GPDs at EIC Meeting

Gepard (tool for studying GPDs)

Krešimir Kumerički (University of Zagreb, Faculty of Science)

3DPartons week, Orsay
27 Oct 2022

HRZZ Croatian Science Foundation

STRONG 2020

EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Gepard - tool for studying GPDs

- Modelling Generalized Parton Distributions (GPDs) and Compton form factors (CFFs).
- Perturbative NLO QCD evolution of GPDs
- Calculation of deeply virtual Compton scattering (DVCS) and deeply virtual (vector) meson production (DVMP) observables to NLO accuracy.
- Fitting parametrized models to the experimental data.

Old Gepard	**New Gepard**
Fortran + C + Python	Python
NNLO, Neural nets	NLO only
2x faster	
also public now	

Old Gepard	New Gepard
Fortran + C + Python	Python
NNLO, Neural nets	NLO only
2x faster	
also public now	

Sources are on `github.com`, but you can just install Gepard as any "official" Python package

```
% pip install gepard
```

For interactive work, it is best to use Jupyter notebooks.

Extensive documentation at `gepard.phy.hr`

gepard

Search docs

CONTENTS:

- Software documentation
- Data sets
- Publications
- Credits

» Tool for studying the 3D quark and gluon distributions in the nucleon   View page source

Tool for studying the 3D quark and gluon distributions in the nucleon

Gepard is software for analysis of three-dimensional distribution of quarks and gluons in hadrons, encoded in terms of the so-called Generalized Parton Distributions (GPDs).

This web site has manifold purpose:

- Documentation of the software
- Examples of the use of software
- Interface to various representations of results: numerical and graphical
- Interface to datasets used in analyses: numerical and graphical

Contents:

Routines are thoroughly tested

```
[CAL:102]~/gepard/(devel|✚3…)% pytest --runslow
================================================ test session starts =================================================
platform linux -- Python 3.10.6, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/kkumer/gepard, configfile: pytest.ini
plugins: anyio-3.6.1, cov-2.12.1, typeguard-2.13.3, profiling-1.7.0
collected 118 items

tests/GK_test.py ..........                                                                                    [  8%]
tests/adacf_test.py .                                                                                          [  9%]
tests/cff_test.py ......................                                                                       [ 27%]
tests/dvmp_test.py .....                                                                                       [ 32%]
tests/elastic_ff_test.py .......                                                                               [ 38%]
tests/evol_test.py .....                                                                                       [ 42%]
tests/fit_test.py .....s                                                                                       [ 47%]
tests/fits_KM_test.py ...s.s..                                                                                 [ 54%]
tests/gpd_test.py ..............                                                                               [ 66%]
tests/observables_test.py ....ss...............                                                                [ 83%]
tests/special_test.py .......                                                                                  [ 89%]
tests/theory_test.py ...........s                                                                              [100%]

====================================== 112 passed, 6 skipped in 95.04s (0:01:35) =====================================
pytest --runslow  95.00s user 0.24s system 99% cpu 1:35.65 total
```

Most of the test values are obtained by independent implementation of formulas by Dieter Müller, Kornelija Passek-Kumerički, and Marija Čuić

Simple use

(Just want to calculate observables / CFFs / GPDs for available models.)

`DataPoint` **object**: contains information about kinematics and, possibly, about particular measurement perfomed at that kinematics

`Theory` **object**: contains algorithms for evaluation of various structure functions (CFFs, GPDs, …) and observables (cross-sections, asymmetries, …) for a given data point

In [2]:

```
import gepard as g
print(g.__version__)
```

```
0.9.11
```

In [3]:

```
# Constructing DataPoint object:
pt = g.DataPoint(xB=0.348, t=-0.3, Q2=3., phi=0.3)
```

In [4]:

```
# Some Theory objects are available already:
from gepard.fits import th_KM15
th_KM15.XGAMMA(pt)   # gamma* cross-section
```

Out[4]:

```
16.33882710095779
```

In [5]:

```
th_KM15.ImH(pt)  # calculating CFF
```

Out[5]:

```
2.807544271408012
```

In [6]:

```
# Experimental measurement datapoint:
pt = g.DataPoint(xB=0.348, t=-0.3, Q2=3., phi=0.3,
                 process='ep2epgamma', in1charge=-1,
                 exptype='fixed target', in1energy=6.
                 in1polarization=+1,
                 observable='XS', val=0.21, err=0.01)

pt.W, pt.xi   # auto-calculated kinematics
```

Out[6]:

```
(2.5497144988314844, 0.21065375302663436)
```

In [7]:

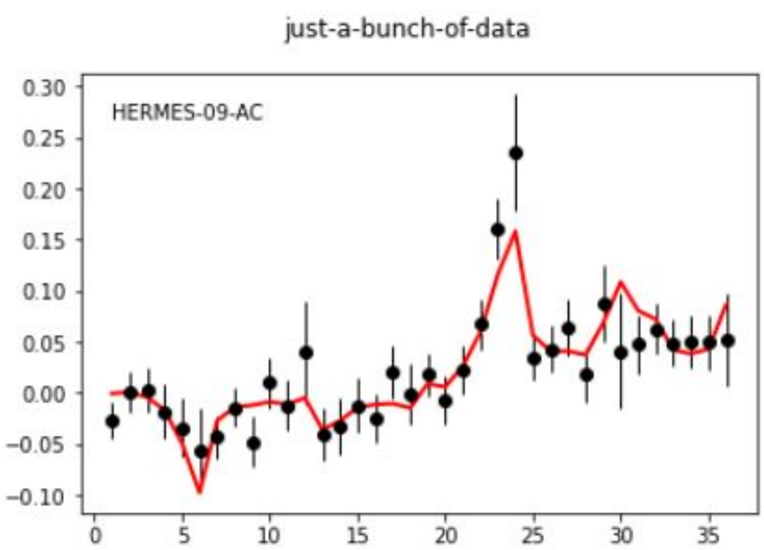
```
# There is a database of datasets available:
g.describe_data(g.dset[32])
```

```
npt x obs     collab  FTn    id  ref.
----------------------------------------------------
18 x AC      HERMES  0.0    32  arXiv:0909.3587
18 x AC      HERMES  1.0    32  arXiv:0909.3587
----------------------------------------------------
TOTAL = 36
```

