PARTONS

Partonic Tomography of Nucleon Software

Herve Dutrieux
William & Mary
hldutrieux@wm.edu

Outline

- 1) What is PARTONS and what can it offer to you?
- 2) GPD models and evolution
- 3) CFFs, observables and the deconvolution problem

1) What is PARTONS and what can it offer to you?

- PARTONS is an extensive **open-source** software dedicated to the study of the 3D structure of hadrons using GPDs and TMDs.
- Focusing on GPDs, it is useful for:

Theoreticians

Implement and propagate in the community new models of GPDs

Phenomenologists

Evaluate the sensitivity of GPDs to some observables

Extract GPDs from fits to experimental data

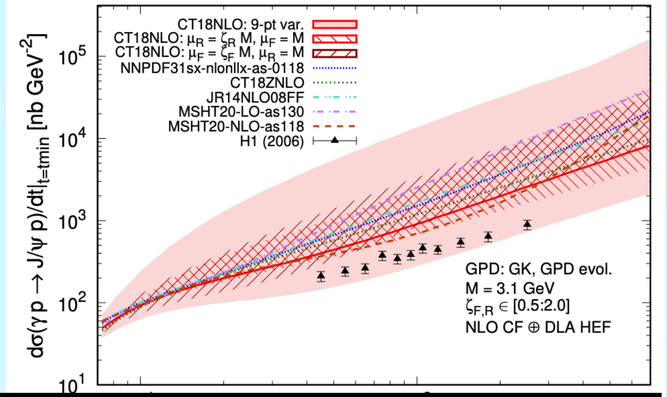
Study the effect of evolution and perturbative corrections

Experimentalists

Design experiments with an event generator

Exclusive vector-quarkonium photoproduction at NLO in α_s in collinear factorisation with evolution of the generalised parton distributions and high-energy resummation

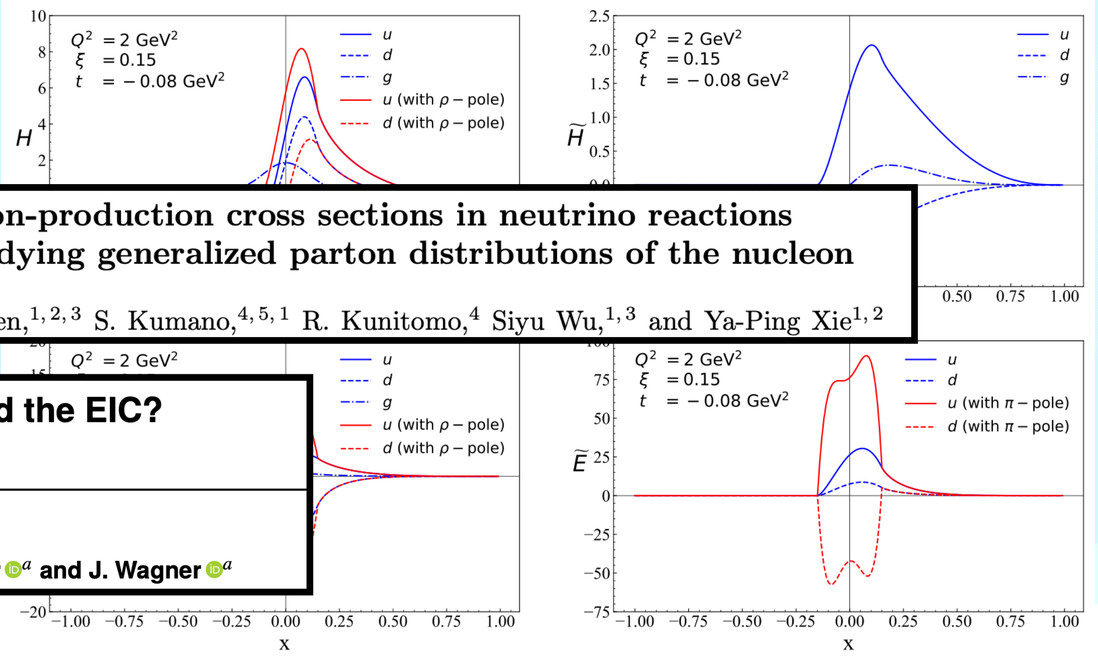
C.A. Flett [a], J.P. Lansberg [a], S. Nabeebaccus [a], M. Nefedov [a], P. Sznajder [b], J. Wagner [b]



And many more…

Proposal to PAC51
PR12+23-002

Beam Charge Asymmetries for Deeply Virtual Compton Scattering on the Proton at CLAS12

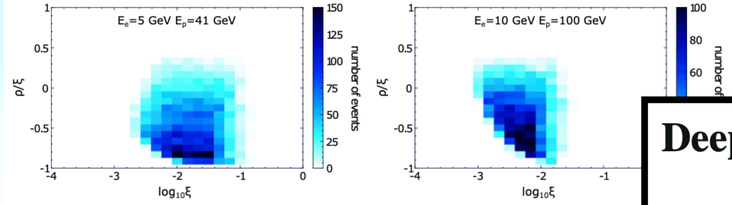


Pion-production cross sections in neutrino reactions for studying generalized parton distributions of the nucleon

Xurong Chen,[1,2,3] S. Kumano,[4,5,1] R. Kunitomo,[4] Siyu Wu,[1,3] and Ya-Ping Xie[1,2]

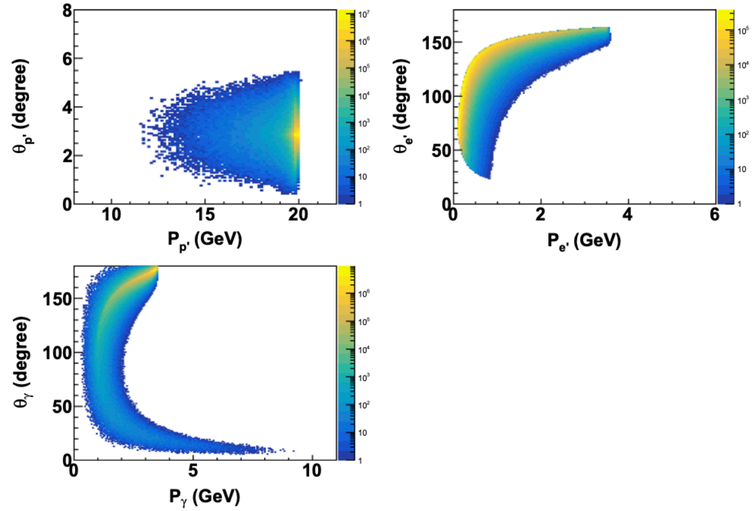
Can we measure Double DVCS at JLab and the EIC?

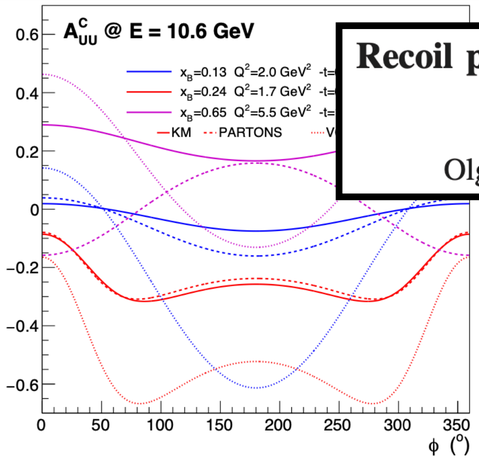
K. Deja [a], V. Martínez-Fernández [a,*], B. Pire [b], P. Sznajder [a] and J. Wagner [a]



Deeply virtual compton scattering at future electron-ion colliders

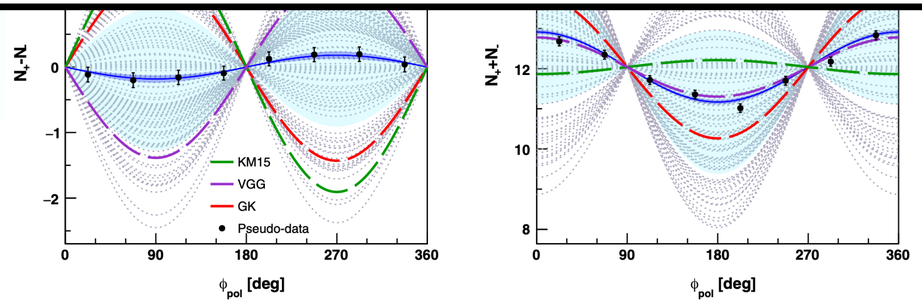
Gang Xie[1,2], Wei Kou[1,2], Qiang Fu[1,2], Zhenyu Ye[3,a], Xurong Chen[1,2,b]





Recoil proton polarization: A new discriminative observable for deeply virtual Compton scattering

Olga Bessidskaia Bylund, Maxime Defurne, and Pierre A. M. Guichon



- PARTONS is a **framework**, a **collection** of tools, models, observables aiming at making it easy for the community to move forward with impact studies, phenomenological extractions, etc.

- Initial preprint in 2015, published description in B. Berthou *et al*. *Eur.Phys.J.C* 78 (2018) 6, 478

- Tutorials, installation guides, documentation, … on

https://partons.cea.fr

- Source code publicly available on the GitLab

https://drf-gitlab.cea.fr/partons/

Outline

- 1) What is PARTONS and what can it offer to you?
- 2) GPD models and evolution
- 3) CFFs, observables and the deconvolution problem

Getting started

- 2 out-of-the-box options: get started directly
  - Download a pre-configured virtual machine with an Eclipse IDE
  - **Download a pre-configured Docker image**
- Or compile directly the sources (better for high performance needs)

- 2 modes to run PARTONS:
  - Either using XML scripts as inputs (simple scenarios)
  - Or write your own C++ code (better for specific / high-performance needs)

Getting started (example docker macOS/ARM64)

- To run the docker image: install docker and then

```
docker pull partons/partons
docker run -it --platform linux/amd64 --rm partons/partons
```

- Our first run

```
cd partons-example
bin/PARTONS_example data/examples/gpd/computeSingleKinematicsForGPD.xml
```

