Readme for LePDF code

Francesco Garosi, David Marzocca, Sokratis Trifinopoulos

1 Introduction

Here you can find useful information to understand the structure of the code and how to run it. We refer to our paper "LePDF: Standard Model PDFs for High-Energy Lepton Colliders", arXiv:2303.16964, where you can find the detailed explanation of what the code does, here we recall some of the main points and describe how the code works. For practical reasons the notation used in the code to label some quantities may vary w.r.t. the paper, as explained in these notes.

2 The Runge-Kutta algorithm

Given the differential equation $y'(t) = F(t, y)$ we can solve it numerically with a 4^{th}-order Runge-Kutta algorithm as follows:

$$\begin{aligned} k_1(t) &= dtF(t, y(t)) \\ k_2(t) &= dtF(t + \frac{dt}{2}, y(t) + \frac{k_1(t)}{2}) \\ k_3(t) &= dtF(t + \frac{dt}{2}, y(t) + \frac{k_2(t)}{2}) \\ k_4(t) &= dtF(t + dt, y(t) + k_3(t)) \\ y(t + dt) &= y(t) + \frac{k_1(t)}{6} + \frac{k_2(t)}{3} + \frac{k_3(t)}{3} + \frac{k_4(t)}{6} + \mathcal{O}(dt^5) \ . \end{aligned} \tag{1}$$

This means that starting with the initial condition $y(t_0)$ we can obtain the solution at any t through the proper number of steps in dt.
In the code we solve the DGLAP equations

$$\frac{df_B(x,t)}{dt} = P_B^v(t)\, f_B(x,t) + \sum_{A,C} \frac{\alpha_{ABC}}{2\pi} \widetilde{P}_{BA}^C(t) \otimes f_A + \frac{v^2}{16\pi^2 m_0^2 e^t} \sum_{A,C} \widetilde{U}_{BA}^C(t) \otimes f_A(t) \ , \tag{2}$$

where the indices A, B, C label the N partons we consider. We discretize the x-space with a grid of N_x points[1] x_j, $j = \{0, \ldots, N_x - 1\}$, and we get a set

[1] Notice that in the paper we call N_x the index of the last point of the grid, counting the points from x_0 to $x_{N_x} = 1$. In other words, the value of N_x in the paper is the one in the code minus 1.

of $N \times N_x$ differential equations for the variables $f_{Bj}(t) \equiv f_B(x_j, t)$,

$$\begin{aligned}
&\frac{df_{Bj}(t)}{dt} = P_B^v(t) f_{Bj}(t) \\
&+ (\log(1-x_j) - X_j) \sum_{A,C} \left[\frac{\alpha_{ABC}(t)}{2\pi} \widetilde{D}_{BA}^C(1) + \frac{v^2}{16\pi^2 m_0^2 e^t} \widetilde{V}_{BA}^C(1) \right] f_{Aj}(t) \\
&+ \sum_{k=j+1}^{N_x-1} \frac{\delta x_k}{x_k} \sum_{A,C} \left[\frac{\alpha_{ABC}(t)}{2\pi} \widetilde{P}_{BA}^C \left(\frac{x_j}{x_k} \right) + \frac{v^2}{16\pi^2 m_0^2 e^t} \widetilde{U}_{BA}^C(t) \left(\frac{x_j}{x_k} \right) \right] f_{Ak}(t) \ ,
\end{aligned} \tag{3}$$

where D and V are the splitting functions without the $1-x$ at the denominator:

$$\widetilde{P}_{BA}^C(x) = \frac{\widetilde{D}_{BA}^C(x)}{(1-x)_+} \ , \qquad \widetilde{U}_{BA}^C(x) = \frac{\widetilde{V}_{BA}^C(x)}{(1-x)_+} \ , \tag{4}$$

and X_j is given by

$$X_j \equiv x_j \sum_{k=j+1}^{N_x-1} \frac{\delta x_k}{x_k^2} \frac{1}{1 - \frac{x_j}{x_k}} \ . \tag{5}$$

The Runge-Kutta coefficients are then computed for all the $N \times N_x$ variables $f_{B,j}$ using Eq. (1), with the right hand side of Eq. (3) as the function F. In the following, we will call j-terms the first two lines of Eq. (3) and k-terms the last one.

