arTeMiDe ver.1

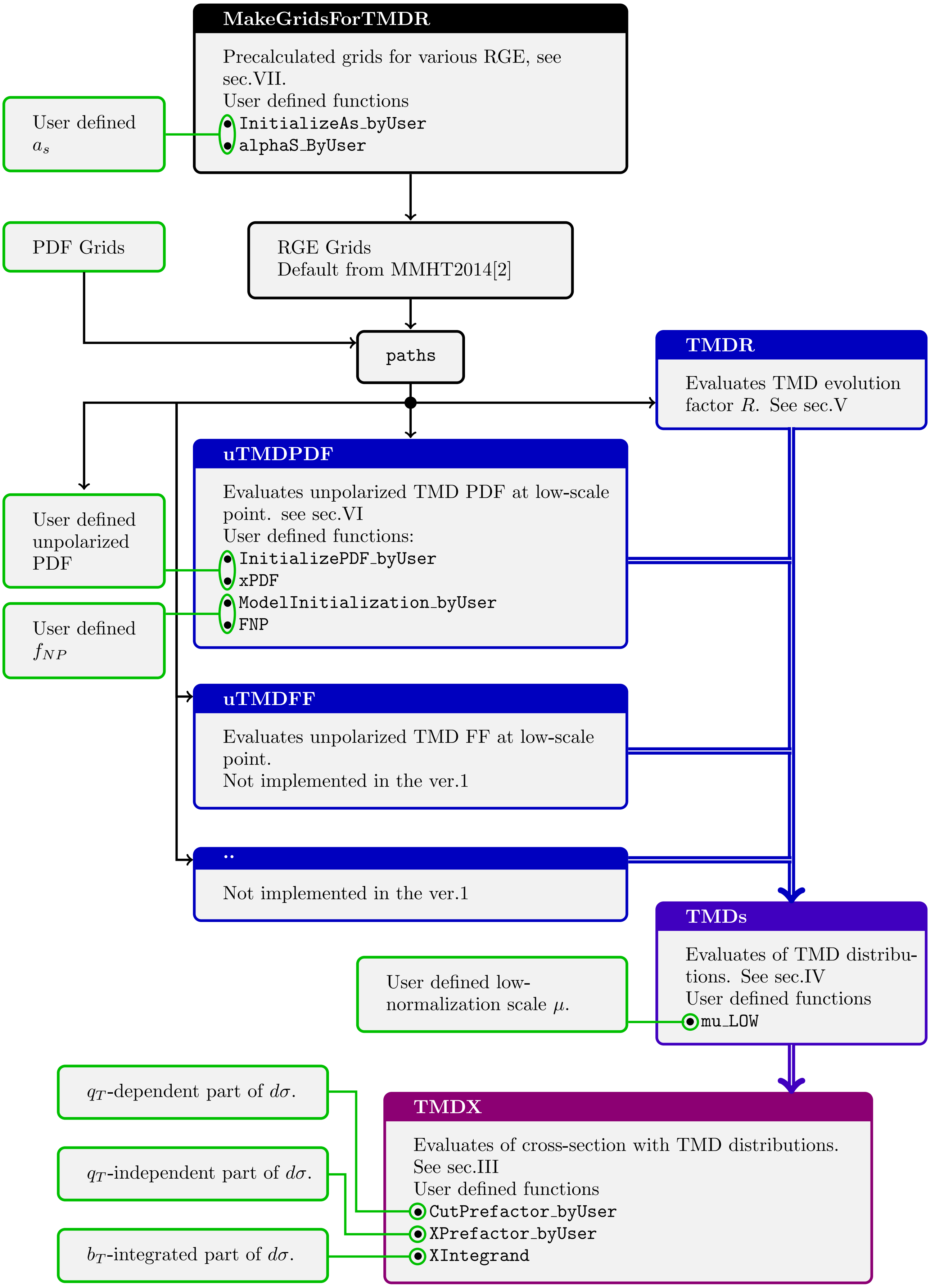
Alexey A. Vladimirov
June 7, 2017

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

I. VERSION HISTORY

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

II. GENERAL STRUCTURE AND USER INPUT OF ARTEMIDE


MakeGridsForTMDR
Precalculated grids for various RGE, see sec.VII.
User defined functions
InitializeAs_byUser
alphaS_ByUser
User defined a_s
PDF Grids
RGE Grids
Default from MMHT2014[2]
paths
TMDR
Evaluates TMD evolution factor R. See sec.V
uTMDPDF
Evaluates unpolarized TMD PDF at low-scale point. see sec.VI
User defined functions:
InitializePDF_byUser
xPDF
ModelInitialization_byUser
FNP
User defined unpolarized PDF
User defined f_{NP}
uTMDFF
Evaluates unpolarized TMD FF at low-scale point.
Not implemented in the ver.1
..
Not implemented in the ver.1
TMDs
Evaluates of TMD distributions. See sec.IV
User defined functions
mu_LOW
User defined low-normalization scale μ.
q_T-dependent part of $d\sigma$.
q_T-independent part of $d\sigma$.
b_T-integrated part of $d\sigma$.
TMDX
Evaluates of cross-section with TMD distributions. See sec.III
User defined functions
CutPrefactor_byUser
XPrefactor_byUser
XIntegrand

A. Functions to check and update

1. Build the grids for coupling constants, if necessary. See section VII. The default ones are evaluated from MMHT2014 [2].
2. Prepare the PDF initialization and reading PDF routine in the end of `uTMDPDF.f90` code. The default one uses MMHT2014 main set[2].
3. Update the file `paths` with paths to new grids, with accordance to the used order.
4. Set the non-pertrubative function f_{NP} and its initialization in the end of the file `uTMDPDF.f90`.
5. Set the value of low-normalization scale $\mu_i(b)$ in the end of the file `TMDs.f90`.

III. TMDX MODULE

