THE EUROPEAN
PHYSICAL JOURNAL C

EpIC: novel Monte Carlo generator for exclusive processes

E. C. Aschenauer[1,a], V. Batozskaya[2,b], S. Fazio[3,c], K. Gates[4,d], H. Moutarde[5,e], D. Sokhan[4,5,f], H. Spiesberger[6,g], P. Sznajder[2,h], K. Tezgin[1,i]

[1] Department of Physics, Brookhaven National Laboratory, Upton, NY 11973, USA
[2] National Centre for Nuclear Research (NCBJ), Pasteura 7, 02-093 Warsaw, Poland
[3] University of Calabria and INFN-Cosenza,, 87036 Rende (CS), Italy
[4] University of Glasgow, Glasgow G12 8QQ, UK
[5] IRFU, CEA, Université Paris-Saclay, 91191 Gif-sur-Yvette, France
[6] PRISMA+ Cluster of Excellence, Institut für Physik, Johannes Gutenberg-Universität, 55099 Mainz, Germany

Abstract We present the EpIC Monte Carlo event generator for exclusive processes based on generalised parton distributions. EpIC utilises the PARTONS framework, which provides a flexible software architecture and a variety of modelling options for the partonic description of the nucleon. The generator offers a comprehensive set of features, including simulation of a variety of exclusive processes and radiative corrections. It may be used both in the analysis of experimental data, as well as for impact studies, especially for the future electron-ion colliders.

1 Introduction

The objective of this study is to develop a Monte Carlo event generator for exclusive processes with the target hadrons remaining intact during the coherent interactions with high-energy leptons. The factorisation theorems developed in the framework of quantum chromodynamics (QCD) allow one to describe such processes in terms of non-perturbative generalised parton distribution functions (GPDs) [1–5] convoluted with perturbatively calculable hard scattering parts of the amplitude.

[a] e-mail: elke@bnl.gov

[b] e-mail: varvara.batozskaya@ncbj.gov.pl

[c] e-mail: salvatore.fazio@unical.it

[d] e-mail: k.gates.1@research.gla.ac.uk

[e] e-mail: herve.moutarde@cea.fr

[f] e-mail: daria.sokhan@cea.fr

[g] e-mail: spiesber@uni-mainz.de

[h] e-mail: pawel.sznajder@ncbj.gov.pl

[i] e-mail: ktezgin@bnl.gov (corresponding author)

GPDs are universal, process-independent functions that parametrise the off-forward nucleon matrix elements of quark and gluon bilinear operators with light-like separations. In case there is no momentum transfer to the nucleon, i.e. in the forward limit, certain GPDs reduce to the collinear parton distribution functions (PDF). Moreover, the first Mellin moments of GPDs are related to elastic form factors [2]. In this regard, GPDs may be viewed as a unified concept of elastic form factors studied through elastic scattering processes and one-dimensional PDFs studied via (semi-) inclusive scattering processes. Another key aspect of GPDs is their relation to nucleon tomography. The Fourier transforms of GPDs are related to the impact parameter space distributions when there is no collinear, but instead finite transverse momentum transfer to the nucleon [6–8]. Through the impact space distributions, GPDs provide information on parton distributions in the transverse plane to the motion of the nucleon, thereby enabling nucleon tomography, the visualisation of the three-dimensional structure of the nucleon, which involves the correlation between impact parameter space distribution functions, parton polarization, and the longitudinal momentum fractions carried by partons. GPDs also exhibit a unique relationship with the energy-momentum tensor (EMT) form factors, which encode fundamental properties of the nucleon, such as mass and spin decomposition of the nucleon into its constituent parts (including orbital angular momentum) [2,3,9] as well as its internal structure based on the so-called "mechanical" forces that contain information about the distribution of pressure and shear forces within the nucleon [10–12]. For more information on GPDs, we refer to the available reviews on the subject, such as Refs. [13,14].

The extraction of GPDs from data measured in exclusive processes is not an easy task, mainly due to the difficulty

Fig. 1 Logo of the project

of solving an inverse problem. There are typically different types of GPDs involved in a process, and they each need to be deconvoluted from the process amplitude. In order to accomplish this, data measured in many different types of exclusive processes are needed with a wide range of kinematic coverage. To this end, measurements of exclusive processes have been performed at several facilities, such as DESY, JLab, and CERN. GPDs constitute also the pillars of the scientific programmes for the next generation of machines. This includes both electron-ion colliders, EIC [15,16], EIcC [17] and LHeC [18], and fixed target experiments at CERN [19] and JLab [20].

As part of this global effort, we have developed a Monte Carlo (MC) generator called EpIC, whose logo appears in Fig. 1. The EpIC generator features a novel architecture using a modular programming approach. In this way, the code structure is kept as simple as possible and the addition of new developments, such as new channels or algorithms to generate random numbers, is made as easy as possible. The architecture of EpIC, including the nomenclature for naming its elements, is based on the PARTONS framework [21]. PARTONS is also used to evaluate the Born cross-section for a given process, which is used to generate MC events after the inclusion of radiative corrections (RCs).

EpIC can generate events for a number of exclusive processes. The following are available at the time of writing this paper: deeply virtual Compton scattering (DVCS), time-like Compton scattering (TCS), and deeply virtual meson production (DVMP) of π^0 and π^+ mesons. In this article, we choose DVCS whenever an exemplary process is needed. Adapting to other processes is straightforward and typically only requires replacing the acronym, e.g. the equivalence of `DVCSGeneratorService` for TCS is `TCSGeneratorService`. EpIC, as in the case of PARTONS, follows the so-called Trento convention [22] for the definition of e.g. scattering angles.

The purpose of this article is to provide a reference and document the structure and functionality of EpIC. We will focus on the explanation of the architecture, starting with a brief description of the PARTONS framework in Sect. 2, which is crucial to understand the technical aspects of EpIC. In Sect. 3, we present the architecture of EpIC. In particular, we list all types of modules and show the relation between them. The user interface is introduced in Sect. 4, while the generator's performance is demonstrated in Sect. 5. The summary is given in Sect. 6. In the appendix (Appendix A), we review the basics of the radiative corrections and explain how to compute them in the collinear approximation. The purpose of this appendix is to help users to familiarise themselves with the parameters of radiative corrections used in EpIC.

EpIC is written in the C++ programming language. The code can be accessed at the GitHub platform [23] and is distributed under the GPL 3.0 licence. The project's webpage, Ref. [24], includes information such as a technical documentation of C++ classes, a guide on how to compile the code, and an example code for users.

