QCD carpentry: 1D structure of the nucleon

Nobuo Sato

ODU/JLab

CFNS summer school

Stony Brook, 2019

Outline

- **Part I:** Basics of DIS
- **Part II:** Elementary treatment of DIS Factorization
- **Part III:** Solving RGE - **coding**
- **Part IV:** DIS phenomenology - **coding**

Outline

- **Part I: Basics of DIS**
- **Part II: Elementary treatment of DIS Factorization**
- **Part III: Solving RGE - coding**
- **Part IV: DIS phenomenology - coding**

References

- Collins “Foundations of perturbative QCD”
- Moffat, Melnitchouk, Rogers, NS (PRD95, 2017)
- Vogt (hep-ph/0408244)

Part III: Solving RGE

- The β function
- Mellin transforms
- DGLAP - non-singlet and singlet evolution
- DIS cross sections

The β function

- RGE for strong coupling

$$a_S(\mu^2) = \frac{\alpha_S(\mu^2)}{4\pi}$$

$$\frac{da_S}{d\ln\mu^2} = \beta(a_s) = -\left(\beta_0 a_S^2 + \beta_1 a_S^3 + ...\right)$$

The β function

- RGE for strong coupling

$$a_S(\mu^2) = \frac{\alpha_S(\mu^2)}{4\pi}$$

$$\frac{da_S}{d\ln\mu^2} = \beta(a_s) = -\left(\beta_0 a_S^2 + \beta_1 a_S^3 + ...\right)$$

- Coefficients up to two loops

$$\beta_0 = 11 - \frac{2}{3}N_f \qquad \beta_1 = 102 - \frac{38}{3}N_f$$

The β function

- The RGE depends on $N_f \rightarrow$ **“mass thresholds”**

scale	N_f	active flavors
$\mu < m_c$	3	u, d, s
$m_c \leq \mu < m_b$	4	u, d, s, c
$m_b \leq \mu$	5	u, d, s, c, b

The β function

- The RGE depends on $N_f \rightarrow$ **"mass thresholds"**

scale	N_f	active flavors
$\mu < m_c$	3	u, d, s
$m_c \leq \mu < m_b$	4	u, d, s, c
$m_b \leq \mu$	5	u, d, s, c, b

- The RGE is discontinuous at the mass thresholds

The β function

- The RGE depends on $N_f \to$ **“mass thresholds”**

scale	N_f	active flavors
$\mu < m_c$	3	u, d, s
$m_c \leq \mu < m_b$	4	u, d, s, c
$m_b \leq \mu$	5	u, d, s, c, b

- The RGE is discontinuous at the mass thresholds

- a_S is continuous at the mass thresholds

The β function

- To solve the RGE we need boundary conditions (BC) for each N_f
- But we need continuous $a_s \rightarrow$ the recipe
  - Start with $a_S(m_Z) = 0.118/4\pi$
  - Compute $a_S(m_b)$ evolved with BC $a_S(m_Z)$ and $N_f = 5$
  - Compute $a_S(m_c)$ evolved with BC $a_S(m_b)$ and $N_f = 4$

QCD carpentry: coding up α_S RGE

- params.py
- alphaS.py

params.py

```
#--euler-
euler=0.57721566490153286060651209008240243104215933593992

#--color factors
CA=3.0
CF=4.0/3.0
TR=0.5
TF=0.5

#--set_masses
mc    = 1.28
mb    = 4.18
mZ    = 91.1876
mW    = 80.398
M     = 0.93891897

mc2    = mc**2
mb2    = mb**2
mZ2    = mZ**2
M2     = M**2
```

params.py

```python
#--QED and QCD couplings
alfa  = 1/137.036
alphaSMZ = 0.118
alfa2  = alfa**2

#--quark charges

eU2 = 4.0/9.0
eD2 = 1.0/9.0

couplings={1:eU2,2:eD2,3:eD2,4:eU2,5:eD2,6:eU2}
```

alphaS.py

```
#!/usr/bin/env python
import sys,os
import numpy as np
import params as par

beta=np.zeros((7,3))
for Nf in range(3,7):
    beta[Nf,0]=11.0-2.0/3.0*Nf
    beta[Nf,1]=102.-38.0/3.0*Nf

def beta_func(a,Nf):
    return -(beta[Nf,0]+a*beta[Nf,1])*a**2

def get_Nf(Q2):
    Nf=3
    if Q2>=(4.*par.mc2): Nf+=1
    if Q2>=(4.*par.mb2): Nf+=1
    return Nf
```

alphaS.py

```
def evolve_a(Q20,a,Q2,Nf):
    #--Runge-Kutta implemented in pegasus
    LR = np.log(Q2/Q20)/20.0
    for k in range(20):
        XK0 = LR * beta_func(a,Nf)
        XK1 = LR * beta_func(a + 0.5 * XK0,Nf)
        XK2 = LR * beta_func(a + 0.5 * XK1,Nf)
        XK3 = LR * beta_func(a + XK2,Nf)
        a+= (XK0 + 2.* XK1 + 2.* XK2 + XK3) * 0.166666666666666666
    return a

#--build boundary conditions
ab=evolve_a(par.mZ2,par.aZ,par.mb2,5)
ac=evolve_a(par.mb2,ab,par.mc2,4)
a0=evolve_a(par.mc2,ac,par.Q20,3)
```

alphaS.py

```
storage={}
def get_a(Q2):
    if Q2 not in storage:
        if par.mb2<=Q2:
            storage[Q2]=evolve_a(par.mb2,ab,Q2,5)
        elif par.mc2<=Q2 and Q2<par.mb2:
            storage[Q2]=evolve_a(par.mc2,ac,Q2,4)
        elif Q2<par.mc2:
            storage[Q2]=evolve_a(par.Q20,a0,Q2,3)
    return storage[Q2]

def get_alphaS(Q2):
    return get_a(Q2)*4*np.pi
```

alphaS.py

```
if __name__=='__main__':

    print '========================'
    print 'test alphaS evolution'
    print '========================'
    print 'Q2=1           alphaS=%0.5f'%get_alphaS(1.0)
    print 'Q2=(1+mc2)/2   alphaS=%0.5f'%get_alphaS(0.5*(1.0+par.mc2))
    print 'Q2=mc2         alphaS=%0.5f'%get_alphaS(par.mc2)
    print 'Q2=(mc2+mb2)/2 alphaS=%0.5f'%get_alphaS(0.5*(par.mc2+par.mb2))
    print 'Q2=mb2         alphaS=%0.5f'%get_alphaS(par.mb2)
    print 'Q2=(mb2+mZ2)/2 alphaS=%0.5f'%get_alphaS(0.5*(par.mb2+par.mZ2))
    print 'Q2=mZ2         alphaS=%0.5f'%get_alphaS(par.mZ2)
```