In [8]:
```
th_KM15.chisq(g.dset[32])  # chi-square for dataset
```

Out[8]:
```
20.525561762363605
```

In [9]:
```
from gepard.plots import jbod  # just a bunch of data
fig = jbod(points=g.dset[32], lines=th_KM15)
```



Some features

In [16]:

```
# conversion to pandas DataFrame
pts.df()[['xB', 't', 'val']].head()
```

Out[16]:

	xB	t	val
0	0.000793	-0.1	29.90
1	0.000793	-0.3	8.00
2	0.000793	-0.5	2.13
3	0.000793	-0.8	0.27
4	0.001189	-0.1	13.30

In [17]:

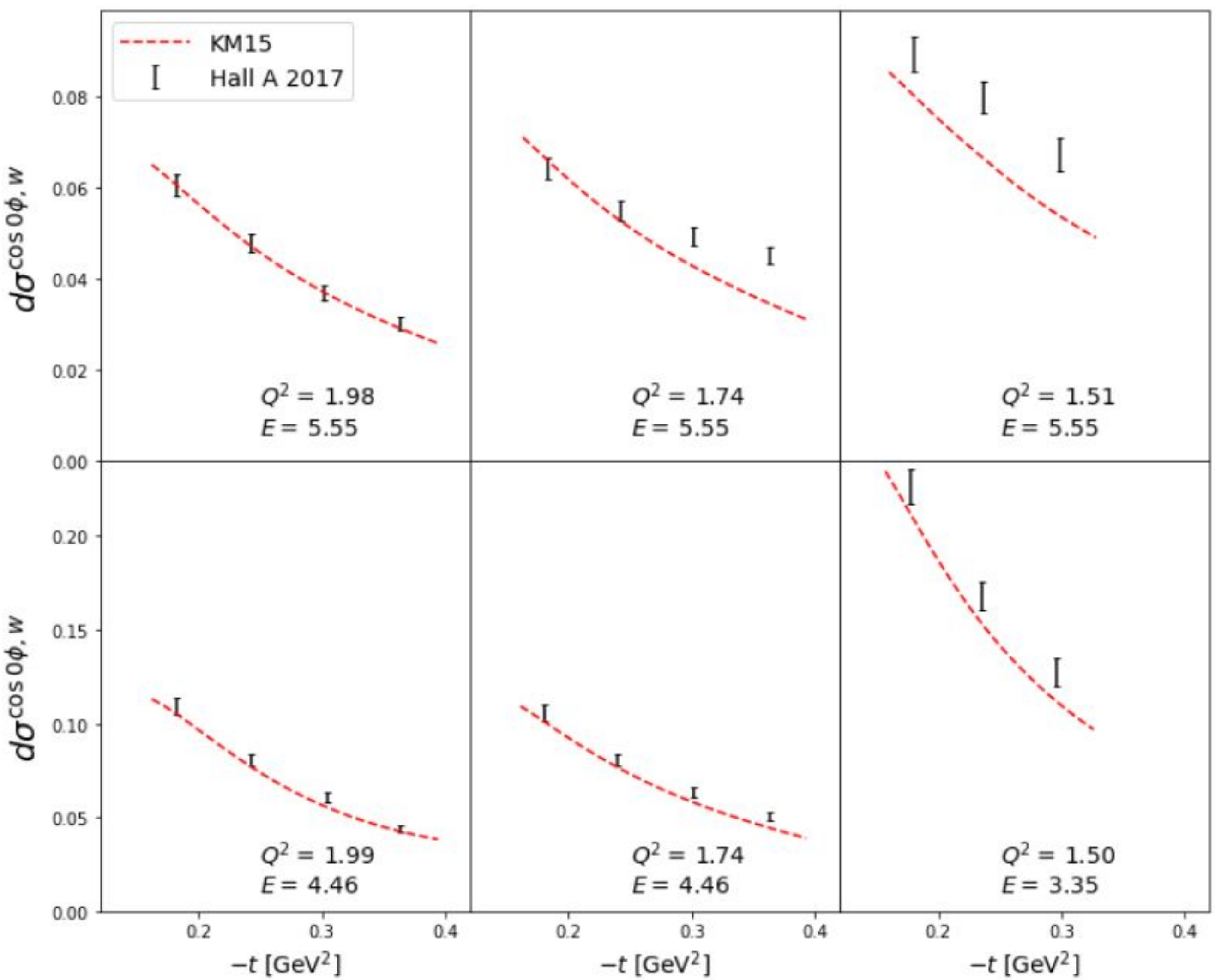
```
# kinematical cuts
cut_pts = g.select(g.dset[39], criteria=['Q2 > 5'])
cut_pts.df().Q2.min()  # minimal Q2 in set after cut
```

Out[17]:

```
8.0
```

In [19]:

```
# Predefined plots for some experimental sets
from gepard.plots import HallA17
fig = HallA17(lines=th_KM15)
```



Gepard includes many experimental data sets

gepard

Search docs

CONTENTS:

Software documentation

Data sets

ZEUS

H1

Collider data for fits

Publications

Credits

ZEUS

Chekanov:2003ya

- Reference: hep-ex/0305028
- Observables: *XGAMMA*

```
>>> import gepard as g
>>> g.list_data(45)
[ 45]     ZEUS   6    XGAMMA    0305028 Table 1
```

We take data from Table 1 with DVCS cross section in dependence on Q2. Tables 2 and 3 are same data but binned for W dependence. Using Q2 dependence gives us handle on evolution.