2 PARTONS framework

PARTONS (PARtonic Tomography Of Nucleon Software) [21] is a software framework for studying the 3D structure of hadrons. It provides essential tools for QCD phenomenology and allows one to compute observables from models describing non-perturbative objects. For instance, it includes various models of GPDs and methods to perform their pQCD evolution. It also includes methods to evaluate amplitudes and cross-sections for a variety of exclusive processes, such as DVCS and TCS, described at LO and NLO of pQCD precision. The framework can be used in analyses to predict observables from existing models, see e.g. Refs. [16,17,25], but also to constrain new models from available experimental data [26,27]. Because of its versatility, the framework can also act as a laboratory for studying new concepts for phenomenology, like new ways of modelling, see e.g. Refs. [28–30].

The architecture of PARTONS utilises a modular programming approach. The central role in this architecture, which has also been adopted in EpIC, is played by *the modules*. A module is a single encapsulated development of a given type, for instance a single GPD model. The only purpose of modules is to provide data for a specific input, like values of GPDs for a given GPD kinematics. The construction of modules utilises both the inheritance and polymorphism mechanisms of C++. Therefore, the development of new modules is remarkably straightforward, as developers need only to provide their code related to the developments they are interested in and adapt it to the predefined classes in the framework. We demonstrate this with the example of `GPDGK11` module, which is used to implement the Goloskokov–Kroll (GK) GPD model [31–33], from the PARTONS library. The following is an excerpt from the header file:

```cpp
namespace PARTONS {

class GPDGK11: public GPDModule {

public:
```

```
//Unique ID to automatically register the
↪   class in the registry.
static const unsigned int classId;

//Constructor.
//Class name is passed to GPDModule
↪   constructor.
GPDGK11(const std::string& className);

//Used for automation.
virtual void configure(const
↪   ElemUtils::Parameters& parameters);

//Function containing the evaluation of GPD H.
virtual PartonDistribution computeH();


...
};
}
```

where only the most relevant members of the class are shown, some of which will be discussed further in this section. The purpose of the `GPDGK11` module is to evaluate the values of GPDs. This task is predefined in `GPDModule`, being in terms of inheritance the parent class of `GPDGK11`. The pre-definition means here the declaration of a virtual function, so that each of the derived classes (`GPDGK11`, `GPDGK16`, …) have a function of the same signature among its members, but the actual implementation of this function in those classes is, in general, different. The evaluation of GPDs takes place, for instance, in the following function:

```
using namespace PARTONS;

PartonDistribution GPDGK11::computeH(){

//Protected variables that store input GPD
↪   kinematics to be used by developers.
m_x; m_xi; m_t; m_MuF2, m_MuR2;

...

//Container to be returned.
PartonDistribution partonDistribution;

//Store values of H, H^(+) and H^(−),
↪   respectively, for up quarks.
//Results for other flavours can be added
↪   analogously.
partonDistribution.addQuarkDistribution(
↪   QuarkDistribution( QuarkFlavor::UP, ...,
↪   ..., ...));

//Store value of H for gluons.
partonDistribution.setGluonDistribution(...);

return partonDistribution;
}
```

Table 1 The variables used in evaluating GPDs

Symbol	Description
x	Average longitudinal momentum fraction carried by the active parton
ξ	The skewness parameter
t	Squared four-momentum transfer to the hadron
μ_{F}^2	The square of the factorisation scale
μ_{R}^2	The square of the renormalisation scale

The function `GPDGK11::computeH()` returns a container that stores the numerical values of twist-2 GPD H, including its singlet, $H^{(+)}$, and non-singlet, $H^{(-)}$, combinations, as defined in Ref. [13]. The input GPD kinematics is accessible in the body of the function via the `m_x`, `m_xi`, `m_t`, `m_MuF2` and `m_MuR2` variables, which are defined in the `GPDModule` class. Those variables correspond to the GPD variables $x, \xi, t, \mu_{\mathrm{F}}^2$, and μ_{R}^2, respectively, as introduced in Table 1. They are set when one evaluates GPDs via `PartonDistribution compute(const GPDKinematic& kinematic, GPDType::Type gpdType)`, which is also defined in `GPDModule`. As one may see in our example, quark flavours and GPD types are distinguished by `QuarkFlavor` and `GPDType` enums, respectively, while inputs and outputs are encapsulated in the `QuarkDistribution`, `PartonDistribution` and `GPDKinematic` containers.

A single instance of each module is loaded into memory at the beginning of the program execution, namely when static constant variables are initialised. The addresses of such instances are stored by *the registry*, which later acts as a phone book, allowing one to access the instances by either their IDs or defined names. We demonstrate this mechanism with the following example:

```
using namespace PARTONS;

const unsigned int GPDGK11::classId =
↪   BaseObjectRegistry::getInstance()->
↪   registerBaseObject(new
↪   GPDGK11("GPDGK11"));
```

When the program starts, `GPDGK11::classId` is initialised with a unique ID assigned by the registry (`BaseObjectRegistry` class being a singleton). During this process, the registry takes and stores the address of each new instance. All modules inherit from the same basic class called `BaseObject`, allowing the registry to store addresses of the same type of objects. The constructor of `BaseObject` requires a unique string of characters that is used as a human-readable name of the module.

The registry stores addresses of single instances of pre-configured modules. Users may get copies of those instances by using *the factory* (also being a singleton), for instance:

```
using namespace PARTONS;

GPDModule* pGPDModel = nullptr;

//Get by ID.
pGPDModel = Partons::getInstance()->
↪   getModuleObjectFactory()->
↪   newGPDModule(GPDGK11::classId);

//Get by name
pGPDModel = Partons::getInstance()->
↪   getModuleObjectFactory()->
↪   newGPDModule("GPDGK11");
```

The registry and factory are the key architectural ingredients, which allow users to add an unlimited number of new modules in the PARTONS architecture, without having to modify existing modules.

There are many types of modules, but one may *configure* all of them, i.e. pass additional information to the modules, in a generic way, avoiding, for instance, the static casting. This is possible thanks to the **void** configure(**const** ElemUtils::Parameters& parameters) virtual functions, which developers may use to allow transferring additional parameters to modules. For instance:

```
using namespace PARTONS;

void configure(const ElemUtils::Parameters&
↪   parameters){

  //Check if parameter is available.
  if (parameters.isAvailable("myKey")) {

    //Set.
    int myVariable =
    ↪   parameters.getLastAvailable().toInt();
  }
}
```

can be used to set myVariable variable to 1 in the following way:

```
//Container for multiple parameters.
ElemUtils::Parameters parameters;

//Set.
parameters.add(ElemUtils::Parameter("myKey",
↪   1));

//Pass.
pMyModule->configure(parameters);
```

where "myKey" is the unique name of the parameter. The way of supplying modules with additional parameters pre-sented here plays a crucial role in constructing the user interface (UI).

