CFNS-SURGE Summer school 2025 (Indico).

Example for collinear factorization in p+p collisions

In this example, we study single inclusive π^0 production in $p+p$ collisions: $p+p \to \pi^0 + X$. Within the collinear factorization formalism, the differential cross section can be written as follows

$$\frac{d\sigma}{dy\, d^2p_T} = \frac{\alpha_s^2}{S} \sum_{a,b,b} \int \frac{dx_a}{x_a} f_{a/p}(x_a,\mu) \int \frac{dx_b}{x_b} f_{b/p}(x_b,\mu) \int \frac{dz_c}{z_c^2} D_{\pi^0/c}(z_c,\mu)\, H_{ab\to c}(\hat{s},\hat{t},\hat{u})\, \delta(\hat{s}+\hat{t}+\hat{u})$$

load proper packages

1. We need to use LHAPDF to call collinear parton distribution functions (PDFs) $f_{a/p}(x,\mu)$ and collinear fragmentation functions (FFs) $D_{\pi^0/c}(z,\mu)$. Follow their website information to install LHAPDF.
2. At the same time, we also have to use VEGAS Monte Carlo integration. One can install through "pip install vegas" at command line

LHAPDF installation on Google Colab

```
LHAPDF_VERSION = '6.5.4'
SITE_PACKGE_DIR = __import__('site').getsitepackages()[0]
PWD = __import__('os').getcwd()
PYTHON_VERSION = '%s.%s' % __import__('sys').version_info[0:2]

# download from SMU
!wget https://www.physics.smu.edu/olness/ftp/misc2/lhapdf/LHAPDF-{LHAPDF_VERSION}.tar.gz -O LHAPDF-{LHAPDF_VERSION}.tar.gz
# official hepforge download
# !wget https://lhapdf.hepforge.org/downloads/?f=LHAPDF-{LHAPDF_VERSION}.tar.gz -O LHAPDF-{LHAPDF_VERSION}.tar.gz
# outdated mirror from ALICE (works up to LHAPDF version 6.4.2)
# !wget https://github.com/alisw/LHAPDF/archive/refs/tags/v{LHAPDF_VERSION}.tar.gz

# install LHAPDF library
!tar xf LHAPDF-{LHAPDF_VERSION}.tar.gz
!cd LHAPDF-{LHAPDF_VERSION} && ./configure
!make -C LHAPDF-{LHAPDF_VERSION} -j 2
!make -C LHAPDF-{LHAPDF_VERSION} install

# make LHAPDF available to python
# use the following if you used the SMU download
!cd {SITE_PACKGE_DIR} && ln -sf {PWD}/LHAPDF-{LHAPDF_VERSION}/wrappers/python/NONE/local/lib/python3.11/dist-packages/lhapdf
# uncomment this for the official hepforge download
# !cd {SITE_PACKGE_DIR} && ln -sf {PWD}/LHAPDF-{LHAPDF_VERSION}/wrappers/python/NONE/lib/python{PYTHON_VERSION}/dist-packages/lha
!cd /usr/lib && ln -sf {PWD}/LHAPDF-{LHAPDF_VERSION}/src/.libs/libLHAPDF.so
!lhapdf update
```

```
make[2]: Entering directory '/content/LHAPDF-6.5.4/examples'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/share/doc/LHAPDF/examples'
 /usr/bin/install -c -m 644 testpdf testpdfset analyticpdf compatibility testpdfunc hessian2replicas reweight testpdf.cc t
make[2]: Leaving directory '/content/LHAPDF-6.5.4/examples'
make[1]: Leaving directory '/content/LHAPDF-6.5.4/examples'
Making install in doc
make[1]: Entering directory '/content/LHAPDF-6.5.4/doc'
make[2]: Entering directory '/content/LHAPDF-6.5.4/doc'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/content/LHAPDF-6.5.4/doc'
make[1]: Leaving directory '/content/LHAPDF-6.5.4/doc'
make[1]: Entering directory '/content/LHAPDF-6.5.4'
make[2]: Entering directory '/content/LHAPDF-6.5.4'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/lib/pkgconfig'
 /usr/bin/install -c -m 644 lhapdf.pc '/usr/local/lib/pkgconfig'
 /usr/bin/mkdir -p '/usr/local/share/LHAPDF'
 /usr/bin/install -c -m 644 lhapdf.conf pdfsets.index '/usr/local/share/LHAPDF'
make  install-data-hook
make[3]: Entering directory '/content/LHAPDF-6.5.4'
test -e /usr/local/share/LHAPDF/sets && rmdir /usr/local/share/LHAPDF/sets || true
make[3]: Leaving directory '/content/LHAPDF-6.5.4'
make[2]: Leaving directory '/content/LHAPDF-6.5.4'
make[1]: Leaving directory '/content/LHAPDF-6.5.4'
make: Leaving directory '/content/LHAPDF-6.5.4'
pdfsets.index:    45.7 KB[100.0%]
```

```
import lhapdf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

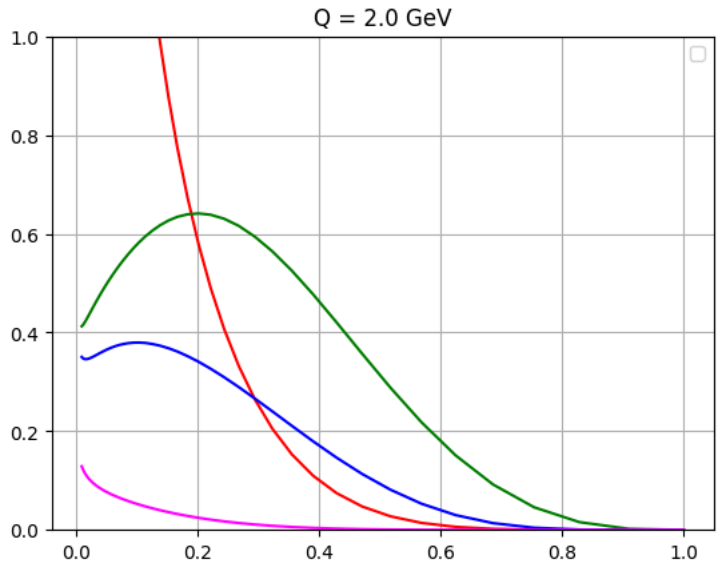
Install specific PDF set (CTEQ6L1) and FF set (MAPFF for pion)

