = no cut)
QMINA	R	0.0	0.47568E+00	Lowest Q2 gridpoint above Lambda2

Table 3: Default settings in Qcdnum

- MCSTF and MBSTF are the charm and bottom mass (in GeV) used in the heavy quark structure function calculations, see section 4.12.
- MCALF, MBALF and MTALF are the charm, bottom and top mass (GeV) which determine the flavor thresholds in the α_s evolution.
- ALFAS and ALFQ0 are the value of α_s and the Q^2 where it is given. These values may be changed at any time.
- AAAR2 and BBBR2 define the renormalization scale $\mu_R^2 = a\mu_M^2 + b$, see section 4.13.
- The mass factorization scale, $\mu_M^2 = aQ^2 + b$, can be defined for the light (heavy) quark structure functions by setting the variables AAM2L, BBM2L (AAM2H, BBM2H), see section 4.13.

Logical, integer and floating point variables can be set by calls to QNLSET, QNISET and QNRSET respectively, e.g:

```
call QNLSET ( 'W1NUM', .true. )
call QNISET ( 'ORDER', iorder )
call QNRSET ( 'ALFAS', 0.187D0 )
```

The actual value of a Qcdnum parameter/option can be retrieved by similar calls to QNLGET, QNIGET and QNRGET. The list as given in table 3 can be written to logical unit number lun by a call to QPRINT(lun,'Parameters').[16]

Also given in table 3 are the thresholds Q_c^2 (TCHRM), Q_b^2 (TBOTT) and four cut parameters XMINC, ..., ROOTS. These cannot be set by QNRSET but by special routines described in section 4.10. The cut QMINA is set automatically by Qcdnum to the lowest Q^2 gridpoint above the current value of Λ^2. You can retrieve the values of the cuts with calls to QNRGET.

4.6 Definition of the grid

There are two ways in which the x–Q^2 grid can be specified:

- Specify the kinematic range and the number of grid points to be generated:

  ```
  call GRXDEF ( nx, xmin )
  call GRQDEF ( nq, qmin, qmax )
  ```

  GRXDEF clears the x–grid (if any) and generates nx grid points in the range $x_{min} \leq x < 1$. Qcdnum sets the (nx+1)th grid point to one. The scale is logarithmic for $x < 0.20$ and linear for $x > 0.20$.[17] The routine GRQDEF clears the Q^2–grid and generates a logarithmic grid of nq points in the range $Q^2_{min} \leq Q^2 \leq Q^2_{max}$.

- Pass a list of n grid points to be *added* to the grid:

[16] A logical unit number other than 6 should be opened by the user before a call to QPRINT.

[17] The default, $x_0 = 0.20$, can be changed by: call QNRSET ('SCAX0', x0). There is a similar parameter for the Q^2–grid (SCAQ0) which at initialization is set to a very high value.

```
call GRXINP ( xarray, nx )
call GRQINP ( qarray, nq )
```

where `xarray` (`qarray`) are arrays containing `nx` (`nq`) points in x (Q^2) which may be given in arbitrary order. One may, or may not specify the point $x = 1$, Qcdnum will always generate it.

As mentioned in section 3.2 the heavy quark structure functions are calculated on an equidistant logarithmic grid in x. This grid is automatically maintained by Qcdnum with the same range (`xmin,1`) and the same number of grid points `nx` as the grid defined by the calls to `GRXDEF` and `GRXINP`.

A call to `GRXNUL` (`GRQNUL`) sets the x (Q^2) grid to zero.

There are several ways to access the grid:

- `GRMXMQ(nxmax,nqmax)` returns the maximum number of grid points the user can define.
- `GRGIVE(nx,xmi,xma,nq,qmi,qma)` returns the number of grid points and the limits of the current grid.
- `GRXOUT(xgrid)` and `GRQOUT(qgrid)` return the list of current grid points in the arrays `xgrid` (`qgrid`) which should be dimensioned in the calling routine to at least `nx` (`nq`).
- The value of x corresponding to a given grid index is returned by the function `x = XFROMIX(ix)` where `x` is set to zero if `ix` is outside the grid boundaries. Likewise the Q^2 value is returned by `q2 = QFROMIQ(iq)`
- The inverse function `ix = IXFROMX(x)` returns the grid index `ix` for given x. If x does not coincide with a grid point, `ix` is negative and `-ix` corresponds to the closest grid point which lies below x. If x is outside the grid boundary `ix` is set to zero. Thus `ABS(IXFROMX(x))` can be thought of as a function which returns for given x the 'bin number' (i) with the grid point x_i being the lower edge of the bin. The corresponding function for the Q^2 grid is `iq = IQFROMQ(q2)`.
- The function `ix = IXNEARX(x)` is similar to `IXFROMX` but here the grid index `ix` corresponds to the grid point closest to x (instead of that below x). Again, if x coincides with a grid point `ix` is positive otherwise negative. `ABS(IXNEARX(x))` can be thought of as a function which returns for given x the 'bin number' (i) with x_i being the center of the bin. The corresponding function for Q^2 is `iq = IQNEARQ(q2)`.

The current grids (including the logarithmic heavy quark x–grid) can be written to logical unit number `lun` by a call to `QPRINT(lun,'Xqgrid')`.

Finally, when defining the x–Q^2 grid one should bear in mind that (see section 3):

- The grid should be dense enough so that the linear (quadratic) interpolation in x ($\ln Q^2$) used by Qcdnum is a sufficiently good approximation.
- The CPU time increases quadratically (linearly) with the number of grid points in x (Q^2).
- The starting scale of the evolutions, Q_0^2, must be included in the grid: see section 4.11.

- It is convenient, but not strictly necessary, to also put the flavor thresholds, Q_c^2 and Q_b^2, into the grid.

A good initial choice is about 100 grid points in x and about 40 in Q^2. It may be worthwhile to spent some effort in optimizing the grid for QCD fit applications where the program is likely to be run many times on the same dataset. A modest decrease in the number of x grid points by, say, 30% will increase the program speed by a factor of two. For further details, see section 5.

4.7 Weight calculation

Weight tables are calculated by a call to[18]

```
call QNFILW ( 0, 0 )
```

By default all weights corresponding to the LO and NLO splitting functions and those for the NLO light quark coefficient functions are evaluated for f = 3, 4 and 5 flavors. The weight calculations depend on the x grid which thus must have been defined before the call to `QNFILW` and cannot be changed afterwards.[19]

The logical variables listed in table 4 define which tables are calculated in `QNFILW`. These variables can be set by the appropriate calls to `QNLSET` e.g:

```
call QNLSET ( 'W1NUM', .TRUE. )
```

forces Qcdnum to calculate the LO weight tables numerically instead of analytically (default). If one is not interested in NLO calculations the generation of NLO weight tables may be switched off by setting `W2NUM` and `W2STF` to `false`.

[18]The two integer arguments `(0,0)` in `QNFILW` are irrelevant and kept only for reasons of backward compatibility.

[19]Doing so invalidates the current weight tables. An attempt to use them afterwards in an evolution or structure function routine will cause a fatal error.

```
+-------+---+-------+---------+-----------------------------------+
| var   |typ| deflt |  value  | description                       |
+-------+---+-------+---------+-----------------------------------+
| W1ANA | L |   T   |    T    | Analytical LO weight calculation  |
| W1NUM | L |   F   |    F    | Numerical  LO weight calculation  |
| W2NUM | L |   T   |    T    | Numerical NLO weight calculation  |
| W2STF | L |   T   |    T    | Structure function NLO weights    |
| WTF2C | L |   F   |    F    | F2_charm  weight calculation      |
| WTF2B | L |   F   |    F    | F2_bottom weight calculation      |
| WTFLC | L |   F   |    F    | FL_charm  weight calculation      |
| WTFLB | L |   F   |    F    | FL_bottom weight calculation      |
+-------+---+-------+---------+-----------------------------------+
```

Table 4: Logical variables which steer the weight calculations

The CPU time needed for the calculation of the light quark weights is modest: about 60 seconds for ~ 100 x grid points but will increase quadratically with the number of grid points.

The logical variables `WTF2C`, `WTF2B`, `WTFLC` and `WTFLB` enable the calculation of weight tables needed for the heavy flavor structure functions. These tables depend on the definition of both the x and the Q^2 grid which should not be changed after the call to `QNFILW`. They also depend on the charm or bottom mass which should not be changed either. The calculation of the heavy quark weight tables typically takes a few CPU minutes only.