- Output

```
GPDResult ChannelType: UNDEFINED ComputationModuleName: GPDGK16
IndexId: -1

Kinematics: GPDKinematic ChannelType: UNDEFINED IndexId: -1
x: 0.1 [none] xi: 0.2 [none] t: -0.1 [GeV2] muF2: 2 [GeV2] muR2: 2
[GeV2]

Result: GPD H
g: 1.21356419772346
u: 5.10001896395725 u(+): 5.01601340863696 u(-): 5.18402451927754
d: 3.25071439632003 d(+): 3.71356261658888 d(-): 2.78786617605118
s: 0.532687756322447 s(+): 1.06537551264489 s(-): 0

Result: GPD E
g: 0.394675181022725
u: 2.82293014585501 u(+): 0.537747853124165 u(-): 5.10811243858586
d: -5.48036520904456 d(+): -4.38146363426515 d(-):
-6.57926678382397
s: -0.855746238742706 s(+): -1.71149247748541 s(-): 0

Result: GPD Ht
g: 0.196713846436613
u: 1.78308763860078 u(+): 2.27315582319 u(-): 1.29301945401156
d: -0.755350623497139 d(+): -1.00225457051068 d(-):
-0.508446676483599
s: 0 s(+): 0 s(-): 0

Result: GPD Et
g: 0
u: 23.1801375984247 u(+): 33.2703905224653 u(-): 13.0898846743841
d: 6.62289645669277 d(+): 9.50582586356152 d(-): 3.73996704982403
s: 0 s(+): 0 s(-): 0
```

- Let’s add a basic editing tool in the container

```
apt-get update
apt-get install nano -y

nano data/examples/gpd/
computeSingleKinematicsForGPD.xml
```

- This code evaluates the Goloskokov-Kroll (GK) model at a given kinematic value

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!--
This scenario demonstrates a simple task as the evaluation of GPD model in a single kinematic point. The result of this
-->

<!-- Scenario starts here -->
<!-- For your convenience and for bookkeeping provide creation date and unique description -->
<scenario date="2017-07-18" description="GPD evaluation for single kinematics example">

   <!-- First task: evaluate GPD model for a single kinematics -->
   <!-- Indicate service and its methods to be used and indicate if the result should be stored in the database -->
   <task service="GPDService" method="computeSingleKinematic" storeInDB="0">

      <!-- Define GPD kinematics -->
      <kinematics type="GPDKinematic">
         <param name="x" value="0.1" />
         <param name="xi" value="0.2" />
         <param name="t" value="-0.1" />
         <param name="MuF2" value="2." />
         <param name="MuR2" value="2." />
      </kinematics>

      <!-- Define physics assumptions -->
      <computation_configuration>

         <!-- Select GPD model -->
         <module type="GPDModule" name="GPDGK16">
         </module>

      </computation_configuration>

   </task>

   <!-- Second task: print results of the last computation into standard output -->
   <task service="GPDService" method="printResults">
   </task>

</scenario>
```

Same result directly in C++ code

```
#include ...

int main(int argc, char** argv) {

    // Initialization Qt4
    QCoreApplication a(argc, argv);
    PARTONS::Partons* pPartons = 0;

    try {

        // Initialization PARTONS
        pPartons = PARTONS::Partons::getInstance();
        pPartons->init(argc, argv);

        // Create a GPDService that will handle the tasks for us
        PARTONS::GPDService* pGPDService =
                PARTONS::Partons::getInstance()->getServiceObjectRegistry()->getGPDService();

        // Create the GPDGK16 model
        PARTONS::GPDModule* pGPDModel =
                PARTONS::Partons::getInstance()->getModuleObjectFactory()->newGPDModule(
                        PARTONS::GPDGK16::classId);

        // Create a GPDKinematic(x, xi, t, MuF2, MuR2) to compute
        PARTONS::GPDKinematic gpdKinematic(0.1, 0.2, -0.1, 2., 2.);

        // Run computation
        PARTONS::GPDResult gpdResult = pGPDService->computeSingleKinematic(
                gpdKinematic, pGPDModel);

        // Print results
        PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__,
                gpdResult.toString());

        // Remove pointer references
        // Module pointers are managed by PARTONS
        PARTONS::Partons::getInstance()->getModuleObjectFactory()->updateModulePointerReference(
                pGPDModel, 0);
        pGPDModel = 0;

}
catch ...
```

1)

2)

3)

4)

5)

1) Initialization

2) Create a GPDService that will handle the tasks for us: loading the kinematics, evaluating the model, constructing the object containing the result.

3) Create the GPD model GK 16, an instance of GPDModule

4) Specify the kinematics into a GPDKinematic object, run the GPDService and print the results

5) Release the pointers and catch exceptions.

PARTONS uses a registry which allows to add new modules easily. The registry lists all available modules and attributes them a unique identifier. Then any instance of a module is created by an “object factory” cloning the object in the registry and returning a pointer.

When parsing the name of an object in an XML file for instance, the code knows what pointer to create immediately, or recognizes a typo and handles correctly the exception.

To evaluate the model on multiple kinematics at once

```xml
<!-- Define GPD kinematics -->
<kinematics type="GPDKinematic">
   <param name="x" value="0.1" />
   <param name="xi" value="0.2" />
   <param name="t" value="-0.1" />
   <param name="MuF2" value="2." />
   <param name="MuR2" value="2." />
</kinematics>
```

```xml
<!-- Define GPD kinematics -->
<kinematics type="GPDKinematic">

   <!-- Path to file defining kinematics -->
   <param name="file" value="kinematics_gpd.csv" />
</kinematics>
```

```
bin/PARTONS_example data/examples/gpd/computeManyKinematicsForGPD.xml
(Modify the path of the CSV file)
```

- To output data in the C++ code, it is useful to use the Logger instead of std::cout to not slow down computations and not interfere with the output of other threads.

```
PARTONS::List<PARTONS::GPDKinematic> gpdKinematicList;
for (int  i = 0; i < nbre_x; i++){
        // Logarithmic grid in x with nbre_x points from min_x to 1
        gpdKinematicList.add( PARTONS::GPDKinematic(exp(-log(min_x) / (nbre_x - 1) * i + log(min_x)), xi, t, mu2, mu2) );
}
// Run computation
PARTONS::List<PARTONS::GPDResult> gpdResultList =
        pGPDService->computeManyKinematic(gpdKinematicList, pGPDModel);
// We form the output as a list of values of x and H^u directly readable by Python
std::string Hu = "[", grid_x = "[";
for (int  i = 0; i < nbre_x; i++){
        Hu += gpdResultList[i].getPartonDistribution(PARTONS::GPDType::H).
                        getQuarkDistribution(PARTONS::QuarkFlavor::UP).toString() + ", ";
        grid_x += gpdResultList[i].getKinematic().getX().toString() + ", ";
}
Hu += "]";
grid_x += "]";

PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__, grid_x);
PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__, Hu);
```

```
cd build
cmake -G"Unix Makefiles" ../ -DCMAKE_BUILD_TYPE=Debug
make
```

- Result: GK 16, x_min = 1e-4, nbre_x = 100, xi = 1e-2, t = -0.1 GeV2, mu2 = 2 GeV2

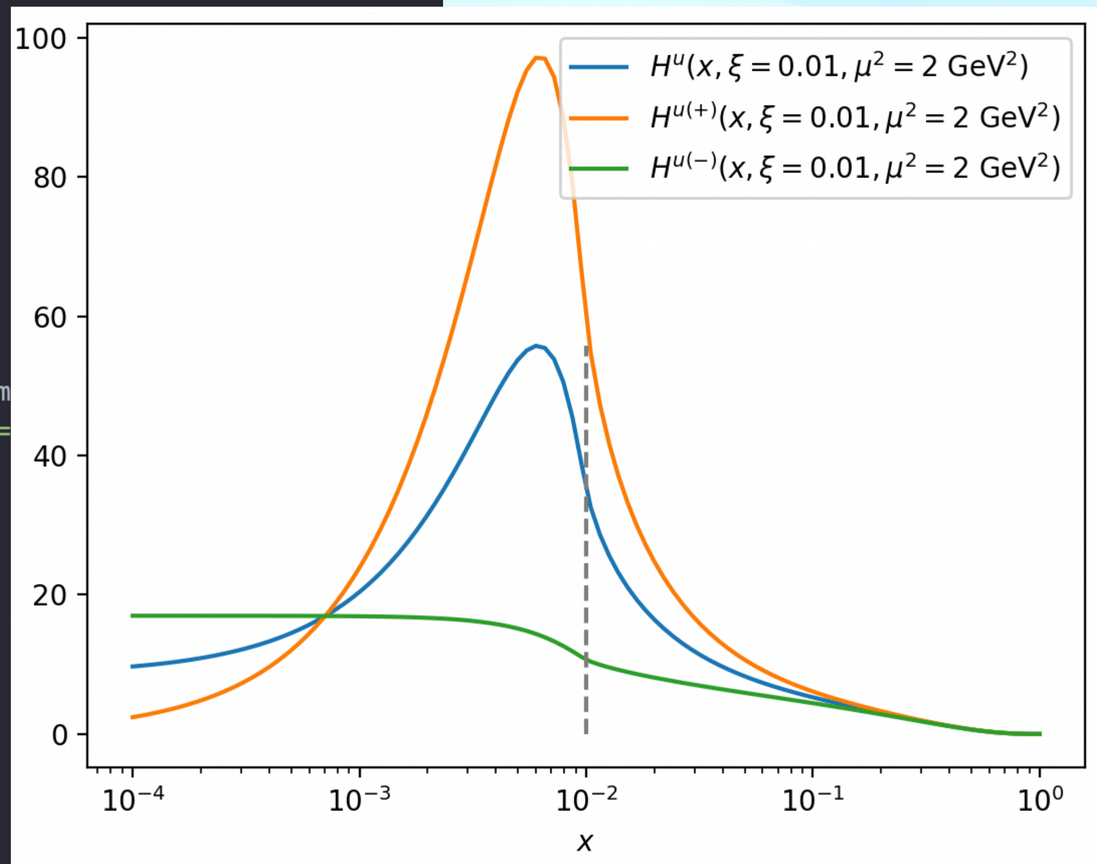
```
grid_x = [0.00040370172585965 6 [none], 0.0011233240329780 3 [none], 0.0033516026509388 5
[none], 0.0030538555088334 2 [none], 0.0027825594022071 3 [none], 0.0025353644939701 1
[none], 0.0023101297000831 6 [none], 0.0021049041445120 2 [none], 0.0019179102616724 9
[none], 0.0017475284000076 8 [none], 0.0015922827933410 9 [none], 0.0014508287784959 4
[none], 0.0013219411484660 3 [none], 0.0036783797718286 4 [none], 0.038535285937105 3
[none], ……
```

```
Hu = [u: 13.338457089619 4 u(+): 9.7111155258891 4 u(-): 16.965798653349 6, u:
21.806904360292 5 u(+): 26.731157238854 7 u(-): 16.882651481730 3, u: 12.517989290197 4
u(+): 8.0663428515416 6 u(-): 16.969635728853, u: 12.161103123265 9 u(+):
7.3511253542577 5 u(-): 16.971080892274 1, u: 11.835695977119 4 u(+): 6.6991112128134 6
u(-): 16.972280741425 4, u: 11.539021994153 7 u(+): 6.1047670772050 4 u(-):
16.973276911102 4, u: 11.268567653528 7 u(+): 5.5630313350706 4 u(-): 16.974103971986 7, u:
11.022033829150 4 u(+): 5.0692770297672 4 u(-): 16.974790628533 5, u: 10.797318806297 6
u(+): 4.6192768986573 u(-): 16.975360713938 3, ……
```

- The code outputs the H^u GPD, but also the singlet component $H^{u(+)}(x) = H^u(x) - H^u(-x)$ and the non-singlet component $H^{u(-)}(x) = H^u(x) - H^u(-x)$

Let's do a bit of plotting! The most convenient is to fetch the previously computed datasets, and run a small Python script outside of the container.