Warning: In the current version `arTeMiDe` evaluates only the unpolarized Drell-Yan-like processes. Therefore, the user interface is somewhat restricted to it. For SIDIS and other processes, as well as, on the polarized processes, it will be updated with inclusion of `uTMDFF` module.

The module `TMDX` joins the lower modules and performs the evaluation of the cross-section. In the current version (version 1) it is restricted to the following form

$$d\sigma(q_T) = prefactor \ \times \ cuts \ \times \int_0^\infty \frac{bdb}{2} J_0(bq_T)X(b), \tag{3.1}$$

where $prefactor$ any q_T independent factor, *cuts* is any q_T dependent factor (typically it includes cuts), $X(b)$ is the expression to integrate. Example, for the Drell-Yan process one has for $d\sigma/dq_T$

$$\begin{aligned} prefactor &= \frac{4\pi}{9sQ^2}|C_V(Q,\mu_H)|^2 \\ cuts &= 1 + \frac{q_T^2}{2Q^2} \\ X(b) &= \sum_f |e_f|^2 F_f(x_A, b; \mu_H, \zeta_A) F_{\bar f}(x_B, b; \mu_H, \zeta_B). \end{aligned}$$

The expressions for these factors can be setup by user within the functions `Xprefactor_byUser`, `cutPrefactor_byUser`, `XIntegrand`, which are located in the end of the code `TMDX.f90`. Several examples of these factors can be found in the code.

In the definition of these function one can use the following variables and functions (list to be updated)

`s_global` the variable s set by `TMDX_Xsetup`

`Q_global` the variable Q set by `TMDX_Xsetup`

`y_global` the variable y set by `TMDX_Xsetup`

`HardCoefficientDY()` the function which evaluate $|C_V|^2$ for the Drell-Yan process.

`xA_global` $= Qe^y/\sqrt{s}$

`xB_global` $= Qe^{-y}/\sqrt{s}$

- `muHard_global` $= Q$
- `zetaA_global` $= Q^2$
- `zetaB_global` $= Q^2$

The variables marked by • are the part of the hard-factorization scale definition. They should be used as hard-factorization scales of TMDs. This variables also used to define the theoretical errors.