4.8 Disk dump/read of weight tables

To avoid the weight initialization for every Qcdnum run there is the possibility to write them to disk (`QNDUMP`) and read them back again (`QNREAD`) in a next pass of your QCD analysis. The rules are as follows:

- Unformatted read/write: this implies that weight files cannot be exchanged across different machines.
- Only those tables which are available at the point where `QNDUMP` is called are written to disk.
- For each table the full common block is dumped. This implies that the size of the output file does not depend on how many x–Q^2 grid points are actually defined but on the size of the common block itself. This depends on the parameters `mxx` and `mq2` which are also written to the file. Weight files cannot be exchanged across Qcdnum versions which have different values of `mxx` and/or `mq2` (see also section 4.1).
- All weight tables depend on the current x grid definition whereas the heavy quark tables depend in addition on the Q^2 grid: these grid definitions are also written to disk.
- The heavy quark tables depend on the charm and/or bottom mass which are dumped too.

The weight tables are written to disk by a call to `QNDUMP` e.g:

```
OPEN(unit=24,file=weights.file,form='unformatted',status='unknown')
CALL QNDUMP(24)
```

Files containing all weight tables (including heavy quarks) typically require about 5 Mbyte of storage.

To read the file back, use:

```
OPEN(unit=24,file=weights.file,form='unformatted',status='unknown')
CALL QNREAD(24, istop, ierr)
```

However this will not only read the weight tables but also overwrite the current definition of the x–grid (if any) and, in case the heavy quark tables are stored, that of the Q^2–grid together with the value of the charm and/or bottom mass.

To avoid unwanted side effects, `QNREAD` compares the grid definition (if any) and the quark masses in memory with those on the file and sets the error flag `ierr` as follows:

```
ierr  = 0 all OK.
ierr  = 1 xgrid exist in memory .ne. that on the file.
ierr  = 2 file contains heavy quark weight tables and
          qgrid exist in memory .ne. that on the file.
ierr  = 3 file contains charm weight tables and
          c mass in memory .ne. that on the file.
ierr  = 4 file contains bottom weight tables and
          b mass in memory .ne. that on the file.
```

The action taken by `QNREAD` depends on the input variable `istop`:

```
istop = 0 read the file whatever the value of ierr.
istop = 1 read only when ierr = 0, do nothing otherwise.
istop = 2 stop the program when ierr .ne. 0.
```

Thus, setting `istop = 1` provides the possibility to inspect the value of `ierr`, take the appropriate action, if any, and then call `QNREAD` again.

4.9 Definition of parton distributions

Qcdnum can hold up to 11 parton momentum densities in memory which internally are addressed by an identifier running from `id = 0` to `id = 10`. The user addresses these memory locations by a name. The identifier `id = 0` is reserved for the gluon and `id = 1` for the quark singlet distribution with the predefined names `'gluon'` and `'singlet'` respectively. The remaining 9 memory locations, which should contain *non-singlet* quark distributions,[20] can be associated to a user defined name by

```
call QNBOOK ( id, 'name' )
```

with $2 \leq$ `id` ≤ 10. The following rules apply:

- `'name'` can be given in upper, lower or mixed case (it is translated to upper case internally).
- `'name'` can be of any length but the first five characters must be unique (Qcdnum truncates `'name'` to five characters or adds trailing blanks if it is shorter). To avoid confusion please refrain from defining names with leading blanks.
- An identifier cannot be booked twice unless it is released previously by setting the name to `'free'`.

Thus, if one does not like the predefined name `'gluon'` for `id = 0`, it can be renamed by:

```
call QNBOOK ( 0, 'free' )
call QNBOOK ( 0, 'xglue' )
```

[20] They can only be evolved as non-singlets and can only enter as non-singlet contributions in the calculation of structure functions.

As an example we give below the calling sequence to book the non-singlet distributions $x\Delta_{ud}$, xq_s^+, xq_c^+, xq_b^+ and $xq_v = xu_v + xd_v$ (these distributions will be used in examples later on):

```
call QNBOOK ( 2, 'umind' )
call QNBOOK ( 3, 'splus' )
call QNBOOK ( 4, 'cplus' )
call QNBOOK ( 5, 'bplus' )
call QNBOOK ( 6, 'xqval' )
```

Up to 20 linear combinations of quark distributions with identifiers `id = 11` to `id = 30` can be defined by:

```
call QNLINC ( id, 'name', nf, factors )
```

where `factors` is a 10 dimensional array which contains the multiplication factor for the singlet distribution in `factors(1)` and that for the non-singlet distributions in `factors(2)` through `factors(10)`. Because such linear combinations often depend on the number of flavors, Qcdnum maintains for each identifier three factor arrays for $f = 3$, 4 or 5.[21]

For instance the proton quark distribution in charged lepton scattering for $f = 3$ flavors can be written as a linear combination of the singlet and non-singlet distributions:

$$q^{\ell p}_{f=3} = \frac{4}{18}\Sigma + \frac{1}{6}\Delta_{ud} - \frac{1}{6}q_s^+$$

Taking the non-singlet distributions from the booking list given above, the proton can be defined by the following code:

```
      double precision factors(10)
      data factors /10*0.D0/

*---  Define proton distribution for 3 flavors
      factors(1) =  4./18.
      factors(2) =  1./6.
      factors(3) = -1./6.
      call QNLINC ( 11, 'proton', 3, factors )
```

Likewise we have for four flavors:

$$q^{\ell p}_{f=4} = \frac{5}{18}\Sigma + \frac{1}{6}\Delta_{ud} - \frac{1}{6}q_s^+ + \frac{1}{6}q_c^+$$

```
      factors(1) =  5./18.
      factors(2) =  1./6.
      factors(3) = -1./6.
      factors(4) =  1./6.
      call QNLINC ( 11, 'proton', 4, factors )
```

[21] Notice that only the factor arrays are stored in memory and not the linear combinations themselves. Each time a linear combination is addressed Qcdnum evaluates it at any given x and Q^2. If $Q^2 < Q_c^2$ Qcdnum uses the factors defined for $f = 3$, if $Q_c^2 \leq Q^2 < Q_b^2$ those for $f = 4$ and if $Q_b^2 \leq Q^2$ those for $f = 5$ (see section 4.10 on how the flavor thresholds are set).

And for five flavors:

$$q^{\ell p}_{f=5} = \frac{11}{45}\Sigma + \frac{1}{6}\Delta_{ud} - \frac{1}{6}q_s^+ + \frac{1}{6}q_c^+ - \frac{1}{6}q_b^+$$

```
factors(1) = 11./45.
factors(2) =  1./6.
factors(3) = -1./6.
factors(4) =  1./6.
factors(5) = -1./6.
call QNLINC ( 11, 'proton', 5, factors )
```

The function

```
id = IPDFID ( 'name' )
```

returns the identifier `id` for a given `name` so that you do not have to remember all identifiers e.g:

```
*---  Define proton distribution for 3 flavors
      factors( IPDFID('singl') ) =  4./18.
      factors( IPDFID('umind') ) =  1./6.
      factors( IPDFID('splus') ) = -1./6.
      call QNLINC ( 11, 'proton', 3, factors )
```

It is guaranteed that $1 \leq$ `id` ≤ 10 because Qcdnum abends with a fatal error message if the `name` passed to `IPDFID` does not correspond to a memory resident quark distribution.

A list of all parton distributions can be written to logical unit number `lun` by a call to `QPRINT(lun,'Booklist')`. This gives for the example given above the list as shown in table 5. Notice from table 5 that Qcdnum has assigned 'trivial' multiplication factors to the memory resident distributions `gluon`,..., `xqval` and also that all names are truncated to five characters and converted to upper case.

```
+-------------+------------------------------------------------------------+
|             | W_ 1  W_ 2  W_ 3  W_ 4  W_ 5  W_ 6  W_ 7  W_ 8  W_ 9  W_10 |
| ID NAME  nf | SING  UMIN  SPLU  CPLU  BPLU  XQVA  FREE  FREE  FREE  FREE |
+-------------+------------------------------------------------------------+
|  0 GLUON    | 0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|  1 SINGL    | 1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|  2 UMIND    | 0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|  3 SPLUS    | 0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|  4 CPLUS    | 0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|  5 BPLUS    | 0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00  0.00 |
|  6 XQVAL    | 0.00  0.00  0.00  0.00  0.00  1.00  0.00  0.00  0.00  0.00 |
+-------------+------------------------------------------------------------+
| 11 PROTO   3 | 0.22  0.17 -0.17  0.00  0.00  0.00  0.00  0.00  0.00  0.00 |
|            4 | 0.28  0.17 -0.17  0.17  0.00  0.00  0.00  0.00  0.00  0.00 |
|            5 | 0.24  0.17 -0.17  0.17 -0.17  0.00  0.00  0.00  0.00  0.00 |
+-------------+------------------------------------------------------------+
```

Table 5: Example of a list of parton distributions

4.10 QCD evolution – parameters and options

Before evolving parton distributions and/or calculating structure functions a few parameters/options can be set as follows:

1. The order of all calculations in Qcdnum is governed by the value of `iorder` = 1 (LO) or 2 (NLO). The default is set to 2 (NLO) which can be changed at any time by

   ```
   call QNISET ( 'ORDER', 1 )
   ```

   and *vice versa.*

2. The QCD evolution of α_s is internally calculated by solving the renormalization group equation, eq. (3) in section 2.1. Input to the calculation is the value of α_s at a given Q^2 (default: 0.180 and 50 GeV^2 respectively). These values can be changed at any time by e.g:

   ```
   Mz = 91.188
   As = 0.116
   call QNRSET ( 'ALFQ0', Mz*Mz )
   call QNRSET ( 'ALFAS', As )
   ```

   As described in section 2.1 the flavor thresholds in the α_s evolution are taken to be equal to the quark masses which can be set by calls to `QNRSET`: see section 4.5. Notice that you can also set the top mass (default 188 GeV) so that α_s can be evolved with up to $f = 6$ flavors. However, there is no top threshold in the parton distribution evolutions or in the structure function calculations so that there Qcdnum uses only up to $f = 5$ flavors (see below). The function

   ```
   alfas = QALFAS ( Q2, qlambda, nf, ierr )
   ```

   gives the value of α_s for any Q^2. The function also returns the value of Λ (`qlambda`) and the number of active flavors (`nf`).[22] If $Q^2 \leq \Lambda^2$ the value of α_s is arbitrarily set to 100, Λ to zero and the flag `ierr = 1` (`0` otherwise).

