# arTeMiDe ver.1.4

Alexey A. Vladimirov
March 26, 2019

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

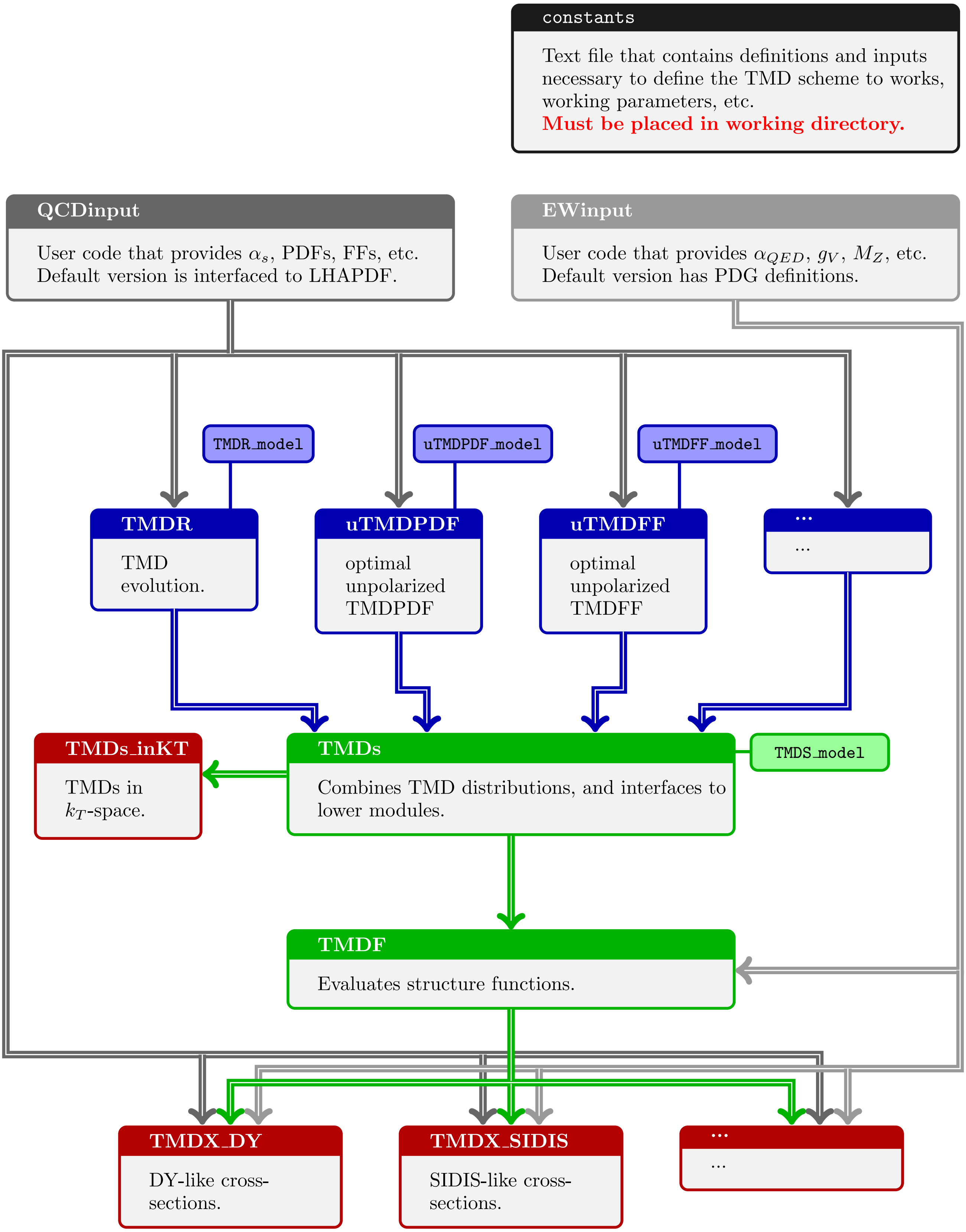
**Manual is updating.**



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*

**Repository:** https://github.com/VladimirovAlexey/artemide-public

**Contents**


constants
Text file that contains definitions and inputs necessary to define the TMD scheme to works, working parameters, etc.
Must be placed in working directory.
QCDinput
User code that provides $\alpha_s$, PDFs, FFs, etc. Default version is interfaced to LHAPDF.
EWinput
User code that provides $\alpha_{QED}$, $g_V$, $M_Z$, etc. Default version has PDG definitions.
TMDR_model
uTMDPDF_model
uTMDFF_model
TMDR
TMD evolution.
uTMDPDF
optimal unpolarized TMDPDF
uTMDFF
optimal unpolarized TMDFF
...
...
TMDs_inKT
TMDs in $k_T$-space.
TMDs
Combines TMD distributions, and interfaces to lower modules.
TMDS_model
TMDF
Evaluates structure functions.
TMDX_DY
DY-like cross-sections.
TMDX_SIDIS
SIDIS-like cross-sections.
...
...

## Glossary

**TMD = Transverse Momentum Dependent**
**PDF = Parton Distribution Function**
**FF = Fragmentation Function**
**NP = Non-perturbative**

# I. GENERAL STRUCTURE AND USER INPUT OF ARTEMIDE

## A. General structure

The arTeMiDe package is a set of Fortran modules for evaluation of individual TMD distributions, TMD evolution factors, and other ingredients of TMD factorization theorem. Each module can be used as it is. The highest-order task is the evaluation of cross-section within the TMD factorization theorem, including all needed integrations, and factor, 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 structure of modules dependencies is presented on the diagram (shown on the previous page).

> **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 user interface. So, the interface is changing from version to version, and often is not compatible with earlier versions. It slowly converge to the perfect (almost) shape convenient for a wide audience. If you have a special task and not sure how to operate with artemide in this case, better write an e-mail.

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

$$\frac{d\sigma}{dX}=d\sigma(q_T)=prefactor1\ \times\ prefactor2\ \times F, \tag{1.1}$$

where $prefactor1$ a process-dependent prefactor, $prefactor2$ is the experiment dependent prefactor, and $F$ is the reduced structure function. The role of prefactors 1 and 2 is naturally interchangeable, however, we expect that all universal factorization dependence (such as hard parts, parton cross-sections) is concentrated in $prefactor1$, while non-universal dependence on the particular measurement (e.g phase space multiplies, cut functions) is concentrated in the $prefactor2$. Example, for the photon induced Drell-Yan process one has for $d\sigma/dq_T$

$$\begin{aligned}
prefactor1 &= \frac{4\pi}{9sQ^2}|C_V(Q,\mu_H)|^2\\
prefactor2 &= 1+\frac{q_T^2}{2Q^2}\\
F &= \int\frac{bdb}{2}J_0(bq_T)\sum_f|e_f|^2F_f(x_A,b;\mu_H,\zeta_A)F_{\bar f}(x_B,b;\mu_H,\zeta_B).
\end{aligned}$$

The structure function $F$ is generally defined as

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

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.