Next steps at command line

```
chmod +x alphaS.py
./alpha.py
```

Mellin transform

- Definition:

$$\boxed{F(N)} = \int dx x^{N-1} \boxed{f(x)} \quad \rightarrow \quad \boxed{f(x)} = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} dN x^{-N} \boxed{F(N)}$$

Mellin transform

- Definition:

$$\boxed{F(N)} = \int dx x^{N-1} \boxed{f(x)} \quad \rightarrow \quad \boxed{f(x)} = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} dN x^{-N} \boxed{F(N)}$$

- The c is chosen to be at the right of the rightmost pole of $F(N)$

Mellin transform

- Definition:

$$\boxed{F(N)} = \int dx x^{N-1} \boxed{f(x)} \quad \rightarrow \quad \boxed{f(x)} = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} dN x^{-N} \boxed{F(N)}$$

- The c is chosen to be at the right of the rightmost pole of $F(N)$

- Example:
  - $f(x) = x \rightarrow F(N) = \frac{1}{N+1}$
  - The rightmost pole is at $N = -1$ so $c > -1$

Mellin transform: numerical recipe

- Parametrize the contour

$$N(z) = c + ze^{i\phi}$$

Mellin transform: numerical recipe

- Parametrize the contour

$$N(z) = c + ze^{i\phi}$$

- A useful identity

$$f(x) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} dN x^{-N} F(N)$$

$$= \frac{1}{\pi}\int_{0}^{\infty} dz \mathrm{Im}\left[e^{i\phi} x^{-N(z)} F(N(z))\right]$$

Mellin transform: numerical recipe

- For efficiency purposes we use Gaussian quadrature, i.e

$$\int_{-1}^{1} dx\ g(x) \approx \sum_{i=1}^{n} w_i g(x_i)$$

Mellin transform: numerical recipe

- For efficiency purposes we use Gaussian quadrature, i.e

$$\int_{-1}^{1} dx \; g(x) \approx \sum_{i=1}^{n} w_i g(x_i)$$

- For an arbitrary integration region we can use

$$\int_a^b dz \; g(z) \approx \frac{b-a}{2} \sum_{i=1}^{n} w_i \; g\left(\frac{1}{2}(b-a)x_i + \frac{1}{2}(a+b)\right)$$

Mellin transform: numerical recipe

- Back to inverse Mellin transform

$$f(x) = \frac{1}{\pi} \int_0^\infty dz \; \mathrm{Im}\left[e^{i\phi} x^{-N(z)} F(N(z)) \right]$$

Mellin transform: numerical recipe

- Back to inverse Mellin transform

$$f(x) = \frac{1}{\pi} \int_0^\infty dz \; \mathrm{Im} \left[e^{i\phi} x^{-N(z)} F(N(z)) \right]$$

- We partition the z integration in k subintervals and apply Gaussian-quadrature on each subinterval

$$\boxed{f(x) \approx \frac{1}{\pi} \sum_{j=1}^{k} \frac{1}{2} \left(z^j_{\mathrm{max}} - z^j_{\mathrm{min}} \right) \sum_i w_i \mathrm{Im} \; \left[e^{i\phi} x^{-N(z_i^j)} F(N(z_i^j)) \right]}$$

$$z_i^j = \frac{1}{2} \left[\left(z^j_{\mathrm{max}} - z^j_{\mathrm{min}} \right) x_i + \left(z^j_{\mathrm{max}} + z^j_{\mathrm{min}} \right) \right]$$

QCD carpentry: coding up Mellin transforms

- mellin.py

mellin.py

```
#!/usr/bin/env python
import sys,os
import numpy as np

c=1.9   #--contour crossing
npts=8  #--number of gaussian quadrature

#--define intervals for z integration
znodes=[0,0.1,0.3,0.6,1.0,1.6,2.4,3.5,5,7,10,14,19,25,32,40,50,63]

#--compute gaussian nodes and weights
x,w=np.polynomial.legendre.leggauss(npts)
```

mellin.py

```
#--generate z and w values along coutour
Z,W,JAC=[],[],[]
for i in range(len(znodes)-1):
    a,b=znodes[i],znodes[i+1]
    Z.extend(0.5*(b-a)*x+0.5*(a+b))
    W.extend(w)
    JAC.extend([0.5*(b-a) for j in range(x.size)])
Z=np.array(Z)
W=np.array(W)
JAC=np.array(JAC)
```

mellin.py

```python
def invert(x,F):
    return np.sum(np.imag(phase * x**(-N) * F)/np.pi * W * JAC)

if __name__=='__main__':

    F=1/(N+1)
    f=lambda x: x
    X=10**np.linspace(-5,-1,10)
    for x in X:
        print 'x=%10.4e  f=%10.4e  inv=%10.4e'%(x,f(x),invert(x,F))
```

Next steps at command line

```
chmod +x mellin.py
./mellin.py
```

Mellin transform:

- Why are we talking about Mellin transforms?

  It converts convolutions into ordinary products

- Example:

$$\sigma(z) = \int_z^1 \frac{d\xi}{\xi} h(\xi) f\left(\frac{z}{\xi}\right)$$

$$\Sigma(N) = \int_z^1 dz z^{N-1} \sigma(z) = H(N)\ F(N)$$

Solving the DGLAP equations

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi, \mu) = \sum_{j'} \int_\xi^1 \frac{dz}{z} P_{jj'}(z, g) f_{j'/H}(\xi/z, \mu)}$$

Solving the DGLAP equations

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi, \mu) = \sum_{j'} \int_{\xi}^{1} \frac{dz}{z} P_{jj'}(z, g) f_{j'/H}(\xi/z, \mu)}$$

- We will solve them using 4 simplifications

Solving the DGLAP equations

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi, \mu) = \sum_{j'} \int_{\xi}^{1} \frac{dz}{z} P_{jj'}(z, g) f_{j'/H}(\xi/z, \mu)}$$

- We will solve them using 4 simplifications
  - Solve DGLAP in Mellin space → ordinary diff. eqs.

