artemide ver.2.06

Alexey A. Vladimirov
August 22, 2022

User manual for `artemide` package, which evaluated TMDs and related cross-sections.

Manual is updating.

If you use the `artemide`, please, quote [1].
If you find mistakes, have suggestions, or have questions write to:
Alexey Vladimirov: *Alexey.Vladimirov@physik.uni-regensburg.de* or *vladimirov.aleksey@gmail.com*

Stable Repository: https://github.com/VladimirovAlexey/artemide-public
Dev Repository: https://github.com/VladimirovAlexey/artemide-development

Contents

Glossary

TMD = Transverse Momentum Dependent
PDF = Parton Distribution Function
FF = Fragmentation Function
NP = Non-perturbative
DY = Drell-Yan process
SIDIS = Semi-Inclusive Deep-Inelastic Scattering

I. GENERAL STRUCTURE AND USER INPUT OF ARTEMIDE

A. General concept

The `artemide` is a package of Fortran modules for calculation in TMD factorization framework. It is has a modular structure, where each module is responsible for evaluation of some theory construct. For instance: a TMD distribution, a TMD evolution factor, cross-section. Each module produces a single function, which is a composite of integrals/products of lower-level functions. Thus, each level of operation can be used as is, in other programs. The highest-level task is the evaluation of cross-section within the TMD factorization theorem, including all needed integrations, and factors, i.e., such that it can be directly compared to the data. It also includes several tools for analysis of the obtained values, such as variation of scales, search for limiting parameters, etc. The theory structure of `artemide` is discussed in the next section. The dependency structure of modules is presented in fig.1.

Initially the `artemide` project was created for pure theoretical games. Since the beginning `artemide` appears to be successful also in the phenomenology, (and nowadays it is mostly used for it). Nonetheless, conceptually `artemide` is build as the theory playground, and will continue to be developed in this direction. That is why it architecture is not very optimal from the pure numerical point of view. In fact, there are several ways to optimize the code, melting together some modules and structures, but in this case `artemide` will lose it theoretical cleanness. Also `artemide` contains a lot of "unpractical" and rare options, and possibility to control each parameter. The positive side is the possibility to easily implement the new theory founding and check them, which is regularly done.

Wide spectrum of application of `artemide` code makes it difficult to create a convenient interface. Moreover, at the current stage of development, I prioritize the quality of computation, to the user interface. So, the interface is changing from version to version and often is not compatible with earlier versions. It slowly converges to the (almost) perfect shape – convenient for a wider community. If you have a particular task and not sure how to operate with `artemide` in this case, better write an e-mail.

The main rule (implemented in ver.2.00) each part encapsulates its theory parameters. It does not affect/change/interact with other modules, except requests for functions. A change of a parameter in a module can require a change in another module for consistency (however, I attempt to avoid such cases). Then each parameter must be changed individually in each module. However, the module `aTMDe_control` does it automatically. So, I suggest to use `aTMDe_control` to avoid possible inconsistencies.

Historical note: In versions before ver.2.00 this rule was not implemented. I tried to make connections between modules such that they automatically control consistency. However, at some moment (after inclusion of many hadrons and different types of cross-sections, and different orders) it became practically tough to keep such a system. So, I rearrange some modules (e.g. variation of c_4 scale was in `TMDs`, while it related to definition of TMD it-self), removed connections between modules (no link for change of NP parameters, etc.), and introduce `aTMDe_control`.

B. Organization of TMD factorized cross-section and its implementation in `artemide`

The ultimate goal of the `artemide` is to evaluate the observables in the TMD factorization framework, such as cross-section, asymmetries, etc. The general structure of the TMD factorized *fully differential* cross-section is

$$\frac{d\sigma}{dX} = d\sigma(q_T) = prefactor \ \times F, \tag{1.1}$$

where *prefactor* a process-dependent and experiment-dependent prefactor, and F is the reduced structure function. Example, for the photon induced Drell-Yan process one has for $d\sigma/dq_T$

$$\begin{aligned} prefactor &= \frac{4\pi}{9sQ^2}|C_V(Q,\mu_H)|^2\mathcal{P}(\text{cuts}), \\ F &= \int \frac{bdb}{2} J_0(bq_T)\sum_f |e_f|^2 F_f(x_A,b;\mu_H,\zeta_A)F_{\bar f}(x_B,b;\mu_H,\zeta_B), \end{aligned}$$

where $\mathcal{P}$(cuts) is the weighting factor for fiducial cuts. For specific expression we refer in corresponding sections of this text. The structure function F is generally defined as

$$F(q_T,x_1,x_2;\mu,\zeta_1,\zeta_2) = \int \frac{bdb}{2} b^n J_n(bq_T)\sum_{ff'} z_{ff'}F_1^f(x_1,b;\mu,\zeta_1)F_2^{f'}(x_2,b;\mu,\zeta_2), \tag{1.2}$$

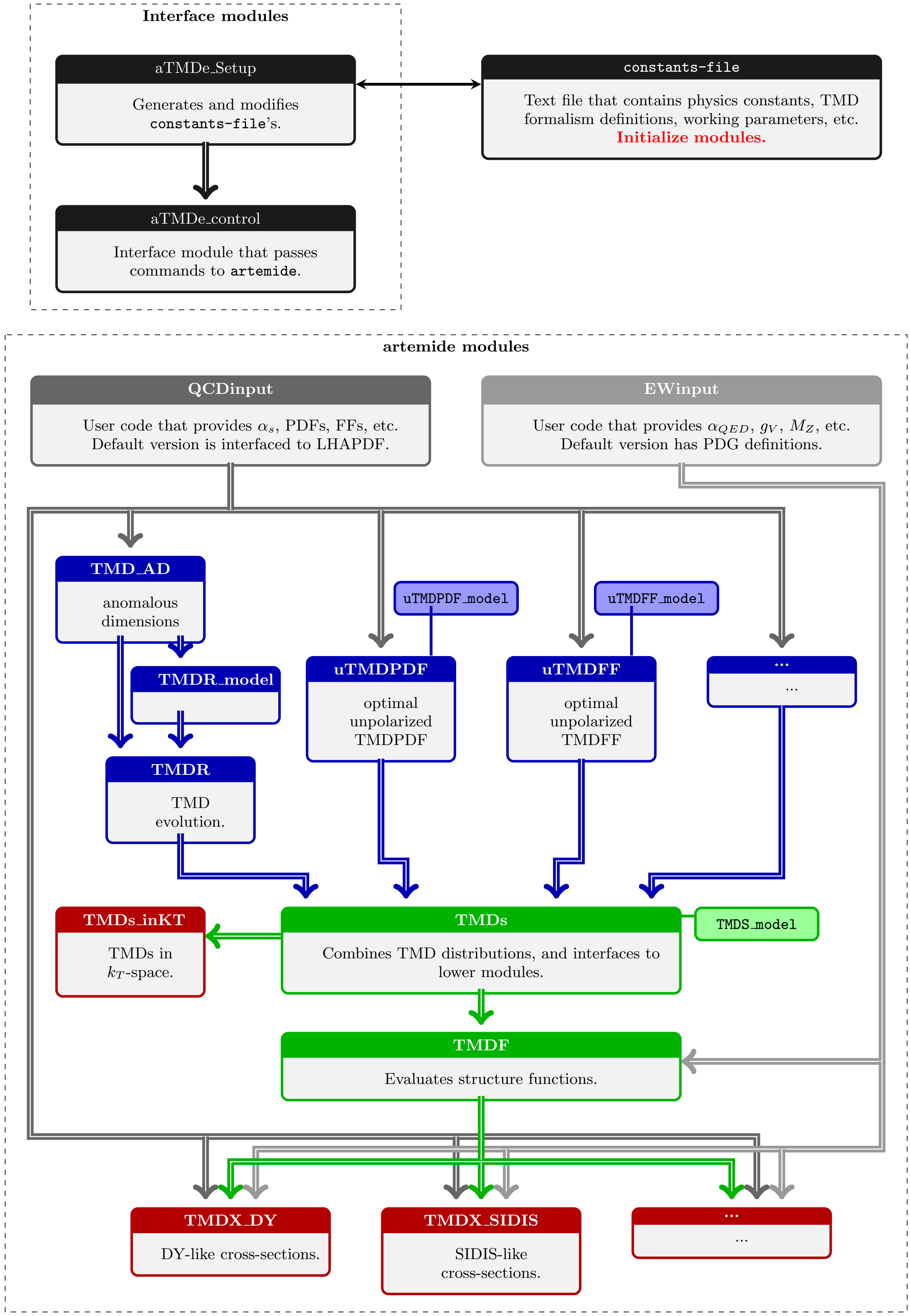


FIG. 1: Modules of `artemide`: purpose and dependencies

where $F_{1,2}$ are TMD distributions (of any origin and polarization), $z_{ff'}$ is the process dependent flavor mixing factor. The number n is also process dependent, e.g. for unpolarized observables it is $n = 0$, while for SSA's it is $n = 1$. Evaluation structure functions F is performed in the module `TMDF`. The evaluation of cross-sections is performed in the modules `TMDX`. The TMD distributions are evaluated by the module `TMDs` and related submodules.

> In this way, the computation of TMD-factorized cross-section can be naturally split into ordered parts. Each part is evaluated in corresponding module of `artemide`. See example, of evaluation scheme in fig.2.

C. User defined functions and options

> The `artemide` package has been created such that it allows to control **each aspect** of TMD factorization theorem. The TMD factorization has a large number of free, and "almost-free" parameters. It is a generally difficult task to provide a convenient interface for all these inputs. I do my best to make the interface convenient; however, some parts (e.g., setup of f_{NP}) could not be simpler (at least within FORTRAN). Also, take care that `artemide` is evolving, and I try to keep back compatibility, but it is not the main option.

> Starting from ver.2.0, `artemide` uses the text initialization file, which contains all required information on static parameters for a given setup. Throughout the text I call this file `constants-file`.

The user has to provide (or **use the default values**) the set of parameters, that control various aspects of evaluation. It includes PDF sets, f_{NP}, perturbative scales, parameters of numerics, non-QCD inputs, etc. There are three input sources for statical parameters.

General parameters: These are working parameters of `artemide`, such as amount of output, tolerance of integration routines, number of NP parameters, type of used evolution, griding parameters, triggering of particular contributions, etc. There are many of them, and typically they are unchanged. These are set in `constants-file`. **Changes do not require recompilation.**

External physics input: It includes the definition of α_s, collinear PDFs, and other distributions. Twist-2 distributions are taken from LHAPDF [5], with routines defined in QCDinput module. For non-QCD parameters, e.g. α_{QED}, SM parameters, there is a module `EWinput`. These are set in `constants-file`. **Changes do not require recompilation.**

NP model: The NP model consists in NP profiles of TMD distributions, NP model for large-b evolution, selection of scales μ, etc. These parameter and functions enter nearly each low level module. The code for corresponding functions is provided by user, in appropriate files, which are collected in the subdirectory `src/Model`. The name of files are shown on diagram in colored blobs adjusted to the related module. **Changes require recompilation.**

Comments:

- **IMPORTANT:** Each module is initialized individually via `constants-file`. So, each module can be used independently on the full package, given proper section of `constants-file` and submodules (see diagram). However, unless you understand what is going on, it is recommended to use `aTMDe_setup` module for creation of `constant-file`, and `aTMDe_control` module for proper control, initialization and operations of sub-modules.
- `constants-file` can be saved and used in future to reproduce setup. I try to keep compatibility between these files.
- `constants-file` is created and modified within `aTMDe_setup` module. It could be also modified manually.
- NP functions are typically defined with a number of numeric parameters. The value of these parameters could be changed without restart (or recompilation) of the `artemide` by appropriate command. E.g. (`call TMDs_SetNPParameters(lambda)`) on the level of `TMDs` module. See sections of corresponding modules.
- The number of parameters in the model for each module is set in `constants-file`.

- The directory `/Model` together with `constants-file` are convenient to keep as they are. They contain full information about particular evaluation, and thus results can be always reproduced (at least within the same version of `artemide`). I provide results of our extraction as such directories.
- Before ver.2.0, the interface was different and chaotic.

D. Installation

Download and unpack `artemide`. The actual code is in the /src. Check the `makefile`. **You must provide options `FC` and `FOPT`, which are defined in the top of it.** `FC` is the FORTRAN compiler, `FOPT` is additional options for compiler (e.g. linking to LHAPDF library).

There is no actual installation procedure, there is just compilation. If model, inputs, etc, are set correctly (typical problem is linking to LHAPDF, be sure that it is installed correctly), then `make` compiles the library. The result are object files (`*.o`) (which are collected in `/obj`) and module files (`*.mod`) (which are collected in `/mod`).

The test of current compilation could be performed by `make test`. It compiles program `test.f90` from /Prog and run it (default test uses `NPDF31_nnlo_as_0118` set from LHAPDF, check that it is present in your LHAPDF installation). Program `test` runs some elementary code with minimum input. Output is shown later

Next, do your code, `include` appropriate modules of `artemide`, and compile it together with object-files (do not forget to add proper references to module files `-I/mod`). It should work! Linking could be done automatically if you call for `make program TARGET=...`, where ... is the name of the file with the code.

```
 artemide.control: initialization done.
uTMDPDF: Grid is built  (  250 x  750)  calc.time=  0.53s.
Calculating some values for cross-section one-by-one (DY around Z-boson peak, ATLAS 8TeV kinematics)
ptMin   --       ptMax           xSec
  1.0000000000000000       --    3.0000000000000000         48.499041273610260
  3.0000000000000000       --    5.0000000000000000         57.189916866533792
  5.0000000000000000       --    7.0000000000000000         49.321931214709110

Now the same by list
It must be faster since you use OPENMP
result:    48.499041273610260         57.189916866533792         49.321931214709110
-----------------------------------------------------------------------------------------
The programm evaluation took    8.8279999999999994       sec. (CPU time)
The programm evaluation took    6.1287177319172770       sec. (wallclock time)

If you do not like so many terminal messages check the parameter outputlevel in constants file.
Not forget to cite artemide [1706.01473]
-----------------------------------------------------------------------------------------
```

E. Python interface: `harpy`

For simplicity of data analysis the `artemide` has a python interface, called `harpy` (linking is made by f2py library). It is not possible to interfacing the `artemide` directly to python since `artemide` is made on fortran95. `Artemide` uses some features of Fortran95, such as interfaces, and indirect list declarations, which are alien to python. Also I have not found any convenient way to include several dependent Fortran modules in f2py (if you have suggestion just tell me). Therefore, I made a wrap module `harpy.f90` that call some useful functions from `artemide` with simple declarations. It contains limited set of functions useful for phenomenology, and definitely cannot replace the FORTRAN interface for deep studies.

The compilation of `harpy` is slightly more complicated.

1. In the makefile check the variable `Fpath` (in the top part of the file). Put-in the full path for fortran compiler. It is needed by f2py.
2. (optional, does not work on Mac (?)) Run `make harpy-signature` . This will create an interface file (`artemide.pyf` for all functions in the `harpy.f90`. This file is already provided in the distribution, so if you did not change `harpy.f90`, you better skip this step. For some reason, it does not work on Mac.
3. Run `make harpy`. It compiles `harpy.f90` with interface `artemide.pyf` linking to `artemide`. The result is `artemide.so`.

All files are in /harpy. Link it to python and work. There is also extra python `harpy.py` which has several most important function.

F. Constants file and version compatibility

Constants files contains the list of work-flow parameters, such as physical constants, LHA grid names, numerical setup, etc. Each constants file has version number (does not coincide with the version of artemide).

On the module initialization the constants file is read, parameter are setup. Thus, the version of constants file should be adjusted to version of artemide *if you run modules separately.*

The workflow is deferent of the initialization of `artemide` is made by the `artemide_control`. In this case, the `artemide` creates a copy of the initial constants-file (`aTMDe_temporary`), and (in the case the versions do not match) fill the absent options by default parameter. After it the lower-level modules are initialized by `aTMDe_temporary`. Naturally, in this case the versions should match (if not something is wrong with installation).

> You can update the `constants-file` to an actual version by
> `make update TARGET=...`
> where ... is the path to `constants-file` to be updated. The updated version will have all content of the original file, + new options setup by default setting.

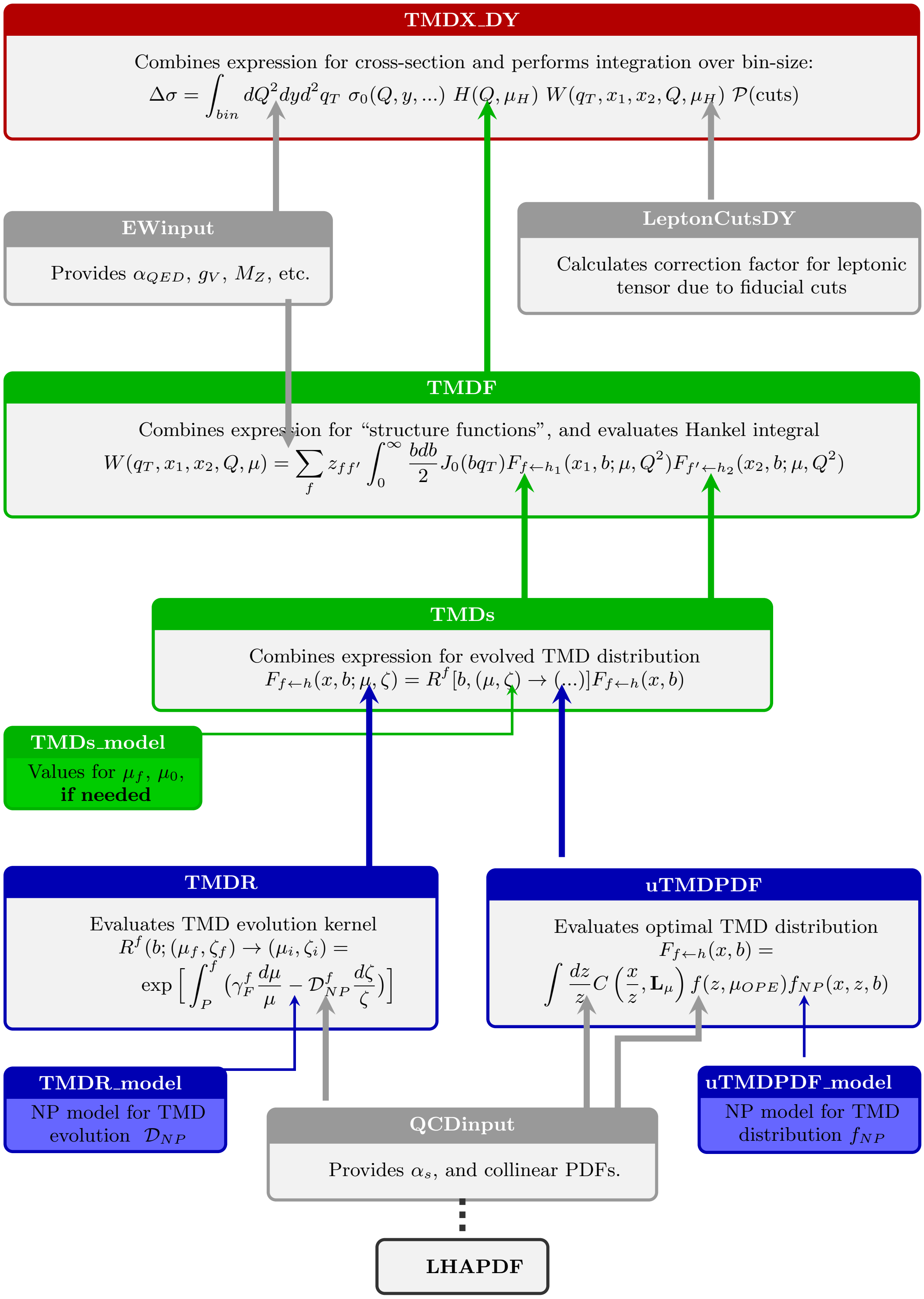


FIG. 2: Evaluation of DY cross-section by `artemide`

II. THEORY

This section is under construction.

In this section, the details of theoretical input coded in `artemide` is given. This section does not pretend to be a comprehensive review of TMD factorization. Only superficial details and references to particular realizations are given. For detailed and accurate description of the theory see specialized literature.

A. Definition of TMD distributions

TO BE WRITTEN. In this section, there will be operator definition of TMD distributions, and details on way the perturbative matching and NP-medeling is realized in `artemide`.

$$F_f(x,b)=\int_x^1 \frac{dz}{z}C_{f\leftarrow f'}(z,b^*,c_4\mu_{\rm OPE})f_{f'}(\frac{z}{x},c_4\mu_{\rm OPE})f^f_{NP}(x,z,b,\{\lambda\}), \tag{2.1}$$

where $f_f(x,\mu)$ is PDF of flavor f, C is the coefficient function in ζ-prescription, f_{NP} is the non-perturbative function. The variable c_4 is used to test the scale variation sensitivity of the TMD PDF. The NNLO coefficient functions used in the module were evaluated in [6] (please, cite it if use).

PLAN TO ADD

- Definition of TMD distributions, operators, Lorenz structures, coordinate and momentum space.
- Perturbative matching, and ζ-prescription
- NP-modeling

B. Evolution of TMD distributions

The detailed theory is given in the article [4]. The NLO rapidity anomalous dimension has been evaluated in [8]. The NNLO rapidity anomalous dimension has been evaluated in [9, 10].

The evolution of TMD distribution of any kind is given by the following pair of equations

$$\mu^2\frac{d}{d\mu^2}F_{f\leftarrow h}(x,b;\mu,\zeta) = \frac{\gamma_F^f(\mu,\zeta)}{2}F_{f\leftarrow h}(x,b;\mu,\zeta), \tag{2.2}$$

$$\zeta\frac{d}{d\zeta}F_{f\leftarrow h}(x,b;\mu,\zeta) = -\mathcal{D}^f(\mu,b)F_{f\leftarrow h}(x,b;\mu,\zeta), \tag{2.3}$$

where $F_{f\leftarrow h}$ is the TMD distribution (TMDPDF or TMDFF) of the parton f in hadron h. The function $\gamma_F(\mu,\zeta)$ is called the TMD anomalous dimension and contains both single and double logarithms. The function $\mathcal{D}(\mu,b)$ is called the rapidity anomalous dimension. TMD and rapidity anomalous dimensions have not unified notation in the literature, for comparison of notation see table I in ref.[4]. The only important quantum number for TMDs is the color representation the initiating parton, which is tied to the parton flavor, namely, quark (fundamental representation) or gluon (adjoint representation). However, as the TMD evolution does not mix the flavors and for simplicity of notation, we omit the flavor index f in this section, unless it is important.

The uniqueness of solution for the coupled system (2.2)-(2.3) is guaranteed by *the integrability condition*

$$\zeta\frac{d}{d\zeta}\gamma_F(\mu,\zeta)=-\mu\frac{d}{d\mu}\mathcal{D}(\mu,b), \tag{2.4}$$

The mutual dependence can be worked out explicitlydue to the fact that the ultraviolet divergences of the TMD operator partially overlap with the rapidity divergences, (see e.g.[2, 10]),

$$\zeta\frac{d}{d\zeta}\gamma_F(\mu,\zeta)=-\Gamma(\mu), \tag{2.5}$$

$$\mu\frac{d}{d\mu}\mathcal{D}(\mu,b)=\Gamma(\mu), \tag{2.6}$$

where Γ is the (light-like) cusp anomalous dimension. The equation (2.5) entirely fixes the logarithm dependence of the TMD anomalous dimension, which reads

$$\gamma_F(\mu,\zeta) = \Gamma(\mu)\ln\left(\frac{\mu^2}{\zeta}\right) - \gamma_V(\mu). \tag{2.7}$$

The anomalous dimension γ_V refers to the finite part of the renormalization of the vector form factor. In contrast, the equation (2.6) cannot fix the logarithmic part of $\mathcal{D}$ entirely, but only order by order in perturbation theory, because the parameter μ is also responsible for the running of the coupling constant.

The solution of eq. (2.2)-(2.3) can be written as

$$F(x,b;\mu_f,\zeta_f) = R[b;(\mu_f,\zeta_f)\to(\mu_i,\zeta_i)]F(x,b;\mu_i,\zeta_i), \tag{2.8}$$

where R is the TMD evolution factor. The general form of the evolution factor is

$$R[b;(\mu_f,\zeta_f)\to(\mu_i,\zeta_i)] = \exp\left[\int_P\left(\gamma_F(\mu,\zeta)\frac{d\mu}{\mu} - \mathcal{D}(\mu,b)\frac{d\zeta}{\zeta}\right)\right], \tag{2.9}$$

where (μ_f,ζ_f) and (μ_i,ζ_i) refer respectively to a final and initial set of scales. Here, the $\int_P$ denotes the line integral along the path P in the (μ,ζ)-plane from the point (μ_f,ζ_f) to the point (μ_i,ζ_i). The integration can be done on an arbitrary path P, and the solution is independent on it, thanks to the integrability condition eq. (2.4).

The TMD evolution factor R obeys the transitivity relation

$$R[b;(\mu_1,\zeta_1)\to(\mu_2,\zeta_2)] = R[b;(\mu_1,\zeta_1)\to(\mu_3,\zeta_3)]R[b;(\mu_3,\zeta_3)\to(\mu_2,\zeta_2)], \tag{2.10}$$

where (μ_3,ζ_3) is arbitrary point in (μ,ζ)-plane and the point inversion property

$$R[b;(\mu_1,\zeta_1)\to(\mu_2,\zeta_2)] = R^{-1}[b;(\mu_2,\zeta_2)\to(\mu_1,\zeta_1)]. \tag{2.11}$$

These equations are the cornerstones of the evolution mechanism, since they allow an universal definition of the non-perturbative distributions and the comparison of different experiments.

Different realizations of TMD evolution (if they are compatible with the factorization theorem) can be casted into selected of different paths of integration ADD PICTURE. All choices are equivalent, up to next-to-given order perturbative corrections, treatment of NP parts, and implementation of matching for TMD distributions. In `artemide` sevaral paths are presented, however, the matching of TMD distribution to collinear distributions is done in ζ-prescription.

1. *Equipotential lines & ζ-prescription*

The ζ-prescription is based on consideration of equipotential lines for evolution equations (2.2)-(2.3) (or lines of null-evolution, for description of evolution potential and its properties see [4]). An equipotential line $\ell(b) = (\mu,\zeta)$ is determined by values of anomalous dimensions γ_F and $\mathcal{D}$ at given b. The equation for line $\ell(b)$ can be written in the form $\zeta = \zeta(\mu,b)$. In this parameterization the for function $\zeta(\mu,b)$ is

$$-\Gamma(\mu)\ln\left(\frac{\zeta(\mu,b)}{\mu^2}\right) - \gamma_V(\mu) = 2\mathcal{D}(\mu,b)\frac{d\ln\zeta(\mu,b)}{d\ln\mu^2}. \tag{2.12}$$

Among set of equipotential lines there is a single special equipotential line, which passes thorough the saddle point of the evolution field. The saddle point $(\mu_{\text{saddle}},\zeta_{\text{saddle}})$ is defined by the equations

$$\mathcal{D}(\mu_{\text{saddle}},b) = 0, \qquad \gamma_V(\mu_{\text{saddle}},\zeta_{\text{saddle}}) = 0. \tag{2.13}$$

Note, that the value of μ_{saddle} depends on value of b, and on NP model for $\mathcal{D}$. In general it could be found only numerically. In typical models μ_{saddle} is decreasing monotonous function of b. At certain large values of b it passes though the Landau pole, and its value could not be determined. This line we denote as

$$\textbf{special equipotential line} = \zeta_\mu(b), \qquad (\mu_{\text{saddle}},\zeta_{\text{saddle}}) \in \zeta_\mu(b). \tag{2.14}$$

By definition a TMD distribution is the same for all points of equipotential line. In particular it means that there is no dependence on variable μ, since a variation of μ is compensated by appropriate variation in $\zeta(\mu)$.