Nesting structure of arTeMiDe and responsible modules

| Quantity | Rough formula | Name | Module |
|---|---|---|---|
| $d\sigma$ | $\int d[\text{bin}](prefactors)\times F$ | cross-section | TMDX |
| $F$ | $\int\frac{bdb}{2}b^nJ_n(bq_T)\sum_{ff'}z_{ff'}F_1^fF_2^{f'}$ | structure function | TMDF |
| $F_i^f$ | $R^f(\mu_f\to\mu_i)F_i^f(\mu_i)$ | evolved TMD distribution | TMDs |
| $R$ | | TMD evolution factor | TMDR |
| $F_i^f(\mu_i)$ | $C\otimes ff_{NP}$ | model for TMD distribution | depends on kind |

### B. 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 `arTeMiDe` is evolving, and I do not really care about back compatibility.

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 the `arTeMiDe` library, such as amount of output tolerance of integration routines, number of NP parameters, type of evolution, griding parameters, triggering of particular contributions, etc. There are many of them, and typically they are unchanged. These are set in the file (in text format) : `constants`. **Changes do not require recompilation.**

**External physics input:** It includes the definition of $\alpha_s$, collinear PDFs, and other distributions. They are provided by user via the "interface" module `QCDinput`. Default version is uses LHAPDF interface [4]. For non-QCD parameters, e.g. $\alpha_{QED}$, SM parameters, there is a module `QEDinput`. **Changes require recompilation.** For further information see sec.II.

**NP model:** The NP model consists in NP profiles of TMD distributions, NP model for large-b evolution, selection of scales $\mu$, 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.** See also section of corresponding modules.

**Comments:**

- `constants` must be in the same directory as the executing file.
- 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 maximum number of parameters in the model for each module is set in `constants`-file.
- The directory `/Model` is convenient to keep as is. We provide our extraction as such directories. However, `QCDinput` and `QEDinput` are not a part of model (although, in bigger sense they are).

### C. Setup

Download and unpack `arTeMiDe`. The actual code is in the /src. Check the `makefile`. You have to fix 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`).

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! It could be done automatically if you put your code in directory /Prog, and call for `make program TARGET=...`, where ... is the name of the file with code.

## II. 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 [4].

List of available commands

| Command | Description |
|---|---|
| `QCDinput_Initialize(order)` | Subroutine to initialize anything what is needed. (string) parameter `order` is passed from arTeMiDe modules and conventionally takes values `LO`,`LO+`,`NLO`,`NLO+`,`NNLO`,`NNLO+` (see e.g. secVIII B). According to this variable the QCD input should be initialized. |
| `QCDinput_SetPDFreplica(num)` | Changes the PDF replica number. |
| `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). |

## III. 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

$$\frac{d\sigma}{dX} = d\sigma(q_T) = prefactor1 \int [bin]\ prefactor2\ \times F, \tag{3.1}$$

where $dX \sim dQdsdydq_T$. The $prefactor2$ is expected to be $x-$independent[1]. All $x$-dependence is set into $F$. Generally, the *prefactor2* $\times F$ is

$$prefactor2 \times F = (\text{cuts for lepton pair}) \times H(Q,\mu_H)F(Q,q_T,x_1,x_2,\mu,\zeta_1,\zeta_), \tag{3.2}$$

where $F$ is defined in (5.1). There are following feature of current implementation

- In the current version the scaling variables are set as

$$\mu^2 = \zeta_1 = \zeta_2 = Q^2. \tag{3.3}$$

  Currently, it is hard coded and could not be easily modified. However, there is a possibility to vary the value of $\mu$ as $\mu = c_2 Q$, where $c_2$ is variation parameter (see sec.III 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 [10]. 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. | III A | Initialization of module. |
| `TMDX_DY_SetNPParameters(...)` | subrout. | III B | Set new NP parameters to the modules |
| `TMDX_DY_SetProcess(p)` | subrout. | III B | Set the process |
| `SetCuts(inc,pT,eta1,eta2)` | subrout. | III B | Set the current evaluation of cut for lepton pair. |
| `TMDX_DY_XSetup(s,Q,y)` | subrout. | III B | Set the kinematic variables |
| `TMDX_DY_SetScaleVariations(c1,c2,c3,c4)` | subrout. | – | Set new values for the scale-variation constants. |
| `TMDX_DY_ShowStatistic()` | subrout. | – | Print current statistic on the number of calls. |
| `CalcXsec_DY...` | subrout. | III C | Evaluates cross-section. Many overloaded versions see sec.III C. |
| `xSec_DY(X,proc,s,qT,Q,y,iC,CutP,Num)` | subrout. | III 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. | III C | Evaluates cross-section completely integrated over the bin over the list. Can be called without preliminary `...Set...`'s. |

[1] Not necessary in ver.1.4.

### A. Initialization

Prior the usage module is to be initialized (once per run) by
`call TMDX_DY_Initialize(order)`
here: `order` is the declaration of order. It can be `'LO'`, `'LO+'`, `'NLO'`, `'NLO+'`,`'NNLO'` or `'NNLO+'`. This is a complex declaration, which implies particular orders for coefficient functions, PDFs, anomalous dimension, etc. For detailed definition see other sections.

### 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, and generally experimental dependent. It also influence the bin-integration definition.

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

`p3` (integer) Defines the structure function $F$. See sec.V D.

Alternatively, process can be declared by a shorthand version
`call TMDX_DY_SetProcess(p)`
where `p`(integer) corresponds to particular combinations of `p1,p2,p3`. See table III 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 `TMDs` and lower-level modules. For user convenience there is a subroutine, which passes the values of parameters to `TMDs`. It is
`call TMDX_DY_SetNPParameters(lambda/n)`
where `lambda` is real*8(1:number of parameters) or `n` is the label of replica. See also VI C.
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

$$|l_T|, |l'_T| < \texttt{pT}, \qquad \texttt{eta1} < \eta, \eta', \texttt{eta2}, \tag{3.4}$$

where $l$ and $l'$ are the momenta of produced leptons, with $l_T$'s being their transverse components and $\eta$'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)`.
**Current realization prevents usage of different cuts in parallel computation. Will be fixed in later versions.**