```
!lhapdf install cteq6l1
```

```
cteq6l1.tar.gz:   310.4 KB[100.0%]
```

```
# call the central number (0) of CTEQ6L1
p = lhapdf.mkPDF("cteq6l1/0")

# let us look at the flavors, they follow convention of particle data group
# d = 1, u = 2, s = 3, c = 4, b = 5, g = 21
print('flavor:', p.flavors())
# lhapdf gives by default - xfxQ2: x * f(x, Q^2)
#                      or - xfxQ:  x * f(x, Q)

q = 2.0
xx = np.geomspace(0.01, 1)
gg = [p.xfxQ(0, x, q) for x in xx]
dd = [p.xfxQ(1, x, q) for x in xx]
uu = [p.xfxQ(2, x, q) for x in xx]
ss = [p.xfxQ(3, x, q) for x in xx]

plt.plot(xx, gg, 'red', label='')
plt.plot(xx, uu, 'green', label='')
plt.plot(xx, dd, 'blue', label='')
plt.plot(xx, ss, 'magenta', label='')
plt.title(f'Q = {q} GeV')
plt.xlabel('')
plt.ylabel('')
plt.legend()
plt.grid()
plt.ylim(0, 1)
plt.show()
plt.close()
```

```
flavor: [-5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 21]
<ipython-input-4-d0969a4f7d58>:24: UserWarning: No artists with labels found to put in legend.  Note that artists whose labe
  plt.legend()
```



FFs: MAP pion fragmentation function at leading-order

In their convention, pisum = $\pi^+ + \pi^-$. From isospin symmetry, we have $\pi^0 = (\pi^+ + \pi^-)/2$. This is why we divide it by 2.