```python
import matplotlib.pyplot as plt

x_grid = "[0.000403701725859656 [none], 0.00112332403297803 [none], 0.000335160265093885 [none], 0.00030538555
Hu = "[u: 13.3384570896194 u(+): 9.71111552588914 u(-): 16.9657986533496, u: 21.8069043602925 u(+): 26.7311572

def plot_data_from_PARTONS(x_grid, Hu, xi, mu2):
    # Cleaning up the data
    x_grid_clean = [float(d) for d in x_grid[1:].split(" [none], ")[:-1]]
    Hu_tmp = Hu[1:-3].replace('u:', ',').replace('u(+):', ',').replace('u(-):', ',').replace(', , ', ' , ')
    Hu_clean = [float(d) for d in Hu_tmp[2:].split(' , ')]
    # Separating u, u+ and u-
    u = Hu_clean[::3]
    uPlus = Hu_clean[1::3]
    uMinus = Hu_clean[2::3]
    # Sorting the data
    u = [x for _, x in sorted(zip(x_grid_clean, u))]
    uPlus = [x for _, x in sorted(zip(x_grid_clean, uPlus))]
    uMinus = [x for _, x in sorted(zip(x_grid_clean, uMinus))]
    x_grid_clean.sort()

    plt.plot(x_grid_clean, u, label=r"$H^u(x, \xi=$"+str(xi)+r"$, \mu^2 = $"+str(m
    plt.plot(x_grid_clean, uPlus, label=r"$H^{u(+)}(x, \xi=$"+str(xi)+r"$, \mu^2 =
    plt.plot(x_grid_clean, uMinus, label=r"$H^{u(-)}(x, \xi=$"+str(xi)+r"$, \mu^2
    plt.plot([xi, xi], [min(u), max(u)], "--", color="gray")
    plt.xlabel(r"$x$")
    plt.xscale("log")
    plt.legend()
    plt.show()



plot_data_from_PARTONS(x_grid, Hu, 1e-2, 2)
```



- Now we want to evolve the scale of the GPD. In momentum-space, we need to solve the integro-differential equation:

$$\frac{F^a(x,\xi,t,\mu)}{x^{p_a}} = \sum_{b=q,g,\ldots} \int_0^1 \frac{\mathrm{d}z}{x} \Gamma^{ab}\left(\frac{z}{x},\frac{\xi}{x};\mu_0,\mu\right) \frac{F^b(z,\xi,t,\mu_0)}{z^{p_b}}$$

This is a linear system in the GPDs (evolution mixes quark and gluon GPDs) at the initial scale mu_0. If the GPDs are discretized on a grid in x [finite element method], then the effect of evolution on the x-dependence at a given value of xi can be represented as a matrix product which is a very efficient operation.

- PARTONS is interfaced with the precise and highly efficient evolution code APFEL++

APFEL++

APFEL

IFUM-1015-FT
CERN-PH-TH/2013-209

APFEL: A PDF Evolution Library with QED corrections

Valerio Bertone[1], Stefano Carrazza[2] and Juan Rojo[1]

There is another “legacy” evolution code implemented in PARTONS under the name of Vinnikov (2006) which is not recommended. APFEL++ (2022) is rigorously tested to guarantee accurate results, see

Bertone et al, arXiv:2206.01412

- Enriching the previous XML scenario to include the evolution

```xml
<task service="GPDService" method="computeSingleKinematic" storeInDB="0">

   <!-- Define GPD kinematics -->
   <kinematics type="GPDKinematic">
      <param name="x" value="0.1" />
      <param name="xi" value="0.2" />
      <param name="t" value="-0.1" />
      <param name="MuF2" value="40." />
      <param name="MuR2" value="40." />
   </kinematics>

   <!-- Define physics assumptions -->
    <computation_configuration>

      <!-- Select GPD model -->
      <module type="GPDModule" name="GPDGK16">

         <!-- Select GPD evolution model -->
         <module type="GPDEvolutionModule" name="GPDEvolutionApfel">

            <!-- Select pQCD order -->
            <param name="qcd_order_type" value="LO" />

            <!-- Select alpha_s model -->
            <module type="RunningAlphaStrongModule" name="RunningAlphaStrongApfel">
               <param name="qcd_order_type" value="LO" />
               <param name="alphasRef" value="0.118" />
               <param name="muRef" value="91.1876" />
               <param name="thresholds" value="0 0 0 1.3 4.75 172" />
            </module>

            <!-- Select model defining number of active flavors -->
            <module type="ActiveFlavorsThresholdsModule" name="ActiveFlavorsThresholdsQuarkMasses">
            </module>

         </module>

      </module>

   </computation_configuration>

</task>
```

At this stage, if xi != 0, only LO evolution is available. For xi = 0, the DGLAP equation is available up to NNLO.

The value of Lambda_QCD and the thresholds for quark masses can be varied straightforwardly.

```
bin/PARTONS_example data/examples/
evolution/
makeUseOfGPDEvolutionApfel.xml
(Modify
ActiveFlavorsThresholdsQuarkMasses to
ActiveFlavorsThresholdsConstant)
```

- But wait a minute! We can evaluate the GK model directly at a different scale without needing to import the evolution module…

```
x: 0.01 [none] xi: 0.01 [none] t: -0.1 [GeV2] muF2: 40 [GeV2] muR2: 40 [GeV2]

Result: GPD H
g: 5.12622238163934
u: 52.0595559616688 u(+): 90.3430850517394 u(-): 13.7760268715981
d: 47.8319258150115 d(+): 86.1154549050822 d(-): 9.54839672494085
s: 29.23601946514 s(+): 58.4720389302799 s(-): 0
```

GK model evaluated at a higher scale

```
x: 0.01 [none] xi: 0.01 [none] t: -0.1 [GeV2] muF2: 40 [GeV2] muR2: 40 [GeV2]