Solving the DGLAP equations

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi, \mu) = \sum_{j'} \int_\xi^1 \frac{dz}{z} P_{jj'}(z, g) f_{j'/H}(\xi/z, \mu)}$$

- We will solve them using 4 simplifications

  - Solve DGLAP in Mellin space → ordinary diff. eqs.
  - Flavor composition → non-singlets and singlet

Solving the DGLAP equations

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi, \mu) = \sum_{j'} \int_{\xi}^{1} \frac{dz}{z} P_{jj'}(z, g) f_{j'/H}(\xi/z, \mu)}$$

- We will solve them using 4 simplifications
  - Solve DGLAP in Mellin space → ordinary diff. eqs.
  - Flavor composition → non-singlets and singlet
  - Solve the equation in terms of a_S

Solving the DGLAP equations

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi,\mu) = \sum_{j'} \int_{\xi}^{1} \frac{dz}{z} P_{jj'}(z,g) f_{j'/H}(\xi/z,\mu)}$$

- We will solve them using 4 simplifications
  - Solve DGLAP in Mellin space → ordinary diff. eqs.
  - Flavor composition → non-singlets and singlet
  - Solve the equation in terms of a_S
  - Use LO splitting kernels ($P_{jj'}$)

DGLAP in Mellin space

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi,\mu) = \sum_{j'} \int_\xi^1 \frac{dz}{z} P_{jj'}(z,g) f_{j'/H}(\xi/z,\mu)}$$

- The Mellin transform is

$$\boxed{\frac{\partial}{\partial \ln \mu^2} F_{j/H}(N,\mu) = \sum_{j'} P_{jj'}(N,\mu)\ F_{j'/H}(N,\mu)}$$

DGLAP in Mellin space

$$\boxed{\frac{\partial}{\partial \ln \mu^2} f_{j/H}(\xi,\mu) = \sum_{j'} \int_{\xi}^{1} \frac{dz}{z} P_{jj'}(z,g) f_{j'/H}(\xi/z,\mu)}$$

- The Mellin transform is

$$\boxed{\frac{\partial}{\partial \ln \mu^2} F_{j/H}(N,\mu) = \sum_{j'} P_{jj'}(N,\mu)\ F_{j'/H}(N,\mu)}$$

- In Mellin space DGLAP is just an ordinary system of differential equations

DGLAP with flavor composition

- Let's write $F_{j/H}(N) = F_j$ and $F_{q^\pm} = F_q \pm F_{\bar{q}}$

DGLAP with flavor composition

- Let's write $F_{j/H}(N) = F_j$ and $F_{q^\pm} = F_q \pm F_{\bar{q}}$
- The flavor composition

$$
\begin{aligned}
F_{\pm 3} &= F_{u^\pm} - F_{d^\pm}, \\
F_{\pm 8} &= F_{u^\pm} + F_{d^\pm} - 2F_{s^\pm}, \\
F_{\pm 15} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} - 3F_{c^\pm}, \\
F_{\pm 24} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} + F_{c^\pm} - 4F_{b^\pm}, \\
F_{\pm 35} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} + F_{c^\pm} + F_{b^\pm}, \\
F_{\pm} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} + F_{c^\pm} + F_{b^\pm}
\end{aligned}
$$

DGLAP with flavor composition

- Let's write $F_{j/H}(N) = F_j$ and $F_{q^\pm} = F_q \pm F_{\bar{q}}$
- The flavor composition

$$\begin{aligned}
F_{\pm 3} &= F_{u^\pm} - F_{d^\pm}, \\
F_{\pm 8} &= F_{u^\pm} + F_{d^\pm} - 2F_{s^\pm}, \\
F_{\pm 15} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} - 3F_{c^\pm}, \\
F_{\pm 24} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} + F_{c^\pm} - 4F_{b^\pm}, \\
F_{\pm 35} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} + F_{c^\pm} + F_{b^\pm}, \\
F_{\pm} &= F_{u^\pm} + F_{d^\pm} + F_{s^\pm} + F_{c^\pm} + F_{b^\pm}
\end{aligned}$$

- Basically we go from $F_j \to F_{\pm j}, F_\pm, F_g$

DGLAP with flavor composition

- The flavor composition allows us to split DGLAP

$$\frac{\partial F_{\pm j}}{\partial \ln \mu^2} = P_{\mathrm{NS}}^{\pm} F_{\pm j}$$

Non-singlet

$$\frac{\partial F_{-}}{\partial \ln \mu^2} = P_{\mathrm{NS}}^{-} F_{-}$$

Non-singlet

$$\frac{\partial}{\partial \ln \mu^2} \begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \begin{pmatrix} P_{qq} & P_{qg} \\ P_{gq} & P_{gg} \end{pmatrix} \begin{pmatrix} F_+ \\ F_g \end{pmatrix}$$

Singlet

DGLAP in terms of a_S and LO expansion

- The μ dependence in $P_{jj'}$ is only via a_S

$$P_{ij}(a_S) = \sum_{m=0}^{\infty} a_S^{m+1}(\mu) P_{ij}^{(m)}$$

DGLAP in terms of a_S and LO expansion

- The μ dependence in $P_{jj'}$ is only via a_S

$$P_{ij}(a_S) = \sum_{m=0}^{\infty} a_S^{m+1}(\mu) P_{ij}^{(m)}$$

- So we can rewrite DGLAP in terms of a_S

$$\begin{aligned}\frac{\partial}{\partial a_S} F_j(a_S) &= \frac{1}{\beta(a_S)} \sum_{j'} P_{jj'}(a_S)\ F_{j'}(a_S) \\ &= -\frac{1}{\beta_0 a_S} \sum_{j'} P_{jj'}^{(0)}\ F_{j'}(a_S) + O(a_S)\end{aligned}$$

Solving the non-singlet at LO

$$\frac{\partial F_{\pm j}}{\partial \ln \mu^2} = P^{\pm}_{\mathrm{NS}} F_{\pm j}$$

Solving the non-singlet at LO

$$\frac{\partial F_{\pm j}}{\partial \ln \mu^2} = P^{\pm}_{\mathrm{NS}} F_{\pm j} \quad \rightarrow \quad \frac{\partial F_{\pm j}}{\partial a_S} = -\frac{1}{\beta_0 a_S} P^{\pm(0)}_{\mathrm{NS}} F_{\pm j}$$

Solving the non-singlet at LO

$$\frac{\partial F_{\pm j}}{\partial \ln \mu^2} = P_{\rm NS}^{\pm} F_{\pm j} \quad \rightarrow \quad \frac{\partial F_{\pm j}}{\partial a_S} = -\frac{1}{\beta_0 a_S} P_{\rm NS}^{\pm(0)} F_{\pm j}$$

- The solution → evolution of $F_\pm$ from μ_0 to μ

$$\boxed{F_{\pm j}(a_S) = \left(\frac{a_S}{a_0}\right)^{-P_{\rm NS}^{\pm(0)}/\beta_0} F_{\pm j}(a_0)}$$

Solving the singlet at LO

$$\frac{\partial}{\partial \ln \mu^2} \begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \begin{pmatrix} P_{qq} & P_{qg} \\ P_{gq} & P_{gg} \end{pmatrix} \begin{pmatrix} F_+ \\ F_g \end{pmatrix}$$

Solving the singlet at LO

$$\frac{\partial}{\partial \ln \mu^2}\begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \begin{pmatrix} P_{qq} & P_{qg} \\ P_{gq} & P_{gg} \end{pmatrix}\begin{pmatrix} F_+ \\ F_g \end{pmatrix} \quad \to \quad \frac{\partial}{\partial a_S}\begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \frac{-1}{\beta_0 a_S}\begin{pmatrix} P_{qq}^{(0)} & P_{qg}^{(0)} \\ P_{gq}^{(0)} & P_{gg}^{(0)} \end{pmatrix}\begin{pmatrix} F_+ \\ F_g \end{pmatrix}$$