Important note: The presence of NP part in $\mathcal{D}$ makes determination of special line involved. We distinguish three "realizations" of the special line. The exact which is determined by equation (2.12) and boundary conditions (2.14) (its values can be calculated by function `zetaSL`). The perturbative realization where NP part is dropped (its values can be calculated by functions `zetaMUperp` and `zetaMUresum`). And user-defined which is given by user in `TMDR_model.f90` file.

2. Exact solution for evolution to special line

There is a possibility to define the evolution to the optimal equipotential line in perturbation theory for any NP input, at any b. Instead of problematic variable b one uses the function $\mathcal{D}(\mu, b)$ as a variable. With the ansatz $\zeta_\mu = \mu^2 \exp(-g(\mu, b))$ we rewrite equation (2.12) as

$$2\mathcal{D}(\mu, b)\left(1 - \frac{dg(\mu, b)}{d\ln\mu^2}\right) - \Gamma(\mu)g(\mu, b) + \gamma_V(\mu) = 0. \tag{2.15}$$

or

$$2\mathcal{D}\left(1 - \frac{\partial g(\mu, \mathcal{D})}{\partial\ln\mu^2} - \frac{\Gamma(\mu)}{2}g'(\mu, \mathcal{D})\right) - \Gamma(\mu)g(\mu, \mathcal{D}) + \gamma_V(\mu) = 0, \tag{2.16}$$

where $g' = dg/d\mathcal{D}$. The boundary condition (2.13) is transformed to the condition that g is regular at $\mathcal{D} \to 0$. Note, that this condition does not depends on b and thus determines the function unambiguously at any b, even if saddle point is behind the Landau-pole values.

The equation (2.16) is simplified further with introduction of new function $\tilde{g}(a_s, \mathcal{D}) = \mathcal{D}g(\mu, \mathcal{D})$:

$$2\mathcal{D} + 2\beta(a_s)\frac{\partial\tilde{g}(a_s, \mathcal{D})}{\partial a_s} - \Gamma(a_s)\tilde{g}'(a_s, \mathcal{D}) + \gamma_V(a_s) = 0, \tag{2.17}$$

with boundary condition $\tilde{g}(a_s, 0) = 0$. The general solution reads

$$\tilde{g}(a_s, \mathcal{D}) = -\frac{\mathcal{D}}{2}\int^{a_s}\frac{da}{\beta(a)} - \int^{a_s} da\frac{\gamma_V(a)}{2\beta(a)} - \int^{a_s} da\frac{\Gamma(a)}{2\beta(a)}\int^{a}\frac{da'}{\beta(a')} + \Phi\left(\mathcal{D} + \int^{a_s} da\frac{\Gamma(a)}{2\beta(a)}\right), \tag{2.18}$$

where Φ is a solution of the following transcendental equation

$$\Phi\left(\int^{a_s} da\frac{\Gamma(a)}{2\beta(a)}\right) = \int^{a_s} da\frac{\gamma_V(a)}{2\beta(a)} + \int^{a_s} da\frac{\Gamma(a)}{2\beta(a)}\int^{a}\frac{da'}{\beta(a')}. \tag{2.19}$$

Unfortunately, this equation is practically impossible to solve for higher then NNLL anomalous dimensions. It is straightforward to show that at large-$\mathcal{D}$ the solution is

$$\tilde{g}(a_s, \mathcal{D} \to \infty) = \mathcal{D}\int^{a_s}\frac{-1}{\beta(a)}da + ... \tag{2.20}$$

Note that asymptotic is defined up to a constant.

The function g is convenient to derive as perturbative expansion

$$g(\mu, \mathcal{D}) = \frac{1}{a_s(\mu)}\sum_{n=0}^{\infty} a_s^n(\mu)g_n(\mathcal{D}). \tag{2.21}$$

We have found

$$g_0 = \frac{e^{-p} + p - 1}{\beta_0 p}, \tag{2.22}$$

$$g_1 = g_0\left(\frac{\beta_1}{\beta_0} - \frac{\Gamma_1}{\Gamma_0}\right) + \frac{\gamma_1}{\gamma_0} - \frac{\beta_1}{2\beta_0^2}p, \tag{2.23}$$

$$g_2 = g_0\frac{\beta_2\Gamma_0 - \beta_1\Gamma_1}{\beta_0\Gamma_0} + \frac{\mathrm{ch}p - 1}{p}\frac{\beta_0\Gamma_1^2 - \beta_0\Gamma_0\Gamma_2 + \beta_1\Gamma_0\Gamma_1 - \beta_2\Gamma_0}{\beta_0^2\Gamma_0^2} + \frac{e^p - 1}{p}\frac{\Gamma_0\gamma_2 - \Gamma_1\gamma_1}{\Gamma_0^2}, \tag{2.24}$$

where $p = 2\beta_0\mathcal{D}/\Gamma_0$.

There is an additional problem that appears at NNNLO, namely, the coefficient g_3 is negative. It results to a singular solution for ζ_μ at large-$\mathcal{D}$. It indicates that the series has bad convergence properties. This problems occur for very large $\mathcal{D} > 0.9 - 1.1$. Currently to by-pass this problem the 4-loop exact ζ-line is not used (3-loop expression is used instead). For small-$\mathcal{D}$ the difference is negligible.

The solution for the evolution from $(\mu, \zeta) \to (\mu, \zeta_\mu)$ is

$$R[b; (\mu, \zeta) \to (\mu, \zeta_\mu)] = \exp\left(-\mathcal{D}(\mu)\ln\left(\frac{\zeta}{\mu^2}\right) - \mathcal{D}(\mu, b)g(\mu, \mathcal{D})\right). \tag{2.25}$$

C. Expressions for evolution functions in `artemide`

Here are the expression for various evolution functions how they are written in `artemide` in `TMDR`-module.

1. *Fixed order RAD:*

> `Dpert(mu,bT,f)`: returns $\mathcal{D}^f(b,\mu)$ [real(dp)]
> `mu, bT` [real(dp)] scales in GeV, `f`[integer] flavour.
> Order is controlled by `orderD` ($= n_{\max}$). At `orderD`=0, `Dpert`=0.

Solving (2.6) we get

$$\mathcal{D}(b,\mu) = \sum_{n=1}^{\infty} a_s^n \sum_{k=0}^{n} \mathbf{L}_\mu^k d^{(n,k)}, \tag{2.26}$$

where $d^{(n,0)}$ are obtained by calculation. The coefficients $d^{(n,k)}$ are

$$\begin{aligned}
d^{(1,1)} &= \frac{\Gamma_0}{2},\\
d^{(2,2)} &= \frac{\Gamma_0\beta_0}{4},\\
d^{(2,1)} &= \frac{\Gamma_1}{2},\\
d^{(3,3)} &= \frac{\Gamma_0\beta_0^2}{6},\\
d^{(3,2)} &= \frac{\Gamma_0\beta_1+2\Gamma_1\beta_0}{4},\\
d^{(3,1)} &= \frac{\Gamma_2+4\beta_0 d^{(2,0)}}{2},
\end{aligned}$$

where flavor on both sides is the same.

2. *Resummed RAD:*

> `Dresum(mu,bT,f)`: returns $\mathcal{D}^f_{\text{resum}}(b,\mu)$ [real(dp)]
> `mu, bT` [real(dp)] scales in GeV, `f`[integer] flavour.
> Order is controlled by `orderDresum` ($= n_{\max}$). At `orderDresum`=0, `Dresum`=$-\frac{\Gamma_0}{2\beta_0}\ln(1-X)$.

One can resumm the logarithms in this expression and write it as

$$\mathcal{D}_{\text{resum}} = -\frac{\Gamma_0}{2\beta_0}\Big[\ln(1-X) + \sum_{n=1}^{\infty}\frac{a_s^n}{(1-X)^n}\sum_{k,l=0}^{n} X^k \ln^l(1-X) d_r^{(n,k,l)}\Big], \qquad X = \beta_0 a_s \mathbf{L}_\mu, \tag{2.27}$$

the coefficients d_r are

$$\begin{aligned}
d_r^{(1,0,1)} &= B_1, \qquad d_r^{(1,1,0)} = B_1 - G_1, \qquad d_r^{(1,0,0)} = 0,\\
d_r^{(2,0,2)} &= -\frac{B_1^2}{2}, \qquad d_r^{(2,0,1)} = B_1 G_1,\\
d_r^{(2,2,0)} &= \frac{1}{2}\left(G_2 - B_1G_1 - B_2 + B_1^2\right) \qquad d_r^{(2,1,0)} = B_1G_1 - G_2,\\
d_r^{(2,0,0)} &= -2\frac{d^{(2,0)}\beta_0}{\Gamma_0}, \qquad d_r^{(2,2,2)} = d_r^{(2,1,1)} = 0,
\end{aligned}$$

$$
\begin{aligned}
d_r^{(3,0,3)} &= \frac{B_1^3}{3}, \qquad d_r^{(3,0,2)} = -\frac{B_1^3}{2} - B_1^2 G_1, \qquad d_r^{(3,0,1)} = B_1 G_2 + 4\frac{d^{(2,0)}\beta_1}{\Gamma_0},\\
d_r^{(3,1,1)} &= B_1 B_2 - B_1^3, \qquad \text{other } d_r^{(3,k>0,l>0)} = 0,\\
d_r^{(3,3,0)} &= \frac{1}{3}\left(B_1^3 - 2B_1B_2 + B_3 - B_1^2G_1 + B_2G_1 + B_1G_2 - G_3\right),\\
d_r^{(3,2,0)} &= -\frac{1}{2}B_1^3 + B_1B_2 - \frac{B_3}{2} + B_1^2G_1 - B_2G_1 - B_1G_2 + G_3,\\
d_r^{(3,1,0)} &= B_1G_2 - G_3, \qquad d_r^{(3,0,0)} = -2\frac{d^{(3,0)}\beta_0}{\Gamma_0}
\end{aligned}
$$

Here,

$$
G_i = \frac{\Gamma_i}{\Gamma_0}, \qquad B_i = \frac{\beta_i}{\beta_0}
$$

3. Fixed order ζ-line:

`zetaMUpert(mu,bT,f)`: returns $\zeta^f(b,\mu)$ [real(dp)]
`mu, bT` [real(dp)] scales in GeV, `f`[integer] flavour.
Order is controlled by `orderZETA` ($=n_{\max}$). At `orderZETA`¡0, `zetaMUpert`=1.

The solution of (2.12) in PT can be written as

$$
\zeta^f(b,\mu) = \frac{C_0\mu}{b}e^{-v}, \qquad v = \sum_{n=0}^{\infty} a_s^n \sum_{k=0}^{n+1} \mathbf{L}_\mu v^{(n,k)}. \tag{2.28}
$$

The *perturbative boundary condition* is that v is regular at $\mathbf{L}\to 0$. The first coefficients are

$$
\begin{aligned}
&v^{(0,0)} = g_1, \qquad v^{(0,1)} = 0,\\
&v^{(1,2)} = \frac{\beta_0}{12}, \qquad v^{(1,1)} = 0, \qquad v^{(1,0)} = \mathbf{d}^{(2,0)} - g_1G_1 + g_2,\\
&v^{(2,3)} = \frac{\beta_0^2}{24}, \qquad v^{(2,2)} = \frac{\beta_0}{12}\left(B_1 + G_1\right), \qquad v^{(2,1)} = \frac{\beta_0}{2}\left(\frac{8}{3}\mathbf{d}^{(2,0)} - g_1G_1 + g_2\right),\\
&v^{(2,0)} = \mathbf{d}^{(3,0)} - \mathbf{d}^{(2,0)}G_1 + g_1G_1^2 - g_2G_1 - g_1G_2 + g_3,
\end{aligned}
$$

here

$$
\mathbf{d}^{(n,k)} = \frac{d^{(n,k)}}{\Gamma_0}, \qquad g_i = \frac{\gamma_V^{(i)}}{\Gamma_0}, \qquad G_i = \frac{\Gamma_i}{\Gamma_0}, \qquad B_i = \frac{\beta_i}{\beta_0}.
$$

4. Resummed ζ-line:

`zetaMUresum(mu,bT,f)`: returns $\zeta^f_{\text{resum}}(b,\mu)$ [real(dp)]
`mu, bT` [real(dp)] scales in GeV, `f`[integer] flavour.
Order is controlled by `orderZETA`.
REMOVED in v.2.03. TO BE REINTRODUCED LATER.

5. Exact ζ-line:

`zetaSL(mu,bT,f)`: returns $\zeta^{f}_{\text{exact}}(b,\mu)$ [real(dp)]
`mu, bT` [real(dp)] scales in GeV, `f`[integer] flavour.
Order is controlled by `orderZETA`. At `orderZETA`¡0, `zetaMUpert`=1.
Currently at the order =3, the result is NNLO. Due to instability of N^{3}LO at very large $\mathcal{D}$.

The solution for ζ can be written as

$$\zeta(b,\mu) = \mu^2 \exp\left(-g(a_s,\mathcal{D})\right). \tag{2.29}$$

The function g is convinient to present as

$$g(a_s,\mathcal{D}) = \frac{\Omega(a_s,p)}{a_s\beta_0}, \qquad p = \frac{2\beta_0\mathcal{D}}{\Gamma_0}, \tag{2.30}$$

where Ω is convenient to present in the following form

$$\Omega(a_s,p) = \sum_{n=0}^{\infty} a_s^n \Omega_n(p), \tag{2.31}$$

with

$$\begin{aligned}
&\Omega_0 = z_1, \qquad \Omega_1 = (B_1 - G_1)z_1 + g_1 - \frac{B_1}{2}p,\\
&\Omega_2 = \frac{B_2 - B_1G_1 + G_1^2 - G_2}{2}z_1 + \frac{-B_2 + B_1G_1 + G_1^2 - G_2 - 2G_1g_1 + 2g_2}{2}z_{-1} + g_2 - g_1G_1\\
&\Omega_3 = \frac{2B_3 + B_1^2G_1 - B_2G_1 - G_1^3 + 3G_1G_2 - B_1B_2 - B_1G_2 - 2G_3}{6}z_1\\
&+\frac{G_1 - B_1}{2}\left(-B_2 + B_1G_1 + G_1^2 - G_2 - 2G_1g_1 + 2g_2\right)z_{-1}\\
&\Big[\frac{-B_3 + B_1^2G_1 + 2B_2G_1 - 4G_1^3 + 6G_1G_2 - B_1B_2 + B_1G_2 - 2G_3}{12} + (2G_1 - B_1)\frac{G_1g_1 - g_2}{2} - \frac{G_2g_1 - g_3}{2}\Big]z_{-2}\\
&+G_1^2g_1 - G_2g_1 - G_1g_2 + g_3.
\end{aligned}$$

Here

$$B_i = \frac{\beta_i}{\beta_0}, \qquad G_i = \frac{\Gamma_i}{\Gamma_0}, \qquad g_i = \gamma^{(i)}\frac{\beta_0}{\Gamma_0}, \qquad z_n = \frac{e^{-np} - 1 + np}{p}. \tag{2.32}$$

Note that $\lim_{p\to 0} z_n \sim n^2p/2$.

III. ATMDE_SETUP MODULE

The module `aTMDe_setup` creates and modifies the `constants-file`. **It is a stand-alone module, which does not require the rest modules** (however, it is called by `aTMDe_control`).

List of commands optional parameters are shown in blue.

Command	Sec.	Short description
`artemide_Setup_Default(order)`	III A	The main command which initializes variables by default values corresponded to a particular order.
`artemide_Setup_fromFile(file,prefix,order)`	III A	The main command which initialize variables by pre-saved values in `file`. `prefix` is path to the file. `order` is the order of default version of parameters (used if versions of files are incompatible).
`CreateConstantsFile(file,prefix)`	-	Write a new `constants-file` according to current modification.
`CheckConstantsFile(file,prefix)`	-	Function (logical). Compare the version of `constants-file`. Returns `.true.` if version of file ¿= version of artemide, `.false.` otherwise.
`artemide_include(arg1,arg2,...,arg10)`	-	Include the modules into the initialization procedure. `arg` is (string) with the module name.
`Set_outputLevel(level,numMessages)`	-	Set the level of `artemide`-messages to (int)`level`. 0= only critical, 1=+warnings, 2=+module evaluation information, 3=+ details. Default=2. (integer)`numMessages` set number of non-critical Warnings of the same type to show (prevent spaming by same messages).
`Set_uPDF(hadron,setName,replica)`	V A	Assign the hadron for uPDF with number `hadron`(int) a PDF set `setName`(string) in LHAPDF library. It will be initialized in with replica `replica`(int)(default =0).
`Set_uFF(hadron,setName,replica)`	V A	Assign the hadron for uFF with number `hadron`(int) a FF set `setName`(string) in LHAPDF library. It will be initialized in with replica `replica`(int)(default =0).
`Set_lpPDF(hadron,setName,replica)`	V A	Assign the hadron for (unpolarized)PDF that is used by `lpTMDPDF` with number `hadron`(int) a PDF set `setName`(string) in LHAPDF library. It will be initialized in with replica `replica`(int)(default =0).
`Set_quarkMasses(mC,mB,mT)`	-	Set values for pole quark masses, charm, bottom and top (real), which determine N_f-thresholds. Default `mC`=1.4, `mB`=4.75, `mT`=173.
`Set_EWparameters(alphaInv,massZ,massW,widthZ,widthW,massH,widthH,vevHIGGS,sin2ThetaW,UD,US,UB,CD,CS,CB)`	-	Set parameters of electro-weak theory. `alphaInv`=inverse $\alpha_{QED}(M_Z)$. `UD,US,UB,CD,CS,CB` are elements of CKM matrix. Masses and widths are in GeV.
`Set_TMDR_order(order)`	IX C	Set the perturbative order of anomalous dimensions used for evolution. `order`=string(8)
`Set_TMDR_evolutionType(num)`	IX E	Set the type of evolution solution used to (int)`num`.
`Set_TMDR_lengthNParray(num)`	IX D	Set the length of λ_{NP} for $\mathcal{D}_{NP}$ to `num`(int).
`Set_uTMDPDF(hadron,setName,replica)`	V A XI	Assign the hadron for uTMDPDF with number `hadron`(int) a PDF set `setName`(string) in LHAPDF library. It will be initialized in with replica `replica`(int)(default =0). Automatically calls for `Set_uPDF`.
`Set_uTMDPDF_order(order)`	XI A	Set the perturbative order of convolution. `order`=string(8)
`Set_uTMDPDF_gridEvaluation(prepareGrid,includeGluon)`	XI E	Set the trigger to prepare the grid, and to include the gluons in the grid (default=.false.). Both `logical`.
`Set_uTMDPDF_lengthNParray(num)`	XI B	Set the length of λ_{NP} for uTMDPDF NP model to `num`(int).
`Set_uTMDFF(hadron,setName,replica)`	V A XII	Assign the hadron for uTMDFF with number `hadron`(int) a FF set `setName`(string) in LHAPDF library. It will be initialized in with replica `replica`(int)(default =0). Automatically calls for `Set_uFF`.

`Set_uTMDFF_order(order)`	XI A	Set the perturbative order of convolution. `order`=string(8)
`Set_uTMDFF_gridEvaluation (prepareGrid,includeGluon)`	XI E	Set the trigger to prepare the grid, and to include the gluons in the grid (default=.false.). Both `logical`.
`Set_uTMDFF_lengthNParray(num)`	XI B	Set the length of λ_{NP} for uTMDFF NP model to `num`(int).
`Set_lpTMDPDF(hadron,setName,replica)`	V A XV	Assign the hadron for lpTMDPDF with number `hadron`(int) a PDF set `setName`(string) in LHAPDF library. It will be initialized in with replica `replica`(int)(default =0). Automatically calls for `Set_lpPDF`.
`Set_lpTMDPDF_order(order)`	XI A	Set the perturbative order of convolution. `order`=string(8)
`Set_lpTMDPDF_gridEvaluation (prepareGrid,includeGluon)`	XI E	Set the trigger to prepare the grid, and to include the gluons in the grid (default=.true.). Both `logical`.
`Set_lpTMDPDF_lengthNParray(num)`	XI B	Set the length of λ_{NP} for lpTMDPDF NP model to `num`(int).

A. aTMDe_Setup

TO BE WRITTEN

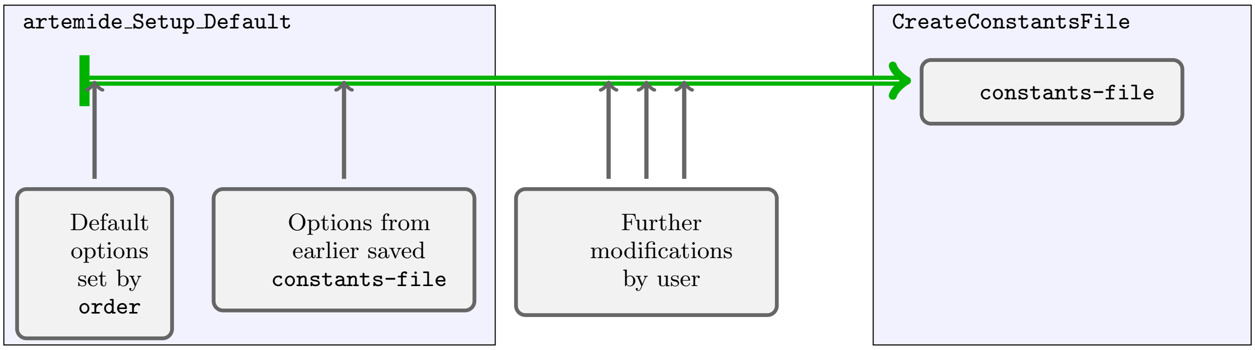


FIG. 3: Scheme of creation and modification of `constants-file` by `aTMDe_setup`.

IV. ATMDE_CONTROL MODULE

The module `aTMDe_setup` is used to coordinate the operation of other modules. It does not bring any new features, just operates other modules in proper order. It is only for convenience.

List of commands optional parameters are shown in blue.

Command	Sec.	Short description
`artemide_Initialize(file,prefix,order)`	III A	The command which initialize modules according to `constants-file` created by `artemide_Setup_fromFile(file,prefix,order)` .
`artemide_ShowStatistics()`		Shows some information.
`artemide_SetNPparameters(lambdaNP)`	IV A	Receive a (real*8)list of NP parameters, split it according to current setup and passes NP parameters to appropriate modules. Reset modules counters.
`artemide_SetNPparameters_TMDR(lambdaNP)`	IV A	Reset NP parameters of `TMDR`-module by (real*8) list `lambdaNP`. Reset modules counters.
`artemide_SetNPparameters_uTMDPDF(lambdaNP)`	IV A	Reset NP parameters of `uTMDPDF`-module by (real*8) list `lambdaNP`. Reset modules counters.
`artemide_SetNPparameters_uTMDFF(lambdaNP)`	IV A	Reset NP parameters of `uTMDFF`-module by (real*8) list `lambdaNP`. Reset modules counters.
`artemide_SetNPparameters_lpTMDFF(lambdaNP)`	IV A	Reset NP parameters of `lpTMDFF`-module by (real*8) list `lambdaNP`. Reset modules counters.
`artemide_SetReplica_TMDR(num)`	IV A	Reset NP parameters of `TMDR`-module by values corresponding to replica (int)`num`. Reset modules counters.
`artemide_SetReplica_uTMDPDF(num)`	IV A	Reset NP parameters of `uTMDPDF`-module by values corresponding to replica (int)`num`. Reset modules counters.
`artemide_SetReplica_uTMDFF(num)`	IV A	Reset NP parameters of `uTMDFF`-module by values corresponding to replica (int)`num`. Reset modules counters.
`artemide_SetReplica_lpTMDPDF(num)`	IV A	Reset NP parameters of `lpTMDPDF`-module by values corresponding to replica (int)`num`. Reset modules counters.
`artemide_SetScaleVariations(c1,c2,c3,c4)`	-	Change the value of scale-variation constants $c_1 - c_4$.
`artemide_GetReplicaFromFile(file,rep,array)`	-	Read the .rep file (path is `file`), and search for the replica (int)`rep`. Return the (real*8,allocatable)`array` of NP parameters. **This command can work without initialization of artemide-control**, in this case, no check of consistency is performed and warning raised.
`artemide_NumOfReplicasInFile(file)`	-	Read the .rep file (path is `file`), and return the number of replicas saved in it.

A. Passing non-perturbative parameters

The important part of the initialization is the number of NP parameters for each TMD distributions under consideration. Each TMD-evaluating module (say, `uTMDPDF`, `uTMDFF`, etc.) requires n_i number of parameters. The numbers are specified in `constants-file`. The number must be greater then zero $n_i > 0$ for any used module, i.e. f_{NP} is at least 1-parametric (if it is not so, just do not use the parameter in the definition of f_{NP}, but keep $n_i > 0$). These numbers are read during the initialization procedure, and allocate the memory.

The set of particular values of these parameters can be done by several ways.

Option I: Set all values in a single call
`call artemide_SetNPParameters(lambda)`
where

$\{\lambda_i\}$ real*8(1:$\sum_i n_i$) The set of parameters which define the non-perturbative functions f_{NP} within modules. It is split into parts and send to corresponding modules. I.e. lambda(1:n_0) $\rightarrow$`uTMDR`, lambda(n_0+1:n_0+n_1) $\rightarrow$`uTMDPDF`, lambda(n_0+n_1+1:$n_0+n_1+n_2$) $\rightarrow$`uTMDFF` (fixed order).

Option II: Set value for particular function. For it call `artemide_SetNPparameters_???(lambdaNP)`, where `???` is module name, e.g.
`artemide_SetNPparameters_uTMDPDF(lambdaNP)`
will set parameters for unpolarized TMDPDF.

Option III: You can use presaved values of λ_{NP} for a given function. They are provided by model file (if provided). To set NP-input given by integer `num`, call `artemide_SetReplica_???(num)`, where `???` is module name, e.g.
`artemide_SetReplica_uTMDPDF(num)`
will set parameters for unpolarized TMDPDF.

V. QCDINPUT MODULE

The module `QCDinput` gives an interface to external function provided by the user, such as PDF, FF, values of alpha-strong. It is completely user defined. In particular, in the default version it is linked to LHAPDF library [5].

List of available commands optional parameters are shown in blue.

Command	Description
`QCDinput_Initialize(file,prefix)`	Subroutine to initialize anything what is needed. (string) `file` is the name of `constants-file`, which contains initialization information. (string)`prefix` is appended to `file` if provided.
`As(Q)`	Returns the (real*8) value of $\alpha_s(Q)/(4\pi)$. Q is (real*8).
`xPDF(x,Q,hadron)`	Returns the (real*8(-5:5)) value of $xf(x,Q)$ for given hadron. `x`, `Q` are (real*8), `hadron` is (integer).
`xFF(x,Q,hadron)`	Returns the (real*8(-5:5)) value of $xd(x,Q)$ for given hadron. `x`, `Q` are (real*8), `hadron` is (integer).
`x_hPDF(x,Q,hadron)`	Returns the (real*8(-5:5)) value of $xg_1(x,Q)$ for given hadron (g_1=helicity distribution). `x`, `Q` are (real*8), `hadron` is (integer).
`mCHARM`	(real*8)Constant for mass of charm quark.
`mBOTTOM`	(real*8)Constant for mass of bottom quark.
`QCDinput_SetPDFreplica(num,h)`	Changes the unpolarized PDF replica number for hadron `h` to `rep`.
`QCDinput_SetFFreplica(num,h)`	Changes the unpolarized FF replica number for hadron `h` to `rep`.
`QCDinput_SetlpPDFreplica(num,h)`	Changes the replica of PDF associated with linearly polarized gluon PDF number for hadron `h` to `rep`.
`QCDinput_SethPDFreplica(num,h)`	Changes the replica of helicity PDF number for hadron `h` to `rep`.