Chekanov:2008vy

- Reference: arXiv:0812.2517
- Observables: *XGAMMA*

```
>>> g.list_data([46, 47, 48, 49])
[ 46]     ZEUS   4    XGAMMA  0812.2517 Table 4
[ 47]     ZEUS   6    XGAMMA  0812.2517 Table 1
[ 48]     ZEUS   6    XGAMMA  0812.2517 Table 2
[ 49]     ZEUS   8    XGAMMA  0812.2517 Table 3
```

We take data from Table 1 (Q2 dependence of cross section, one might cut low-Q2 points) while Tables 2 (and 3) are same data binned in W (W and Q2). We also take data from Table 4, which is differential cross section in t, extracted from the subset of the above data, so strictly it is not statistically independent.

H1

Adloff:2001cn

- Reference: hep-ex/0107005
- Observables: *XGAMMA*

Aiming for fully reproducible research:

https://github.com/openhep

kkumer initial commit 10e20e3 on Feb 15 **2** commits

.gitignore	initial commit	8 months ago
README.md	initial commit	8 months ago
npb07.ipynb	initial commit	8 months ago
npb09.ipynb	initial commit	8 months ago
plb06.ipynb	initial commit	8 months ago

README.md

dvcs-old

code for arXiv:0904.0458, hep-ph/0703179, and hep-ph/0605237

These are Jupyter notebooks reproducing some numerics from papers

- K. Kumerički, D. Mueller, K. Pasek-Kumerički, and A. Schaefer, *Deeply virtual Compton scattering beyond next-to-leading order: the flavor singlet case*, Phys. Lett. B 648 (2007) 186-194, arXiv:hep-ph/0605237

