The evolution of APFEL: APFEL++

Valerio Bertone
INFN and Università di Pavia

Istituto Nazionale di Fisica Nucleare

UNIVERSITÀ DI PAVIA

European Research Council
Established by the European Commission

xFitter External Meeting 2019

March 18, 2019, Minsk

3DSPIN
MAPPING THE PROTON IN 3D

Motivation

- Since its born, APFEL has undergone a large number of developments:
  - FONLL structure functions,
  - NLO QED evolution,
  - lepton PDFs,
  - Scale variations,
  - intrinsic charm,
  - displaced thresholds,
  - $\overline{\rm MS}$ masses,
  - small-x resummation,
  - interface to the NNPDF code,
  - …
- Way beyond the purposes for which it was conceived:
  - very large memory footprint,
  - non-optimal "convenience" solutions for the new modules,
  - hard to maintain.
- APFEL is written in FORTRAN77 that is not suitable for large projects:
  - lack of modularity,
  - non-optimal (built-in) memory management.
- Compelling reasons to rewrite APFEL keeping in mind its applications.

Design of the code

- Concerning the **language**, **C++** was a somewhat natural choice:
  - **modularity** ensured by the object-oriented nature,
  - **dynamical** allocation of the **memory**,
  - used for the new-generation tools (*e.g.* LHAPDF) and thus easier **interface**,
  - powerful features coming with the **C++11/17 standard**.
- The code **design** was driven by a profound rethinking of the strategy:
  - the main application field is **collinear/TMD factorisation**.
  - In this context, many relevant quantities are computed as **convolutions**:

$$M(x) = \int_x^1 \frac{dy}{y} O(y) d\left(\frac{x}{y}\right) = \int_x^1 \frac{dz}{z} O\left(\frac{x}{z}\right) d(z) \equiv O(x) \otimes d(x)$$

operator O: typically a complicated object slow to compute: *e.g.* a **perturbative** hard cross section.
distribution d: typically a fast-to-access function: *e.g.* a **non-perturbative** PDF or a FF.

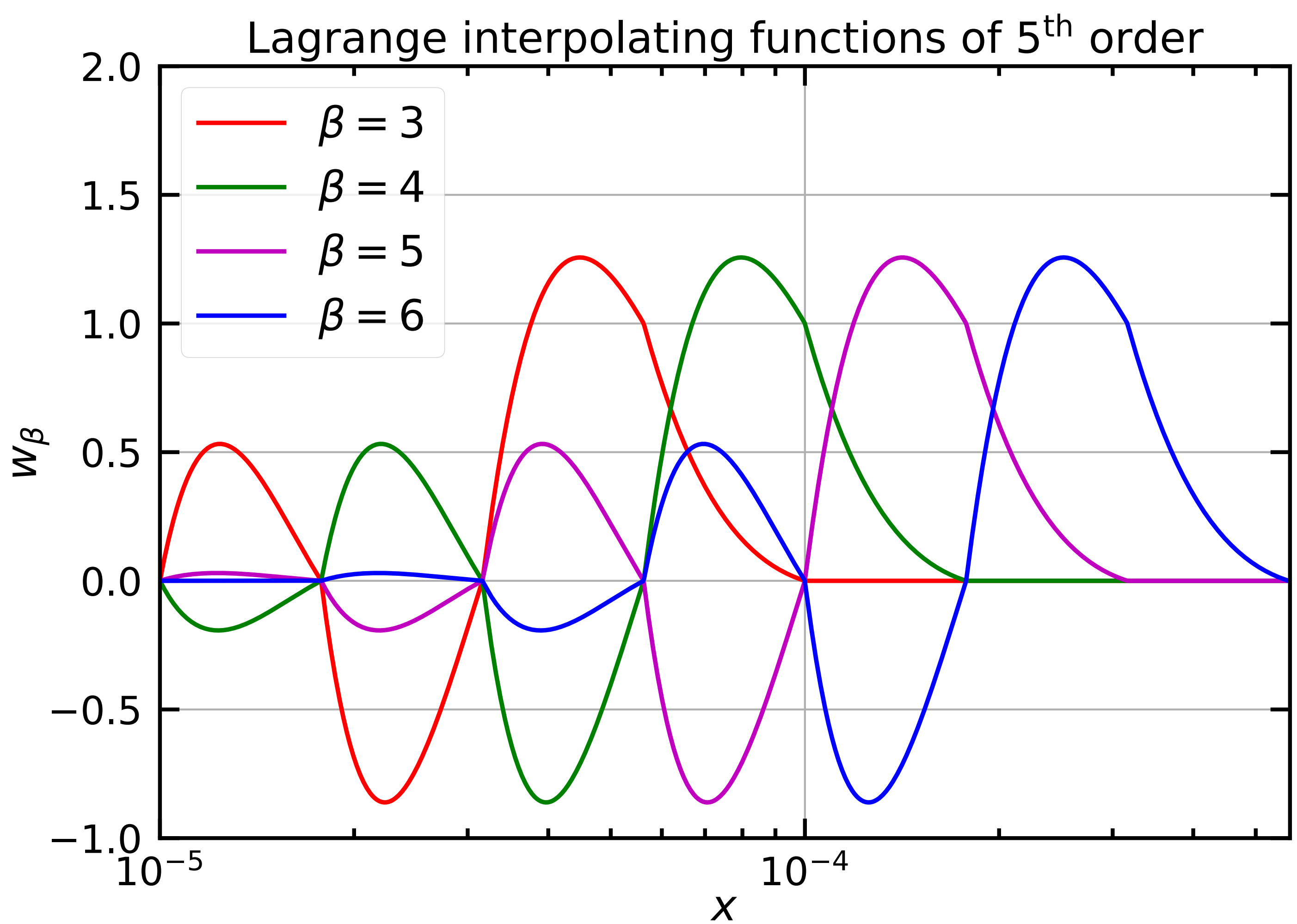
  - Adopt the ***x*-space** (as opposed to *N*-space) formalism:
    - most of the results are available in *x*-space,
    - no restriction on the parameterisations.
  - The purpose is to make convolutions **fast**.

