QCD evolution with QCDNUM

Michiel Botje

Nikhef, PO Box 41882, 1009DB Amsterdam

m.botje@nikhef.nl

Proton Structure in the LHC Era

DESY Hamburg 22-24 October 2012

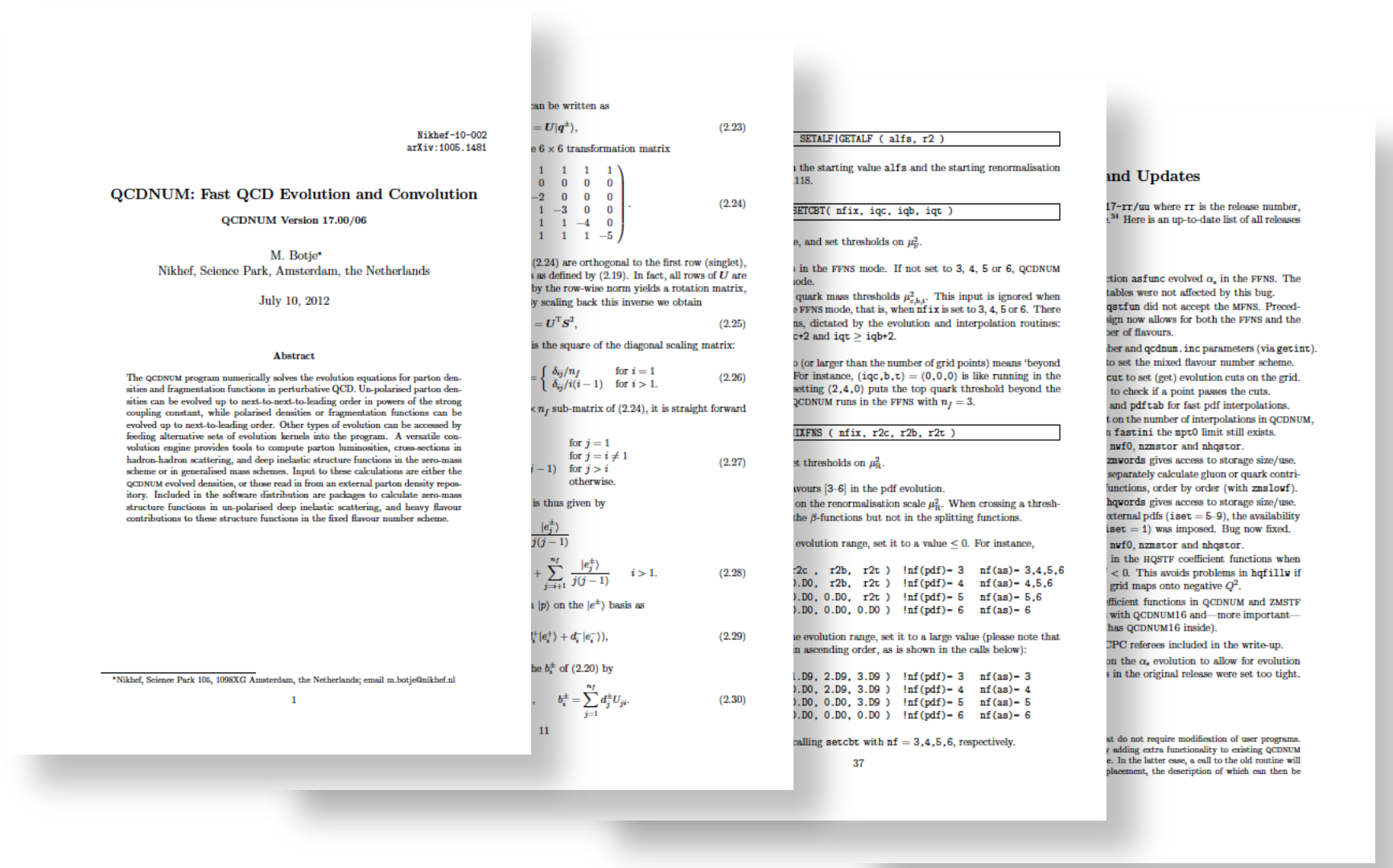
What is QCDNUM?

- Fortran program
- Evolves α_s and the parton densities up to NNLO on a grid in x and Q^2
- Also evolves polarised PDFs and fragmentation functions up to NLO
- Supports fixed and variable flavour number schemes
- Vary renormalisation scale with respect to the factorisation scale
- Light (NNLO) and heavy quark (NLO-FFNS) structure functions
- Convolution toolbox to write your own add-on package
- Enter your own splitting functions for customised evolution
- Fast, accurate and user friendly

QCDNUM has a long history

1988	Code by Ouraou and Virchaux (BCDMS)	CRAY vectorized Fortran
1993	NMC adaptation to low x	CRAY vectorized Fortran
1998	QCDNUM16.12 used by ZEUS	Unix Fortran77
2007	Start NNLO upgrade QCDNUM17	Unix Fortran77
2010	Release QCDNUM17-00	Unix Fortran77

QCDNUM has a manual



Nikhef-10-002
arXiv:1005.1481

QCDNUM: Fast QCD Evolution and Convolution

QCDNUM Version 17.00/06

M. Botje*
Nikhef, Science Park, Amsterdam, the Netherlands

July 10, 2012

Abstract

The QCDNUM program numerically solves the evolution equations for parton densities and fragmentation functions in perturbative QCD. Un-polarised parton densities can be evolved up to next-to-next-to-leading order in powers of the strong coupling constant, while polarised densities or fragmentation functions can be evolved up to next-to-leading order. Other types of evolution can be accessed by feeding alternative sets of evolution kernels into the program. A versatile convolution engine provides tools to compute parton luminosities, cross-sections in hadron-hadron scattering, and deep inelastic structure functions in the zero-mass scheme or in generalised mass schemes. Input to these calculations are either the QCDNUM evolved densities, or those read in from an external parton density repository. Included in the software distribution are packages to calculate zero-mass structure functions in un-polarised deep inelastic scattering, and heavy flavour contributions to these structure functions in the fixed flavour number scheme.

*Nikhef, Science Park 105, 1098XG Amsterdam, the Netherlands; email m.botje@nikhef.nl

1

can be written as

$$= \boldsymbol{U}|q^{\pm}\rangle, \qquad (2.23)$$

e 6×6 transformation matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 \\ 1 & -3 & 0 & 0 \\ 1 & 1 & -4 & 0 \\ 1 & 1 & 1 & -5 \end{pmatrix}. \qquad (2.24)$$

(2.24) are orthogonal to the first row (singlet),
s as defined by (2.19). In fact, all rows of $\boldsymbol{U}$ are
by the row-wise norm yields a rotation matrix,
y scaling back this inverse we obtain

$$= \boldsymbol{U}^{\mathrm{T}}\boldsymbol{S}^2, \qquad (2.25)$$

is the square of the diagonal scaling matrix:

$$= \begin{cases} \delta_{ij}/n_f & \text{for } i = 1 \\ \delta_{ij}/i(i-1) & \text{for } i > 1. \end{cases} \qquad (2.26)$$

$\times\, n_f$ sub-matrix of (2.24), it is straight forward

$$\begin{array}{ll} & \text{for } j = 1 \\ & \text{for } j = i \neq 1 \\ j-1) & \text{for } j > i \\ & \text{otherwise.} \end{array} \qquad (2.27)$$

is thus given by

$$\frac{|e_j^{\pm}\rangle}{j(j-1)}$$

$$+ \sum_{j=i+1}^{n_f} \frac{|e_j^{\pm}\rangle}{j(j-1)} \qquad i > 1. \qquad (2.28)$$

$|p\rangle$ on the $|e^{\pm}\rangle$ basis as

$$|e_i^{\pm}\rangle + d_i^{-}|e_i^{-}\rangle), \qquad (2.29)$$

he $b_i^{\pm}$ of (2.20) by

$$b_i^{\pm} = \sum_{j=1}^{n_f} d_j^{\pm} U_{ji}. \qquad (2.30)$$

11

```
SETALF|GETALF ( alfs, r2 )
```

the starting value `alfs` and the starting renormalisation
118.

```
SETCBT( nfix, iqc, iqb, iqt )
```

e, and set thresholds on μ_{F}^2.

in the FFNS mode. If not set to 3, 4, 5 or 6, QCDNUM
ode.
quark mass thresholds $\mu_{c,b,t}^2$. This input is ignored when
e FFNS mode, that is, when `nfix` is set to 3, 4, 5 or 6. There
ns, dictated by the evolution and interpolation routines:
`c+2` and `iqt` $\geq$ `iqb+2`.

o (or larger than the number of grid points) means 'beyond
For instance, `(iqc,b,t)` = `(0,0,0)` is like running in the
setting `(2,4,0)` puts the top quark threshold beyond the
QCDNUM runs in the FFNS with $n_f = 3$.

```
IXFNS ( nfix, r2c, r2b, r2t )
```

t thresholds on μ_{R}^2.

avours [3–6] in the pdf evolution.
on the renormalisation scale μ_{R}^2. When crossing a thresh-
the β-functions but not in the splitting functions.

evolution range, set it to a value ≤ 0. For instance,

```
:2c ,  r2b,  r2t )  !nf(pdf)= 3   nf(as)= 3,4,5,6
).D0,  r2b,  r2t )  !nf(pdf)= 4   nf(as)= 4,5,6
).D0, 0.D0,  r2t )  !nf(pdf)= 5   nf(as)= 5,6
).D0, 0.D0, 0.D0 )  !nf(pdf)= 6   nf(as)= 6
```

e evolution range, set it to a large value (please note that
in ascending order, as is shown in the calls below):

```
.D9, 2.D9, 3.D9 )  !nf(pdf)= 3   nf(as)= 3
).D0, 2.D9, 3.D9 )  !nf(pdf)= 4   nf(as)= 4
).D0, 0.D0, 3.D9 )  !nf(pdf)= 5   nf(as)= 5
).D0, 0.D0, 0.D0 )  !nf(pdf)= 6   nf(as)= 6
```

calling `setcbt` with `nf` = 3,4,5,6, respectively.