code for arXiv:0904.0458, hep-ph/0703179, and hep-ph/0605237

Readme

0 stars

2 watching

0 forks

Releases

No releases published

Packages

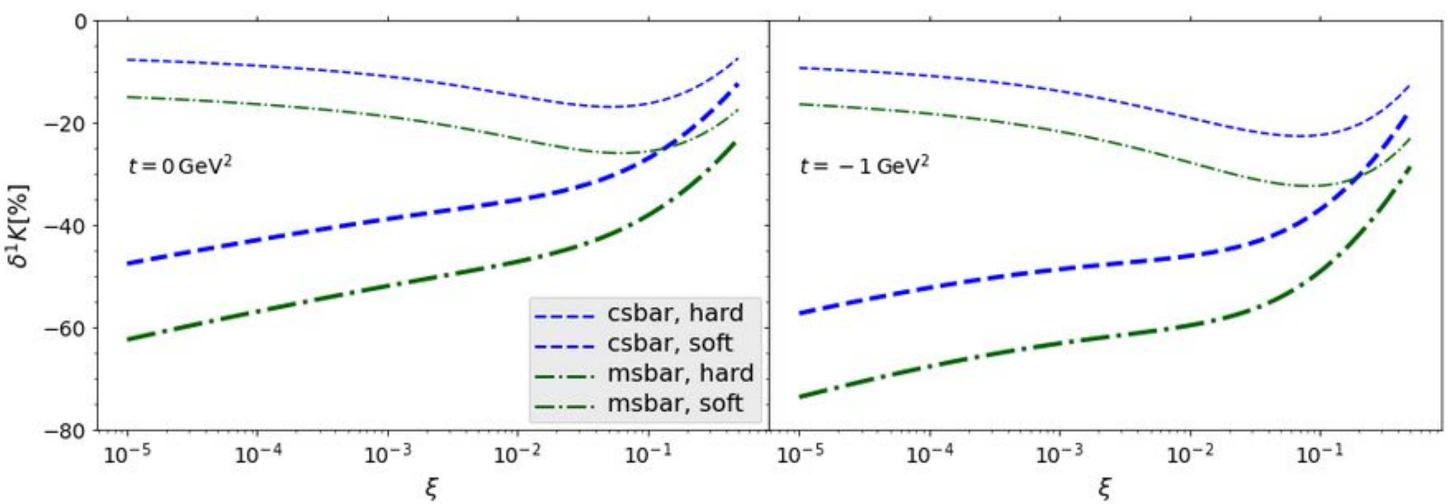
No packages published

Languages

Jupyter Notebook 100.0%

In [47]:

```
fig, axs = plt.subplots(1, 2, figsize=(16,5), sharey=True)
lstyle = {('csbar', 'hard') : (3, 'blue', '--'),
          ('csbar', 'soft') : (1.5, 'blue', '--'),
          ('msbar', 'hard') : (3, 'darkgreen', '-.'),
          ('msbar', 'soft') : (1.5, 'darkgreen', '-.')}
for pn, ax in enumerate(axs):
    t = {0: 0, 1: -1}[pn]
    for scheme in ['csbar', 'msbar']:
        for type in ['hard', 'soft']:
            width, color, style = lstyle[(scheme, type)]
            ax.plot(xis, Ks[(scheme, type, t)],
                    lw=width, ls=style, color=color, label=f'{scheme}, {type}')
    ax.set_xscale('log')
    ax.set_xlabel(r'$\xi$', fontsize=16)
    ax.set_ylim(-80, 0)
    ax.text(1.e-5, -30, r'$t={}\, \mathrm{{GeV}}^2$'.format(t), fontsize=14)
    ax.yaxis.set_major_locator(matplotlib.ticker.MultipleLocator(20))
    ax.yaxis.set_minor_locator(matplotlib.ticker.MultipleLocator(5))
    ax.tick_params(axis='both', which='major', labelsize=14, top=True)
    if pn == 0:
        ax.set_ylabel(r'$\delta^1 K [\%]$', fontsize=16)
        leg = ax.legend(handlelength=4, fancybox=True)
        frame  = leg.get_frame()
        frame.set_facecolor('0.90')
        for t in leg.get_texts():
            t.set_fontsize(16)
        for l in leg.get_lines():
            l.set_linewidth(2.0)
fig.subplots_adjust(wspace=0.0, hspace=0.0)
fig.canvas.draw()
```



GPD evolution codes

In momentum fraction x-space

- Freund and McDermott, 2002
- Vinnikov, 2006
- Bertone et al. APFEL++, 2022

In conformal moment j-space

- Gepard, 2022

Towards the “Les Houches” benchmark

PDF Les Houches benchmark:

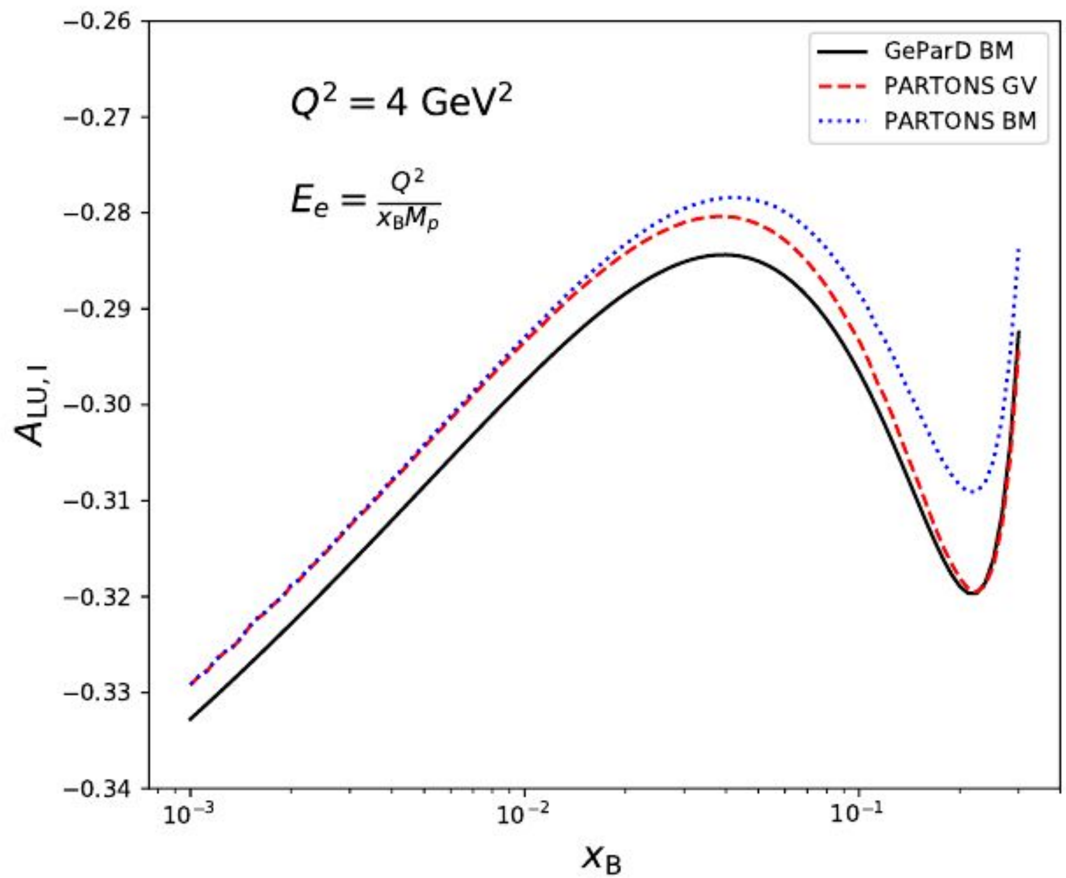
Table 3: Reference results for the $N_f = 4$ next-to-leading-order evolution for the initial conditions (30) – (32). The corresponding value of the strong coupling is $\alpha_s(\mu_r^2 = 10^4 \text{ GeV}^2) = 0.110902$. As in the leading-order case, the valence distributions s_v and c_v vanish for the input (31). The notation is explained in the first two paragraphs of Section 1.33.