Design of the code

- Define an **interpolation grid** in x with N+1 nodes $g \equiv \{z_0, \ldots, z_N\}$
- Use the interpolation formula for the **distribution** d:

$$d(z) = \sum_{\beta=0}^{N} w_\beta(z) d_\beta \quad \text{with} \quad d_\beta = d(x_\beta)$$

  - $w_\beta(z)$ *interpolating function* (typically a Lagrange polynomial of some degree $\boldsymbol{n}$).



  - **piecewise** function different from zero over $\boldsymbol{n+1}$ intervals around $\boldsymbol{\beta}$. **Zero** elsewhere. **Hard to integrate**.

Design of the code

- Compute the integral of the **operator** O with the interp. functions:

$$O_{\alpha\beta} \equiv \int_{x_\alpha}^{1} \frac{dy}{y} O(y) w_\beta\left(\frac{x_\alpha}{y}\right)$$

such that:

$$M_\alpha \equiv \sum_{\beta=0}^{N} O_{\alpha\beta} d_\beta \quad \text{with} \quad M(x) = \sum_{\alpha=0}^{N} w_\alpha(x) M_\alpha$$

- This reduces convolutions to multiplications between a matrices and vectors: **linear algebra**.
- Therefore, the **three main ingredients** of of APFEL++ are:
  1. the **interpolation grid** g along with the interpolating functions,
  2. the **distribution** d_β,
  3. the **operator** $O_{\alpha\beta}$.
- They can be **encapsulated** in **C++ objects** to compute convolutions.

Design of the code

- An additional complication is given by the **flavour structure**:
  - distributions are **vectors in flavour space**,
  - operators are either **matrices or vectors in the flavour space**.

$$\frac{d f_{i\alpha}}{d \ln \mu^2} = \sum_{j,\beta} P^{ij}_{\alpha\beta} f_{j\beta}$$

DGLAP equations

$$F_\alpha = \sum_{j,\beta} C^{j}_{\alpha\beta} f_{j\beta}$$

DIS structure functions

- Matrices in flavour space are often **sparse**.
- Define **sets of objects** with a **flavour map**:
  - associate one operator to one distribution, *e.g.*:

$$\begin{array}{lcl} P_{qq}, P_{gq} & \to & \Sigma \\ P_{qg}, P_{gg} & \to & g \\ P^{v} & \to & V \\ P^{+} & \to & T_{3,8,15,24,35} \\ P^{-} & \to & V_{3,8,15,24,35} \end{array}$$

    - the same operator can be assigned to more than a distribution and viceversa.
  - avoid **multiplications by zero**.

Design of the code

- An additional complication is given by the **flavour structure**:
  - distributions are **vectors in flavour space**,
  - operators are either **matrices or vectors in the flavour space**.

$$\frac{df_{i\alpha}}{d\ln\mu^2} = \sum_{j,\beta} P^{ij}_{\alpha\beta} f_{j\beta}$$

DGLAP equations

$$F_{\alpha} = \sum_{j,\beta} C^{j}_{\alpha\beta} f_{j\beta}$$

DIS structure functions

- Matrices in flavour space are often **sparse**.
- Define **sets of objects** with a **flavour map**:
  - associate one operator to one distribution, *e.g.*:

Set of operators	Flavour map	Set of distributions
P_{qq}, P_{gq}	$\rightarrow$	Σ
P_{qg}, P_{gg}	$\rightarrow$	g
P^{v}	$\rightarrow$	V
P^{+}	$\rightarrow$	$T_{3,8,15,24,35}$
P^{-}	$\rightarrow$	$V_{3,8,15,24,35}$

  - the same operator can be assigned to more than a distribution and viceversa.
  - avoid **multiplications by zero**.

Design of the code

- Define **multiplication** between sets of distributions and operators:
  - **overload** multiplication operator in C++.
  - Making convolutions with a flavour structure becomes very **easy**.

```
//_________________________________________________________________________________
Set<Distribution> Dglap::Derivative(int const& nf, double const& t, Set<Distribution> const& f) const
{
  return _SplittingFunctions(nf, exp(t/2)) * f;
}
```

Design of the code

- Define **multiplication** between sets of distributions and operators:
  - **overload** multiplication operator in C++.
  - Making convolutions with a flavour structure becomes very **easy**.

```
//______________________________________________________________________________
Set<Distribution> Dglap::Derivative(int const& nf, double const& t, Set<Distribution> const& f) const
{
  return _SplittingFunctions(nf, exp(t/2)) * f;
}
```

Set of distributions

Set of operators

Overloaded multiplication:
takes care of convolutions
and flavour structure

- The flavour structure is completely defined by the flavour map:
  - **same procedure** for convolutions in **any flavour basis**,
  - sets of operators and distributions multiplied only if they **share** the same map,
  - easy to account for n_f **dependence**.