### 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_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)` | $\int_{Q_{\min}}^{Q_{\max}} 2QdQd\sigma(q_T)$ | |
| `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 | | |
| **In version 1.3** | | |
| `CalcXsec_DY_XFint(X,qt,xfMin,xfMax)` | $\int_{x_F \min}^{x_F \max} dx_F d\sigma(q_T)$ | $x_F = \frac{2\sqrt{Q^2+\mathbf{q}_T^2}}{\sqrt{s}} \sinh y$ $dx_F = \frac{2\sqrt{Q^2+\mathbf{q}_T^2}}{\sqrt{s}} \cosh y\, dy$ |
| `CalcXsec_DY_Qint_XFint (X,qt,Qmin,Qmax,xfMin,xfMax)` | $\int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{x_F \min}^{x_F \max} dx_F d\sigma(q_T)$ | |
| `CalcXsec_DY_PTint_Qint_XFint (X,qtmin,qtmax,Qmin,Qmax,xfMin,xfMax)` | $\int_{q_T \min}^{q_T \max} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{x_F \min}^{x_F \max} dxFd\sigma(q_T)$ | Integration over $q_T$ is Simpsons by default number of sections. |
| `CalcXsec_DY_PTint_Qint_XFint (X,qtmin,qtmax,Qmin,Qmax,xfMin,xfMax,N)` | $\int_{q_T \min}^{q_T \max} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{x_F \min}^{x_F \max} dx_F d\sigma(q_T)$ | Integration over $q_T$ is Simpsons by $N$-section. |

**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).

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 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

For the historical reasons the initial code of `arTeMiDe` did not allow parallelization (it was due to intensive usage of shared grids). I am working on this problem, improving different parts of code and excluding/encapsulating sharing constructions. After ver.1.32 there is a limited parallelization on the level of computation of the cross-section (the most expansive level of computation). The parallelization option is planned to improve in the future, making possible to evaluate large cross-experiment lists of cross-sections in parallel.

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.

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, in the section 6.

> **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 `CutFactor4`.

- 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 `CutFactor4(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. 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

$$\begin{aligned} x_1 &= \frac{q^+}{p_1^+} = \frac{Q}{\sqrt{s}} e^y \sqrt{1 + \frac{\boldsymbol{q}_T^2}{Q^2}} \simeq \frac{Q}{\sqrt{s}} e^y, \\ x_1 &= \frac{q^+}{p_1^+} = \frac{Q}{\sqrt{s}} e^{-y} \sqrt{1 + \frac{\boldsymbol{q}_T^2}{Q^2}} \simeq \frac{Q}{\sqrt{s}} e^{-y}. \end{aligned} \tag{3.5}$$

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 by the parameter 5.A.1) in `constants`.

### 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 | $\frac{4\pi}{9}\frac{\alpha_{\text{em}}^2(Q)}{sQ^2}\lvert C_V^{DY}(c_2Q)\rvert^2 R\times\text{cut}(q_T)$ | DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$. 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 | $\frac{4\pi}{9}\frac{\alpha_{\text{em}}^2(Q)}{sQ^2}\lvert C_V^{DY}(c_2Q)\rvert^2 R\times\text{cut}(q_T)$ | DY-cross-section $\frac{d\sigma}{dQ^2dyd(q_T^2)}$. 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 | $\frac{4\pi^2}{3}\frac{\alpha_{\text{em}}(Q)}{s}Br_Z\lvert C_V^{DY}(c_2Q)\rvert^2 R\times\text{cut}(q_T)$ | 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. 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 | $\frac{4\pi^2}{3}\frac{\alpha_{\text{em}}(Q)}{s}Br_W\lvert C_V^{DY}(c_2Q)\rvert^2 R\times\text{cut}(q_T)$ | 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. 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$^2$ |

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. |

## IV. TMDX_SIDIS MODULE

**In ver.1.4, this module has not been checked for bugs, and errors so well as DY module. I do not guaranty all correct factors. I hope to do so in ver. 1.5.**

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

$$\frac{d\sigma}{dX} = d\sigma(q_T) = prefactor1 \int [bin]\; prefactor2\; \times F, \tag{4.1}$$

where $dX \sim dl'$, with $l'$ momentum of scattered lepton.

The $prefactor2$ is expected to be $x-$independent. All $x$-dependence is set into $F$.

This section is to be updated by definition of kinematics

List of available commands

| Command | Type | Sec. | Short description |
|---|---|---|---|
| `TMDX_SIDIS_Initialize(order)` | subrout. | III A | * Initialization of module. |
| `TMDX_SIDIS_SetNPParameters(...)` | subrout. | III B | * Set new NP parameters to the modules |
| `TMDX_SIDIS_SetProcess(p)` | subrout. | IV A | Set the process |
| `TMDX_SIDIS_XSetup(s,z,x,Q,mTARGET,mPRODUCT)` | subrout. | IV A | Set the kinematic variables (`mTARGET` and `mPRODUCT` are optional variables) |
| `TMDX_SIDIS_SetCuts(inc,yMin,yMax,W2)` | subrout. | IV C | Set global values of cuts |
| `TMDX_SIDIS_SetScaleVariations(c1,c2,c3,c4)` | subrout. | – | * Set new values for the scale-variation constants. |
| `TMDX_SIDIS_ShowStatistic()` | subrout. | – | * Print current statistic on the number of calls. |
| `CalcXsec_SIDIS...` | subrout. | IV B | Evaluates cross-section. Many overloaded versions see sec.IV B. |
| `xSec_SIDIS(X,process,s,pT,z,x,Q)` | subrout. | IV 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. | IV 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.V D.

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 VI C.

### 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`<br>`(X,pt,zMin,zMax,xMin,xMax,Qmin,Qmax)` | $\int_{z_{\min}}^{z_{\max}} dz \int_{x_{\min}}^{x_{\max}} dx$<br>$\int_{Q_{\min}}^{Q_{\max}} 2QdQd\sigma(p_T)$ | $0 < z_{\min} < z_{\max} < 1$<br>$0 < x_{\min} < x_{\max} < 1$<br>$0 < Q_{\min} < Q_{\max}$ |
| `CalcXsec_SIDIS_PTint_Zint_Xint_Qint`<br>`(X,ptMin,ptMax,zMin,zMax,xMin,xMax,Qmin,Qmax)` | $\int_{z_{\min}}^{z_{\max}} dz \int_{x_{\min}}^{x_{\max}} dx$<br>$\int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{p_{T\min}}^{p_{T\max}} 2p_T dp_T d\sigma(p_T)$ | $0 < z_{\min} < z_{\max} < 1$<br>$0 < x_{\min} < x_{\max} < 1$<br>$0 < Q_{\min} < Q_{\max}$ |
| 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,W0)` 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,W0)
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

$$y_{\min} < y < y_{\max}, \qquad W^2 > W_0^2. \tag{4.2}$$

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

$$\max\{x_{\min}, \frac{Q^2}{y_{\max}(s-M^2)}\} < x < \min\{x_{\max}, \frac{Q^2}{y_{\min}(s-M^2)}, \frac{Q^2}{Q^2-M^2-W_0^2}\}, \tag{4.3}$$

$$\max\{Q^2_{\min}, x_{\min}y_{\min}(s-M^2), \frac{x_{\min}(W_0^2-M^2)}{1-x_{\min}}\} < Q^2 < \min\{Q^2_{\max}, x_{\max}y_{\max}(s-M^2)\}. \tag{4.4}$$

### 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/phas 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.IV F, 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).