Solving the singlet at LO

$$\frac{\partial}{\partial \ln \mu^2} \begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \begin{pmatrix} P_{qq} & P_{qg} \\ P_{gq} & P_{gg} \end{pmatrix} \begin{pmatrix} F_+ \\ F_g \end{pmatrix} \quad \rightarrow \quad \frac{\partial}{\partial a_S} \begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \frac{-1}{\beta_0 a_S} \begin{pmatrix} P_{qq}^{(0)} & P_{qg}^{(0)} \\ P_{gq}^{(0)} & P_{gg}^{(0)} \end{pmatrix} \begin{pmatrix} F_+ \\ F_g \end{pmatrix}$$

- Apply eigenvalue decomposition

$$\begin{aligned} \boldsymbol{R}_0 &= \frac{1}{\beta_0} \begin{pmatrix} P_{qq}^{(0)} & P_{qg}^{(0)} \\ P_{gq}^{(0)} & P_{gg}^{(0)} \end{pmatrix} = r_- \boldsymbol{e}_- + r_+ \boldsymbol{e}_+ \\ \boldsymbol{e}_\pm &= \frac{1}{r_\pm - r_\mp} \left[\boldsymbol{R}_0 - r_\mp \boldsymbol{I} \right] \\ r_\pm &= \frac{1}{2\beta_0} \left[P_{qq}^{(0)} + P_{gg}^{(0)} \pm \sqrt{\left(P_{qq}^{(0)} - P_{gg}^{(0)} \right)^2 + 4 P_{qg}^{(0)} P_{gq}^{(0)}} \right] \end{aligned}$$

Solving the singlet at LO

$$\frac{\partial}{\partial \ln \mu^2}\begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \begin{pmatrix} P_{qq} & P_{qg} \\ P_{gq} & P_{gg} \end{pmatrix}\begin{pmatrix} F_+ \\ F_g \end{pmatrix} \quad \rightarrow \quad \frac{\partial}{\partial a_S}\begin{pmatrix} F_+ \\ F_g \end{pmatrix} = \frac{-1}{\beta_0 a_S}\begin{pmatrix} P^{(0)}_{qq} & P^{(0)}_{qg} \\ P^{(0)}_{gq} & P^{(0)}_{gg} \end{pmatrix}\begin{pmatrix} F_+ \\ F_g \end{pmatrix}$$

- Apply eigenvalue decomposition

$$\begin{aligned} \boldsymbol{R}_0 &= \frac{1}{\beta_0}\begin{pmatrix} P^{(0)}_{qq} & P^{(0)}_{qg} \\ P^{(0)}_{gq} & P^{(0)}_{gg} \end{pmatrix} = r_-\boldsymbol{e}_- + r_+\boldsymbol{e}_+ \\ \boldsymbol{e}_\pm &= \frac{1}{r_\pm - r_\mp}\left[\boldsymbol{R}_0 - r_\mp \boldsymbol{I}\right] \\ r_\pm &= \frac{1}{2\beta_0}\left[P^{(0)}_{qq} + P^{(0)}_{gg} \pm \sqrt{\left(P^{(0)}_{qq} - P^{(0)}_{gg}\right)^2 + 4P^{(0)}_{qg}P^{(0)}_{gq}}\right] \end{aligned}$$

$$\boxed{\begin{pmatrix} F_+(a_S) \\ F_g(a_S) \end{pmatrix} = \left[\boldsymbol{e}_-\left(\frac{a_S}{a_0}\right)^{-r_-} + \boldsymbol{e}_+\left(\frac{a_S}{a_0}\right)^{-r_+}\right]\begin{pmatrix} F_+(a_0) \\ F_g(a_0) \end{pmatrix}}$$

LO splitting functions

$$P_{\mathrm{NS}}^{\pm(0)} = P_{qq}^{(0)} = C_F \left[3 + \frac{2}{N(N+1)} - 4S_1(N) \right]$$

$$P_{qg}^{(0)} = 2N_f \frac{N^2+N+2}{N(N+1)(N+2)}$$

$$P_{gq}^{(0)} = 2C_F \frac{N^2+N+2}{(N-1)N(N+1)}$$

$$P_{gg}^{(0)} = C_A \left[\frac{11}{3} + \frac{4}{N(N-1)} + \frac{4}{(N+1)(N+2)} - 4S_1(N) \right] - \frac{2}{3} N_f$$

$$S_1(N) = \sum_{j=1}^{N} \frac{1}{j} = \gamma_E + \psi^{(0)}(N+1)$$

LO Δsplitting functions

$$\Delta P_{\mathrm{NS}}^{\pm(0)} = \Delta P_{qq}^{(0)} = C_F \left[3 + \frac{2}{N(N+1)} - 4S_1(N)\right]$$

$$\Delta P_{qg}^{(0)} = 2N_f \frac{N-1}{N(N+1)}$$

$$\Delta P_{gq}^{(0)} = 2C_F \frac{N+2}{N(N+1)}$$

$$\Delta P_{gg}^{(0)} = C_A \left[\frac{11}{3} + \frac{8}{(N+1)(N+2)} - 4S_1(N)\right] - \frac{2}{3}N_f$$

Flavor decomposition

- After having evolved $F_{\pm j}, F_{\pm}, F_g$ we need to get back to the original flavor basis

Flavor decomposition

- After having evolved $F_{\pm j}, F_{\pm}, F_g$ we need to get back to the original flavor basis
- Here is how

$$
\begin{aligned}
F_{b^\pm} &= (F_{\pm 35} - F_{\pm 25})/5 \\
F_{c^\pm} &= F_{b^\pm} + (F_{\pm 24} - F_{\pm 15})/4 \\
F_{s^\pm} &= F_{c^\pm} + (F_{\pm 15} - F_{\pm 8})/3 \\
F_{d^\pm} &= F_{s^\pm} + (F_{\pm 8} - F_{\pm 3})/2 \\
F_{u^\pm} &= F_{s^\pm} + (F_{\pm 8} + F_{\pm 3})/2
\end{aligned}
$$

Flavor decomposition

- After having evolved $F_{\pm j}, F_{\pm}, F_g$ we need to get back to the original flavor basis
- Here is how

$$
\begin{aligned}
F_{b^\pm} &= (F_{\pm 35} - F_{\pm 25})/5 \\
F_{c^\pm} &= F_{b^\pm} + (F_{\pm 24} - F_{\pm 15})/4 \\
F_{s^\pm} &= F_{c^\pm} + (F_{\pm 15} - F_{\pm 8})/3 \\
F_{d^\pm} &= F_{s^\pm} + (F_{\pm 8} - F_{\pm 3})/2 \\
F_{u^\pm} &= F_{s^\pm} + (F_{\pm 8} + F_{\pm 3})/2
\end{aligned}
$$