Design of the code

- Use (e.g.) 4th order Runge-Kutta to solve systems of **ordinary differential equations:**

$$\begin{cases} \dfrac{d\mathbf{y}}{dt} = \mathbf{F}(t, \mathbf{y}) \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases}$$

Template function…
… that returns a **std::function**…
… of a **std::function**

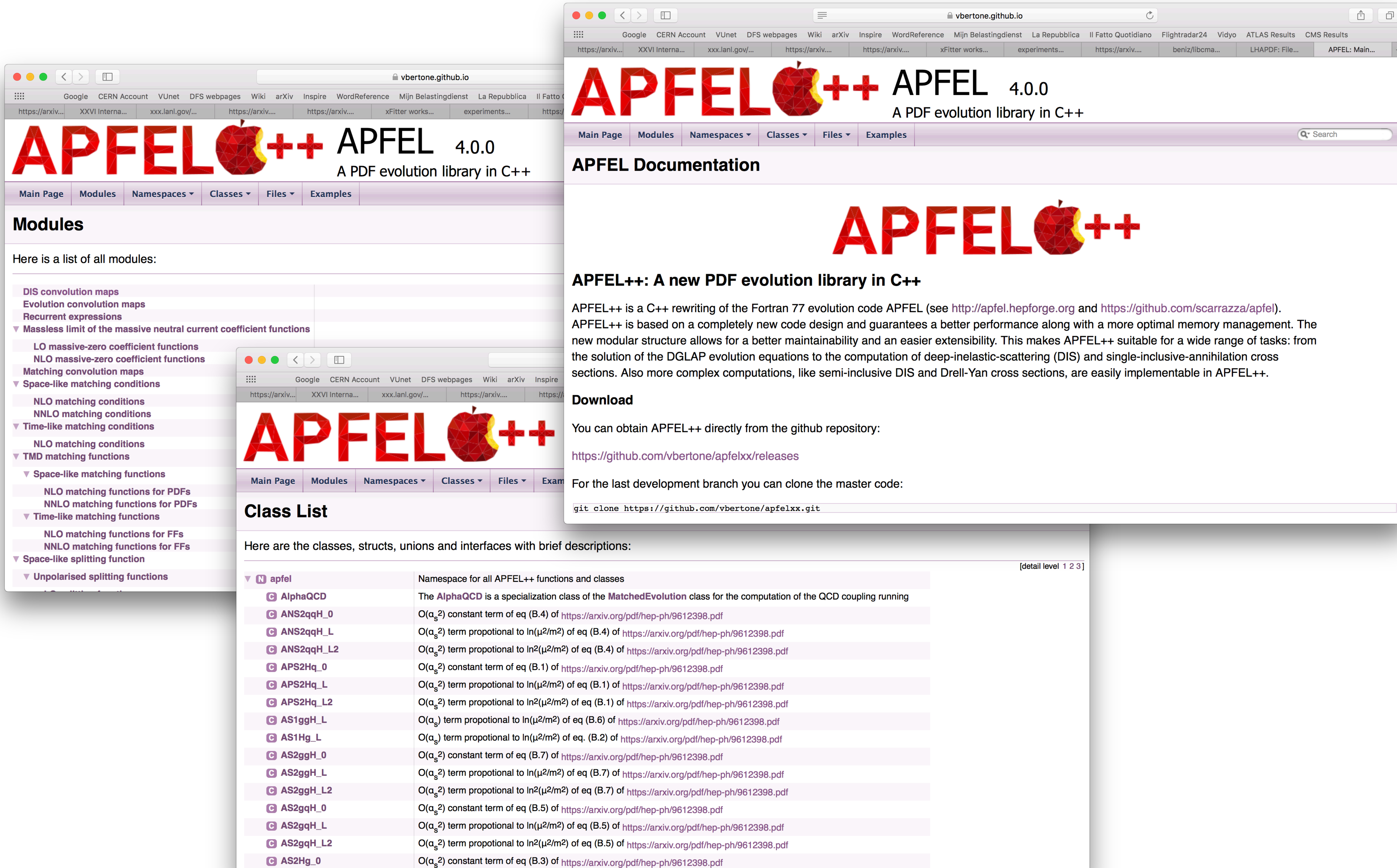
Five nested **Lambda** functions

```
template<class U>
function<U(double const&, U const&, double const&)>
rk4(function<U(double const& t, U const& Obj)> const& f)
{
  return
    [       f            ](double const& t, U const& y,  double const& dt) -> U{ return
    [t,y,dt,f            ](                 U const& dy1                  ) -> U{ return
    [t,y,dt,f,dy1        ](                 U const& dy2                  ) -> U{ return
    [t,y,dt,f,dy1,dy2    ](                 U const& dy3                  ) -> U{ return
    [t,y,dt,f,dy1,dy2,dy3](                 U const& dy4                  ) -> U{ return
    ( dy1 + 2 * dy2 + 2 * dy3 + dy4 ) / 6   ;} (
    dt * f( t + dt     , y + dy3     )     );} (
    dt * f( t + dt / 2, y + dy2 / 2 )     );} (
    dt * f( t + dt / 2, y + dy1 / 2 )     );} (
    dt * f( t          , y           )     );} ;
}
```

- Very same function used to solve **both** the DGLAP and the α_s RGE.