The accounting of these corrections is switched on/off in `constants`. For a moment (v1.41) the values of masses $M$ and $m$ are set universally for all processes.

### E. 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 [11]. |

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

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

## F. 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.[11]. Note, that some parts of definition were derived by me, since I have not found them in the literature.

### 1. Kinematics

The process is

$$H(P) + l(l) \to l(l') + h(p_h) + X, \tag{4.5}$$

with

$$P^2 = M^2, l^2 = l'^2 = m_l^2 \simeq 0, \qquad p_h^2 = m^2. \tag{4.6}$$

The standard variables used for the description of SIDIS are

$$q = l - l' \qquad \Rightarrow \qquad Q^2 = -q^2, \qquad x = \frac{Q^2}{2(Pq)}, \qquad y = \frac{(Pq)}{(Pl)}, \qquad z = \frac{(Pp_h)}{(Pq)}. \tag{4.7}$$

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

$$\gamma = \frac{2Mx}{Q}, \qquad \rho = \frac{m}{zQ}, \qquad \boldsymbol{\rho}_\perp^2 = \frac{m^2 + \boldsymbol{p}_\perp^2}{z^2Q^2}. \tag{4.8}$$

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 $dydQ^2$:

$$dxdQ^2 = \frac{Q^2}{y}dxdy = \frac{x}{y}dydQ^2. \tag{4.9}$$

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

$$\{s, x, Q^2, z, \boldsymbol{p}_T^2\}. \tag{4.10}$$

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

### 2. Cross-section

The cross-section for SIDIS has the general form

$$\frac{d\sigma}{dxdQ^2dzd\psi d\phi_h d\boldsymbol{p}_{h\perp}^2} = \frac{\alpha_{\text{em}}^2(Q)}{Q^6}\frac{y^2}{8z}\frac{L_{\mu\nu}W^{\mu\nu}}{\sqrt{1+\boldsymbol{\rho}_\perp^2\gamma^2}}, \tag{4.11}$$

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

$$W^{\mu\nu} = \frac{-zg_T^{\mu\nu}}{\pi}\int |\boldsymbol{b}|d|\boldsymbol{b}|J_0(|\boldsymbol{b}||\boldsymbol{q}_T|)\sum_f e_f^2|C_V(-Q^2,\mu^2)|^2 F_1^f(x_1,\boldsymbol{b})D_1^f(z_1,\boldsymbol{b}) + ... \ , \tag{4.12}$$

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

$$x_1 = -\frac{2x}{\gamma^2}\left(1 - \sqrt{1+\gamma^2\left(1-\frac{\boldsymbol{q}_T^2}{Q^2}\right)}\right), \tag{4.13}$$

$$z_1 = -z\frac{1-\sqrt{1+\left(1-\frac{\boldsymbol{q}_T^2}{Q^2}\right)\gamma^2}}{\gamma^2}\frac{1+\sqrt{1-\rho^2\gamma^2}}{1-\frac{\boldsymbol{q}_T^2}{Q^2}}. \tag{4.14}$$

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

$$|\boldsymbol{q}_T| = \frac{|\boldsymbol{p}_{h\perp}|}{z}\sqrt{\frac{1+\gamma^2}{1-\gamma^2\rho^2}}, \qquad \Leftrightarrow \qquad |\boldsymbol{p}_{h\perp}| = z|\boldsymbol{q}_T|\sqrt{\frac{1-\gamma^2\rho^2}{1+\gamma^2}}. \tag{4.15}$$

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

$$\begin{aligned}\frac{d\sigma}{dxdQ^2dzd\boldsymbol{p}_{h\perp}^2} &= 2\pi\frac{\alpha_{\text{em}}^2}{Q^4}\frac{1}{\sqrt{1+\boldsymbol{\rho}_\perp^2\gamma^2}}\frac{y^2}{2(1-\varepsilon)}\left\{1+\left(\varepsilon-\frac{\gamma^2}{2}\right)\frac{\boldsymbol{\rho}_\perp^2-\rho^2}{1-\gamma^2\rho^2}\right\} \\ &\int|\boldsymbol{b}|d|\boldsymbol{b}|J_0\left(\frac{|\boldsymbol{b}||\boldsymbol{p}_{h\perp}|}{z}\sqrt{\frac{1+\gamma^2}{1-\gamma^2\rho^2}}\right)\sum_f e_f^2|C_V(-Q^2,\mu^2)|^2F_1^f(x_1,\boldsymbol{b})D_1^f(z_1,\boldsymbol{b}),\end{aligned} \tag{4.16}$$

where

$$\varepsilon = \frac{1-y-\frac{\gamma^2y^2}{4}}{1-y+\frac{y^2}{2}+\frac{y^2\gamma^2}{4}}. \tag{4.17}$$

## V. TMDF MODULE

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

$$F(Q^2, q_T, x_1, x_2, \mu, \zeta_1, \zeta_2) = \int_0^\infty \frac{bdb}{2} b^n J_n(bq_T) \sum_{ff'} z_{ff'}(Q^2) F_1^f(x_1, b; \mu, \zeta_1) F_2^{f'}(x_2, b; \mu, \zeta_2), \tag{5.1}$$

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.
- $\mu$ is the hard factorization scale $\mu \sim Q$
- $\zeta_{1,2}$ are rapidity factorization scales. In the standard factorization scheme $\zeta_1\zeta_2 = Q^4$.
- $f, f'$ 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.

The module has simple structure since it evaluates only this integral and does not require any exra input.

List of available commands

| Command | Type | Sec. | Short description |
|---|---|---|---|
| `TMDF_Initialize(order)` | subrout. | V A | Initialization of module. |
| `TMDF_SetNPParameters(...)` | subrout. | V A | Set new NP parameters to the modules |
| `TMDF_SetScaleVariations(c1,c3,c4)` | subrout. | ?? | Set new values for the scale-variation constants. |
| `TMDF_ShowStatistic())` | subrout. | – | Print current statistic on the number of calls. |
| `TMDF_F(Q2,qT,x1,x2,mu,zeta1,zeta2,N)` | (real*8) | V C | Evaluates the structure function |

### A. Initialization

Prior the usage module is to be initialized (once per run) by
`call TMDF_Initialize(order)`
here:

`order` declaration of order the order used by package. It can be `'LO'`, `'LO+'`, `'NLO'`, `'NLO+'`, `'NNLO'` and `'NNLO+'`. This declaration is passed to the lower packages, where is should be defined.

### B. NP parameters and scale variation

To set particular values of these NP parameters, and to vary the scales use
`call TMDF_SetNPParameters(lambda/n)`
and
`call TMDFs_SetScaleVariations(c1,c3,c4)`
respectively. **In fact, this subroutine just pass the initialization request to the module `TMDs`, see VI A and VI F.** The arguments of subroutines also defines in these sections.