```
!lhapdf install MAPFF10NLOPIsum
```

```
MAPFF10NLOPIsum.tar.gz:    65.9 MB[100.0%]
```

```
pisum = lhapdf.mkPDF("MAPFF10NLOPIsum/0")
pisum.flavors()

x = 0.3
Q = 10.0
bbf= pisum.xfxQ(-5, x, Q)/2.0
cbf= pisum.xfxQ(-4, x, Q)/2.0
sbf= pisum.xfxQ(-3, x, Q)/2.0
ubf= pisum.xfxQ(-2, x, Q)/2.0
dbf= pisum.xfxQ(-1, x, Q)/2.0
df = pisum.xfxQ(1, x, Q)/2.0
uf = pisum.xfxQ(2, x, Q)/2.0
sf = pisum.xfxQ(3, x, Q)/2.0
cf = pisum.xfxQ(4, x, Q)/2.0
bf = pisum.xfxQ(5, x, Q)/2.0
gf = pisum.xfxQ(21, x, Q)/2.0
print('%0.2e, %0.2e, %0.2e, %0.2e, %0.2e' %(bbf,cbf,sbf,ubf,dbf))
print('%0.2e, %0.2e, %0.2e, %0.2e, %0.2e, %0.2e' %(df,uf,sf,cf,bf,gf))
```

```
1.13e-01, 2.55e-01, 2.73e-01, 2.39e-01, 2.06e-01
2.06e-01, 2.39e-01, 2.73e-01, 2.55e-01, 1.13e-01, 1.19e-01
```

For more convenient use below, I will define PDF and FF function

```
# construct light flavor PDFs and FFs
# note: for now, we only need PDFs and FFs of quarks
def PDF(x, Q):

    u = p.xfxQ(2, x, Q)/x
    d = p.xfxQ(1, x, Q)/x
    ub= p.xfxQ(-2, x, Q)/x
    db= p.xfxQ(-1, x, Q)/x
    s = p.xfxQ(3, x, Q)/x
    g = p.xfxQ(21, x, Q)/x

    return u,d,ub,db,s,g
```

```
def FF(hh, x, Q):
    fu = hh.xfxQ(2, x, Q)/x/2.0
    fub= hh.xfxQ(-2, x, Q)/x/2.0
    fd = hh.xfxQ(1, x, Q)/x/2.0
    fdb= hh.xfxQ(-1, x, Q)/x/2.0
    fs = hh.xfxQ(3, x, Q)/x/2.0
    fsb= hh.xfxQ(-3, x, Q)/x/2.0
    fg = hh.xfxQ(21, x, Q)/x/2.0

    return fu,fub,fd,fdb,fs,fsb,fg
```

I also need VEGAS Monte Carlo integration routine

```
!pip install vegas
```

```
Collecting vegas
  Downloading vegas-6.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (1.9 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from vegas) (2.0.2)
Collecting gvar>=13.1.5 (from vegas)
  Downloading gvar-13.1.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.0 kB)
Requirement already satisfied: scipy>=1.11 in /usr/local/lib/python3.11/dist-packages (from gvar>=13.1.5->vegas) (1.15.3)
Downloading vegas-6.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.1/4.1 MB 21.6 MB/s eta 0:00:00
Downloading gvar-13.1.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 11.9 MB/s eta 0:00:00
Installing collected packages: gvar, vegas
Successfully installed gvar-13.1.6 vegas-6.3
```

Strong coupling constant

At leading order, strong coupling constant α_s runs through the following equation

$$\alpha_s(Q) = \frac{4\pi}{\beta_0 \ln\Big(Q^2/\Lambda^2_{\rm QCD}(n_f)\Big)},$$

where $\beta_0 = 11 - 2/3 n_f$ with n_f the active flavor of quarks. $\Lambda_{\rm QCD}$ in general also depends on n_f. Here the values is quoted from CTEQ6L1.