To avoid "manually" repeating some tasks, *the services* are used. In this way, users can perform complex tasks in a straightforward and robust manner by hiding the complexity of low-level functions. The services link such tasks as parsing input files, running many computations, employing multi-threading computing, filling the database, printing the output to the screen etc. Without services, all those tasks would have to be done explicitly by users, requiring a profound knowledge of the functioning of the project. This way of working would also be time consuming and vulnerable to mistakes.

3 EpIC's architecture

The generic architecture of the generator, including the flow of data transferred between the modules, is shown in Fig. 2. Each module takes input data, processes it according to pre-defined tasks, and returns the corresponding output. The actual implementation of some types of modules depends on the simulated process. For instance, RCModule is the parent class for DVCSRCModule and TCSRCModule, which are used for DVCS and TCS processes, respectively. These two types of modules process data according to the same task predefined in RCModule, but they use different inputs. It is either DVCS or TCS kinematics. Because of this difference, RCModule has been developed as a C++ template, allowing DVCSRCModule and TCSRCModule to handle input containers of various types. With those templates, the addition of new processes to the generator is as easy as possible. DVCSRCModule and TCSRCModule are used as base classes for actual modules, where specific physics developments are encoded. In Fig. 3 we present the entire tree of inheritance for modules of RCModule type.

Below, we provide a brief description of each type of module and its purpose. Additionally, we indicate whether the actual implementation of modules of a given type depends on the simulated process. This information is useful in recognising which parts of the generator need to be extended when a new process is added to the generator.
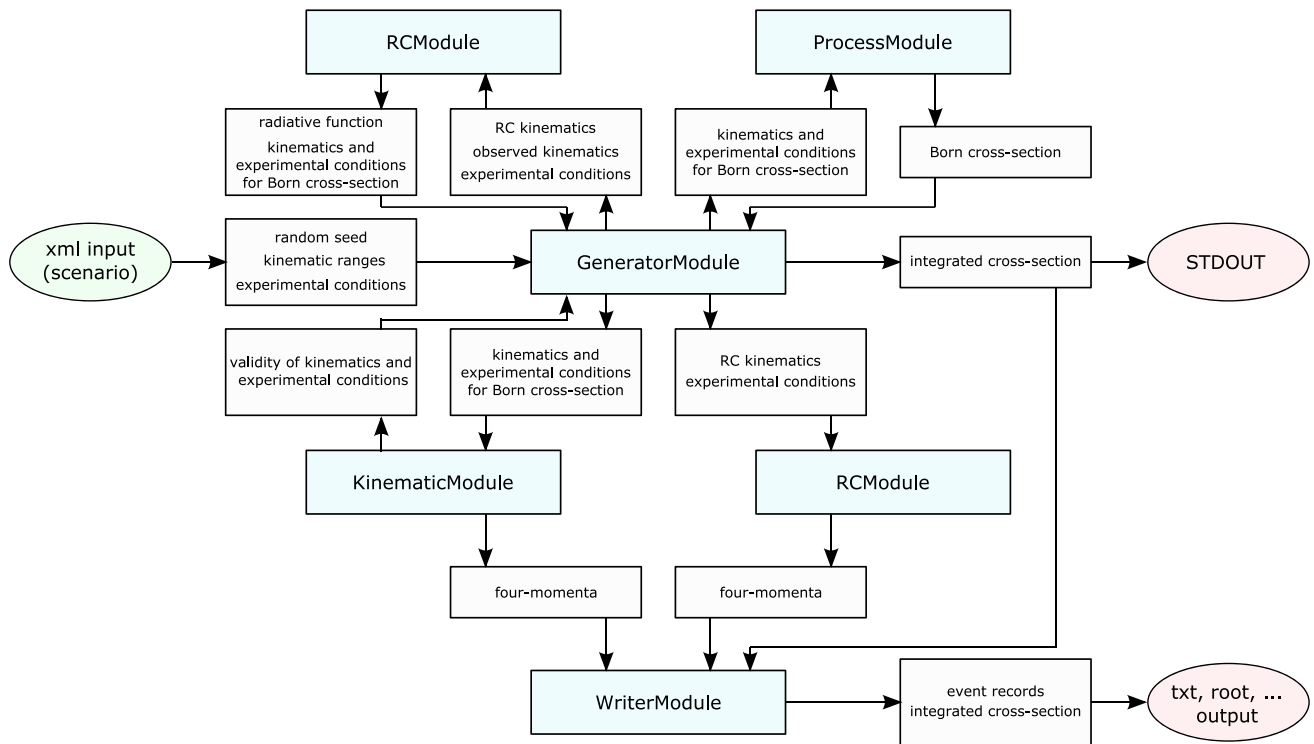
3.1 RCModule

Purpose: Simulation of radiative corrections.
*Does it depend on the generated process?* Yes.
Description: In EpIC, events are generated according to the probability distribution given by:

$$\mathrm{d}\sigma(X', Z) = \mathrm{d}\sigma_0(X)\, R(Z)\,. \tag{1}$$

Here, $d\sigma_0(X)$ is the Born cross-section (given by PARTONS), while $R(Z)$ is the radiator function. The set of vari-

Fig. 2 Sketch showing the generic architecture of the project, i.e. without specifying the corresponding physics process for some modules. Details are provided in the text
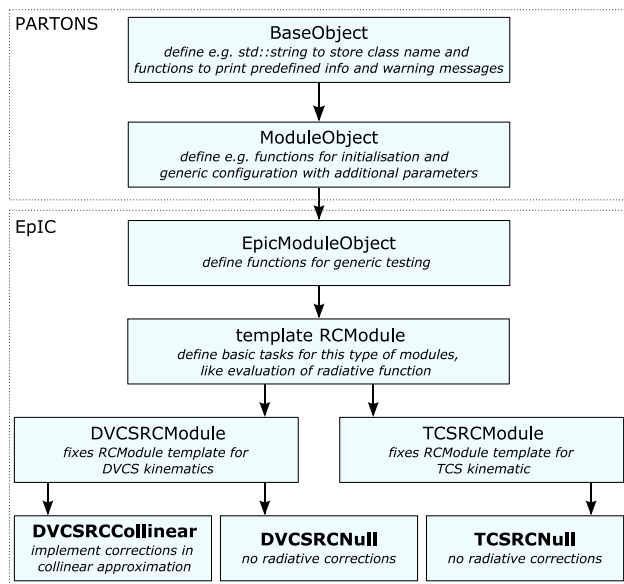
Fig. 3 Inheritance tree of modules used for the implementation of radiative corrections. Each box represents a single C++ class. Two first classes are the part of PARTONS library, the rest are the part of EpIC project. Classes acting as modules to be used by users are indicated by boldface fonts