### C. 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 $\mu$ in GeV. Typically, $\mu = Q$.

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

`zeta2` (real*8) The scale $\zeta_f$ 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

$$F^N(Q^2, q_T, x_1, x_2, \mu, \zeta_1, \zeta_2) = \int_0^\infty \frac{bdb}{2} b^n J_n(bq_T) \sum_{ff'} z^N_{ff'}(Q^2) F_1^f(x_1, b; \mu, \zeta_1) F_2^{f'}(x_2, b; \mu, \zeta_2). \tag{5.2}$$

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.

$$\int_0^\infty \frac{bdb}{2} b^n I(b) J_n(q_T b) \simeq \frac{1}{q_T^{n+2}} \sum_{k=1}^\infty \tilde{\omega}_{nk} b_{nk}^{n+1} I\left(\frac{b_{nk}}{q_T}\right), \tag{5.3}$$

where

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

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

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.

- For $q_T < 1$ the integrand is narrower, and precision of the integration is not covered by given $h$. For this region we use tables with $\bar{h} = 0.05h$. Typically, it is enough to have reasonable precision at $q_T \sim 0.01$.
- 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.

### D. Enumeration of structure functions

List of enumeration of structures functions
`N<10000`

| `N` | $z_{ff'}$ | $F_1$ | $F_2$ | Short description | Gluon req. |
|---|---|---|---|---|---|
| 0 | – | – | – | Test case | no |
| 1 | $\delta_{\bar{f}f'}\|e_f\|^2$ | $f_1$ | $f_1$ | (unpol.)$p + p \to \gamma$ | no |
| 2 | $\delta_{ff'}\|e_f\|^2$ | $f_1$ | $f_1$ | (unpol.)$p + \bar{p} \to \gamma$ | no |
| 3 | $\delta_{\bar{f}f'}\frac{(1-2\|ef\|s_w^2)^2 - 4e_f^2 s_w^4}{8s_w^2 c_w^2}$ | $f_1$ | $f_1$ | (unpol.)$p + p \to Z$ | no |
| 4 | $\delta_{ff'}\frac{(1-2\|ef\|s_w^2)^2 - 4e_f^2 s_w^4}{8s_w^2 c_w^2}$ | $f_1$ | $f_1$ | (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 | $\frac{1}{4s_w^2}\frac{\|V_{ff'}\|^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2 M_W^2}$ | $f_1$ | $f_1$ | (unpol.)$p + p \to W^+$ | no |
| 8 | $\frac{1}{4s_w^2}\frac{\|V_{ff'}\|^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2 M_W^2}$ | $f_1$ | $f_1$ | (unpol.)$p + p \to W^-$ | no |
| 9 | $\frac{1}{4s_w^2}\frac{\|V_{ff'}\|^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2 M_W^2}$ | $f_1$ | $f_1$ | (unpol.)$p + p \to W^+ + W^-$ | no |
| 10 | $\frac{1}{4s_w^2}\frac{\|V_{ff'}\|^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2 M_W^2}$ | $f_1$ | $f_1$ | (unpol.)$p + \bar{p} \to W^+$ | no |
| 11 | $\frac{1}{4s_w^2}\frac{\|V_{ff'}\|^2}{4s_w^2}\frac{Q^4}{(Q^2-M_W^2)^2+\Gamma_W^2 M_W^2}$ | $f_1$ | $f_1$ | (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.)$p+\bar{p}\to W^++W^-$ | 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.)$p+p\to W^++W^-$ (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.)$p+\bar{p}\to W^++W^-$ (for narrow-width approx.) | no |
| 1001 | $R^{Cu}_{\bar{f}f'}e_fe_{f'}$ 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 | $R^{^2H}_{\bar{f}f'}e_fe_{f'}$ 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 |
| 2001 | $\delta_{ff'}\lvert e_f\rvert^2$ | $f_1^H$ | $d_1^{h_1}$ | (unpol)$H+\gamma\to h_1$ | no |
| 200N | $\delta_{ff'}\lvert e_f\rvert^2$ | $f_1^H$ | $d_1^{h_N}$ | (unpol)$H+\gamma\to h_N$, for $N=\{1,...,9\}$ (including case 2001) | no |

10000⩽N<20000

| N | $z_{ff'}$ | $F_1$ | $F_2$ | Short description | Gluon req. |
|---|---|---|---|---|---|
| 10000 | – | – | – | Test case | no |

20000⩽N<30000

| N | $z_{ff'}$ | $F_1$ | $F_2$ | Short description | Gluon req. |
|---|---|---|---|---|---|
| 20000 | – | – | – | Test case | no |

30000⩽N

| N | $z_{ff'}$ | $F_1$ | $F_2$ | Short description | Gluon req. |
|---|---|---|---|---|---|
| 30000 | – | – | – | Test case | no |

**Notes on parameters and notation:**
$s_w^2$, $M_Z$, $\Gamma_Z$, etc. are defined in `constants`
$f_1$-unpolarized TMDPDF
$d_1$-unpolarized TMDFF

## VI. TMDS MODULE

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

$$F_f(x, b; \mu, \zeta) = R_f[b, (\mu, \zeta) \to (\mu_i, \zeta_{\mu_i})]\tilde{F}_f(x, b), \tag{6.1}$$

where $R$ is the TMD evolution kernel, $\tilde{F}$ is a TMD distribution at low scale. The scale $\mu_i$ 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(order)` | subrout. | VI A | Initialization of module. |
| `TMDs_SetNPParameters(lambda)` | subrout. | VI A | Set new NP parameters to the modules |
| `TMDs_SetNPParameters(n)` | subrout. | VI A | Set NP parameters corresponding to replica $n$. |
| `TMDs_SetPDFreplica(n)` | subrout. | – | Set replica for PDF. (temporary) |
| `TMDs_SetScaleVariations(c1,c3,c4)` | subrout. | VI F | Set new values for the scale-variation constants. |
| `uTMDPDF_5(x,b,mu,zeta,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD PDF (gluon term undefined) |
| `uTMDPDF_50(x,b,mu,zeta,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD PDF (gluon term defined) |
| `uTMDFF_5(x,b,mu,zeta,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD FF (gluon term undefined) |
| `uTMDFF_50(x,b,mu,zeta,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD FF (gluon term defined) |
| `uTMDPDF_5(x,b,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD PDF at optimal line (gluon term undefined) |
| `uTMDPDF_50(x,b,,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD PDF at optimal line (gluon term defined) |
| `uTMDFF_5(x,b,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD FF at optimal line(gluon term undefined) |
| `uTMDFF_50(x,ba,h)` | (real*8(-5:5)) | VI C | Unpolarized TMD FF at optimal line (gluon term defined) |
| `uPDF_uPDF(x1,x2,b,mu,zeta,h1,h2)` | (real*8(-5:5)) | VI D | 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)) | VI D | 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 inputs

| Input | Setup by | Short description |
|---|---|---|
| `mu_LOW(b)` | write-in | The value of $\mu_i$ used in the evolutions of type 1 and 2 (improved $\mathcal{D}$ and $\gamma$). See [3]. |
| `mu0(b)` | write-in | The value of $\mu_0$ used in the evolution of type 1 (improved $\mathcal{D}$). See [3]. |

### A. Initialization

Prior the usage module is to be initialized (once per run) by
`call TMDs_Initialize(order)`
here:

`order` declaration of order the order used by package. It can be `'LO'`, `'LO+'`, `'NLO'`, `'NLO+'`, `'NNLO'` and `'NNLO+'`. This declaration is passed to the lower packages, where is should be defined.