A. Initialization

TOBE WRITTEN

VI. EWINPUT MODULE

`EWinput` module contains definitions of various physical parameters of elector-weak interactions, such as masses of Z and W bosons, CKM matrix, code for evolution of α_{QED} etc.
TOBE WRITTEN

A. α_{QED}

This implementation has been added in ver.2.03. In earlier versions the code for α_{QED} had a bug that generated slightly lower values (independently on the values of μ).
The QED coupling constant is set by the one-loop QED evolution with LO matching at thresholds. In order to tune it finer, the value of β_0 is modified by $\beta_0 \to \beta_0 + \delta\beta$, where $\delta\beta$ is found by exact implementation of two boundary values, $\alpha_{\text{QED}}(m_\tau)$ and $\alpha_{\text{QED}}(m_Z)$.
The LO evolution is

$$\alpha(\mu) = \frac{1}{\alpha_0^{-1} + 2\beta_0 \ln(\mu/\mu_0)}, \tag{6.1}$$

where $\beta_0 = -(3\pi)^{-1} N_{eff}$ with

$$N_{eff} = \sum_{\substack{\text{active}\\\text{leptons}}} 1 + N_c \sum_{\substack{\text{active}\\\text{quarks}}} e_q^2. \tag{6.2}$$

In artemide I account the following threasholds

μ	$\mu < m_e(\sim m_u, m_d)$	$\mu < m_\mu(\sim m_s)$	$\mu < m_c$	$\mu < m_\tau$	$\mu < m_b$	$\mu < m_t$	$\mu > m_t$
N_{eff}	0	$\frac{8}{3}$	4	$\frac{16}{3}$	$\frac{19}{3}$	$\frac{20}{3}$	8

So the actual formula for α_{QED} is

$$\alpha_{\text{QED}}(\mu) = \frac{1}{\alpha_Z^{-1} + 2\bar{\beta}\left[N_{eff}(\mu)\ln\left(\frac{\mu}{m_{th}}\right) + \sum_{m_i \in m_{th}} N_{eff} \ln\left(\frac{m_i}{m_{i+1}}\right)\right]}, \tag{6.3}$$

where m_{th} are all threasholds from μ till M_Z. The constant $\bar{\beta}$ is defined by exact matching at M_Z and m_τ. For default values $\bar{\beta}/\beta_0 \sim 1.002\%$, so it accounts for really tiny effects.

VII. TMD_AD MODULE

The module TMD_AD give access to various perturbative anomalous dimensions needed for TMD evolution. This module is essential part of TMDR module, and thus initialized by it.

List of available commands

Command	Sec.	Short description
`TMD_AD_Initialize(oC,oV,oD,oDr,oZ)`	IX C	Initialization of module. The (integer) parameters (`oC,oV,oD,oDr,oZ`) are orders of definition for anomalous dimensions (Γ_{cusp}, γ_V, $\mathcal{D}_{\text{pert}}$, $\mathcal{D}_{\text{resum}}$, ζ_{pert}).
`GammaCusp(mu,f)`		$= \Gamma^f_{\text{cusp}}(\mu)$
`gammaV(mu,f)`		$= \gamma^f_V(\mu)$
`Dpert(mu,bT,f)`		$= \mathcal{D}^f_{\text{pert}}(b,\mu)$
`Dresum(mu,bT,f)`		$= \mathcal{D}^f_{\text{resum}}(b,\mu)$
`zetaMUpert(mu,bT,f)`		$= \zeta^f_{\text{pert}}(b,\mu)$
`zetaSL(mu,rad,f)`		$= \zeta^f_{\text{exact}}(\mathcal{D},\mu)$ (Note, the argument.)
`RADEvolution(mu0,mu1,f)`		$= \int_{\mu_0}^{\mu_1} \frac{d\mu^2}{\mu^2} \frac{\Gamma(\mu)}{2}$ The evolution constant for $\mathcal{D}$

VIII. TMDR_MODEL

This module gives the expression for $\mathcal{D}$ and ζ_μ. It could be written by user, and one of modules that form a TMD model (together with models for TMD-distributions).
Making this module follow the interface

```
public:: ModelInitialization(NParray)
public:: ModelUpdate(NParray)
real(dp),public:: DNP(mu,b,f)
real(dp),public:: zetaNP(mu,b,f)
public:: GetReplicaParameters(num,NParray)
```

`ModelInitialization(NParray)`: this subroutine is called during the initialization of `TMDR`. The `real(dp),intent(in)::NParray(:)` is the array of initial NP-parameters. Use this subroutine to make preparation of the model.

`ModelUpdate(NParray)`: this subroutine is called during the `TMDR_setNPparameters(NParray)`. The `real(dp),intent(in)::NParray(:)` is the array of NP-parameters passed to `TMDR_setNPparameters`. Use this subroutine to recompute model parameters (if needed), save NP-parameters (if needed), etc.

`DNP(mu,b,f)`: the function returns the value of RAD, $\mathcal{D}^f(\mu, b)$. It contains both perturbative and NP parts. The perturbative expressions could be get from `TMD_AD`, see `Dpert` and `Dresum`.

`zetaNP(mu,b,f)`: the function returns the value of the special equipotential line, $\zeta^f(\mu, b)$. To define it, you can use perturbative and exact expressions from `TMD_AD`, see `zetaMUpert` and `zetaSL`.

`GetReplicaParameters(num,NParray)`: subroutine returns `real(dp),allocatable,intent(out)::NParray(:)` associated with `integer::num`. In such way, one could call replicas of a model.

Important: the module `TMDR_model` is compiled before `TMDR`, and thus can use only lower-level modules: `TMD_AD`, `QCDinput`, etc.

IX. TMDR MODULE

The module `TMDR` performs the evaluation of the TMD evolution kernel in the (μ, ζ)-plane.

List of available commands

Command	Sec.	Short description
`TMDR_Initialize(file,prefix)`	IX C	Initialization of module. (string) `file` is the name of `constants-file`, which contains initialization information. (string)`prefix` is appended to `file` if provided.
`TMDR_setNPparameter(...)`	IX D	Set new NP parameters used in `DNP` and `zetaNP`.
`TMDR_CurrentNPparameters(var)`	IX D	Return the (real*8) array of current values of NP parameters.
`TMDR_R(...)`	IX E	Evolution kernel from (μ_f, ζ_f) to (μ_i, ζ_i).
`TMDR_Rzeta(...)`	IX E	Evolution kernel from (μ_f, ζ_f) to $(\mu_i, \zeta_{\mu_i}^{\text{NP}})$ (Special Line is defined by `zetaNP`).
`LowestQ()`	IX F	Returns the values of Q (and the band) for which the evolution inverts.

List of inputs

Input	Setup by	Short description
`DNP(mu,b,f)`	write-in	NP-model for $\mathcal{D}_{\text{NP}}$. Should follow $\mathcal{D}^{\text{pert}}$ at small values of b and satisfy evolution equation (2.6).
`zetaNP(mu,b,f)`	write-in	NP-model for ζ_μ. Should follow equipotential at small values of b.
`NPparam`	`TMDR_setNPparameter(input)`	NP parameters used in `DNP` and `zetaNP`.
a_s	defined in `QCDinput`	Strong coupling. See sec.V

There are functions to help with definitions of `DNP` and `zetaNP`: `Dpert(mu,bT,f)`, `Dresum(mu,b,f)`, `zetaMUpert(mu,bt,f)`,`zetaMUresum(mu,b,f)`, `zetaMUresum4(mu,b,f)`. ADD EXPLANATION ON FUNCTIONS

A. Types of evolution

There are plenty of implementation of TMD evolution. In exact perturbation theory they are equivalent, however, in truncated there are tiny differences. In `artemide` there are following types (for detailed definitions see [4])

- **Type 1**: The improved-$\mathcal{D}$evolution. Exactly equivalent to CSS evolution. Requires (model-)definition of scales μ_0 and μ_{LOW}. Requires definition of ζ_μ-line.

FORMULA TO BE WRITTEN (9.1)

- **Type 2**: The improved-γ evolution. Requires (model-)definition of scale μ_{LOW}. Requires definition of ζ_μ-line.

FORMULA TO BE WRITTEN (9.2)

- **Type 3**: Evolution along ζ is with fixed-μ solution, evolution along μ along equi-potential line. Requires definition of ζ_μ-line.

$$R[b, (\mu_1, \zeta_1) \to (\mu_2, \zeta_2)] = \frac{R[b, (\mu_1, \zeta_1) \to (\mu_1, \zeta_{\mu_1})]}{R[b, (\mu_2, \zeta_2) \to (\mu_2, \zeta_{\mu_2})]}, \tag{9.3}$$

$$R[b, (\mu, \zeta) \to (\mu, \zeta_\mu)] = \exp\left(-\mathcal{D}(\mu, b) \ln\left(\frac{\zeta}{\zeta_\mu(b)}\right)\right) \tag{9.4}$$

B. Assumptions and approximations

- The small values of b are frizzed, at 10^{-6}.
- `TMDR_Rzeta`, for the evolution of type 4. For $b > 20\text{GeV}^{-1}$ the check of $g\mathcal{D}(\zeta_\mu) > g\mathcal{D}(\zeta_\mu^{\text{NP}})$ is performed. If it fails then $g\mathcal{D}(\zeta_\mu) = g\mathcal{D}(\zeta_\mu^{\text{NP}})$. It is done is order to avoid possible ambiguities 0/0, since at very large b evolution factor is very small, and change of sign in $g\mathcal{D}(\zeta_\mu) - g\mathcal{D}(\zeta_\mu^{\text{NP}})$ could lead to infinite values of the function.
- For very large positive values of $\mathcal{D}$ (i.e. for $b > 25\text{GeV}^{-1}$ or so), the exact solution for ζ-line diverges exponentially at 3- and 4- loops. It is because the coefficient infront of rising terms is negative. To prevent the divergence, the corresponding terms are expanded in a_s up to needed power (α_s^6).

C. Initialization

Prior the usage module is to be initialized (once per run prior to any other module-related command). By
`call TMDR_Initialize(order)`
This command read the input from the `constants`-file, and set the other parameters according to

`order` (string) declaration of order for the evolution kernel. Typically, one set Γ_{cusp} one order higher then the rest of anomalous dimensions. There are following set of orders

`order`	Γ_{cusp}	γ_V	$\mathcal{D}$ *	$\mathcal{D}_{\text{resum}}$	ζ_μ **
`LO`	a_s^1	a_s^0	a_s^0	a_s^0	a_s^0
`LO+`	a_s^1	a_s^1	a_s^1	a_s^0	a_s^1
`NLO`	a_s^2	a_s^1	a_s^1	a_s^1	a_s^2
`NLO+`	a_s^2	a_s^2	a_s^2	a_s^1	a_s^2
`NNLO`	a_s^3	a_s^2	a_s^2	a_s^2	a_s^3
`NNLO+`	a_s^3	a_s^3	a_s^3	a_s^2	a_s^3
`N3LO`	a_s^4	a_s^3	a_s^3	a_s^2	a_s^4
`N3LO+`	a_s^4	a_s^4	a_s^4	a_s^3	a_s^4
`N4LO`	a_s^5	a_s^4	a_s^4	a_s^3	a_s^{5***}

* $\mathcal{D}_{\text{resum}}$ starts from a_s^0, it already contains Γ_0.

** Definition of ζ_μ is correct only in the natural ordering, i.e. `LO`,`NLO`,`NNLO`. Proper definition in `+` orders would make the function too heavy. The resumed version has the same counting.

*** The finite part =0, since $d^{(5,0)}$ is unknown.

D. NP input and NP parameters

The NP parameters are used in the definition of the function $\mathcal{D}_{\text{NP}}$. Their number is read from the `constants`-file, and allocated (and set = 0) during the initialization procedure. Their values are set by command
`call TMDR_setNPparameter(input)`
where `input` is real*8 list NP parameters, with the length equals to the number of NP parameters.

– `DNP` –

The important part of TMD evolution is the rapidity anomalous dimension. It has a NP part which is to be parameterized by user. It should be done in the function `DNP(mu,b,f)` in the end of the file, where `mu` is (real*8) scale, `b` is (real*8) parameter b, `f` is (integer) flavor. **This functions is used for all evolution kernels.** Specifying it, you can use build-in functions `Dpert(mu,b,f)` and `Dresum(mu,b,f)` for the perturbative expressions of $\mathcal{D}$. Also the NP parameters from the set which are given by variables `NPparam(i)`.

– `zetaNP` –

The artemide is founded on the notion of ζ-prescription, therefore, the ζ_μ line plays essential role. For $\mathcal{D} \neq \mathcal{D}_{\rm NP}$ (which is standard situation), the ζ_μ line is different from the perturbative. It should be set within the artemide. It is done by user in the function `zetaNP(mu,b,f)`, with the same arguments as for `DNP`. Note, that **it MUST approach ζ_μ perturbation in small-b regeme. Otherwise, the evolution is calculated incorrectly.** E.g. if $\mathcal{D}_{\rm NP} = \mathcal{D}_{\rm pert}(b^*) + g_K b^2$ the $\zeta_{\rm NP} = \zeta_{perp}(b^*) + ..$, where dots are power suppressed, and thus can be dropped. Defining this function you can use `zetaMUpert` and `zetaMUresum` for perturbative and resumed versions of ζ_μ, as well as, `NPparam(i)`.

In the model code user can provide the `ReplicaParameters(n)`, which returns the array of NP parameters corresponding to integer number `n`. It is convenient to specify initializing values here, or indeed, the values for fit replicas. i

E. Evaluating TMD evolution kernel

The evolution kernel are presented in two types for the evolution from arbitrary point to arbitrary, and for the evolution from the arbitrary point to the ζ-line. Since various types of evolution have different number of arguments, the routines for different types are overloaded with different number of variables.

Function	Evol.type	Comments
Evolution from (μ_f, ζ_f) to (μ_i, ζ_i)		
`TMDR_R(b,muf,zetaf,mui,zetai,mu0,f)`	1,2,3	For type=2,3,4 parameter μ_0 is ignored
`TMDR_R(b,muf,zetaf,mui,zetai,f)`	1,2,3	For type=1 evaluated at $\mu_0 = \mu_i$
Evolution from (μ_f, ζ_f) to (μ_i, ζ_{μ_i})		
`TMDR_Rzeta(b,muf,zetaf,mui,mu0,f)`	1	
`TMDR_Rzeta(b,muf,zetaf,mui,f)`	2	
`TMDR_Rzeta(b,muf,zetaf,f)`	3	Absolutely fastest.

where

- `b` (real*8) Transverse distance ($b > 0$) in GeV
- `zetaf,muf` (real*8) hard-factorization scales (ζ_f, μ_f) in GeV. Typically, $=(Q^2, Q)$
- `zetai,mui` (real*8) low-factorization scales (ζ_i, μ_i) in GeV.
- `mu0` (real*8) The scale of perturbative definition of rapidity anomalous dimension $\mathcal{D}$ μ_0 in GeV.
- `f` (integer) parton flavor. 0 for gluon, $\neq 0$ for quarks.

The parameter evolution type is set in `constants`-file and is used by `TMDs` to call particular version of evolution. Within only the `TMDR`-module it is not needed.

F. Inverted evolution and the lowest available Q

At small values of parameter $Q = Q_0$ the point (Q, Q^2) crosses the ζ-lines. The value of Q_0 dependents on b. The dangerous situation is then hard scale of the process Q is smaller then Q_0 at large $b = b_\infty$. In this case the evolution kernel $R[b_\infty, (Q, Q^2) \to \zeta_\mu] > 1$, which is generally implies that it grows to infinity. However, it happens only at small values of Q. E.g. at NNLO the typical value of Q_0 is ~ 1.5GeV. That should be taken into account during consideration of low-energy experiment and especially their error-band, since the point $(c_2 Q, Q^2)$ could cross the point at larger values of Q.

The function `LowestQ()` returns the values (real*8(1:3)) $\{Q_{-1}, Q_0, Q_{+1}\}$, which are solution of equation $Q^2 = \zeta_{cQ}(b)$, for (fixed bu) large values of b. Q_{-1} corresponds to $c = 0.5$, Q_0 corresponds to $c = 1$ and Q_{+1} corresponds to $c = 2$.

X. UTMDPDF_MODEL

same for `uTMDFF_model` and `lpTMDPDF_model`.

This module gives the expression for f_{NP}, $\mu_{\mathbf{OPE}}$ and b^*. It could be written by user, and one of modules that form a TMD model (together with models for TMD-evolution).
Making this module follow the interface

```
public:: ModelInitialization(NParray)
public:: ModelUpdate(NParray)
real(dp),public,dimension(-5:5):: FNP
real(dp),public:: bSTAR
real(dp),public:: mu_OPE
public:: GetCompositionArray
public:: GetReplicaParameters
```

`ModelInitialization(NParray)`: this subroutine is called during the initialization of `TMDR`. The `real(dp),intent(in)::NParray(:)` is the array of initial NP-parameters. Use this subroutine to make preparation of the model.

`ModelUpdate(NParray)`: this subroutine is called during the `TMDR_setNPparameters(NParray)`. The `real(dp),intent(in)::NParray(:)` is the array of NP-parameters passed to `TMDR_setNPparameters`. Use this subroutine to recompute model parameters (if needed), save NP-parameters (if needed), etc.

`FNP(x,z,b,hadron,lambdaNP)`: the function returns $f_{NP}^{f,h}(x,z,b)$. See (11.1)

`bSTAR(b,lambdaNP)`: the function returns $b^*(b)$ function.See (11.1)

`mu_OPE(z,bt)`: the function returns $\mu_{\rm OPE}(z,b)$ function.See (11.1)

`GetCompositionArray(h,lambdaNP,includeArray,CA)`: subroutine returns `real(dp),allocatable,intent(out)::CA(:` and `logical,allocatable,intent(out)::includeArray(:)` which define the matrix A_{hr} (see sec.XI D). Particularly, `CA(i)`$=A_{hi}$. The logical array `includeArray` indicates which terms should be included in the sum (`.true.`=include, `.false.`=do not include).

`GetReplicaParameters(num,NParray)`: subroutine returns `real(dp),allocatable,intent(out)::NParray(:)` associated with `integer::num`. In such way, one could call replicas of a model.

Important: the module `uTMDPDF_model` is compiled before `uTMDPDF`, and thus can use only lower-level modules: `TMD_AD, QCDinput`, etc.

XI. UTMDPDF MODULE

The module `uTMDPDF` performs the evaluation of the unpolarized TMD PDF at low scale μ_i in ζ-prescription. It is given by the following integral

$$F_f(x,b) = \int_x^1 \frac{dz}{z} C_{f\leftarrow f'}(z, b^*(b), c_4\mu_{\text{OPE}}(z,b)) f_{f'}(\frac{z}{x}, c_4\mu_{\text{OPE}}(z,b)) f_{NP}^f(x,z,b,\{\lambda\}), \tag{11.1}$$

where $f_f(x,\mu)$ is PDF of flavor f, C is the coefficient function in ζ-prescription, f_{NP} is the non-perturbative function. The variable c_4 is used to test the scale variation sensitivity of the TMD PDF. The NNLO coefficient functions used in the module were evaluated in [6] (please, cite it if use).

List of available commands optional parameters are shown in blue.

Command	Type	Sec.	Short description
`uTMDPDF_Initialize(file,prefix)`	subrout.	XI A	Initialization of module. (string) `file` is the name of `constants-file`, which contains initialization information. (string)`prefix` is appended to `file` if provided.
`uTMDPDF_SetLambdaNP(lambda, buildGrid, gluonRequared)` `uTMDPDF_SetLambdaNP(num, buildGrid, gluonRequared)`	subrout	XI B	Set new NP parameters used in `FNP` and `bSTAR`. The option with (real*8 array)`lambda` set the parameters directly. The option with (int)`num` set the parameters according to `ReplicaParameters` defined in model. Optional (logical) parameter `buildGrid, gluonRequared` override the options for grid contraction.
`uTMDPDF_CurrentNPparameters(var)`	subrout.	XI B	Return the (real*8) array of current values of λ_{NP}.
`uTMDPDF_lowScale5(x,b,h)`	(real*8(-5:5))	XI C	Returns unpolarized TMD PDF at x, b and hadron h. Gluon flavour undefined.
`uTMDPDF_lowScale50(x,b,h)`	(real*8(-5:5))	XI C	Returns unpolarized TMD PDF at x, b and hadron h.
`uTMDPDF_SetScaleVariation(c4))`	subrout		Set new value of c_4 (default value $c_4 = 1$).
`uTMDPDF_resetGrid(bG,g)`	subrout	XI E	Force reset or deconstruct the grid.
`uTMDPDF_SetPDFreplica(num)`	subrout	–	Call `QCDinput` to change the PDF replica number, deconstructs grid.

List of functions which must be provided by a model code

Input	Short description
`ModelInitialization()`	Necessary predefinitions by user. E.g. some precalculations for `FNP`. Can be black.
`FNP(x,z,b,hadron)`	NP-model for $f_{NP}(x,z,b,\{\lambda\})$ depends on the hadron. See sec.XI B
`mu_OPE(x,bt)`	The value of μ_{OPE}.
`bSTAR(bT,lambdaNP)`	The value of b^*, can be just `bSTAR=bT`.
`ReplicaParameters(rep)`	Returns a presaved array of λ_{NP} corresponding to integer `rep`. Can be black.

A. Initialization

Prior the usage module is to be initialized (once per run). By
`call uTMDPDF_Initialize(file)`
In `constants-file` the order of the perturbative input is defined by

$$\texttt{LO,LO+} = a_s^0, \qquad \texttt{NLO,NLO+} = a_s^1, \qquad \texttt{NNLO,NNLO+} = a_s^2.$$

There is also a special option `NA`. In this case no matching to perturbation theory is done and the TMD distribution is replaced by $f_{NP}(x,1,b)$. I.e.

$$F_f(x,b) = f^f_{NP}(x,1,b,\{\lambda\}). \tag{11.2}$$

In this case, the griding options are ignored. The composition option works as usual.
Within the ζ-prescription this option is theory-save. I.e. it can be used together with any-order evolution without theory tensions.

B. Definition of TMD model, f_{NP} and parameters

The model for TMD is given by f_{NP}, b^*, and in smaller amount by μ_{OPE}. The definitions of these functions is provided by user in the file `uTMDPDF_model.f90`, which is located in the `scr/model` directory.

- The function `FNP` is dependent on x, z, b and λ (and the hadron flavor). It is an array for all flavors (-5:5). It uses the parameters $\lambda_{1,2,...}$ which are passed to it by main module. The total number of NP parameters `LambdaNPLength`, is declared in the `constants`-file.
- User provides the value of μ_{OPE} (or use the default one) in the function `mu_OPE(x,b)`. This scale is used inside the convolution $F(x,b) = C(x,b;\mu_{\text{OPE}}) \otimes q(x,\mu_{\text{OPE}})$. The function could depend on x (the one which enter $f(x)$ in the convolution).
- User provides the value of b^* (or use the default one) in the function `bSTAR(b,lambdaNP)`. This function is used withing the coefficient function $C(x,b^*;\mu)$. Generally, the (twist-2) coefficient function depends only on logarithms $\ln(\mu_{OPE}b^*)$. The function could depend on λ.
- Together with the model user can provide the function `ReplicaParameters(n)`, which returns NP parameters in accordance to input integer number n. These parameters will be set as be current $\lambda_{1,2..}$, upon the call `uTMDPDF_SetLambdaNP(n)`, where `n` is integer number of the replica. It is convenient to specify initializing values here, or indeed, the values for fit replicas.

To set the values for array `lambdaNP` use the subroutine
`call uTMDPDF_SetLambdaNP((/`λ_1,λ_2`,...`/`))`
Optional: There exist the overloaded version of `uTMDPDF_SetLambdaNP`, with two additional boolean parameters
`call uTMDPDF_SetLambdaNP((/`λ_1,λ_2`,...`/`),makeGrid,includeGluons)`
If parameter `makeGrid=.true.` then for *this* run of non-perturbative parameters the grid for TMD will be evaluated. Then until new NP parameters set the TMDs are reconstructed from the grid, see sec.XI E.

If parameter `includeGluons=.true.`, the grid is calculated with gluons. If parameter `includeGluons=.false.`, the grid is calculated without gluons (but the mixture of quark with gluon is taken into account. The difference is the same as, e.g. between `uTMDPDF_lowScale5` and `uTMDPDF_lowScale50` functions (see next section).

Default version has `makeGrid=.false.,includeGluons=.false.`. Note, that this command compare new values of parameters to the old one. If they coincides, the grid is not renewed.

Optional: There exist the overloaded version of `uTMDPDF_SetLambdaNP(n)`, with `n` being an integer. It attempt to load user defined set of NP parameters associated with number n.

C. Evaluating unpolarized TMD PDFs

The expression for unpolarized TMD PDFs is given by the function
`uTMDPDF_lowScale??(x,b,h)`
where

`x` (real*8) Bjorken-x ($0 < x < 1$)

`b` (real*8) Transverse distance ($b > 0$) in GeV

`h` (integer) The number that indicates the hadron. Since coefficient function is hadron independent, this number influence the PDF that used, and `FNP`.

The questions marks stand for a flavor content of TMD-vector. The functions evaluate the TMD PDFs of different flavours simultaneously.

`uTMDPDF_lowScale5(x,b,h)` returns (real*8) array(-5:5) for $\bar{b}, \bar{c}, \bar{s}, \bar{u}, \bar{d}, ?, d, u, s, c, b$. Gluon contribution is undefined, but taken into account in the mixing contribution.

`uTMDPDF_lowScale50(x,b,h)` returns (real*8) array(-5:5) for $\bar{b}, \bar{c}, \bar{s}, \bar{u}, \bar{d}, g, d, u, s, c, b$. This procedure is slower ($\sim 10-50\%$ depending on parameters, mainly on x) in comparison to the previous command. The slowdown is presented since the gluon coefficient function has $1/x$ behavior, and requires more iteration to reach the demanded precision. If gluons are not needed use previous.

Important note: there is no arguments μ and ζ, because the `artemide` uses the ζ-prescription, where a TMD distribution is scaleless. The scale of matching procedure μ_{OPE} is set in the function `mu_OPE` (see previous subsection). Note, that the TMD at different then ζ-prescription point can be evaluated within `TMDs` package (which uses `uTMDPDF` in turn).

Additional points:

- In order to avoid possible problems at $b=0$, at $b<10^{-6}$ the value of b is set to $b=10^{-6}$. This region is numerically non-important, since in any cross-section it is suppressed by b^n ($n\geqslant 1$) within the Fourier integral.
- The convolution procedure $C\otimes f$ is the most costly procedure in the package. Its timing seriously increases from NLO to NNLO coefficient function (about 10 times). In the current version we implement the Gauss-Kronrod adaptive algorithm, with estimation of accuracy as $|(G7-K15)/(f(x)f_{NP}(1))|<\epsilon$, where the default value of ϵ is 10^{-3}. According to our checks default estimation guaranties the 4-digit precision of the evaluation. If integrand does not converge fast enough at $z\to 1$ (e.g. for gluon contribution at NNLO, where $\ln^3\bar{z}$ is presented), the integral at $(x_0,1)$ is replaced by exact integral with constant $f(x)f_{NP}$. The value of x_0 is determined by $f'(x_0)<\epsilon$ and $x_0>1-\epsilon$. This additional procedure is needed to ensure convergence of the integral. However, in our experience (which uses only quark TMDs), this extra procedure is not used at all.

D. Composite TMD

Starting from the ver.2.03 there is an option "composite TMD" (constants-file A.p2). This option overrides the definition of hadrons F_h.

If the option is False, the evaluation goes in the transitional way, i.e. the enumeration of hadrons h corresponds to the enumeration of the PDFs. So,

$$\texttt{IsComposite=.false.} \Rightarrow \qquad F_{h\leftarrow q} = f_{h\leftarrow q'}\otimes C_{q'\leftarrow q} f_{NP;h,q} = \mathbf{F}_{h\leftarrow q}.$$

If the option is True, the enumeration of hadrons h does not correspond to the enumeration of the PDFs. In this case, a TMD is evaluated by the expressions

$$\texttt{IsComposite=.true.} \Rightarrow \qquad F_{h\leftarrow q} = \sum_r A_{hr} f_{r\leftarrow q'}\otimes C_{q'\leftarrow q} f_{NP;r,q} = \sum_r A_{hr}\mathbf{F}_{r\leftarrow q}.$$

The arrays A_h are defined in the model-file within subroutine `GetCompositionArray`, and could depend on NP-parameters.