3. The flavor thresholds used in the evolution and structure function routines are *not* related to the quark masses. Qcdnum uses instead the thresholds Q_c^2 and Q_b^2 such that $f = 3$ for $Q^2 < Q_c^2$, $f = 4$ for $Q_c^2 \leq Q^2 < Q_b^2$ and $f = 5$ for $Q_b^2 \leq Q^2$. At initialization Qcdnum sets Q_c^2 (Q_b^2) at minus (plus) a very large value so that by default $f = 4$ for all Q^2. You can change this at any time by a call to

   ```
   call QTHRES ( Q2c, Q2b )
   ```

   Thus, if you want to use three flavors throughout then set both Q_c^2 and Q_b^2 beyond the highest Q^2 grid point; to use 5 flavors set them both below the lowest Q^2 grid point.[23] As an example we give below the code to set the thresholds to the square of the quark masses (as in the α_s evolution):

[22] Thus, `qlambda` and `nf` are *output* variables and not input variables as the syntax may suggest.
[23] However always keep $Q_c^2 < Q_b^2$ in the call to `QTHRES`, otherwise a (fatal) error will result.

```
call QNRGET ( 'MCALF', Cmass )
call QNRGET ( 'MBALF', Bmass )
call QTHRES ( Cmass*Cmass, Bmass*Bmass )
```

The function

```
nf = NFLGET ( iq )
```

returns the number of flavors as used by Qcdnum for any grid point `iq`.

4. As explained in section 4.8, weight tables can be calculated once and for all and dumped to disk for the next passes of your analysis. It is convenient to do this on a large grid, say, 120 x-grid points covering $10^{-6} < x < 1$ and 60 Q^2 points covering $0.1 < Q^2 < 10^4$ GeV^2. However one might not always be interested in such a large kinematic range. Instead of creating a new weight file on a smaller grid you can set cuts so that parton distributions are evolved over part of the x–Q^2 grid only. This may result in a considerable reduction in CPU time. The cuts are passed to Qcdnum by

   ```
   call GRCUTS ( xmin, qmin, qmax, roots )
   ```

   where the meaning of `xmin`, `qmin` and `qmax` should be obvious; `roots` $= \sqrt{s}$ restricts the x–Q^2 domain to within the kinematic limit $Q^2 \leq xs$.[24] Once a cut is applied the region outside it becomes inaccessible to the user. `GRCUTS` can be called at any time: to release one or more cuts (or not apply it at all) the corresponding parameter should be set to a value ≤ 0. The function

   ```
   ifail = IFAILXQ ( x, q2 )
   ```

   can be used to investigate if a given x, Q^2 fails or passes the cuts:

   ```
   ifail = ijklm : i = 0/1  no/yes fail QMINA cut (see below)
                   j = 0/1  no/yes fail ROOTS cut
                   k = 0/1  no/yes fail QMAX  cut
                   l = 0/1  no/yes fail QMIN  cut
                   m = 0/1  no/yes fail XMIN  cut
   ```

   Thus, `ifail = 1001` means that x, Q^2 is below the `xmin` cut and above the kinematic limit as defined by `roots`. The corresponding function for the grid point `ix`, `iq` is

   ```
   ifail = IFAILIJ ( ix, iq )
   ```

   The `QMINA` cut mentioned above acts as a Q^2_{min} cut and is automatically set by Qcdnum to the lowest Q^2-grid point above the current value of Λ^2.

[24] In QCD fits for instance you can set `roots` to the highest center of mass energy of your datasets e.g. 300 GeV if HERA data are included.

4.11 QCD evolution of parton distributions

The QCD evolution of parton distributions consists of two steps:

1. Input of the parton distribution for all grid points in x at some fixed value of Q_0^2 (i.e. at a fixed Q^2–grid point `iq0`).
2. QCD evolution over the whole x–Q^2 grid (or part of it when cuts are applied, see section 4.10) by calling one of the evolution routines described below.

The procedure is straight forward for the singlet/gluon evolution and for the non-singlet evolutions of xq^- (valence) and $x\Delta$. However the non-singlet evolution of a xq^+ distribution is more complicated because xq^+ is – per definition – discontinuous at the flavor thresholds.

The value of a parton distribution can be set for a given grid point (`ix,iq`) by

```
call QNPSET ( 'name', ix, iq, value )
```

where `'name'` is a valid name of a memory resident distribution. To set the distribution to zero for all x and Q^2,

```
call QNPNUL ( 'name' ).
```

The following call evolves the gluon and the singlet quark density in Q^2 :

```
call EVOLSG ( iq0, iqmin, iqmax )
```

where `iq0` is the starting point and `[iqmin,iqmax]` the range of the evolution. You must take care that `iqmin` $\leq$ `iq0` $\leq$ `iqmax` (failing to do so will result in a fatal error) and also that the gluon and singlet quark distributions are set to their proper starting values at `iq0` by previous calls to `QNPSET`. The calculation is in LO or in NLO depending on the value of `order` (see section 4.10). For an example of user code evolving the gluon and singlet distributions we refer to the program listing given in section 4.3. We mention at this point that Qcdnum evolution routines do not clear the memory before the evolution. This allows to 'chain' the evolutions, i.e. if `iqmin < iq0 < iq1 < iq2 < iqmax` then the sequence

```
call EVOLSG ( iq0, iqmin, iq1   )
call EVOLSG ( iq1, iq1  , iq2   )
call EVOLSG ( iq2, iq2  , iqmax )
```

is equivalent to

```
call EVOLSG ( iq0, iqmin, iqmax )
```

This feature is of course irrelevant for the singlet/gluon evolution but useful in the context of xq^+ evolution with discontinuities at the flavor thresholds: see below.

A non-singlet distribution of the type xq^- (valence) is evolved by

```
call EVOLNM ( 'name', iq0, iqmin, iqmax )
```

and non-singlet density of the type $x\Delta$ by

```
call EVOLNP ( 'name', iq0, iqmin, iqmax )
```

where `'name'` is a valid name of a non-singlet distribution, that is, not `'gluon'` or `'singlet'` (or whatever names are associated with `id = 0` and `1`).

A non-singlet xq^+ distribution can also be evolved by a call to `EVOLNP` but here one has to be careful with discontinuities at the flavor thresholds (see section 2.2 and 2.5). As an example we assume that

```
1 < iqc < iq0 < iqb < nq2
```

where `iqc` and `iqb` are the grid indices corresponding to the charm and bottom thresholds respectively. To evolve for instance the xs^+ distribution you might code

```
*---    Evolve from iq0 down to iqc and up to iqb
        call EVPLUS ( 'splus', iq0, iqc, iqb )

*---    Downward: splus jumps at iqc because f = 4 --> 3
        factor = 1./4.-1./3.
        call QADDSI ( 'splus', iqc, factor )
        call EVPLUS ( 'splus', iqc, 1, iqc)

*---    Upward: splus jumps at iqb because f = 4 --> 5
        factor = 1./4.-1./5.
        call QADDSI ( 'splus', iqb, factor )
        call EVPLUS ( 'splus', iqb, iqb, nq2 )
```

Here we have introduced the routine `EVPLUS` which is identical to `EVOLNP` except that Qcdnum checks that you do not evolve over the thresholds (without adding the discontinuities) i.e:

```
call EVPLUS ( 'splus', iq0, 1, nq2 )
```

produces, in this example, a (fatal) error message. The routine

```
call QADDSI ( 'name', iq, factor )
```

adds, for all x at the grid point `iq`, `factor` times the singlet distribution to a non-singlet distribution.