ables that the Born cross-section depends on is denoted by X, for instance $X = \{x_{Bj}, t, Q^2, \phi, \phi_S, E_l\}$ for DVCS, where x_{Bj} and Q^2 are the usual DIS variables, ϕ is the angle between

the lepton scattering plane and the production plane, ϕ_S is the angle between the lepton scattering plane and the target spin component perpendicular to the direction of the virtual photon, and E_l is the lepton energy in the fixed target frame, which is the reference frame for the evaluation of cross-section in PARTONS [21]. Because of the radiative corrections, the values of those variables may be different from those for a generated event, X'. The radiator function, on the other hand, depends on the set of variables Z. The number of variables making this set depends on the approximation one uses. For instance, if one considers the collinear approximation and only the initial and final state radiations of single photons from the incoming and scattered leptons, two variables make the set Z, namely $Z = \{z_1, z_3\}$, see Appendix A for more details.

Taking the above into account, modules of type `RCModule` perform three actions: (i) they define a number of variables that the radiative corrections depend on, i.e. they define the set Z, (ii) they implement a function used for the evaluation of $R(Z)$, but also for the evaluation of the "true" kinematics entering the Born cross-section, $X = f(X', Z)$, (iii) they implement a function that stores the four-momenta of radiated photons in the final event records.

3.2 ProcessModule

Purpose: Evaluation of Born cross-section.

Does it depend on the generated process? Yes.

Description: The Born cross-section is evaluated in EpIC using the modules of `ProcessModule` type, which are defined in the PARTONS library. In the most general case, when the Born cross-section is calculated directly from GPDs, the `ProcessModule` relies upon other modules: `ConvolCoeffFunctionModule` for the evaluation of the process amplitudes, `GPDModule` for the evaluation of GPDs at a reference scale, `GPDEvolutionModule` for the evolution of GPDs, `XiConverterModule` for the evaluation of the GPD skewness variable, ξ, from the process kinematics, and `ScalesModule` for the evaluation of the renormalisation and factorisation scales from the process kinematics. However, this general case can hardly be used in the Monte Carlo generation, because of the nested integrations evaluating the Born cross-section. Instead, one may start the evaluation, for instance, from the level of amplitudes parametrised in lookup tables. In this case, `GPDModule` and `GPDEvolutionModule` are not used as there is no need for a convolution of GPDs with hard scattering coefficient functions. More details about the `ProcessModule` modules can be found in Ref. [21].

3.3 EventGeneratorModule

Purpose: Generation of kinematic configurations according to the probability distribution given by the product of the Born cross-section and the radiator function.

Does it depend on the generated process? No.

Description: Modules of this type play a central role in the generation of events. Three actions are implemented in those modules: (i) Initialisation, where the algorithm implemented in the module probes the probability distribution, for instance to "memorise" it using an interpolation method, or to capture its key features, like the location of places in which the probability distribution tends to change rapidly. For some complicated probability distribution functions, the initialisation process can be time-consuming, but the results of the process can be stored in a file for use in subsequent jobs. (ii) Generation of kinematic configurations according to the probability distribution, based on the data collected during the initialisation stage. (iii) Evaluation of the total cross-section, which is needed to normalise the event distributions to a given integrated luminosity.

3.4 KinematicModule

Purpose: Evaluation of four-momenta for a given kinematic configuration.

Does it depend on the generated process? Yes.

Description: Modules of this type are responsible for the evaluation of four-momenta for a given kinematic configuration. In addition, they implement a predefined function that determines if a given kinematic configuration is physical, i.e. if it does not break any kinematic limit. This function is utilised by `GeneratorModule` to probe only valid kinematics.

3.5 WriterModule

Purpose: Composing and saving event records.

Does it depend on the generated process? No.

Description: The purpose of modules of this type is to create and store event records in output files. Different formats may be implemented by different modules. Additional information, like the integrated cross-section, can also be stored in output files, e.g. in their headers.

4 User interface

Both compilation and linking of the project make use of the CMake tool [34]. In the current version (1.0.0) EpIC requires the following external libraries: PARTONS [21] (providing elements of the architecture and used for the evaluation of Born cross-sections), ROOT [35] (used in one of the `EventGeneratorModule` modules), HepMC3 [36] (used in one of the `WriterModule` modules), GSL [37] (used for the generation of random numbers). For more detailed and always up-to-date information we refer to the online documentation [24].

The executable of the project, `epic`, must be invoked with two arguments, like this:

```
./epic --seed=SEED --scenario=SCENARIO_PATH
```

where `SEED` is the random seed (unsigned integer) to be used in the initialisation of modules that deal with random numbers, and `SCENARIO_PATH` is the relative or absolute path to *the scenario* containing all options used in the generation.

The EpIC scenarios are written in XML markup language which, thanks to the usage of tags based on standard words, is an easy-to-read format for humans while being straightforward for machines to process. EpIC assumes input data to be provided in units of GeV (and its powers, whenever applicable), and radians for angles. The general structure of a single scenario is the following:

```
<!-- XML header -->
<?xml version="1.0" encoding="UTF-8"
↪  standalone="yes" ?>

<!-- Definition of scenario -->
```

```xml
<!-- For bookkeeping it includes date and
↪   description -->
<scenario date="2022-01-01" description="My
↪   first scenario">

<!-- Selection of service and its method-->
<task service="DVCSGeneratorService"
↪   method="generate">

  <!-- General configuration -->
  <general_configuration>
  ...
  </general_configuration>

  <!-- Selection of kinematic ranges -->
  <kinematic_range>
  ...
  </kinematic_range>

  <!-- Indication of experimental conditions
  ↪   -->
  <experimental_conditions>
  ...
  </experimental_conditions>

  <!-- Configuration of ProcessModule -->
  <computation_configuration>
  ...
  </computation_configuration>

  <!-- Configuration of GeneratorModule -->
  <generator_configuration>
  ...
  </generator_configuration>

  <!-- Configuration of KinematicModule -->
  <kinematic_configuration>
   ...
  </kinematic_configuration>

  <!-- Configuration of RCModule -->
  <rc_configuration>
  ...
   </rc_configuration>

  <!-- Configuration of WriterModule -->
  <writer_configuration>
  ...
  </writer_configuration>

</task>
</scenario>
```

Here, we only expose blocks containing specific information, like **`<general_configuration>`** ... `</general_configuration>`. Each block is described in one of the subsequent subsections. In our demonstration we only show an exemplary scenario for the DVCS case. More examples, including those for other processes, can be found online [24].