The composite TMD are useful in some cases. For example, the charged hadron TMDFF (the composition of pion+kaon+...), the estimation of PDF uncertanty band in with eigen-vectors PDF, and others.

E. Grid construction

For fitting procedure one often needs to evaluate TMDs multiple times. For example, for fit performed in [1] the evaluation of singe χ^2 entry requires $\sim 16\times 10^6$ calls of `uTMDPDF_`... I recall that a TMD is given by the expression

$$F(x,\boldsymbol{b}) \simeq \int dy C(x/y, \ln(\boldsymbol{b}^2)) f(y) f_{NP}(x,y,\boldsymbol{b},\lambda). \tag{11.3}$$

So, every call of `uTMDPDF_..` at NNLO order, requires ~ 200 calls of pdfs, depending on x, b and λ's, in order to evaluate the integral over y. In such situations, it is much faster to make a grid of TMD distributions for a given set of non-pertrubative parameters (i.e. the grid is in x and b), and then use this grid for the interpolation of TMD values. The calculation of a grid is not a very fast procedure, nonetheless, for large computation (number of TMD calls $> 10^4 - 10^5$) is more efficient.

The griding is turned on by the call of overloaded version of `uTMDPDF_SetLambdaNP(lambda,makeGrid,includeGluons)` with `makeGrid=.true.` (see also sec.XI B). After this call the grid will be built (the corresponding massage will be shown on the screen, if output level is > 1). This grid is used for the interpolation of TMD distribution until the next call of `uTMDPDF_SetLambdaNP`, which resets/cancels grid. To speed up the multiple changes of parameters, the packages checks the function $f_{NP}(x,y)$ onto the y-dependence, and b^* on λ-dependence. If any of them is observed, it implies that the convolution integral depends on λ. If convolution integral does not depend on λ, then the grid is not renewed (unless it is forcibly reseted) but reweighted with new f_{NP}.

The interpolation is cubic. The grid is build for the finite domain of $x \in (x_{\min}, 1)$ and $b \in (0, b_M)$. For $x < x_{\min}$ the program will be terminated (with an error). In the default set we have

$$x_{\min} = 10^{-5}, \qquad b_M = 100.$$

The default grid is 250×750 (the grid is logarithmic in both x and b, small x and $b \to 0$), we have found that it gives in average 5-6 digit precision. All this parameter can be changed in `constants` file in the section 3.D. These parameters have been used to fit a large domain of energies and q_T. However, we recommend, to check the obtained result by exact evaluation without a grid to ensure the precision in particular cases.

For $b > b_M$ we use the following approximate formula

$$\text{for } b > b_M: \quad F(x,b) = f_{NP}(x,x,b)\frac{F(x,b_M)}{f_{NP}(x,x,b_M)}. \tag{11.4}$$

This formula is an approximation which is exact only in the case: b(in convolution) freezes at large values, and f_{NP} is y-independent. For the overwhelming part of models it is the case. Otherwise, there is a numerical error. However, I have checked this error is typically not large (for smooth models) $\sim 0.1 - 10\%$ at $b = 1.5b_M$ (at $b_M = 100$). Numerically, such error is absolutely negligible, since typical values at $b_M = 100$ of F is 10^{-50}, and thus it gives unobserved correction to the Fourier integral. This approximation has been interoduced in ver.1.5, in previous versions an interpolation procedure has been used, which produced seriously higher error, that actually affected small-q_T bins in small but visible amount $\sim 0.1\%$.

The subroutine `uTMDPDF_resetGrid(makeGrid,includeGluons)` changes the current behaviour (for the meaning of arguments see `uTMDPDF_SetLambdaNP`). If `makeGrid=.true.` the grid will be recalculated.

F. Theoretical uncertainties

`uTMDPDF_SetScaleVariation(c4)` changes the scale multiplicative factor c_4 (see [1], eqn.(2.46)).

G. Technical notes

Convolution

The convolution integral reads

$$I(x) = \int_x^1 \frac{dz}{z} C(z) f\left(\frac{x}{z}\right) f_{NP}(x,z), \tag{11.5}$$

where the function $C(z)$ has a general form

$$C(z) = C_0\delta(1-z) + (C_1(z))_+ + C_2(z). \tag{11.6}$$

Here the plus-distribution is undestood in the usual way

$$(C_1(z))_+ = C_1(z) - \delta(1-z)\int_0^1 dy C_1(y) dy. \tag{11.7}$$

In NNLO coefficient function there are only possible two $(..)_+$ terms, $1/(1-z)$ and $\ln(1-z)/(1-z)$.

In order to simplify the integration we rewrite

$$I(x) = \frac{1}{x}\int_x^1 dz C(z)\tilde{f}\left(\frac{x}{z}\right) f_{NP}(x,z), \qquad \tilde{f}(z) = zf(z). \tag{11.8}$$

Then the integral is split as

$$I(x) = \frac{1}{x}\Big\{I_2(x) + C_0\tilde{f}\,(x)\, f_{NP}(x,1) + \tilde{f}\,(x)\, f_{NP}(x,1)\int_0^x dz C_1(z)\Big\}, \tag{11.9}$$

where

$$I_2(x) = \int_x^1 dz\Big[C_1(z)\left(\tilde{f}\left(\frac{x}{z}\right) f_{NP}(x,z) - \tilde{f}\,(x)\, f_{NP}(x,1)\right) + C_1(z)\tilde{f}\left(\frac{x}{z}\right) f_{NP}(x,z)\Big] \tag{11.10}$$

Presumably, the term $\sim C_0$ give the dominant contribution w, since it is ~ 1 whereas the other terms $\sim a_s$. Therefore, it serves as the estimation of the integral value, with respect to which the integration convergence is calculated. The convolution integral is evaluated by the G7K15 rule adaptively with given tolerance with respect to w.

The integral I is calculated in the procedure `Common_lowScale50` and `Common_lowScale5`, which is common for all twist-2 terms. The integral I_2 is calculated in the iterative procedures `MellinConvolutionVectorPart50` and `MellinConvolutionVectorPart5`, which is common for all twist-2 terms.

In the case of TMDFF we have the coefficient function with the structure $C(z)/z^2$ (plus-distribution, δ, etc, are multiplied by $1/z^2$), and collinear function $f(z)/z^2$. In this case it is convenient to rewrite

$$I(x) = \int_x^1 \frac{dz}{z}\frac{C(z)}{z^2}\frac{f\left(\frac{x}{z}\right)}{(x/z)^2} f_{NP}(x,z) = \frac{1}{x^3}\int_x^1 dz C(z)\hat{f}\left(\frac{x}{z}\right) f_{NP}(x,z), \qquad \hat{f}(z) = zf(z). \tag{11.11}$$

Since the common-code calculates the integral PDF-like convolution, I divide by factor $1/x^2$ all output of the common-code.

Coefficient functions

The coefficient function is spit to 6 terms.

$$C_{q\leftarrow q}, \quad C_{q\leftarrow g}, \quad C_{g\leftarrow q}, \quad C_{g\leftarrow g}, \quad C_{q\leftarrow \bar{q}}, \quad C_{q\leftarrow \bar{q}}, \quad C_{q\leftarrow q'}, \tag{11.12}$$

where $C_{q\leftarrow q}$ and $C_{q\leftarrow \bar{q}}$ contain *non-singlet* part only. The singlet contribution is given in $C_{q\leftarrow q'}$. So, the convolution for a quark (e.g. $q=u$) reads

$$F_u = C_{q\leftarrow q}\otimes q_u + C_{q\leftarrow g}\otimes q_g + C_{q\leftarrow \bar{q}}\otimes q_{\bar{u}} + C_{q\leftarrow q'}\otimes \sum_{f\in q,\bar{q}} q_f. \tag{11.13}$$

For gluon

$$F_g = C_{g\leftarrow g}\otimes q_g + C_{g\leftarrow q}\otimes \sum_{f\in q,\bar{q}} q_f. \tag{11.14}$$

The kernels are split into regular, singular and δ-parts, where singular contains $(\ln(1-x)/(1-x)))_+^n$ terms, δ contains only $\delta(1-x)$ terms, and regular is the rest. Singular and δ-terms are present only for $C_{q\leftarrow q}$ and $C_{g\leftarrow g}$.

The regular part is decomposed further, into singular (but integrable) terms at $x\to 0$ and $x\to 1$, and the finite part. The singular terms are computed exactly, while the finite part is fitted to linear combination of several functions. The basis of fit is such that the expressions are exact for coefficients which do not contain PolyLog functions. These are NLO, leading log [at all orders], large-Nf [at all orders], and some others. The full basis for the regular part is

$$\Big\{\underbrace{\ln\bar{x}, \ln^2\bar{x}, \ln^3\bar{x}, \ln^4\bar{x}, \ln^5\bar{x}, \frac{1}{x}, \frac{\ln x}{x}, \frac{\ln^2 x}{x}, \ln x, \ln^2 x, \ln^3 x, \ln^4 x, \ln^5 x,}_{\substack{\text{singular-integrable}\\ \text{exact}}} \tag{11.15}$$

$$\underbrace{1, x}_{\text{exact}}, x\bar{x}, x\left(\frac{\ln x}{1-x}+1\right), \frac{x\ln^2 x}{1-x}, \frac{x\ln^3 x}{1-x}, x\ln x, x\ln^2 x, x\ln^3 x, x\ln^4 x, x\bar{x}\ln x, x\bar{x}\ln^2 x, x\bar{x}\ln^3 x,$$

$$\bar{x}\ln\bar{x}, \bar{x}\ln^2\bar{x}, \bar{x}\ln^3\bar{x}, x\bar{x}\ln\bar{x}, x\bar{x}\ln^2\bar{x}, x\bar{x}\ln^3\bar{x}, \ln x\ln\bar{x}, x\ln x\ln\bar{x}, x\bar{x}\ln x\ln\bar{x}, \frac{\ln\bar{x}(1-x^2+2\ln x)}{1-x}, \frac{\bar{x}(\ln x + x)}{x}\Big\}.$$

Decomposed in this basis the coefficient functions are very presice. The typical relative precision is $10^{-8} - 10^{-10}$ *for the finite part* (other terms are exact). The worst presicion happen at small $x \sim 10^{-5}$ for $g \leftarrow g$ channel where relative precision is $\sim 10^{-5} - 10^{-6}$.

Similar ansatz is used for uTMDFF, with some term removed (which are sensitive to small-x region). Precision in the case of TMDFF is slightly worse $10^{-6} - 10^{-8}$. The worst precision is for N^{3}LO $q \rightarrow q'$ channel, where it is $\sim 10^{-4} - 10^{-6}$. However, it is very good precision which guaranties about 10-12 presice digits for the physical convolution.

XII. uTMDFF MODULE

The TMDFF functions structurally repeats the TMDPDF functions. Therefore, the module is practically the same as `uTMDPDF`. Command are the same as for `uTMDPDF` with replacement `uTMDPDF_...` to `uTMDFF_...`. The NNLO coefficient functions used in the module were evaluated in [6, 7] (please, cite it if use).

The definition of TMDFF does not perfectly match the definition of collinear FF. There is a relative factor $1/z^2$. So the convolution reads

$$D(z,b) = \int_z^1 \frac{dy}{y}\mathbb{C}\left(\frac{z}{y},\mathbf{L}_\mu\right)\frac{d(y,\mu)}{y^2}D_{NP}(z,\frac{z}{y},b) = \frac{1}{z^2}\int_z^1 \frac{dy}{y}C\left(y,\mathbf{L}_\mu\right)(y^2 d(y,\mu))D_{NP}(z,y,b), \tag{12.1}$$

where $\mathbb{C}$ starts from $\delta(\bar{y})$.

In the case of `NA` the TMDFF is replaced by D_{NP} entirely. I.e. $D(z,b) = D_{NP}(z,1,b)$.

XIII. lpTMDPDF MODULE

The module is practically the same as `uTMDPDF`. Command are the same as for `uTMDPDF` with replacement `uTMDPDF_...` to `lpTMDPDF_...`. The NNLO coefficient functions used in the module were evaluated in [15] (please, cite it if use).

> **NOTE:** Since linearly polarized gluon TMD PDF contains only gluon all options related to gluons are ignored (although the argument is preserved to keep common interface). E.g. `gluonRequared` (in `lpTMDPDF _SetLambdaNP` is automatically replaced by `.true.`. Also the function `lpTMDPDF lowScale5` is absent, since it is identically equal to zero.

XIV. SiversTMDPDF MODULE

The module implements the Sivers function $f_{1T}^{\perp}(x,b)$ defined as

$$\Phi^{[\gamma^+]}(x,b) = f_1(x,b) + i\epsilon_T^{\mu\nu}b_\mu s_\nu M f_{1T}^{\perp}(x,b), \tag{14.1}$$

where $\Phi^{[\gamma^+]}(x,b)$ is ordinary TMD operators with coordinates $\bar{q}(b)...q(0)$, and f_1 is unpolarized distribution. The Wilson line in this operator is taken pointing to $+\infty$. **This definition corresponds to SIDIS definition of $f_{1T}^{\perp}$.** For DY definition one should multiply it by (-1). This sign is explictely carried out in definitions of structure functions for DY processes. The variable M is uniformly taken as mass of proton.

The function is computed by the form

$$f_{1T}^{\perp}(x,b) = C \otimes T(x,\ln(b))f_{\text{NP}}(x,b), \tag{14.2}$$

where T are twist-3 distributions. The function f_{NP} is defined in the `SiverTMDPDF_model` module.

> For the moment (v2.05) there is no perturbative matching. Therefore, the usage of `NA`-order is advisable. The main reason is absence of phenomenological Qui-Sterman function. The NLO perturbative matching is known [16], and will be implemented in future versions.

The module is practically the same as `uTMDPDF`. Command are the same as for `uTMDPDF` with replacement `uTMDPDF_...` to `SiversTMDPDF_...`.

XV. wgtTMDPDF MODULE

The module implements the worm-gear T (it explains the name `wgt`) function $g_{1T}(x,b)$ defined as

$$\Phi^{[\gamma^+\gamma^5]}(x,b) = \lambda g_{1L}(x,b) + ib_\mu s^\mu M g_{1T}(x,b), \tag{15.1}$$

where $\Phi^{[\gamma^+]}(x,b)$ is ordinary TMD operators with coordinates $\bar{q}(b)...q(0)$, and g_{1L} is the helicity TMDPDF. The variable M is uniformly taken as mass of proton.

The function is computed by the form

$$g_{1T}(x,b) = (C_{\text{tw2}} \otimes g_1(x,\ln(b)) + C_{\text{tw3}} \otimes T)\, f_{\text{NP}}(x,b), \tag{15.2}$$

where g_1 is the helicity PDF, T are twist-3 distributions and coefficients C are small-b matching coefficients. The leading term for twsit-t part is

$$C_{\text{tw2}} \otimes g_1 = x \int_x^1 \frac{du}{u} g_1(u) + \mathcal{O}(\alpha_s). \tag{15.3}$$

Since the functions T are not known the twist-3 convolution term is replaced by a single NP function. It is defined in `wgtTMDPDF_model` with `g1T_tw3NP(x,hadron,lambdaNP)` function (depending only on x). The function f_{NP} is defined in addition. Note, that both functions use the same NP-input vector.

The order =`NA` corresponds to $g_{1T}(x,b) = f_{\text{NP}}(x,b)$ (without NP-tw3 contribution).

The module is practically the same as `uTMDPDF`. Command are the same as for `uTMDPDF` with replacement `uTMDPDF_...` to `wgtTMDPDF_...`.

XVI. TMDS MODULE

The module `TMDs` joins the lower modules and performs the evaluation of various TMD distributions in the ζ-prescription. Generally a TMD distribution is given by the expression

$$F_f(x,b;\mu,\zeta) = R_f[b,(\mu,\zeta) \to (\mu_i,\zeta_{\mu_i})]\tilde{F}_f(x,b), \tag{16.1}$$

where R is the TMD evolution kernel, $\tilde{F}$ is a TMD distribution at low scale. The scale μ_i is dependent on the evolution type, and could be out of use. Note, that `TMDs` initializes the lower modules automatically. Therefore, no special initializations should be done.

List of available commands

Command	Type	Sec.	Short description
TMDs_Initialize(file,prefix)	subrout.	-	Initialization of module. (string) file is the name of constants-file, which contains initialization information. (string)prefix is appended to file if provided.
TMDs_SetScaleVariations(c1,c3)	subrout.	XVI D	Set new values for the scale-variation constants.
uTMDPDF_5(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Unpolarized TMD PDF (gluon term undefined)
uTMDPDF_50(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Unpolarized TMD PDF (gluon term defined)
uTMDFF_5(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Unpolarized TMD FF (gluon term undefined)
uTMDFF_50(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Unpolarized TMD FF (gluon term defined)
lpTMDPDF_50(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Linearly polarized gluon TMD PDF (quarks are zero)
SiversTMDPDF_5(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Sivers TMD PDF (gluon term undefined)
SiversTMDPDF_50(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	Sivers TMD PDF (gluon term defined)
wgtTMDPDF_5(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	worm-gear T TMD PDF (gluon term undefined)
wgtTMDPDF_50(x,b,mu,zeta,h)	(real*8(-5:5))	XVI B	worm-gear T TMD PDF (gluon term defined)
uTMDPDF_5(x,b,h)	(real*8(-5:5))	XVI B	Unpolarized TMD PDF at optimal line (gluon term undefined)
uTMDPDF_50(x,b,,h)	(real*8(-5:5))	XVI B	Unpolarized TMD PDF at optimal line (gluon term defined)
uTMDFF_5(x,b,h)	(real*8(-5:5))	XVI B	Unpolarized TMD FF at optimal line(gluon term undefined)
uTMDFF_50(x,b,h)	(real*8(-5:5))	XVI B	Unpolarized TMD FF at optimal line (gluon term defined)
lpTMDPDF_50(x,b,,h)	(real*8(-5:5))	XVI B	Linearly polarized gluon TMD PDF at optimal line (quarks are zero)
SiversTMDPDF_5(x,b,h)	(real*8(-5:5))	XVI B	Sivers TMD PDF at optimal line (gluon term undefined)
SiversTMDPDF_50(x,b,,h)	(real*8(-5:5))	XVI B	Sivers TMD PDF at optimal line (gluon term defined)
wgtTMDPDF_5(x,b,h)	(real*8(-5:5))	XVI B	Worm gear T TMD PDF at optimal line (gluon term undefined)
wgtTMDPDF_50(x,b,,h)	(real*8(-5:5))	XVI B	Worm gear T TMD PDF at optimal line (gluon term defined)
uPDF_uPDF(x1,x2,b,mu,zeta,h1,h2)	(real*8(-5:5))	XVI C	Product of Unpolarized TMD PDF $f_{q\leftarrow h_1}(x_1)f_{\bar{q}\leftarrow h_1}$ at the same scale (gluon term undefined)
uPDF_anti_uPDF(x1,x2,b,mu,zeta,h1,h2)	(real*8(-5:5))	XVI C	Product of Unpolarized TMD PDF $f_{q\leftarrow h_1}(x_1)f_{q\leftarrow h_1}$ at the same scale (gluon term undefined)

List of functions which must be provided by a model code

Input	Short description
mu_LOW(b)	The value of μ_i used in the evolutions of type 1 and 2 (improved $\mathcal{D}$ and γ). See [4].
mu0(b)	The value of μ_0 used in the evolution of type 1 (improved $\mathcal{D}$). See [4].

A. Definition of low-scales

The low scales μ_i and μ_0 are defined in the functions `mu_LOW(bt)` and `mu0(bt)` which can be found in the end of `TMDs.f90` code. Modify it if needed.

B. Evaluating TMDs

The expression for unpolarized TMD PDF is obtained by the functions
(real*8(-5:5))`uTMDPDF_5(x,b,mu,zeta,h)`
where

`x` (real*8) Bjorken-x ($0 < x < 1$)

`b` (real*8) Transverse distance ($b > 0$) in GeV

`mu` (real*8) The scale μ_f in GeV. Typically, $\mu_f = Q$.

`zeta` (real*8) The scale ζ_f in GeV^2. Typically, $\zeta_f = Q^2$.

`h` (integer) The hadron type.

This function return the vector real*8(-5:5) for $\bar{b}, \bar{c}, \bar{s}, \bar{u}, \bar{d}, ?, d, u, s, c, b$.

- Gluon contribution in `uTMDPDF_5` is undefined, but taken into account in the mixing contribution. The point is that evaluation of gluons slow down the procedure approximately by 40%, and often is not needed. To calculate the full flavor vector with the gluon TMD, call `uTMDPDF_50(x,b,mu,zeta,h)`, where all arguments defined in the same way.
- The other TMDs, such as unpolarized TMDFF, transversity, etc. are obtained by similar function see the table in the beginning of the section.
- Each function has version without parameters `mu` and `zeta`. It corresponds to the evaluation of a TMS at optimal line [4]. Practically, it just transfers the outcome of corresponding TMD module, e.g.module `uTMDPDF`, see sec.XI.

C. Products of TMDs

The the evaluation of majority of cross-sections one needs the product of two TMDs at the same scale. There are set of functions which return these products. They are slightly faster then just evaluation and multiplication, due to the flavor blindness of the TMD evolution. The function have common interface
(real*8(-5:5)) `uPDF_uPDF(x1,x2,b,mu,zeta,h1,h2)`
where

`x1,x2` (real*8) Bjorken-x's ($0 < x < 1$)

`b` (real*8) Transverse distance ($b > 0$) in GeV

`mu` (real*8) The scale μ_f in GeV. Typically, $\mu_f = Q$.

`zeta` (real*8) The scale ζ_f in GeV^2. Typically, $\zeta_f = Q^2$.

`h1,h2` (integer) The hadron's types.

The function return a product ofthe form $F_{f_1 \leftarrow h_1}(x_1, b; \mu, \zeta) F_{f_2 \leftarrow h_2}(x_2, b; \mu, \zeta)$, where $f_{1,2}$ and the type of TMDs depend on the function.

D. Theoretical uncertainties

`TMDs_SetScaleVariations(c1,c3,c4)` changes the scale multiplicative factors c_i (see [4], sec.6). The default set of arguments is (1,1,1), i.e. the scales as they given in corresponding functions. This subroutine changes $c1$ and $c3$ constants and call corresponding subroutines for variation of $c4$ in TMD defining packages. Note, that in some types of evolution particular variations absent.

XVII. TMDF MODULE

This module evaluates the structure functions, that are universally defined as

$$F(Q^2, q_T, x_1, x_2, \mu, \zeta_1, \zeta_2) = \int_0^\infty \frac{bdb}{2} b^n J_n(bq_T) \sum_{ff'} z_{ff'}(Q^2) F_1^f(x_1, b; \mu, \zeta_1) F_2^{f'}(x_2, b; \mu, \zeta_2), \tag{17.1}$$

where

- Q^2 is hard scale.
- q_T is transverse momentum in the factorization frame. It coincides with measured q_T in center-mass frame for DY, but $q_T \sim p_T/z$ for SIDIS.
- x_1 and x_2 are parts of collinear parton momenta. I.e. for DY $x_{1,2} \simeq Qe^{\pm y}/\sqrt{s}$, while for SIDIS $x_2 \sim z$. It can also obtain power correction, ala Nachmann variables.
- μ is the hard factorization scale $\mu \sim Q$
- $\zeta_{1,2}$ are rapidity factorization scales. In the standard factorization scheme $\zeta_1\zeta_2 = Q^4$. It can also be modified by power corrections.
- f, f' are parton flavors.
- $z_{ff'}$ is the process related function. E.g. for photon DY on $p + \bar{p}$, $z_{ff'} = \delta_{ff'}|e_f|^2$.
- n The order of Bessel transformation is defined by structure function. E.g. for unpolarized DY $n = 1$. For SSA's $n = 1$. In general for angular modulation $\sim \cos(n\theta)$.
- $F_{1,2}^f$ TMD distribution (PDF or FF) of necessary polarization and flavor.

Some structure functions are multiplied by a constant with mass dimension (e.g. Sivers asymmetry). This constant (called global mass scale $= M$) is defined globally in the constants file.

List of available commands

Command	Type	Sec.	Short description
`TMDF_Initialize(file,prefix)`	subrout.	XVII A	Initialization of module. (string) `file` is the name of `constants-file`, which contains initialization information. (string)`prefix` is appended to `file` if provided.
`TMDF_ShowStatistic())`	subrout.	–	Print current statistic on the number of calls.
`TMDF_ResetCounters()`	subrout.	–	Reset intrinsic counters. E.g.release convergenceISlost state.
`TMDF_F(Q2,qT,x1,x2,mu,zeta1,zeta2,N)`	(real*8)	XVII B	Evaluates the structure function

A. Initialization

Prior the usage module is to be initialized (once per run) by
`call TMDF_Initialize(file)`
It reads `constants-file` and initialize it-self accordingly.

B. Evaluating Structure functions

The value of the structure function is obtained by
(real*8)`TMDF_F(Q2,qT,x1,x2,mu,zeta1,zeta2,N)`
where

`Q2` (real*8) hard scale in GeV^2.

`qT` (real*8) modulus of transverse momentum in the factorization frame in GeV, $q_T > 0$

`x1` (real*8) x passed to the first TMD distribution ($0 < x1 < 1$)

`x2` (real*8) x passed to the first TMD distribution ($0 < x2 < 1$)

`mu` (real*8) The hard scale μ in GeV. Typically, $\mu = Q$.

`zeta1` (real*8) The scale ζ_f in GeV^2 for the first TMD distribution. Typically, $\zeta_f = Q^2$.

`zeta2` (real*8) The scale ζ_f in GeV^2 for the second TMD distribution. Typically, $\zeta_f = Q^2$.

`N` (integer) The number of process.

The function returns the value of

$$F^N(Q^2, q_T, x_1, x_2, \mu, \zeta_1, \zeta_2) = \int_0^\infty \frac{bdb}{2} b^n J_n(bq_T) \sum_{ff'} z_{ff'}^N(Q^2) F_1^f(x_1, b; \mu, \zeta_1) F_2^{f'}(x_2, b; \mu, \zeta_2). \tag{17.2}$$

The parameter n depends on the argument `N` and uniformly defined as

$$\begin{array}{ll} n = 0 & \text{for } \mathtt{N} < 10000 \\ n = 1 & \text{for } \mathtt{N} \in [10000, 20000] \\ n = 2 & \text{for } \mathtt{N} \in [20000, 30000] \\ n = 3 & \text{for } \mathtt{N} > 30000 \end{array}$$

The particular values of $z_{ff'}$ and $F_{1,2}$ are given in the following table. User function can be implemented by code modification.

Notes on the integral evaluation:

- The integral is uniformly set to 0 for $q_T < 10^{-7}$.
- The integrand is uniformly set to 0 for $b > 10^3$.
- If for any element of evaluation (including TMD evolution factors and convolution integrals, and the integral of the structure function it-self) obtained divergent value. The trigger is set to ON. In this case, the integral returns uniformly large value 10^{100} for all integrals without evaluation. The trigger is reset by new values of NP parameters. It is done in order to cut the improper values of NP parameters in the fastest possible way, which speed up fitting procedures.
- The Fourier is made by Ogata quadrature, which is double exponential quadrature. I.e.

$$\int_0^\infty \frac{bdb}{2} b^n I(b) J_n(q_T b) \simeq \frac{1}{q_T^{n+2}} \sum_{k=1}^{\infty} \tilde{\omega}_{nk} b_{nk}^{n+1} I\left(\frac{b_{nk}}{q_T}\right), \tag{17.3}$$

  where

$$b_{nk} = \frac{\psi(\tilde{h}\tilde{\xi}_{nk})}{\tilde{h}} = \tilde{\xi}_{nk} \tanh(\frac{\pi}{2} \sinh(\tilde{h}\tilde{\xi}_{nk}))$$

$$\tilde{\omega}_{nk} = \frac{J_n(\tilde{b}_{nk})}{\tilde{\xi}_{nk} J_{n+1}^2(\tilde{\xi}_{nk})} \psi'(\tilde{h}\tilde{\xi}_{nk})$$

  Here, $\tilde{\xi}_{nk}$ is k'th zero of $J_n(x)$ function. Note, that $\tilde{h} = h/\pi$ in the original Ogata's notation.