NLO, $N_f = 4$, $\mu_f^2 = 10^4$ GeV^2							
x	xu_v	xd_v	xL_-	xL_+	xs_+	xc_+	xg
$\mu_r^2 = \mu_f^2$							
10^{-7}	1.0616^{-4}	6.2328^{-5}	4.2440^{-6}	1.3598^{+2}	6.6913^{+1}_{*}	6.6195^{+1}	1.1483^{+3}_{*}
10^{-6}	5.4177^{-4}	3.1719^{-4}	1.9241^{-5}	6.8396^{+1}	3.3342^{+1}	3.2771^{+1}	5.3911^{+2}
10^{-5}	2.6870^{-3}	1.5677^{-3}	8.3575^{-5}	3.2728^{+1}	1.5685^{+1}	1.5231^{+1}	2.3528^{+2}
10^{-4}	1.2841^{-2}	7.4558^{-3}	3.4911^{-4}	1.4746^{+1}	6.8355^{+0}	6.4769^{+0}	9.2872^{+1}_{*}
10^{-3}	5.7926^{-2}	3.3337^{-2}	1.4162^{-3}	6.1648^{+0}_{*}	2.6659^{+0}	2.3878^{+0}	3.1502^{+1}
10^{-2}	2.3026^{-1}	1.2928^{-1}	5.3251^{-3}	2.2527^{+0}	8.4220^{-1}_{*}	6.5246^{-1}	8.1066^{+0}
0.1	5.5452^{-1}	2.7336^{-1}	1.0011^{-2}	3.9336^{-1}	1.1489^{-1}	6.0351^{-2}	8.9867^{-1}
0.3	3.5393^{-1}	1.3158^{-1}	3.0362^{-3}	3.5848^{-2}	9.2030^{-3}	3.3890^{-3}	8.3451^{-2}
0.5	1.2271^{-1}	3.1067^{-2}	3.8265^{-4}	2.4126^{-3}	5.8424^{-4}	1.6955^{-4}	8.0473^{-3}

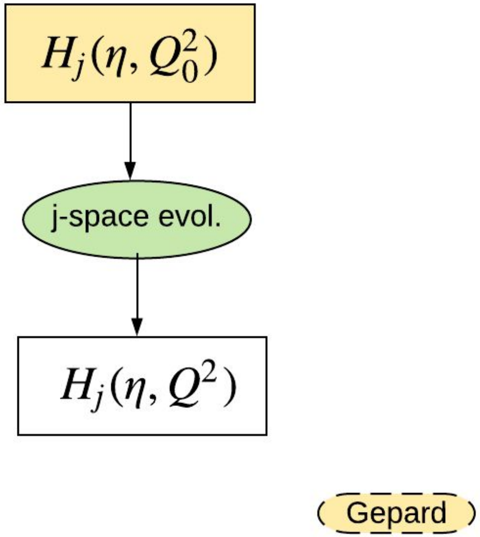
[Alekhin et al., hep-ph/0204316]

GeParD vs PARTONS, beam spin asymmetry

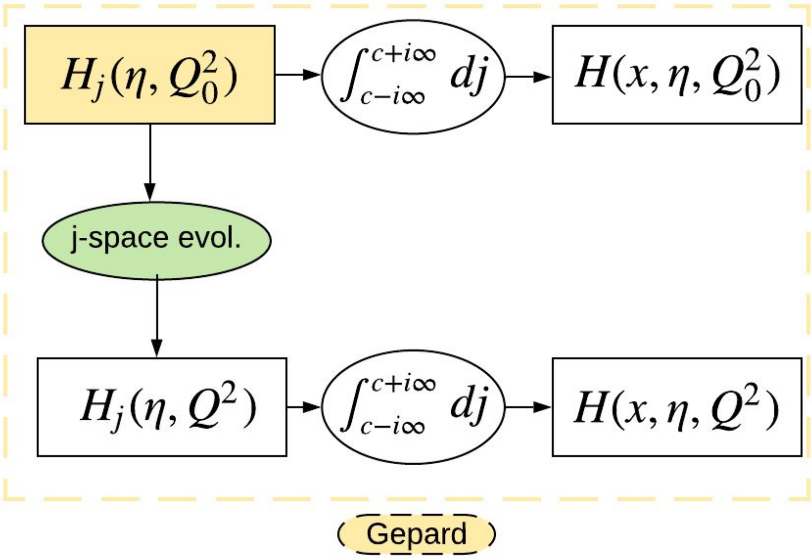
- BM = [Belitsky & Müller], GV = [Guichon & Vanderhaeghen]