```
def alpha(Q):
    mb = 4.5
    if(Q <= mb):
        nf = 4
        lam_QCD = 0.215
    else:
        nf = 5
        lam_QCD = 0.165

    b0 = 11.0 - 2.0/3.0*nf
    tt = 2.0*np.log(Q/lam_QCD)
    return 4.0*np.pi/(b0*tt)
```

Partonic cross section

These are $H_{ab\to c}(\hat{s},\hat{t},\hat{u})$ given above.

```
def Upp(s, t, u):
    Nc = 3.0
#   qq' --> qq'
    WQ1=(Nc**2-1.0)/(2.0*Nc**2)*(s*s+u*u)/(t*t)
#   qq  ->qq
    WQ2=(Nc**2-1.0)/(2.0*Nc**2)*( (s*s+u*u)/(t*t)+(s*s+t*t)/(u*u) ) \
         -(Nc**2-1.0)/(Nc**3)*(s*s)/(t*u)
#   qqb ->q'qb'
    WQ3=(Nc**2-1.0)/(2.0*Nc**2)*(t*t+u*u)/(s*s)
#   qqb ->qqb
    WQ4=(Nc**2-1.0)/(2.0*Nc**2)*( (s*s+u*u)/(t*t)+(t*t+u*u)/(s*s) ) \
        -(Nc**2-1.0)/(Nc**3)*(u*u)/(s*t)
#   qqb ->gg
    WQ5=(Nc**2-1.0)**2/(2.0*Nc**3)*(u/t+t/u) \
    -(Nc**2-1.0)/Nc*(t*t+u*u)/(s*s)
#   gg  ->qqb
```

```
    WQ6=1.0/(2.0*Nc)*(t/u+u/t)-Nc/(Nc**2-1.0)*(t*t+u*u)/(s*s)
#   qg  ->qg
    WQ7=(Nc**2-1.0)/(2.0*Nc**2)*(-s/u-u/s)+(s*s+u*u)/(t*t)
#   gg  ->gg
    WQ8=4.0*Nc**2/(Nc**2-1.0)*(3.0-t*u/(s*s)-s*u/(t*t)-s*t/(u*u))

    return WQ1, WQ2, WQ3, WQ4, WQ5, WQ6, WQ7, WQ8
```

Below is the differential cross section

1. Since the theory formula has a delta-function $\delta\left(\hat{s}+\hat{t}+\hat{u}\right)$, one can use it to integrate out the variable x_b.
2. Then we will have integration over x_a and z_c.
3. If the experiment further measures the cross section for a specific range of rapidity y and transverse momentum p_T, we have to integrate over those ranges.
4. We also have to choose the factorization scale μ over there. In the program below, this μ is called `scale`. Usually, we choose `scale` to be the typical hard scale of the process. In our situation, this should be transverse momentum of the pion.
5. This is why we write the cross section as a function of x_a, z_c, y, p_T and `scale`.
6. There are so many partonic channels you have to sum over. This is the $\sum_{a,b,c}$ in the theory formula given above.

```
def sigma(xa, zc, y, roots, pT, scale):
    roots2 = roots**2
    AA=xa*roots2-pT/zc*roots*np.exp(y)
    BB=xa*pT/zc*roots*np.exp(-y)
    xb=BB/AA

    sh=xa*xb*roots2
    uh=-pT/zc*xb*roots*np.exp(y)
    th=-pT/zc*xa*roots*np.exp(-y)

    if(xa < 1. and xa > 0. and xb < 1. and xb > 0. and zc < 1. and zc > 0.):
        U1,D1,UB1,DB1,S1,GL1 = PDF(xa, scale)
        U2,D2,UB2,DB2,S2,GL2 = PDF(xb, scale)
        fu,fub,fd,fdb,fs,fsb,fg = FF(pisum, zc, scale)
        WQ1,WQ2,WQ3,WQ4,WQ5,WQ6,WQ7,WQ8 = Upp(sh,th,uh)
        WT1,WT2,WT3,WT4,WT5,WT6,WT7,WT8 = Upp(sh,uh,th)