Likewise the charm distribution can be evolved from $xq^+(x, Q_c^2) = -1/4x\Sigma(x, Q_c^2)$ by:[25]

[25] Notice that we first set xq_c^+ to zero (`QNPNUL`) for all x and Q^2 before supplying the starting value with `QADDSI`. Otherwise in QCD fits for instance Qcdnum would continue to add $-1/4\Sigma$ to the starting value of xq_c^+ at each Minuit iteration.

```
call QNPNUL ( 'cplus' )

factor = -1./4.
call QADDSI ( 'cplus', iqc, factor )
call EVPLUS ( 'cplus', iqc, iqc, iqb )

factor = 1./4.-1./5.
call QADDSI ( 'cplus', iqb, factor )
call EVPLUS ( 'cplus', iqb, iqb, nq2 )
```

Finally we remark that although Qcdnum knows that it deals with a non-singlet distribution it has no information on what kind of distribution it actually is, xq^-, $x\Delta$ or xq^+. It is therefore the responsibility of the user to call the proper evolution routine and, in case of xq^+ evolution, to handle the threshold discontinuities correctly.

4.12 Access to parton distributions and structure functions

In this section we describe how to access parton distributions and how to calculate structure functions from these. Qcdnum does of course not care where these distributions come from: they may be the result of a Q^2 evolution or, for instance, be read in directly from Pdflib and passed to memory by the appropriate calls to `QNPSET`, see section 4.15.

The current value of a parton density is returned by:

```
value = QPDFIJ ( 'name', ix, iq, iflag )
value = QPDFXQ ( 'name',  x, q2, iflag )
```

where `'name'` is a valid name of a memory resident distribution or of a linear combination (e.g. `'proton'`, see section 4.9). The function `QPDFXQ` gives by linear interpolation the parton distribution at any value of x and Q^2 within the grid. If `(ix,iq)` or `(x,q2)` are outside the grid boundaries or cuts a zero value is returned and `iflag = -1` (`0` otherwise). To protect against use of undefined parton distributions or structure functions Qcdnum will abend with an error message. This behavior can be switched off by setting the logical variable `LIMCK` to `false`

```
call QNLSET ( 'LIMCK', .false. )
```

The structure function corresponding to a quark density or linear combination is calculated by:

```
value = QSTFIJ ( 'opt', 'name', ix, iq, iflag )
value = QSTFXQ ( 'opt', 'name',  x, q2, iflag )
```

where `'opt'` can be set to `'F2'`, `'FL'` or `'XF3'` (see below for the heavy quark structure functions). The calculation is in LO or in NLO depending on the value of `order` (see section 4.10). It is the responsibility of the user to evaluate the structure function from the appropriate distribution, e.g. xF_3 should be calculated from a *non-singlet* quark density and not from something else.[26] The variable `iflag` is set to `-1` if Q^2 (or any other scale like μ_M^2 or μ_R^2, see section 4.13)

[26] An attempt to calculate any structure function from the gluon distribution will result in a fatal error. Apart from this Qcdnum accepts any combination of `'opt'` and `'name'`.

runs outside the grid boundaries or cuts, to `0` if the structure function calculation is successful and to `+1` if the result is obtained from fast calculations (see section 4.14).

The heavy quark contribution to F_2 and F_L can be calculated with `QSTFIJ` or `QSTFXQ` by setting the input option `'opt'` to `'F2C'`, `'FLC'`, `'F2B'` or `'FLB'`, provided of course that the heavy quark weight tables are calculated beforehand. There are some restrictions however; we refer to section 2.5 and make a few remarks below:

- The quark distributions making up the target nucleon and the gluon distribution must have been evolved with $f = 3$ flavors for all Q^2. In fact, a QCD analysis which uses heavy quark structure functions must have Q_c^2 and Q_b^2 set beyond the highest Q^2 grid point to guarantee that Qcdnum uses $f = 3$ in *all* calculations (except α_s).
- The heavy quark structure functions can only be calculated for the neutral current processes $\gamma^* A \to X$ where A stands for the target nucleon i.e. p (proton), n (neutron) or d = (n+p)/2 (deuteron). Qcdnum cannot check this so that it is the responsibility of the user that the input argument `'name'` refers to a correct linear combination, for instance to an object like the `'proton'` as defined in section 4.9 (for $f = 3$ flavors).
- Heavy quark structure functions can be calculated for all Q^2 (within the grid or cuts) but might be numerically inaccurate for $Q^2 < 1.5$ GeV^2 in this version of Qcdnum. Attempts to calculate below $Q^2 = 1.5$ GeV^2 result in a fatal error unless you switch this check off by[27]

```
call QNLSET ( 'CLOWQ', .false. )
```

4.13 The renormalization and factorization scale

In Qcdnum the renormalization and mass factorization scales are defined as

$$\begin{aligned} \mu_R^2 &= a_R \mu_M^2 + b_R \\ \mu_M^2 &= a_M Q^2 + b_M. \end{aligned} \tag{41}$$

The default values for the parameters $a_{R,M}$ and $b_{R,M}$ are given in table 6: notice that the mass factorization scale can be defined independently for the light and the heavy quark structure functions.[28] Qcdnum insists that all scales are within the Q^2 grid boundaries and above the current value of Λ^2. In addition the scale μ_M^2 is required to lie within the cuts, if any.

In practice only one scale at the time is varied. For instance one may identify the factorization scale μ_M^2 with Q^2 and vary μ_R^2 in the range $Q^2/4 < \mu_R^2 < 4Q^2$, say. This variation affects the value of α_s (being taken at μ_R^2 instead of Q^2) and also the evolution of the parton distributions through a modification of the NLO splitting functions.

In the second case μ_R^2 is identified with μ_M^2 which is varied in the range $Q^2/4 < \mu_M^2 < 4Q^2$ for instance. This variation does not affect the evolution but changes the coefficient functions and therefore relation between the structure functions (given at the scale Q^2) and the parton distributions (given at the scale μ_M^2).

[27]It is probably safe to calculate the structure functions down to $Q^2 = 0.5$ GeV^2 or so.
[28]Changing the scales μ_R^2 or μ_M^2 does not require a re-calculation of the weight tables.

```
+-------+---+-------+--------------+------------------------------------+
| var   |typ| deflt |     value    | description                        |
+-------+---+-------+--------------+------------------------------------+
| AAAR2 | R |  1.0  |  0.10000E+01 | R2 = A*M2 + B (ren. scale)         |
| BBBR2 | R |  0.0  |  0.00000E+00 | R2 = A*M2 + B (ren. scale)         |
| AAM2L | R |  1.0  |  0.10000E+01 | M2 = A*Q2 + B (light fact. scale)  |
| BBM2L | R |  0.0  |  0.00000E+00 | M2 = A*Q2 + B (light fact. scale)  |
| AAM2H | R |  1.0  |  0.10000E+01 | M2 = A*Q2 + B (heavy fact. scale)  |
| BBM2H | R |  0.0  |  0.00000E+00 | M2 = A*Q2 + B (heavy fact. scale)  |
+-------+---+-------+--------------+------------------------------------+
```

Table 6: Default settings for the renormalization and mass factorization scale

Because in the previous sections no reference is made to the different scales (they were all identified with Q^2) some confusion may arise when μ_M^2 is varied with respect to Q^2. The rule is that parton distribution input, evolution and output are defined on the scale μ_M^2. The structure functions however are given on the scale Q^2. As an example let us assume that $\mu_M^2 = Q^2/4$. Then

```
value = QPDFXQ ( 'name', x, 8.D0, iflag )
```

returns the parton distribution at $\mu_M^2 = 8$ GeV^2 whereas

```
value = QSTFXQ ( 'opt', 'name', x, 8.D0, iflag )
```

gives the structure function at $Q^2 = 8$ GeV^2. This structure function is calculated from the parton distributions at $\mu_M^2 = Q^2/4 = 2$ GeV^2. Indeed, the reader may verify that in LO the F_2 structure function given by `QSTFXQ` at 8 GeV^2 is equal to the quark distribution given by `QPDFXQ` at 2 GeV^2.

4.14 Fast structure function calculation

The calculation of structure functions is expensive because they require (in NLO) the evaluation of convolution integrals. This is particularly true for `QSTFXQ(..,x,q2,..)` where, in general, the result is obtained from linear interpolation of *four* structure functions calculated at the four grid points surrounding x and Q^2. The use of `QSTFXQ` is inefficient if one or more of these grid points coincide with those of a previous call. This overhead can be eliminated if Qcdnum knows beforehand for which values of x and Q^2 it has to calculate the structure functions.

The structure function calculations might thus be speeded up considerably as follows:

1. Pass a list of x, Q^2 points where structure functions are to be calculated. This is done by calling the subroutine `QFMARK` e.g:

```
do i = 1,ndata
  x  = xdata(i)
  q2 = qdata(i)
  call QFMARK ( x, q2 )
enddo
```

This routine just marks the four grid points surrounding each data point x, Q^2. The routine `QFMNUL` clears all the marks. The grid points have to be marked only once somewhere in the initialization phase of the user program, but of course *after* the x–Q^2 grid has been defined.