Result: GPD H
g: 5.63930118164917
u: 56.9566825556237 u(+): 98.9122054759322 u(-): 15.0011596353152
d: 50.5057804679107 d(+): 92.5600953805613 d(-): 8.45146555526005
s: 32.1488678080864 s(+): 64.2977356161727 s(-): 0
c: -1.42845833009132e-15 c(+): -3.95107879148357e-15 c(-): 1.09416213130094e-15
b: -5.06249897468583e-15 b(+): -7.70869335000798e-15 b(-): -2.41630459936368e-15
t: 0 t(+): 0 t(-): 0
```

GK model evolved from the reference scale of 4 GeV^2
to a higher scale with LO evolution n_f = 3

- The GK model implements its own evolution, so can be evaluated directly at any scale. Most other models are only defined at one reference scale, and then need to be evolved.

- What's going on behind the scenes?
  APFEL constructs a superposition of 4 logarithmic grids in x: 100 points in [1e-7, 1] + 60 points in [0.1, 1] + 50 points in [0.6, 1] + 50 points in [0.8, 1] with 3rd degree interpolation, and solves the differential equation with a Runge-Kutta technique on a range in mu from 1 to 1000 GeV.

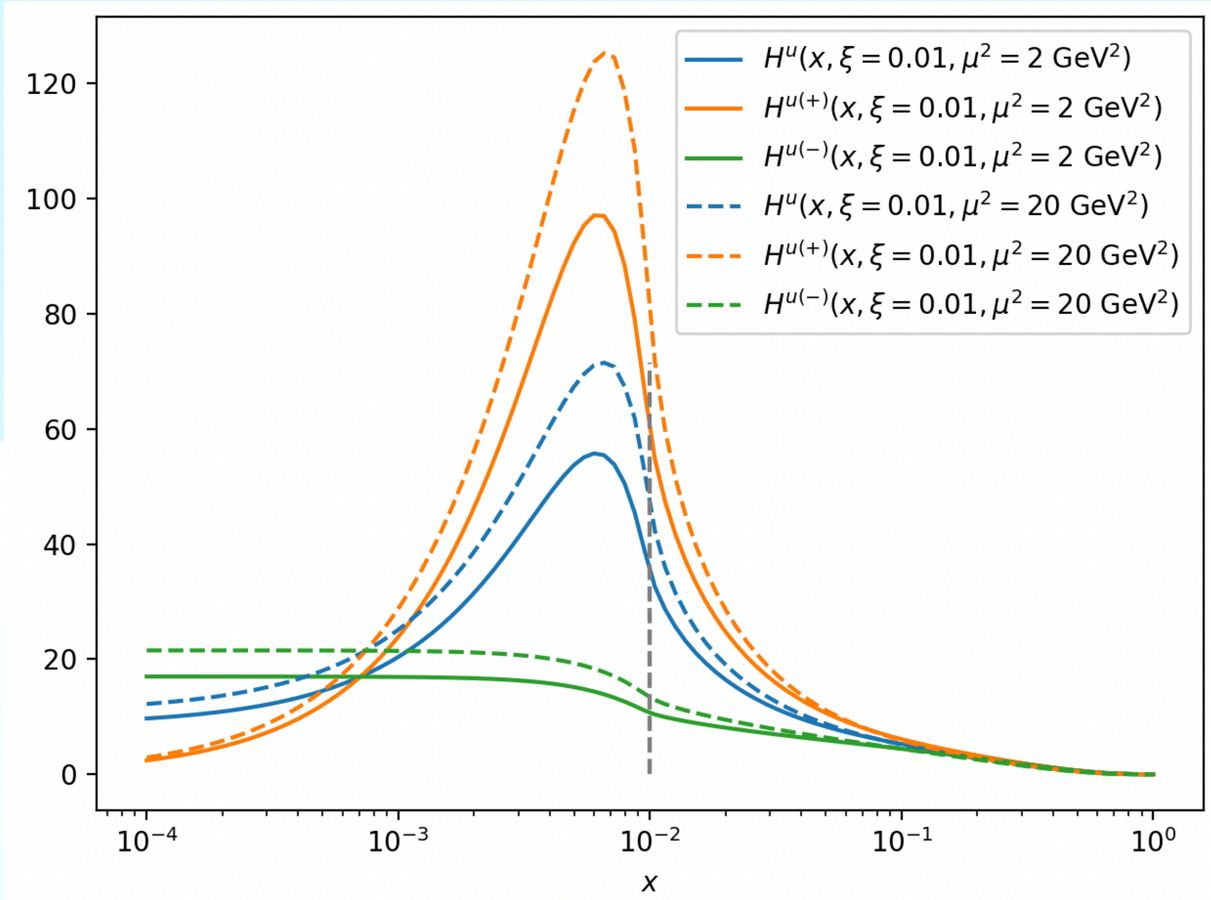
- For every new value of xi, we need to redo the calculation (or construct an interpolating scheme in xi), but in principle, all the necessary calculation is done for whatever (x, t, mu^2)

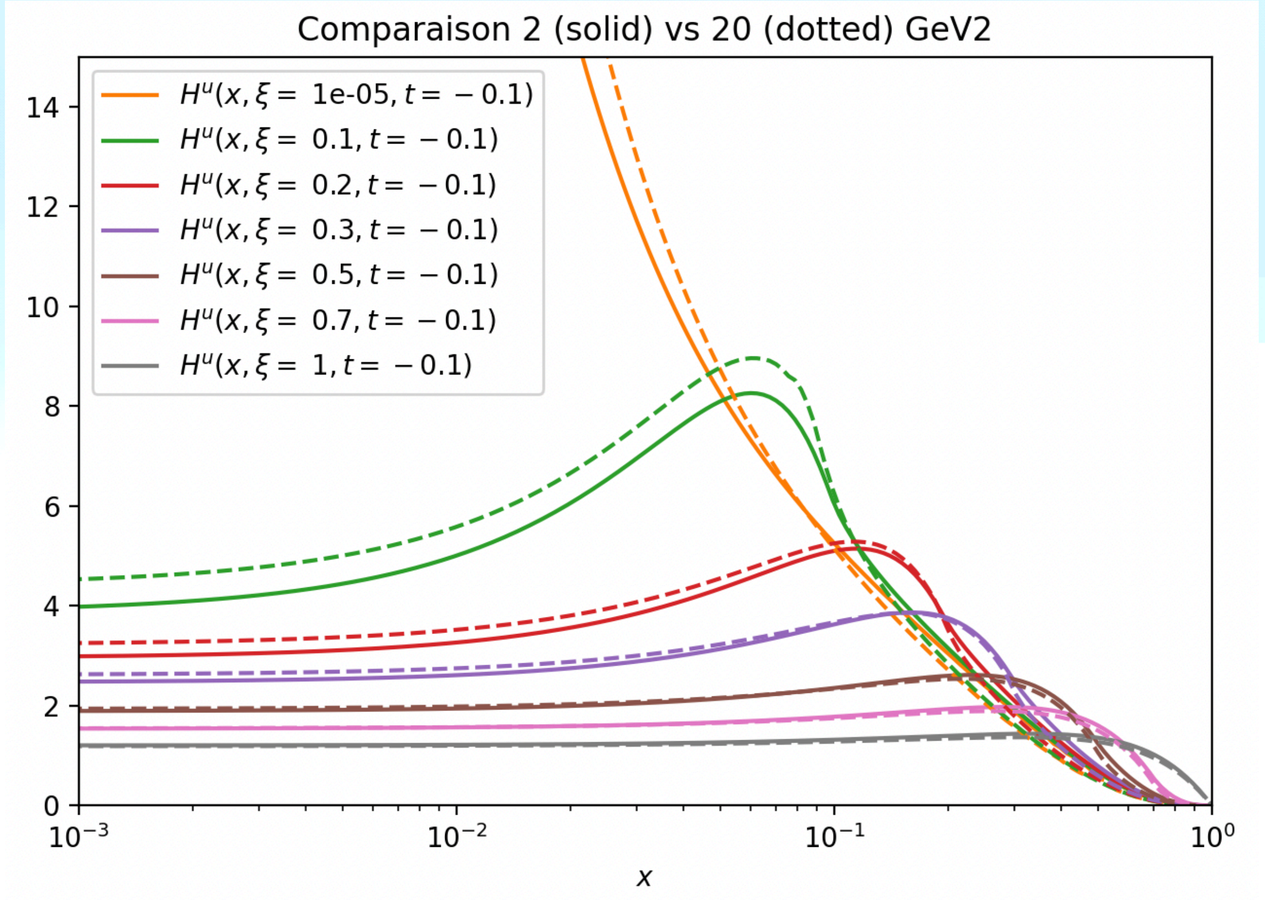
- For high performance use of evolution, it is possible to manipulate directly the APFEL objects inside the PARTONS code.

```cpp
std::map<int, double> GPDModel(double const& x, double const&)
{
  // Retrieve GPD service
  PARTONS::GPDService* pGPDService =
          PARTONS::Partons::getInstance()->getServiceObjectRegistry()->getGPDService();

  // Create GPD module with the BaseModuleFactory
  PARTONS::GPDModule* pGPDModel =
          PARTONS::Partons::getInstance()->getModuleObjectFactory()->newGPDModule(
                  PARTONS::GPDGK16::classId);

  // Create a GPDKinematic(x, xi, t, MuF2, MuR2) to compute
  const double xi = 1e-2;
  PARTONS::GPDKinematic gpdKinematic(x, xi, -0.1, 4., 4.);
  // Run computation
  PARTONS::GPDResult gpdResult = pGPDService->computeSingleKinematic(
          gpdKinematic, pGPDModel);

  // Call all functions only once.
  const double uplus = x * gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getQuarkDistribution(PARTONS::QuarkFlavor::UP).getQuarkDistributionPlus();
  const double uminus = x * gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getQuarkDistribution(PARTONS::QuarkFlavor::UP).getQuarkDistributionMinus();
  const double dplus = x * gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getQuarkDistribution(PARTONS::QuarkFlavor::DOWN).getQuarkDistributionPlus();
  const double dminus = x * gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getQuarkDistribution(PARTONS::QuarkFlavor::DOWN).getQuarkDistributionMinus();
  const double splus = x * gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getQuarkDistribution(PARTONS::QuarkFlavor::STRANGE).getQuarkDistributionPlus();
  const double sminus = x * gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getQuarkDistribution(PARTONS::QuarkFlavor::STRANGE).getQuarkDistributionMinus();

  const double upv  = uminus;
  const double dnv  = dminus;
  const double glu  = gpdResult.getPartonDistribution(PARTONS::GPDType::H).
              getGluonDistribution().getGluonDistribution();
  const double dbar = 0.5 * (dplus - dminus);
  const double ubar = 0.5 * (uplus - uminus);
  const double sbar = 0.5 * (splus - sminus);

  // Construct QCD evolution basis conbinations.
  double const Gluon   = glu;
  double const Singlet = dnv + 2 * dbar + upv + 2 * ubar + 2 * sbar;
  double const T3      = upv + 2 * ubar - dnv - 2 * dbar;
  double const T8      = upv + 2 * ubar + dnv + 2 * dbar - 4 * sbar;
  double const Valence = upv + dnv;
  double const V3      = upv - dnv;

  // Fill in map in the QCD evolution basis.
  std::map<int, double> QCDEvMap;
  QCDEvMap[0]  = Gluon;
  QCDEvMap[1]  = Singlet;
  QCDEvMap[2]  = Valence;
  QCDEvMap[3]  = T3;
  QCDEvMap[4]  = V3;
  QCDEvMap[5]  = T8;
  QCDEvMap[6]  = Valence;
  QCDEvMap[7]  = Singlet;
  QCDEvMap[8]  = Valence;
  QCDEvMap[9]  = Singlet;
  QCDEvMap[10] = Valence;
  QCDEvMap[11] = Singlet;
  QCDEvMap[12] = Valence;

  return QCDEvMap;
}
```

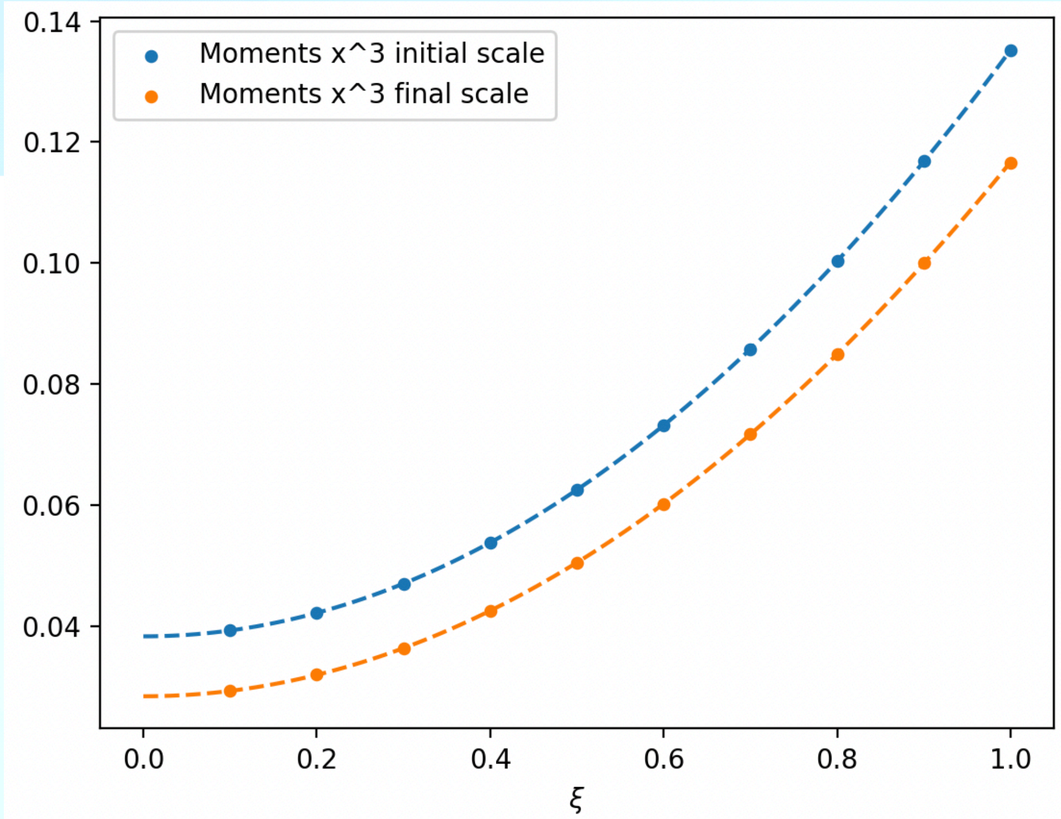
```cpp
void using_GPDEvolution_high_perf(double mu2F, double mu2I, double min_x = 1e-4, int nbre_x = 100) {
    // Choice of grid
    const double xi = 1e-2;
    const apfel::Grid g{{apfel::SubGrid{100, 1e-7, 3}, apfel::SubGrid{50, 1e-1, 5}, apfel::SubGrid{40, 8e-1, 5}}};