A. Initialization

Prior the usage module is to be initialized (once per run) by
`call TMDX_Initialize(orderTMD,orderH)`
here:

`orderTMD` declaration of order a unpolarized TMDPDF. It can be `'NLL'`, `'NLO'`, `'NNLL'` or `'NNLO'`. This is a complex declaration, which implies particular orders for coefficient functions, PDFs, anomalous dimension, etc. For detailed definition see [1].

`orderH` declaration of the order of hard-coefficient to be used.

B. Setting up the cross-section

Prior to evaluation of cross-section set up the kinematics and the definition of the cross-section, by
`call TMDX_XSetup(s,Q,y,process)`
where `s` is the Mandelshtam variable s, `Q` is hard virtuality Q, `y` is the rapidity parameter y and `process` is the number of process to be used (it is used in the functions `Xprefactor_byUser`, and `XIntegrand` only).

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_SetNPParameters(bmax,gK,lambda)`
where `bmax` and `gK` are real*8, `lambda` is real*8(1:number of parameters). For details see IV B.

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 CalculateXsection(X,qt)`
where `X` is real*8(1,N) variable where cross-section will be stored, `qt` is real*8(1,N) is the list of values of q_T's at which the `X` is to be calculated.

This command has following extensions

`CalculateXsection_Yint(X,qt,yMin,yMax)` $= \int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$ here `yMin,yMax` are real*8.

`CalculateXsection_YintComplete(X,qt)` $= \int_{-y_0}^{y_0} dy d\sigma(q_T)$ where y_0 is maximum allowed y, i.e. $y_0 = \ln(s/Q^2)/2$.

`CalculateXsection_Qint_Yint(X,qt,QMin,QMax,yMin,yMax)` $= \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$ here `QMin,QMax` are real*8.

`CalculateXsection_PTint_Yint(X,qtMin,qtMax,yMin,yMax)`$=$ $\int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{y_{\min}}^{y_{\max}} dy d\sigma(q_T)$ here `qtMin,qtMax` are real*8(1:N)

`CalculateXsection_PTint_YintComplete(X,qtMin,qtMax)` $= \int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{-y_0}^{y_0} dy d\sigma(q_T)$

`CalculateXsection_PTint_Qint_Yint(X,qtMin,qtMax,QMin,QMax,yMin,yMax)` $= \int_{q_{T\min}}^{q_{T\max}} 2q_T dq_T \int_{Q_{\min}}^{Q_{\max}} 2QdQ \int_{-y_0}^{y_0} dy d\sigma(q_T)$.

Take care that every next function is heavier to evaluate then the previous one.

Important: The evaluation of integrand over cross-sections is made by Simpson method with fixed numbers of sections. This is done for speeding up the calculation. It is expected that the function (i.e. cross-section) does not dramatically changed within the range of integration. These procedures will be updated in the future versions. Currently, do not use the too wide range of integrations!

Note: If the quadrature does not converge after 64 Ogata nodes, which is a lot, it implies some blowing up integrand. It can happen if e.g. TMD distribution rise at $b \to \infty$ (if e.g. $g_K < 0$, which should not happen). In this case, some generic very large number will be returned. It will be returned for any further evaluations of cross-sections, until `TMDX_SetNPParameters` called.

D. Theoretical uncertainties

There are three perturbative scales in the definition of the cross-section and TMD distributions, $c_{1,2,3}$. For precise definition check [1]. The theoretical uncertainty resulted from the truncation of the pertrubative series is estimated by variation of $c_{1,2,3} \in [0.5, 2]$.

The subroutines `CalculateXsection` have overloaded versions with two extra variables following the cross-section variable `X`

`CalculateXsection_...(X,Xmin,Xmax,qt,...)` where `Xmin(Xmax)` is real*8(1,N) which filled by the lowest (highest) value of cross-section obtained from the variation of constants $c_{1,2,3}$.

Note, that during evaluation the overloaded function calculates cross-section 7 times, and therefore, 7 times slower then the initial version.