        SIG1=WQ2*(U1*U2*fu+UB1*UB2*fub+D1*D2*fd+DB1*DB2*fdb+S1*S2*fs+S1*S2*fsb) \
            +WQ3*(U1*UB2*(fd+fs)+D1*DB2*(fu+fs)+S1*S2*(fu+fd) \
            +UB1*U2*(fdb+fsb)+DB1*D2*(fub+fsb)+S1*S2*(fub+fdb)) \
            +WT3*(UB1*U2*(fd+fs)+DB1*D2*(fu+fs)+S1*S2*(fu+fd) \
            +U1*UB2*(fdb+fsb)+D1*DB2*(fub+fsb)+S1*S2*(fub+fdb))

        SIG2=WQ4*(U1*UB2*fu+D1*DB2*fd+S1*S2*fs+UB1*U2*fub+DB1*D2*fdb+S1*S2*fsb) \
            +WT4*(UB1*U2*fu+DB1*D2*fd+S1*S2*fs+U1*UB2*fub+D1*DB2*fdb+S1*S2*fsb) \
            +WQ1*(U1*(D2+DB2+S2+S2)*fu+D1*(U2+UB2+S2+S2)*fd+UB1*(D2+DB2+S2+S2)*fub \
            +DB1*(U2+UB2+S2+S2)*fdb+S1*(U2+UB2+D2+DB2)*fs+S1*(U2+UB2+D2+DB2)*fsb)

        SIG3=WT1*((D1+DB1+S1+S1)*U2*fu+(U1+UB1+S1+S1)*D2*fd \
            +(D1+DB1+S1+S1)*UB2*fub+(U1+UB1+S1+S1)*DB2*fdb \
            +(U1+UB1+D1+DB1)*S2*fs+(U1+UB1+D1+DB1)*S2*fsb) \
            +WQ5*(U1*UB2+D1*DB2+S1*S2)*fg+WT5*(UB1*U2+DB1*D2+S1*S2)*fg

        SIG4=WQ6*GL1*GL2*(fu+fd+fs)+WT6*GL1*GL2*(fub+fdb+fsb) \
            +WQ7*(GL2*(U1*fu+D1*fd+S1*fs+UB1*fub+DB1*fdb+S1*fsb) \
            +GL1*fg*(U2+UB2+D2+DB2+S2+S2)) \
            +WT7*(GL1*(U2*fu+D2*fd+S2*fs+UB2*fub+DB2*fdb+S2*fsb) \
            +GL2*fg*(U1+UB1+D1+DB1+S1+S1)) \
            +WQ8*GL1*GL2*fg

        SIG=SIG1+SIG2+SIG3+SIG4

        sigma = 1./xa/(zc*zc)/BB*SIG*alpha(scale)**2/roots2
    else:
        sigma = 0.

    return sigma
```

Let us now set up the vegas integration

1. We will generate the differential cross section as a function of transverse momentum p_T, so we integrate over x_a, z_c, y.
2. We will compare with data from PHENIX collaboration at RHIC, which integrate over $-0.35 < y < 0.35$. They defined as a normalized rapidity distribution, this is why we divide by the rapidity interval $dY = y_{\max} - y_{\min} = 0.7$.