    // Running of the coupling
    const std::vector<double> Thresholds = {0, 0, 0, 1.4, 4.75, 175};
    const int PerturbativeOrder = 0;
    const double AlphaQCDRef = 0.118;
    const double MuAlphaQCDRef = 91.1876;
    apfel::AlphaQCD a{AlphaQCDRef, MuAlphaQCDRef, Thresholds, PerturbativeOrder};
    const apfel::TabulateObject<double> Alphas{a, 100, 0.9, 1001, 3};
    const auto as = [&] (double const& mu) -> double{ return Alphas.Evaluate(mu); };

    const auto GpdObj   = InitializeGpdObjects(g, Thresholds, xi);
    const auto EvolvedGPDs = BuildDglap(GpdObj, GPDModel, sqrt(mu2I), PerturbativeOrder, as);
    const apfel::TabulateObject<apfel::Set<apfel::Distribution>> TabulatedGPDs{*EvolvedGPDs, 30, 1, 100, 3};

    // Evaluates the GPD at mu2F (can evaluate at many values simultaneously)
    const std::map<int, apfel::Distribution> gpds = apfel::QCDEvToPhys(TabulatedGPDs.Evaluate(sqrt(mu2F)).GetObjects());
    // Constructs the result of the GPD
    std::string Hu = "[", grid_x = "[";
    for (int  i = 0; i < nbre_x; i++){
            // Logarithmic grid in x with nbre_x points from min_x to 1
            double x = exp(-log(min_x) / (nbre_x - 1) * i + log(min_x));
            grid_x += std::to_string(x) + " [none], ";
            double u = (gpds.at(2).Evaluate(x)) / x;
            double uMinus = (gpds.at(2).Evaluate(x) - gpds.at(-2).Evaluate(x)) / x;
            double uPlus = (gpds.at(2).Evaluate(x) + gpds.at(-2).Evaluate(x)) / x;
            Hu += "u: " + std::to_string(u) + " u(+): " + std::to_string(uPlus) + " u(-): " + std::to_string(uMinus) + ", ";
    }
    Hu += "]";
    grid_x += "]";

    PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__, grid_x);
    PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__, Hu);
}
```

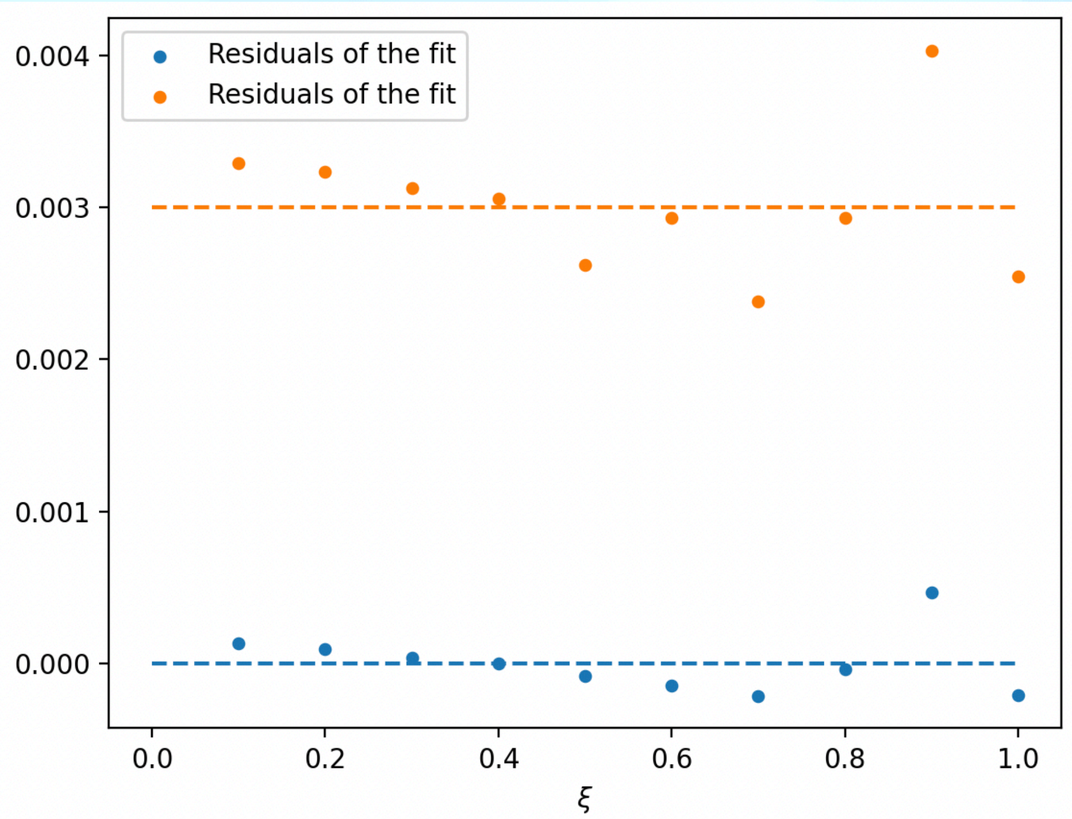
- Evolution takes less than a second (and we are running on an underperforming container).





- Let’s measure how well polynomiality is preserved under evolution:
  - At initial scale, the violation of polynomiality is dispersed with a spread ~0.02% (effect of discretizing the GPD with 300 points + integral only up to 1e-5)
  - At final scale, the violation of polynomiality is dispersed with ~0.04%: increase comparable to a second layer of discretization. In fact, running with finer evolution grids does not change the dispersion: the violation results mostly either from the computation of the moments (300 points + 1e-5) or from the true evolution of slight uncertainties in the model.





Outline

- 1) What is PARTONS and what can it offer to you?
- 2) GPD models and evolution
- 3) CFFs, observables and the deconvolution problem

- PARTONS implements a plethora of observables and modules related to the following exclusive processes: DVCS, DVMP, DDVCS, TCS, exclusive diphoton production

Database (PARTONS)
DatabaseFileObject (PARTONS)
DatabaseManager (PARTONS)
DateTimeUtils (PARTONS)
DDVCSAluPhi (PARTONS)
DDVCSAluPhiL (PARTONS)
DDVCSAluPhiLParameters (PARTONS)
DDVCSAluPhiParameters (PARTONS)
DDVCSCFFStandard (PARTONS)
DDVCSCFFTEST
DDVCSConvolCoeffFunctionKinematic (PARTONS)
DDVCSConvolCoeffFunctionKinematicDao (PARTONS)
DDVCSConvolCoeffFunctionKinematicDaoService (PARTONS)
DDVCSConvolCoeffFunctionModule (PARTONS)
DDVCSConvolCoeffFunctionResult (PARTONS)
DDVCSConvolCoeffFunctionResultDao (PARTONS)
DDVCSConvolCoeffFunctionResultDaoService (PARTONS)
DDVCSConvolCoeffFunctionService (PARTONS)
DDVCSCrossSectionTotal (PARTONS)
DDVCSCrossSectionTotalParameters (PARTONS)
DDVCSCrossSectionUUMinus (PARTONS)
DDVCSCrossSectionUUMinusDVCSLimit (PARTONS)
DDVCSCrossSectionUUMinusDVCSLimitParameters (PARTONS)
DDVCSCrossSectionUUMinusTCSLimit (PARTONS)
DDVCSCrossSectionUUMinusTCSLimitParameters (PARTONS)
DDVCSObservable (PARTONS)
DDVCSObservableKinematic (PARTONS)
DDVCSObservableKinematicDao (PARTONS)
DDVCSObservableKinematicDaoService (PARTONS)
DDVCSObservableResult (PARTONS)
DDVCSObservableResultDao (PARTONS)
DDVCSObservableResultDaoService (PARTONS)
DDVCSObservableService (PARTONS)
DDVCSProcessDMSW22 (PARTONS)
DDVCSProcessModule (PARTONS)
DDVCSScalesModule (PARTONS)
DDVCSScalesTEST
DDVCSScalesVirtualitiesSum (PARTONS)
DDVCSXiConverterModule (PARTONS)
DDVCSXiConverterTEST
DDVCSXiConverterTNeglected (PARTONS)
DefaultXMLParser (PARTONS)
Double (PARTONS)
DVCSAc (PARTONS)
DVCSAcCos0Phi (PARTONS)

DVCSAutDVCSSinPhiMPhis (PARTONS)
DVCSAutDVCSSinPhiMPhisCos0Phi (PARTONS)
DVCSAutIntSinPhiMPhis (PARTONS)
DVCSAutIntSinPhiMPhisCos0Phi (PARTONS)
DVCSAutIntSinPhiMPhisCos1Phi (PARTONS)
DVCSAutIntSinPhiMPhisSin1Phi (PARTONS)
DVCSAutMinusSinPhiMPhis (PARTONS)
DVCSAutMinusSinPhiMPhisCos0Phi (PARTONS)
DVCSCFFConstant (PARTONS)
DVCSCFFDispersionRelation (PARTONS)
DVCSCFFHeavyQuark (PARTONS)
DVCSCFFNN (PARTONS)
DVCSCFFStandard (PARTONS)
DVCSConvolCoeffFunctionKinematic (PARTONS)
DVCSConvolCoeffFunctionKinematicDao (PARTONS)
DVCSConvolCoeffFunctionKinematicDaoService (PARTONS)
DVCSConvolCoeffFunctionModule (PARTONS)
DVCSConvolCoeffFunctionResult (PARTONS)
DVCSConvolCoeffFunctionResultDao (PARTONS)
DVCSConvolCoeffFunctionResultDaoService (PARTONS)
DVCSConvolCoeffFunctionService (PARTONS)
DVCSCrossSectionDifferenceLUMinus (PARTONS)
DVCSCrossSectionTotal (PARTONS)
DVCSCrossSectionTotalParameters (PARTONS)
DVCSCrossSectionUUBHSubProc (PARTONS)
DVCSCrossSectionUUDVCSSubProc (PARTONS)
DVCSCrossSectionUUDVCSSubProcPhiIntegrated (PARTONS)
DVCSCrossSectionUUMinus (PARTONS)
DVCSCrossSectionUUMinusPhiIntegrated
DVCSCrossSectionUUVirtualPhotoProduct
DVCSCrossSectionUUVirtualPhotoProduct
(PARTONS)
DVCSObservable (PARTONS)
DVCSObservableKinematic (PARTONS)
DVCSObservableKinematicDao (PARTONS
DVCSObservableKinematicDaoService (PA
DVCSObservableResult (PARTONS)
DVCSObservableResultDao (PARTONS)
DVCSObservableResultDaoService (PART
DVCSObservableService (PARTONS)
DVCSProcessBMJ12 (PARTONS)
DVCSProcessGV08 (PARTONS)
DVCSProcessModule (PARTONS)
DVCSProcessVGG99 (PARTONS)
DVCSScalesModule (PARTONS)
DVCSScalesQ2Multiplier (PARTONS)

DVCSAcCos1Phi (PARTONS)
DVCSAcCos2Phi (PARTONS)
DVCSAcCos3Phi (PARTONS)
DVCSAllMinus (PARTONS)
DVCSAllMinusCos0Phi (PARTONS)
DVCSAllMinusCos1Phi (PARTONS)
DVCSAllMinusCos2Phi (PARTONS)
DVCSAllPlus (PARTONS)
DVCSAllPlusCos0Phi (PARTONS)
DVCSAllPlusCos1Phi (PARTONS)
DVCSAllPlusCos2Phi (PARTONS)
DVCSAltDVCSCosPhiMPhis (PARTONS)
DVCSAltDVCSCosPhiMPhisCos0Phi (PARTONS)
DVCSAltDVCSCosPhiMPhisCos1Phi (PARTONS)
DVCSAltDVCSSinPhiMPhis (PARTONS)
DVCSAltDVCSSinPhiMPhisSin1Phi (PARTONS)
DVCSAltIntCosPhiMPhis (PARTONS)
DVCSAltIntCosPhiMPhisCos0Phi (PARTONS)
DVCSAltIntCosPhiMPhisCos1Phi (PARTONS)
DVCSAltIntCosPhiMPhisCos2Phi (PARTONS)
DVCSAltIntSinPhiMPhis (PARTONS)
DVCSAltIntSinPhiMPhisSin1Phi (PARTONS)
DVCSAltIntSinPhiMPhisSin2Phi (PARTONS)
DVCSAluDVCS (PARTONS)
DVCSAluDVCSSin1Phi (PARTONS)
DVCSAluInt (PARTONS)
DVCSAluIntSin1Phi (PARTONS)
DVCSAluIntSin2Phi (PARTONS)
DVCSAluMinus (PARTONS)
DVCSAluMinusSin1Phi (PARTONS)
DVCSAluPlus (PARTONS)
DVCSAulMinus (PARTONS)
DVCSAulMinusSin1Phi (PARTONS)
DVCSAulMinusSin2Phi (PARTONS)
DVCSAulMinusSin3Phi (PARTONS)
DVCSAulPlus (PARTONS)
DVCSAulPlusSin1Phi (PARTONS)
DVCSAulPlusSin2Phi (PARTONS)
DVCSAulPlusSin3Phi (PARTONS)

DVCSXiConverterModule (PARTONS)
DVCSXiConverterXBToXi (PARTONS)
DVMPAluMinus (PARTONS)
DVMPAluMinusCos0Phi (PARTONS)
DVMPAluMinusCos1Phi (PARTONS)
DVMPAluMinusCos2Phi (PARTONS)
DVMPAluuMinus (PARTONS)
DVMPAluuMinusSin1Phi (PARTONS)
DVMPAuluMinus (PARTONS)
DVMPAuluMinusSin1Phi (PARTONS)
DVMPAuluMinusSin2Phi (PARTONS)
DVMPAuluMinusSin3Phi (PARTONS)
DVMPCFFGK06 (PARTONS)
DVMPCFFGK06IntegrationParameters (PARTONS)
DVMPConvolCoeffFunctionKinematic (PARTONS)
DVMPConvolCoeffFunctionKinematicDao (PARTONS)
DVMPConvolCoeffFunctionKinematicDaoService (PARTONS)
DVMPConvolCoeffFunctionModule (PARTONS)
DVMPConvolCoeffFunctionResult (PARTONS)
DVMPConvolCoeffFunctionResultDao (PARTONS)
DVMPConvolCoeffFunctionResultDaoService (PARTONS)
DVMPConvolCoeffFunctionService (PARTONS)
DVMPCrossSectionTotal (PARTONS)
DVMPCrossSectionTotalParameters (PARTONS)
DVMPCrossSectionUUUMinus (PARTONS)
DVMPCrossSectionUUUMinusPhiIntegrated (PARTONS)
DVMPObservable (PARTONS)
DVMPObservableKinematic (PARTONS)
DVMPObservableKinematicDao (PARTONS)
DVMPObservableKinematicDaoService (PARTONS)
DVMPObservableResult (PARTONS)
DVMPObservableResultDao (PARTONS)
DVMPObservableResultDaoService (PARTONS)
DVMPObservableService (PARTONS)
DVMPProcessGK06 (PARTONS)
DVMPProcessModule (PARTONS)
DVMPScalesModule (PARTONS)
DVMPScalesQ2Multiplier (PARTONS)
DVMPXiConverterModule (PARTONS)
DVMPXiConverterXBToXi (PARTONS)

Task (PARTONS)
TCSAcu (PARTONS)
TCSAcuThetaIntegrated (PARTONS)
TCSAfb (PARTONS)
TCSAutCosPhiMPhis (PARTONS)
TCSAutCosPhiMPhisThetaIntegrated (PARTONS)
TCSAutSinPhiMPhis (PARTONS)
TCSAutSinPhiMPhisThetaIntegrated (PARTONS)
TCSCFFFromDVCS (PARTONS)
TCSCFFFromDVCSDifferentiationParameters (PARTONS)
TCSCFFFromDVCSOnlyNLOPart (PARTONS)
TCSCFFStandard (PARTONS)
TCSConvolCoeffFunctionKinematic (PARTONS)
TCSConvolCoeffFunctionKinematicDao (PARTONS)
TCSConvolCoeffFunctionKinematicDaoService (PARTONS)
TCSConvolCoeffFunctionModule (PARTONS)
TCSConvolCoeffFunctionResult (PARTONS)
TCSConvolCoeffFunctionResultDao (PARTONS)
TCSConvolCoeffFunctionResultDaoService (PARTONS)
TCSConvolCoeffFunctionService (PARTONS)
TCSCrossSectionDifferenceCU (PARTONS)
TCSCrossSectionDifferenceCUThetaIntegrated (PARTONS)
TCSCrossSectionTotal (PARTONS)
TCSCrossSectionTotalParameters (PARTONS)

TCSCrossSectionUU (PARTONS)
TCSCrossSectionUUThetaIntegrated (PARTONS)
TCSCrossSectionUUThetaPhiIntegrated (PARTONS)
TCSCrossSectionUUWeighted (PARTONS)
TCSCrossSectionUUWeightedThetaIntegrated (PARTONS)
TCSObservable (PARTONS)
TCSObservableKinematic (PARTONS)
TCSObservableKinematicDao (PARTONS)
TCSObservableKinematicDaoService (PARTONS)
TCSObservableResult (PARTONS)
TCSObservableResultDao (PARTONS)
TCSObservableResultDaoService (PARTONS)
TCSObservableService (PARTONS)
TCSProcessBDP01 (PARTONS)
TCSProcessBDPGW19 (PARTONS)
TCSProcessModule (PARTONS)
TCSR (PARTONS)
TCSScalesModule (PARTONS)
TCSScalesQ2PrimMultiplier (PARTONS)
TCSXiConverterExact (PARTONS)
TCSXiConverterModule (PARTONS)
TCSXiConverterTauToXi (PARTONS)
TDependentPDFKinematic (PARTONS)
ThreadManager (PARTONS)
ThreadQueue (PARTONS)

- Compton form factors (CFFs): Lorentz-invariant quantities that enter the description of DVCS observables and factorize in terms of the convolution of a GPD and a perturbative coefficient function.

Observables

beam-spin asymmetry: $$A_{LU}(\phi) = \frac{d\sigma^{\uparrow}(\phi) - d\sigma^{\downarrow}(\phi)}{d\sigma^{\uparrow}(\phi) + d\sigma^{\downarrow}(\phi)} \propto \mathfrak{Im}\left\{ F_1\mathcal{H} + \xi\left(F_1 + F_2\right)\widetilde{\mathcal{H}} - \frac{\Delta^2}{4M^2}F_2\mathcal{E} \right\} \sin(\phi)$$

target-spin asymmetry: $$A_{UL}(\phi) = \frac{d\sigma^{\Rightarrow}(\phi) - d\sigma^{\Leftarrow}(\phi)}{d\sigma^{\Rightarrow}(\phi) + d\sigma^{\Leftarrow}(\phi)} \propto \mathfrak{Im}\left[F_1\widetilde{\mathcal{H}} + \xi\left(F_1 + F_2\right)\left(\mathcal{H} + \frac{x_B}{2}\mathcal{E}\right) - \xi\left(\frac{x_B}{2}F_1 + \frac{t}{4M^2}F_2\right)\widetilde{\mathcal{E}} \right] \sin(\phi)$$

double spin asymmetry: $$A_{LL}(\phi) = \frac{d\sigma^{\uparrow\Uparrow}(\phi) - d\sigma^{\downarrow\Uparrow}(\phi) - d\sigma^{\uparrow\Downarrow}(\phi) + d\sigma^{\downarrow\Downarrow}(\phi)}{d\sigma^{\uparrow\Uparrow}(\phi) + d\sigma^{\downarrow\Uparrow}(\phi) + d\sigma^{\uparrow\Downarrow}(\phi) + d\sigma^{\downarrow\Downarrow}(\phi)} \propto \mathfrak{Re}\left[F_1\widetilde{\mathcal{H}} + \xi\left(F_1 + F_2\right)\left(\mathcal{H} + \frac{x_B}{2}\mathcal{E}\right) - \xi\left(\frac{x_B}{2}F_1 + \frac{t}{4M^2}F_2\right)\widetilde{\mathcal{E}} \right] \cos\phi + BH$$

beam-charge asymmetry: $$A_C = \frac{\sigma^{+}_{UU} - \sigma^{-}_{UU}}{\sigma^{+}_{UU} + \sigma^{-}_{UU}} \propto \mathfrak{Re}\left[F_1\mathcal{H} + \xi\left(F_1 + F_2\right)\widetilde{\mathcal{H}} - \frac{\Delta^2}{4M^2}F_2\mathcal{E} \right] \cos\phi$$

Marija Čuić, Monday's lecture on GPD analysis

$$\mathcal{H}(\xi, t, Q^2) = \int_{-1}^{1} dx \sum_a C^a(x, \xi, Q^2, \mu^2)\, H^a(x, \xi, t; \mu^2)$$

Eric Moffat, Wednesday's lecture on the inverse problem and shadow GPDs

```
<!-- First task: evaluate DVCS CFF for a single kinematics -->
<!-- Indicate service and its methods to be used and indicate if the result should be stored in the database -->
<task service="DVCSConvolCoeffFunctionService" method="computeSingleKinematic" storeInDB="0">