E. Example

```
program example
!to include the module
use TMDX
use LeptonCutDY
implicit none
real*8,dimension(1:4)::xSec,pt_list,xSecMin,xSecMax
pt_list=(/5d0,10d0,15d0,20d0/)
!initialize all at NNLO
call TMDX_Initialize('NNLO','NNLO')
! state b_max = 1, g_k = 0, λ_1 = 0.18 and λ_2 = 0.0024.
call TMDX_SetNPParameters(1d0,0d0,(/0.180d0,0.0024d0/))
! let Q = 91GeV and √s = 8TeV, mid-rapidity, process=2.
call TMDX_XSetUp(8000d0**2,91d0,0d0,2)
! ATLAS cuts: p_T > 20, −2.4 < η < 2.4
call SetCuts(.true.,20d0,-2.4d0,2.4d0)
! Evaluate cross-section
call CalculateXsection(xSec,pt_list)
write(*,*) xSec
! Evaluate cross-section and theoretical variations
call CalculateXsection(xSec,xSecMin,xSecMax,pt_list)
end program example
```

Compile e.g. by line

```
f95 src/*.f*
```

IV. TMDS MODULE

The module `TMDs` joins the lower modules and performs the evaluation of various TMD distributions in the ζ-prescription (in version 1 only unpolarized TMDPDF is accessible). Generally a TMD distribution is given by the expression

$$F_f(x,b;\mu,\zeta) = R_f[b,(\mu,\zeta) \to (\mu_i,\zeta_{\mu_i})]\tilde{F}_f(x,b;\mu_i,f_{NP}), \tag{4.1}$$

where R is the TMD evolution kernel, $\tilde{F}$ is a TMD distribution at low scale. It depends on f_{NP} which is defined by user for each TMD distribution, see secVI C. It also uses the external PDFs which should be provided by user, see VI B. Note, that `TMDs` initializes the lower modules automatically. Therefore, no special initializations should be done.

Note: this module (and consequently the manual section) will be seriously in the next versions.

A. Initialization

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

`order` declaration of order a unpolarized TMDPDF. It can be `'NLL'`, `'NLO'`, `'NNLL'` or `'NNLO'`. This is a complex declaration, which implies particular orders for coefficient functions, PDFs, anomalous dimension, etc. For detailed definition see [1].

B. Definition of non-pertrubative parameters, and low-scale

The low scale μ_i is stated in the function `mu_LOW(bt)` which can be found in the end of `TMDs.f90` code. Modify it if needed.

There are multiple possibilities to include the non-perturbative parameters. The parameters are distributed between different modules. In the current version (ver 1.0, which includes only uTMDPDF module), there are following non-parameters

$b_{\max}$ The parameter defines the low scale μ_i.

g_K The parameter parametrizes the non-perturbative part of TMD evolution.

$\{\lambda_i\}$ The set of parameters which define the non-pertrubative function f_{NP}, for unpolarized TMD PDF.

To set particular values of these parameters use
`call TMDs_SetNPParameters(bmax,gK,lambda)`
where `bmax` and `gK` are real*8, `lambda` is real*8(1:number of parameters).

C. Evaluating unpolarized TMD PDFs

The expression for unpolarized TMD PDF is obtained by the functions
`uTMDPDF_3(x,b,mu,zeta)`
where

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

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

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

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

This function return the vector real*8(-3:3) for $\bar{s}, \bar{u}, \bar{d}, ?, d, u, s$. Gluon contribution is undefined, but taken into account in the mixing contribution. The mixing with c and b quarks is not included.

The following additional functions evaluate the TMD PDFs of different flavours simultaneously.

`uTMDPDF_30(x,b,mu,zeta)` returns (real*8) array(-3:3) for $\bar{s}, \bar{u}, \bar{d}, g, d, u, s$. The mixing with c and b quarks is not included. It is a bit ($\sim 5-15\%$ depending on order) slower then the previous command. If gluons are not needed use previous.