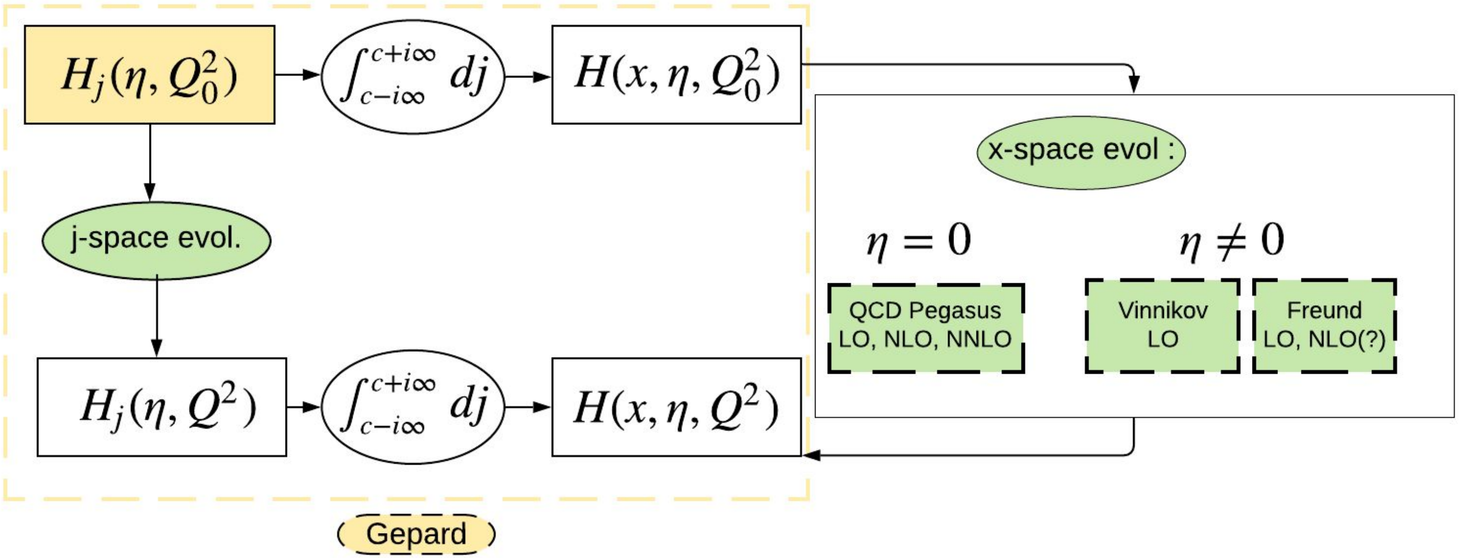
From j-space to x-space



Fron j-space to x-space

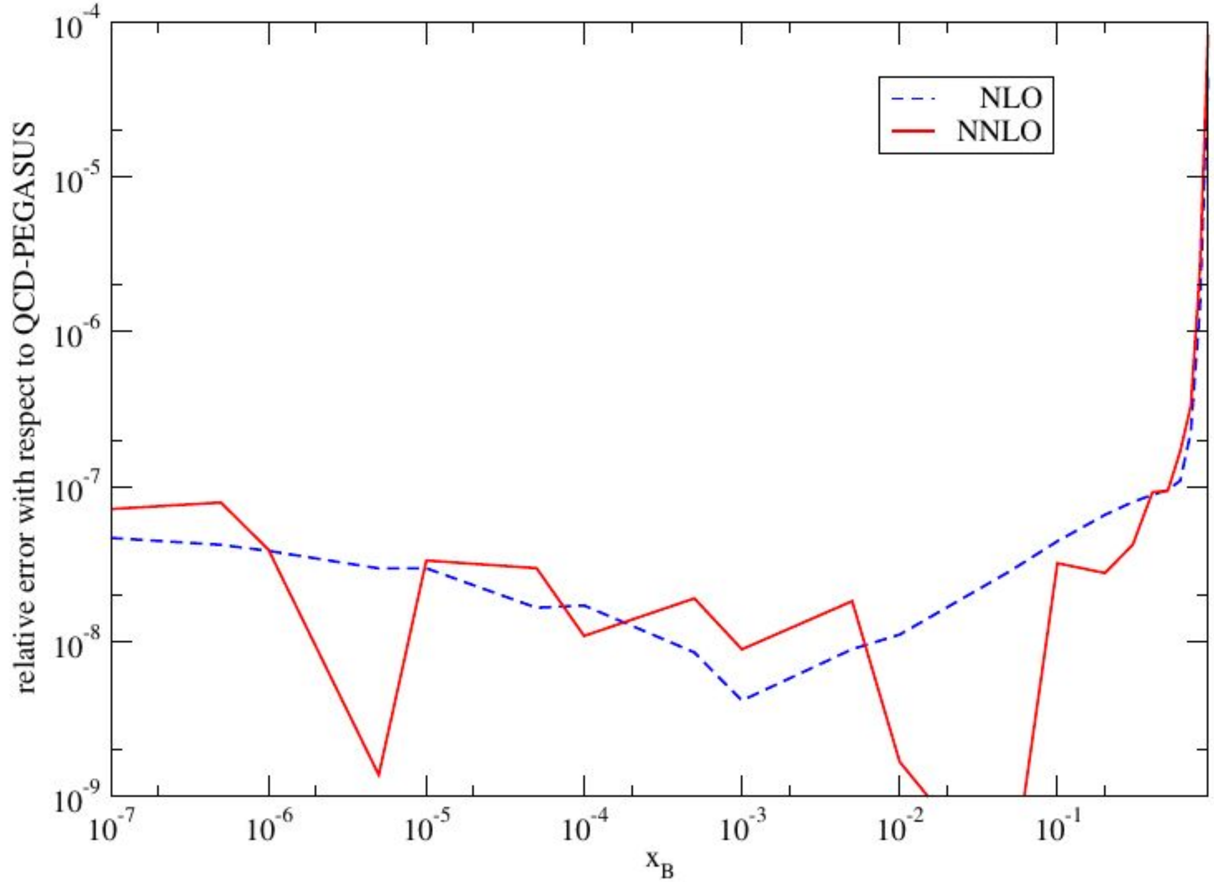


From j-space to x-space



Forward case comparison

- Comparison to QCD-Pegasus PDF evolution software [A. Vogt '04]