- Finally $q = \frac{1}{2}(q^+ + q^-)$ and $\bar{q} = \frac{1}{2}(q^+ - q^-)$

Physical constraints

- PDF constraints

$$\int_0^1 dx(u - \bar{u}) = 2$$

$$\int_0^1 dx(d - \bar{d}) = 1$$

$$\int_0^1 dxx(g + u^+ + d^+ + s^+ + c^+ + b^+) = 1$$

Physical constraints

- PDF constraints

$$\int_0^1 dx(u-\bar{u}) = 2$$
$$\int_0^1 dx(d-\bar{d}) = 1$$
$$\int_0^1 dxx(g+u^+ + d^+ + s^+ + c^+ + b^+) = 1$$

- ΔPDF constraints from $SU(3)_F$ baryon octect

$$\int_0^1 dx(\Delta u^+ - \Delta d^+) = a_3 = g_A \sim 1.269 \to SU(2)$$
$$\int_0^1 dx(\Delta u + \Delta d - 2\Delta s) = a_8 \sim 0.586 \to SU(3)$$

DIS observables

- For upol. DIS we will focus on $F_2^{(p,d)}$ experimental data. At LO the structure function is simply

$$F_2(x, Q^2) = x \sum_q e_q^2 f_q(x, Q^2)$$

DIS observables

- For upol. DIS we will focus on $F_2^{(p,d)}$ experimental data. At LO the structure function is simply

$$F_2(x,Q^2) = x\sum_q e_q^2 f_q(x,Q^2)$$

- For pol. DIS we will focus on $A_1^{(p,d)} = g_1/F_1$ experimental data. At LO the structure functions are given by

$$g_1(x,Q^2) = \frac{1}{2}\sum_q e_q^2 \Delta f_q(x,Q^2)$$

$$F_1(x,Q^2) = \frac{(1+4\frac{M^2}{Q^2}x^2)}{2x} F_2(x,Q^2)$$

PDF and $\triangle$PDF parametrizations

- We parametrize PDFs and ΔPDFs as follow

$$f_q(x,\mu_0) = M\frac{x^\alpha(1-x)^\beta P(x;c,d)}{\int_0^1 dz z^n z^\alpha (1-z)^\beta P(z;c,d)}$$

$$P(x;c,d) = c\,\sqrt{x} + d\,x$$

PDF and ∆PDF parametrizations

- We parametrize PDFs and ∆PDFs as follow

$$f_q(x,\mu_0) = M\frac{x^{\alpha}(1-x)^{\beta}P(x;c,d)}{\int_0^1 dz z^n z^{\alpha}(1-z)^{\beta}P(z;c,d)}$$
$$P(x;c,d) = c\,\sqrt{x} + d\;x$$

- Mellin moments of f_q are then by

$$F_q(N,\mu_0) = M\frac{B(N+a,b+1)+B(N+a+1/2,b+1)+B(N+a+1,b+1))}{B(n+a,b+1)+B(n+a+1/2,b+1)+B(n+a+1,b+1))}$$

QCD carpentry:
coding up DGLAP

- pdf.py
- ppdf.py
- idis.py

pdf.py

```python
#!/usr/bin/env python
import sys

#--external libs
import numpy as np
from scipy.special import digamma
from scipy.special import gamma

#--local libs
from params import euler,CF,CA
import mellin
import params as par
import alphaS as aS

N=mellin.N    #--retrieve mellin contour
beta=aS.beta  #--retrieve beta coeffs
```

pdf.py

```
#--compute splitting functions
S1=euler + digamma(N+1)
PNS=np.zeros((6,N.size),dtype=complex)
Pqq=np.zeros((6,N.size),dtype=complex)
Pqg=np.zeros((6,N.size),dtype=complex)
Pgq=np.zeros((6,N.size),dtype=complex)
Pgg=np.zeros((6,N.size),dtype=complex)
P=np.zeros((6,2,2,N.size),dtype=complex)
for Nf in range(3,6):
    Pqq[Nf]=CF*(3+2/N/(N+1)-4*S1)
    Pqg[Nf]=2*Nf*(N**2+N+2)/(N*(N+1)*(N+2))
    Pgq[Nf]=2*CF*(N**2+N+2)/(N-1)/N/(N+1)
    Pgg[Nf]=CA*(11/3.0+4/N/(N-1)+4/(N+1)/(N+2)-4*S1)-2/3.0*Nf

    P[Nf,0,0,:] = Pqq[Nf]
    P[Nf,0,1,:] = Pqg[Nf]
    P[Nf,1,0,:] = Pgq[Nf]
    P[Nf,1,1,:] = Pgg[Nf]
    PNS[Nf]=Pqq[Nf]
```

pdf.py

```
#--construct evolution operators
RNS=np.zeros((6,N.size),dtype=complex)
RS =np.zeros((6,2,2,N.size),dtype=complex)
rp=np.zeros((6,N.size),dtype=complex)
rm=np.zeros((6,N.size),dtype=complex)

for Nf in range(3,6):
    RNS[Nf]=PNS[Nf]/beta[Nf,0]
    RS[Nf]=P[Nf]/beta[Nf,0]
    rp[Nf]=Pqq[Nf]+Pgg[Nf]+np.sqrt((Pqq[Nf]-Pgg[Nf])**2+4*Pqg[Nf]*Pgq[Nf])
    rm[Nf]=Pqq[Nf]+Pgg[Nf]-np.sqrt((Pqq[Nf]-Pgg[Nf])**2+4*Pqg[Nf]*Pgq[Nf])
    rp[Nf]/=2*beta[Nf,0]
    rm[Nf]/=2*beta[Nf,0]
```

pdf.py

```
ep =np.zeros((6,2,2,N.size),dtype=complex)
em =np.zeros((6,2,2,N.size),dtype=complex)

for Nf in range(3,6):

    ep[Nf,0,0]=(RS[Nf,0,0]-rm[Nf])/(rp[Nf]-rm[Nf])
    ep[Nf,1,1]=(RS[Nf,1,1]-rm[Nf])/(rp[Nf]-rm[Nf])
    ep[Nf,0,1]=RS[Nf,0,1]/(rp[Nf]-rm[Nf])
    ep[Nf,1,0]=RS[Nf,1,0]/(rp[Nf]-rm[Nf])

    em[Nf,0,0]=(RS[Nf,0,0]-rp[Nf])/(rm[Nf]-rp[Nf])
    em[Nf,1,1]=(RS[Nf,1,1]-rp[Nf])/(rm[Nf]-rp[Nf])
    em[Nf,0,1]=RS[Nf,0,1]/(rm[Nf]-rp[Nf])
    em[Nf,1,0]=RS[Nf,1,0]/(rm[Nf]-rp[Nf])
```