**This subroutine also initializes modules for individual TMDs according to the number of NP parameters.**

The important part of the initialization is the number of NP parameters for each TMD under consideration. Each TMD-evaluating module (say, `uTMDPDF`, `uTMDFF`, etc.) requires $n_i$ number of parameters. We expect $n_i > 0$, 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 $n_i > 0$). The numbers $n_i$ are read from the `constants`-file, in fixed order, i.e. $n_0$ is for `TMDR`, $n_1$ is for `uTMDPDF`, $n_2$ is for `uTMDFF`, and so on (see `constants`-file). These numbers are read during the initialization procedure `TMDs_Initialize`.

**If $n_i = 0$ $(i > 0)$ the corresponded module would be not-initialized and not available in calculation.**

To set particular values of these parameters use
`call TMDs_SetNPParameters(lambda)`
where

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

**Important:** There is a check on the minimal input length of `lambda`. It should be at least $\sum_i n_i$. Otherwise, the command is ignored (with warning).

**Optional:** There exist the overloaded version of `TMDs_SetNPParameters(n)`, with `n` being an integer. It attempt to load user defined set of NP parameters associated with number $n$. Practically, it calls submodules with request to set replica $n$.

### B. Definition of low-scales

The low scales $\mu_i$ and $\mu_0$ 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.

### C. 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 $\mu_f$ in GeV. Typically, $\mu_f = Q$.

`zeta` (real*8) The scale $\zeta_f$ 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 [3]. Practically, it just transfers the outcome of corresponding TMD module, e.g.module `uTMDPDF`, see sec.IX.

### D. 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 $\mu_f$ in GeV. Typically, $\mu_f = Q$.

`zeta` (real*8) The scale $\zeta_f$ 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.

### 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 $\chi^2$ entry requires $\sim 16 \times 10^6$ calls of `uTMDPDF_...` Every call of `uTMDPDF_..` at NNLO order, requires $\sim 200$ calls of pdfs, depending on $x$, $b$ and $\lambda$'s. Therefore, in such situations it is much more cheaper to make a grid of TMD distributions for 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 griding procedure is switches by the changing `makeGrid` in the file `constants`. If grid precalculation is ON, then every change of non-perturbative parameters activate the procedure of grid calculation (long). After that (until the next change of non-perturbative) the TMD values will be extracted from the grid (fast).

### F. Theoretical uncertainties

`TMDs_SetScaleVariations(c1,c3,c4)` changes the scale multiplicative factors $c_i$ (see [3], 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.

## VII. TMDS_INKT MODULE

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

$$F(x, \boldsymbol{k}_T; \mu, \zeta) = \int \frac{d^2\boldsymbol{b}}{(2\pi)^2} F(x, \boldsymbol{b}; \mu, \zeta) e^{-i(\boldsymbol{k}_T \boldsymbol{b})}. \tag{7.1}$$

Since all evaluated TMDs depends only on the modulus of $\boldsymbol{b}$, within the module we evaluate

$$F(x, |\boldsymbol{k}_T|; \mu, \zeta) = \int \frac{d|\boldsymbol{b}|}{2\pi} |\boldsymbol{b}| J_0(|\boldsymbol{b}||\boldsymbol{k}_T|) F(x, |\boldsymbol{b}|; \mu, \zeta). \tag{7.2}$$

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 VI. For the evaluation of Hankel integral we use the Ogata quadrature see V C (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. | VI A | Initialization of module. |
| TMDs_inKT_SetNPParameters(lambda) | subrout. | VI A | Set new NP parameters to the modules |
| TMDs_inKT_SetNPParameters(n) | subrout. | VI A | Set NP parameters corresponding to replica $n$. |
| TMDs_inKT_SetScaleVariations(c1,c3,c4) | subrout. | VI F | Set new values for the scale-variation constants. |
| TMDs_inKT_ShowStatistic | subrout. | - | Print current statistic on the number of calls. |
| uTMDPDF_kT_5(x,kT,h) | (real*8(-5:5)) | VI C | Unpolarized TMD PDF at the optimal line (gluon term undefined) |
| uTMDPDF_kT_50(x,kT,h) | (real*8(-5:5)) | VI C | Unpolarized TMD PDF at the optimal line (gluon term defined) |
| uTMDPDF_kT_5(x,kT,mu,zeta,h) | (real*8(-5:5)) | VI C | Unpolarized TMD PDF (gluon term undefined) |
| uTMDPDF_kT_50(x,kT,mu,zeta,h) | (real*8(-5:5)) | VI C | Unpolarized TMD PDF (gluon term defined) |
| uTMDFF_kT_5(x,kT,h) | (real*8(-5:5)) | VI C | Unpolarized TMD FF at the optimal line (gluon term undefined) |
| uTMDFF_kT_50(x,kT,h) | (real*8(-5:5)) | VI C | Unpolarized TMD FF at the optimal line (gluon term defined) |
| uTMDFF_kT_5(x,kT,mu,zeta,h) | (real*8(-5:5)) | VI C | Unpolarized TMD FF (gluon term undefined) |
| uTMDFF_kT_50(x,kT,mu,zeta,h) | (real*8(-5:5)) | VI C | Unpolarized TMD FF (gluon term defined) |

**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 $|u| + |d| + |g|$ combination.

## VIII. TMDR MODULE

The module TMDR performs the evaluation of the TMD evolution kernel in the $(\mu, \zeta)$-plane.

List of available commands

| Command | Sec. | Short description |
|---|---|---|
| `TMDR_Initialize(order)` | VIII B | Initialization of module. |
| `TMDR_setNPparameter(...)` | VIII C | Set new NP parameters used in `DNP` and `zetaNP`. |
| `TMDR_R(...)` | VIII D | Evolution kernel from $(\mu_f, \zeta_f)$ to $(\mu_i, \zeta_i)$. |
| `TMDR_Rzeta(...)` | VIII D | Evolution kernel from $(\mu_f, \zeta_f)$ to $(\mu_i, \zeta_{\mu_i})$. |
| `LowestQ()` | VIII E | 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 $\mathcal{D}_{\text{NP}}$. |
| `zetaNP(mu,b,f)` | write-in | NP-model for $\zeta_\mu$. Should follow equipotential. |
| `NPparam` | `TMDR_setNPparameter(input)` | NP parameters used in `DNP` and `zetaNP`. |
| $a_s$ | defined in `QCDinput` | Strong coupling. See sec.II |

### A. Theory

The detailed theory is given in the article [3]. The NLO rapidity anomalous dimension has been evaluated in [7]. The NNLO rapidity anomalous dimension has been evaluated in [8, 9].