`uTMDPDF_5(x,b,mu,zeta)` 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_50(x,b,mu,zeta)` returns (real*8) array(-5:5) for $\bar{b}, \bar{c}, \bar{s}, \bar{u}, \bar{d}, g, d, u, s, c, b$. It is a bit ($\sim 5-15\%$ depending on order) slower then the previous command. If gluons are not needed use previous.

D. Example

```
program example
!to include the module
use TMDs
implicit none
!initialize all at NNLO
call TMDs_Initialize('NNLO')
! state b_max = 1, g_k = 0, λ_1 = 0.18 and λ_2 = 0.0024.
call TMDs_SetNPParameters(1d0,0d0,(/0.180d0,0.0024d0/))
!obtain TMD PDF for all flavours at x = 0.1 b = 1GeV, μ = 91GeV and ζ = 91^2GeV^2
write(*,*) uTMDPDF_50(0.1d0,1d0,91d0,91d0**2)
end program example
```

Compile e.g. by line

```
f95 src/*.f*
```

V. TMDR MODULE

The module `TMDR` performs the evaluation of the TMD evolution kernel in the (μ,ζ)-plane from the point (μ_f,ζ_f) to (μ_i,ζ_i). It is given by the following expression

$$R_f[\boldsymbol{b};(\mu_f,\zeta_f)\to(\mu_i,\zeta_i);\mu_0]=$$
$$\exp\left[\int_{\mu_i}^{\mu_f}\frac{d\mu}{\mu}\gamma_F(\mu,\zeta_f)-\int_{\mu_0}^{\mu_i}\frac{d\mu}{\mu}\Gamma(\mu)\ln\left(\frac{\zeta_f}{\zeta_i}\right)\right]\left(\frac{\zeta_f}{\zeta_i}\right)^{-\mathcal{D}^f_{\text{perp}}(\mu_0,\boldsymbol{b})-g_K\boldsymbol{b}^2}. \tag{5.1}$$

where f is the flavour of parton, γ_F is the ultraviolet TMD anomalous dimension and $\mathcal{D}$ is the rapidity anomalous dimension. The following entries defines this expression

- The coefficient functions can be of NLO, NNLO,NNNLO.
- The order of coupling constant a_s used in the integrals is defined by user.
- By definition $g_K=0$, but can be switched on.

A. Initialization

Prior the usage module is to be initialized (once per run). By
`call TMDR_Initialize(oA,oCUSP,oAD,oZETA)`
here:

`oA` declaration of order for a_s. It can be `'LO'`, `'NLO'` or `'NNLO'`. It affects the grid for a_s which would be used. The path to grid is set in the file `paths`.

`oCUSP` declaration of order for cusp-anomalous dimension (typically one order higher then the rest anomalous dimensions). It can be `'NLO'`, `'NNLO'` or `'NNNLO'`.

`oAD` declaration of order for (non-cusp) part of γ_F and $\mathbf{D}_{\text{pert}}$. It can be `'NLO'`, `'NNLO'` or `'NNNLO'`.

`oZETA` declaration of order for ζ_μ, which is used in ζ-prescription. It can be `'NLL'`, `'NLO'`, `'NNLO'` or `'NNLL'` (see definition in [1]).

B. Definition of g_K

The value of the non-pertrubative parameter g_K is 0 default. It can be changed by
`call TMDR_SetgK(gK)`

C. Evaluating TMD evolution kernel

The expression for TMD evolution kernel is given by the function
`TMDR_R(b,zetaf,muf,zetai,mui,mu0,f)`
where

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

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

`zetai,mui` (real*8) low-factorization scales (ζ_i,μ_i) in GeV.

`mu0` (real*8) The scale of perturbative definition of rapidity anomalous dimension $\mathcal{D}$ μ_0 in GeV.

`f` (integer) parton flavor. 0 for gluon, $\neq 0$ for quarks.

Typically, one does not need the function R in the full glory. Therefore, this function has following derivatives, which are optimized with accordance to set of scales.

`TMDR_full_zetaP(bT,zetaf,muf,mui,mu0,f)` same as `TMD_R` but with $\zeta_i=\zeta_\mu$.