37

and Updates

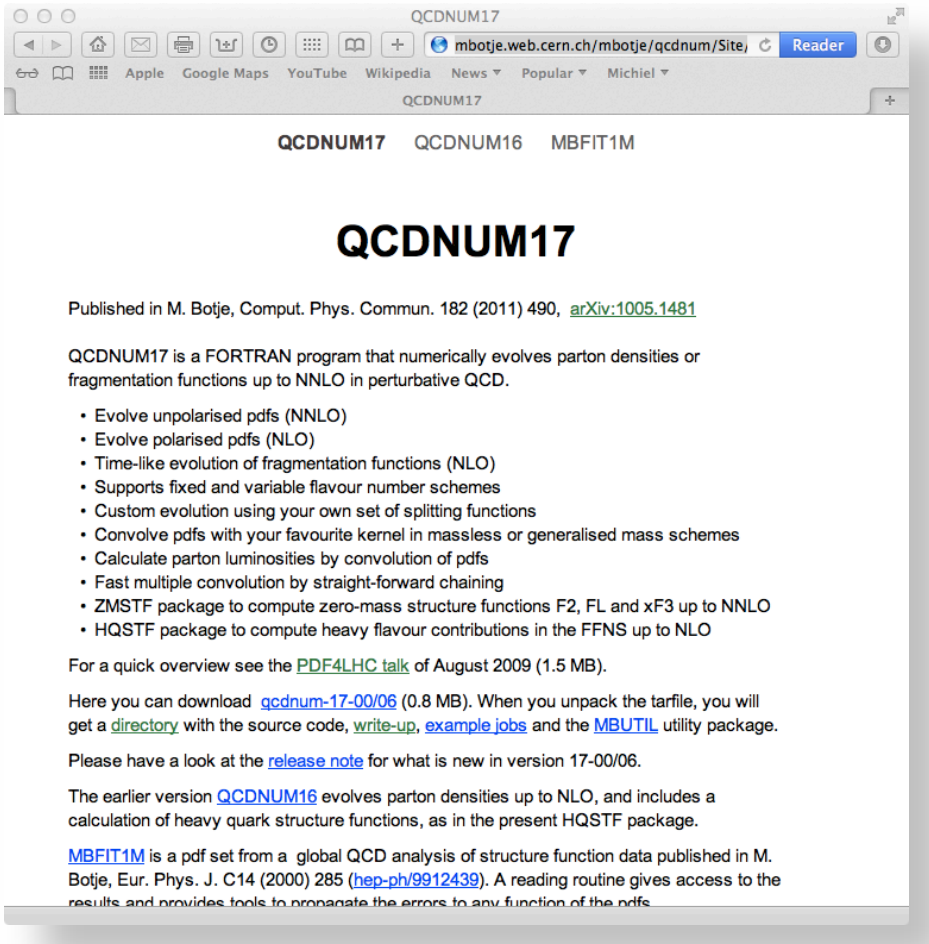
17-rr/uu where rr is the release number,
[34] Here is an up-to-date list of all releases

ction `asfunc` evolved α_s in the FFNS. The
tables were not affected by this bug.
`qstfun` did not accept the MFNS. Preced-
sign now allows for both the FFNS and the
er of flavours.
ber and `qcdnum.inc` parameters (via `getint`).
to set the mixed flavour number scheme.
`cut` to set (get) evolution cuts on the grid.
to check if a point passes the cuts.
and `pdftab` for fast pdf interpolations.
t on the number of interpolations in QCDNUM,
n `fastini` the `mpt0` limit still exists.
`nwf0`, `nzmstor` and `nhqstor`.
`zmwords` gives access to storage size/use.
separately calculate gluon or quark contri-
functions, order by order (with `zmslowf`).
`hqwords` gives access to storage size/use.
xternal pdfs (`iset` = 5–9), the availability
`iset` = 1) was imposed. Bug now fixed.
`nwf0`, `nzmstor` and `nhqstor`.
in the HQSTF coefficient functions when
< 0. This avoids problems in `hqfillw` if
grid maps onto negative Q^2.
fficient functions in QCDNUM and ZMSTF
with QCDNUM16 and—more important—
has QCDNUM16 inside).
CPC referees included in the write-up.
on the α_s evolution to allow for evolution
s in the original release were set too tight.

at do not require modification of user programs.
adding extra functionality to existing QCDNUM
e. In the latter case, a call to the old routine will
placement, the description of which can then be

QCDNUM has a web page

http://www.nikhef.nl/~h24/qcdnum



QCDNUM has presentation slides

- QCD evolution
- Numerical method
- QCDNUM program
- Structure functions
- Convolution engine

1

QCD evolution of α_s and parton densities

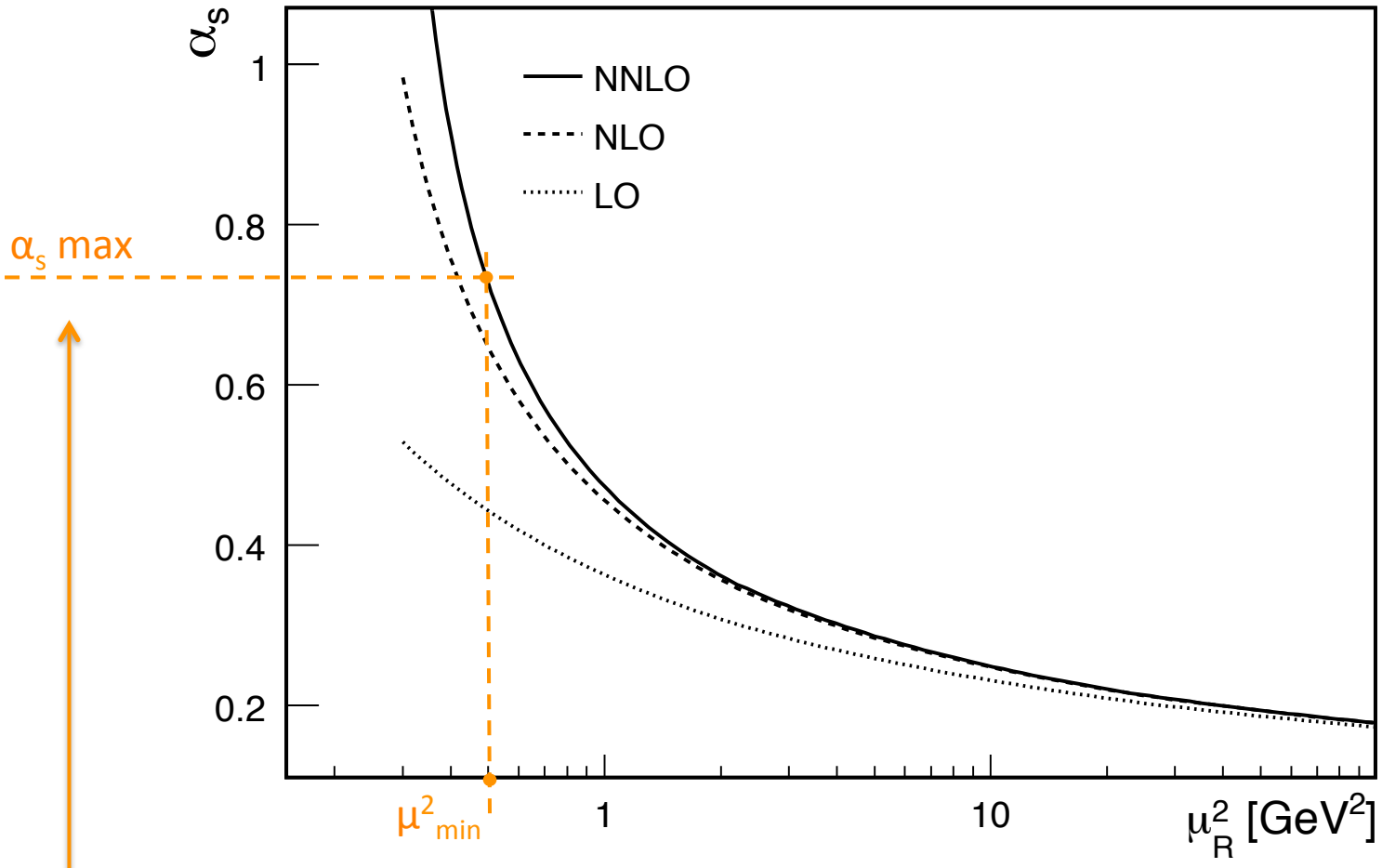
The strong coupling constant α_s evolves on the renormalisation scale μ_R^2

$$\frac{\mathrm{d}a_\mathrm{s}(\mu^2)}{\mathrm{d}\ln\mu^2} = -\sum_{i=0}^{2}\beta_i\, a_\mathrm{s}^{i+2}(\mu^2) \qquad a_\mathrm{s} \equiv \alpha_\mathrm{s}/2\pi$$

- Solved in QCDNUM by Runge-Kutta integration in standalone routine

$$\begin{aligned}\beta_0 &= \tfrac{11}{2} - \tfrac{1}{3}\,n_f \\ \beta_1 &= \tfrac{51}{2} - \tfrac{19}{6}\,n_f \\ \beta_2 &= \tfrac{2857}{16} - \tfrac{5033}{144}\,n_f + \tfrac{325}{432}\,n_f^2\end{aligned}$$

- Input: $\alpha_s(\mu_0^2)$



- Limit α_s value in QCDNUM to avoid blow-up at low μ^2
- At larger order the limit will correspond to larger μ^2_{min}
- If α_s exceeded: increase lower edge of μ^2 grid or set cuts

DGLAP evolution of PDFs

$$\frac{\partial f_i(x,\mu^2)}{\partial \ln \mu^2} = \sum_{j=q,\bar{q},g} [P_{ij} \otimes f_j](x,\mu^2)$$

$$i = \begin{cases} 1\ ,\ldots,\ n_f & \text{quarks} \\ 0 & \text{gluon} \\ -1\ ,\ldots,-n_f & \text{antiquarks} \end{cases}$$

$$[P \otimes f](x) = \int_x^1 \frac{\mathrm{d}z}{z} P\left(\frac{x}{z}\right) f(z)$$

- These are $2n_f+1$ coupled integro-differential equations
- Define singlet/non-singlet PDFs to decouple equations
- PDFs evolve on the factorisation scale μ_F^2
- DGLAP can be solved when $f(x,\mu_0^2)$ is given