2. The call

```
call STFAST ( 'opt', 'name' )
```

   calculates the structure function for all marked grid points. As before, `'opt'` can be set to `'F2'`, `'FL'`, ... and `'name'` to a valid name of a memory resident distribution or linear combination, see section 4.12. `STFAST` does of course the same as `QSTFIJ` but the code is more efficient. `STFAST` stores the results for later interpolation in 20 memory locations reserved for this purpose. If more than 20 different structure functions are calculated (i.e. more than 20 different combinations of `'opt'` and `'name'`) `STFAST` runs out of storage space and acts as a do-nothing. Qcdnum then switches automatically to slow structure function calculations.

3. Finally the value of the structure function can be obtained by calling `QSTFIJ` or `QSTFXQ` as described in section 4.12. These functions will, if possible, pick up the results calculated by `STFAST` and perform the interpolation, if any; if not possible the structure function is calculated from scratch.

4. In the unlikely event that you want to calculate more than 20 structure functions with `STFAST` you can, after step (3), clear the memory allocation by calling `STFCLR` and repeat steps (2) and (3) for a next batch of 20 structure functions.

A call to `QPRINT(lun,'Stats')` prints a log as shown in table 7. From this log it is seen that (almost) all structure functions are obtained from `STFAST` and that no structure functions are calculated outside the grid or cuts.

To summarize, you can speed up the structure function calculations by simply marking grid points using `QFMARK` and calling `STFAST` before the calls to `QSTFIJ` or `QSTFXQ`. See section 5 on the performance.

Structure function calls	F2	FL	xF3	F2h	FLh
Slow calculation	577	0	10	624	0
Fast calculation	1423961	0	93241	1425183	0
Outside grid or cuts	0	0	0	0	0
Total	1424538	0	93251	1425807	0

Table 7: Qcdnum log of the number of structure function calls

4.15 Structure functions from external parton distribution sets

As mentioned in section 4.12 Qcdnum can be used to calculate structure functions from any given parton distribution set. These parton distributions can be obtained from e.g. Pdflib and stored in Qcdnum memory using the routine `QNPSET`. Doing so you should keep the following in mind:

- The parton distribution set should have been defined in the $\overline{\mathrm{MS}}$ scheme.
- The order (LO or NLO), the flavor thresholds and α_s in Qcdnum should be set identical to those of the parton distribution set.[29]
- The quark distributions stored in Qcdnum memory should either be singlet ($x\Sigma$) or non-singlet. Composite objects like the proton can be defined using the routine `QNLINC`, see section 4.9. It is for instance *wrong* to store the charged weighted sum of quark distributions and calculate structure functions from this.
- When NLO structure functions are calculated from the singlet quark distribution or from a linear combination containing the singlet, the gluon distribution must be supplied as well. For non-singlet structure functions the gluon distribution is not needed.
- When heavy flavor structure functions are calculated the parton distribution set must have been evolved with $f = 3$ flavors only.
- When the parton densities are read in for part of the x–Q^2 grid only one may delimit this region by appropriate cuts so that Qcdnum can verify that no undefined distributions are used.
- Once the parton distributions are read in one can study the factorization scale dependence of the structure functions by varying μ_M^2. A variation of the renormalization scale does not make sense because this requires an evolution of the parton distributions, see section 2.4.

4.16 Access to more information

Qcdnum info is printed on logical unit number `lun` by a call to

```
call QPRINT ( lun, 'opt' )

opt = 'P'  Qcdnum parameters and options
      'X'  x-Q2 grid
      'B'  Booking list of parton distributions
      'S'  Stats on structure function calls
      'T'  Timelog
      'A'  All of the above
```

A logical unit number other than 6 should be opened by the user. An example of a timelog is given in table 8. It gives for the evolution routines the number of calls, the number of evolutions

[29] Notice that the authors of the set may have used a different α_s evolution algorithm than Qcdnum. This can lead to (small) inconsistencies.

```
-------------------------------------------------
Routine     # calls   # evols   CPU sec  CPU/evol
-------------------------------------------------
EVOLNM         1000     431.6      14.7      0.03
EVOLNP         1000     431.6      15.2      0.04
EVOLSG          500     215.8      22.7      0.11
-------------------------------------------------
AP total       2500    1079.0      52.6      0.05

STFAST          500  250000.0       7.3
QNFILW            0                 0.0
Other                              13.1
-------------------------------------------------
Total                              73.0
-------------------------------------------------
```

Table 8: Example of a Qcdnum timelog

normalized to the size of the x–Q^2 grid,[30] the CPU time spent and the number of CPU seconds per evolution. Also given is the time spent in fast structure function calculations[31] and that in creating the weight tables (none in this example since the weights were taken from disk). The entry 'other' gives the time spent in any other Qcdnum routine (e.g. `QSTFXQ`) or user routine (e.g. Minuit) between the moment `QNTIME` was called and the moment of the printout.[32]

Splitting and coefficient functions can be accessed by

```
val = QNSPLF ( 'opt', x, q2, nf )
```

[30]Parton distributions are often evolved over part of the grid only.
[31]The second column gives the number of structure function calculations.
[32]If you want to continue logging after the printout please call `QNTIME` with argument `'Start'` (start a new log) or `'Continue'` (continue with the present log).

Option	Function	Option	Function
PFF1	$P^{(0)}_{FF}(x)$	C1Q	$C^{(1)}_{1,q}(x)$
PFG1	$P^{(0)}_{FG}(x,f)$	C1G	$C^{(1)}_{1,g}(x,f)$
PGF1	$P^{(0)}_{GF}(x)$	C2Q	$C^{(1)}_{2,q}(x)$
PGG1	$P^{(0)}_{GG}(x)$	C2G	$C^{(1)}_{2,g}(x,f)$
PFF2	$P^{(1)}_{FF}(x,f)$	CLQ	$C^{(1)}_{L,q}(x)$
PFG2	$P^{(1)}_{FG}(x,f)$	CLG	$C^{(1)}_{L,g}(x,f)$
PGF2	$P^{(1)}_{GF}(x,f)$	C3Q	$C^{(1)}_{3,q}(x)$
PGG2	$P^{(1)}_{GG}(x,f)$		
PPL2	$P^{(1)}_{+}(x,f)$		
PMI2	$P^{(1)}_{-}(x,f)$		

Table 9: Selection options in `QNSPLF`

The input character variable `'opt'` selects the splitting or coefficient function as listed in table 9. As indicated in this table, some functions depend on x only (in which case the value of `q2` and `nf` are irrelevant) whilst others depend in addition on either f (then `q2` is irrelevant) or Q^2 (then `nf` is irrelevant). The function `QNSPLF` is not affected by grid definitions and may thus be called at any time after `QNINIT`. Several splitting/coefficient functions are not defined for $x = 1$ (only their integral over an interval containing $x = 1$ as an upper limit): these functions are set to zero at $x = 1$.

4.17 Qcdnum error handling

Qcdnum is robust or at least attempts to be: it should either give the correct answer (numerically) or grind to a halt printing an error message. An example of such a message is given in table 10: here we have attempted to calculate the proton F_2 structure function outside the grid boundaries or cuts. Fatal errors might have their cause upstream. For example if weights are calculated and the x–Q^2 grid is changed afterwards, the weight tables (which are defined on the grid) are invalidated. The first call to a routine which uses them will cause an abend complaining that the weights are not available. To avoid such errors we recommend to stick to the flow of action as described in section 4.2.

In spite of the error checking mechanism there is still room to go astray with a Qcdnum based analysis. As a reminder we summarize below the most important do's and dont's described in the previous sections.

- `QNINIT` must be called before any other Qcdnum routine.
- When run in double precision mode floating point variables passed to Qcdnum should

```
+---------------------------------------------------+
|                                                   |
|  ------------------------------------             |
|  QCDNUM error in s/r QSTFXQ ---> STOP             |
|  ------------------------------------             |
|  Input Opt  : F2                                  |
|        Name : PROTO                               |
|        x    :  0.63000E-04                        |
|        Q2   :  0.35000E+01                        |
|                                                   |
|  X, Q2 or mu2 outside grid or cuts                |
|                                                   |
|  x  = 0.63000E-04 xmin        =  0.10000E-04 pass |
|  Q2 = 0.35000E+01 Qmin        =  0.75000E+00 pass |
|  Q2 = 0.35000E+01 Qmax        =  0.60000E+04 pass |
|  s  = 0.55556E+05 Smax        =  0.44100E+05 fail |
|  Q2 = 0.35000E+01 Qmin_alphas =  0.16818E+00 pass |
|                                                   |
+---------------------------------------------------+
```

Table 10: Example of a Qcdnum error message

be declared double precision in the calling routine and actual values should be written in double precision format e.g. `0.188D0`. Qcdnum floating point functions must also be declared double precision.