   <!-- Define DVCS CFF kinematics -->
   <kinematics type="DVCSConvolCoeffFunctionKinematic">
      <param name="xi" value="0.01" />
      <param name="t" value="-0.1" />
      <param name="Q2" value="4." />
      <param name="MuF2" value="4." />
      <param name="MuR2" value="4." />
   </kinematics>

   <!-- Define physics assumptions -->
   <computation_configuration>

      <!-- Select DVCS CFF model -->
      <module type="DVCSConvolCoeffFunctionModule" name="DVCSCFFStandard">

         <!-- Indicate pQCD order of calculation -->
         <param name="qcd_order_type" value="LO" />

         <!-- Select GPD model -->
         <module type="GPDModule" name="GPDGK16">
         </module>

      </module>

   </computation_configuration>

</task>

<!-- Second task: print results of the last computation into standard output -->
<task service="DVCSConvolCoeffFunctionService" method="printResults">
</task>
```

```
19-09-2024 01:51:11 [WARN] (DExpIntegrator1D::integrate) Cannot reach tolerances !
19-09-2024 01:51:11 [WARN] (DExpIntegrator1D::integrate) Cannot reach tolerances !
19-09-2024 01:51:11 [WARN] (DExpIntegrator1D::integrate) Cannot reach tolerances !
19-09-2024 01:51:11 [WARN] (DExpIntegrator1D::integrate) Cannot reach tolerances !
19-09-2024 01:51:11 [INFO] (DVCSConvolCoeffFunctionService::printResultListBuffer)
DVCSConvolCoeffFunctionResult ChannelType: DVCS ComputationModuleName: DVCSCFFStandard IndexId: -1

Kinematics: DVCSConvolCoeffFunctionKinematic ChannelType: DVCS IndexId: -1
xi: 0.01 [none] t: -0.1 [GeV2] muF2: 4 [GeV2] muR2: 4 [GeV2] Q2: 4 [GeV2]

Result:
CFF H: Re: 12.7069327914622 Im: 115.89613690083
CFF E: Re: -34.0324138944908 Im: -82.1656069304532
CFF Ht: Re: 4.41399082500419 Im: 5.8865393014224
CFF Et: Re: 1,458.08680425897 Im: 150.26977988776
```

```
<param name="integrator_type" value="GL" />
```

```
Result:
CFF H: Re: 12.643552126886 Im: 115.89613690083
CFF E: Re: -33.9857890972551 Im: -82.1656069304532
CFF Ht: Re: 4.41234822938851 Im: 5.8865393014224
CFF Et: Re: 1,458.04099271015 Im: 150.26977988776
```

Pick from:
- GL (Gauss-Legendre)
- DEXP
- GK21_ADAPTIVE (Gauss-Kronrod)
- TRAPEZOIDAL
- TRAPEZOIDALLOG

$$\Re\mathcal{H}^q(\xi) \overset{\text{LO}}{=} e_q^2 \fint_{-1}^{1} H^q(x,\xi) \left[\frac{1}{\xi - x} - \frac{1}{\xi + x}\right] \mathrm{d}x,$$

$$\Im\mathcal{H}^q(\xi) \overset{\text{LO}}{=} \pi e_q^2 \left[H^q(\xi,\xi) - H^q(-\xi,\xi)\right],$$

- Let’s play with a shadow GPD model. The publication **Bertone, HD, Mezrag, Moutarde, Sznajder, *Eur.Phys.J.C* 81 (2021) 4, 300** was accompanied by a PARTONS module: **GPDBDMMS21**

Cf. Eric Moffat, Wednesday’s lecture on the inverse problem and shadow GPDs

- Beware, numerically unstable at xi > 0.8 because of a required perfect cancellation between the numerator and denominator

LO CFF

```
xi: 0.5 [none] t: -0.1 [GeV2] muF2: 1 [GeV2] muR2: 1 [GeV2] Q2: 1 [GeV2]

Result:
CFF H: Re: -1.070649428966 06e-08 Im: 0
```

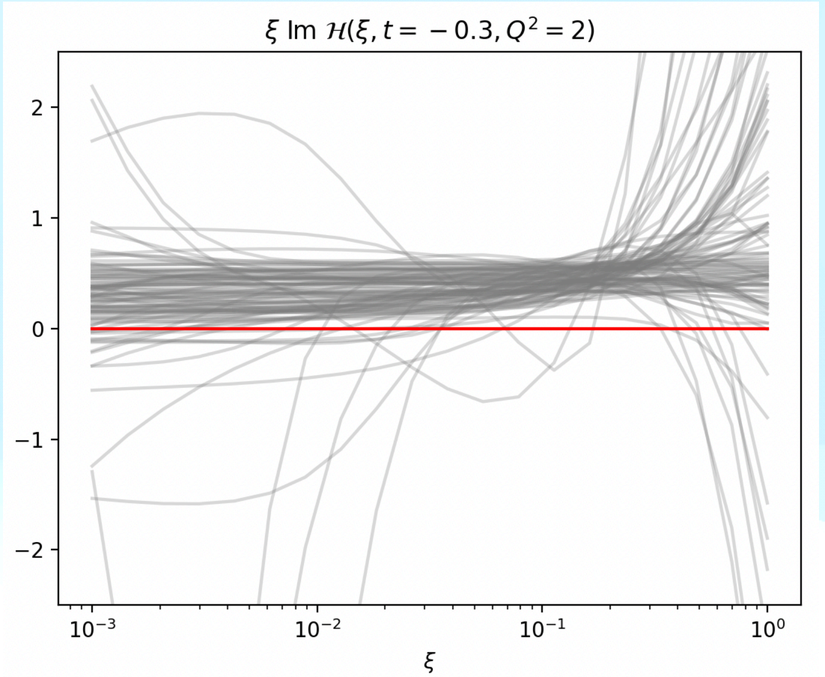
NLO CFF