TO BE WRITTEN

### B. 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 $\Gamma_{\text{cusp}}$ one order higher then the rest of anomalous dimensions. There are following set of orders

| order | $\Gamma_{\text{cusp}}$ | $\gamma_V$ | $\mathcal{D}$ * | $\mathcal{D}_{\text{resum}}$ | $\zeta_\mu$ ** |
|---|---|---|---|---|---|
| `LO` | $a_s^1$ | $a_s^0$ | $a_s^0$ | $a_s^0$ | $a_s^0$ |
| `LO+` | $a_s^1$ | $a_s^1$ | $a_s^1$ | $a_s^0$ | $a_s^0$ |
| `NLO` | $a_s^2$ | $a_s^1$ | $a_s^1$ | $a_s^1$ | $a_s^1$ |
| `NLO+` | $a_s^2$ | $a_s^2$ | $a_s^2$ | $a_s^1$ | $a_s^1$ |
| `NNLO` | $a_s^3$ | $a_s^2$ | $a_s^2$ | $a_s^2$ | $a_s^2$ |
| `NNLO+` | $a_s^3$ | $a_s^3$ | $a_s^3$ | $a_s^2$ | $a_s^2$ |

* $\mathcal{D}_{\text{resum}}$ starts from $a_s^0$, it already contains $\Gamma_0$.

** Definition of $\zeta_\mu$ 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.

### C. NP input and NP parameters

The NP parameters are used in the definition of the function $\mathcal{D}_{\rm NP}$. 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 $\zeta$-prescription, therefore, the $\zeta_\mu$ line plays essential role. For $\mathcal{D} \neq \mathcal{D}_{\rm NP}$ (which is standard situation), the $\zeta_\mu$ 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 $\zeta_\mu$ 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 $\zeta_\mu$, 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

### D. Evaluating TMD evolution kernel

The evolution kernel are presented in two types for the evolution from arbitrary point to arbitrary, and for th evolution from the arbitrary point to the $\zeta$-line. Since all three types of evolution discussed here has different number of arguments, they could not be confused.

| Function | solution | Evol.type | Comments |
|---|---|---|---|
| Evolution from $(\mu_f, \zeta_f)$ to $(\mu_i, \zeta_i)$ | | | |
| `TMDR_R(b,muf,zetaf,mui,zetai,mu0,f)` | improved $\mathcal{D}$ | 1 | |
| `TMDR_R(b,muf,zetaf,mui,zetai,f)` | improved $\gamma$ | 2 | |
| Evolution from $(\mu_f, \zeta_f)$ to $(\mu_i, \zeta_{\mu_i})$ | | | |
| `TMDR_Rzeta(b,muf,zetaf,mui,mu0,f)` | improved $\mathcal{D}$ | 1 | |
| `TMDR_Rzeta(b,muf,zetaf,mui,f)` | improved $\gamma$ | 2 | |
| `TMDR_Rzeta(b,muf,zetaf,f)` | fixed $\mu$ | 3 | Evolution along $\zeta$. Absolutely fastest. |

where

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

`zetaf,muf` (real*8) hard-factorization scales $(\zeta_f, \mu_f)$ in GeV. Typically, =$(Q^2, Q)$

`zetai,mui` (real*8) low-factorization scales $(\zeta_i, \mu_i)$ in GeV.

`mu0` (real*8) The scale of perturbative definition of rapidity anomalous dimension $\mathcal{D}$ $\mu_0$ 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.

### E. Inverted evolution and the lowest available Q

At small values of parameter $Q = Q_0$ the point $(Q, Q^2)$ crosses the $\zeta$-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 $\sim 1.5$GeV. 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 $c = 2$.

## IX. UTMDPDF MODULE

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

$$F_f(x,b) = \int_x^1 \frac{dz}{z} C_{f\leftarrow f'}(z,b,c_4\mu_{\rm OPE}) f_{f'}(\frac{z}{x}, c_4\mu_{\rm OPE}) f_{NP}^f(x,z,b,\{\lambda\}), \tag{9.1}$$

where $f_f(x,\mu)$ is PDF of flavor $f$, $C$ is the coefficient function in $\zeta$-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 [5] (please, cite it if use).

List of available commands

| Command | Type | Sec. | Short description |
|---|---|---|---|
| `uTMDPDF_Initialize(order)` | subrout. | IX A | Initialization of module. |
| `uTMDPDF_SetLambdaNP(..)` | subrout | IX B | Set new NP parameters used in `FNP` and `mu_OPE`. |
| `uTMDPDF_lowScale5(x,b,h)` | (real*8(-5:5)) | IX C | Returns unpolarized TMD PDF at $x$, $b$ and hadron $h$. Gluon flavour undefined. |
| `uTMDPDF_lowScale50(x,b,h)` | (real*8(-5:5)) | IX 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 | IX D | Force reset or deconstruct the grid. |
| `uTMDPDF_SetPDFreplica(num)` | subrout | – | Call `QCDinput` to change the PDF replica number, deconstructs grid. |

List of inputs

| Input | Setup by | Short description |
|---|---|---|
| `ModelInitialization()` | write-in | Necessary predefinitions by user. E.g. some precalculations for `FNP`. |
| `FNP(x,z,b,hadron)` | write-in | NP-model for $f_{NP}(x,z,b,\{\lambda\})$ depends on the hadron. See sec.IX B |
| `mu_OPE(x,bt)` | write-in | The value of $\mu_{\rm OPE}$. |
| `lambdaNP` | `uTMDPDF_SetLambdaNP(..)` | NP parameters used in `FNP` and `mu_OPE`. |
| $a_s$ | defined in `QCDinput` | Strong coupling. See sec.II |
| $xf(x)$ | defined in `QCDinput` | Unpolarized PDF. See sec.II |

### A. Initialization

Prior the usage module is to be initialized (once per run). By
`call uTMDPDF_Initialize(order)`
here `order` defines the perturbative order of the coefficient function according to the

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

### B. Definition of non-perturbative part, $f_{NP}$ and parameters

The model for TMD is given by $f_{NP}$ (and in smaller amount by $\mu_{OPE}$). In the current version there is no possibility to implement $b*$-prescription, to be added in future. 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 $\lambda$ (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.

- Also user can provide the value of $\mu_{\text{OPE}}$ (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), *however, this option has not been accurately tested yet.*.

- 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((/`$\lambda_1, \lambda_2, ...$`/))`
**Optional:** There exist the overloaded version of `uTMDPDF_SetLambdaNP`, with two additional boolean parameters
`call uTMDPDF_SetLambdaNP((/`$\lambda_1, \lambda_2, ...$`/),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.IX D.