```
import vegas


def dist_vegas(roots, pT, scale):
    convert = 0.389
    def cross_before(val):
        xa = val[0]
        zc = val[1]
        y  = val[2]
        return sigma(xa, zc, y, roots, pT, scale)

    ymin = -0.35; ymax = 0.35; dY = ymax - ymin
    integ = vegas.Integrator([[0, 1], [0, 1], [ymin, ymax]])
    result = integ(cross_before, nitn=10, neval=10000)
    return convert/dY*result.mean
```

Computing

```
roots = 200.
pT_a = [1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, 12.5, 13.5, 14.5, 15.5, 16.5, 17.5, 18.5, 19.5]
num0 = len(pT_a)
cross = [0.]*num0
for i in range(num0):
    pT = pT_a[i]
    scale = pT
    cross[i] = dist_vegas(roots, pT, scale)
    print('%0.3e\t %0.3e' %(pT, cross[i]))
```

```
1.500e+00        7.867e-02
2.500e+00        2.997e-03
3.500e+00        2.899e-04
4.500e+00        4.725e-05
5.500e+00        1.055e-05
6.500e+00        2.929e-06
7.500e+00        9.544e-07
8.500e+00        3.498e-07
9.500e+00        1.407e-07
1.050e+01        6.128e-08
1.150e+01        2.825e-08
1.250e+01        1.375e-08
1.350e+01        6.992e-09
1.450e+01        3.685e-09
1.550e+01        2.012e-09
1.650e+01        1.127e-09
1.750e+01        6.497e-10
1.850e+01        3.802e-10
1.950e+01        2.276e-10
```

Get ready for plots

1. Export the theory calculations to a file using pandas
2. Set plot features, e.g. font, font size, etc.
3. Make sure to change Google Colab directory to your Google Drive folder to store and call files

```
from google.colab import drive
drive.mount('/content/drive')
import sys
sys.path.append('/content/drive/MyDrive/Colab\ Notebooks/CFNS_SURGE_school')
```

```
Mounted at /content/drive
```

```
df = pd.DataFrame({
    "pT": pT_a,
    "cross": cross})
df.to_csv('python_thy.csv', index=False)

# prepare for plots
```

```
python_thy = pd.read_csv('python_thy.csv')
python_thy.head()
```

	pT	cross
0	1.5	0.078672
1	2.5	0.002997
2	3.5	0.000290
3	4.5	0.000047
4	5.5	0.000011

Next steps: Generate code with python_thy | View recommended plots | New interactive sheet

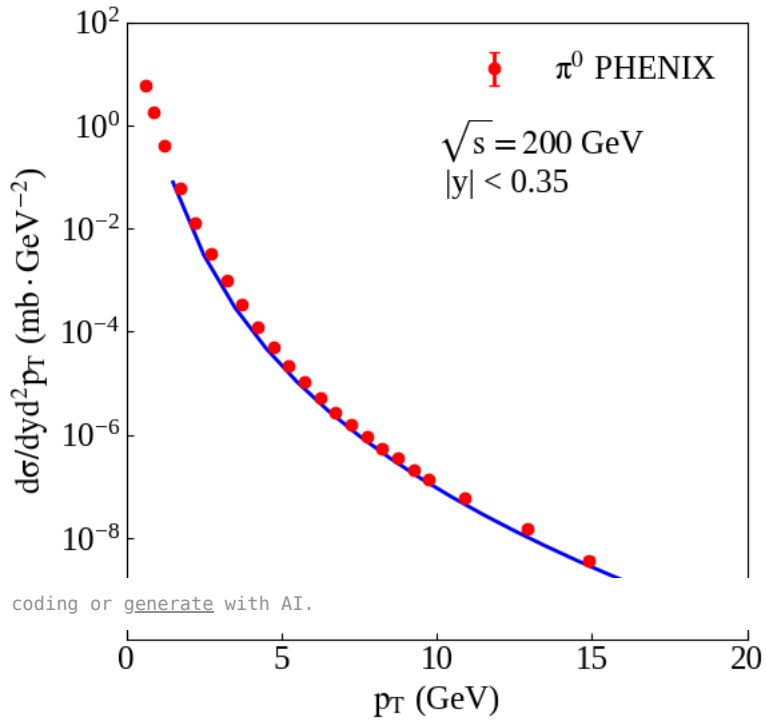
```
# set the font globally
# by default, I typically use font "Times New Roman".
# However, usually at Linux, such a font is not available (unless you install).
# so you need to use an alternative version that is very close to Times New Roman.
# such a font exists at Linux, it is "Liberation Serif".
# So if I am using it at Mac, you will see I am using "Times New Roman"; at Linux, I am using "Liberation Serif".
plt.rcParams.update({'font.family':'Liberation Serif', 'mathtext.fontset':'stix', 'mathtext.default':'rm', 'font.size':18, 'line
```

We compare our calculation with the experimental data from the PHENIX experiment at RHIC.

- Original paper: **PHENIX Collaboration Phys.Rev.D 76 (2007) 051106** (https://inspirehep.net/literature/749394)
- Access to data also here: https://www.hepdata.net/record/ins749394