4.1 general_configuration

```xml
<general_configuration>

  <!-- Number of events to be generated -->
  <param name="number_of_events" value="10" />

  <!-- Subprocess. Possible values: "DVCS"
  ↪   (pure DVCS), "BH" (pure Bethe-Heitler),
  ↪   "DVCS|BH" (pure DVCS and BH),
  ↪   "DVCS|BH|INT" (pure DVCS, BH and the
  ↪   interference between the two) -->
  <param name="subprocess_type" value="DVCS"
  ↪   />

</general_configuration>
```

4.2 kinematic_range

```xml
<kinematic_range>

  <!-- Range of xB (Bjorken variable)-->
  <param name="range_xB" value="0.|1." />

  <!-- Range of t -->
  <param name="range_t" value="-1.|0." />

  <!-- Range of Q2 -->
  <param name="range_Q2" value="1.|10." />

  <!-- Range of phi -->
  <param name="range_phi" value="0.|2*pi" />

  <!-- Range of phiS -->
  <param name="range_phiS" value="0.|2*pi" />

  <!-- Range of y -->
  <param name="range_y" value="0.01|0.95" />

</kinematic_range>
```

4.3 experimental_conditions

```xml
<experimental_conditions>

  <!-- Energy of the lepton beam -->
  <param name="lepton_energy" value="5." />

  <!-- Type of the lepton beam, here electron
  ↪   -->
  <param name="lepton_type" value="e-" />

  <!-- Polarisation of the lepton beam -->
  <param name="lepton_helicity" value="1" />

  <!-- Energy of the hadron beam -->
  <!-- For a target in rest frame use:
  ↪   value="fixed_target" -->
  <param name="hadron_energy" value="10." />

  <!-- Type of the hadron beam, here proton
  ↪   -->
  <param name="hadron_type" value="p" />
```

```xml
<!-- Polarisation of the hadron beam -->
<!-- Possible values:  "0|0|0" for
↪   unpolarised target, "0|0|±1" for long.
↪   polarised target, "±1|0|0" or "0|±1|0"
↪   for trans. polarised target -->
<param name="hadron_polarisation"
↪   value="0.|0.|0." />


</experimental_conditions>
```

4.4 computation_configuration

```xml
<computation_configuration>

<!-- Selection of module -->
<!-- DVCSProcessBMJ12 module encodes the BH,
↪   DVCS, and interference terms of the Born
↪   cross-section according to Refs.~[38]
↪   and [39] -->
<module type="DVCSProcessModule"
↪   name="DVCSProcessBMJ12">

  <!-- Selection of module used for
  ↪   evaluation of factorisation and
  ↪   renormalisation scales from DVCS
  ↪   kinematics -->
  <!-- DVCSScalesQ2Multiplier module
  ↪   identifies both scales in terms of the
  ↪   Q^2 variable -->
  <module type="DVCSScalesModule"
  ↪   name="DVCSScalesQ2Multiplier">
  </module>

  <!-- Selection of module used for
  ↪   evaluation of the GPD skewness
  ↪   variable, ξ, from DVCS kinematics -->
  <!-- DVCSXiConverterXBToXi module use the
  ↪   conversion ξ = x_Bj/(2 − x_Bj) -->
  <module type="DVCSXiConverterModule"
  ↪   name="DVCSXiConverterXBToXi">
  </module>

  <!-- Selection of module used for the
  ↪   evaluation DVCS Compton form factors
  ↪   (CFFs) -->
  <!-- DVCSCFFCMILOU3DTables module does not
  ↪   evaluate CFFs from a GPD model during
  ↪   EpIC run-time. Instead, it uses
  ↪   look-up tables of CFFs evaluated ahead
  ↪   of time for a particular GPD module. A
  ↪   number of these tables are included
  ↪   with EpIC.-->
  <module
  ↪   type="DVCSConvolCoeffFunctionModule"
  ↪   name="DVCSCFFCMILOU3DTables">

    <!-- pQCD order of evaluation -->
    <param name="qcd_order_type" value="LO"
    ↪   />

    <!-- Path to a look-up table, here the
    ↪   one based on the GK GPD model is
    ↪   specified -->
```

```xml
    <param name="cff_set_file"
    ↪   value="PATH/epic/data/
    ↪   DVCSCFFCMILOU3DTables/tables_GK.root"
    ↪   />
  </module>

</module>
</computation_configuration>
```

4.5 generator_configuration

```xml
<generator_configuration>

<!-- Selection of module -->
<!-- EventGeneratorFOAM module uses
↪   mini-FOAM library [40] that is issued
↪   with ROOT [35] -->
<module type="EventGeneratorModule"
↪   name="EventGeneratorFOAM">

  <!-- Parameters of the FOAM algorithm -->

  <!-- Maximum number of cells -->
  <param name="nCells" value="10000" />

  <!--Number of MC events when exploring a
  ↪   cell -->
  <param name="nSamples" value="2000" />

  <!-- Number of bins in edge histogram for
  ↪   a cell -->
  <param name="nBins" value="2000" />

  <!-- After the initialisation process, the
  ↪   state of FOAM will be saved in
  ↪   PATH/state.root. To use this file in
  ↪   another EpIC run, so as to skip the
  ↪   initialisation in that run, use the
  ↪   read_state_file_path option. -->
  <param name="save_state_file_path"
  ↪   value="PATH/state.root" />
</module>
</generator_configuration>
```

4.6 kinematic_configuration

```xml
<kinematic_configuration>

<!-- Selection of module -->
<!-- DVCSKinematicDefault module provides
↪   default evaluation of four-momenta from
↪   DVCS kinematics -->
<module type="DVCSKinematicModule"
↪   name="DVCSKinematicDefault">
</module>
</kinematic_configuration>
```

4.7 rc_configuration

```
<rc_configuration>

  <!-- Selection of module -->
  <!-- DVCSRCCollinear module provides
  ↪  evaluation of initial and final state
  ↪  radiative corrections from lepton lines
  ↪  using the collinear approximation, see
  ↪  Sect.~Appendix A for more details -->
  <module type="DVCSRCModule"
  ↪  name="DVCSRCCollinear">
  </module>
</rc_configuration>
```