- Do not change Qcdnum parameters in between calculations such as evolving in NLO and calculating structure functions in LO.

- Input parton distributions must be the gluon, the singlet or non-singlet momentum densities and not a mixture of these. Qcdnum assumes that the gluon is stored in memory location `ID = 0`, the singlet in `ID = 1` and the non-singlet distributions in the remaining locations `ID = 2-10`.

- Qcdnum does not verify if starting values of parton distributions are supplied (it checks if they are booked though).

- Evolution routines do not clear the memory before the evolution. Thus, if you evolve over a restricted range `iqmin`, `iqmax` in Q^2 (or apply cuts) then outside this range the value of the parton distribution will remain to be whatever it was before the evolution.

- Non-singlet xq^+ distributions are discontinuous at the flavor thresholds (if any). The discontinuities are not automatically taken into account by the evolution routines.

- xF_3 is a non-singlet structure function and should therefore be calculated from (a linear combination of) non-singlet densities only.

- Heavy quark $F_{2,L}$ structure functions should only be calculated from linear combinations representing the proton, deuteron or neutron quark distributions in charged lepton scattering. The parton distributions must have been evolved with $f = 3$ flavors for all Q^2.

5 Size, accuracy and speed

The size of the Qcdnum executable can be controlled by setting the parameters `mxx` and `mq2` to the maximum number of grid points you want to use in your analysis (see section 4.1). For `mxx = 100 (150)` and `mq2 = 40` the size is 3.3 (5.2) Mbyte (as measured with the Unix command 'size').

The accuracy of Qcdnum depends on the choice of the x–Q^2 grid as illustrated in fig. 2. Here we take as a reference parton distributions evolved from $Q^2 = 4$ up to $Q^2 = 10^4$ GeV^2 on a 400×60 x–Q^2 grid covering a large range $10^{-5} \leq x \leq 1$. For such a grid Qcdnum results were shown to be in agreement with several other NLO evolution codes to within 0.05% [5]. Choosing $n_x =$ (200, 100, 70) the accuracy on F_2 (and on the quark distributions) is within (0.1, 0.5, 1)% at the highest Q^2 as shown in fig. 2a.[33] This is slightly worse for the gluon distribution (fig. 2b), in particular at high x but here the gluon density vanishes. The deviations are of course zero at the input scale (4 GeV^2) and increase linearly with $\ln Q^2$ to the values at $Q^2 = 10^4$ GeV^2 shown in fig. 2.

[33] The accuracy deteriorates rapidly if the x–grid has less than 70 points.

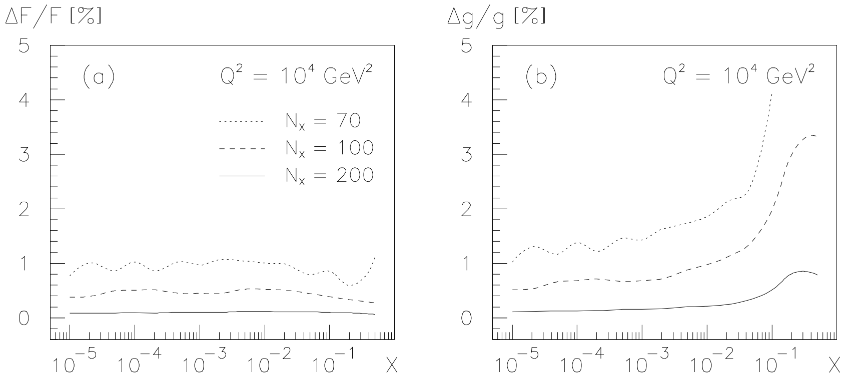

Figure 2: (a) The deviation $\Delta F_2/F_2$ and (b) $\Delta xg/xg$ at $Q^2 = 10^4$ GeV^2 as a function of x for a 70, 100 and 200 point x grid and a 60 point Q^2 grid. The deviations are taken with respect to parton distributions evolved on a 400×60 x–Q^2 grid.

It turns out that Qcdnum is rather insensitive to the number of Q^2 grid points: the change in xg is not more than 0.06% if n_q is reduced from 60 to 20.[34]

Figure 3 shows the CPU time spent for 500 passes through a Qcdnum based calculation as function of the number of x-grid points. Each pass consisted of one singlet/gluon evolution and four non-singlet evolutions. Furthermore F_2 structure functions were calculated for 500 points randomly distributed below the kinematic limit $Q^2 = xs$ with $\sqrt{s}$ set to 300 GeV. The amount of CPU time increases quadratically with the number of grid points in x (and linearly with that in Q^2, not shown) but is still only about 8 minutes for 2500 evolutions and 25×10^4 structure function evaluations on a 150×60 grid. Also shown in fig. 3 is the effect of a $\sqrt{s}$ cut which speeds up the evolutions by more than a factor of two (fig. 3b).[35] Only a quarter of the CPU time is spent when `STFAST` is used to calculate the structure functions instead of `QSTFXQ` (fig. 3c). In practical cases however the gain will be less (factor 2 typically) since structure function data are usually ordered in bins of x and Q^2 instead of being randomly distributed as in this example.

To summarize, a good initial choice is about 100 grid points in x and about 40 in Q^2. This gives an executable size of about 3 Mbyte, an accuracy of well within 1% over a large kinematic range and execution times of the order of 10 minutes for several thousand evolutions and a few hundred thousand structure function evaluations.

[34] If heavy quark structure functions are calculated the number of Q^2 grid points should however not be taken too small.

[35] In QCD fits you can still have the evolution on the whole x–Q^2 grid by releasing the cut once the fit has converged ('iflag = 3' in Minuit language).

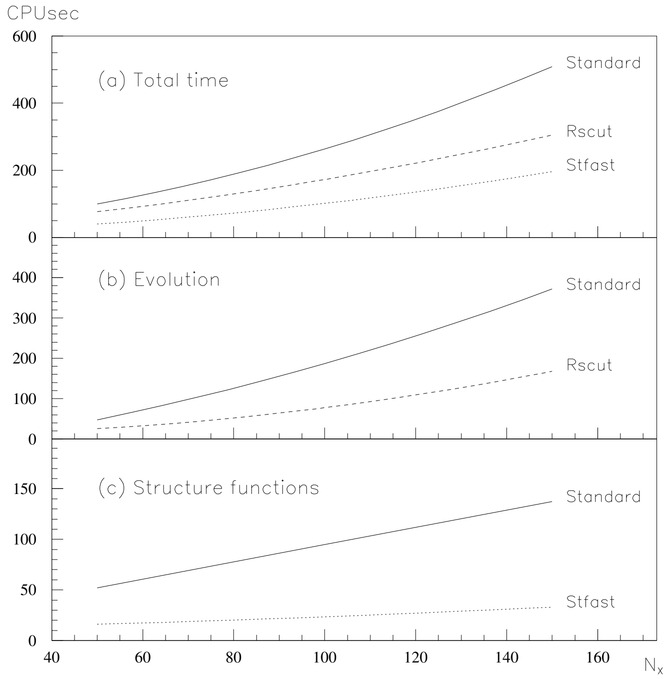

Figure 3: CPU time spent for 2500 evolutions and 250000 F_2 calculations as function of the number of x–grid points for a 60 point Q^2–grid. (a) Total time; (b) time spent in QCD evolutions; (c) time spent in structure function calculations. The dashed curves correspond to evolutions on part of the grid and the dotted curves to invoking fast structure function calculation in addition.

6 Qcdnum subroutine calls (reference section)

In this section we give a short description of all Qcdnum subroutine calls (see also table 2 in section 4.2). We use the convention described in section 4.2 to denote logical, integer, floating point and character variables. Output variables are indicated with an asterisk.

6.1 Description of subroutines

```
call QNINIT
```

To be called before anything else. Initializes a large set of constants and sets reasonable defaults.

```
call QNVERS ( *'version', *Ldouble, *nxmax, *nqmax )
```

Output:

- `'version'`: current version number. Should be declared `character*8` in the calling program.

- `Ldouble`: returned as `.true.` (`.false.`) if the current version is in double (single) precision. Should be declared `logical` in the calling program.
- `nxmax`, `nqmax`: the maximum number of grid points in x and Q^2 the user can define.

call QNTIME ('option')

Start Qcdnum timelog (option `'Start'`), stop the log (`'Hold'`) or continue with the current log (`'Continue'`). A printout (see `QPRINT`) stops the current timelog.

call QNtSET ('tvar', tval)

- `QNLSET('Lvar',Lval)`: set the logical Qcdnum variable `'Lvar'` to `Lval`.
- `QNISET('Ivar',Ival)`: as above, but for integer variables.
- `QNRSET('Rvar',Rval)`: as above, but for real/double precision variables.