```
CFF H: Re: -4.08166327256 14e-09 Im: -8.104128819653 99e-09
```

```
trans = pow(m_xi, 2) * (pow(absX, 2) - pow(m_xi, 2)) * pow(absX - 1, 5) .09946851 * pow(10., -8) * absX + 9.8905768270549734256 * pow(10., -8) * pow(absX, 2) +
5.4207095851873525136 * pow(10., -6) * pow(absX, 3) + 0.000026312301779 absX, 4) - 0.0021598622937353977786 * pow(absX, 5) - 0.011008702138310537654 * pow(absX, 6) -
0.56305855622681362802 * pow(absX, 7) + 7.0072252445919459353 * pow(absX
11) - 97.011276613079764504 * pow(absX, 12) - 181.72143366229912121 * p
pow(absX, 16) - 56.114891535498807746 * pow(absX, 17) - 39.657921691446
2) - 0.00002289981298652957 1349 * pow(absX, 2) * pow(m_xi, 2) + 0.00271
0.28836205489179660897 * pow(absX, 5) * pow(m_xi, 2) + 10.591771592324
pow(absX, 8) * pow(m_xi, 2) + 1157.8242830169957934 * pow(absX, 9) * po
728.14359319075315680 * pow(absX, 12) * pow(m_xi, 2) - 2983.40275031890
pow(absX, 15) * pow(m_xi, 2) + 639.64983674072894688 * pow(absX, 16) *
- 3.1726337353156862355 * pow(absX, 19) * pow(m_xi, 2) - 0.00079628510480319772166 * absX * pow(m_xi, 4) - 0.0039814255240159886083 * pow(absX, 2) * pow(m_xi, 4) + 0.97810774434163170660 *
pow(absX, 3) * pow(m_xi, 4) - 18.553693906623295123 * pow(absX, 4) * pow(m_xi, 4) + 80.019322701584110464 * pow(absX, 5) * pow(m_xi, 4) + 205.57669342342535603 * pow(absX, 6) * pow(m_xi, 4) -
1108.9720925829313412 * pow(absX, 7) * pow(m_xi, 4) - 5292.451535556220 * pow(m_xi, 4) + 32931.486866169982130 * pow(absX, 9) * pow(m_xi, 4) - 55942.856316219247591 *
pow(absX, 10) * pow(m_xi, 4) + 25457.226013562047867 * pow(absX, 11) * .022390386173170 * pow(absX, 12) * pow(m_xi, 4) - 5965.5382819086752892 * pow(absX, 13) * pow(m_xi, 4)
- 5267.5836235431551521 * pow(absX, 14) * pow(m_xi, 4) - 681.2977787967 15) * pow(m_xi, 4) + 231.40303918672976096 * pow(absX, 16) * pow(m_xi, 4) + 44.870548399427869420 *
pow(absX, 17) * pow(m_xi, 4) - 0.88128714869880173208 * pow(absX, 18) * 7(m_xi, 6) +
1.9943886251995084972 * pow(absX, 2) * pow(m_xi, 6) + 44.423991242391992 01142 *
pow(absX, 5) * pow(m_xi, 6) + 10202.070562178728021 * pow(absX, 6) * pow(m_xi, 6) - 15249.753237475805994 * pow(absX, 7) * pow(m_xi, 6) - 57095.368928101023144 * pow(absX, 8) * pow(m_xi, 6) +
192110.31232352455849 * pow(absX, 9) * pow(m_xi, 6) - 179121.24408611189640 * pow(absX, 10) * pow(m_xi, 6) + 18406.489385218757264 * pow(absX, 11) * pow(m_xi, 6) + 31455.322140932687386 *
pow(absX, 12) * pow(m_xi, 6) + 3515.3877982545091957 * pow(absX, 13) * pow(m_xi, 6) - 1712.5462403270528480 * pow(absX, 14) * pow(m_xi, 6) - 317.07878705916505185 * pow(absX, 15) * pow(m_xi, 6)
+ 15.894038128903448588 * pow(absX, 16) * pow(m_xi, 6) + 3.1788076257806897176 * pow(absX, 17) * pow(m_xi, 6) + 0.97024584933205212617 * pow(m_xi, 8) - 22.821221936891537592 * absX * pow(m_xi,
8) + 80.546663366834170180 * pow(absX, 2) * pow(m_xi, 8) + 461.56501780904158235 * pow(absX, 3) * pow(m_xi, 8) + 587.65889317797101130 * pow(absX, 4) * pow(m_xi, 8) - 30942.846621894570688 *
pow(absX, 5) * pow(m_xi, 8) + 116058.28987373951330 * pow(absX, 6) * pow(m_xi, 8) - 106136.24140963659717 * pow(absX, 7) * pow(m_xi, 8) - 172560.74571146452891 * pow(absX, 8) * pow(m_xi, 8) +
374932.30846467831723 * pow(absX, 9) * pow(m_xi, 8) - 175075.44024096178211 * pow(absX, 10) * pow(m_xi, 8) - 20654.346082834079890 * pow(absX, 11) * pow(m_xi, 8) + 8810.6358164337153885 *
pow(absX, 12) * pow(m_xi, 8) + 1573.7524068122202844 * pow(absX, 13) * pow(m_xi, 8) - 117.73422279657674585 * pow(absX, 14) * pow(m_xi, 8) - 23.546844559315349169 * pow(absX, 15) * pow(m_xi, 8)
+ 14.952833221954524282 * pow(m_xi, 10) - 56.315554285042311432 * absX * pow(m_xi, 10) - 1594.9038256576631971 * pow(absX, 2) * pow(m_xi, 10) + 9533.2068012653309691 * pow(absX, 3) * pow(m_xi,
10) + 8797.16745045814630 96 * pow(absX, 4) * pow(m_xi, 10) - 164654.50613438123593 * pow(absX, 5) * pow(m_xi, 10) + 378000.78902134162791 * pow(absX, 6) * pow(m_xi, 10) - 217468.08809982818278
* pow(absX, 7) * pow(m_xi, 10) - 217336.82472243109333 * pow(absX, 8) * pow(m_xi, 10) + 254648.05826653967989 * pow(absX, 9) * pow(m_xi, 10) - 38774.842040282214100 * pow(absX, 10) * pow(m_xi,
10) - 6902.9304818604595745 * pow(absX, 11) * pow(m_xi, 10) + 532.52370387248952846 * pow(absX, 12) * pow(m_xi, 10) + 106.50474077449790569 * pow(absX, 13) * pow(m_xi, 10) -
53.007998147416263552 * pow(m_xi, 12) + 2429.8528315988857087 * absX * pow(m_xi, 12) - 21351.325462984932197 * pow(absX, 2) * pow(m_xi, 12) + 60407.737431538303098 * pow(absX, 3) * pow(m_xi,
12) + 3402.5646475321875629 * pow(absX, 4) * pow(m_xi, 12) - 286936.74861146392456 * pow(absX, 5) * pow(m_xi, 12) + 428790.42237029855502 * pow(absX, 6) * pow(m_xi, 12) - 125978.514111274113 73
* pow(absX, 7) * pow(m_xi, 12) - 108872.51935779705046 * pow(absX, 8) * pow(m_xi, 12) + 44740.517983802130789 * pow(absX, 9) * pow(m_xi, 12) - 1859.3641735903611877 * pow(absX, 10) * pow(m_xi,
12) - 371.87283471807223753 * pow(absX, 11) * pow(m_xi, 12) - 853.29844833922602184 * pow(m_xi, 14) + 13236.674848582701410 * absX * pow(m_xi, 14) - 65719.669334875971679 * pow(absX, 2) *
pow(m_xi, 14) + 119859.88542902686035 * pow(absX, 3) * pow(m_xi, 14) - 10652.576069538706354 * pow(absX, 4) * pow(m_xi, 14) - 194199.58887279685824 * pow(absX, 5) * pow(m_xi, 14) +
164028.73526016797471 * pow(absX, 6) * pow(m_xi, 14) - 10607.750816680038307 * pow(absX, 7) * pow(m_xi, 14) - 11848.731198908898554 * pow(absX, 8) * pow(m_xi, 14) + 1601.1111621099985052 *
pow(absX, 9) * pow(m_xi, 14) - 2298.80210273164815 29 * pow(m_xi, 16) + 22302.756908501618752 * absX * pow(m_xi, 16) - 71566.303830252158958 * pow(absX, 2) * pow(m_xi, 16) +
82286.5187497520461 73 * pow(absX, 3) * pow(m_xi, 16) - 442.91596095874039645 * pow(absX, 4) * pow(m_xi, 16) - 47196.680019254635270 * pow(absX, 5) * pow(m_xi, 16) + 13994.198385616129654 *
pow(absX, 6) * pow(m_xi, 16) + 39.220299591306186264 * pow(absX, 7) * pow(m_xi, 16) - 145.98740448131537559 * pow(absX, 8) * pow(m_xi, 16) - 2275.5125529794829239 * pow(m_xi, 18) +
14444.913950371349354 * absX * pow(m_xi, 18) - 28770.250505356937508 * pow(absX, 2) * pow(m_xi, 18) + 17387.476362618452112 * pow(absX, 3) * pow(m_xi, 18) + 2446.0343387147832737 * pow(absX, 4)
* pow(m_xi, 18) - 2021.5120025470624955 * pow(absX, 5) * pow(m_xi, 18) + 134.62595327551924933 * pow(absX, 6) * pow(m_xi, 18) - 913.10064154886320225 * pow(m_xi, 20) + 3527.0113185998332592 *
absX * pow(m_xi, 20) - 3564.3766684119398832 * pow(absX, 2) * pow(m_xi, 20) + 495.42631216184088147 * pow(absX, 3) * pow(m_xi, 20) + 51.307015970142626343 * pow(absX, 4) * pow(m_xi, 20) -
139.70810880715583728 * pow(m_xi, 22) + 252.57036625594730332 * absX * pow(m_xi, 22) - 39.456318679884862690 * pow(absX, 2) * pow(m_xi, 22) - 6.1171615640755502789 * pow(m_xi, 24)) / pow(1 -
pow(m_xi, 2), 17);
```

- A 2019 analysis of the world DVCS dataset is available under the form of 100 neural network replicas in PARTONS. Cf. **Moutarde, Sznajder, Wagner, *Eur.Phys.J.C* 79 (2019) 7, 614**

```
void CFFNeuralNetwork() {

    // Retrieve service
    PARTONS::DVCSConvolCoeffFunctionService* pDVCSConvolCoeffFunctionService =
            PARTONS::Partons::getInstance()->getServiceObjectRegistry()->getDVCSConvolCoeffFunctionService();

    // Create CFF module with the BaseModuleFactory
    PARTONS::DVCSConvolCoeffFunctionModule* pDVCSCFFModule =
            PARTONS::Partons::getInstance()->getModuleObjectFactory()->newDVCSConvolCoeffFunctionModule(
                    PARTONS::DVCSCFFNN::classId);

    std::string res = "[";
    for(int i = 1; i < 101; i++){
        // Select the replica no. i
        ElemUtils::Parameters parameters(
                PARTONS::DVCSCFFNN::PARAMETER_NAME_REPLICA, i);
        pDVCSCFFModule->configure(parameters);

        // Create the list of kinematics
        PARTONS::List<PARTONS::DVCSConvolCoeffFunctionKinematic> cffKinematicList;
        int nbre_xi = 20;
        double min_xi = 1e-3;
        for (int  j = 0; j < nbre_xi; j++){
                // Logarithmic grid in x with 20 points from 1e-3 to 1
                cffKinematicList.add( PARTONS::DVCSConvolCoeffFunctionKinematic
                        (exp(-log(min_xi) / (nbre_xi - 1) * j + log(min_xi)), -0.3, 2., 2.) );
        }
        // Run computation
        PARTONS::List<PARTONS::DVCSConvolCoeffFunctionResult> cffResult =
                pDVCSConvolCoeffFunctionService->computeManyKinematic(
                        cffKinematicList, pDVCSCFFModule);
```




```
        res += "[";
        for (int  j = 0; j < nbre_xi; j++){
                res += "(" + std::to_string(cffResult[j].getKinematic().getXi().getValue()) + ", " +
                    std::to_string(cffResult[j].getResult(PARTONS::GPDType::H).imag())  + "), ";
        }
        res += "], ";
    }
    res += "]";

    // Print results for DVCSCFFModule
    PARTONS::Partons::getInstance()->getLoggerManager()->info("main", __func__,
            res);

    // Remove pointer references
    // Module pointers are managed by PARTONS
    PARTONS::Partons::getInstance()->getModuleObjectFactory()->updateModulePointerReference(
            pDVCSCFFModule, 0);
    pDVCSCFFModule = 0;
}
```

- Finally to compute observables, good old XML file

```xml
<!-- First task: evaluate DVCS observable for a single kinematics -->
<!-- Indicate service and its methods to be used and indicate if the result should be stored in the database -->
<task service="DVCSObservableService" method="computeSingleKinematic" storeInDB="0">

   <!-- Define DVCS observable kinematics -->
   <kinematics type="DVCSObservableKinematic">
      <param name="xB" value="0.2" />
      <param name="t" value="-0.1" />
      <param name="Q2" value="2." />
      <param name="E" value="6." />
      <param name="phi" value="0." />
   </kinematics>

   <!-- Define physics assumptions -->
   <computation_configuration>

      <!-- Select DVCS observable -->
      <module type="DVCSObservableModule" name="DVCSAllMinus">

         <!-- Select DVCS process model -->
         <module type="DVCSProcessModule" name="DVCSProcessGV08">

            <!-- Select scales module -->
            <!-- (it is used to evaluate factorization and renormalization scales out of kinematics) -->
            <module type="DVCSScalesModule" name="DVCSScalesQ2Multiplier">

               <!-- Configure this module -->
               <param name="lambda" value="1." />
            </module>

            <!-- Select xi-converter module -->
            <!-- (it is used to evaluate GPD variable xi out of kinematics) -->
            <module type="DVCSXiConverterModule" name="DVCSXiConverterXBToXi">
            </module>

            <!-- Select DVCS CFF model -->
            <module type="DVCSConvolCoeffFunctionModule" name="DVCSCFFStandard">

               <!-- Indicate pQCD order of calculation -->
               <param name="qcd_order_type" value="LO" />

               <!-- Select GPD model -->
               <module type="GPDModule" name="GPDGK16">
               </module>

            </module>
```

E: beam energy (GeV)
phi: angle between leptonic and hedonic planes (rad - Trento)

DVCSAllMinus? check the PARTONS Class Index:
https://partons.cea.fr/partons/doc/html/classes.html

Using the Guichon-Vanderhaeghen formulas

Conventionally affects the renormalization and factorization scale to the virtuality of the photon

Relates Bjorken-x and xi

Outlook

- PARTONS is a tool for the community, with open-source models of GPDs, an efficient x-space evolution code, a plethora of observables and accessible results from publications on GPD phenomenology.

- There is more than what I have talked about, notably: **EpIC: novel Monte Carlo generator for exclusive processes** (arXiv:2205.01762) for multichannel exclusive analyses with radiative corrections.

- The documentations and tutorials online are extensive. The code is actively developed so you can also get in touch with the collaboration and hopefully contribute with your own analyses to the extension of the framework!

Thank you for your attention!

- The codes used for this initiation are available at https://github.com/hervedutrieux/PARTONS_femtoschool.git. Although inspired by standard PARTONS / APFEL++ tutorials, those are personal suggestions which have not been endorsed by the PARTONS collaboration, and offer no guarantee of efficiency or best practices. Refer to the online resources or get in touch through the website https://partons.cea.fr with the development team in case of dissatisfaction.