4.8 writer_configuration

```
<writer_configuration>

  <!-- Selection of module -->
  <!-- WriterHepMC3 module is used to save
  ↪  event records in HepMC3 format  [36] -->
  <module type="WriterModule"
  ↪  name="WriterHepMC3">

    <!-- Path to output file containing event
    ↪  records -->
    <param name="output_file_path"
    ↪  value="test.txt" />

    <!-- Format of output file containing
    ↪  event records: "ascii" for text
    ↪  format, or "root" for binary ROOT
    ↪  format-->
    <param name="HepMC3_writer_type"
    ↪  value="ascii" />
  </module>
</writer_configuration>
```

5 EpIC performance

In the following, we demonstrate the detailed performance of EpIC by using specific components of the PARTONS framework. The demonstration is based on one million MC events generated for the DVCS sub-process, i.e. exclusive leptoproduction of a single photon without the Bethe-Heitler contribution, colliding a 10 GeV positive helicity electron with a 100 GeV unpolarized proton. The following modules are used during the generation process: DVCSCFFCMILOU3DTables for the parameterisation of the Compton Form Factors (CFFs) obtained from the GK GPD model [31–33] and LO coefficient functions, DVCSProcessBMJ12 for the evaluation of the DVCS cross-section based on the set of expressions published in Ref. [39], DVCSRCNull for the simulation without radiative corrections, EventGeneratorFOAM for the generation of

DVCS kinematics based on a cross-section with the FOAM algorithm [40], DVCSKinematicDefault for the default evaluation of the four-momenta of the DVCS kinematics, and WriterHepMC3 to save the four-momenta in the HepMC3 format.

The following cuts are used in the generation process: $0.0001 \leq x_B \leq 0.6$, $0.01 \leq y \leq 0.95$ (here, y is the inelasticity parameter), $1 \leq Q^2 \leq 100$ GeV^2, $0 \leq |t| \leq 1$ GeV^2, $0 \leq \phi \leq 2\pi$, and $0 \leq \phi_S \leq 2\pi$. The event distributions based on these cuts are shown in Fig. 4. The following FOAM library parameters are used in producing those events [40]: nCells = 3000, nSamples = 600, nBins = 600. For the FOAM library, the initialisation lasted about 40 minutes, whereas the generation time after initialisation took around 0.0052 seconds per event at the BNL SDCC computer farm. The initialisation time can be shortened by changing the parameters of the FOAM library, which may, however, lead to a longer time to generate a single event. We remind that the result of the initialisation can be stored in a file, allowing one to avoid this step in subsequent MC runs.

To ensure that our results are consistent, we also plot the theory expectation values, as solid lines, on top of each histogram. The number of events associated with the theory curves is calculated as follows: at each bin, the cross-section accounting for the bin width is calculated, and then the result is divided by the total cross-section in the respective phase space. One can then multiply this ratio by the total number of events to find the number of events in that specific bin. To obtain those curves in Fig. 4, we first assign the number of events to the middle point of each bin, and then interpolate between results at these points. As a result, we observe that EpIC generates events in agreement with the theory values.

6 Summary

In this work, we described in detail the overall structure and essential features of the novel MC generator EpIC. This generator is based on the PARTONS framework in the model selection. It is designed to generate events for exclusive processes, such as DVCS, TCS, DVMP and more broadly for all exclusive processes implemented in the PARTONS framework. As these processes have the potential to reveal the 3D structure of the nucleon through GPDs, they have been studied extensively at facilities such as DESY, JLab, and CERN, and they will remain crucial for future EIC programmes.

The EpIC MC generator stands out with its flexible architecture that utilises a modular programming paradigm. Consequently, the code is easier to understand, as each element of the architecture is designed to perform a specific task. The distribution of events generated by EpIC is highly precise since it utilises an accurate representation of the cross-section of the underlying process in the FOAM library. Besides gen-
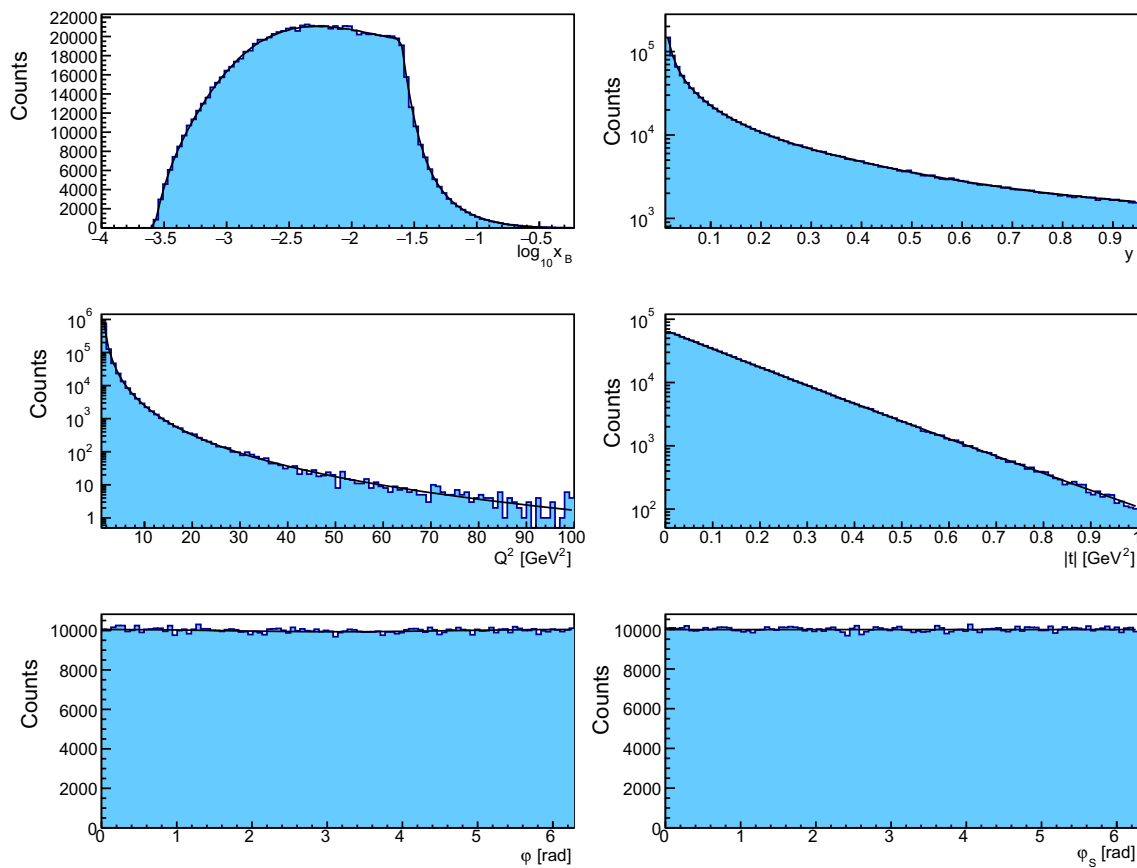
Fig. 4 Distribution of one million DVCS events produced by the `DVCSProcessBMJ12` module utilising `GPDGK16` GPDs within the PARTONS framework. Events are binned with respect to $\log_{10}(x_B)$, y, Q^2, $|t|$, ϕ and ϕ_S. The solid curves on top of the his-tograms depict the expected number of events for the given model. The curves are normalised according to the integrated luminosity estimated by EpIC