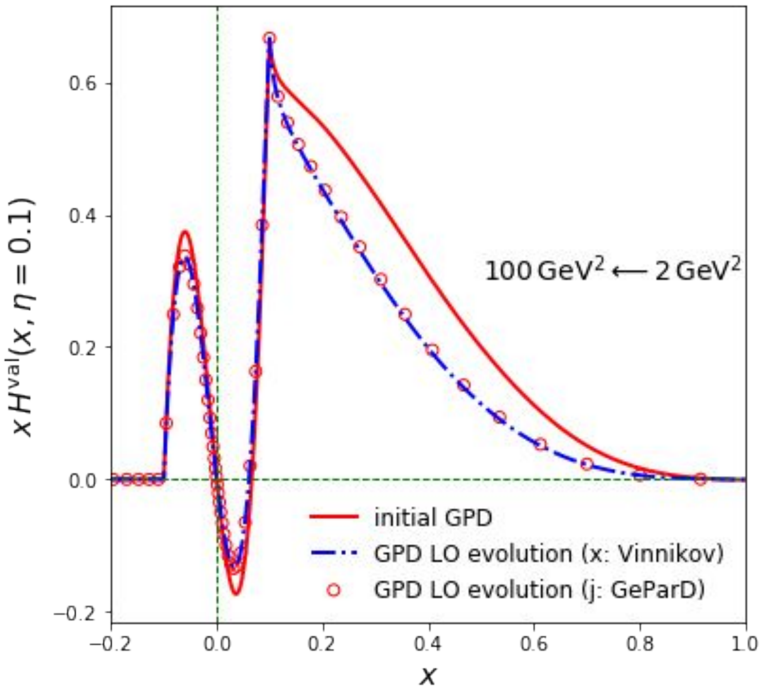


Gepard vs. Vinnikov

Python wrapper around Vinnikov evolution code (non-singlet only):

```
pyvinnikov.evol_ns(1, log(Q02), log(Q2), xi, x, gpd)
```

`Q02` - initial scale
`Q2` - final scale
`xi` - fixed ξ
`x` - array of x values
`gpd` - array of gpd values



Writing Python wrappers

- Writting Python wrapper around **Vinnikov C code** is dead easy:
  - `f2py` ("Fortran-2-Python") package does it almost automatically

- Writing Python wrapper around **Freund Fortran code** is tricky:
  - Most of the communication is via Fortran `COMMON` blocks
  - `some COMMON` blocks have mixed types (float+int) which have to be separated (otherwise one is hit by nasty bugs)
  - `f2py` does not treat Fortran `ENTRY` statements correctly (Fixed by writing explicitely signatures to Fortran interface file.)

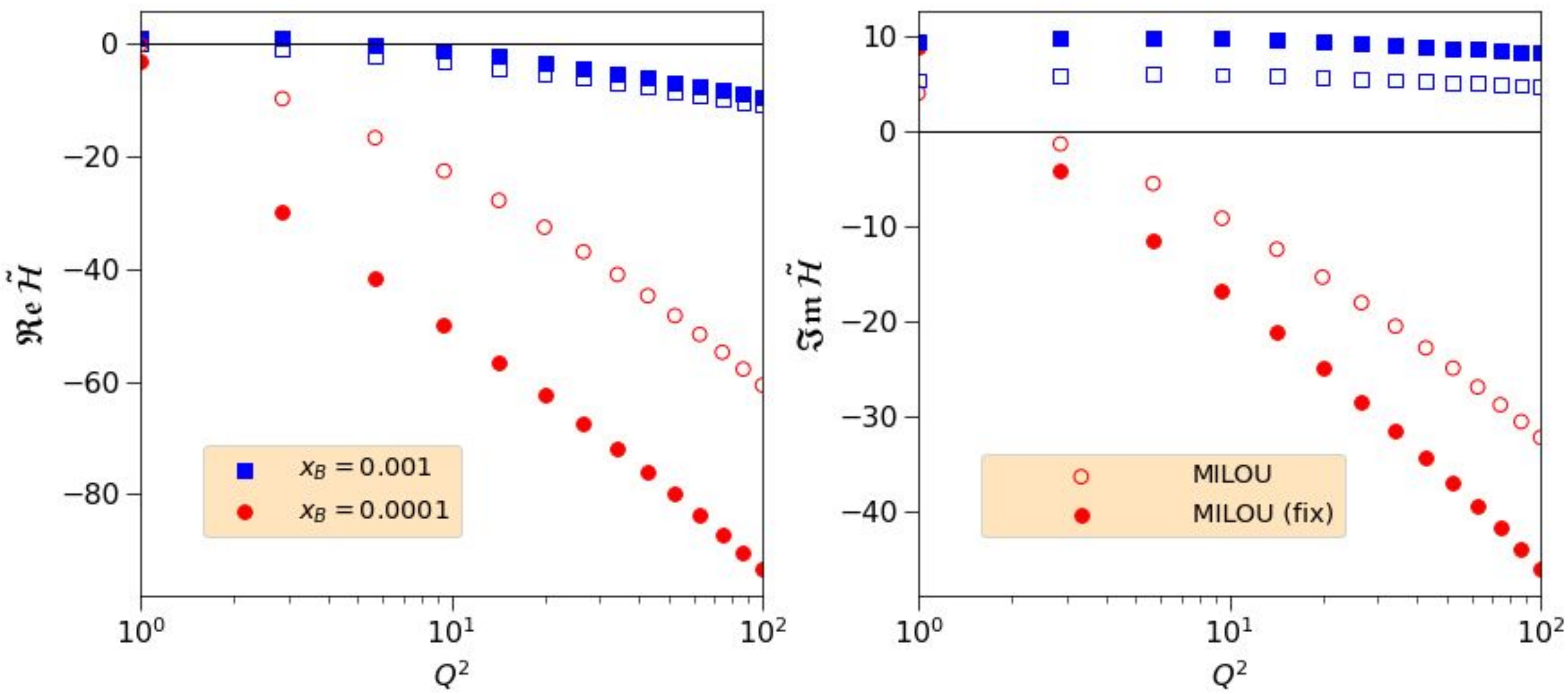
(Now evolving GPDs defined in Python should work in principle but was not yet tried.)

Bug in MILOU affecting H-tilde?