Doxygen documentation

https://vbertone.github.io/apfelxx/html/index.html



Convoluting operators

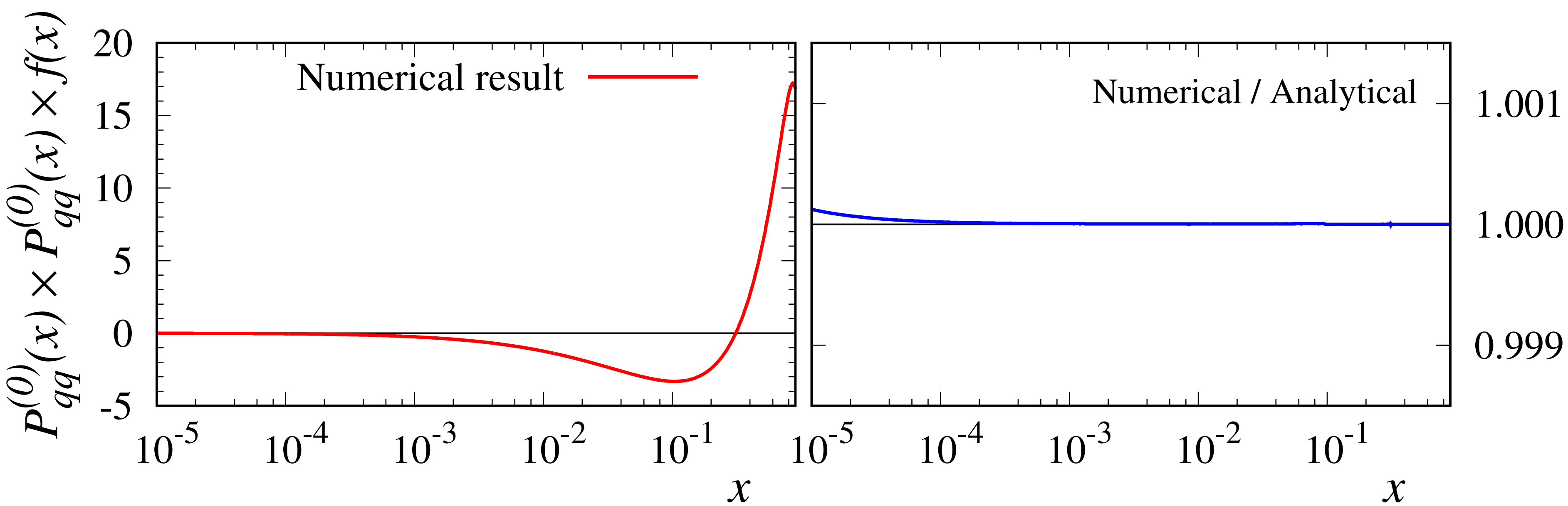
- An operation that is often needed is the convolution between operators:
  - involved in the computation of factorisation **scale variations**,
  - computation of the **PDF evolution operator**.

$$M(x) = \underbrace{O^{(1)}(x) \otimes O^{(2)}(x)}_{O(x)} \otimes d(x) \;\rightarrow\; M_\alpha = \sum_\beta \underbrace{\sum_\gamma O^{(1)}_{\alpha\gamma} O^{(2)}_{\gamma\beta}}_{O_{\alpha\beta}} d_\beta$$

- Consider for example:

$$P^{(0)}_{qq}(x) = \left(\frac{1+x^2}{1-x}\right)_+ \quad \text{such that} \quad P^{(0)}_{qq}(x) \otimes P^{(0)}_{qq}(x) = \left(\frac{4(x^2+1)\ln(1-x)+x^2+5}{1-x}\right)_+ - \frac{(3x^2+1)\ln x}{1-x} - 4 + \left(\frac{9}{4} - \frac{2\pi^2}{3}\right)\delta(1-x)$$

- Compare the numerical convolution with the analytic result (using a test function *f*(*x*)):



Evolution operator

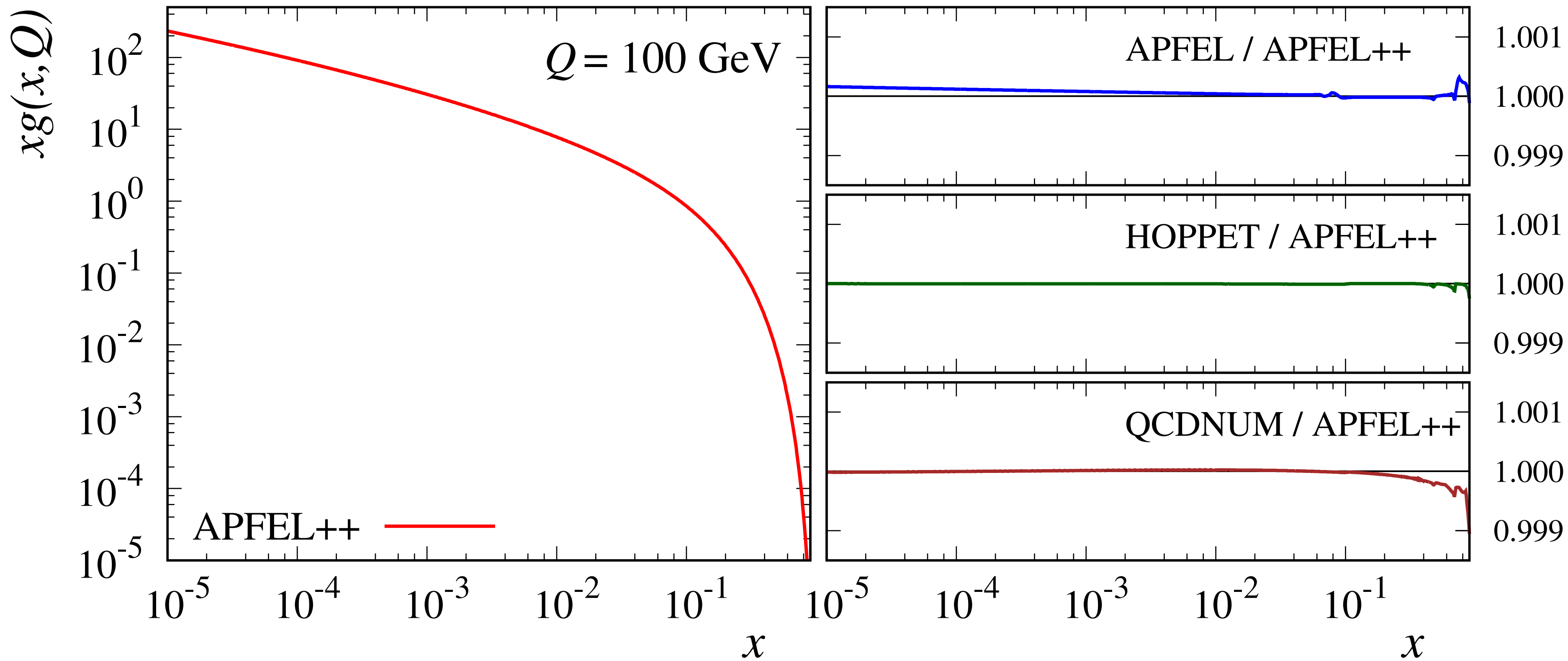
- The DGLAP can be written in terms the **evolution operator**:

$$
\begin{cases}
\dfrac{d}{d\ln\mu^2}\Gamma^{ij}_{\alpha\beta}(\mu_0,\mu) = \displaystyle\sum_{k,\gamma} P^{ik}_{\alpha\gamma}(\mu)\Gamma^{kj}_{\gamma\beta}(\mu_0,\mu) \\[2em]
\Gamma^{ij}_{\alpha\beta}(\mu_0,\mu_0) = \delta_{ij}\delta_{\alpha\beta}
\end{cases}
$$

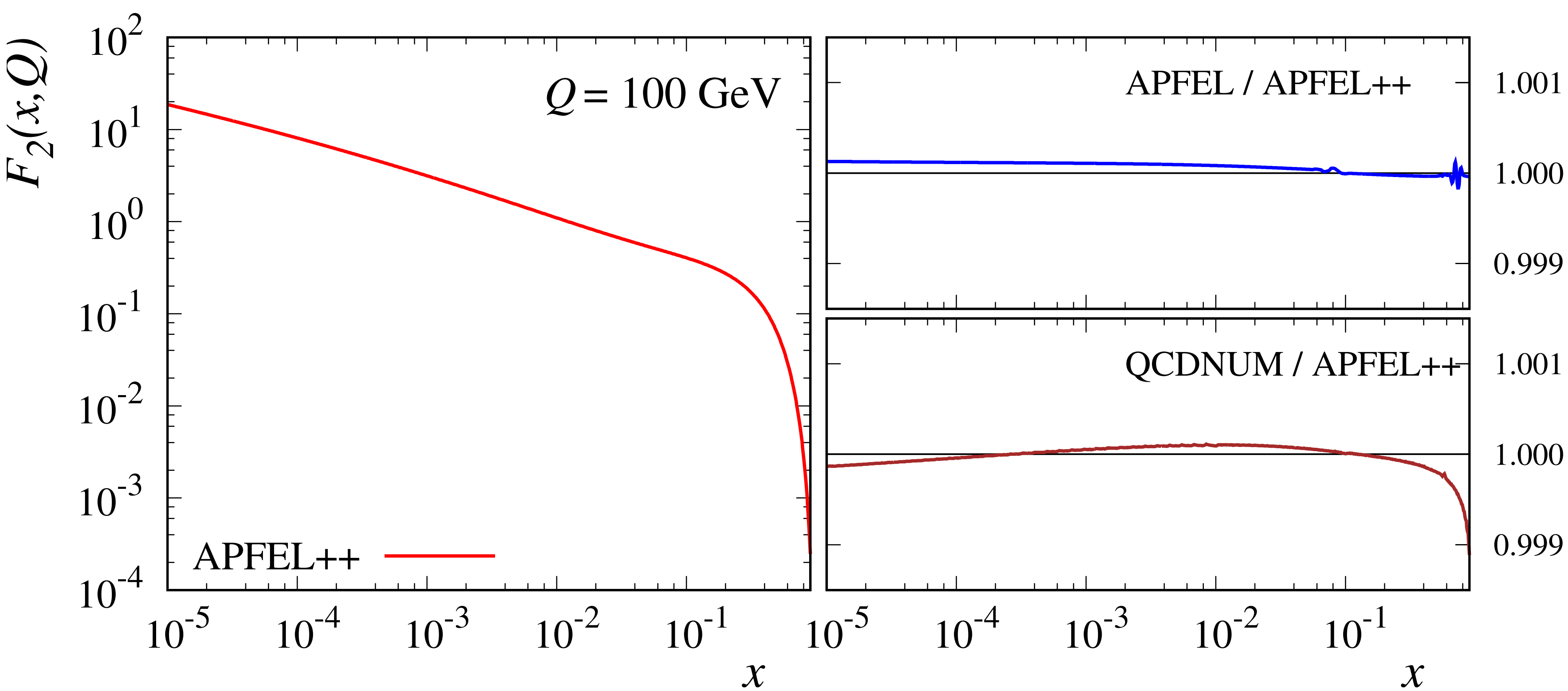
- The evolution operator can be used to evolve any initial scale PDF:
  - **harder** to compute than evolving PDFs directly,
  - it has to be computed **only once** (for each μ_0 and μ).
- This object is used for the construction of the **APFELgrid** tables:
  - extremely hard to compute in APFEL (Fortran),
  - very easy with APFEL++.