See table 3 in section 4.5 for a list of variables you can set.

call QNtGET ('tvar', *tval)

As `QNtGET` but now retrieve the value `tval` corresponding to the Qcdnum variable `'tvar'`.

call GRXDEF (nx, xmi)

Clear the current x grid and generate a grid covering the range $x_{min} \leq x \leq 1$ with `nx` points. The grid is logarithmic (linear) below (above) the value `'scaxo'` (default 0.2, can be reset by a call to `QNRSET`).

call GRQDEF (nq, qmi, qma)

Clear the current Q^2 grid and generate a logarithmic grid covering the range $Q^2_{min} \leq x \leq Q^2_{max}$ with `nq` points.

call GRXINP (xarray, nx)

Add `nx` points as specified in the input array `xarray` to the x–grid. The list of grid points may be given in arbitrary order. The routine will always generate the grid point $x = 1$ even if it is not included in the list `xarray`.

call GRQINP (qarray, nq)

As `GRXINP`, but now add `nq` points to the Q^2–grid.

```
call GRtNUL
```

- GRXNUL: clear the x grid.
- GRQNUL: clear the Q^2 grid.

```
call GRMXMQ ( *nxmax, *nqmax )
```

Returns the maximum number of grid points the user can define. See section 4.1 on how to set these (in the Qcdnum source code).

```
call GRGIVE ( *nx, *xmi, *xma, *nq, *qmi, *qma )
```

Output:

- `nx, xmi, xma`: number of grid points and limits of the current x–grid. The grid point $x = 1$ is not included in the count `nx`. `GRGIVE` should always return `xma = 1`.
- `nq, qmi, qma`: number of grid points and limits of the current Q^2–grid.

```
call GRXOUT ( *xgrid )
```

Copy the current x–grid to a local array `xgrid` which must be dimensioned to at least `nx` in the calling routine. The array `xgrid` does not contain the point $x = 1$.

```
call GRQOUT ( *qgrid )
```

Copy the current Q^2–grid to a local array `qgrid` which must be dimensioned to at least `nq` in the calling routine.

```
x = XFROMIX ( ix )
```

Returns the value corresponding to the grid point `ix`. If `ix` is outside the grid boundaries or if the x-grid is not defined, `XFROMIX` returns zero.

```
q2 = QFROMIQ ( iq )
```

As above, but now for the Q^2–grid.

```
ix = IXFROMX ( x )
```

Returns the grid index `ix` corresponding to a given value of `x`.

- If `x` is outside the grid boundaries, `ix` is set to zero.

- If x coincides with a grid point ix is positive and corresponds to that grid point.
- If x does not coincide with a grid point, ix is negative and -ix corresponds to the closest grid point which lies below x.

```
iq = IQFROMQ ( q2 )
```

As above, but now for the Q^2–grid.

```
ix = IXNEARX ( x )
```

As IXFROMX but ix now corresponds to the grid point closest to x instead of that below x.

```
iq = IQNEARQ ( q2 )
```

As above, but now for the Q^2–grid.

```
call GRCUTS ( xmi, qmi, qma, roots )
```

Pass cuts to Qcdnum such that evolutions are only performed in the kinematic domain defined by $x_{min} \leq x \leq 1$, $Q^2_{min} \leq Q^2 \leq Q^2_{max}$ and $Q^2 \leq xs$, where roots $= \sqrt{s}$ is the center-of-mass energy. To release a cut or not apply it at all, set the corresponding parameter to a value ≤ 0. GRCUTS can be called at any time even before the x–Q^2 grid is defined.

```
istat = IFAILIJ ( ix, iq )
```

Returns a non-zero value if ix or iq falls outside the grid boundaries or cuts:

```
ifail = ijklm : i = 0/1  no/yes fail QMINA cut
                j = 0/1  no/yes fail ROOTS cut
                k = 0/1  no/yes fail QMAX  cut
                l = 0/1  no/yes fail QMIN  cut
                m = 0/1  no/yes fail XMIN  cut
```

The QMINA cut is set automatically by Qcdnum to the lowest Q^2 grid point above the current value of the QCD scale parameter Λ^2.

```
istat = IFAILXQ ( x, q2 )
```

As above but now for x, Q^2.

```
call QNFILW ( 0, 0 )
```

Generate weight tables needed for the calculation of convolution integrals in the QCD evolution and structure function routines. Should be called after the x–Q^2 grid is defined. For a complete description of QNFILW see section 4.7.

```
call QNDUMP ( lun )
```

Write weight tables to logical unit number lun which should have been opened before by the user e.g:

```
OPEN(unit=24,file=weights.file,form='unformatted',status='unknown')
```

Written to the file are (i) the Qcdnum version number, (ii) the parameters mxx and mq2 (see section 4.1), (iii) the x and Q^2 grid, (iv) the current value of the charm and bottom mass and (v) those weight tables which are available when QNDUMP is called. See section 4.8 for details.

```
call QNREAD ( lun, istop, *ierr )
```

Read weight tables from logical unit number lun which should have been opened before by the user (see above). This routine will not only read the weight tables but also overwrite the x–Q^2 grid (if any) and the charm/bottom mass in memory. See section 4.8 for a description of the input parameter istop and the output flag ierr.

```
call QNBOOK ( id, 'name' )
```

Associate memory location id with the name of a parton density:

- id = 0: reserved for the gluon distribution with the predefined name 'gluon'.
- id = 1: reserved for the singlet quark distribution with the predefined name 'singl'.
- id = 2-10: reserved for non-singlet distributions as defined by the user. At initialization the names of these locations are set to 'free'.
- Qcdnum translates all names to upper case and truncates them to the first five characters which must be unique.
- An identifier cannot be booked twice unless the name has been previously set to 'free'.

```
call QNLINC ( id, 'name', nf, factors )
```

Define a linear combination of quark distributions:

- id = 11-30: input identifier.
- 'name': name of the linear combination.
- nf = 3-5: the number of flavors. For each combination of id and 'name' you can define three different linear combinations for $f = 3$, 4 or 5 flavors.

- `factors`: 10-dimensional floating point input array containing the multiplication factor for the singlet distribution in `factors(1)` and that for the non-singlet distributions in `factors(2)` through `(10)`.

```
id = IPDFID ( 'name' )
```

Returns for a given `name` the identifier `id`. The input name should correspond to that of a memory resident quark distribution i.e. not `gluon` or the name of a linear combination. Qcdnum abends with a fatal error message if it cannot find an identifier associated with the input name.

```
call QTHRES ( q2c, q2b )
```

Set the flavor thresholds Q_c^2 and Q_b^2; $Q_c^2 < Q_b^2$. Can be called at any time, even before the x–Q^2 grid is defined. At initialization Q_c^2 (Q_b^2) is set to minus (plus) a very large value so that by default $f = 4$ in all Qcdnum calculations. Actual values can be retrieved by `QNRGET('tchrm',q2c)` and `QNRGET('tbott',q2b)` respectively.

```
nf = NFLGET ( iq )
```

Get the number of flavors associated with the grid point `iq`. If `iq` is outside the grid boundaries `NFLGET` returns with a fatal error.

```
alphas = QALFAS ( q2, *qlambda, *nf, *ierr )
```

Calculate $\alpha_s(Q^2)$. The calculation depends on the input value of $\alpha_s(Q_0^2)$ at some reference scale Q_0^2, the quark mass thresholds and the order of the Qcdnum calculations. All these parameters are set at initialisation and can be changed by calls to `QNISET/QNRSET`, see section 4.5. Apart from α_s the function also returns value of the QCD scale parameter Λ, the number of active flavours f and an error flag `ierr` which is set to 0 (1) if Q^2 is above (below) Λ^2. If $Q^2 \leq \Lambda^2$ (`ierr = 1`) then (arbirarily) `QALFAS = 100` and `qlambda = 0`.

```
call QNPSET ( 'name', ix, iq, value )
```

Set the value of the parton distribution `'name'` at the grid point (`ix`,`iq`). A fatal error occurs if `'name'` does not correspond to that of a memory resident distribution, if the x–Q^2 grid is not defined or if (`ix`,`iq`) is outside the grid boundaries.

```
call QADDSI ( 'name', iq, factor )
```

Add `factor` times the singlet distribution to a non-singlet distribution. The addition is done at fixed `iq` for all grid points in x. See section 4.11 on how `QADDSI` is used to add discontinuities of xq^+ distributions at the flavor thresholds. Error messages: as for `QNPSET`; in addition `QADDSI` cannot be called for the gluon and the singlet quark distribution.

```
call QNPNUL ( 'name')
```

Set the patron distribution `'name'` to zero. A fatal error occurs if `'name'` does not correspond to a memory resident distribution.

```
call EVOLSG ( iq0, iqmin, iqmax )
```

Evolve the gluon and the singlet quark distribution over the range `[iqmin,iqmax]` starting from the grid point `iq0`. A fatal error occurs if `iq0` lies outside the range `[iqmin,iqmax]`. This range might have been adjusted internally by Qcdnum if Q^2 cuts are made. For further details see section 4.11.

```
call EVOLNP ( 'name', iq0, iqmin, iqmax )
```