erating accurate events, EpIC also allows users to generate events integrating radiative corrections as well as to run multi-channel analyses. The comprehensive set of features and the variety of models available in EpIC make it ideal for the systematic study of the 3D structure of the nucleon.

The generator was designed in a way that facilitates easy expansion. The encapsulation of elements is one of the fundamental characteristics of the generator when it comes to maintaining the project for a long time and with a changing team of developers. This allows us to easily adapt the architecture to the latest developments. Finally, to make this project as useful to the particle physics community as possible, we promote open-source standards and offer easy access to its development (EpIC is released through GitHub service [23] under a GPL licence). This perspective is well suited to the experimental timeline of the 3D hadron structure community.

Appendix A: Radiative corrections

Electromagnetic radiative corrections can have a significant impact on the interpretation of experimental data, as they introduce uncertainties in the reconstruction of the "true" kinematics of the measured process. On the other hand, neglecting radiative corrections in simulations typically leads to unrealistic distributions of generated events, resulting in e.g. impact studies that are much less robust. To address both of these problems it is highly desirable to have radiative corrections implemented in MC generators.

In the literature, it has been discussed that the leptonic initial and final state radiative corrections, i.e. when photons are emitted by either incoming or outgoing leptons, respectively, can be computed with a good accuracy in the collinear approximation [41,42]. In this approximation, the transverse component of the momenta of emitted photons is neglected. Therefore, the radiated photons are restricted to move along the direction of the source leptons.

The collinear approximation allows one to introduce a single parameter, z_1 or z_3, for the initial or final state radiation, respectively. The parameter describes the energy of the emitted photon:

$$z_1 = \frac{E_e - E_\gamma}{E_e}, \qquad z_3 = \frac{E_{e'}}{E_{e'} + E_{\gamma'}}, \qquad (A.1)$$

where E_e ($E_{e'}$) and E_γ ($E_{\gamma'}$) are the energies of the incoming (outgoing) lepton and the initial (final) state emitted photon, respectively. The DIS cross section can be then expressed by [42]:

$$\frac{d^2\sigma}{dx\,dy} = \int_{z_1^{min}}^1 \frac{dz_1}{z_1} D(z_1) \int_{z_3^{min}}^1 \frac{dz_3}{z_3^2} \overline{D}(z_3) \frac{y}{\hat{y}} \frac{d^2\hat{\sigma}_{Born}}{d\hat{x}\,d\hat{y}}, \qquad (A.2)$$

where $d^2\hat{\sigma}_{Born}$ is the differential Born cross-section evaluated for "true" variables, which are related to the observed ones by [42]:

$$\hat{x} = \frac{z_1 xy}{z_1 z_3 + y - 1}, \quad \hat{y} = \frac{z_1 z_3 + y - 1}{z_1 z_3}, \qquad (A.3)$$

and where, due to kinematic limits, one has:

$$z_1^{min} = \frac{1-y}{1-xy}, \qquad z_3^{min} = 1 - y(1-x). \qquad (A.4)$$

In Eq. (A.2) the radiator functions, $D(z_1)$ and $\overline{D}(z_3)$, describe the weight of the rescaled cross-section in the presence of the initial and final state radiations, respectively. At the leading order $D(z_1)$ and $\overline{D}(z_3)$ are equal to each other and are given by:

$$D(z) = \left[\delta(1-z) \left[1 + \frac{\alpha}{2\pi} L \left(2 \ln \epsilon + \frac{3}{2} \right) \right] \right.$$
$$\left. + \theta(1 - \epsilon - z) \frac{\alpha}{2\pi} L \frac{1 + z^2}{1 - z} \right], \qquad (A.5)$$

with $L = \ln(Q^2/m_l^2)$. Here, α is the fine-structure constant, and m_l is the lepton mass. The parameter ϵ acts as a cutoff avoiding the generation of very soft photons characterised by energies $E_\gamma \leq \epsilon E_e$ and $E_{\gamma'} \leq \epsilon E_{e'}$. It is crucial to choose the cutoff carefully so that ϵE_e and $\epsilon E_{e'}$ are much smaller than the energy resolution of the experiment. Due to the cutoff, Eq. (A.5) consists of two parts: the term proportional to the Dirac delta function collectively describes the contribution of very soft photons and virtual corrections, while the term proportional to the step function describes the effects of real radiation of photons with sizeable energies.

The generalisation of leading order radiative corrections for the DVCS case can be achieved as follows:

$$\frac{d^5\sigma}{dx\,dQ^2\,dt\,d\phi\,d\phi_S} = \int_{z_1^{min}}^1 dz_1 z_1 D(z_1) \int_{z_3^{min}}^1 \frac{dz_3}{z_3^2} \overline{D}(z_3)$$
$$\times \frac{y}{\hat{y}} \frac{d^5\hat{\sigma}_{Born}}{d\hat{x}\,d\hat{Q}^2\,dt\,d\phi\,d\phi_S}. \qquad (A.6)$$