2.1 Implementation of the algorithm

The computation of the Runge-Kutta coefficients is done with three different functions (see the next section for details), **Rj** and **Rjuc** for the j terms (for massless and ultra-collinear contributions respectively) and **RK** for the k terms. The coefficients are collected in four $(N_t - 1) \times N_x N$ matrices, each row corresponding to the variables $f_{A\alpha}(t)$. For instance in the second phase we start with e_L, so the first N_x entries of a row look like

$$f_{e_L,x_0} \quad f_{e_L,x_1} \quad \dots \quad f_{e_L,1}. \tag{6}$$

Then we move to the second parton and so on until the last one.

3 File list

The code for the computation of LePDF consists of the following files:

- **Main.cpp**: it is the file to be run, containing the calculations.
- **Matrix-alloc.cpp**: to allocate and free $n \times m$ matrices. When allocated, all the entries of the matrix are set to zero.

- **Couplings-RG.cpp**: it contains the one-loop running coupling of α_γ (for the QED+QCD phase), α_1 and α_2 (for the SM phase). α_s instead is obtained through a linear interpolation starting from a matrix with the scale t in the first column and the corresponding value of the coupling in the second one. The file ends with functions calculating the running of other couplings starting from α_1, α_2 and α_s.

- **Cuts.cpp**: it contains the function to impose the cut z_{MAX} [2] and the definition of the $\theta-$function, which is used to insert the mass thresholds.

- **EVA.cpp**: here we write the formulas for the PDFs of W^- and Z in Effective Vector Approximation, which are used at $t = t_0$ to match between the two phases.

- **Prop-corr.cpp**: contains the mass corrections to the splitting functions coming from the propagators, including the one for the mixed PDFs.

- **Massless-splitting.cpp** and **Uc-splitting.cpp**: they collect all the splitting functions.

- **QED-QCD.cpp**: it contains the functions to implement the Runge-Kutta algorithm for the first phase of the evolution. Here you can find the formulas for the coefficients of the algorithm, the implementation of the step and the imposition of momentum conservation.

- **Massless-DGLAP.cpp**: it contains the recurring massless splittings contributing to the DGLAP equations.

- **Massless-eqs.cpp**: using the splittings in "Massless-DGLAP.cpp", it contains the expressions for the k massless contributions to the evolution coefficients for the SM degrees of freedom.

- **Uc-eqs.cpp**: same as the one above, but for ultra-collinear contributions.

- **RK-sum.cpp**: here is reported the function RK, which is used to update the matrices of the evolution coefficients adding both the massless and ultra-collinear k contributions (see the section "Main" for its usage).

- **Massless-virt.cpp** and **Uc-virtual.cpp**: they contain the contributions to the virtual coefficients P_B^v from massless and ultra-collinear terms respectively.

[2] In the code we use the DGLAP equations with the splitting functions computed in x/z and the PDFs in z, while is the opposite in the paper. In this way the upper bound z_{MAX} becomes a lower bound, which is the one implemented in the function "cut".

- **RK-virtual.cpp** and **RK-virtual-uc.cpp**: as in "RK-sum.cpp", here are reported the functions to update the matrices with the evolution coefficients adding the j contribution (virtual emissions and terms from the + distribution) from massless and ultra-collinear terms respectively. The former contains also the imposition of momentum conservation.

- **Writing.cpp**: it contains the functions used to write the results in LHAPDF format, both for the particle and the antiparticle.

4 Structure of "Main.cpp"

"Main.cpp" is structured as follows:

- In order to be used by all the files in the code, we declare as global variables all the constants we need: charges, group factors, masses, energy scales, beta function coefficients, initial conditions for running couplings and discretization constants such as the number of points in t and x. The constants depending on the choice of the valence lepton and of the flavour scheme are set through functions.

- We allocate the matrices containing the PDFs, for both phases, with the rows corresponding to the energy scale t and the columns to the point of the x-grid. In particular there are 7 $N_t^0 \times N_x$ matrices for the first phase and 42 $N_t \times N_x$ for the second one: in the former case the first row contains the initial conditions at $t = 0$ and in the last one are reported the PDFs at $t = t_0$. We then use the last row to generate the initial conditions for the second phase, reported in the first row of the other 42 matrices. We then allocate the matrices for the evolution coefficients.