Applications

- DGLAP evolution at NNLO:



- DIS structure functions at NNLO:



PDF evolution performance

- Comparison between different codes:

NNLO QCD evolution ~200 points in x ~50 points in Q	Initialisation [s]	Interpolate PDFs 10^6 times [s]
APFEL++	0.4 (0.27 in. + 0.13 tab.)	0.6
APFEL	2.4	1.9
HOPPET	0.4	1.3
QCDNUM	8.7	1.3

- Compare APFEL++ to LHAPDF when interpolating a **std::map**:

PDF set: NNPDF31_nlo_as_0118	Interpolate a PDF map 10^5 times [s]
APFEL++	0.7
LHAPDF	0.5

APFEL through LHAPDF

- Idea: delegate LHAPDF to interpolate over the (x, Q^2) grid:
  - more performing,
  - standardise the access to PDFs.

```cpp
// APFEL++ default EvolutionSetup object
apfel::EvolutionSetup es{};

// Feed it to the initialisation class of APFEL++
apfel::InitialiseEvolution ev{es, true};

// Construct pointer to LHAPDF::PDF object
LHAPDF::PDF* distAP = mkPDF(ev);

// Call PDFs
const std::map<int, double> PDFmap = distAP->xfxQ(x, mu);
```

APFEL++ evolution:
AlphaQCD(Q) = 1.1638e-01

x	u-ubar	d-dbar	2(ubr+dbr)	c+cbar	gluon
1.0e-05	5.0597e-03	2.9000e-03	2.9868e+01	1.3409e+01	1.8825e+02
1.0e-04	2.0510e-02	1.1486e-02	1.4191e+01	6.0136e+00	8.0275e+01
1.0e-03	7.7062e-02	4.3014e-02	6.0966e+00	2.3549e+00	2.9215e+01
1.0e-02	2.4305e-01	1.3669e-01	2.2041e+00	7.1400e-01	7.9998e+00
1.0e-01	5.1440e-01	2.4974e-01	3.9254e-01	7.5071e-02	8.9562e-01
3.0e-01	3.3154e-01	1.1785e-01	3.5867e-02	6.7898e-03	9.6761e-02
5.0e-01	1.0288e-01	2.6959e-02	2.4053e-03	5.4369e-04	9.5518e-03
7.0e-01	1.4110e-02	2.6031e-03	5.1163e-05	1.3984e-05	3.6137e-04
9.0e-01	1.8836e-04	2.4841e-05	2.1042e-08	8.2414e-09	7.3548e-07

LHAPDF (tabulated) evolution:
AlphaQCD(Q) = 1.1638e-01

x	u-ubar	d-dbar	2(ubr+dbr)	c+cbar	gluon
1.0e-05	5.0549e-03	2.9037e-03	2.9871e+01	1.3411e+01	1.8827e+02
1.0e-04	2.0510e-02	1.1491e-02	1.4193e+01	6.0140e+00	8.0282e+01
1.0e-03	7.7066e-02	4.3017e-02	6.0971e+00	2.3550e+00	2.9217e+01
1.0e-02	2.4306e-01	1.3670e-01	2.2043e+00	7.1472e-01	8.0004e+00
1.0e-01	5.1438e-01	2.4974e-01	3.9226e-01	7.5312e-02	8.9450e-01
3.0e-01	3.3153e-01	1.1785e-01	3.5865e-02	6.8228e-03	9.6755e-02
5.0e-01	1.0288e-01	2.6957e-02	2.4027e-03	5.4586e-04	9.5468e-03
7.0e-01	1.4109e-02	2.6030e-03	5.1163e-05	1.4071e-05	3.6138e-04
9.0e-01	1.8834e-04	2.4838e-05	2.1027e-08	8.3325e-09	7.3525e-07

Old functionalities

- The FORTRAN version of APFEL implements a **very large** number of functionalities.
- I'm currently working to implement all of them also in APFEL++.
- **Missing** functionalities in APFEL++ to be implemented:
  - QED corrections,
  - intrinsic charm,
  - $\overline{\rm MS}$ masses,
  - small-x resummation (need to interface APFEL++ to HELL),
  - scale variations,
  - "minor" functionalities:
    - target mass corrections,
    - different solutions for the DGLAP and coupling evolutions (?).

New functionalities

- I have already started using APFEL++ for tasks difficult to implement in (or even out of reach) for the FORTRAN version.
- Examples are:
  - Semi-Inclusive DIS (SIDIS) in collinear factorisation:
    - double convolution with time- and space-like evolution at the same time.
  - TMD phenomenology:
    - evolution and matching,
    - Drell-Yan and SIDIS q_T distributions.
  - DGLAP evolution with splitting explicitly depending the factorisation scale:
    - e.g. "Physical"-scheme evolution (by Martin and Ryskin).
  - Transversity distributions (PDFs and FFs).