- The convergence properties of the quadrature essentially depends on h. I have found that $h \sim q_T$. For accurate evolution of integrals at different values of q_T I set $h = h_0 s$, where s is defined for intervals of q_T.
- The sum over k is restricted by $N_{\max}$ where $N_{\max}$ is hard coded number, $N_{\max} = 200$.

- The sum over k is evaluated until the sum of absolute values of last four terms is less then *tolerance*. If $M > N_{\max}$ the integral declared divergent, and the trigger is set to ON.

WARNING!
The error for Ogata quadrature is defined by parameters h and M(number of terms in the sum over k). In the parameter M quadrature is double-exponential, i.e. converges fast as M approaches $N_{\max}$. And the convergence of the sum can be simply checked. In the parameter h the quadrature is quadratic (the convergence is rather poor). The convergence to the true value of integral is very expensive especially at large q_T (it requires the complete reevaluation of integral at all nodes). Unfortunately, $N_{\max} \sim h^{-1}$, and has to find the balance value for h. In principle, TMD functions decays rather fast, and suggested default value $h = 0.005$ is trustful. **Nonetheless, we suggest to test other values (*/2) of h to validate the obtained values in your model.**
The adaptive check of convergence will implemented in the future versions.

C. Enumeration of structure functions

List of enumeration of structures functions
N<10000

N	$z_{ff'}$	F_1	F_2	Short description	Gluon req.
0 9999 9998	–	–	–	Test cases (see XVII D)	no
1	$\delta_{\bar{f}f'}\lvert e_f\rvert^2$	f_1	f_1	(unpol.)$p+p\to\gamma$	no
2	$\delta_{ff'}\lvert e_f\rvert^2$	f_1	f_1	(unpol.)$p+\bar{p}\to\gamma$	no
3	$\delta_{\bar{f}f'}\frac{(1-2\lvert ef\rvert s_w^2)^2-4e_f^2s_w^4}{8s_w^2c_w^2}$	f_1	f_1	(unpol.)$p+p\to Z$	no
4	$\delta_{ff'}\frac{(1-2\lvert ef\rvert s_w^2)^2-4e_f^2s_w^4}{8s_w^2c_w^2}$	f_1	f_1	(unpol.)$p+\bar{p}\to Z$	no
5	$\delta_{\bar{f}f'}\frac{z_{ll'}z_{ff'}}{\alpha_{\text{em}}^2}$ given in (2.8) of [1]	f_1	f_1	(unpol.)$p+p\to Z+\gamma$	no
6	$\delta_{ff'}\frac{z_{ll'}z_{ff'}}{\alpha_{\text{em}}^2}$ given in (2.8) of [1]	f_1	f_1	(unpol.)$p+\bar{p}\to Z+\gamma$	no
7	$\frac{1}{4s_w^2}\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	f_1	f_1	(unpol.)$p+p\to W^+$	no
8	$\frac{1}{4s_w^2}\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	f_1	f_1	(unpol.)$p+p\to W^-$	no
9	$\frac{1}{4s_w^2}\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	f_1	f_1	(unpol.)$p+p\to W^\pm$	no
10	$\frac{1}{4s_w^2}\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	f_1	f_1	(unpol.)$p+\bar{p}\to W^+$	no
11	$\frac{1}{4s_w^2}\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	f_1	f_1	(unpol.)$p+\bar{p}\to W^-$	no
12	$\frac{1}{4s_w^2}\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	f_1	f_1	(unpol.)$p+\bar{p}\to W^\pm$	no

13	$\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}$	f_1	f_1	(unpol.)$p+p \to W^+$ (for narrow-width approx.)	no
14	$\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}$	f_1	f_1	(unpol.)$p+p \to W^-$ (for narrow-width approx.)	no
15	$\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}$	f_1	f_1	(unpol.)$p+p \to W^\pm$ (for narrow-width approx.)	no
16	$\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}$	f_1	f_1	(unpol.)$p+\bar{p} \to W^+$ (for narrow-width approx.)	no
17	$\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}$	f_1	f_1	(unpol.)$p+\bar{p} \to W^-$ (for narrow-width approx.)	no
18	$\frac{\lvert V_{ff'}\rvert^2}{4s_w^2}$	f_1	f_1	(unpol.)$p+\bar{p} \to W^\pm$ (for narrow-width approx.)	no
20	$f_g(x_1)f_g(x_2)+h_1^\perp(x_1)h_1^\perp(x_2)$	-	-	(unpol.)$h+h \to H$ Elementary Higgs production	yes
21	$f_g(x_1)f_g(x_2)$	f_1	f_1	(unpol.)$h+h \to H$ Elementary Higgs production. Only unpol.TMDPDF term	yes
22	$h_{1g}^\perp(x_1)h_{1g}^\perp(x_2)$	$h_1^\perp$	$h_1^\perp$	(unpol.)$h+h \to H$ Elementary Higgs production. Only linearly pol.TMDPDF term	yes
101	$\delta_{\bar{f}f'}\lvert e_f\rvert^2$	f_1^h	f_1	(unpol.)$h+p \to \gamma$ (with $h=2$)	no
102	$\delta_{ff'}\lvert e_f\rvert^2$	f_1^h	f_1	(unpol.)$h+\bar{p} \to \gamma$(with $h=2$)	no
103	$\delta_{ff'}\lvert e_f\rvert^2$	f_1^h	f_1	(unpol.)$\bar{h}+p \to \gamma$(with $h=2$). Note, that it is equal 102	no
104	$\delta_{\bar{f}f'}\lvert e_f\rvert^2$	f_1^h	f_1	(unpol.)$\bar{h}+\bar{p} \to \gamma$ (with $h=2$). Note, that it is equal 101	no
1001	$R_{\bar{f}f'}^{Cu}e_f e_{f'}$ see (3.1) of [1]	f_1	f_1^{Cu}	(unpol.)$p+Cu \to \gamma$ (roughly simulates isostructure of Cu, used to describe E288 experiment in [1])	no
1002	$R_{\bar{f}f'}^{^2H}e_f e_{f'}$ see (3.1) of [1] with $Z=1$ and $A=2$	f_1	$f_1^{^2H}$	(unpol.)$p+{}^2H \to \gamma$ (roughly simulates isostructure of ${}^2H$, used to describe E772 experiment)	no
1003	$R_{ff'}^{W}e_f e_{f'}$ see (3.1) of [1] with $Z=74$ and $A=183$	f_1	f_1^W	(unpol.)$\bar{p}+W \to \gamma$ (roughly simulates isostructure of W(tungsten), used to describe E537 experiment)	no
1004	$R_{\bar{f}f'}^{W}e_f e_{f'}$ see (3.1) of [1] with $Z=74$ and $A=183$	f_1	f_1^W	(unpol.)$h+W \to \gamma$, where h is hadron 2 (roughly simulates isostructure of $\pi^- + W$(tungsten), used to describe E537 experiment)	no
2001	$\delta_{ff'}\lvert e_f\rvert^2$	f_1^p	$d_1^{h_1}$	(unpol)$p+\gamma \to h_1$	no
200N	$\delta_{ff'}\lvert e_f\rvert^2$	f_1^p	$d_1^{h_N}$	(unpol)$p+\gamma \to h_N$, for $N=\{1,...,9\}$ (including case 2001)	no

2011	$\delta_{ff'}\|e_f\|^2$	f_1^d	$d_1^{h_1}$	(unpol)$d+\gamma \to h_1$. Here, d is isoscalar target $(p+n)/2$.	no
201N	$\delta_{ff'}\|e_f\|^2$	f_1^d	$d_1^{h_N}$	(unpol)$d+\gamma \to h_N$, for $N = \{1,...,9\}$ (including case 2011). Here, d is isoscalar target $(p+n)/2$.	no
2021	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^p	$d_1^{\bar{h}_1}$	(unpol)$p+\gamma \to \bar{h}_1$	no
202N	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^p	$d_1^{\bar{h}_N}$	(unpol)$p+\gamma \to \bar{h}_N$, for $N = \{1,...,9\}$ (including case 2021)	no
2031	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^d	$d_1^{\bar{h}_1}$	(unpol)$d+\gamma \to \bar{h}_1$. Here, d is isoscalar target $(p+n)/2$.	no
203N	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^d	$d_1^{\bar{h}_N}$	(unpol)$d+\gamma \to \bar{h}_N$, for $N = \{1,...,9\}$ (including case 2031). Here, d is isoscalar target $(p+n)/2$.	no
204N	$\delta_{ff'}\|e_f\|^2$	f_1^n	$d_1^{h_N}$	(unpol)$n+\gamma \to h_N$, for $N = \{1,...,9\}$. n is neutron target $n = p(u \leftrightarrow d)$	no
205N	$\delta_{ff'}\|e_f\|^2$	f_1^n	$d_1^{h_N}$	(unpol)$n+\gamma \to \bar{h}_N$, for $N = \{1,...,9\}$. n is neutron target $n = p(u \leftrightarrow d)$.	no
2101	$\delta_{ff'}\|e_f\|^2$	f_1^p	$d_1^{h_{1+2}}$	(unpol)$p+\gamma \to h_{1+2}$, where $h_{1+2} = h_1 + h_2$	no
2102	$\delta_{ff'}\|e_f\|^2$	f_1^p	$d_1^{h_{1+2+3}}$	(unpol)$p+\gamma \to h_{1+2+3}$, where $h_{1+2+3} = h_1+h_2+h_3$	no
2103	$\delta_{ff'}\|e_f\|^2$	f_1^d	$d_1^{h_{1+2}}$	(unpol)$d+\gamma \to h_{1+2}$, where $h_{1+2} = h_1 + h_2$ and d is isoscalar target $(p+n)/2$.	no
2104	$\delta_{ff'}\|e_f\|^2$	f_1^d	$d_1^{h_{1+2+3}}$	(unpol)$d+\gamma \to h_{1+2+3}$, where $h_{1+2+3} = h_1+h_2+h_3$ and d is isoscalar target $(p+n)/2$.	no
2105	$\delta_{ff'}\|e_f\|^2$	f_1^n	$d_1^{h_{1+2}}$	(unpol)$n+\gamma \to h_{1+2}$, where $h_{1+2} = h_1 + h_2$ and n is neutron target $n = p(u \leftrightarrow d)$.	no
2106	$\delta_{ff'}\|e_f\|^2$	f_1^n	$d_1^{h_{1+2+3}}$	(unpol)$n+\gamma \to h_{1+2+3}$, where $h_{1+2+3} = h_1+h_2+h_3$ and n is neutron target $n = p(u \leftrightarrow d)$.	no
2111	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^p	$d_1^{h_{1+2}}$	(unpol)$p+\gamma \to \bar{h}_{1+2}$, where $h_{1+2} = h_1 + h_2$	no
2112	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^p	$d_1^{h_{1+2+3}}$	(unpol)$p+\gamma \to \bar{h}_{1+2+3}$, where $h_{1+2+3} = h_1+h_2+h_3$	no
2113	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^d	$d_1^{h_{1+2}}$	(unpol)$d+\gamma \to \bar{h}_{1+2}$, where $h_{1+2} = h_1 + h_2$ and d is isoscalar target $(p+n)/2$.	no
2114	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^d	$d_1^{h_{1+2+3}}$	(unpol)$d+\gamma \to \bar{h}_{1+2+3}$, where $h_{1+2+3} = h_1+h_2+h_3$ and d is isoscalar target $(p+n)/2$.	no

2115	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^n	$d_1^{h_{1+2}}$	(unpol)$n+\gamma \to \bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
2116	$\delta_{f\bar{f}'}\|e_f\|^2$	f_1^n	$d_1^{h_{1+2+3}}$	(unpol)$n+\gamma \to \bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and n is neutron target $n=p(u\leftrightarrow d)$.	no

10000⩽N<20000

N	$z_{ff'}$	F_1	F_2	Short description	Gluon req.
10000 19999 19998	–	–	–	Test cases (see XVII D)	no
10001	$-M\delta_{\bar{f}f'}\|e_f\|^2$	$-(f_{1T}^{\perp})^p$	f_1	(Sivers)$p^{\uparrow}+p\to\gamma$	no
10005	$-M\delta_{\bar{f}f'}\dfrac{z_{ll'}z_{ff'}}{\alpha_{\text{em}}^2}$ given in (2.8) of [1]	$-(f_{1T}^{\perp})^p$	f_1	(Sivers)$p^{\uparrow}+p\to Z+\gamma$	no
10007	$-M\dfrac{1}{4s_w^2}\dfrac{\|V_{ff'}\|^2}{4s_w^2}\dfrac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	$-(f_{1T}^{\perp})^p$	f_1	(Sivers)$p^{\uparrow}+p\to W^+$	no
10008	$-M\dfrac{1}{4s_w^2}\dfrac{\|V_{ff'}\|^2}{4s_w^2}\dfrac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2M_W^2}$	$-(f_{1T}^{\perp})^p$	f_1	(Sivers)$p^{\uparrow}+p\to W^-$	no
10101	$+M\delta_{\bar{f}f'}\|e_f\|^2$	f_1^h	$-(f_{1T}^{\perp})^p$	(Sivers)$h+p^{\uparrow}\to\gamma$ (with $h=2$)	no
10102	$+M\delta_{ff'}\|e_f\|^2$	f_1^h	$-(f_{1T}^{\perp})^{\bar{p}}$	(Sivers)$h+\bar{p}^{\uparrow}\to\gamma$(with $h=2$)	no
10103	$+M\delta_{ff'}\|e_f\|^2$	f_1^h	$-(f_{1T}^{\perp})^p$	(Sivers)$\bar{h}+p^{\uparrow}\to\gamma$(with $h=2$). Note, that it is equal 10102	no
10104	$+M\delta_{\bar{f}f'}\|e_f\|^2$	f_1^h	$-(f_{1T}^{\perp})^{\bar{p}}$	(Sivers)$\bar{h}+\bar{p}^{\uparrow}\to\gamma$ (with $h=2$). Note, that it is equal 10101	no
12001	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{h_1}$	(Sivers)$p^{\uparrow}+\gamma\to h_1$	no
1200N	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{h_N}$	(Sivers)$p^{\uparrow}+\gamma\to h_N$, for $N=\{1,...,9\}$ (including case 12001)	no
12011	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_1}$	(Sivers)$d^{\uparrow}+\gamma\to h_1$. Here, d is isoscalar target $(p+n)/2$.	no
1201N	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_N}$	(Sivers)$d^{\uparrow}+\gamma\to h_N$, for $N=\{1,...,9\}$ (including case 12011). Here, d is isoscalar target $(p+n)/2$.	no
12021	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{\bar{h}_1}$	(Sivers)$p^{\uparrow}+\gamma\to\bar{h}_1$	no
1202N	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{\bar{h}_N}$	(Sivers)$p^{\uparrow}+\gamma\to\bar{h}_N$, for $N=\{1,...,9\}$ (including case 12021)	no

12031	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{\bar{h}_1}$	(Sivers)$d^{\uparrow}+\gamma\to\bar{h}_1$. Here, d is isoscalar target $(p+n)/2$.	no
1203N	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{\bar{h}_N}$	(Sivers)$d^{\uparrow}+\gamma\to\bar{h}_N$, for $N=\{1,...,9\}$ (including case 12031). Here, d is isoscalar target $(p+n)/2$.	no
1204N	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_N}$	(Sivers)$n^{\uparrow}+\gamma\to h_N$, for $N=\{1,...,9\}$, n is neutron target $n=p(u\leftrightarrow d)$.	no
1205N	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{\bar{h}_N}$	(Sivers)$n^{\uparrow}+\gamma\to\bar{h}_N$, for $N=\{1,...,9\}$, n is neutron target $n=p(u\leftrightarrow d)$.	no
12101	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{h_{1+2}}$	(Sivers)$p^{\uparrow}+\gamma\to h_{1+2}$, where $h_{1+2}=h_1+h_2$	no
12102	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{h_{1+2+3}}$	(Sivers)$p^{\uparrow}+\gamma\to h_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$	no
12103	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_{1+2}}$	(Sivers)$d^{\uparrow}+\gamma\to h_{1+2}$, where $h_{1+2}=h_1+h_2$ and d is isoscalar target $(p+n)/2$.	no
12104	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_{1+2+3}}$	(Sivers)$d^{\uparrow}+\gamma\to h_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and d is isoscalar target $(p+n)/2$.	no
12105	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^n$	$d_1^{h_{1+2}}$	(Sivers)$n^{\uparrow}+\gamma\to h_{1+2}$, where $h_{1+2}=h_1+h_2$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
12106	$-M\delta_{ff'}\|e_f\|^2$	$(f_{1T}^{\perp})^n$	$d_1^{h_{1+2+3}}$	(Sivers)$n^{\uparrow}+\gamma\to h_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
12111	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{h_{1+2}}$	(Sivers)$p^{\uparrow}+\gamma\to\bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$	no
12112	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^p$	$d_1^{h_{1+2+3}}$	(Sivers)$p^{\uparrow}+\gamma\to\bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$	no
12113	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_{1+2}}$	(Sivers)$d^{\uparrow}+\gamma\to\bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$ and d is isoscalar target $(p+n)/2$.	no
12114	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^d$	$d_1^{h_{1+2+3}}$	(Sivers))$d^{\uparrow}+\gamma\to\bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and d is isoscalar target $(p+n)/2$.	no
12115	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^n$	$d_1^{h_{1+2}}$	(Sivers)$n^{\uparrow}+\gamma\to\bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
12116	$-M\delta_{f\bar{f}'}\|e_f\|^2$	$(f_{1T}^{\perp})^n$	$d_1^{h_{1+2+3}}$	(Sivers)$n^{\uparrow}+\gamma\to\bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
$\downarrow\downarrow\downarrow$ Enumeration for F_{LT} in SIDIS is same as for F_{UT} with $(12\to13)$ $\downarrow\downarrow\downarrow$					
13001	$+M\delta_{ff'}\|e_f\|^2$	$(g_{1T}^{\perp})^p$	$d_1^{h_1}$	(wgt)$p^{\uparrow}+\gamma\to h_1$	no
1300N	$+M\delta_{ff'}\|e_f\|^2$	$(g_{1T}^{\perp})^p$	$d_1^{h_N}$	(wgt)$p^{\uparrow}+\gamma\to h_N$, for $N=\{1,...,9\}$ (including case 12001)	no

13011	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_1}$	(wgt)$d^{\uparrow}+\gamma\to h_1$. Here, d is isoscalar target $(p+n)/2$.	no
1301N	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_N}$	(wgt)$d^{\uparrow}+\gamma\to h_N$, for $N=\{1,...,9\}$ (including case 12011). Here, d is isoscalar target $(p+n)/2$.	no
13021	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^p$	$d_1^{\bar{h}_1}$	(wgt)$p^{\uparrow}+\gamma\to\bar{h}_1$	no
1302N	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^p$	$d_1^{\bar{h}_N}$	(wgt)$p^{\uparrow}+\gamma\to\bar{h}_N$, for $N=\{1,...,9\}$ (including case 12021)	no
13031	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{\bar{h}_1}$	(wgt)$d^{\uparrow}+\gamma\to\bar{h}_1$. Here, d is isoscalar target $(p+n)/2$.	no
1303N	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{\bar{h}_N}$	(wgt)$d^{\uparrow}+\gamma\to\bar{h}_N$, for $N=\{1,...,9\}$ (including case 12031). Here, d is isoscalar target $(p+n)/2$.	no
1304N	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_N}$	(wgt)$n^{\uparrow}+\gamma\to h_N$, for $N=\{1,...,9\}$, n is neutron target $n=p(u\leftrightarrow d)$.	no
1305N	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{\bar{h}_N}$	(wgt)$n^{\uparrow}+\gamma\to\bar{h}_N$, for $N=\{1,...,9\}$, n is neutron target $n=p(u\leftrightarrow d)$.	no
13101	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^p$	$d_1^{h_{1+2}}$	(wgt)$p^{\uparrow}+\gamma\to h_{1+2}$, where $h_{1+2}=h_1+h_2$	no
13102	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^p$	$d_1^{h_{1+2+3}}$	(wgt)$p^{\uparrow}+\gamma\to h_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$	no
13103	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_{1+2}}$	(wgt)$d^{\uparrow}+\gamma\to h_{1+2}$, where $h_{1+2}=h_1+h_2$ and d is isoscalar target $(p+n)/2$.	no
13104	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_{1+2+3}}$	(wgt)$d^{\uparrow}+\gamma\to h_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and d is isoscalar target $(p+n)/2$.	no
13105	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^n$	$d_1^{h_{1+2}}$	(wgt)$n^{\uparrow}+\gamma\to h_{1+2}$, where $h_{1+2}=h_1+h_2$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
13106	$+M\delta_{ff'}\lvert e_f\rvert^2$	$(f_{1T}^{\perp})^n$	$d_1^{h_{1+2+3}}$	(wgt)$n^{\uparrow}+\gamma\to h_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
13111	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^p$	$d_1^{h_{1+2}}$	(wgt)$p^{\uparrow}+\gamma\to\bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$	no
13112	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^p$	$d_1^{h_{1+2+3}}$	(wgt)$p^{\uparrow}+\gamma\to\bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$	no
13113	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_{1+2}}$	(wgt)$d^{\uparrow}+\gamma\to\bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$ and d is isoscalar target $(p+n)/2$.	no
13114	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^d$	$d_1^{h_{1+2+3}}$	(wgt)$d^{\uparrow}+\gamma\to\bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and d is isoscalar target $(p+n)/2$.	no

13115	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^n$	$d_1^{h_{1+2}}$	(wgt)$n^{\uparrow}+\gamma\to\bar{h}_{1+2}$, where $h_{1+2}=h_1+h_2$ and n is neutron target $n=p(u\leftrightarrow d)$.	no
13116	$+M\delta_{f\bar{f}'}\lvert e_f\rvert^2$	$(g_{1T}^{\perp})^n$	$d_1^{h_{1+2+3}}$	(wgt)$n^{\uparrow}+\gamma\to\bar{h}_{1+2+3}$, where $h_{1+2+3}=h_1+h_2+h_3$ and n is neutron target $n=p(u\leftrightarrow d)$.	no

20000⩽N<30000

N	$z_{ff'}$	F_1	F_2	Short description	Gluon req.
20000 29999 29998	–	–	–	Test cases (see XVII D)	no

30000⩽N

N	$z_{ff'}$	F_1	F_2	Short description	Gluon req.
30000 39999 39998	–	–	–	Test cases (see XVII D)	no

Notes on parameters and notation:
s_w^2, M_Z, Γ_Z, etc. are defined in `constants`
f_1-unpolarized TMDPDF
d_1-unpolarized TMDFF
Everywhere proton is expected to be hadron number 1.

D. Tests

The test options are called by process number $N+0$, $N+9999$ and $N+9998$. In these case, the integrand set by a fixed function. In the case $N+0$

$$N+0: \qquad \sum zFF \to e^{-0.2b}. \tag{17.4}$$

The cases $N+9999$ and $N+9998$ are dependent on parameters of the function:

$$N+9999: \qquad \sum zFF \to e^{-\mu b}(1+x_1b^2+x_2b^4), \tag{17.5}$$
$$N+9998: \qquad \sum zFF \to e^{-\mu b^2}(1+x_1b^2+x_2b^4). \tag{17.6}$$

In all cases the Hankel integral could be evaluated. They read

$$9999: \qquad F = \frac{1}{2\mu^2(1+X)^{3/2}}\left(1+\frac{x_1}{\mu^2}\frac{6-9X}{(1+X)^2}+\frac{15x_2}{\mu^4}\frac{8-40X+15X^2}{(1+X)^4}\right), \tag{17.7}$$

$$19999: \qquad F = \frac{3\sqrt{X}}{2\mu^3(1+X)^{5/2}}\left(1+\frac{5x_1}{\mu^2}\frac{4-3X}{(1+X)^2}+\frac{105x_2}{\mu^4}\frac{8-20X+5X^2}{(1+X)^4}\right), \tag{17.8}$$

$$29999: \qquad F = \frac{15X}{2\mu^4(1+X)^{7/2}}\left(1+\frac{21x_1}{\mu^2}\frac{2-X}{(1+X)^2}+\frac{63x_2}{\mu^4}\frac{48-80X+15X^2}{(1+X)^4}\right), \tag{17.9}$$

$$39999: \qquad F = \frac{105X^{3/2}}{2\mu^5(1+X)^{9/2}}\left(1+\frac{9x_1}{\mu^2}\frac{8-3X}{(1+X)^2}+\frac{495x_2}{\mu^4}\frac{16-20X+3X^2}{(1+X)^4}\right), \tag{17.10}$$

where $X = q_T^2/\mu^2$.

$$9998: \qquad F = \frac{e^{-Y}}{4\mu}\left(1+\frac{x_1}{\mu}(1-Y)+\frac{x_2}{\mu^2}(2-4Y+Y^2)\right), \tag{17.11}$$

$$19998: \qquad F = \frac{e^{-Y}\sqrt{Y}}{4\mu^{3/2}}\left(1+\frac{x_1}{\mu}(2-Y)+\frac{x_2}{\mu^2}(6-6Y+Y^2)\right), \tag{17.12}$$

$$29998: \qquad F = \frac{e^{-Y}Y}{4\mu^2}\left(1+\frac{x_1}{\mu}(3-Y)+\frac{x_2}{\mu^2}(12-8Y+Y^2)\right), \tag{17.13}$$

$$39998: \qquad F = \frac{e^{-Y}Y^{3/2}}{4\mu^{5/2}}\left(1+\frac{x_1}{\mu}(4-Y)+\frac{x_2}{\mu^2}(20-10Y+Y^2)\right), \tag{17.14}$$

where $Y = q_T^2/(4\mu)$.

XVIII. `TMDX_DY` MODULE

This module evaluates cross-sections with the Drell-Yan-like kinematics. I.e. it expects the following kinematic input, (s, Q, y) which defines the variables x's, etc.

The general structure of the cross-section is

$$\Delta\sigma(q_T) = prefactor1 \int_{bin} dX\ prefactor2\ \times F, \tag{18.1}$$

where

$$dX = dQ^2 dy dq_T^2.$$

So, for a small bin

$$\Delta\sigma(q_T) \approx \Delta X \frac{d\sigma}{dX}, \qquad \Delta X = (Q^2_{\max} - Q^2_{\min})(y_{\max} - y_{\min})(q^2_{T\max} - q^2_{T\min}). \tag{18.2}$$

Prefactor1 is related to the definition of the phase-space, while *prefactor2* is about process and cross-section. Generally, the *prefactor2* $\times F$ is

$$prefactor2 \times F = (\text{cuts for lepton pair}) \times H(Q, \mu_H) F(Q, q_T, x_1, x_2, \mu, \zeta_1, \zeta), \tag{18.3}$$

where F is defined in (17.1). There are following feature of current implementation

- In the current version the scaling variables are set as

$$\mu^2 = \zeta_1 = \zeta_2 = Q^2. \tag{18.4}$$

  Currently, it is hard coded and could not be easily modified. However, there is a possibility to vary the value of μ as $\mu = c_2 Q$, where c_2 is variation parameter (see sec.XVIII F).

- For the definition of *(cuts for lepton pair)*-function see [1]. It is evaluated within module `LeptonCutsDY.f90`. The presence of cuts, and their parameters are set by the `SetCuts` subroutine.