pdf.py

```
#--evolution functions

def evolve_nonsinglet(FNS0,Q20,Q2,Nf):
    a0 = aS.get_a(Q20)
    a  = aS.get_a(Q2)
    L=np.power(a/a0,-RNS[Nf])
    return L*FNS0

def evolve_singlet(FS0,Q20,Q2,Nf):
    a0 = aS.get_a(Q20)
    a  = aS.get_a(Q2)
    L= np.power(a/a0,-rm[Nf])*em[Nf] + np.power(a/a0,-rp[Nf])*ep[Nf]
    return np.einsum('ij...,j...->i...',L,FS0,dtype=complex)
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):

    Fm = evolve_nonsinglet(BC['Fm'],Q20,Q2,Nf)
    FS = evolve_singlet(BC['FS'],Q20,Q2,Nf)
    Fp = FS[0]
    g  = FS[1]

    Fm_={}
    Fp_={}

    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):
    ...
    if   Nf==3:
         Fm_[3] =evolve_nonsinglet(BC['Fm_'][3],Q20,Q2,Nf)
         Fm_[8] =evolve_nonsinglet(BC['Fm_'][8],Q20,Q2,Nf)
         Fm_[15]=Fm
         Fm_[24]=Fm
         Fm_[35]=Fm
         Fp_[3] =evolve_nonsinglet(BC['Fp_'][3],Q20,Q2,Nf)
         Fp_[8] =evolve_nonsinglet(BC['Fp_'][8],Q20,Q2,Nf)
         Fp_[15]=Fp
         Fp_[24]=Fp
         Fp_[35]=Fp
    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):
    ...
    elif Nf==4:
         Fm_[3] =evolve_nonsinglet(BC['Fm_'][3] ,Q20,Q2,Nf)
         Fm_[8] =evolve_nonsinglet(BC['Fm_'][8] ,Q20,Q2,Nf)
         Fm_[15]=evolve_nonsinglet(BC['Fm_'][15],Q20,Q2,Nf)
         Fm_[24]=Fm
         Fm_[35]=Fm
         Fp_[3] =evolve_nonsinglet(BC['Fp_'][3] ,Q20,Q2,Nf)
         Fp_[8] =evolve_nonsinglet(BC['Fp_'][8] ,Q20,Q2,Nf)
         Fp_[15]=evolve_nonsinglet(BC['Fp_'][15],Q20,Q2,Nf)
         Fp_[24]=Fp
         Fp_[35]=Fp
    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):
    ...
    elif Nf==5:
         Fm_[3] =evolve_nonsinglet(BC['Fm_'][3] ,Q20,Q2,Nf)
         Fm_[8] =evolve_nonsinglet(BC['Fm_'][8] ,Q20,Q2,Nf)
         Fm_[15]=evolve_nonsinglet(BC['Fm_'][15],Q20,Q2,Nf)
         Fm_[24]=evolve_nonsinglet(BC['Fm_'][24],Q20,Q2,Nf)
         Fm_[35]=Fm
         Fp_[3] =evolve_nonsinglet(BC['Fp_'][3] ,Q20,Q2,Nf)
         Fp_[8] =evolve_nonsinglet(BC['Fp_'][8] ,Q20,Q2,Nf)
         Fp_[15]=evolve_nonsinglet(BC['Fp_'][15],Q20,Q2,Nf)
         Fp_[24]=evolve_nonsinglet(BC['Fp_'][24],Q20,Q2,Nf)
         Fp_[35]=Fp
    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):
    ...
    #--flav decomposition
    qp =np.zeros((6,N.size),dtype=complex)
    qm =np.zeros((6,N.size),dtype=complex)

    qm[5]=(Fm_[35]-Fm_[24])/5.
    qm[4]=qm[5]+(Fm_[24]-Fm_[15])/4.
    qm[3]=qm[4]+(Fm_[15]-Fm_[8])/3.
    qm[2]=qm[3]+(Fm_[8]-Fm_[3])/2.
    qm[1]=qm[3]+(Fm_[8]+Fm_[3])/2.

    qp[5]=(Fp_[35]-Fp_[24])/5.
    qp[4]=qp[5]+(Fp_[24]-Fp_[15])/4.
    qp[3]=qp[4]+(Fp_[15]-Fp_[8])/3.
    qp[2]=qp[3]+(Fp_[8]-Fp_[3])/2.
    qp[1]=qp[3]+(Fp_[8]+Fp_[3])/2.
    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):

    ...

    output={}

    #--generate output for BC
    output['Fm_'] = Fm_
    output['Fp_'] = Fp_
    output['Fm']  = Fm
    output['Fp']  = Fp

    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):

    ...

    #--generate output for pheno
    output['up']=qp[1]
    output['dp']=qp[2]
    output['sp']=qp[3]
    output['cp']=qp[4]
    output['bp']=qp[5]
    output['um']=qm[1]
    output['dm']=qm[2]
    output['sm']=qm[3]
    output['cm']=qm[4]
    output['bm']=qm[5]

    ...
```

pdf.py

```
def _evolve(BC,Q20,Q2,Nf):

    ...

    output['g'] =g
    output['u']=0.5*(qp[1]+qm[1])
    output['d']=0.5*(qp[2]+qm[2])
    output['s']=0.5*(qp[3]+qm[3])
    output['c']=0.5*(qp[4]+qm[4])
    output['b']=0.5*(qp[5]+qm[5])
    output['ub']=0.5*(qp[1]-qm[1])
    output['db']=0.5*(qp[2]-qm[2])
    output['sb']=0.5*(qp[3]-qm[3])
    output['cb']=0.5*(qp[4]-qm[4])
    output['bb']=0.5*(qp[5]-qm[5])

    return output
```

pdf.py

```
def get_flav_comp(g,up,um,dp,dm,sp,sm,cp,cm,bp,bm):

    Fm_={}
    Fm_[35]= bm + cm + dm + sm + um
    Fm_[24]= -4*bm + cm + dm + sm + um
    Fm_[15]= -3*cm + dm + sm + um
    Fm_[8] = dm - 2*sp + 2*(-sm + sp) + um
    Fm_[3] = -dm + um
    Fm_[0] = np.zeros(N.size,dtype=complex)

    Fp_={}
    Fp_[0] = np.zeros(N.size,dtype=complex)
    Fp_[3] = -dp + up
    Fp_[8] = dp - 2*sp + up
    Fp_[15]= -3*cp + dp + sp + up
    Fp_[24]= -4*bp + cp + dp + sp + up
    Fp_[35]= bp + cp + dp + sp + up

    ...
```

pdf.py

```python
def get_flav_comp(g,up,um,dp,dm,sp,sm,cp,cm,bp,bm):

    ...

    Fp = bp + cp + dp + sp + up
    Fm = bm + cm + dm + sm + um
    FS = np.zeros((2,N.size),dtype=complex)
    FS[0]=Fp
    FS[1]=g

    output={}
    output['Fm_'] = Fm_
    output['Fp_'] = Fp_
    output['Fm']  = Fm
    output['Fp']  = Fp
    output['FS']  = FS
    return output
```