As above, but now evolve a non-singlet xq^+ or $x\Delta$ quark distribution. A fatal error will occur if `'name'` does not correspond to a non-singlet memory resident distribution. For further details see section 4.11.

```
call EVPLUS ( 'name', iq0, iqmin, iqmax )
```

As above, but check that the number of flavors does not change in the range `[iqmin,iqmax]`. To be used for xq^+ evolution which is discontinuous at the flavor thresholds. For further details see section 4.11.

```
val = QPDFIJ ( 'name', ix, iq, *iflag )
```

Returns the value of the parton distribution `name` or linear combination at the grid point `ix`, `iq`. If `ix` or `iq` are outside the grid boundaries or cuts `val` is set to zero and `iflag` to `-1` (`0` otherwise). Qcdnum will then abend with an error message unless the flag `LIMCK` has been set to `false`.

```
val = QPDFXQ ( 'name', x, q2, *iflag )
```

As above but now for x and Q^2.

```
val = QSTFIJ ( 'opt', 'name', ix, iq, *iflag )
```

As `QPDFIJ` but calculate the structure function `opt` of parton distribution (or linear combination) `name`. The input variable `opt` can be set to `F2`, `FL`, `xF3`, `F2C`, `FLC`, `F2B` or `FLB`. Qcdnum accepts every combination of `opt` and `name` except the name associated with the gluon distribution. The output variable `iflag` is set to

- `-1`: `ix` and/or `iq` outside grid or cuts. In this case `val` is set to zero. Qcdnum abends with a fatal error unless `LIMCK` is set to `false`.

- `0`: calculation successful.
- `+1`: structure function obtained from the results of `STFAST` (see below).

For further details on the use of `QSTFIJ` see section 4.12, 4.14 and 4.15.

```
val = QSTFXQ ( 'opt', 'name', x, q2, *iflag )
```

As above but now for x and Q^2.

```
call QFMARK ( x, q2 )
```

Mark the four grid points surrounding x and Q^2 for fast structure function calculation. Qcdnum abends with a fatal error message if x and/or Q^2 are outside the grid boundaries.

```
call QFMNUL
```

Clear all the marks previously set by `QFMARK`.

```
call STFAST ( 'opt', 'name' )
```

Calculate the structure function `opt` (see above) of parton distribution `name` for all grid points marked by `QFMARK`. The results are stored internally for later interpolation (by `QSTFIJ` or `QSTFXQ`) for up to 20 different combinations of `opt` and `name`. If there are more than 20 different combinations `STFAST` acts as a do-nothing.

```
call STFCLR
```

Clear the memory allocation made by `STFAST`. Previous results calculated by `STFAST` are invalidated.

```
call QPRINT ( lun, 'opt' )
```

Print Qcdnum info on logical unit number `lun`. A logical unit number other than 6 should be opened by the user. The input parameter `opt` can be set to

```
'P'  Qcdnum parameters and options.
'X'  x-Q2 grid.
'B'  Booking list of parton distributions.
'S'  Stats on structure function calls.
'T'  Timelog.
'A'  All of the above.
```

A printout of the timelog (option `T`) causes a halt of the logging. If the log is to be continued please call `QNTIME('Start')` or `QNTIME('Continue')` after `QNPRINT`.

7 Acknowledgments

I am of course much indebted to the original authors of Qcdnum: M. Virchaux and A. Ouraou. I would like to thank M. Virchaux for making the code available to me in 1991 and for his contributions during the first steps on the tortuous road to bring the program up to its present version. J. O'Mara and M. Vreeswijk contributed to the Qcdnum upgrades for the QCD analysis of the ZEUS 1993 F_2 data. The NLO heavy quark coefficient functions were taken from a program kindly provided by S. Riemersma. Qcdnum results were regularly checked against those obtained from a similar program developed independently by M. Vreeswijk. Detailed comparison with several other NLO evolution codes was made during the HERA 1995-1996 workshop: here I much enjoyed the collaboration with J. Blümlein, C. Pascaud, S. Riemersma, A. Vogt and F. Zomer. I am grateful to A. Vogt for many discussions and for further checks beyond those made during the HERA workshop. Finally I would like to thank my colleagues from NMC and ZEUS for help and encouragement.

A QCD splitting and coefficient functions

The leading order splitting functions are [14]:

$$\begin{aligned}
P^{(0)}_{FF}(z) &= \frac{4}{3}\left[\frac{1+z^2}{(1-z)_+} + \frac{3}{2}\,\delta(1-z)\right] \\
P^{(0)}_{FG}(z) &= \frac{f}{2}\left[z^2 + (1-z)^2\right] \\
P^{(0)}_{GF}(z) &= \frac{4}{3}\left[\frac{1+(1-z)^2}{z}\right] \\
P^{(0)}_{GG}(z) &= 6\left[\frac{z}{(1-z)_+} + \frac{1-z}{z} + z(1-z) + \left(\frac{11}{12} - \frac{f}{18}\right)\delta(1-z)\right]
\end{aligned} \tag{42}$$

where the so-called '+' prescription is defined by:

$$[f(x)]_+ = f(x) - \delta(1-x)\int_0^1 f(y)dy. \tag{43}$$

Notice that

$$\int_x^1 f(z)[g(z)]_+ dz = \int_x^1 [f(z) - f(1)]g(z)dz - f(1)\int_0^x g(z)dz$$

and in particular that

$$\int_x^1 dz \frac{f(z)}{(1-z)_+} = \int_x^1 dz\, \frac{f(z)-f(1)}{(1-z)} + f(1)\ln(1-x).$$

The coefficient functions are given by [7]:

$$\begin{aligned}
C^{(1)}_{2,q}(x) &= \frac{4}{3}\left[\frac{1+x^2}{1-x}\left(\ln\frac{1-x}{x} - \frac{3}{4}\right) + \frac{1}{4}(9+5x)\right]_+ \\
C^{(1)}_{2,g}(x) &= f\left[(x^2+(1-x)^2)\ln\left(\frac{1-x}{x}\right) - 1 + 8x(1-x)\right]
\end{aligned} \tag{44}$$

$$\begin{aligned} C_{L,q}^{(1)}(x) &= \frac{8}{3}x \\ C_{L,g}^{(1)}(x) &= 4fx(1-x) \\ C_{3,q}^{(1)}(x) &= C_{2,q}^{(1)} - \frac{4}{3}(1+x). \end{aligned} \tag{45}$$

Notice the '+' prescription in $C_{2,q}^{(1)}$: a non-singlet F_2 structure function, for instance, is thus given by:

$$F^{ns}(x) = xq^{ns}(x) + \frac{\alpha_s}{2\pi}\left\{x\int_x^1 \frac{dy}{y^2}\left[yq^{ns}(y) - xq^{ns}(x)\right]C_{2,q}^{(1)}(x/y) - xq^{ns}(x)\int_0^x dyC_{2,q}^{(1)}(y)\right\}. \tag{46}$$

References

[1] A. Ouraou, Ph. D. Thesis, Université de Paris-XI (1988);
M. Virchaux, Ph. D. Thesis, Université de Paris-VII (1988).

[2] M. Virchaux and A. Milsztajn, Phys. Lett B274 (1992) 221.

[3] NMC, M. Arneodo et al., Phys. Lett. B309 (1993) 222.

[4] ZEUS Collab., M. Derrick et al., Phys. Lett. B345 (1995) 576.

[5] J. Blümlein et al., Proc. workshop 'Future Physics at HERA', eds. G. Ingelman, A. de Roeck and R. Klanner, Vol. 1, p. 23 (1996).

[6] V.N. Gribov and L.N. Lipatov, Sov. J. Nucl. Phys. 15 (1972) 438, 675;
L.N. Lipatov, Sov. J. Nucl. Phys. 20 (1974) 181;
G. Altarelli and G. Parisi, Nucl. Phys. B126 (1977) 297.

[7] W. Furmanski and R. Petronzio, Z. Phys. C11 (1982) 293.

[8] G. Gurci, W. Furmanski and R. Petronzio, Nucl. Phys. B175 (1980) 27.

[9] W. Furmanski and R. Petronzio, Phys. Lett. 97B (1980) 437.

[10] G. Ingelman and R. Rückl, Z. Phys. C44 (1989) 291;
J. Blümlein et al., Z. Phys. C45 (1990) 501;
R. G. Roberts, 'The Structure of the Proton', Cambridge University Press (1990).

[11] A.D. Martin, W.J. Stirling and R.G. Roberts, Phys. Lett. B266 (1991) 173.

[12] J. Blümlein et al., Nucl. Phys. B (Proc. Suppl.) 51C (1996) 97.

[13] S. Riemersma, J. Smith and W.L. van Neerven, Phys. Lett. B347 (1995) 143 and references therein.

[14] A. Buras, Rev. Mod. Phys. 52 (1980) 199.