`TMDR_zetaP(bT,zetaf,muf,mui,f)` same as `TMD_R` but with $\zeta_i=\zeta_\mu$ and $\mu_0=\mu_i$. This is the most often case.

D. Example

```
program example
!to include the module
use TMDR
implicit none
!initialize all at NNLO, $\Gamma_{cusp}$ at NNNLO
call TMDR_Initialize('NNLO','NNNLO','NNLO','NNLO')
!state $g_K = 0.01$
call TMDR_SetgK(0.001d0)
!obtain TMD evolution kernel for quark in $\zeta$-prescription from $(\mu_f, \zeta_f) = (91, 91^2)$ to $\mu_i = 5$GeV, at $b = 0.1$GeV
write(*,*) TMDR_zetaP(0.1d0,91d0**2,91d0,5d0,1)
end program example
```

Compile e.g. by line (if compile separately take care to include the module `readAs.f90`)

```
f95 src/TMDR.f90
```

VI. `uTMDPDF` MODULE

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

$$F_f(x,b;\mu_i)=\int_x^1 \frac{dz}{z}C_{f\leftarrow f'}(z,b,\mu_i)f_{f'}(\frac{z}{x},\mu_i)f_{NP}(z,b,\{\lambda\}), \tag{6.1}$$

where $f_f(x,\mu)$ is PDF of flavor f, C is the coefficient function in ζ-prescription, f_{NP} is the non-perturbative function. The following entries defines this expression

- The coefficient function can be of LO, NLO, NNLO.
- The order of coupling constant a_s and PDF $f(x)$ are defined by used they can be LO, NLO, NNLO.
- Optionally, the renormalon contribution can be added. In this case the coefficient function takes the form

$$C_{f\leftarrow f'}(z,b,\mu_i)\to C_{f\leftarrow f'}(z,b,\mu_i)+\lambda_2 b^2\delta_{ff'}C^{\text{ren}}_{f\leftarrow q}(z,b,\mu_i). \tag{6.2}$$

A. Initialization

Prior the usage module is to be initialized (once per run). By
`call uTMDPDF_Initialize(oC,oA,oPDF)`
here:

`oC` declaration of order for coefficient function. It can be `'LO'`, `'NLO'` or `'NNLO'`.

`oA` declaration of order for a_s. It can be `'LO'`, `'NLO'` or `'NNLO'`. It affects the grid for a_s which would be used. The path to grid is set in the file `paths`.

`oPDF` declaration of order for PDF. It can be `'LO'`, `'NLO'` or `'NNLO'`. It affects the grid PDF which would be used. The path to grid is set in the file `paths`. The initialization for PDF set is made in the subroutine `InitializePDF_byUser` in the end of the module. Update this subroutine if needed.

B. Definition of PDFs

The PDFs are defined by user. The initialization routine `InitializePDF_byUser` is called during the initialization stage. It refers to the grid file under the path defined in `paths`-file. The function for evaluation of a PDF `xPDF` is given in the end of module. Modify it if needed.

C. Definition of non-pertrubative part, f_{NP} and parameters

To modify the definition of the non-perturbative part two function should be modified. Both located in the end of the file `uTMDPDF.f90`.

The first is the subroutine `ModelInitialization_byUser()`. It contains definition of constants `includeRenomalon` and `LambdaNPLength`.

`includeRenomalon` (logical) statement to include the renormalon correction into coefficient function. It is `.true.` or `.false.`. If renormalon correction is included the constant λ_2 would be used as prefactor for it. See eqn.(6.2).

`LambdaNPLength` (integer) number of non-perturbative constants λ to be used.

The second, user should provide the definition of the non-pertrubative function in `FNP`. It uses the parameters $\lambda_{1,2,...}$ which are given by (real*8(1:`LambdaNPLength`)) variable `lambdaNP`.

To set the values for array `lambdaNP` use
`call uTMDPDF_SetLambdaNP((/`λ_1,λ_2,...`/))`
Note, that the constant λ_2 is reserved for the renormalon-contribution prefactor (if included).