pdf.py

```
#--initialize where to store the boundary conditions
BC3={}
BC4={}
BC5={}

def get_BC(moms):
    #-- here moms are the moments at the input scale

    global BC3,BC4,BC5

    zero=np.zeros(N.size,dtype=complex)

    ...
```

pdf.py

```python
def get_BC(moms):
    ...
    #--BC for Nf=3
    g   = moms['g']
    up  = moms['up']
    um  = moms['um']
    dp  = moms['dp']
    dm  = moms['dm']
    sp  = moms['sp']
    sm  = moms['sm']
    cp  = zero
    cm  = zero
    bp  = zero
    bm  = zero
    BC3=get_flav_comp(g,up,um,dp,dm,sp,sm,cp,cm,bp,bm)

    ...
```

pdf.py

```
def get_BC(moms):
    ...
    #--BC for Nf=4
    BC4=_evolve(BC3,par.Q20,par.mc2,3)
    g =BC4['g']
    up=BC4['up']
    dp=BC4['dp']
    sp=BC4['sp']
    cp=BC4['cp']
    bp=BC4['bp']
    um=BC4['um']
    dm=BC4['dm']
    sm=BC4['sm']
    cm=BC4['cm']
    bm=BC4['bm']
    BC4.update(get_flav_comp(g,up,um,dp,dm,sp,sm,cp,cm,bp,bm))

    ...
```

pdf.py

```python
def get_BC(moms):
    ...
    #--BC for Nf=5
    BC5=_evolve(BC4,par.mc2,par.mb2,4)
    g =BC5['g']
    up=BC5['up']
    dp=BC5['dp']
    sp=BC5['sp']
    cp=BC5['cp']
    bp=BC5['bp']
    um=BC5['um']
    dm=BC5['dm']
    sm=BC5['sm']
    cm=BC5['cm']
    bm=BC5['bm']
    BC5.update(get_flav_comp(g,up,um,dp,dm,sp,sm,cp,cm,bp,bm))
```

pdf.py

```
#--parametrization
#--f(x) = norm * x**a * (1-x)**b * (1+c*x**0.5+d*x)

params={}
params['g']   =np.array([1,-0.5,3,0,0])
params['uv']  =np.array([1, 0.5,3,0,0])
params['dv']  =np.array([1, 0.5,4,0,0])
params['sea'] =np.array([1, 0.5,4,0,0])
params['db']  =np.array([1,-0.5,6,0,0])
params['ub']  =np.array([1,-0.5,6,0,0])
params['s']   =np.array([1,-0.5,6,0,0])
params['sb']  =np.array([1,-0.5,6,0,0])
```

pdf.py

```
parmin=np.zeros(9)
parmax=np.zeros(9)
parmin[0],parmax[0] = -1.9,2   # g_a
parmin[1],parmax[1] =  0.0,10  # g_b
parmin[2],parmax[2] = -0.9,2   # uv_a
parmin[3],parmax[3] =  0.0,10  # uv_b
parmin[4],parmax[4] = -0.9,2   # dv_a
parmin[5],parmax[5] =  0.0,10  # dv_b
parmin[6],parmax[6] =  0.0,100 # sea_N
parmin[7],parmax[7] = -1.9,2   # sea_a
parmin[8],parmax[8] =  0.0,10  # sea_b
```

pdf.py

```
#--hooks
def get_params():
    p=[]
    p=np.append(p,params['g'][1:3])
    p=np.append(p,params['uv'][1:3])
    p=np.append(p,params['dv'][1:3])
    p=np.append(p,params['sea'][:3])
    return p

def set_params(p):
    params['g'][1:3]   = p[0:2]
    params['uv'][1:3]  = p[2:4]
    params['dv'][1:3]  = p[4:6]
    params['sea'][0:3] = p[6:9]
```

pdf.py

```
def beta(a,b):
    return gamma(a)*gamma(b)/gamma(a+b)

def get_moments(flav,iN=None):
    if iN==None: n=N
    else:        n=iN
    M,a,b,c,d=params[flav]
    norm=beta(2+a,b+1)+c*beta(2+a+0.5,b+1)+d*beta(2+a+1.0,b+1)
    mom=beta(n+a,b+1)+c*beta(n+a+0.5,b+1)+d*beta(n+a+1.0,b+1)
    return M*mom/norm
```

pdf.py

```
def set_sumrules():

    #--valence
    params['uv'][0]=1
    params['uv'][0]=2/get_moments('uv',1)

    params['dv'][0]=1
    params['dv'][0]=1/get_moments('dv',1)

    #--msr
    sea=get_moments('sea',2)
    up=get_moments('uv',2)+2*(sea+get_moments('ub',2))
    dp=get_moments('dv',2)+2*(sea+get_moments('db',2))
    sp=(sea+get_moments('s',2))+(sea+get_moments('sb',2))
    params['g'][0]=1
    params['g'][0]=(1-up-dp-sp)/get_moments('g',2)
```

pdf.py

```
#--compute moments
storage={}
moms={}
def set_moms():
    sea=get_moments('sea')
    moms['g']  = get_moments('g')
    moms['up'] = get_moments('uv')+2*(sea+get_moments('ub'))
    moms['dp'] = get_moments('dv')+2*(sea+get_moments('db'))
    moms['sp'] = 2*sea+get_moments('s')+get_moments('sb')
    moms['um'] = get_moments('uv')
    moms['dm'] = get_moments('dv')
    moms['sm'] = np.zeros(N.size,dtype=complex)
    get_BC(moms)
    global storage
    storage={}
```

pdf.py

```
def evolve(Q2):
    if Q2 not in storage:
      if par.mb2<Q2:
          storage[Q2]=_evolve(BC5,par.mb2,Q2,5)
      elif par.mc2<=Q2 and Q2<=par.mb2:
          storage[Q2]=_evolve(BC4,par.mc2,Q2,4)
      elif Q2<par.mc2:
          storage[Q2]=_evolve(BC3,par.Q20,Q2,3)

def get_xF(x,Q2,flav):
    evolve(Q2)
    return x*mellin.invert(x,storage[Q2][flav])
```

pdf.py

```
if __name__=="__main__":

    set_sumrules()
    set_moms()
    x=0.5
    Q2=10
    print get_xF(x,Q2,'um')
    print get_xF(x,Q2,'u')
    print get_xF(x,Q2,'g')
```