- **Singlet/gluon evolution**

$$q_{\rm s} = \sum_{i=1}^{n_f} (q_i + \bar{q}_i)$$

$$\frac{\partial}{\partial \ln \mu^2} \begin{pmatrix} q_{\rm s} \\ g \end{pmatrix} = \begin{pmatrix} P_{\rm qq} & P_{\rm qg} \\ P_{\rm gq} & P_{\rm gg} \end{pmatrix} \otimes \begin{pmatrix} q_{\rm s} \\ g \end{pmatrix}$$

- **Non-singlet evolution**

$$q_{ij}^{\pm} = (q_i \pm \bar{q}_i) - (q_j \pm \bar{q}_j)$$

$$q_{\rm v} = \sum_{i=1}^{n_f} (q_i - \bar{q}_i)$$

$$\frac{\partial q_{\rm ns}}{\partial \ln \mu^2} = P_{\rm ns} \otimes q_{\rm ns}$$

QCDNUM internally uses a standard set of singlet/non-singlet basis functions

singlet

nonsinglets

$$\begin{pmatrix} e_1^{\pm} \\ e_2^{\pm} \\ e_3^{\pm} \\ e_4^{\pm} \\ e_5^{\pm} \\ e_6^{\pm} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & & & & \\ 1 & 1 & -2 & & & \\ 1 & 1 & 1 & -3 & & \\ 1 & 1 & 1 & 1 & -4 & \\ 1 & 1 & 1 & 1 & 1 & -5 \end{pmatrix} \begin{pmatrix} d^{\pm} \\ u^{\pm} \\ s^{\pm} \\ c^{\pm} \\ b^{\pm} \\ t^{\pm} \end{pmatrix}$$

$$q_i^{\pm} = q_i \pm \bar{q}_i$$

Perturbative expansion of evolution kernels

$$\mathrm{N}^{\ell}\mathrm{LO}: \qquad P_{\mathrm{ab}}(x,\mu^2) = \sum_{n=0}^{\ell} \alpha_{\mathrm{s}}^{n+1}(\mu^2)\; P_{\mathrm{ab}}^{(n)}(x)$$

- Factorisation in x and μ^2
- Nonsinglet evolution proliferates at increasing order LO, NLO, NNLO

	LO	NLO	NNLO
q_{ij}^{+}	$P_{\mathrm{qq}}^{(0)}$	$P_{+}^{(1)}$	$P_{+}^{(2)}$
q_{ij}^{-}	$P_{\mathrm{qq}}^{(0)}$	$P_{-}^{(1)}$	$P_{-}^{(2)}$
q_{v}	$P_{\mathrm{qq}}^{(0)}$	$P_{-}^{(1)}$	$P_{\mathrm{v}}^{(2)}$

Renormalisation scale dependence

- Can set $\mu_R^2 = a\mu_F^2 + b$ in QCDNUM
- Powers of $\alpha_s(\mu_F^2)$ calculated by Taylor expansion

$$
\begin{aligned}
a_{\rm s}(\mu_{\rm F}^2) &= a_{\rm s}(\mu_{\rm R}^2) - \beta_0 L_{\rm R}\, a_{\rm s}^2(\mu_{\rm R}^2) - (\beta_1 L_{\rm R} - \beta_0^2 L_{\rm R}^2)\, a_{\rm s}^3(\mu_{\rm R}^2) + {\rm O}(a_{\rm s}^4) \\
a_{\rm s}^2(\mu_{\rm F}^2) &= a_{\rm s}^2(\mu_{\rm R}^2) - 2\beta_0 L_{\rm R}\, a_{\rm s}^3(\mu_{\rm R}^2) + {\rm O}(a_{\rm s}^4) \\
a_{\rm s}^3(\mu_{\rm F}^2) &= a_{\rm s}^3(\mu_{\rm R}^2) + {\rm O}(a_{\rm s}^4) \qquad\qquad L_{\rm R} = \ln(\mu_{\rm F}^2/\mu_{\rm R}^2)
\end{aligned}
$$

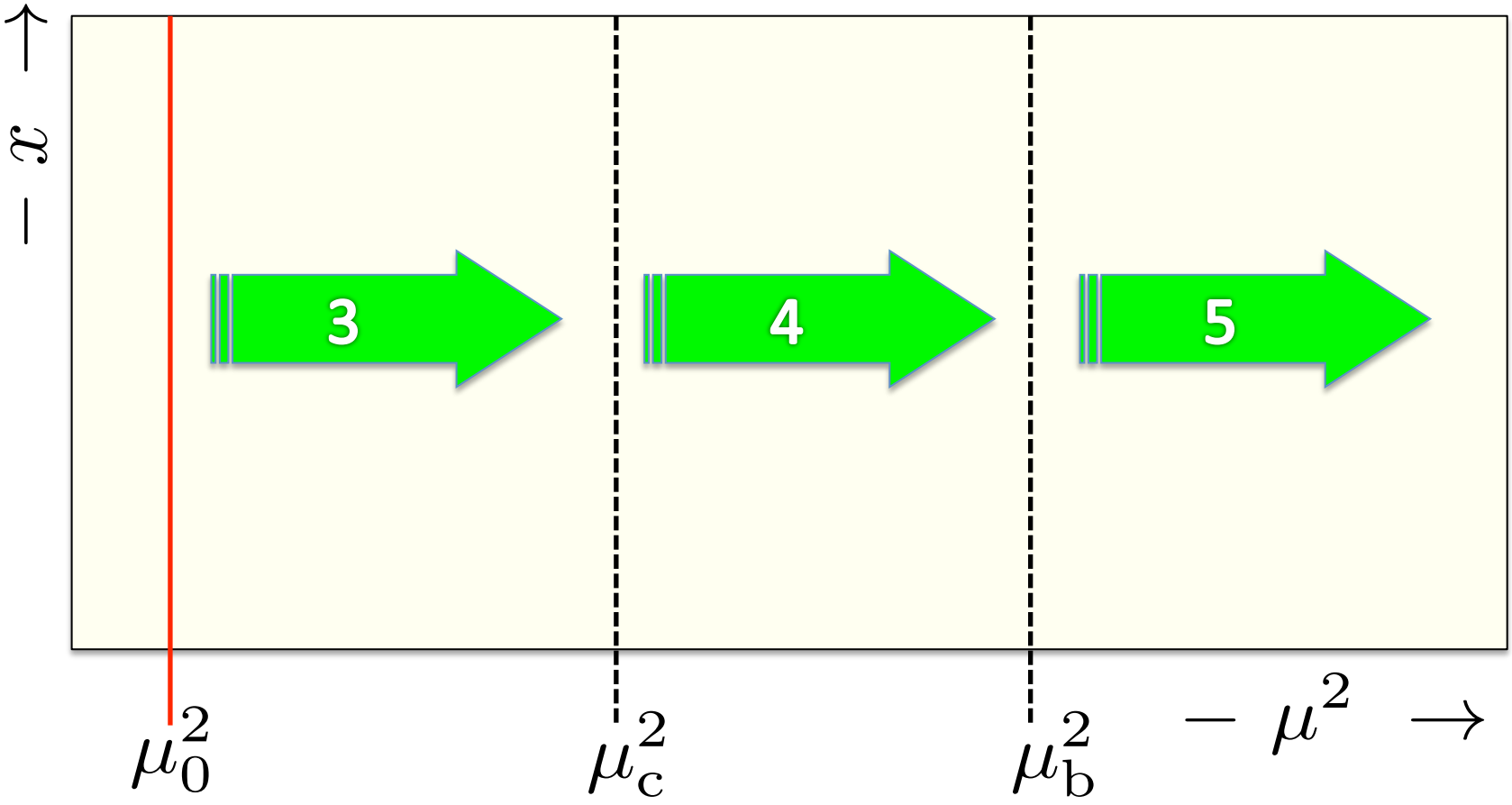
- Truncate expansion to (α_s, α_s^2, α_s^3) at (LO,NLO,NNLO) and use the truncated $\alpha_s^n(\mu_F^2)$ in DGLAP
- QCDNUM maintains lookup tables of these α_s^n

Evolution schemes in QCDNUM

- FFNS: number of active flavours is kept constant $3 < n_f < 6$ for all μ^2
- VFNS: number of flavours changes from n_f to n_f+1 at the thresholds $\mu^2_{c,b,t}$
  - Input: gluon + 6 light quark PDFs at $\mu^2_0 < \mu^2_c$
  - Heavy quarks are generated by DGLAP evolution
  - The PDFs and α_s are continuous at the thresholds at LO and NLO but are discontinuous at NNLO

K.G. Chetyrkin et al.. PRL 79 (1997) 2184
M.Buza et al., EPJ C1 (1988) 301

Evolution with 3, 4, 5 flavours in the VFNS



In QCDNUM the starting scale must be below μ_c^2

- NNLO PDFs are discontinuous at the thresholds (BMSN)