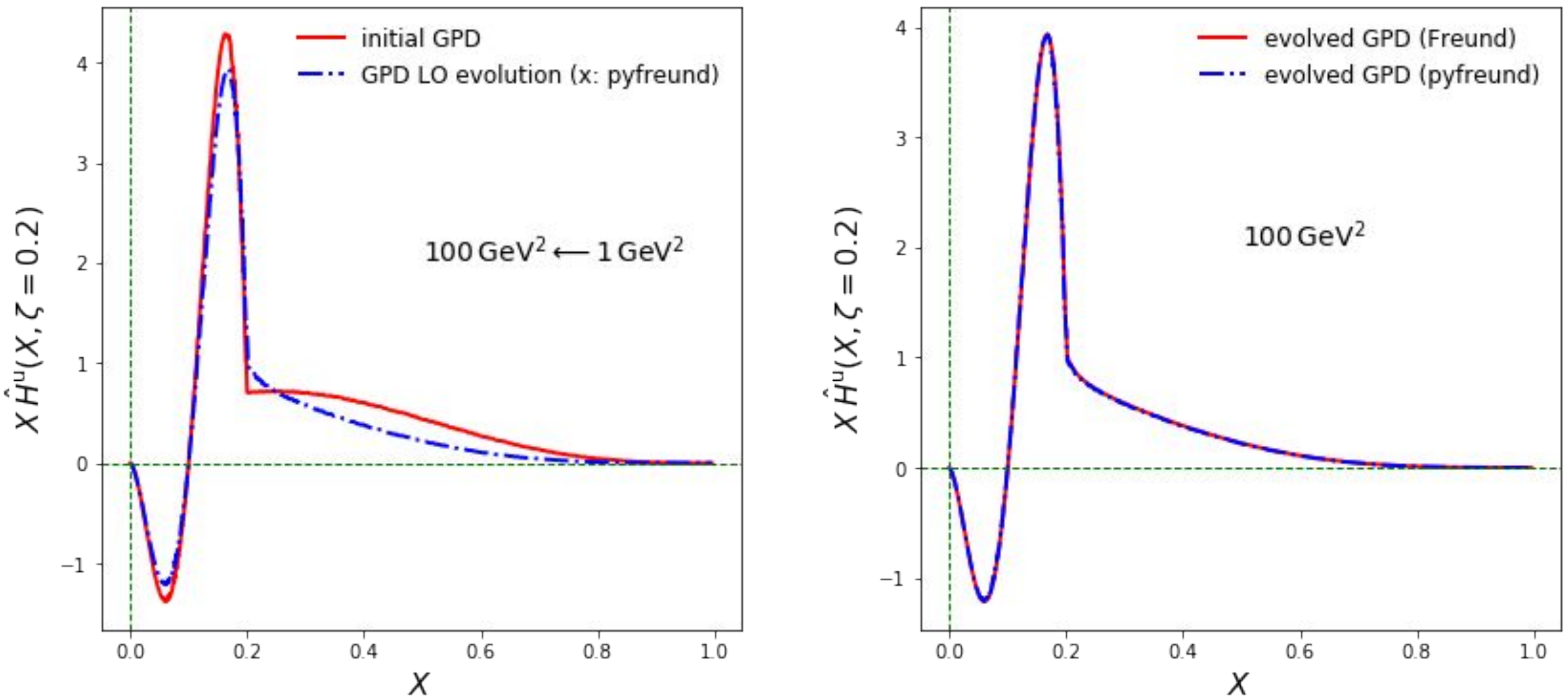
```
*    POLARIZED RADIATIVELY GENERATED LO AND NLO PARTON DENSITIES
*
*         M. GLUCK, E. REYA, M. STRATMANN AND W. VOGELSANG,
      [...]
*   COMMON:  The main program or the calling routine has to have
*            a common block  COMMON / INTINI / IINI , and  IINI
*            has always to be zero when PARPOL is called for the
*            first time or when 'ISET' has been changed.
      [...]
      SUBROUTINE PARPOL (ISET, X, Q2, U, D, UB, DB, ST, GL, G1P, G1N
```

In Freund/McDermott code `inputgpdglobalgrid.f` there is no required COMMON block to reset polarized PDFs.

Bug in MILOU affecting H-tilde



MILOU vs. Pythonized Freund



j-space to x-space GPDs within Gepard

$$H^{(+)}(x,\eta,t) = \sum_{\nu=0}^{\infty} \frac{1}{2i} \int_{c-i\infty}^{c+i\infty} dj\, \frac{\eta^{2\nu}\left[p_{j+2\nu}(x,\eta) - p_{j+2\nu}(-x,\eta)\right]}{\sin(\pi[j+1])}\, H_{j+2\nu,j+1}(t)\, \hat{d}_{00}^{j+1}(\eta)$$
$$- \sum_{\nu=1}^{\infty} \eta^{2\nu}\, 2p_{2\nu-1}(x,\eta) H_{2\nu-1,0}(t)\,. \qquad (4.5)$$

[Muller, Polyakov, Semenov-Tian-Shansky, 1412.4165]

$$p_j(x>\eta,\eta) = \frac{\sin(\pi[j+1])}{\pi}\, x^{-j-1}\, {}_2F_1\!\left(\left. \begin{matrix} (j+1)/2,\,(j+2)/2 \\ 5/2+j \end{matrix} \right| \frac{\eta^2}{x^2} \right)$$

We need implementation of 2F1 Hypergeometric function for complex j !

Summary

- Gepard needs users and contributors.
- Community needs benchmarks for GPD evolution and GPD-related observables