- The expression for the hard factor H is taken from [11]. It is the function of $\ln(Q/\mu_H)$ and $a_s(\mu_H)$). Since in the current realization $\mu_H = Q$, the logarithm is replaced by $\ln(c_2)$, where c_2 is the variation constant.

This section is to be updated by definition of kinematics

List of available commands

Command	Type	Sec.	Short description
`TMDX_DY_Initialize(order)`	subrout.	XVIII A	Initialization of module.
`TMDX_DY_ShowStatistic()`	subrout.	–	Print current statistic on the number of calls.
`TMDX_DY_ResetCounters()`	subrout.	–	Reset intrinsic counters of the module.
`TMDX_DY_SetProcess(p)`	subrout.	XVIII B	Set the process
`TMDX_DY_SetCuts(inc,pT,eta1,eta2)`	subrout.	XVIII B	Set the current evaluation of cut for lepton pair.
`TMDX_DY_XSetup(s,Q,y)`	subrout.	XVIII B	Set the kinematic variables
`TMDX_DY_SetScaleVariation(c2)`	subrout.	–	Set new value for the scale-variation constant c_2.
`CalcXsec_DY...`	subrout.	XVIII C	Evaluates cross-section. Many overloaded versions see sec.XVIII C.
`xSec_DY(X,proc,s,qT,Q,y,iC,CutP,Num)`	subrout.	XVIII C	Evaluates cross-section completely integrated over the bin. Can be called without preliminary `...Set...`'s.
`xSec_DY_List(X,proc,s,qT,Q,y,iC,CutP,Num)`	subrout.	XVIII C	Evaluates cross-section completely integrated over the bin over the list. Can be called without preliminary `...Set...`'s.

A. Initialization

Prior the usage module is to be initialized (once per run) by
`call TMDX_DY_Initialize(file)`
The order of used coefficient function is set in `constant-file` by `'LO'`, `'LO+'`, `'NLO'`, `'NLO+'`,`'NNLO'` or `'NNLO+'`, that corresponds to a_s^0, a_s^0, a_s^1, a_s^1, a_s^2, or a_s^2.

B. Setting up the parameters of cross-section

Prior to evaluation of cross-section one must declare which process is considered and to set up the kinematics. The declaration of the process is made by
`call TMDX_DY_SetProcess(p1,p2,p3)`
`call TMDX_DY_SetProcess(p0)`
where `p0=(/p1,p2,p3/)` and

`p1` (integer) Defines the $prefactor1$ that contains the phase space elements.

`p2` (integer) Defines the $prefactor2$ that contains the universal part of factorization formula.

`p3` (integer) Defines the structure function F. See sec.XVII C.

Alternatively, process can be declared by a shorthand version (not updated any more, too many options)
`call TMDX_DY_SetProcess(p)`

where `p`(integer) corresponds to particular combinations of `p1,p2,p3`. See table XVIII H.

The kinematic of the process is declared by

`call TMDX_DY_XSetup(s,Q,y)`

where `s` is the Mandelshtam variable s, `Q` is hard virtuality Q, `y` is the rapidity parameter y. Note, that these values will be used for the evaluation of the cross-section. In the case, the integration over the bin is made these values are overridden. Also in the case of the parallel computation these variables are overridden. So, often there is little sense in this command, nonetheless it set initial values.

All non-perturbative parameters are defined in the lower-level modules, and could be set via `artemide_control` or directly in the module.

Finally, one must specify the fiducial cuts on the lepton pair. It is made by calling

`call SetCuts(inc,pT,eta1,eta2)`

where parameter `inc` is (logical). If `inc=.true.` the evaluation of cuts will be done, otherwise it will be ignored. The cuts are defined as

$$|l_T|, |l'_T| < \texttt{pT}, \qquad \texttt{eta1} < \eta, \eta', \texttt{eta2}, \tag{18.5}$$

where l and l' are the momenta of produced leptons, with l_T's being their transverse components and η's being their rapidities. For accurate definition of the cut-function see sec.2.6 of [1] (particularly equations (2.40)-(2.42)). For asymmetric cuts use `call SetCuts(inc,pT1,pT2,eta1,eta2)`.

C. Cross-section evaluation

After the parameters of cross-section are set up, the values of the cross-section at different q_T can be obtained by

`call CalcXsec_DY(X,qt)`

where `X` is real*8 variable where cross-section will be stored, `qt` is real*8 is the list of values of q_T's at which the `X` is to be calculated. There exists an overloaded version with `X` (real*8)(1:N) and `qT`(real*8)(1:N), which evaluates the array of crossections over array of q_T.

Typically, one needs to integrate over bin. There is a whole set of subroutines which evaluate various integrals over bin they are

Subroutine	integral	Comment
`CalcXsec_DY(X,qt)`	-	Just cross-section at given point.
`CalcXsec_DY_Yint(X,qt)`	$\int_{-y_0}^{y_0} dy d\sigma(q_T)$	y_0 is maximum allowed y by kinematics, $y_0 = \ln(s/Q^2)/2$.
`CalcXsec_DY_Yint(X,qt,yMin,yMax)`	$\int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$	$\lvert y_{\max/\min}\rvert < y_0$
`CalcXsec_DY_Qint (X,qt,Qmin,Qmax)`	$\int_{Q_{\min}}^{Q_{\max}} 2QdQ d\sigma(q_T)$	
`CalcXsec_DY_Qint_Yint (X,qt,Qmin,Qmax)`	$\int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{-y_0}^{y_0} dy d\sigma(q_T)$	
`CalcXsec_DY_Qint_Yint (X,qt,Qmin,Qmax,yMin,yMax)`	$\int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$	
`CalcXsec_DY_PTint_Qint_Yint (X,qtmin,qtmax,Qmin,Qmax)`	$\int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{-y_0}^{y_0} dy d\sigma(q_T)$	Integration over q_T is Simpsons by default number of sections.
`CalcXsec_DY_PTint_Qint_Yint (X,qtmin,qtmax,Qmin,Qmax,yMin,yMax)`	$\int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$	Integration over q_T is Simpsons by default number of sections.
`CalcXsec_DY_PTint_Qint_Yint (X,qtmin,qtmax,Qmin,Qmax,N)`	$\int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{-y_0}^{y_0} dy d\sigma(q_T)$	Integration over q_T is Simpsons by N-section.
`CalcXsec_DY_PTint_Qint_Yint (X,qtmin,qtmax,Qmin,Qmax,yMin,yMax,N)`	$\int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$	Integration over q_T is Simpsons by N-section.
More to be added		

Note 1: Each command has overloaded version with arrays for X and qt.
Note 2: Cross-section integrated over q_T bins have overloaded versions with `X`, `qtmin` and `qtmax` being arrays. Then the integral is done for `X`(i) from `qtmin(i)` to `qtmax(i)`.
Note 2+: There exists an overloaded version for `CalcXsec_DY_PTint_Qint_Yint` with `X`, `qtmin` and `qtmax`, `yMin`, `yMax` being arrays. Then the integral is done for `X`(i) from `qtmin(i)` to `qtmax(i)`, `yMin(i)` to `yMax(i)`.
Note 3: There exist special overloaded case for integrated over q_T-bins, with successive bins. I.e. for bins that adjust to each other. In this case only one argument `qtlist` is required (instead of `qtmin`,`qtmax`). E.g. `CalcXsec_DY_PTint_Qint_Yint (X,qtList,Qmin,Qmax)`. The length of `qtlist` must be larger then the length of `X` by one. The integration for `X(i)` is done from `qtlist(i)` till `qtlist(i+1)`. This function saves boundary values and therefore somewhat faster than the usual evaluation. (Improvement takes a place is only for un-parallel calculation).
Note 4: Integration over p_T is numerically problematic at $p_T \to 0$. Thus, the integration over $p_T < 0.1$MeV is cut out. Anyway, contribution of this region is negligible, due to prefactor p_T.

Take care that every next function is heavier to evaluate then the previous one. The integrations over Q and y are adaptive Simpsons. We have found that it is the fastest (adaptive) method for typical cross-sections with tolerance $10^{-3} - 10^{-4}$. Naturally, it is not suitable for higher precision, which however is not typically required. The integration over p_T is not adaptive, since typically p_T-bins are rather smooth and integral is already accurate if approximated by 4-8 points (default minimal value is set in `constants`, which is automatically increased for larger bins). For unexceptionally large q_T-bins, or very rapid behavior we suggest to use overloaded versions with manual set of N (number of integral sections).

There is a subroutine that evaluate cross-section without preliminary call of `SetCuts`,`TMDX_DY_SetProcess`,`TMDX_DY_XSetup`. It is called `xSec_DY` and it have the following interface:

`xSec_DY(X,process,s,qT,Q,y,includeCuts,CutParameters,Num)`

where

- `X` is (real*8) value of cross-section.
- `process` is integer `p`, or (integer)array `(p1,p2,p3)`.
- `s` is Mandelshtam variable s
- `qT` is (real*8)array `(qtmin,qtmax)`
- `Q` is (real*8)array `(Qmin,Qmax)`
- `y` is (real*8)array `(ymin,ymax)`
- `includeCuts` is logical
- `CutParameters` is (real*8) array `(k1,k2,eta1,eta2)` **OPTIONAL**
- `Num` is even integer that determine number of section in q_T integral **OPTIONAL**

Note, that optional variables could be omit during the call.

IMPORTANT: Practically, the call of this function coincides with `X` evaluated by the following set of commands

```
call TMDX_DY_SetProcess(process)
call TMDX_DY_XSetup(s,any,any)
call SetCuts(includeCuts,k1,k2,eta1,eta2)
call CalcXsec_DY_PTint_Qint_Yint (X,qtmin,qtmax,Qmin,Qmax,yMin,yMax,Num)
```

However, there is an important difference: the values of `s, process, cutParameters` which are set by routines `..Set..`, are global. For that reason such approach could not be used in a parallel computation. Contrary, in the function `xSec_DY` these variables are set locally and thus this function can be used in parallel computations.

?BUG?: Take care that some Fortran compilers, do not understand the usage of arrays directly within function with optional arguments. E.g. `xSec_DY(p,s,(/q1,q2/),Q,y,iC,CutParameters=cc)`. Although it compiles without problems but lead to a freeze during the calculation. In this case, it helps to define: `qT=(/q1,q2/)` and call `xSec_DY(p,s,qT,Q,y,iC,CutParameters=cc)`. The same problem can appear if `xSec_DY` is used as argument of another function. I guess there is some problem with memory references.

There is also analogous subroutine that evaluate cross-section by list. It is called `xSec_DY_List` and it have the following interface:

`xSec_DY_List(X,process,s,qT,Q,y,includeCuts,CutParameters,Num)`

All variables are analogues to those of `xSec_DY`, but should come in lists, i.e. `X` is (1:n), `process` is (1:n,1:3), `s` is (1:n), `qT`,`Q`,`y` are (1:n,1:2), `includeCuts` is (1:n), `CutParameters` is (1:n,1:4), and `Num` is (1:n). **Only the `Num` argument is OPTIONAL arguments. The argument `CutParameters` must be presented**. This command compiled by OPENMP, so runs in parallel on multi-core computers.

D. Parallel computation

Currently, the parallel evaluation is used within the commands `CalcXsec_DY_...` with array variable `qt` (see **Note 1** in sec.TMDX:xsec). Basically, in this case, individual values for the list of cross-sections are evaluated in parallel. Unfortunately the parallel scaling-rate is not very high, on 8 processors it is about 400%-500% only.

The parallelization is made with OPENMP library. To make the parallel computation possible, add `-fopenmp` option in the compilation instructions. The maximum number of allowed threads is set `constants-file`.

> **WARNING:** the parallel operation is possible only together with grid option, otherwise it will cause a running condition in TMD modules (which typically results into the program crush). There is no check for grid option trigger, check it manually.

E. `LeptonCutsDY`

The calculation of cut prefactor is made in `LeptonCutsDY.f90`. It has two public procedures: `SetCutParameters`, and `CutFactor`.

- The subroutine `SetCutParameters(kT,eta1,eta2)` set a default version of cut parameters: $p_{1,2} < k_T$ and $\eta_1 < \eta < \eta_2$.
- The overloaded version of the subroutine `SetCutParameters(k1,k2,eta1,eta2)` set a default version of asymmetric cut parameters. $p_1 < k_1$,$p_2 < k_2$ and $\eta_1 < \eta < \eta_2$.
- Function `CutFactor(qT,Q_in,y_in, CutParameters)` calculates the cut prefactor at the point q_T, Q, y. The argument `CutParameters` is **optional**. If it is not present, cut parameters are taken from default values (which are set by `SetCutParameters`). `CutParameters` is array `(/ k1,k2,eta1,eta2/)`. The usage of global definition for `CutParameters` is not recommended, since it can result into running condition during parallel computation. This is interface is left fro compatibility with earlier version of `artemide`.

F. Variation of scale

TO BE WRITTEN

G. Options for evaluation of DY-like cross-section

1. π^2-resummation

The coefficient function of DY-like cross-section $|C_V|^2$ is evaluated at $(-q^2)$ (space-like momentum). Thus, it has contributions $\sim \pi^2$, which could be large, especially in the case of Higgs-boson production, see discussion in refs.Ahrens:2008nc,Ahrens:2008qu. These corrections could be resummed by RG technique [14]. So,

$$|C_V(-q^2)|^2 \to U_\pi |C_V(q^2)|^2, \tag{18.6}$$

where U_π is the resummation exponent for πa_s-correction. At LO it reads [14]

$$U_\pi = \exp\left(\frac{\Gamma_0}{2a_s\beta_0^2}[2a\arctan a - \ln(1+a^2)]\right), \qquad a = \pi a_s. \tag{18.7}$$

To use the π-resummed coefficient function switch the corresponded option in `constants-file` (Option 9.A.p3).

2. Power corrections

There are many sources of power corrections. For a moment there is no systematic studies of power corrections for TMD factorization. Nonetheless I include some options in `artemide`, and plan to make more systematic treatment in the future.

Exact values of $x_{1,2}$ for DY: The TMD distributions enter the cross-section with x_1 and x_2 equal to

$$x_1 = \frac{q^+}{p_1^+} = \frac{Q}{\sqrt{s}} e^y \sqrt{1 + \frac{\boldsymbol{q}_T^2}{Q^2}} \simeq \frac{Q}{\sqrt{s}} e^y, \tag{18.8}$$
$$x_1 = \frac{q^+}{p_1^+} = \frac{Q}{\sqrt{s}} e^{-y} \sqrt{1 + \frac{\boldsymbol{q}_T^2}{Q^2}} \simeq \frac{Q}{\sqrt{s}} e^{-y}.$$

Traditionally, the corrections $\sim q_T/Q$ are dropped, since they are power corrections. Nonetheless, they could be included since they have different sources in comparison to power corrections to the factorized cross-section. Usage of one or another version of $x_{1,2}$ is switched in `constants-file` (Option 9.A.p2).

Exact values for hard scale for DY: The TMD hard coefficient function contains $\ln(|2q^+q^-|/\mu^2)$ which is usually approximated by $\ln(Q^2)$. Similarly, the *zeta*-scales are normalized as $\zeta\bar{\zeta} = (2q^+q^-)^2$, which is usually approximated by Q^4. So, the exact values of scales are

$$\mu = \sqrt{Q^2 + \boldsymbol{q}_T^2}, \qquad \zeta_1 = \zeta_2 = Q^2 + \boldsymbol{q}_T^2. \tag{18.9}$$

Traditionally, the corrections $\sim q_T/Q$ are dropped, since they are power corrections. Nonetheless, they could be included since they have different sources in comparison to power corrections to the factorized cross-section. Usage of one or another version of *scales* is switched in `constants-file` (Option 9.A.p4).

H. Enumeration of processes

List of enumeration for $prefactor1$
`p1`

`p1`	$prefactor1$	Short description
1	1	$dX = dydQ^2dq_T^2$.
2	$\frac{2\sqrt{Q^2+\boldsymbol{q}_T^2}}{\sqrt{s}} \cosh y$	$dX = dx_F dQ^2 dq_T^2$ where $x_F = \frac{2\sqrt{Q^2+\boldsymbol{q}_T^2}}{\sqrt{s}} \sinh y$ is Feynman x. **Important:** In this case the integration y is preplaced by the integration over x_F. I.e. `..._Yint`$(..., a, b, ..)$ which usually evaluates $\int_a^b dy$ evaluates $\int_a^b dx_F$.

List of enumeration for $prefactor2$
`p2`

p2	$prefactor2$	Short description
1	$\frac{4\pi}{9}\frac{\alpha_{\rm em}^2(Q)}{sQ^2}\lvert C_V^{DY}(c_2Q)\rvert^2 R\times \mathrm{cut}(q_T)$	DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$. $\mathrm{cut}(q_T)$ is the weight for the lepton tensor with fiducial cuts (see sec.2.6 in [1]). Corresponds to DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$. **Process declared $y\leftrightarrow -y$ symmetric.** The result is in pb/GeV^2
2	$\frac{4\pi}{9}\frac{\alpha_{\rm em}^2(Q)}{sQ^2}\lvert C_V^{DY}(c_2Q)\rvert^2 R\times \mathrm{cut}(q_T)$	DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$. $\mathrm{cut}(q_T)$ is the weight for the lepton tensor with fiducial cuts (see sec.2.6 in [1]). **Process declared $y\leftrightarrow -y$ non-symmetric.** The result is in pb/GeV^2
3	$\frac{4\pi^2}{3}\frac{\alpha_{\rm em}(Q)}{s}Br_Z\lvert C_V^{DY}(c_2Q)\rvert^2 R\times \mathrm{cut}(q_T)$	DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$ for the Z-boson production in the narrow width approximation. $Br_Z=0.03645$ is Z-boson branching ration to leptons. $\mathrm{cut}(q_T)$ is the weight for the lepton tensor with fiducial cuts (see sec.2.6 in [1]). **Process declared $y\leftrightarrow -y$ symmetric.** The result is in pb/GeV^2
4	$\frac{4\pi^2}{3}\frac{\alpha_{\rm em}(Q)}{s}Br_W\lvert C_V^{DY}(c_2Q)\rvert^2 R\times \mathrm{cut}(q_T)$	DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$ for the W-boson production in the narrow width approximation. $Br_W=0.1086$ is W-boson branching ration to leptons. $\mathrm{cut}(q_T)$ is the weight for the lepton tensor with fiducial cuts (see sec.2.6 in [1]). Use together with `p3`=13,...,18. In this case, Q is be M_Z. The result is in pb/GeV^2
5	$\frac{2}{\pi}\frac{\pi m_H^2 a_s^2(Q)}{36sv^2}C_t^2(m_t,Q)\lvert C_g(Q)\rvert^2\lvert A(x_t)\rvert^2$	Cross-section $\frac{d\sigma}{dyd(q_T^2)}$ for the exclusive Higgs-boson production. The result is in pb/GeV^2. For definition of functions see [13].

Here $R=0.3893379\cdot 10^9$ the transformation factor from GeV to pb.

List of shorthand enumeration for processes

`p`	`p1`	`p2`	`p3`	Description
1	1	1	5	$p+p\to Z+\gamma^*\to ll$. Standard DY process measured in the vicinity of the Z-peak. E.g. for LHC measurements.
2	1	1	6	$p+\bar{p}\to Z+\gamma^*\to ll$. Standard DY process measured in the vicinity of the Z-peak. E.g. for Tevatron measurements.

XIX. TMDX_SIDIS MODULE

This module evaluates cross-sections with the SIDIS-like kinematics. I.e. it expects the following kinematic input, (Q, x, z) or (Q, y, z) or (x, y, z), that are equivalent.

The general structure of the cross-section is

$$\frac{d\sigma}{dX} = d\sigma(q_T) = prefactor1 \int [bin]\ prefactor2\ \times F, \tag{19.1}$$

where $dX \sim dl'$, with l' momentum of scattered lepton. For better definition of parameters see sec.XIX G.

List of available commands

Command	Type	Sec.	Short description
`TMDX_SIDIS_Initialize(file,prefix)`	subrout.	XVIII A	* Initialization of module.
`TMDX_SIDIS_ResetCounters()`	subrout.	–	* Reset intrinsic counters of module.
`TMDX_SIDIS_ShowStatistic()`	subrout.	–	* Print current statistic on the number of calls.
`TMDX_SIDIS_SetProcess(p)`	subrout.	XIX A	Set the process
`TMDX_SIDIS_XSetup(s,z,x,Q,mTARGET,mPRODUCT)`	subrout.	XIX A	Set the kinematic variables (`mTARGET` and `mPRODUCT` are optional variables)
`TMDX_SIDIS_SetCuts(inc,yMin,yMax,W2)`	subrout.	XIX C	Set global values of cuts
`TMDX_SIDIS_SetScaleVariation(c2)`	subrout.	–	* Set new values for the scale-variation constants.
`CalcXsec_SIDIS...`	subrout.	XIX B	Evaluates cross-section. Many overloaded versions see sec.XIX B.
`xSec_SIDIS(X,process,s,pT,z,x,Q)`	subrout.	XIX B	Evaluates cross-section completely integrated over the bin. Can be called without preliminary `...Set...`'s.
`xSec_SIDIS_List(X,process,s,pT,z,x,Q)`	subrout.	XIX B	Evaluates cross-section completely integrated over the bin over the list. Can be called without preliminary `...Set...`'s.

*)The structure of the module repeats the structure of `TMDX_DY` module, with the main change in the kinematic definition. Most part of routines has the same input and output with only replacement `_DY`→`_SIDIS`. We do not comment such commands. They marked by $*$ in the following table.

A. Setting up the parameters of cross-section

Prior to evaluation of cross-section one must declare which process is considered and to set up the kinematics.

The declaration of the process is made by
`call TMDX_SIDIS_SetProcess(p1,p2,p3)`
`call TMDX_SIDIS_SetProcess(p0)`
where `p0=(/p1,p2,p3/)` and

`p1` (integer) Defines the $prefactor1$ that contains the phase space elements, and generally experimental dependent. Could be set outside of bin-integration.

`p2` (integer) Defines the $prefactor2$ that contains the universal part of factorization formula. Participate in the bin-integration

`p3` (integer) Defines the structure function F. See sec.XVII C.

Alternatively, process can be declared by shorthand version
`call TMDX_SIDIS_SetProcess(p)`
where `p`(integer) corresponds to particular combinations of `p1,p2,p3`. See table.

The kinematic of the process is declared by variables (s, Q, x, z)
`call TMDX_SIDIS_XSetup(s,Q,x,z)`
where `s` is Mandelshtam variable s, `Q` is hard virtuality Q, and (x,z) are standard SIDIS variables. Additionally one can declare masses of target and produced hadrons, by using optional variables
`call TMDX_SIDIS_XSetup(s,Q,x,z,mTARGET,mPRODUCT)`
where corresponding masses are given in GeV. Note, that during initialization procedure the target and produced masses are set from `constants`-file. These values are used or not used according to triggers for 'mass corrections' set in `constants`-file.

All non-perturbative parameters are defined in the `TMDs` and lower-level modules. For user convenience there is a subroutine, which passes the values of parameters to `TMDs`. It is
`call TMDX_SIDIS_SetNPParameters(lambda/)`
where `lambda` is real*8(1:number of parameters)/ or `n` is the number of replica. See also XVI B.

B. Cross-section evaluation

After the parameters of cross-section are set up, the values of the cross-section at different q_T can be obtained by
`call CalcXsec_DY(X,qt)`
where `X` is real*8 variable where cross-section will be stored, `qt` is real*8 is the list of values of q_T's at which the `X` is to be calculated. There exists an overloaded version with `X` (real*8)(1:N) and `qT`(real*8)(1:N), which evaluates the array of crossections over array of q_T.

Typically, one needs to integrate over bin. There is a whole set of subroutines which evaluate various integrals over bin they are

Subroutine	integral	Comment
`CalcXsec_SIDIS(X,pt)`	single-point cross-section at given p_T	
`CalcXsec_SIDIS_Zint_Xint_Qint` `(X,pt,zMin,zMax,xMin,xMax,Qmin,Qmax)`	$\int_{z_{\min}}^{z_{\max}} dz \int_{x_{\min}}^{x_{\max}} dx \int_{Q_{\min}}^{Q_{\max}} 2QdQ d\sigma(p_T)$	$0 < z_{\min} < z_{\max} < 1$ $0 < x_{\min} < x_{\max} < 1$ $0 < Q_{\min} < Q_{\max}$
`CalcXsec_SIDIS_PTint_Zint_Xint_Qint` `(X,ptMin,ptMax,zMin,zMax,xMin,xMax,Qmin,Qmax)`	$\int_{z_{\min}}^{z_{\max}} dz \int_{x_{\min}}^{x_{\max}} dx \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{pT_{\min}}^{pT_{\max}} 2p_T dp_T d\sigma(p_T)$	$0 < z_{\min} < z_{\max} < 1$ $0 < x_{\min} < x_{\max} < 1$ $0 < Q_{\min} < Q_{\max}$
More to be added		

Note 1: The point $p_T = 0$ is somewhat problematic, since it leads to flat Hankel integral. Thus for $p_T < 1$MeV it is set to 1MeV.
Note 2: Each command has overloaded version with arrays for X and qt.
Note 3: Cross-section integrated over p_T bins have overloaded versions with `X`, `qtmin` and `qtmax` being arrays. Then the integral is done for `X`(i) from `ptmin(i)` to `ptmax(i)`.
Note 4: There exist special overloaded case for integrated over p_T-bins, with successive bins. I.e. for bins that adjust to each other. In this case only one argument `ptlist` is required (instead of `qtmin`,`qtmax`). The integration for `X(i)` is done from `ptlist(i)` till `ptlist(i+1)`.

Take care that every next function is heavier to evaluate then the previous one. The integrations over Q and y are adaptive Simpsons. We have found that it is the fastest (adaptive) method for typical cross-sections with tolerance $10^{-3} - 10^{-4}$. Naturally, it is not suitable for higher precision, which however is not typically required. The integration over pt is not adaptive, since typically p_T-bins are rather smooth and integral is already accurate if approximated by 5-7 points (default value is set in `constants`). For larger p_T-bins we suggest to use overloaded versions with manual set of N (number of integral sections).

There is a subroutine that evaluate cross-section without preliminary call of `SetCuts`,`TMDX_SIDIS_SetProcess`,`TMDX_SIDIS_XSetup`. It is called `xSec_SIDIS` and it have the following interface:

```
xSec_SIDIS(X,process,s,pT,z,x,Q,incC,cuts,masses)
```

where

- `X` is (real*8) value of cross-section.
- `process` is (integer)array `(p1,p2,p3)`.
- `s` is (real*8)Mandelshtam variable s
- `pT` is (real*8)array `(qtmin,qtmax)`
- `z` is (real*8)array `(zmin,zmax)`
- `x` is (real*8)array `(xmin,xmax)`
- `Q` is (real*8)array `(Qmin,Qmax)`
- `incC` is (logical) flag to include cuts.
- `cuts` is (real*8) array `(ymin,ymax,W2min,W2max)` that parameterizes kinematic cuts.
- `masses` is (real*8) array `(mt,mp)` which is $(M_{target}, m_{produced})$ in GeV OPTIONAL.

IMPORTANT: Practically, the call of this function coincides with `X` evaluated by the following set of commands

```
call TMDX_SIDIS_SetProcess(process)
call TMDX_SIDIS_XSetup(s,any,any,any)
call TMDX_SIDIS_SetCuts(incC,ymin,ymax,W2min,W2max)
call CalcXsec_SIDIS_PTint_Zint_Xint_Qint (X,ptMin,ptMax,zMin,zMax,xMin,xMax,Qmin,Qmax)
```

However, there is an important difference: the values of `s, process` which are set by routines `..Set..`, are global. For that reason such approach could not be used in a parallel computation. Contrary, in the function `xSec_SIDIS` these variables are set locally and thus this function can be used in parallel computations.