D. Evaluating unpolarized TMD PDFs

The expression for unpolarized TMD PDF is given by the function
`uTMDPDF_lowScale(f,x,b,mu)`
where

`f` (integer) parton flavor. 0 for gluon, i for quarks ({1,2,3,4,5}={d,u,s,c,b}), $-i$ for anti-quarks.

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

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

`mu` (real*8) The scale μ_i in GeV. It should be bigger then 0.5GeV. we suggest to take it larger then 1 GeV, since the variation of scales can occasionally make it smaller then 0.5 GeV.

Warning: we do not recommend use this function, because it is slower and numerically less accurate then the following functions.

The following additional functions evaluate the TMD PDFs of different flavours simultaneously.

`uTMDPDF_lowScale3(x,b,mu)` returns (real*8) array(-3:3) for $\bar{s}, \bar{u}, \bar{d}, ?, d, u, s$. Gluon contribution is undefined, but taken into account in the mixing contribution. The mixing with c and b quarks is not included.

`uTMDPDF_lowScale30(x,b,mu)` returns (real*8) array(-3:3) for $\bar{s}, \bar{u}, \bar{d}, g, d, u, s$. The mixing with c and b quarks is not included. It is a bit ($\sim 5 - 15\%$ depending on order) slower then the previous command. If gluons are not needed use previous.

`uTMDPDF_lowScale5(x,b,mu)` 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,mu)` returns (real*8) array(-5:5) for $\bar{b}, \bar{c}, \bar{s}, \bar{u}, \bar{d}, g, d, u, s, c, b$. It is a bit ($\sim 5 - 15\%$ depending on order) slower then the previous command. If gluons are not needed use previous.

These commands have essentially less number of calls for `xPDF` and, therefore, significantly, faster then `uTMDPDF_lowScale`. We recommend to use these functions.

E. Example

```
program example
!to include the module
use uTMDPDF
implicit none
!initialize all at NNLO
call uTMDPDF_Initialize('NNLO','NNLO','NNLO')
!state λ1 = 0.18 and λ2 = 0.0024.
!It is assumed includeRenomalon=.true. and lambdaNPLenght=2
call uTMDPDF_SetLambdaNP((/0.180d0,0.0024d0/))
!obtain TMD PDF for all flavours at x = 0.1 b = 1GeV and μi = 5GeV
write(*,*) uTMDPDF_lowScale50(0.1d0,1d0,5d0)
end program example
```

Compile e.g. by line (if compile separately take care to include the module `readAs.f90`)
`f95 src/*.f*`

VII. PREPARING GRIDS FOR `TMDR`

To improve the evaluation timing `TMDR` uses the precalculated grids of the renormalization group integrals such as

$$\int_{\mu_r}^{\mu} \frac{d\mu'}{\mu'} a_s^n(\mu'), \qquad \int_{\mu_r}^{\mu} \frac{d\mu'}{\mu'} a_s^n(\mu') \ln(\mu), \qquad \int_{\mu_r}^{\mu} \frac{d\mu'}{\mu'} \Gamma_{cusp}(a_s) \ln \mu', \qquad \text{etc.} \tag{7.1}$$

Here, $\mu_r = 1$GeV is the common reference point. Grids can be prepared by evaluation of the program `MakeGridsForTMDR.f90` in the directory `/Precalculate` (note, that default version already contains necessary grids made on the MMHT2014 values of a_s).

As an input for the program user should prepare functions `InitializeAs_byUser` and `alphaS_ByUser`. Both are located in the module `UserDefinedAlphaS.f90`. The subroutine `InitializeAs_byUser` runs initialization routine, if needed. The function `alphaS_ByUser` gives access to the values of $\alpha_s(\mu) = g^2(\mu)/(4\pi)$.

As a result of evaluation program create the file `As_grid.dat`. Rename it conveniently at put to the directory `/Grids`. Update the file `paths` with new names.

[1] I. Scimemi and A. Vladimirov, 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]].