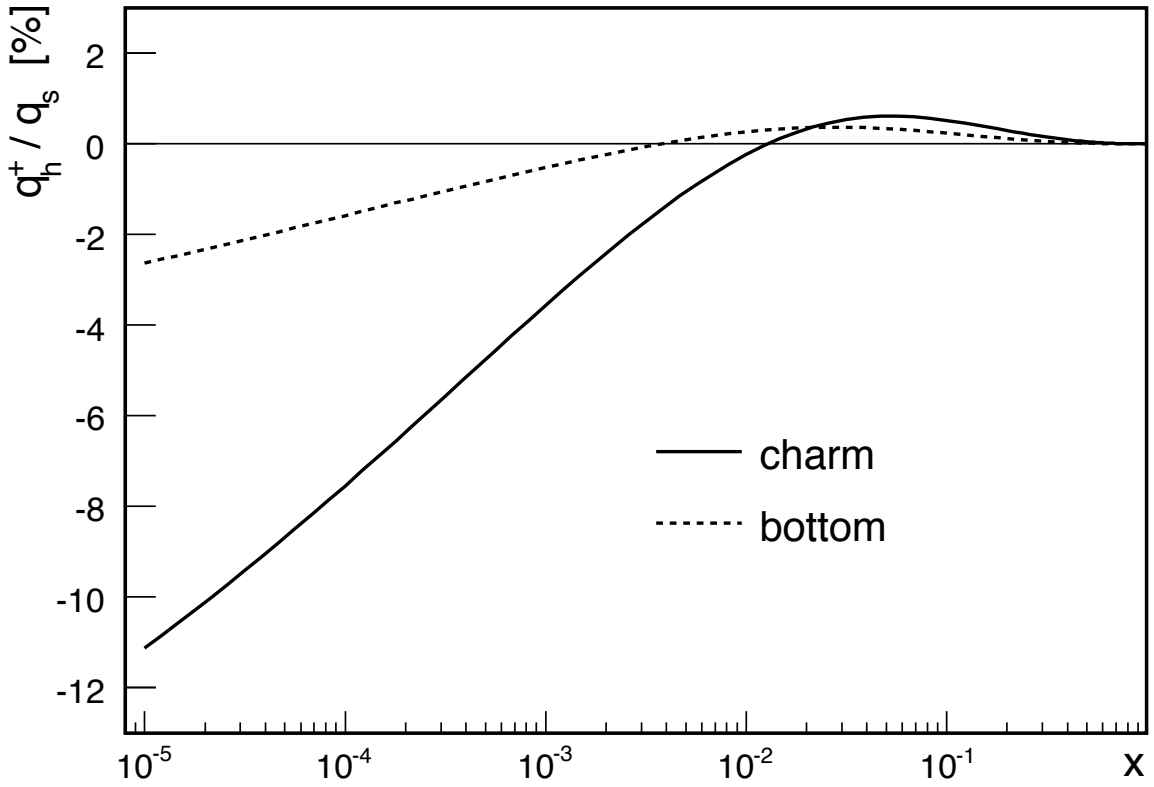
$$\begin{aligned}
\Delta g &= a_s^2[A_{gq} \otimes q_s + A_{gg} \otimes g] \\
\Delta q_i^{\pm} &= a_s^2\ [A_{qq} \otimes q_i^{\pm}] \\
\Delta h^{+} &= a_s^2[A_{hq} \otimes q_s + A_{hg} \otimes g] \\
\Delta h^{-} &= 0
\end{aligned}$$

M.Buza et al., EPJ C1 (1988) 301

- In QCDNUM the value of n_f changes at the same moment in <u>both</u> the α_s and the PDF evolution
- This much simplifies the calculation of discontinuities
- Also we then have <u>no</u> discontinuities at LO or NLO
- QCDNUM PDFs are bi-valued at the thresholds but <u>you</u> will get the value corresponding to the largest flavour
- Thus for you n_f = 4 not 3 at μ^2_c and 5 not 4 at μ^2_b

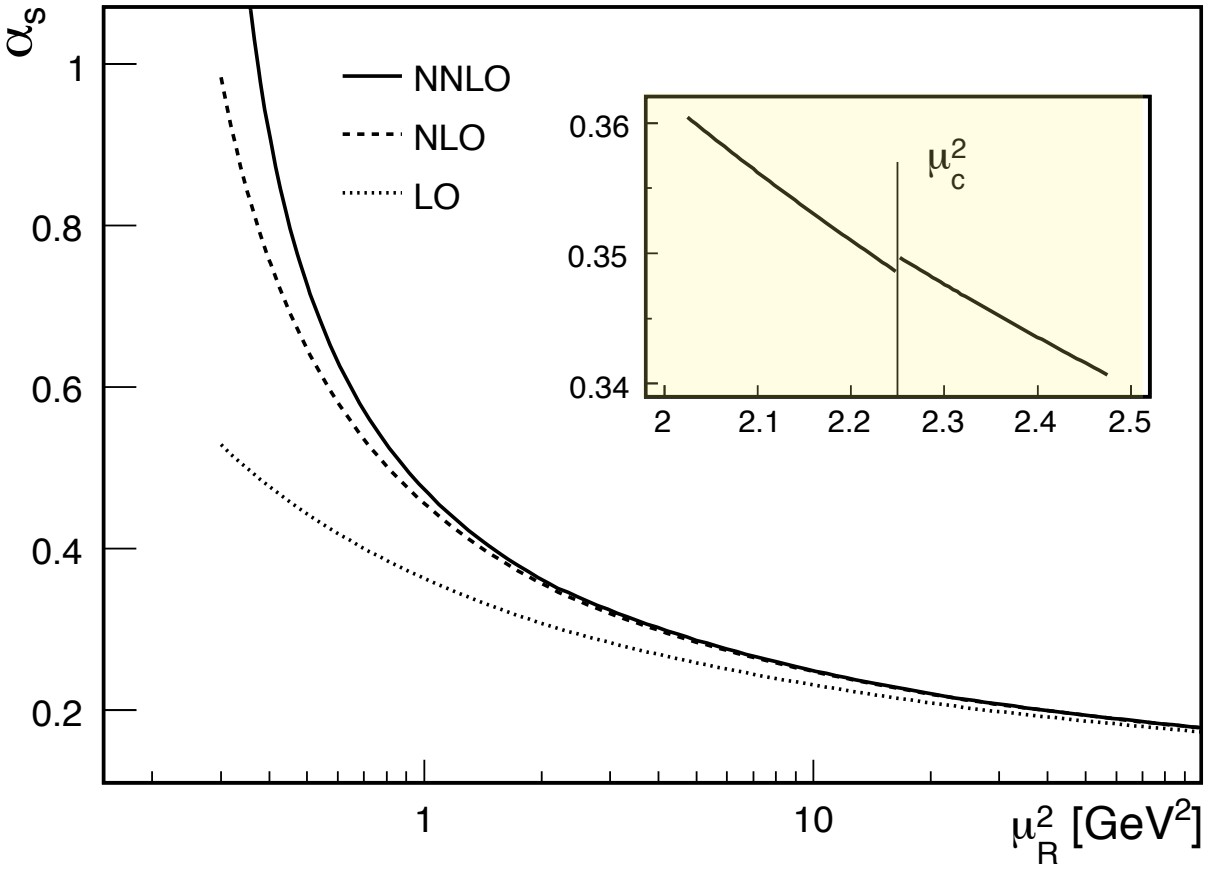
Example: NNLO discontinuity of charm and bottom at their thresholds

- Bottom ~3% of singlet at low x
- Charm ~10%
- PDFs negative below $x = 10^{-2}$
- No problem since PDFs are not observables



There are also NNLO discontinuities in the α_s evolution but these are small

K.G. Chetyrkin et al.. PRL 79 (1997) 2184



2 Numerical method in a nutshell

- QCDNUM works on an equidistant grid in y, t

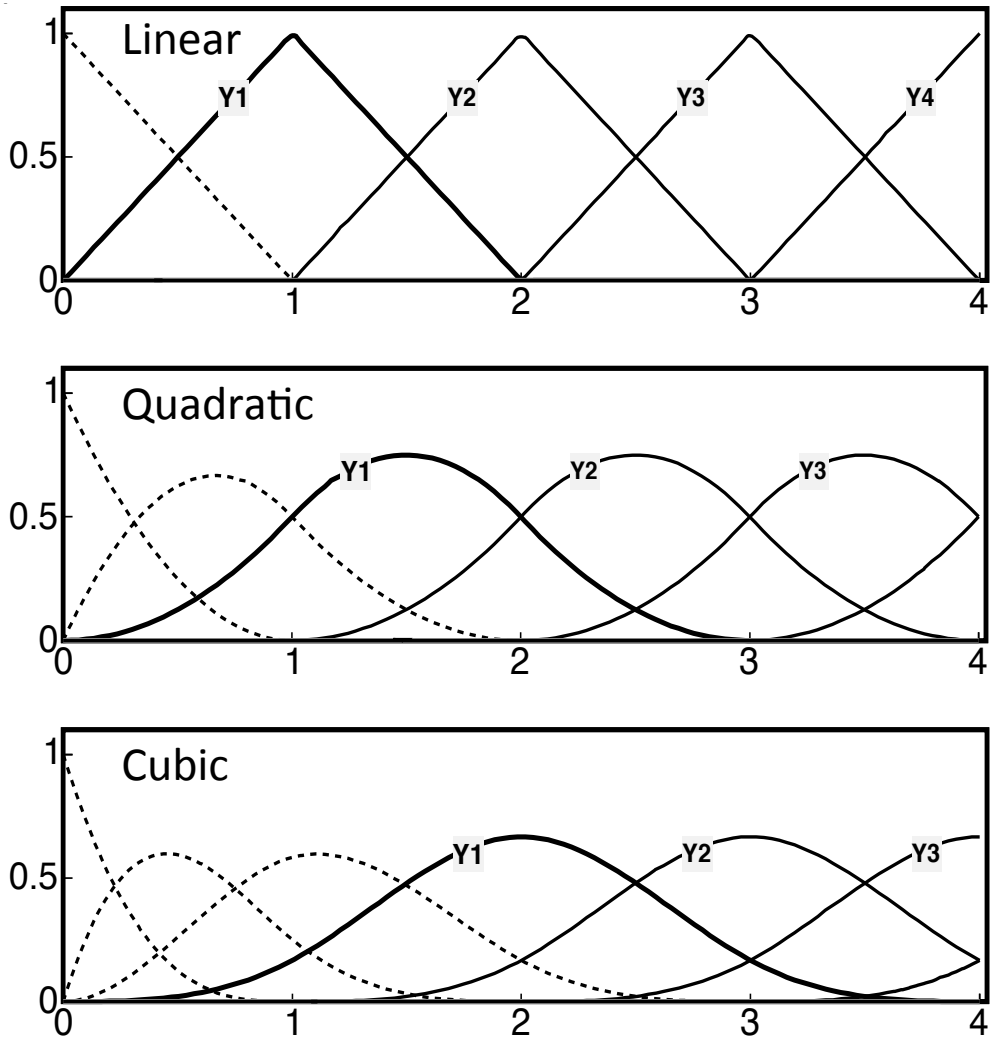
$$\begin{aligned} \mu^2 &\rightarrow t = \ln \mu^2 \\ x &\rightarrow y = -\ln x \\ xf(x) &\rightarrow h(y) \\ xP_{ij}(x) &\rightarrow Q_{ij}(y) \end{aligned}$$

- Mellin convolution then becomes Fourier convolution and the nonsinglet DGLAP reads

$$\frac{\partial h(y,t)}{\partial t} = \int_0^y \mathrm{d}z\, Q(z,t)\, h(y-z,t) = [Q \otimes h](y,t)$$