There is also analogous subroutine that evaluate cross-section by list. It is called `xSec_SIDIS_List` and it have the following interface:

```
xSec_SIDIS_List(X,process,s,pT,z,x,Q,incC,cuts,masses)
```

All variables are analogues to those of `xSec_SIDIS`, but should come in lists, i.e. `X` is (1:n), `process` is (1:n,1:3), `s` is (1:n), `pT`,`x`,`Q`,`z` are (1:n,1:2). This command compiled by OPENMP, so runs in parallel on multi-core computers.

C. Kinematic cuts

Some experiments put extra constraints on the measurement. Typically, such constraint has the form

$$y_{\text{min}} < y < y_{\text{max}}, \qquad W^2_{\text{min}} < W^2 < W^2_{\text{max}}. \tag{19.2}$$

In this case the integration over x and Q^2 has additional restrictions. Say, if one integrates over a bin: $x_{\text{min}} < x < x_{\text{max}}$ and $Q_{\text{min}} < Q < Q_{\text{max}}$ the boundaries of the bin should be modified as (here integration over x is before the integration over Q)

$$\hat{x}_{\text{min}}(Q) < x < \hat{x}_{\text{max}}(Q), \qquad \hat{Q}^2_{\text{min}} < Q^2 < \hat{Q}^2_{\text{max}}. \tag{19.3}$$

where

$$\begin{aligned}
\hat{x}_{\text{min}}(Q) &= \max\{x_{\text{min}}, \frac{Q^2}{y_{\text{max}}(s-M^2)}, \frac{Q^2}{Q^2+W^2_{\text{max}}-M^2}\},\\
\hat{x}_{\text{max}}(Q) &= \min\{x_{\text{max}}, \frac{Q^2}{y_{\text{min}}(s-M^2)}, \frac{Q^2}{Q^2+W^2_{\text{min}}-M^2}\},\\
\hat{Q}^2_{\text{min}} &= \max\{Q^2_{\text{min}}, x_{\text{min}}y_{\text{min}}(s-M^2), \frac{x_{\text{min}}}{1-x_{\text{min}}}(W^2_{\text{min}}-M^2)\},\\
\hat{Q}^2_{\text{max}} &= \min\{Q^2_{\text{max}}, x_{\text{max}}y_{\text{max}}(s-M^2), \frac{x_{\text{max}}}{1-x_{\text{max}}}(W^2_{\text{max}}-M^2)\}.
\end{aligned}$$

If some cut is not required, then set corresponding parameter to limiting value, such as $y_{\text{min}} = 0$, $y_{\text{max}} = 1$, $W^2_{\text{min}} = 0$, $W^2_{\text{max}} \gg Q^2$.

D. Power corrections

There are many sources of power corrections. For a moment there is no systematic studies of of effect of power corrections for TMD factorization. Nonetheless we include some options in `artemide`, and plan to make systematic treatment in the future.

Kinematic corrections to the variables/phase space. There are three sources of kinematic corrections to the variables

$$\frac{p_\perp}{Q}, \qquad \frac{M}{Q}, \qquad \frac{m}{Q},$$

where M is the target mass, m is the mass of the fragmented hadron. These terms appear in many places of the SIDIS expression (see sec.XIX G, for some minimal details), even without accounting for power-suppressed terms in the factorization formula. The nice feature of these correction is that they could be accounted exactly. The majority of these terms appears during of the Lorentz transformation of factorization frame (where the factorization is performed) to the laboratory frame (where the measurement is made).

Exact values for hard scale for SIDIS: The TMD hard coefficient function contains $\ln(|2q^+q^-|/\mu^2)$ which is usually approximated by $\ln(Q^2)$. Similarly, the *zeta*-scales are normalized as $\zeta\bar{\zeta} = (2q^+q^-)^2$, which is usually approximated by Q^4. So, the exact values of scales are

$$\mu = \sqrt{Q^2 - \boldsymbol{q}_T^2}, \qquad \zeta_1 = \zeta_2 = Q^2 - \boldsymbol{q}_T^2. \tag{19.4}$$

Traditionally, the corrections $\sim q_T/Q$ are dropped, since they are power corrections. Nonetheless, they could be included since they have different sources in comparison to power corrections to the factorized cross-section. Usage of one or another version of *scales* is switched in `constants-file` (Option 10.A.p6).

The accounting of these corrections is switched on/off in `constants`.

E. Bin-integration routines

The inclusion of bin-integration is essential for comparison with experiment. The integration over a bin is automatically included in the calculation of cross-section (although routines for unintegrated calculation are also presented). Typically, one does not need an extreme precision from such integration i.e. 4-5 digit precision is typically more then enough to overshoot experimental precision. Also the cross-section is relatively slow-function (exception is Z-boson peak). Therefore, as a based method for bin-integration I have selected the Simpson integration rule. By default, the adaptive Simpson rule is used for integration over z, x, Q^2, and fixed-number-of-points Simpson method for p_T-integration.

If the sizes of bins are small and the cross-section is smooth in this region, then it is worth to consider faster (but less accurate) methods for integration. They are set in constants-file in the section.10:B. The methods are

`SA` Default. Simpson-adaptive algorithm. Runs with relative tolerance specified in constants-file. Initial estimation of the integral is done with 5-points. Minimal number of points for integration is 9.

`S5` Simpson-fixed-number-of-points algorithm. Estimation of the integral is done with 5-points. No checks for convergence.

`IO` No-integration. Evaluates the central point of the bin and multiply by the bin size (not recommended).

Note, the selection of method for integration is available only for some integration (so far integration over x and z). Other methods and integration will be added in future.

F. Enumeration of processes

List of enumeration for $prefactor1$

`p1`

`p1`	$prefactor1$	Short description
1	1	No comments.
2	$\frac{Q^2}{y}$	The cross-section $\frac{d\sigma}{dxdydzd(p_\perp^2)}$, in this case integration over Q^2 is replaced by integration over y.
3	$\frac{x}{y}$	The cross-section $\frac{d\sigma}{dydQ^2dzd(p_\perp^2)}$, in this case integration over x is replaced by integration over y.

List of enumeration for $prefactor2$

`p2`

`p2`	$prefactor2$	Short description
1	$\frac{2\pi\alpha_{\text{em}}^2(Q)}{Q^4}\frac{y^2}{1-\varepsilon}\lvert C_V^{SIDIS}(c_2Q)\rvert^2 c_0^{(un)}R$	(unpol.) SIDIS-cross-section $\frac{d\sigma}{dxdQ^2dzd(p_\perp^2)}$. [pb/GeV^2].
2	$\frac{x}{\pi}\frac{c_0^{(un)}}{\left(1+\frac{\gamma^2}{2x}\right)}\lvert C_V^{SIDIS}(c_2Q)\rvert^2$	Prefactor for $F_{UU,T}$ according to the tabulated definition (2.7) of [12].

Here $R = 0.3893379 \times 10^9$ the transformation factor from GeV to pb.

$$c_0^{(un)} = \frac{1 + \left(\varepsilon - \frac{\gamma^2}{2}\right)\frac{\boldsymbol{\rho}_\perp^2 - \rho^2}{1-\gamma^2\rho^2}}{\sqrt{1-\gamma^2\boldsymbol{\rho}_\perp^2}}.$$

G. SIDIS theory

In many aspects the theory for SIDIS is more complicated then for Drell-Yan. Here I collect the main definition which were used in the artemide. For a more detailed description see ref.[12]. Note, that some parts of definition were derived by me, since I have not found them in the literature.

1. Kinematics

The process is

$$H(P) + l(l) \to l(l') + h(p_h) + X, \tag{19.5}$$

with

$$P^2 = M^2, l^2 = l'^2 = m_l^2 \simeq 0, \qquad p_h^2 = m^2. \tag{19.6}$$

The standard variables used for the description of SIDIS are

$$q = l - l' \qquad \Rightarrow \qquad Q^2 = -q^2, \qquad x = \frac{Q^2}{2(Pq)}, \qquad y = \frac{(Pq)}{(Pl)}, \qquad z = \frac{(Pp_h)}{(Pq)}. \tag{19.7}$$

There are is also a set of power variables used to define power-corrections

$$\gamma = \frac{2Mx}{Q}, \qquad \rho = \frac{m}{zQ}, \qquad \boldsymbol{\rho}_\perp^2 = \frac{m^2 + \boldsymbol{p}_\perp^2}{z^2Q^2}. \tag{19.8}$$

The variables x, y and Q^2 are dependent: $xy(s - M^2) = Q^2$. The phase-volume $dxdQ^2$ can be expressed via $dxdy$ or $dydQ^2$:

$$dxdQ^2 = \frac{Q^2}{y}dxdy = \frac{x}{y}dydQ^2. \tag{19.9}$$

A phase space point is totally characterized by the following numbers:

$$\{s, x, Q^2, z, \boldsymbol{p}_T^2\}. \tag{19.10}$$

Alternatively, one can replace x or Q^2 by y.

2. Cross-section

The cross-section for SIDIS has the general form

$$\frac{d\sigma}{dxdQ^2dzd\psi d\phi_h d\boldsymbol{p}_{h\perp}^2} = \frac{\alpha_{\text{em}}^2(Q)}{Q^6}\frac{y^2}{8z}\frac{L_{\mu\nu}W^{\mu\nu}}{\sqrt{1+\boldsymbol{\rho}_\perp^2\gamma^2}}, \tag{19.11}$$

where $L_{\mu\nu}$ is the lepton tensor, and $W_{\mu\nu}$ is the hadron tensor. The hadron tensor contains many term accompanied by polarization angles.

The TMD factorization is performed in the factorization frame with respect to $\boldsymbol{q}_T^2 \ll Q^2$. In the factorization frame, the unpolarized part of the hadron tensor

$$W^{\mu\nu} = \frac{-zg_T^{\mu\nu}}{\pi}\int |\boldsymbol{b}|d|\boldsymbol{b}|J_0(|\boldsymbol{b}||\boldsymbol{q}_T|)\sum_f e_f^2|C_V(-Q^2,\mu^2)|^2F_1^f(x_1,\boldsymbol{b})D_1^f(z_1,\boldsymbol{b}) + \dots\ , \tag{19.12}$$

where dots denote, polarized terms, and power corrections. The variables x_1 and z_1 are

$$x_1 = -\frac{2x}{\gamma^2}\left(1-\sqrt{1+\gamma^2\left(1-\frac{\boldsymbol{q}_T^2}{Q^2}\right)}\right), \tag{19.13}$$

$$z_1 = -z\frac{1-\sqrt{1+\left(1-\frac{\boldsymbol{q}_T^2}{Q^2}\right)\gamma^2}}{\gamma^2}\frac{1+\sqrt{1-\rho^2\gamma^2}}{1-\frac{\boldsymbol{q}_T^2}{Q^2}}. \tag{19.14}$$

The factorization variable $|\boldsymbol{q}_T|$ is related to the measured variable $\boldsymbol{p}_{h\perp}$ as

$$|\boldsymbol{q}_T| = \frac{|\boldsymbol{p}_{h\perp}|}{z}\sqrt{\frac{1+\gamma^2}{1-\gamma^2\rho^2}}, \qquad \Leftrightarrow \qquad |\boldsymbol{p}_{h\perp}| = z|\boldsymbol{q}_T|\sqrt{\frac{1-\gamma^2\rho^2}{1+\gamma^2}}. \tag{19.15}$$

Making the convolution with unpolarized leptonic tensor, we get the following expression for the cross-section

$$\begin{aligned}\frac{d\sigma}{dxdQ^2dzd\boldsymbol{p}_{h\perp}^2} &= 2\pi\frac{\alpha_{\text{em}}^2}{Q^4}\frac{1}{\sqrt{1+\boldsymbol{\rho}_\perp^2\gamma^2}}\frac{y^2}{2(1-\varepsilon)}\left\{1+\left(\varepsilon-\frac{\gamma^2}{2}\right)\frac{\boldsymbol{\rho}_\perp^2-\rho^2}{1-\gamma^2\rho^2}\right\} \\ &\int|\boldsymbol{b}|d|\boldsymbol{b}|J_0\left(\frac{|\boldsymbol{b}||\boldsymbol{p}_{h\perp}|}{z}\sqrt{\frac{1+\gamma^2}{1-\gamma^2\rho^2}}\right)\sum_f e_f^2|C_V(-Q^2,\mu^2)|^2F_1^f(x_1,\boldsymbol{b})D_1^f(z_1,\boldsymbol{b}),\end{aligned} \tag{19.16}$$

where

$$\varepsilon = \frac{1-y-\frac{\gamma^2y^2}{4}}{1-y+\frac{y^2}{2}+\frac{y^2\gamma^2}{4}}. \tag{19.17}$$

XX. TMDS_INKT MODULE

The module `TMDs_inKT` is derivative of the module `TMDs` that provides the TMD in the transverse momentum space. We define

$$F(x,\boldsymbol{k}_T;\mu,\zeta)=\int\frac{d^2\boldsymbol{b}}{(2\pi)^2}F(x,\boldsymbol{b};\mu,\zeta)e^{-i(\boldsymbol{k}_T\boldsymbol{b})}. \tag{20.1}$$

Since all evaluated TMDs depends only on the modulus of $\boldsymbol{b}$, within the module we evaluate Hankel transformations. These transformations depends on the kind of TMD. Generally the integrals are

$$F(x,|\boldsymbol{k}_T|;\mu,\zeta)=\int\frac{d|\boldsymbol{b}|}{2\pi}|\boldsymbol{b}|^{n+1}J_n(|\boldsymbol{b}||\boldsymbol{k}_T|)F(x,|\boldsymbol{b}|;\mu,\zeta). \tag{20.2}$$

On top of it there is could be extra prefactors specified in the table. For simplicity these cases indecated by numbers

$$n=0,\qquad F(x,|\boldsymbol{k}_T|;\mu,\zeta)=\int\frac{d|\boldsymbol{b}|}{2\pi}|\boldsymbol{b}|J_0(|\boldsymbol{b}||\boldsymbol{k}_T|)F(x,|\boldsymbol{b}|;\mu,\zeta), \tag{20.3}$$

$$n=1,\qquad F(x,|\boldsymbol{k}_T|;\mu,\zeta)=\int\frac{d|\boldsymbol{b}|}{2\pi}|\boldsymbol{b}|^2J_1(|\boldsymbol{b}||\boldsymbol{k}_T|)F(x,|\boldsymbol{b}|;\mu,\zeta), \tag{20.4}$$

$$n=2,\qquad F(x,|\boldsymbol{k}_T|;\mu,\zeta)=-\int\frac{d|\boldsymbol{b}|}{2\pi}|\boldsymbol{b}|^3J_2(|\boldsymbol{b}||\boldsymbol{k}_T|)F(x,|\boldsymbol{b}|;\mu,\zeta). \tag{20.5}$$

The module `TMDs_inKT` uses all functions from the module `TMDs`. In fact, all technical commands (like `Initialize`) just transfer the request to `TMDs`. Thus, the information on these commands can be found in the section XVI. For the evaluation of Hankel integral we use the Ogata quadrature see XVII B (the parameters for it are set in corresponding section of `constants`.

List of available commands

Command	Type	Sec.	Short description
`TMDs_inKT_Initialize(order)`	subrout.	XVI	Initialization of module.
`TMDs_inKT_ResetCounters()`	subrout.	-	Reset intrinsic counters of modules
`TMDs_inKT_ShowStatistic`	subrout.	-	Print current statistic on the number of calls.
`testTMDPDF_kT(x,kT)`	(real*8(-5:5))	-	Evaluates the Fourier of the text function (see below). $n = 0$
`uTMDPDF_kT_5(x,kT,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD PDF at the optimal line (gluon term undefined). $n = 0$
`uTMDPDF_kT_50(x,kT,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD PDF at the optimal line (gluon term defined). $n = 0$
`uTMDPDF_kT_5(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD PDF (gluon term undefined). $n = 0$
`uTMDPDF_kT_50(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD PDF (gluon term defined). $n = 0$
`uTMDFF_kT_5(x,kT,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD FF at the optimal line (gluon term undefined). $n = 0$
`uTMDFF_kT_50(x,kT,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD FF at the optimal line (gluon term defined). $n = 0$
`uTMDFF_kT_5(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD FF (gluon term undefined). $n = 0$
`uTMDFF_kT_50(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Unpolarized TMD FF (gluon term defined). $n = 0$
`lpTMDPDF_kT_50(x,kT,h)`	(real*8(-5:5))	XVI B	Linearly polarized gluon TMD PDF at the optimal line (quark terms undefined). $n = 2$
`lpTMDPDF_kT_50(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Linearly polarized gluon TMD PDF (quark terms undefined). $n = 2$
`SiversTMDPDF_kT_5(x,kT,h)`	(real*8(-5:5))	XVI B	Sivers TMD PDF at the optimal line (gluon term undefined). $n = 1$
`SiversTMDPDF_kT_50(x,kT,h)`	(real*8(-5:5))	XVI B	Sivers TMD PDF at the optimal line (gluon term defined). $n = 1$
`SiversTMDPDF_kT_5(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Sivers TMD PDF (gluon term undefined). $n = 1$
`SiversTMDPDF_kT_50(x,kT,mu,zeta,h)`	(real*8(-5:5))	XVI B	Sivers TMD PDF (gluon term defined). $n = 1$

Comments:

- The value of k_T is expected bigger 1MeV for smaller values the function evaluates at 1MeV.
- For gluonless TMDs (_5) the gluon term is identically 0.
- The convergence of the integral is checked by convergence of each flavor (until all flavors are convergent).

The function `testTMDPDF_kT(x,kT)` returns the Fourier of the following test functions (x is the free parameter

$$\{b^3e^{-xb}, b^2e^{-xb}, be^{-xb}, e^{-xb}, \frac{1}{(b^2+x^2)^4}, \frac{1}{(b^2+x^2)^2}, \frac{1}{b^2+x^2}, b^2e^{-xb^2}, e^{-xb^2}, \frac{J_0(x/b)}{b}, \frac{1}{b}\}. \tag{20.6}$$

The result of evaluation should be

$$\begin{aligned} \Bigg\{ \quad & \frac{3}{2\pi}\frac{3k_T^4-24k_T^2x^2+8x^4}{(k_T^2+x^2)^{9/2}}, \frac{3x}{2\pi}\frac{2x^2-3k_T^2}{(k_T^2+x^2)^{7/2}}, \frac{1}{2\pi}\frac{2x^2-k_T^2}{(k_T^2+x^2)^{5/2}}, \frac{1}{2\pi}\frac{x}{(k_T^2+x^2)^{3/2}}, \\ & \frac{k_T^3}{96\pi x^3}K_3\left(k_Tx\right), \frac{k_T}{4\pi x}K_1\left(k_Tx\right), \frac{K_0\left(k_Tx\right)}{2\pi}, \\ & \frac{4x-k_T^2}{16\pi x^3}e^{-k_T^2/(4x)}, \frac{e^{-k_T^2}/(4x)}{4\pi x}, \frac{J_0(2\sqrt{xk_T})}{2\pi k_T}, \frac{1}{2\pi k_T}\Bigg\}. \end{aligned} \tag{20.7}$$

The expressions for functions are selected such that the integral gives reasonable values for $0.05 < x < 1$.

XXI. SUPPLEMENTARY CODES

In the `artemide` archive you can find some example codes. They are located in `Prog/` and can be compiled by `make program TARGET=name`

`test.f90`	Makes some elementary computation. Can be run by `make test`
`DY_example_v14.f90`	Commented example of the code for computation of DY cross-section for `artemide` ver.1.4
`DY_example_v2.f90`	Commented example of the code for computation of DY cross-section for `artemide` ver.2.00.
`DY_for_RHIC.f90`	Example computation of Z-boson production in RHIC kinematics with error-estimation by replicas.
`DY_for_CMS.f90`	Example computation of Z-boson production in CMS kinematics with error-estimation by scale-variations.

XXII. HARPY

The `harpy` (from combination of Hybrid ARtemide+PYthon)) is an interface to `artemide` to the python.

It is directly possible to interfacing the `artemide` to python since `artemide` is made on fortran95. It uses some of it features, such as interfaces, and indirect list declarations, which are alien to python. Also I have not found any convenient way to include several dependent Fortran modules in f2py (if you have suggestion just tell me). Therefore, I made a wrap module `harpy.f90` that call some useful functions from `artemide` with simple declarations. So, it could be linked to python by f2py library.

So, in the current realization I create the signature file that declare python module artemide, which has an wrap module harpy. In python it looks like

>>> import artemide

>>> artemide.harpy.initialize("NNLO")

>>> print artemide.harpy.utmdpdf_5_optimal(0.1,1.,1)

Ugly, but it works.

Note, that not all functions of `artemide` are available in `harpy`. I have added only the most useful, however, you can add them by our-self, or write me an e-mail. Here, list the functions from `artemide` and their synonym in `harpy`

artemide		harpy.f90
module	function	
`aTMDe_control`	`artemide_Initialize(file)`	`Initialize(file)`
	`artemide_ShowStatistics()`	`ShowStatistics()`
	`artemide_SetScaleVariations(c1,c2,c3,c4)`	`SetScaleVariations(c1,c2,c3,c4)`
	`artemide_SetNPparameters(lambda)`	`SetLambda_Main(lambda)`
	`artemide_SetNPparameters_TMDR(lambda)`	`SetLambda_TMDR(lambda)`
	`artemide_SetNPparameters_uTMDPDF(lambda)`	`SetLambda_uTMDPDF(lambda)`
	`artemide_SetNPparameters_uTMDFF(lambda)`	`SetLambda_uTMDFF(lambda)`
	`artemide_SetReplica_TMDR(lambda)`	`SetReplica_TMDR(lambda)`
	`artemide_SetReplica_uTMDPDF(lambda)`	`SetReplica_uTMDPDF(lambda)`
	`artemide_SetReplica_uTMDFF(lambda)`	`SetReplica_uTMDFF(lambda)`
`TMDX_DY`	`TMDX_DY_SetNPParameters`(array)	`SetLambda`(array)
	`TMDX_DY_SetNPParameters`(integer)	`SetLambda_ByReplica`(integer)
	`TMDX_DY_SetScaleVariations(c1,c2,c3,c4)`	`SetScaleVariation(c1,c2,c3,c4)`
`TMDs`	`uTMDPDF_5(x,bt,muf,zetaf,h)`	`uTMDPDF_5_Evolved(x,bt,muf,zetaf,h)`
	`uTMDPDF_50(x,bt,muf,zetaf,h)`	`uTMDPDF_50_Evolved(x,bt,muf,zetaf,h)`

	uTMDPDF_5(x,bt,h)	uTMDPDF_5_Optimal(x,bt,h)
	uTMDPDF_50(x,bt,h)	uTMDPDF_50_Optimal(x,bt,h)
	TMDs_SetPDFreplica(n)	SetPDFreplica(n)
	Same with uTMDFF	
	Same with lpTMDPDF	(only _50 version)
	Same with SiversTMDPDF	
TMDs_inKT	uTMDPDF_kT_5(x,kt,muf,zetaf,h)	uTMDPDF_kT_5_Evolved(x,kt,muf,zetaf,h)
	uTMDPDF_kT_50(x,kt,muf,zetaf,h)	uTMDPDF_kT_50_Evolved(x,kt,muf,zetaf,h)
	uTMDPDF_kT_5(x,kt,h)	uTMDPDF_kT_5_Optimal(x,kt,h)
	uTMDPDF_kT_50(x,kt,h)	uTMDPDF_kT_50_Optimal(x,kt,h)
	Same with uTMDFF	
	Same with lpTMDPDF	(only _50 version)
	Same with SiversTMDPDF	
TMDX_DY	xSec_DY(X,proc,s,qT,Q,y,iC,cuts)	X=DY_xSec_Single(proc,s,qT,Q,y,iC,cuts)
	xSec_DY_List(X,proc,s,qT,Q,y,iC,cuts)	X=DY_xSec_List(proc,s,qT,Q,y,iC,cuts,L)
TMDX_SIDIS	xSec_SIDIS(X,proc,s,pT,z,x,Q,iC,cuts)	X=SIDIS_xSec_Single(proc,s,pT,z,x,Q,iC,cuts)
	xSec_SIDIS(X,proc,s,pT,z,x,Q,iC,cuts,masses)	X=SIDIS_xSec_Single_withMasses(proc,s,pT,z,x,Q,i
	xSec_SIDIS_List(X,proc,s,pT,z,x,Q,iC,cuts)	X=SIDIS_xSec_List(proc,s,qT,z,x,Q,iC,cuts,L)
	xSec_SIDIS_List(X,proc,s,pT,z,x,Q,iC,cuts,masses)	X=SIDIS_xSec_List_withMasses(proc,s,qT,z,x,Q,iC

`L` is the length of the input/output lists.

NOTE: there is no **OPTIONAL** parameters. All parameters should be defined (it is necessary for f2py).

For convenience I have also created a more user-friendly interface, written on python. It is called `harpy.py` and located in `/harpy`, and can be imported as is.

[1] I. Scimemi and A. Vladimirov, Eur. Phys. J. C **78**, no. 2, 89 (2018) doi:10.1140/epjc/s10052-018-5557-y [arXiv:1706.01473 [hep-ph]].
[2] J. Collins, Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol. **32** (2011) 1.
[3] L. A. Harland-Lang, A. D. Martin, P. Motylinski and R. S. Thorne, Eur. Phys. J. C **75** (2015) no.5, 204 doi:10.1140/epjc/s10052-015-3397-6 [arXiv:1412.3989 [hep-ph]].
[4] I. Scimemi and A. Vladimirov, JHEP **1808** (2018) 003 doi:10.1007/JHEP08(2018)003 [arXiv:1803.11089 [hep-ph]].
[5] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht, M. Schönherr and G. Watt, Eur. Phys. J. C **75**, 132 (2015) doi:10.1140/epjc/s10052-015-3318-8 [arXiv:1412.7420 [hep-ph]].
[6] M. G. Echevarria, I. Scimemi and A. Vladimirov, JHEP **1609**, 004 (2016) doi:10.1007/JHEP09(2016)004 [arXiv:1604.07869 [hep-ph]].
[7] M. G. Echevarria, I. Scimemi and A. Vladimirov, Phys. Rev. D **93**, no. 1, 011502 (2016) Erratum: [Phys. Rev. D **94**, no. 9, 099904 (2016)] doi:10.1103/PhysRevD.93.011502, 10.1103/PhysRevD.94.099904 [arXiv:1509.06392 [hep-ph]].
[8] M. G. Echevarria, I. Scimemi and A. Vladimirov, Phys. Rev. D **93**, no. 5, 054004 (2016) doi:10.1103/PhysRevD.93.054004 [arXiv:1511.05590 [hep-ph]].
[9] A. A. Vladimirov, Phys. Rev. Lett. **118**, no. 6, 062001 (2017) doi:10.1103/PhysRevLett.118.062001 [arXiv:1610.05791 [hep-ph]].
[10] A. Vladimirov, JHEP **1804**, 045 (2018) doi:10.1007/JHEP04(2018)045 [arXiv:1707.07606 [hep-ph]].
[11] T. Gehrmann, E. W. N. Glover, T. Huber, N. Ikizlerli and C. Studerus, JHEP **1006**, 094 (2010) doi:10.1007/JHEP06(2010)094 [arXiv:1004.3653 [hep-ph]].
[12] A. Bacchetta, M. Diehl, K. Goeke, A. Metz, P. J. Mulders and M. Schlegel, JHEP **0702**, 093 (2007) doi:10.1088/1126-6708/2007/02/093 [hep-ph/0611265].
[13] V. Ahrens, T. Becher, M. Neubert and L. L. Yang, Eur. Phys. J. C **62**, 333 (2009) doi:10.1140/epjc/s10052-009-1030-2 [arXiv:0809.4283 [hep-ph]].
[14] V. Ahrens, T. Becher, M. Neubert and L. L. Yang, Phys. Rev. D **79** (2009) 033013 doi:10.1103/PhysRevD.79.033013 [arXiv:0808.3008 [hep-ph]].
[15] D. Gutierrez-Reyes, S. Leal-Gomez, I. Scimemi and A. Vladimirov, JHEP **11** (2019), 121 doi:10.1007/JHEP11(2019)121 [arXiv:1907.03780 [hep-ph]].
[16] I. Scimemi, A. Tarasov and A. Vladimirov, JHEP **05** (2019), 125 doi:10.1007/JHEP05(2019)125 [arXiv:1901.04519 [hep-ph]].