SIDIS in collinear factorisation

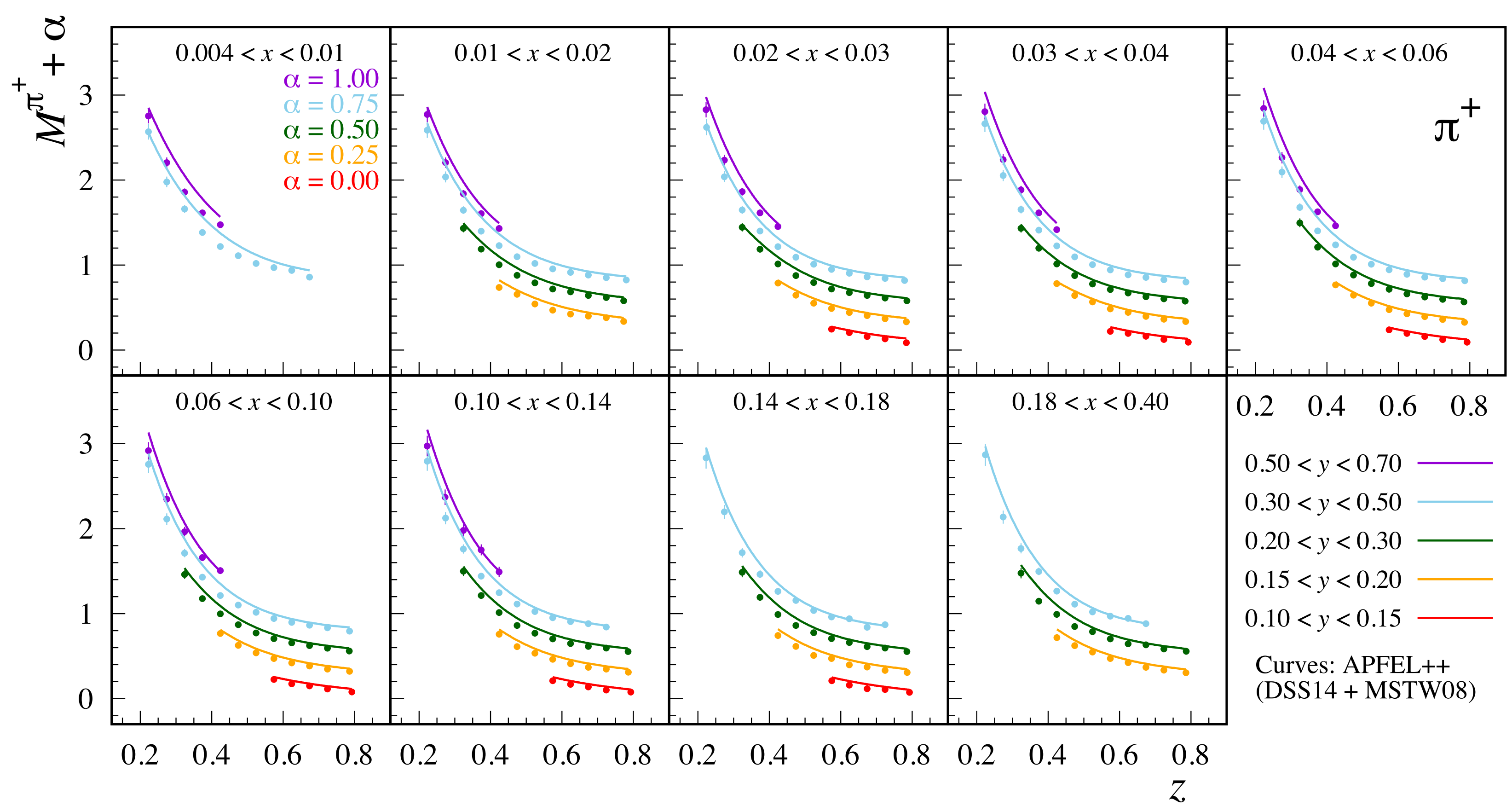
- SIDIS cross sections (integrated over q_T) have this structure:

$$D(x,z) = \int_x^1 \frac{d\xi}{\xi} \int_z^1 \frac{d\zeta}{\zeta} O\left(\frac{x}{\xi}, \frac{z}{\zeta}\right) d^{(1)}(\xi) d^{(2)}(\zeta)$$

- But the hard cross sections (at least up to NLO) factorise as:

$$O(x,z) = \sum_i K_i C_i^{(1)}(x) C_i^{(2)}(z)$$

- Combination of single convolutions:



- Next, I will also try with Drell-Yan cross sections.

TMD Evolution (PDFs)

$$F_{f/P}(x, \mathbf{b}_T; \mu, \zeta) = \sum_j C_{f/j}(x, b_*; \mu_b, \zeta_F) \otimes f_{j/P}(x, \mu_b) \quad : A$$

$$\times \exp\left\{ K(b_*; \mu_b) \ln \frac{\sqrt{\zeta_F}}{\mu_b} + \int_{\mu_b}^{\mu} \frac{d\mu'}{\mu'} \left[\gamma_F - \gamma_K \ln \frac{\sqrt{\zeta_F}}{\mu'} \right] \right\} \quad : B$$

$$\times \exp\left\{ g_{j/P}(x, b_T) + g_K(b_T) \ln \frac{\sqrt{\zeta_F}}{\sqrt{\zeta_{F,0}}} \right\} \quad : C$$

TMD Evolution (PDFs)

$$F_{f/P}(x, \mathbf{b}_T; \mu, \zeta) = \sum_j C_{f/j}(x, b_*; \mu_b, \zeta_F) \otimes f_{j/P}(x, \mu_b) \quad : A$$

$$\times \exp\left\{ K(b_*; \mu_b) \ln \frac{\sqrt{\zeta_F}}{\mu_b} + \int_{\mu_b}^{\mu} \frac{d\mu'}{\mu'} \left[\gamma_F - \gamma_K \ln \frac{\sqrt{\zeta_F}}{\mu'} \right] \right\} \quad : B$$

$$\times \exp\left\{ g_{j/P}(x, b_T) + g_K(b_T) \ln \frac{\sqrt{\zeta_F}}{\sqrt{\zeta_{F,0}}} \right\} \quad : C$$