B-spline interpolation

$$h(y) = \sum_i a_i Y_i(y)$$



- Boundary condition
  h(0) = h'(0) = 0
- Remaining B-splines translation invariant
- Translation invariance much simplifies the evolution algorithm
- Does not work for cubic splines because numerically instable

- Very simple relation between $h(y_i)$ and spline coefficients a_i both for linear and quad

$$\boldsymbol{h} = \boldsymbol{S}\ \boldsymbol{a}$$

$$\boldsymbol{S}^{(1)} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \qquad \boldsymbol{S}^{(2)} = \frac{1}{2}\begin{pmatrix} 1 & & & \\ 1 & 1 & & \\ & 1 & 1 & \\ & & 1 & 1 \end{pmatrix}$$

- QCDNUM has very fast and simple routines to solve triangular bandmatrix equations

- Convolution integral becomes weighted sum

$$[Q \otimes h](y_i) = \sum_{j=1}^{i} W_{ij} a_j = \boldsymbol{W}\boldsymbol{a}$$

- Lower triangular Toeplitz weight matrices

$$W_{ij} = \begin{pmatrix} w_1 & & & \\ w_2 & w_1 & & \\ w_3 & w_2 & w_1 & \\ w_4 & w_3 & w_2 & w_1 \end{pmatrix}$$

- Weights are computed at program initialisation

$$w_i = \int_0^{y_i} \mathrm{d}z \; Q(y_i - z) Y_1(z)$$

- Vector storage of weight matrices gives very fast summation of the perturbative series inside evolution routine

$$\boldsymbol{W}(t) = a_s(t)\{ \boldsymbol{W}^{(0)} + a_s(t)\boldsymbol{W}^{(1)} + \cdots \}$$

- The DGLAP evolution at t_0 can now be written as

$$\frac{\mathrm{d}\boldsymbol{h_0}}{\mathrm{d}t} = \frac{\mathrm{d}\boldsymbol{S a_0}}{\mathrm{d}t} = \boldsymbol{W_0}\,\boldsymbol{a_0} \quad \rightarrow \quad \boldsymbol{a'_0} = \boldsymbol{S}^{-1}\boldsymbol{W_0 a_0}$$

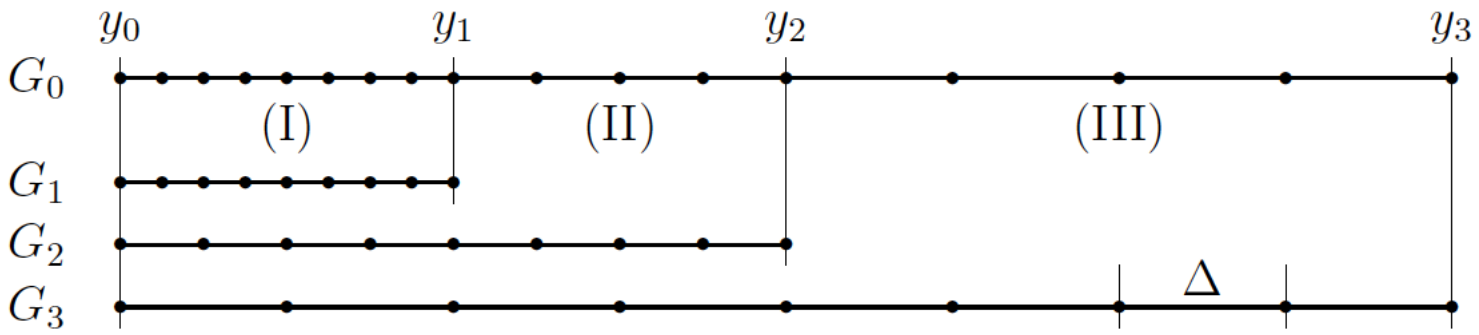
- DGLAP + quad interpolation in t gives a relation between known spline coefficients $a(t_0)$ and the unknown coefficients $a(t_1)$

$$\left.\begin{aligned} \boldsymbol{a'_0} &= \boldsymbol{S}^{-1}\boldsymbol{W_0 a_0} \\ \boldsymbol{a'_1} &= \boldsymbol{S}^{-1}\boldsymbol{W_1 a_1} \\ \boldsymbol{a_1} &= \boldsymbol{a_0} + \tfrac{1}{2}(\boldsymbol{a'_0} + \boldsymbol{a'_1})\Delta t \end{aligned}\right\} \rightarrow \begin{aligned} &(2\boldsymbol{S} - \boldsymbol{W_1}\Delta t)\boldsymbol{a_1} = \\ &(2\boldsymbol{S} + \boldsymbol{W_0}\Delta t)\boldsymbol{a_0} \end{aligned}$$

- This leads to a lower triangular Toeplitz matrix equation that can be solved very fast for a_1

Multiple grids

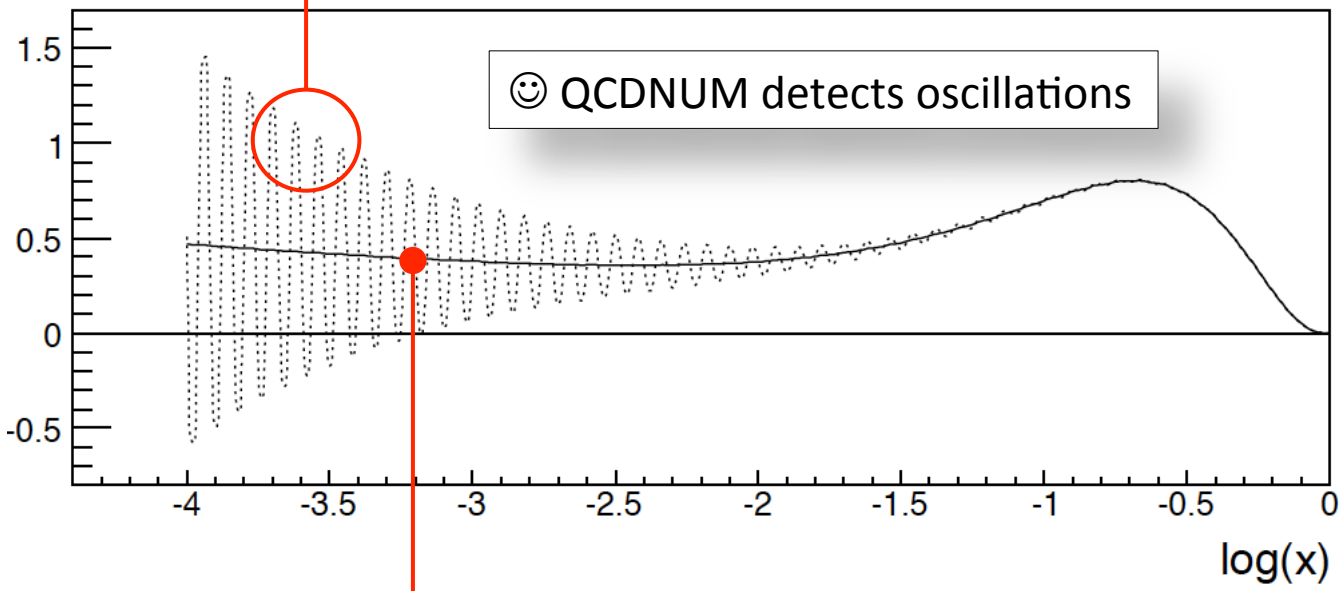
- Equidistant grid over a large range in y usually is adequate at large y (small x) but too coarse at low y (large x)
- QCDNUM uses multiple grids with larger density at low y



- Evolve with grids G_1, G_2, and G_3 and copy results to the regions I, II and III of G_0
- Quad splines + multiple grids give huge gains in accuracy

QCDNUM caveat: backward evolution

Backward evolution with quadratic splines may oscillate
(forward evolution never oscillates nor do linear splines)



QCDNUM can handle this instability but quad backward evolution over a large range in μ^2 is not recommended