- The output files are opened with name "LePDF l nFS 0000.dat" (l = e, mu is the valence lepton and $n = 5, 6$ is the flavour scheme). The code compiles the name automatically depending on the choice of the valence lepton and of the flavour scheme.

- α_s is imported from the file "alphas NNLO.dat" and interpolated.

- The grid in x is generated through the formula reported in our paper (but it can also be loaded from file) and the grid spacings are computed.

- We perform the first phase of the evolution, for QED+QCD, with three for-cycles: the first one spans the scale t through $i = 0, \ldots, N_t^0 - 1$ and for each i we compute the $(j + nN_x)^{\text{th}}$ entries of the matrices ($j = 0, \ldots, N_x - 1$) and $n = 0, \ldots, N^0$) with the functions RKj0, and RK0 (no ultra-collinear contributions in this phase). The latter, giving the k terms, is summed over k with the last cycle (for $k > j$). The

PDFs at the scale $i+1$ are then computed summing the coefficients according to Eq. (1). We do not consider $j = N_x$, since at $x = 1$ we use momentum conservation, which is imposed through the function $Lt0$.

- After the first phase we match the 7 PDFs of QED+QCD to the full set of 42 variables of the second one, including the EVA, and we repeat the three for-cycles to perform the Runge-Kutta.

- The PDFs are saved in LHAPDF format. The code generates two files, one with the PDFs for the lepton (electron or muon) and one with the PDFs for the antiparticle, obtained through CP conjugation. Notice that the output is different for the two flavour schemes, since in the 5FS the top is absent and we do not report it.

5 The choice of the parameters

Most of the parameters used in the code are fixed and must be left unchanged. Here you can find a list of what you can modify.

- Choice of the valence lepton, through the integer variable **lepton**: 0 corresponds to the electron, any other value to the muon. The code is exactly the same, with a different value of the valence mass m_0 (set automatically by the function "leptonmass"), since the equations are the same in both cases. We use the same notation for both cases, labelling the valence lepton as the muon: then, when we insert the results in the output files, we do it accordingly to what the valence lepton actually is.[3]

- Choice of the flavour scheme, through the integer variable **FS**: 5 means 5FS (without the top quark), while any other integer corresponds to the 6FS. Again the code is the same and the 5FS is obtained setting the top threshold tt to a value above the maximum t reached during the evolution: this is done by the function "ttchoice".

- α_s: the strong coupling is loaded from file, which can be changed if you want to use a different order approximation (you can even insert an analytic formula if you want, without using a file with points, but in this case the code must be modified accordingly). In this case, modify the variables **Na** and **dta**, corresponding to the number of points in the file and their spacing in t.

- QCD starts contributing at **tQCD**. In our implementation it corresponds to 0.7 GeV, but it can be changed.

[3]For instance, fml is written in the column corresponding to e_L if lepton = 0 and in the one corresponding to μ_L otherwise.

- Grid in x: you can use the grid you want, by changing the formula which generates it (function **xgrid**) or loading it from a file. Be careful to change the number of points **Nx** accordingly. In our computation $Nx = 1001$.

- Evolution steps of the first phase: we compute the PDFs at **Nt0** values of t, including $t = 0$, which means that we perform $\mathrm{Nt0} - 1$ steps of the algorithm. We use a uniform spacing $\mathrm{dt0} = \mathrm{t0}/(\mathrm{Nt0} - 1)$. Nt0 can be changed through the function "Nt0choice": we use $\mathrm{Nt0} = 351$ for the electron and $\mathrm{Nt0} = 201$ for the muon, in order to have a similar dt0 in both cases (since t0 is different in the two cases).

- The t step can be modified also in the second phase, choosing the number of values at which we compute the PDFs, **Nt** (set with the function "Ntchoice"), and the spacing **dt**. We choose $\mathrm{Nt} = 150$ for the electron and $\mathrm{Nt0} = 200$ for the muon, with $\mathrm{dt} = \mathrm{dt0}$ in both cases.

- Finally you can change what you write in the output files: you can print the results only for a subset of values of (x,t) and you can change the value of t at which you start writing (change the value in GeV inside the variable **twrite**, we start from 10 GeV, safely above the non-perturbative regime of QCD). By default, we save the PDFs every 6 values of t and every 5 values of x, with the exception of the last 20, which are all saved. You can change these parameters through the variables **itSteps**, **ixStep** and **NixLast** respectively.