Next steps at command line

```
chmod +x pdf.py
./pdf.py
```

ppdf.py

- copy pdf.py to ppdf.py
- change the splitting functions
- change the parametrization
- change the sum rules

ppdf.py

```
for Nf in range(3,6):
    Pqq[Nf]=CF*(3+2/N/(N+1)-4*S1)
    Pqg[Nf]=TR*Nf*4* (N-1)/N/(N+1)
    Pgq[Nf]=CF*2*(N+2)/N/(N+1)
    Pgg[Nf]=CA*(11/3. + 8/N/(N+1)-4*S1)-2/3.*Nf

    P[Nf,0,0,:] = Pqq[Nf]
    P[Nf,0,1,:] = Pqg[Nf]
    P[Nf,1,0,:] = Pgq[Nf]
    P[Nf,1,1,:] = Pgg[Nf]
    PNS[Nf]=Pqq[Nf]
```

ppdf.py

```python
params={}
params['g'] =np.array([1,-0.5,3,0,0])
params['up']=np.array([1, 0.5,3,0,0])
params['dp']=np.array([1, 0.5,3,0,0])
params['sp']=np.array([1,-0.5,3,0,0])
params['um']=np.array([1,-0.5,3,0,0])
params['dm']=np.array([1,-0.5,3,0,0])
params['sm']=np.array([0,-0.5,3,0,0])
```

ppdf.py

```
parmin=np.zeros(10)
parmax=np.zeros(10)
parmin[0],parmax[0] = -10,10   # g_N
parmin[1],parmax[1] = -1.9,2   # g_a
parmin[2],parmax[2] =  0.0,10  # g_b
parmin[3],parmax[3] = -1.9,2   # up_a
parmin[4],parmax[4] =  0.0,10  # up_b
parmin[5],parmax[5] = -1.9,2   # dp_a
parmin[6],parmax[6] =  0.0,10  # dp_b
parmin[7],parmax[7] = -10,10   # sp_N
parmin[8],parmax[8] = -1.9,2   # sp_a
parmin[9],parmax[9] =  0.0,10  # sp_b
```

ppdf.py

```python
def get_params():
    p=[]
    p=np.append(p,params['g'][:3])
    p=np.append(p,params['up'][1:3])
    p=np.append(p,params['dp'][1:3])
    p=np.append(p,params['sp'][:3])
    return p

def set_params(p):
    params['g'][:3]   = p[0:3]
    params['up'][1:3] = p[3:5]
    params['dp'][1:3] = p[5:7]
    params['sp'][:3]  = p[7:10]
```

ppdf.py

```
def set_sumrules():
    gA=1.269
    g8=0.586
    params['up'][0]=1
    params['up'][0]=(gA + g8)/2. + get_moments('sp',1)
    params['dp'][0]=1
    params['dp'][0]=(g8 - gA)/2. + get_moments('sp',1)

moms={}
def set_moms():
    moms['g']  = get_moments('g')
    moms['up'] = get_moments('up')
    moms['dp'] = get_moments('dp')
    moms['sp'] = get_moments('sp')
    moms['um'] = np.zeros(N.size,dtype=complex)
    moms['dm'] = np.zeros(N.size,dtype=complex)
    moms['sm'] = np.zeros(N.size,dtype=complex)
    get_BC(moms)
```

idis.py

```python
#!/usr/bin/env python
import sys
import alphaS as aS
import pdf,ppdf
from params import eU2,eD2,M2
import mellin
```

idis.py

```
def get_F2N(Q2,tar):
    """
    get moments of F2/x
    """
    Nf=aS.get_Nf(Q2)
    pdf.evolve(Q2)
    if tar=='p':
        up=pdf.storage[Q2]['up']
        dp=pdf.storage[Q2]['dp']
    elif tar=='n':
        dp=pdf.storage[Q2]['up']
        up=pdf.storage[Q2]['dp']
    sp=pdf.storage[Q2]['sp']
    cp=pdf.storage[Q2]['cp']
    bp=pdf.storage[Q2]['bp']

    F2N= eU2*(up+cp) + eD2*(sp+dp)
    if Nf>=4: F2N+=eU2*cp
    if Nf>=5: F2N+=eD2*bp
    return F2N
```

idis.py

```python
def get_F2(x,Q2,tar):
    if   tar=='p': F2N=get_F2N(Q2,'p')
    elif tar=='n': F2N=get_F2N(Q2,'n')
    elif tar=='d': F2N=0.5*(get_F2N(Q2,'p')+get_F2N(Q2,'n'))
    return x*mellin.invert(x,F2N)
```

idis.py

```
def get_g1N(Q2,tar):
    """
    get moments of g1N
    """
    Nf=aS.get_Nf(Q2)
    ppdf.evolve(Q2)
    if tar=='p':
        up=ppdf.storage[Q2]['up']
        dp=ppdf.storage[Q2]['dp']
    elif tar=='n':
        dp=ppdf.storage[Q2]['up']
        up=ppdf.storage[Q2]['dp']
    sp=ppdf.storage[Q2]['sp']
    cp=ppdf.storage[Q2]['cp']
    bp=ppdf.storage[Q2]['bp']

    g1= eU2*(up+cp) + eD2*(sp+dp)
    if Nf>=4: g1+=eU2*cp
    if Nf>=5: g1+=eD2*bp
    return 0.5*g1
```

idis.py

```
def get_g1(x,Q2,tar):
    if   tar=='p': g1N=get_g1N(Q2,'p')
    elif tar=='n': g1N=get_g1N(Q2,'n')
    elif tar=='d': g1N=0.5*(get_g1N(Q2,'p')+get_g1N(Q2,'n'))
    return mellin.invert(x,g1N)

def get_A1(x,Q2,tar):
    gamma2=4*M2*x**2/Q2
    gamma=gamma2**0.5
    rho2=1+gamma2
    g1=get_g1(x,Q2,tar)
    F2=get_F2(x,Q2,tar)
    F1=rho2*F2/(2*x)
    return g1/F1
```

idis.py

```
if __name__=='__main__':

    pdf.set_sumrules()
    pdf.set_moms()
    ppdf.set_sumrules()
    ppdf.set_moms()

    x=0.5
    Q2=10

    print get_F2(x,Q2,'p')
    print get_F2(x,Q2,'n')
    print get_F2(x,Q2,'d')
    print get_g1(x,Q2,'p')
    print get_g1(x,Q2,'n')
    print get_g1(x,Q2,'d')
    print get_A1(x,Q2,'p')
    print get_A1(x,Q2,'n')
    print get_A1(x,Q2,'d')
```

Next steps at command line

```
chmod +x idis.py
./idis.py
```

For later today

- Clone the following repo

  https://github.com/nobuosato/QCD-carpentry

For later today

- Clone the following repo

  https://github.com/nobuosato/QCD-carpentry

- Place your codes inside the repo
  - params.py
  - alphaS.py
  - mellin.py
  - pdf.py
  - ppdf.py

For later today

- Clone the following repo

  https://github.com/nobuosato/QCD-carpentry

- Place your codes inside the repo
  - params.py
  - alphaS.py
  - mellin.py
  - pdf.py
  - ppdf.py

- We will tune PDFs and ΔPDFs to DIS data