3

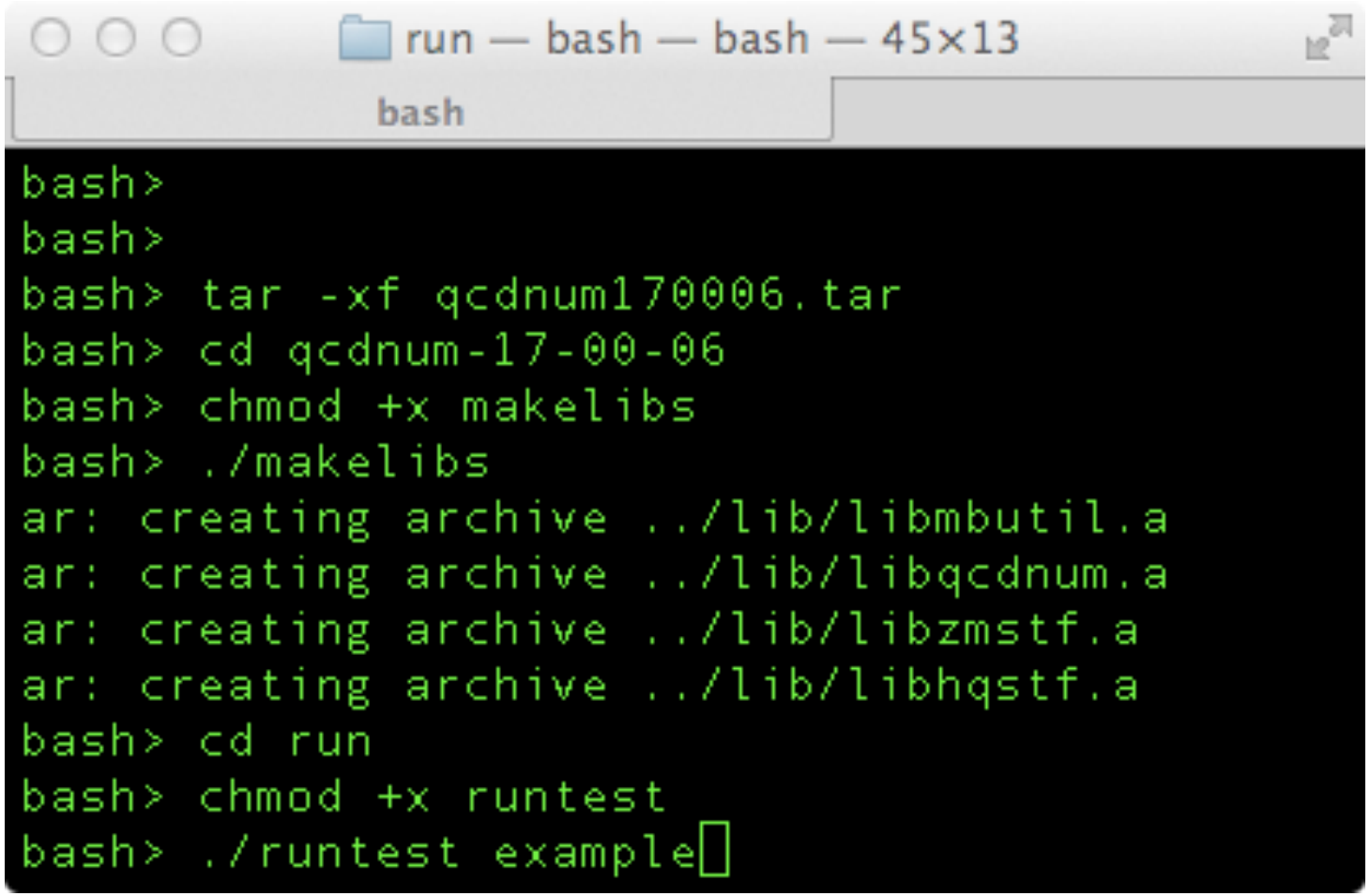
The QCDNUM program

QCDNUM-17.00/06 distribution

- MBUTIL pool of Fortran utility routines
- QCDNUM evolution program
  - PDF evolution fully NNLO
  - Polarized PDFs and FFs in NLO
  - Convolution engine to build your own add-ons
- ZMSTF zero-mass structure function add-on
  - F_2, F_L, xF_3, and F_L' up to NNLO
- HQSTF heavy quark stfs in 3-flavour FFNS
  - F_2, F_L contribution from (c,b,t) up to NLO

E. Laenen et al.,NP B392 (1993) 162
S.Riemersma et al., PL B347 (1995) 143

Installation is really very easy



```
run — bash — bash — 80×42
bash

                ///                                        .().
               (. .)                                      (--)
  +----------ooO--()--Ooo----------------------------ooO------Ooo---------+
  |                                                                        |
  |    ######      ######    ######     ##    ##   ##    ##   ##     ##  |
  |   ##   ##     ##    ##  ##    ##    ###   ##   ##    ##   ###   ###  |
  |  ##     ##    ##    ##  ##     ##   #### ##   ##    ##   #### #### |
  |  ##     ##    ##        ##     ##  ## ## ##   ##    ##   ## ### ## |
  |  ##     ##    ##        ##     ##  ##  ####   ##    ##   ##  #  ## |
  |   ##   ##     ##    ##  ##    ##   ##   ###   ##    ##   ##     ## |
  |    ######     ######    ######    ##    ##    ######    ##     ## |
  |         ##                                                           |
  |                                                                      |
  |    Version 17-00/06    10-07-12          Author m.botje@nikhef.nl    |
  |                                                                      |
  +----------------------------------------------------------------------+


  +----------------------------------------------------------------------+
  |                                                                      |
  |    If you use QCDNUM, please refer to:                               |
  |                                                                      |
  |    M. Botje, Comput. Phys. Commun. 182(2011)490, arXiV:1005.1481     |
  |                                                                      |
  +----------------------------------------------------------------------+

 FILLWT: start unpolarised weight calculations
 Subgrids    1 Subgrid points  100
 Pij LO    for ospline = 2
 Pij NLO   for ospline = 2
 Pij NNLO  for ospline = 2
 Aij NNLO  for ospline = 2
 Pij LO    for ospline = 3
 Pij NLO   for ospline = 3
 Pij NNLO  for ospline = 3
 Aij NNLO  for ospline = 3
 FILLWT: weight calculations completed

 x, q, CharmSea =  0.10000E-02  0.10000E+04  0.18708E+01
 as(mz2)        =  0.11807E+00
bash>
```

Installing takes a few minutes …

… et voila your first QCDNUM evolution

QCDNUM memory space

- QCDNUM has its own (primitive but adequate) dynamic memory management
- But Fortran is Fortran, and memory space must be declared at compilation time (in an include file)
- If you run out of memory (error message) then increase **`nwf0`** in **`qcdnum.inc`** and recompile
- By default **`nwf0 = 1.2M`** should be large enough

QCDNUM: compact user interface

```
call QCINIT(6,'')
call GXMAKE(xmin,1,1,nxin,nx,iosp)
call GQMAKE(qq,wt,2,nqin,nq)
call FILLWT(0,id1,id2,nwords)
call SETORD(3)
call SETALF(as0,r20)
call SETCBT(0,iqc,iqb,iqt)
call EVOLFG(1,func,def,iq0,eps)
call FPDFXQ(1,x,q,pdf,0)
```

✓ Full NNLO evolution in 9 lines

QCDNUM: grids and weights

```
call QCINIT(6,'')
call GXMAKE(xmin,1,1,nxin,nx,iosp)
call GQMAKE(qq,wt,2,nqin,nq)
call FILLWT(0,id1,id2,nwords)
call SETORD(3)
call SETALF(as0,r20)
call SETCBT(0,iqc,iqb,iqt)
call EVOLFG(1,func,def,iq0,eps)
call FPDFXQ(1,x,q,pd
```

- ✓ Initialise QCDNUM
- ✓ Define x-μ^2 grid and lin/quad interpolation
- ✓ Calculate weight tables

QCDNUM: evolution parameters

```
call QCINIT(6,' ')
call GXMAKE(xmin,1,1,nxin,nx,iosp)
call GQMAKE(qq,wt,2,nqin,nq)
call FILLWT(0,id1,id2,nwords)
call SETORD(3)
call SETALF(as0,r20)
call SETCBT(0,iqc,iqb,iqt)
call EVOLFG(1,func,def,iq0,eps)
call FPDFXQ(1,x,q,pdf,0)
```

- ✓ Set LO, NLO, NNLO
- ✓ Input strong coupling constant
- ✓ FFNS, VFNS, and thresholds $\mu^2_{c,b,t}$

QCDNUM: evolution of all PDFs

```
call QCINIT(6,'')
call GXMAKE(xmin,1,1,nxin,nx,iosp)
call GQMAKE(qq,wt,2,nqin,nq)
call FILLWT(0,id1,id2,nwords)
call SETORD(3)
call SETALF(as0,r20)
call SETCBT(0,iqc,iqb,iqt)
call EVOLFG(1,func,def,iq0,eps)
call FPDFXQ(1,x,q,pdf,0)
```

- ✓ User function **func** provides input PDFs at μ^2_0
- ✓ Array **def** describes flavour composition
- ✓ Several routines to return PDFs at x, μ^2

- The example needs only 6 light quark input PDFs in the VFNS
- The array **def** gives the flavour composition of these PDFs

```
      data def  /
C--   tb  bb  cb  sb  ub  db   g   d   u   s   c   b   t
C--   -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6
     +  0., 0., 0., 0., 0.,-1., 0., 1., 0., 0., 0., 0., 0.,   !dval
     +  0., 0., 0., 0.,-1., 0., 0., 0., 1., 0., 0., 0., 0.,   !uval
     +  0., 0., 0.,-1., 0., 0., 0., 0., 0., 1., 0., 0., 0.,   !sval
     +  0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,   !dbar
     +  0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,   !ubar
     +  0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,   !sbar
     +  78*0.    /
```

- User function **func** is a switchyard

```
C     ------------------------------------------------------------
      double precision function func(id,x)     !momentum density xf(x)
C     ------------------------------------------------------------
      implicit double precision (a-h,o-z)
      if(id.eq.0) func = gluon(x)               !0 = always gluon
      if(id.eq.1) func = dvalence(x)            !1 = defined in def
      ..                                        ..
      if(id.eq.6) func = strangebar(x)          !6 = defined in def
      return
      end
```

- QCDNUM cannot impose the sum rules

$$\begin{aligned}
\int_0^1 xg(x)\mathrm{d}x + \int_0^1 xq_{\mathrm{s}}(x)\mathrm{d}x &= 1 \\
\int_0^1 [d(x) - \bar{d}(x)]\mathrm{d}x &= 1 \\
\int_0^1 [u(x) - \bar{u}(x)]\mathrm{d}x &= 2
\end{aligned}$$

- You should build them into the input PDFs
- Once imposed at μ^2_0 then DGLAP guarantees that they remain valid at all μ^2

Weight tables

`FILLWT ( itype, *idmi, *idma, *nw )`	Fill weight tables
`FILLWC ( mysub, *idmi, *idma, *nw )`	Custom weights
`DMPWGT ( itype, lun, 'filename' )`	Dump weight tables
`READWT ( lun, 'fn', *idmi, *idma, *nw, *ie )`	Read weight tables

- QCDNUM has routines to create weight tables (takes a few minutes), dump these on disk and read them back
- Tables depend on the grid definition but otherwise can handle all number of flavours and all orders LO, NLO, NNLO
- Different sets for unpolarised PDFs, polarised PDFs, FF

QCDNUM can have different PDF sets in memory

```
call FILLWT ( iset, idmin, idmax, nwds )
call FILLWC ( subr, idmin, idmax, nwds )
call PDFINP ( subr, iset, offset, epsi, nwds )
```

- Standard weights with **FILLWT**
  - 1 = Unpolarised PDFs
  - 2 = Polarised PDFs
  - 3 = Fragmentation functions
- Custom evolution weights with **FILLWC**
  - 4 = Custom PDFs
- External PDFs with **PDFINP**
  - 5-9 = PDFs from external PDF set

Mixed Flavour Number Scheme

- QCDNUM supports a scheme with flavour thresholds in the α_s evolution but without flavour thresholds in the PDF evolution
- Thus, α_s evolves in the VFNS and the PDFs in the FFNS
- MFNS is rarely used but there exist structure function calculations that need MFNS PDFs

```
call MIXFNS ( nfix, r2c, r2b, r2t )
```

Cuts on the evolution grid

```
Call SETCUT ( xmi, q2mi, q2ma, dummy )
```

- Evolve only on part of the grid
- Allows to restrict the evolution range to that of the data in the χ^2 loop of a QCD fit
- Once the fit has converged you can open the cuts and evolve (only once) over the full grid
- Can be quite a CPU time saver!