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 $\mu$ and $\zeta$, because the `arTeMiDe` uses the $\zeta$-prescription, where a TMD distribution is scaleless. The scale of matching procedure $\mu_{\text{OPE}}$ is set in the function `mu_OPE` (see previous subsection). Note, that the TMD at different then $\zeta$-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 $\epsilon$ 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. Grid construction

For fitting procedure one often needs to evaluate TMDs multiple times. For example, for fit performed in [1] the evaluation of singe $\chi^2$ entry requires $\sim 16\times 10^6$ calls of `uTMDPDF_...`. Every call of `uTMDPDF_..` at NNLO order, requires $\sim 200$ calls of pdfs, depending on $x$, $b$ and $\lambda$'s. Therefore, in such situations it is much more cheaper to make a grid of TMD distributions for 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 griding turns on by the call overloaded version of `uTMDPDF_SetLambdaNP(lambda,makeGrid,includeGluons)` with `makeGrid=.true.` (see also sec.IX 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 `FNP` onto the x-dependance. If it is x-independent, then the grid (unless it is forcibly reseted) is not renewed but reweighted with new **FNP**. It is possible since in this case `FNP` does not enter the convolution.

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). For $b>b_M$ the extrapolation by the function $\exp(-\alpha(x)-\beta(x)b)$ (or $\exp(-\alpha(x)-\beta(x)b^2)$) is made (with the common sign equal to sign of TMD at $b=b_M$). 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.

The form of the extrapolation (exponential or Gauassian) can be also changed in the file `constants`, sec. 3.D.

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.

### E. Theoretical uncertainties

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

### F. Technical note

The convolution integral reads

$$I(x)=\int_x^1\frac{dz}{z}C(z)f\left(\frac{x}{z}\right)f_{NP}(x,z),\tag{9.2}$$

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

$$C(z)=C_0\delta(1-z)+(C_1(z))_+ + C_2(z).\tag{9.3}$$

Here the plus-distribution is undestood in the usual way

$$(C_1(z))_+ = C_1(z)-\delta(1-z)\int_0^1 dyC_1(y)dy.\tag{9.4}$$

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

$$I(x) = \frac{1}{x}\int_x^1 dz C(z)\tilde{f}\left(\frac{x}{z}\right) f_{NP}(x,z), \qquad \tilde{f}(z) = zf(z). \tag{9.5}$$

Then the integral is split as

$$I(x) = \frac{1}{x}\Big\{I_2(x) + C_0\tilde{f}\,(x)\,f_{NP}(x,1) + \tilde{f}\,(x)\,f_{NP}(x,1)\int_0^x dz C_1(z)\Big\}, \tag{9.6}$$

where

$$I_2(x) = \int_x^1 dz\Big[C_1(z)\left(\tilde{f}\left(\frac{x}{z}\right) f_{NP}(x,z) - \tilde{f}\,(x)\,f_{NP}(x,1)\right) + C_1(z)\tilde{f}\left(\frac{x}{z}\right) f_{NP}(x,z)\Big] \tag{9.7}$$

Presumably, the term $\sim C_0$ give the dominant contribution $w$, since it is $\sim 1$ 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-distinctions, $\delta$, etc, are multiplied by $1/z^2$). In this case it is convenient to rewrite

$$I(x) = \frac{1}{x^3}\int_x^1 dz C(z)\hat{f}\left(\frac{x}{z}\right) f_{NP}(x,z), \qquad \hat{f}(z) = z^3 f(z). \tag{9.8}$$

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

## X. uTMDFF MODULE

The TMDFF functions structurally repeats the TMDPDF functions. Therefore, the module is practically the same as `uTMDFF`. The NNLO coefficient functions used in the module were evaluated in [5, 6] (please, cite it if use).

## XI. SUPPLEMENTARY CODES

In the `arTeMiDe` archive you can find the codes which evaluates the cross-sections using our fit parameters.

### A. Drell-Yan cross-section

The compile the program for the evaluation of Drell-Yan cross-section evaluate: `make dy`. As a result of evaluation in the executable file `arTeMiDe_DY` and the file of input parameters `input` will be placed in `/bin`. The program `arTeMiDe_DY` evaluates the differential cross-section $d\sigma/dQ^2dydq_T^2$ for the production of Z and $\gamma^*$ in the Drell-Yan process (take care that in the current fit we expect agreement with the data for $q_T < 0.2Q$). The program also evaluates various integrals of cross-section such as

$$\int_{Q_1}^{Q_2} dQ^2, \qquad \int_{y_1}^{y_2} dy, \qquad \int_{q_{T1}}^{q_{T2}} \frac{dq_T^2}{q_{T2}-q_{T1}}, \tag{11.1}$$

and their combinations. It also takes (if necessary) into account lepton cuts. Therefore, can be used to build DY cross-section for wide range of experiments. To input all these options check the file `input`, which is self-explanatory.

## XII. 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 | |
| General | | `Initialize(order)` |
| `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)` |
| `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)` |
| `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,iC,cuts` |
| | `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,cuts,m` |

`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] 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]].
[3] I. Scimemi and A. Vladimirov, JHEP **1808** (2018) 003 doi:10.1007/JHEP08(2018)003 [arXiv:1803.11089 [hep-ph]].
[4] A. Buckley, J. Ferrando, S. Lloyd, K. Nordstrm, B. Page, M. Rfenacht, M. Schnherr and G. Watt, Eur. Phys. J. C **75**, 132 (2015) doi:10.1140/epjc/s10052-015-3318-8 [arXiv:1412.7420 [hep-ph]].
[5] M. G. Echevarria, I. Scimemi and A. Vladimirov, JHEP **1609**, 004 (2016) doi:10.1007/JHEP09(2016)004 [arXiv:1604.07869 [hep-ph]].
[6] 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]].
[7] 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]].
[8] A. A. Vladimirov, Phys. Rev. Lett. **118**, no. 6, 062001 (2017) doi:10.1103/PhysRevLett.118.062001 [arXiv:1610.05791 [hep-ph]].
[9] A. Vladimirov, JHEP **1804**, 045 (2018) doi:10.1007/JHEP04(2018)045 [arXiv:1707.07606 [hep-ph]].
[10] 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]].
[11] 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].

## XIII. VERSION HISTORY

**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.IX F. 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 $\lambda_{NP}$
- **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 $(x, z, b, h, \lambda)$. 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, $\Gamma_3$ 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 $\mu$, from `uTMDPDF_....(x,b)`. Added function `mu_OPE(b)`, which is used as $\mu$-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.

## XIV. BACKUP

### TO DO LIST

- Add possibility for non-perturbative definition of $\zeta$-line.
- Update QCDinput file to eat input from constants.

### A. arTeMiDe structure before v1.3