References

1. D. Müller, D. Robaschik, B. Geyer, F.M. Dittes, J. Hořejši, Fortschr. Phys. **42**, 101 (1994). https://doi.org/10.1002/prop.2190420202
2. X.D. Ji, Phys. Rev. Lett. **78**, 610 (1997). https://doi.org/10.1103/PhysRevLett.78.610
3. X.D. Ji, Phys. Rev. D **55**, 7114 (1997). https://doi.org/10.1103/PhysRevD.55.7114
4. A.V. Radyushkin, Phys. Lett. B **385**, 333 (1996). https://doi.org/10.1016/0370-2693(96)00844-1
5. A.V. Radyushkin, Phys. Rev. D **56**, 5524 (1997). https://doi.org/10.1103/PhysRevD.56.5524
6. M. Burkardt, Phys. Rev. D **62**, 071503 (2000). https://doi.org/10.1103/PhysRevD.62.071503. [Erratum: Phys. Rev. D **66**, 119903 (2002)]
7. M. Burkardt, Int. J. Mod. Phys. A **18**, 173 (2003). https://doi.org/10.1142/S0217751X03012370
8. M. Burkardt, Phys. Lett. B **595**, 245 (2004). https://doi.org/10.1016/j.physletb.2004.05.070
9. X.D. Ji, W. Melnitchouk, X. Song, Phys. Rev. D **56**, 5511 (1997). https://doi.org/10.1103/PhysRevD.56.5511
10. M.V. Polyakov, Phys. Lett. B **555**, 57 (2003). https://doi.org/10.1016/S0370-2693(03)00036-4
11. M.V. Polyakov, P. Schweitzer, Int. J. Mod. Phys. A **33**(26), 1830025 (2018). https://doi.org/10.1142/S0217751X18300259
12. C. Lorcé, H. Moutarde, A.P. Trawiński, Eur. Phys. J. C **79**(1), 89 (2019). https://doi.org/10.1140/epjc/s10052-019-6572-3
13. M. Diehl, Phys. Rep. **388**, 41 (2003). https://doi.org/10.1016/j.physrep.2003.08.002

14. S. Boffi, B. Pasquini, Riv. Nuovo Cim. **30**(9), 387 (2007). https://doi.org/10.1393/ncr/i2007-10025-7
15. A. Accardi et al., Eur. Phys. J. A **52**(9), 268 (2016). https://doi.org/10.1140/epja/i2016-16268-9
16. R. Abdul Khalek et al., Science requirements and detector concepts for the electron-ion collider: EIC yellow report. Nucl. Phys. A **1026**, 122447 (2022). https://doi.org/10.1016/j.nuclphysa.2022.122447. arXiv:2103.05419 [physics.ins-det]
17. D.P. Anderle et al., Front. Phys. (Beijing) **16**(6), 64701 (2021). https://doi.org/10.1007/s11467-021-1062-0
18. J.L. Abelleira Fernandez et al., J. Phys. G **39**, 075001 (2012). https://doi.org/10.1088/0954-3899/39/7/075001
19. B. Adams et al., COMPASS++/AMBER: proposal for Measurements at the M2 beam line of the CERN SPS Phase-1: 2022-2024. Technical report, CERN, Geneva (2019). CERN-SPSC-2019-022, SPSC-P-360
20. J. Arrington et al., Physics with CEBAF at 12 GeV and future opportunities (2021). arXiv:2112.00060 [nucl-ex]
21. B. Berthou et al., Eur. Phys. J. C **78**(6), 478 (2018). https://doi.org/10.1140/epjc/s10052-018-5948-0
22. A. Bacchetta, U. D'Alesio, M. Diehl, C.A. Miller, Phys. Rev. D **70**, 117504 (2004). https://doi.org/10.1103/PhysRevD.70.117504
23. EpIC project on GitHub. https://github.com/pawelsznajder/epic. Accessed 13 June 2022
24. EpIC project webpage. https://pawelsznajder.github.io/epic. Accessed 13 June 2022
25. O. Grocholski, H. Moutarde, B. Pire, P. Sznajder, J. Wagner, Eur. Phys. J. C **80**(2), 171 (2020). https://doi.org/10.1140/epjc/s10052-020-7700-9
26. H. Moutarde, P. Sznajder, J. Wagner, Eur. Phys. J. C **78**(11), 890 (2018). https://doi.org/10.1140/epjc/s10052-018-6359-y
27. H. Moutarde, P. Sznajder, J. Wagner, Eur. Phys. J. C **79**(7), 614 (2019). https://doi.org/10.1140/epjc/s10052-019-7117-5
28. V. Bertone, H. Dutrieux, C. Mezrag, H. Moutarde, P. Sznajder, Phys. Rev. D **103**(11), 114019 (2021). https://doi.org/10.1103/PhysRevD.103.114019
29. J.M.M. Chavez, V. Bertone, F.D.S. Borrero, M. Defurne, C. Mezrag, H. Moutarde, J. Rodríguez-Quintero, J. Segovia, Pion GPDs: a path toward phenomenology. Phys. Rev. D **105**(9), 094012 (2022). https://doi.org/10.1103/PhysRevD.105.094012. arXiv:2110.06052 [hep-ph]
30. H. Dutrieux, O. Grocholski, H. Moutarde, P. Sznajder, Eur. Phys. J. C **82**(3), 252 (2022). https://doi.org/10.1140/epjc/s10052-022-10211-5
31. S.V. Goloskokov, P. Kroll, Eur. Phys. J. C **42**, 281 (2005). https://doi.org/10.1140/epjc/s2005-02298-5
32. S.V. Goloskokov, P. Kroll, Eur. Phys. J. C **53**, 367 (2008). https://doi.org/10.1140/epjc/s10052-007-0466-5
33. S.V. Goloskokov, P. Kroll, Eur. Phys. J. C **65**, 137 (2010). https://doi.org/10.1140/epjc/s10052-009-1178-9
34. CMake project webpage. https://cmake.org. Accessed 13 June 2022
35. R. Brun, F. Rademakers, Nucl. Instrum. Methods A **389**, 81 (1997). https://doi.org/10.1016/S0168-9002(97)00048-X
36. A. Buckley, P. Ilten, D. Konstantinov, L. Lönnblad, J. Monk, W. Pokorski, T. Przedzinski, A. Verbytskyi, Comput. Phys. Commun. **260**, 107310 (2021). https://doi.org/10.1016/j.cpc.2020.107310
37. M. Galassi et al., *GNU Scientific Library: Reference Manual* (Network Theory, 2009). ISBN:0954612078
38. A.V. Belitsky, D. Mueller, A. Kirchner, Nucl. Phys. B **629**, 323 (2002). https://doi.org/10.1016/S0550-3213(02)00144-X
39. A.V. Belitsky, D. Müller, Y. Ji, Nucl. Phys. B **878**, 214 (2014). https://doi.org/10.1016/j.nuclphysb.2013.11.014
40. S. Jadach, P. Sawicki, Comput. Phys. Commun. **177**, 441 (2007). https://doi.org/10.1016/j.cpc.2007.02.112
41. L.W. Mo, Y.S. Tsai, Rev. Mod. Phys. **41**, 205 (1969). https://doi.org/10.1103/RevModPhys.41.205
42. J. Kripfganz, H.J. Mohring, H. Spiesberger, Z. Phys. C **49**, 501 (1991). https://doi.org/10.1007/BF01549704