Access to PDFs

`FVALXQ|IJ ( iset, id, x|ix, qmu2|iq, ichk )`	Interpolate $\|g, q, \bar{q}\rangle$
`FPDFXQ|IJ ( iset, x|ix, qmu2|iq, *pdfs, ichk )`	All pdfs $\|g, q, \bar{q}\rangle$
`FSUMXQ|IJ ( iset, def, x|ix, qmu2|iq, ichk )`	Linear combination
`FSNSXQ|IJ ( iset, id, x|ix, qmu2|iq, ichk )`	Interpolate $\|g, e^{\pm}\rangle$
`SPLCHK ( iset, id, iq )`	Check spline
`FSPLNE ( iset, id, x, iq )`	Spline interpolation
`PDFLST ( iset, def, x, qmu2, *pdf, n, ichk )`	Fast pdf interpolation
`PDFTAB ( iset, def, x, nx, q, nq, *f, ichk )`	Fast pdf interpolation

- QCDNUM offers many routines to interpolate the PDFs that reside in the internal tables
- **SPLCHK** and **FSPLNE** can be used to investigate spline oscillations (which QCDNUM will autodetect if there are any)
- **PDFLST** and **PDFTAB** are for the fast creation of PDF tables

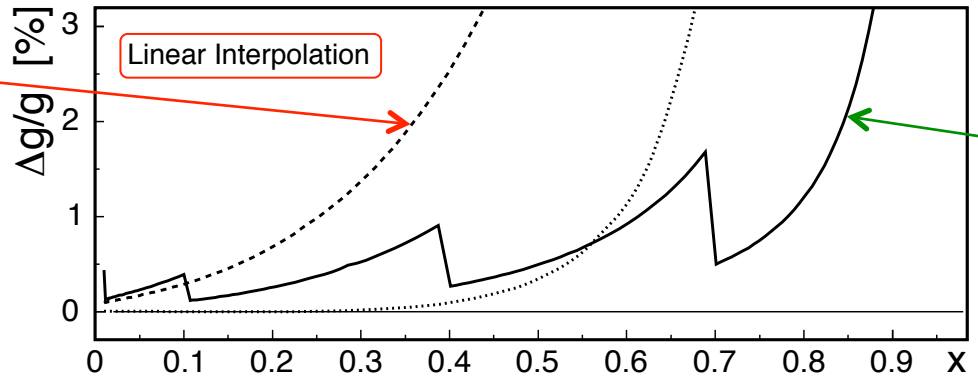
A.Vogt, CPC 170 (2005) 65

QCDNUM-PEGASUS comparison

NLO gluon evolution from $\mu^2 = 2$ to 10^4 GeV^2

Single 200 point grid

Linear

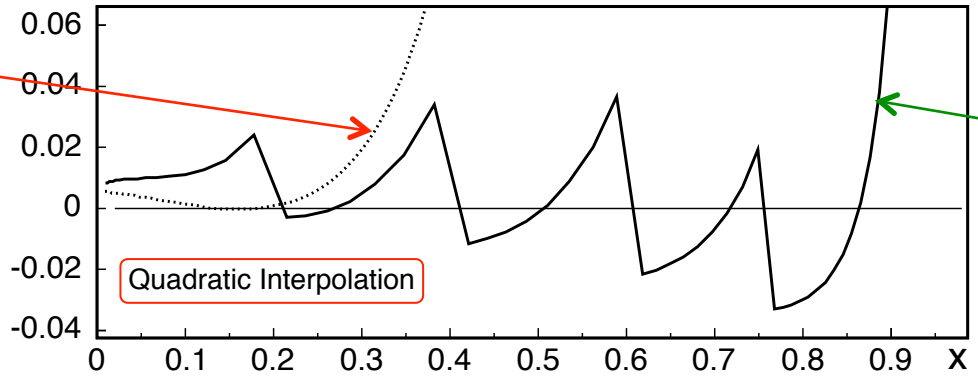


5-fold 200 point grid

Linear

Single 100 point grid

Quadratic



5-fold 100 point grid

Quadratic

✓ Comparison QCDNUM – PEGASUS O(10^{-4})

Recommended multiple grid

- The choice of grid is very important because it determines speed and accuracy
- For a grid down to $x = 10^{-5}$ and up to $\mu^2 = 10^4$ GeV^2 use a 5-fold grid with quad interpolation

	n	x_1	x_2	x_3	x_4	x_5
Linear interpolation	200	10^{-5}	0.01	0.10	0.40	0.70
Quadratic interpolation	100	10^{-5}	0.20	0.40	0.60	0.75
Relative point density		1	2	4	8	16

- You can add (subtract) 12 points for each decade lower (higher) in x

Fast?

- Use 100x50 point 5-fold grid $x > 10^{-5}$ and $\mu^2 < 10^4$ GeV^2
- Do 1000 NNLO evolutions of 11 PDFs (no top) in the VFNS
- For each evolution do 1000 NNLO F_2 and F_L in HERA kin range
- Code compiled with gfortran (w/o array boundary check)
- MacBook 2GHz Intel core 2 Duo

Routine	Calls	CPU sec	CPU/call
Evolution	1000	8.5	0.009
F_2, F_L	2.10^6	10.0	5.10^{-6}

✓ Takes 18.5 sec which is pretty fast!

4 Structure Functions

The ZMSTF and HQSTF add-ons

- The calculation of structure functions and cross-sections is not a part of QCDNUM
- Structure functions are Mellin convolutions of the PDFs with coefficient functions so they can be calculated very fast with the QCDNUM convolution engine
- You can wrap your calculations in an add-on package with a nicely documented user interface
- The QCDNUM release comes with two add-on packages ZMSTF (zero mass stfs) and HQSTF (heavy quark stfs)

Zero mass structure functions

- Convolution of PDFs and Wilson coefficients

$$\frac{1}{x}\mathcal{F}_i^{(\mathrm{s})}(x,Q^2) = [C_{i,\mathrm{s}} \otimes q_\mathrm{s}](x,\mu^2) + [C_{i,\mathrm{g}} \otimes g](x,\mu^2) \qquad i = 2, \mathrm{L}$$

$$\frac{1}{x}\mathcal{F}_i^{(\mathrm{ns})}(x,Q^2) = [C_{i,\mathrm{ns}} \otimes q_\mathrm{ns}](x,\mu^2) \qquad i = 2, \mathrm{L}, 3$$

- Perturbative expansion in α_s

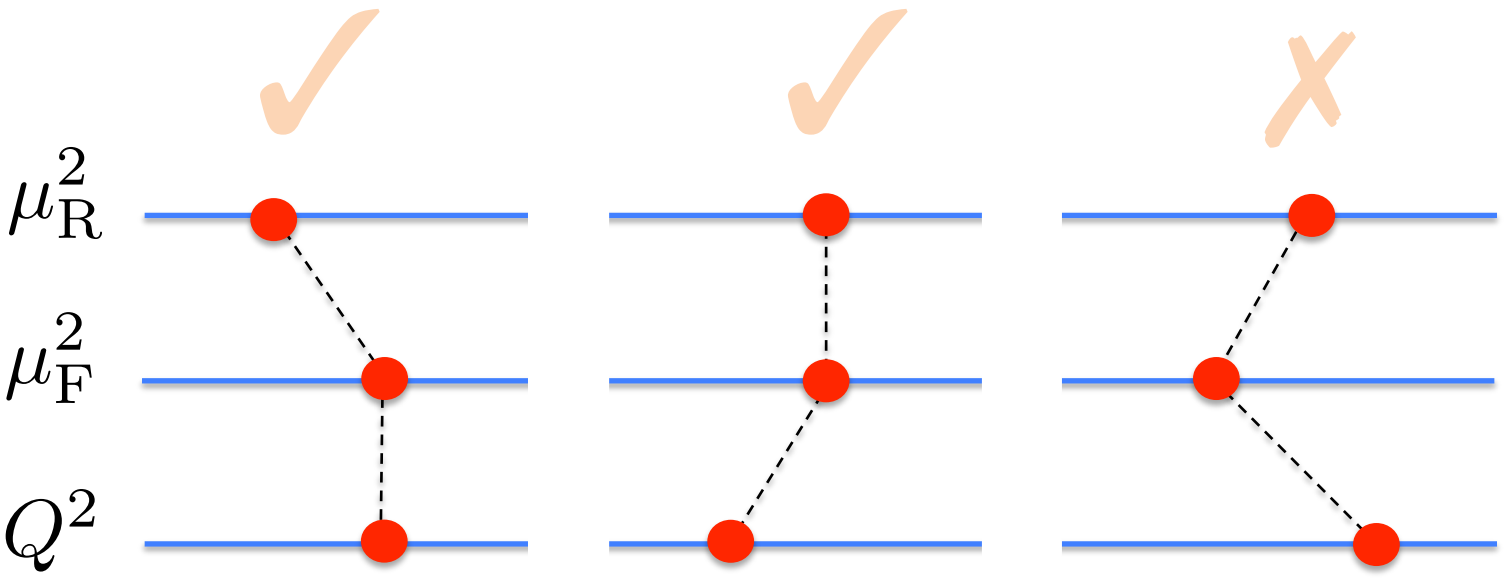
$$C_{i,j}^{\mathrm{N}^\ell\mathrm{LO}} = \sum_{k=0}^{\ell} a_\mathrm{s}^k \; C_{i,j}^{(k)} \qquad i = 2, \mathrm{L}, 3 \qquad j = \mathrm{g}, \mathrm{s}, +, -, \mathrm{v}$$

Remark about scales

- We now have two scales to vary

$$\mu_{\rm R}^2 = a\,\mu_{\rm F}^2 + b \qquad Q^2 = c\,\mu_{\rm F}^2 + d$$

- But in QCDNUM you cannot vary <u>both</u> simultaneously!