XXIII. VERSION HISTORY

Ver.2.06

- **common:tw2Convolution** Fixed bug, with incorrect estimation of the integral reminder at $x_0 \to 1$
- **uTMDFF:coeffFunc** 3-loop coefficient function added.
- **uTMDPDF:coeffFunc** 3-loop coefficient function added.
- **uTMDFF:coeffFunc** Corrected missprint in the NNLO coeff.function ($\sim \pi^4\delta(\bar{z})$ term)
- **all .TMD.** Singular list updated to include $(\ln^3 \bar{x}/(1-x))_+$ terms [appear at N^{3}LO]
- **TMD_SIDIS** 4-loop coefficient function added.
- **TMD_DY** 4-loop coefficient function added. Fixed a misprint in 3-loops (¡0.01% numerically)
- **TMD_AD:AD_Integral**, implemented `RADEvolution` function, and accompanying expressions (roots, Lagrange coefficients).
- **TMDR**, Fixed mistake with counting of orders (order of Γ_{cusp} was higher by 1).
- **TMD_AD:AD_atMu**, minor optimizations.
- **TMD_AD:AD_secondary**, added 4-loop expressions for RAD, and zeta-lines.
- **TMD_AD:AD_primary**, added 4-loop expressions for RAD, and anomalous dimensions, 5-loop Γ_{cusp}. The values are taken from [2202.04660, 2205.02249, 1812.11818]
- **TMDF**, Added processes: 13001-13059, 13101-13106, 13111-13116 (for A_{LT})
- **TMDs**, **TMDs_inKT**, **harpy** Added **wgtTMDPDF** routines.
- **aTMDe_control** Added **wgtTMDPDF** routines.
- **aTMDe_setup** Fixed bug with absent list of hadrons for Sivers.
- **aTMDe_setup** Modifications due to **QCDinput** and **wgtTMDPDF** updates.
- **wgtTMDPDF** Implemented.
- **EWinput**: Fixed bug with the S-quark parameter for the interference term in Z-boson production (Thanks to S. Leal-Gomez).
- **QCDinput**: Helicity PDF input is added.
- **QCDinput**: Bug in the enumeration of hadrons fixed.
- **TMDX_SIDIS**: Added option for exact factorization scales.
- **TMDX_DY**: Added option for exact factorization scales.

Ver.2.05

- **TMDX_SIDIS**: Minor optimization.
- **TMDF** Added the global mass scale used for normalization of TMD structure functions.
- **TMDF** Processes unpolarized DY, 101-104, are changed to $h+p$ (was $p+h$).
- **TMDF** Added processes for Sivers 10001,10005,10007,10008,10101-10104,12001–12059,12105,12106,12115,12116,
- **TMDF** Added processes for unpolarized SIDIS 2105,2106,2115,2116,2041-2049.
- **harpy** Update to python3.
- **harpy** Added functions for values of uTMDFF, lpTMDPDF, SiversTMDPDF, and DNP, and for TMD in kT space.
- **harpy** Added functions `Set_NPparameters` for **lpTMDPDF** and **SiversTMDPDF**.
- **IO_functions** Added a check for errors in during the file-reading.
- **aTMDe_control** Added version check for replica files
- **aTMDe_control** Some code refactoring + **SiversTMDPDF** (fixed couple of bugs in lpTMD-part)
- **aTMDe_setup** Some code refactoring + **SiversTMDPDF**
- **TMDs** Some code refactoring + **SiversTMDPDF**
- **SiversTMDPDF** Implemented.
- ***TMD**** Added `NA`-option for the order-definition.

- ***TMD**** Grid-code is moved to a separate directory.

Ver.2.04

- ***TMD****&`*TMD**_model`: Deep refactoring of the code. In particular
  * `*TMD**_model` turned to separate modules.
  * `*TMD**_modelInterface` are removed. Their functionality is totally inside the model-module.
  * The code is distributed into separate text-files.
  * Few optimization twicks.
- **TMDR**&`TMD_AD`: Deep refactoring of the code. In particular
  * `TMDR_model` turned to a separate module.
  * The definitions of $\mathcal{D}$ and ζ are moved to `TMD_AD`-module
  * Removed evolution type-4, since it coincides with type-3 at $\zeta = \zeta_{NP}$.
  * Removed function `TMDR R toSL`, since it coincides with `TMDR Rzeta` at $\zeta = \zeta_{NP}$
  * The code is distributed into separate text-files.
  * Few optimization twicks.
- **aTMDe_control**: Fixed bug in the check of length for λ_{NP} (Thanks to M.Echevarria).
- **TMDs_inKT**: The integration routine is updated. Changed the convergence check and added test cases.
- **TMDX_SIDIS**: Added integration method `'IO'` for x and z integration.
- **TMDX_SIDIS**: Fixed bug in the evaluation of cross-section without bin integration and without cuts

Ver.2.03

- **TMDR**: Added `orderZETA=`-1, which is used in the LO case.
- **TMDR**: `zeMUresum` temporary removed.
- **QEDinput** Changed routine for evaluation of $\alpha_{\rm QED}$. Previous had incorrect determination of boundary value.
- **TMDF**: Added extra test for null-value integrals in OGATA quadrature.
- **TMDF**: OGATA quadrature is updated, with different values of h for qT-intervals
- **TMDF**: Added test processes N+9999, N+9998
- All: Update of variables definition. Several “magic numbers” removed. Fixed several possible precision leaks.
- **TMDR**: Functions for derived anomalous dimensions modified. No precalculated numbers, all precalculation is explicit and done inside TMD_AD.
- **TMDR** and **TMDF**: Fixed minor leaks of precision, due to rounding errors in real to double real conversions.
- **TMDR→TMD_AD**: definition of anomalous dimension migrated. Definitions redone in explicit way. Fixed bug in Γ_3.
- **TMD_AD**: Implemented. This module contains definition of anomalous dimensions used in TMDR.
- **TMD_numerics**: Implemented. This module contains definition of precision, and definition of various constants.
- **All**: Some refactoring of warning generation.
- **All TMD-modules and models**: Composite TMD option is added.

Ver.2.02

- **Common**: Added utility module `IO_functions`, that contains common function for IO-control. Added added colored output.
- **All TMD-modules**: The initialization array is added to the initialization file. Now, after the initialization each module known the initial NP array.
- **aTMDe_control**: The initialization procedure read initial values of NP arrays, and (after the initialization procedure) set them.
- **aTMDe_control**: Added commands `artemide_GetReplicaFromFile` and `artemide_NumOfReplicasInFile`. The module commands `setReplica` are not used anymore.
- **aTMDe_setup**: Added command `CheckConstantsFile`.
- **TMDF**: Added a check for input values of $x, z, ...$ Now, for $x > 1$ the cross-section is evaluated to zero.

- **TMDF**: Added processes 2011-2039;2101-2104;2111-2114
- **TMDX_DY**: Added additional checks for TMDF_ConvergenceIsLost trigger. Now, each cross-section-evaluation routine returns 10^9 if convergence is lost.
- **TMDX_SIDIS**: Added α_s^3-term for hard-matching coefficient. NNNLO defined.
- **TMDX_SIDIS**: Updated fiducial cut function. Now, it includes $W^2_{\max}$ and needs 4-arguments.
- **TMDX_SIDIS**: Added options for accounting of q_T-correction in x_1 and z_1 irrespectively kinematic q_T-corrections.
- **TMDX_SIDIS**: Added options for selection of integration method for X- and Z- bins (SA and S5).
- **TMDX_SIDIS**: Fixed an error is sign for produced mass-correction. Added missed factor (z_1/z).
- **TMDX_SIDIS**: Fixed rare bug in determination of cuts.
- **TMDX_SIDIS**: Fixed a bug in the initialization routine (wrong line numbering for const-file.)

Ver.2.01

- **All modules:** Added a check for current input-file version. + Small corrections in massages, and bug reports.
- **TMDX_DY**: Added coefficient functions for Higgs-boson production, and corresponding process 5.
- **TMDX_DY**: Added option for coefficient function with π^2-resummation [14].
- **TMDX_DY**: Added a check: if y$\notin[-y_0,y_0]$ the $\sigma=0$.
- **lpTMDPDF**: Implemented.
- **TMDR**: Added expressions for gluon evolution.
- **TMDR**: Fixed numeric error in N^{3}LO resummed expression for $\mathcal{D}$ and NNLO+ expression for resummed ζ_μ.
- **TMDR&TMDs**: added evolution type 4. (Exact solution).
- **aTMDe_control**: fixed a bug with change of NP-parameters for individual modules
- **QCDinput**: fixed a bug with resetting of the PDF-grid
- **TMDF**: Added processes 101-104,1004, 20-22.

Ver.2.00

- **TMDX_DY**: fixed a memory leak in adaptive integration. It could cause to possible source of Segmentation fault error.
- **TMDX_DY**: Added cut for very-small p_T.
- **uTMDPDF&uTMDFF**: Changed priority of grid-calculation during scale-variation.
- **TMD-models&Twist2Convolution**: Added b^* parameter. **Model files are not compatible with earlier versions.**
- **Twist2Grid**: The routine for large-b is changed. Now it is faster and more accurate.
- **aTMDe_control&aTMDe_setup**: Implemented.
- **ALL MODULES**: Total change of initialization routines, and interface subroutines.

Ver.1.41

- **TMDR&TMDs&TMDF**: fixed issues that arise with some older Fortran compilers (thanks to Wen-Chen Chang).
- **TMDX_SIDIS**: Totally rewritten: processes redefined, interface redefined, integration routines rewritten, masses correction, parallelization, etc.
- **TMDX_DY**: Added extra checks.
- **uTMDFF**: The factor z^2 added as an external, see sec.XI G. It should improve the convergence of "common"-convolution at small z.
- **TMDF**: Added Ogata tables for $\tilde{h}=h0.05$. They are used for integrations at smaller q_T.
- **TMDF**: Fixed potential bug in the initialization order.
- **TMDF**& higher: Fixed misprint in the name of function `TMDF_F`.
- **TMDF**: Added processes 2002-2009.
- **TMDF&TMDX_DY**: Added processes [?,4,2013-2018]. W-boson in the narrow width approximation.

- **TMDs**: Fixed error with the passing to NO parameters in the case of multiple distributions.

Ver.1.4
- **constants**: The format of EW input is changed. **Not compatible with older constants-file.**
- **TMDF**: Added processes 7-12. Fixed a mistake in processes 1,2,2001 (thanks to L.Zoppi & D.Gutierrez-Reyes)
- **TMDs**, **uTMDPDF** & **QCDinput**: Added `_SetPDFreplica` routine.
- **HARPY**: Implemented.
- **TMDs and sub-modules**: Added function `SetReplica`.
- **TMDs**: Added interface to optimal TMDs
- **TMDs**: Added check for length of incoming λ_{NP}
- **TMDs**: Fixed bug with incorrect gluon TMDs in functions `_50`.
- **TMDs_inKT**: Implemented.
- **TMDX_DY**: Added `xSec_DY` subroutines.
- **TMDX_DY**: Encapsulated process, and cut-parameter variables.
- **TMDX_DY**: Defined `p1`=2, which corresponds to integration over x_F. Removed old functions for x_F integrations.
- **TMDX_DY**: Fixed a bug with variation of c_2 (introduced in ver.1.3).
- **TMDX_DY**: Fixed a (potential) bug with y-symmetric processes.
- **LeptonCutsDY** : Old version of function removed, cut-parameters encapsulated into single array variable.
- **LeptonCutsDY** : asymmetric cuts in p_T are introduced.
- **LeptonCutsDY** : New function `CutFactor4`, which is analogous to `CutFactor3` but with one integral integrated analytically. Thus, it is more accurate, and faster by 5-20%
- **LeptonCutsDY** : some rearrangement of variables that makes `CutFactor4` and `CutFactor3` faster by 20%.
- **uTMDPDF** & **uTMDFF** : F_{NP} is now function of (x, z, b, h, λ). For that reason **this version incompatible with earlier versions.**
- **uTMDPDF** & **uTMDFF**: The common block of the code is extracted into a separate files. It include calculation of Mellin convolution and Grid construction.

Ver.1.32
- **TMDX_DY**: Added the routine with lists of y-bins, in addition to the lists of pt-bins.
- **TMDX_DY**: The implementation of parallel computation over the list of cross-sections.
- **LeptonCutsDY**: The kinematic variables are encapsulated.
- **TMDX_DY**: The kinematic variables are encapsulated (by the cost of small reduction of performance).
- **uTMDR**: Changed behavior at extremely small-b. Now values of b freeze at $b = 10^{-6}$.
- **uTMDPDF** & **uTMDFF**: Changed behavior at extremely small-b. Now values of b freeze at $b = 10^{-6}$.
- **TMDR**: Added NNNLO evolution (only for quarks, Γ_3 is from 1808.08981). Not tested.
- **TMDs**: Functions `RuPDFuPDF` and `antiRuPDFuPDF` are added.

Ver.1.31
- **Global**: The module `TMDF` is split out from `TMDX_..` modules.
- **Global**: Constant tables are moved to the folder \tables.
- **TMDX_..**: Change the structure of process definition.
- **TMDF**: Fixed bug with throwing exception for failed check of convergence of Ogata quadrature.
- **TMDF**: Added possibility to vary the Ogata quadrature parameters.
- **TMDX_DY**: The structure of interface to integrated cross-section simplified.
- **TMDX_DY**: Added trigger for exact power-corrected values of $x_{1,2}$.
- **uTMDPDF** & **uTMDFF**: Fixed rare error for exceptional restoration of TMD distribution from grid, then f_{NP} evaluated to zero.

Ver.1.3
- **Global**: Complete change of interface. Interface update for all modules.

- **uTMDPDF**: Added hadron dependence. `FNP` is now flavour and hadron dependent.
- **uTMDPDF**: Renormalon correction is removed. As not used.
- **TMDR**: The grid (and pre-grid) option is removed. Since it was incompatible with new interface. Also the new evolution (type 3) is faster any previous (with grids).

Ver.1.2(unpub.)
- `TMDR`: Older version is changed to `uTMDR1`. New evolution routine implemented.
- `uTMDFF`: Implemented.
- `uTMDPDF`: Fixed bug in evaluation of gluon TMDs, within the evaluation of $(..)_+$ part.
- **Global**: Removed functions for the evaluation of only 3-flavours TMDs. As outdated and not used.
- **Global**: Number of non-pertrubative parameters is now read from 'constants'-file. Module `TMDs` initialize sub-modules with accordance to this set.
- **Global**: Module `TMDX` is renamed into `TMDX_DY`, also many functions in it renamed.
- `uTMDX_SIDIS`: Implemented.
- `TMDs`: As an temporary solution introduced a rigid cut for TMD($\mu < m_q$).
- `TMDX`: Update of Ogata quadrature, with more accurate estimation of convergence.

Ver.1.1 hotfix Bugs in `uTMDPDF` and TMDR related to the evaluation of gluon TMDs fixed (thanks to Valerio Bertone).

Ver.1.1
- **Global:** The physical, numerical and option constant are moved to the file `constants`, where they are read during the initialization stage.
- `MakeGridsForTMDR`: Update of integration procedures to adaptive. Default grids accordingly updated (no significant effect).
- `uTMDPDF`: Update of the integration procedure in `uTMDPDF`, to adaptive Gauss-Kronrod (G7-K15). with special treatment of the $x \to 1$ singularity.
- `uTMDPDF`: The procedure for evaluation of TMD for individual flavour (`uTMDPDF_lowScale(f,x,b,mu)`) is removed, as outdated.
- `uTMDPDF`: Removed argument μ, from `uTMDPDF_...(x,b)`. Added function `mu_OPE(b)`, which is used as μ-definition for TMDs.
- `uTMDPDF`: Optional griding of TMDs is added. See sec.**??**
- `TMDR`: fixed potential error in the "close-to-Landau-pole" exception.
- `TMDs`: fixed potential error in the evaluation of the gluon evolution factor.
- `TMDX`: the name convention of subroutines `CalculateXsection_...`, changed to `CalculateXsec_...`, to shorten the name length.
- `TMDX`: added functions `CalculateXsec_PTint_Qint_YintComplete(X,qtMin,qtMax,QMin,QMax)` and `CalculateXsec_Qint_YintComplete(X,qtMin,qtMax,QMin,QMax)`.
- `TMDX`&`TMDs`&`uTMDPDF`: the independent variation constant c_4 is added (in the ver.1 variation of c_3 and c_4 was simultaneous). The corresponding routines are updated.

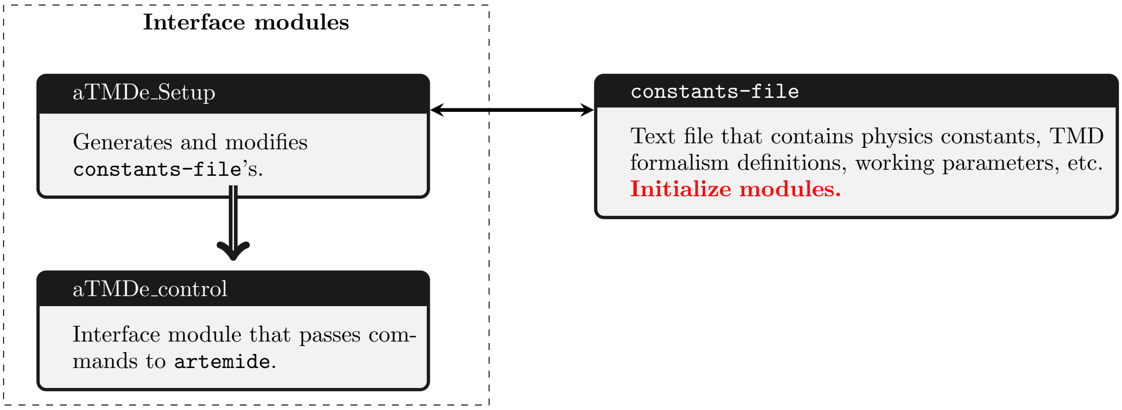
Ver.1 Release: `uTMDPDF`, `TMDR`, `TMDs` and `TMDX` modules. Only Drell-Yan-like cross-sections.

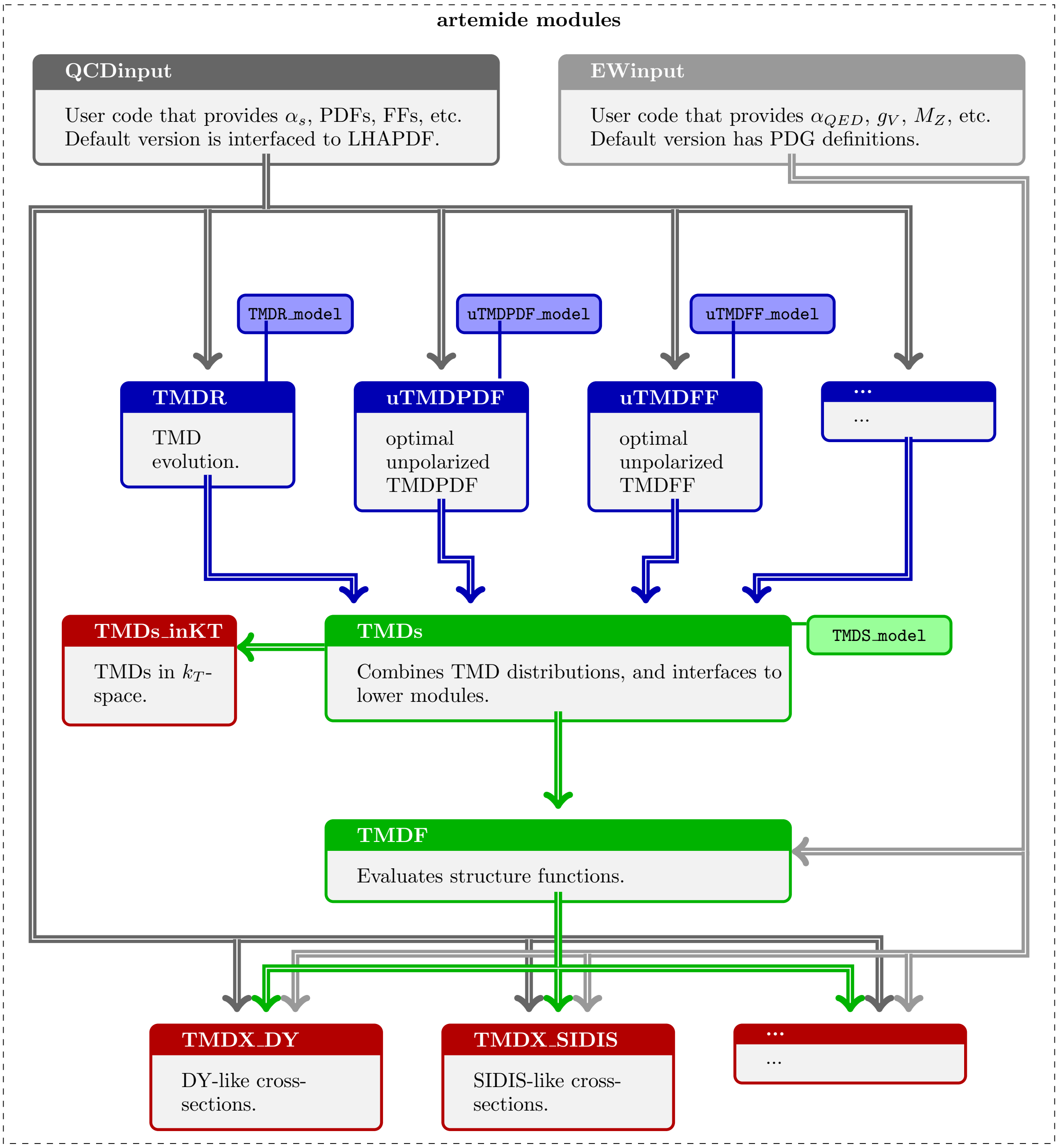
XXIV. VERSION OF INPUT-FILE

1. Initial version (corresponds to clean ver.2.00)
2. +TMD evolution type 4.
3. +linearly polarized gluon TMDPDF (sections 11, and 1.D)
4. +mass of top-quark (sec.1.A.p3)
5. +parameters of Higgs boson (mass,width,vev) (sec.2.D.p1-2.D.p3)
6. +option for accounting of π^2 corrections in coef.function for DY-like processes (sec.9.A.p3)
7. +options for specification of z-bin integrations in SIDIS module (sec.10.B.p3 & 10.B.p4)
8. +options for specification of x-bin integrations in SIDIS module (sec.10.B.p5 & 10.B.p6)
9. +options accounting q_T-correction in x_1 and z_1 in SIDIS module (sec.10.A.p5)
10. +initialization arrays are added to TMDR, uTMDPDF, uTMDFF, lpTMDPDF (secs.3.B.p2; 4.B.p2; 5.B.p2; 11.B.p2)
11. +added primary section of parameters for aTMDe-control (0.C) and the check of trigger of initialization by NP arrays (secs.0.C.p1)
12. +composite TMD option for each TMD module (secs. 4.A.p2, 5.A.p2, 11.A.p2)
13. +masses of leptons e, μ, τ (secs. 2.E.p1, 2.E.p2, 2.E.p3)
14. +value of α_{QED} at mass of tau-leptons (secs. 2.A.p4)
15. +Sivers function (sections 12)
16. +global mass scale value used in TMDF (sections 7.p2)
17. +option for accounting of exact values of factorization scales in DY (sections 9.A.p4)
18. + helicity PDF in QCDinput (section 1.E)
19. + worm gear T function (section 13)
20. +option for accounting of exact values of factorization scales in SIDIS (sections 10.A.p6)

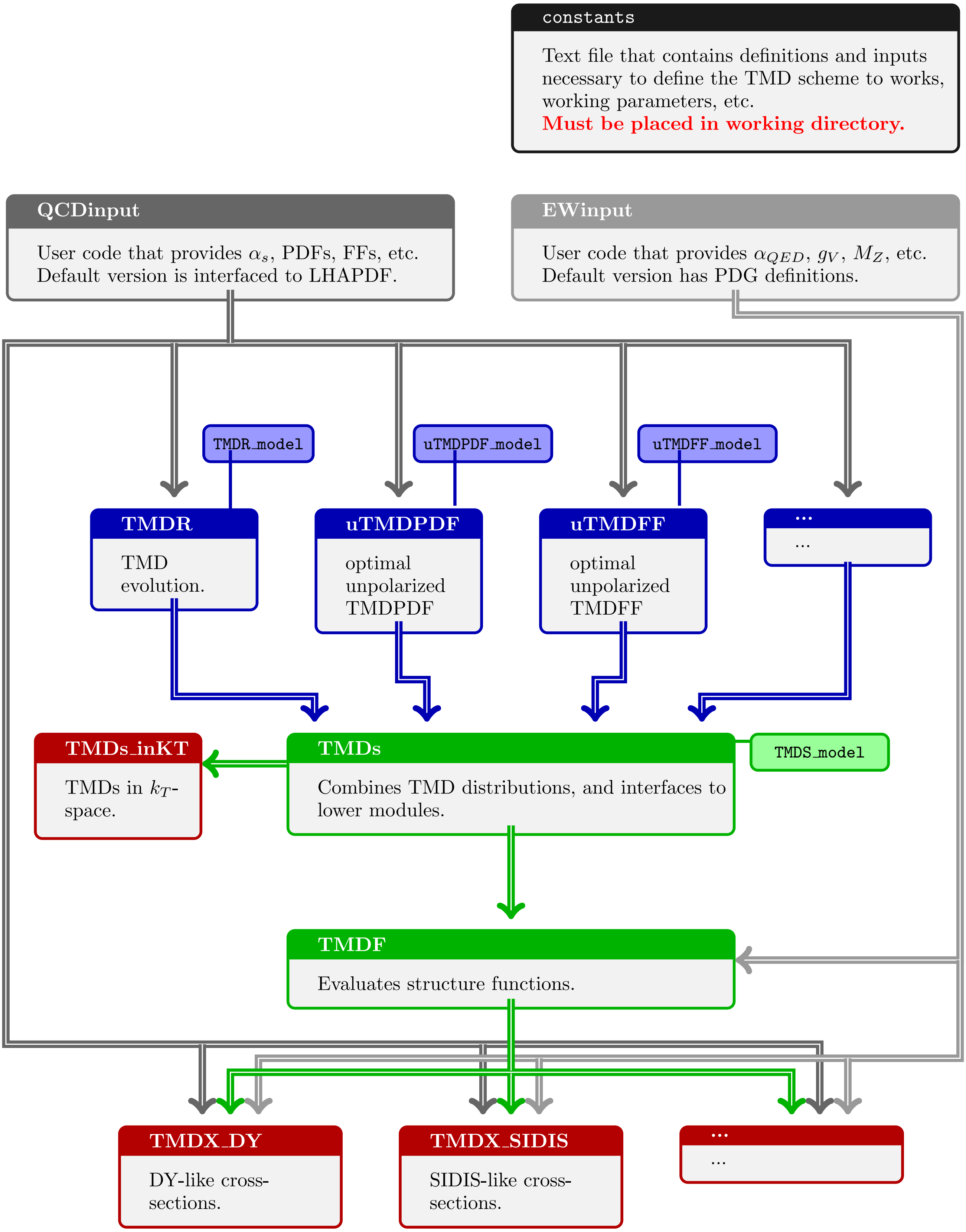
XXV. BACKUP

A. artemide structure before v2.04





B. artemide structure before v2.00



C. artemide structure before v1.3