```
# experimental data, to compare with theory
data = pd.read_csv('HEPData-ins749394-v1-Figure_1.csv', header=None,skiprows=8)
data = pd.DataFrame(data)

# central of pT bin
x = data[3]
# diff. cross section value
y = data[4]
# yerr = data[2]
num2 = len(x)
yerr = np.sqrt(data[5]**2+data[7]**2)
print('p_T','\t\t x-sec','\t\t combined uncertainty')
for i in range(num2):
    print('%0.3e\t %0.3e\t %0.3e' %(x[i], y[i], yerr[i]))

plt.figure(figsize=(6, 6))
plt.yscale('log')
plt.tick_params(axis='both', which='both', direction='in')
plt.xlabel('$p_T$ (GeV)')
plt.ylabel('$d\sigma/dyd^2p_T$ ($mb\cdot GeV^{-2}$)')
plt.xlim(0,20)
plt.ylim(1.1e-10,1e2)

# plot experimental data points
plt.errorbar(x, y, yerr = yerr, fmt = 'ro', label='$\pi^0$ PHENIX', capsize=3, markersize = 6)

# plot theory as curve
x_val = python_thy["pT"]
# over here, I added this "multiply()" function to just show
# that pandas has this feature to multiply each element of the column
y_val = python_thy["cross"].multiply(1.0)
plt.plot(x_val, y_val, 'b-')

plt.legend(frameon=False)
plt.text(10, 0.05, '$\sqrt{s} = 200$ GeV \n |y| < 0.35')
plt.savefig('RHIC_pp.pdf', bbox_inches='tight')
plt.show()
```

p_T	x-sec	combined uncertainty
6.160e-01	5.950e+00	7.520e-01
8.660e-01	1.780e+00	1.300e-01
1.215e+00	3.960e-01	2.710e-02
1.719e+00	6.150e-02	4.561e-03
2.223e+00	1.300e-02	1.030e-03
2.725e+00	3.290e-03	2.692e-04
3.228e+00	9.790e-04	8.205e-05
3.730e+00	3.280e-04	2.802e-05
4.231e+00	1.190e-04	1.090e-05
4.732e+00	4.890e-05	4.522e-06
5.234e+00	2.160e-05	2.012e-06
5.735e+00	1.040e-05	9.705e-07
6.237e+00	5.190e-06	4.874e-07
6.738e+00	2.750e-06	2.602e-07
7.238e+00	1.560e-06	1.491e-07
7.739e+00	9.130e-07	8.783e-08
8.240e+00	5.450e-07	5.314e-08
8.740e+00	3.470e-07	3.435e-08
9.241e+00	2.140e-07	2.150e-08
9.741e+00	1.330e-07	1.376e-08
1.088e+01	5.920e-08	6.188e-09
1.290e+01	1.490e-08	1.797e-09
1.491e+01	3.700e-09	5.411e-10
1.692e+01	1.250e-09	2.398e-10
1.893e+01	5.470e-10	1.430e-10



Start coding or generate with AI.