Renormalisation scale dependence $\mu_R^2 \neq \mu_F^2$

- Taylor expand α_s but truncate one order less than for the DGLAP splitting functions

Factorisation scale dependence $Q^2 \neq \mu_F^2$

$$C_{i,j}^{(0)} \rightarrow C_{i,j}^{(0)} \qquad C_{i,j}^{(k)} \rightarrow C_{i,j}^{(k)} + \sum_{m=1}^{k} C_{i,j}^{(k,m)} L_F^m \qquad k \geq 1$$

$$\begin{aligned}
\boldsymbol{C}_i^{(1,1)} &= \boldsymbol{C}_i^{(0)} \otimes \boldsymbol{P}^{(0)} \\
\boldsymbol{C}_i^{(2,1)} &= \boldsymbol{C}_i^{(0)} \otimes \boldsymbol{P}^{(1)} + \boldsymbol{C}_i^{(1)} \otimes \left[\boldsymbol{P}^{(0)} - \beta_0 \, \boldsymbol{I}\right] \\
\boldsymbol{C}_i^{(2,2)} &= \tfrac{1}{2} \, \boldsymbol{C_i^{(1,1)}} \otimes \left[\boldsymbol{P}^{(0)} - \beta_0 \, \boldsymbol{I}\right] \\
\boldsymbol{C}_i^{(3,1)} &= \boldsymbol{C}_i^{(0)} \otimes \boldsymbol{P}^{(2)} + \boldsymbol{C}_i^{(1)} \otimes \left[\boldsymbol{P}^{(1)} - \beta_1 \, \boldsymbol{I}\right] + \boldsymbol{C}_i^{(2)} \otimes \left[\boldsymbol{P}^{(0)} - 2\beta_0 \, \boldsymbol{I}\right] \\
\boldsymbol{C}_i^{(3,2)} &= \tfrac{1}{2} \left\{ \boldsymbol{C}_i^{(1,1)} \otimes \left[\boldsymbol{P}^{(1)} - \beta_1 \, \boldsymbol{I}\right] + \boldsymbol{C}_i^{(2,1)} \otimes \left[\boldsymbol{P}^{(0)} - 2\beta_0 \, \boldsymbol{I}\right] \right\} \\
\boldsymbol{C}_i^{(3,3)} &= \tfrac{1}{3} \boldsymbol{C}_i^{(2,2)} \otimes \left[\boldsymbol{P}^{(0)} - 2\beta_0 \, \boldsymbol{I}\right]
\end{aligned}$$

- Complicated but convolution engine can handle this

The ZMSTF add-on package

1: Calculate weight tables

Subroutine or function	Description
`ZMWORDS ( *ntotal, *nused )`	Words available, used
`ZMFILLW ( *nused )`	Fill weight tables
`ZMDUMPW ( lun, 'filename' )`	Dump weight tables
`ZMREADW ( lun, 'filename', *nused, *ierr )`	Read weight tables
`ZMDEFQ2 ( a, b )`	Define Q^2
`ZMABVAL ( *a, *b )`	ve a and b coefficients
`ZMQFRMU ( qmu2 )`	Convert μ_F^2 to Q^2
`ZMUFRMQ ( Q2 )`	Convert Q^2 to μ_F^2
`ZSWITCH ( iset )`	Switch pdf set
`ZMSTFUN ( istf, def, x, Q2, *f, n, ichk )`	Structure functions

Output arguments are pre-fixed with an asterisk (*).

2: Define Q^2 scale

3: Calculate structure function

Alternative for F_L

- Unlike F_2 we have that F_L is zero at LO

$$C_2 = C_2^{(0)} + \alpha_s C_2^{(1)} + \alpha_s^2 C_2^{(2)}$$

$$C_L = 0 + \alpha_s C_L^{(1)} + \alpha_s^2 C_L^{(2)}$$

- The expansion below makes F_L' nonzero at LO

$$C_L' = \alpha_s C_L^{(1)} + \alpha_s^2 C_L^{(2)} + \alpha_s^3 C_L^{(3)}$$

- Implemented in ZMSTF with the NNNLO coefficient functions from Moch et al

S. Moch et al., PL B606 (2005) 123

FFNS heavy quark structure functions

- Idea: proton consists of light quarks only and heavy quark is produced in hard scattering

  E. Laenen et al.,NP B392 (1993) 162
  S.Riemersma et al., PL B347 (1995) 143

$$F_2 = \begin{cases} F_2(\underbrace{\mathrm{uds}}_{n_f=3}) + F_2^{\mathrm{c}} \\ F_2(\underbrace{\mathrm{udsc}}_{n_f=4}) + F_2^{\mathrm{b}} \end{cases}$$

- Only one heavy quark
- Only F_2 and F_L
- Only up to NLO
- Only EM contribution thus no Z, W exchange
- Needs FFNS PDFs
- $F_{2,L}$(light) from the ZMSTF package

The HQSTF add-on package

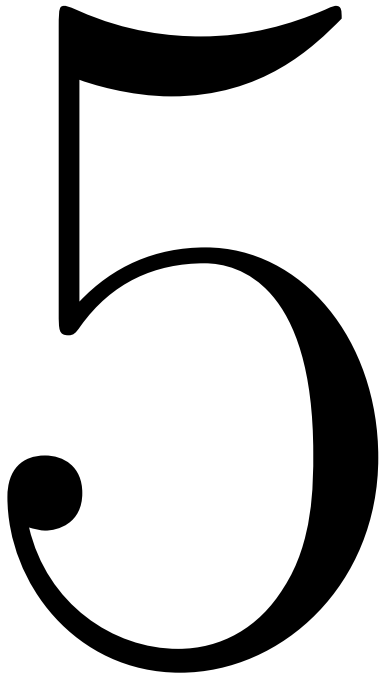
1: Calculate weight tables

Subroutine or function	Description
HQWORDS (*ntotal, *nused)	Words available, used
HQFILLW (istf, qmass, aq, bq, *nused)	Fill weight tables
HQDUMPW (lun, 'filename')	Dump weight tables
HQREADW (lun, 'filename', *nused, *ierr)	Read weight tables
HQPARMS (*qmass, *aq, *bq)	Retrieve parameters
HQQFRMU (qmu2)	Convert μ^2 to Q^2
HQMUFRQ (Q2)	
HSWITCH (iset)	Switch pdf set
HQSTFUN (istf, icbt, def, x, Q2, *f, n, ichk)	Structure functions

Output arguments are pre-fixed with an asterisk (*).

2: Define hq masses and Q^2 scale

3: Calculate structure function



Convolution Engine

Convolution Engine

$$f \otimes C \equiv x \int_\chi^1 \frac{\mathrm{d}z}{z} f(z,\mu^2)\, C\left(\frac{\chi}{z}, \mu^2, Q^2, m_\mathrm{h}^2\right)$$

- Uses rescaling variable like $\chi \equiv ax = \left(1 + \frac{\mu_h^2}{Q^2}\right) x$
- C(…) and a(…) must be supplied as Fortran functions
- You can generate weight tables at initialisation and then enjoy very fast convolution as weighted sum

- Weight routines can handle singularities

$$C = A + [B]_+ + R[S]_+ + D\delta(1-x)$$

- Each term has its own Makewt routine

Store

Coefficient function

```
call MAKEWTA(w,id,afun,achi)
call MAKEWTB(w,id,bfun,achi,0)
call MAKEWRS(w,id,rfun,sfun,achi,0)
call MAKEWTD(w,id,dfun,achi)
```

Table identifier

Defines rescaling variable

How to get your Structure Function

- Generate weight tables at initialisation
- Write a structure function function like

```
function stf(ix,iq)
fcc = FCROSSK(w,idw,iset,idf,ix,iq)
stf = GETALFN(iq,n,ierr)*fcc
return
```

- Pass this function to an interpolation routine

```
call STFUNXQ(stf,x,Q2,result,1,0)
```

The convolution engine is a pretty big beast …

Subroutine or function	Description
`BOOKTAB ( w, nw, itypes, *nwords )`	Partition into tables
`MAKEWTA ( w, id, afun, achi )`	Regular piece $A(x)$
`MAKEWTB ( w, id, bfun, achi, nodelta )`	Singular piece $[B(x)]_+$
`MAKEWRS ( w, id, rfun, sfun, achi, nodelta )`	Product $R(x)[S(x)]_+$
`MAKEWTD ( w, id, dfun, achi )`	Delta function $D(x)\delta(1-x)$
`MAKEWTX ( w, id )`	Weight table for $x[f_a \otimes f_b]$
`SCALEWT ( w, c, id )`	Scale weight table
`IDSPFUN ( 'pij', iord, itype )`	Splitting function index
`COPYWGT ( w, id1, id2, iadd )`	Copy weight table
`WCROSSW ( w, ida, idb, idc, iadd )`	Double convolution weights
`WTIMESF ( w, fun, id1, id2, iadd )`	Multiply by $f(\mu^2, n_f)$
`SETWPAR ( w, pars, n )`	Store extra information
`GETWPAR ( w, *pars, n )`	Read extra information
`TABDUMP ( w, lun, 'filename', 'key' )`	Dump to disk
`TABREAD ( w, n, lun, 'fn', 'key', *nw, *ierr )`	Read from disk

Output arguments are pre-fixed with an asterisk (*).

Description
Convolution $x[f \otimes K]$
Convolution $x[f_a \otimes f_b]$
Transform from $q, \bar{q}$ to $e^{\pm}$
Transform from $e^{\pm}$ to $q, \bar{q}$
Returns n_f
Returns $(\alpha_s/2\pi)^n$

`EFROMQQ ( qvec, *evec, nf )`
`QQFROME ( evec, *qvec, nf )`
`NFLAVOR ( iq )`
`GETALFN ( iq, n, *ierr )`

Output arguments are pre-fixed with an asterisk (*).

	…n
	…of x and μ^2 values
	…fer
	…$e^{\pm}\rangle$ in a scratch buffer
	…glet/non-singlet component
	…ghted sum of $\|e^{\pm}\rangle$
`FASTFXK ( w, idw, idf, idout )`	Convolution $x[f \otimes K](x)$
`FASTFXF ( w, idw, ida, idb, idout )`	Convolution $x[f_a \otimes f_b](x)$
`FASTKIN ( id, fun )`	Scale by a kinematic factor
`FASTCPY ( idin, idout, iadd )`	Copy or accumulate result
`FASTFXQ ( id, *f, n )`	Interpolation

Output arguments are pre-fixed with an asterisk (*).

… cannot fully describe all its features here

We're done!