- $b_T \ll 1/\Lambda_{QCD}$
- matching to the collinear region
- factorises as hard and non-perturbative
- numerically cumbersome
- precompute using APFEL

- CS evolution
- perturbative

- matching between the small and large b_T
- non perturbative
- parametrised and fitted to data

SIDIS in TMD factorisation

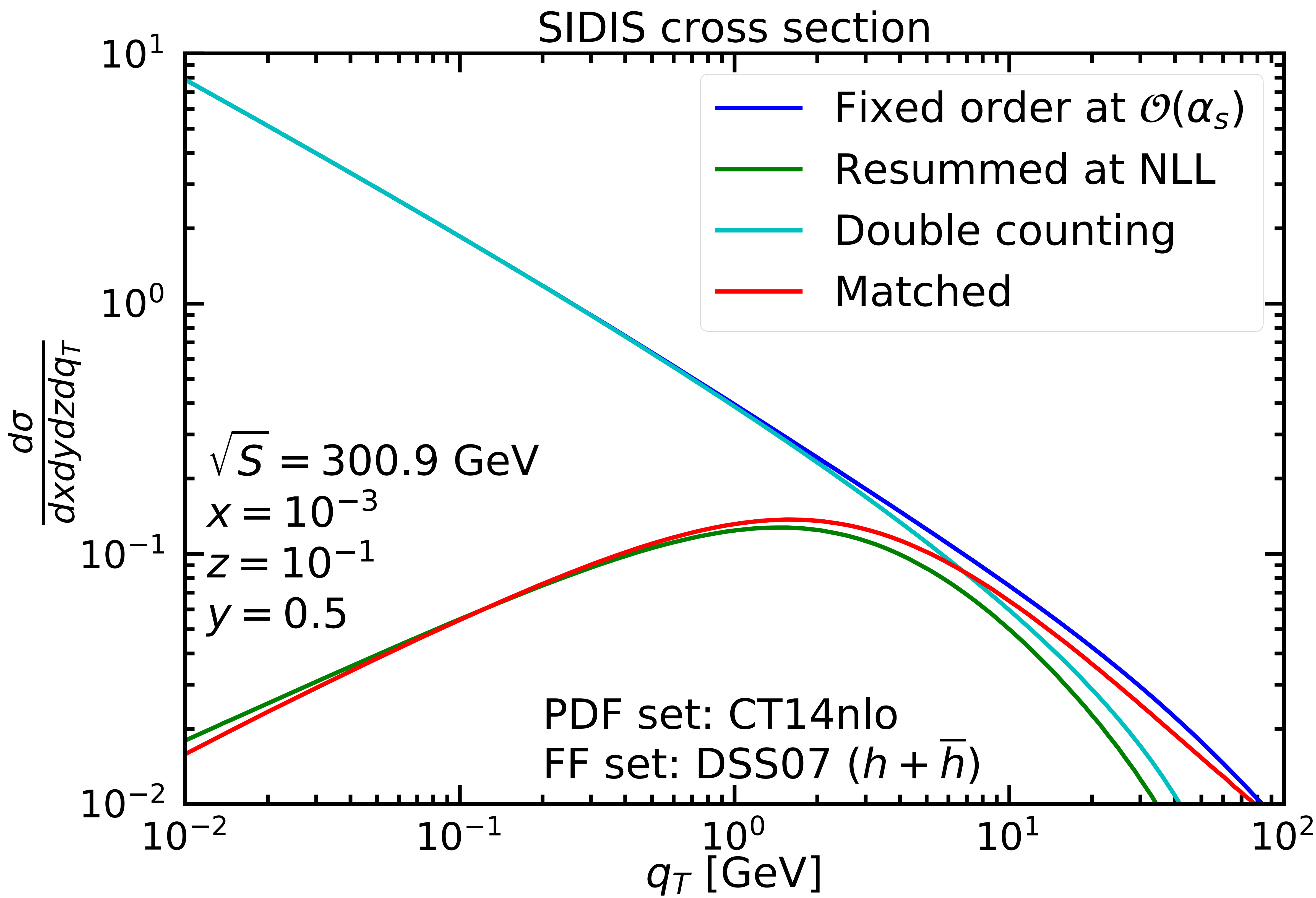
- In SIDIS, what enters the computation of the cross sections is:

$$\mathcal{L}_{\rm SIDIS} = \int \frac{d^2\mathbf{b}_T}{(2\pi)^2} e^{-i\mathbf{q}_T\cdot\mathbf{b}_T} F_{f/P}(x,\mathbf{b}_T;\mu,\zeta_F) D_{H/f}(x,\mathbf{b}_T;\mu,\zeta_D)$$

Fourier transform PDFs FFs

- The ingredients are:
  - a set of evolved TMD-PDFs,
  - a set of evolved TMD-FFs,
  - the Fourier transform of its product.
- Complex set of tasks that have to be performed optimally
- APFEL provides the ideal environment for this computation:
  - fast and accurate interpolation techniques,
  - precomputation of the time consuming bits.

Matching collinear and TMD regimes



Plans for the future

- Interface to **yaml** for parsing of evolution parameters.
- Interface to **APFELgrid**.
- Interface to **APPLgrid/FastNLO**:
  - Drell-Yan and SIDIS cross sections (?).
- PDF evol. and structure functions in a "OO" fashion useful for **xFitter**:
  - many possible evolution and structure functions available at the same time,
  - assign different evolutions to different datasets (e.g. H-VFNS),
  - fit PDFs and FFs at the same time (space- and time-like evolution).