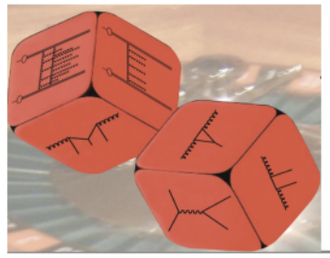
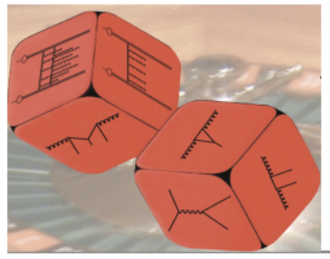


Summer School 2020

Terascale Summer School: Tutorial/Exercises - QCD and Monte Carlo techniques

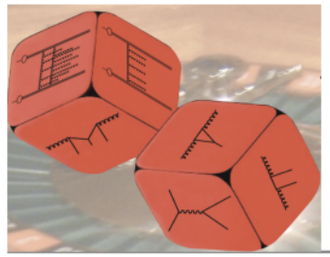


Summer School 2020

- Welcome to the tutorial exercises on Monte Carlo techniques and QCD

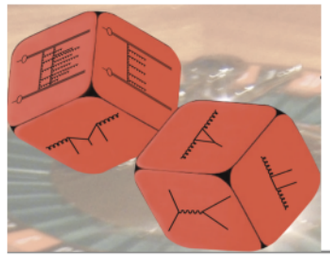
- Goal:
  - leave no one behind
  - everyone should be able to do the exercises
    - please ask questions

- additional infos under:
  - https://www.desy.de/~jung/Terascale_SummerSchool_2020_Tutorial



Summer School 2020

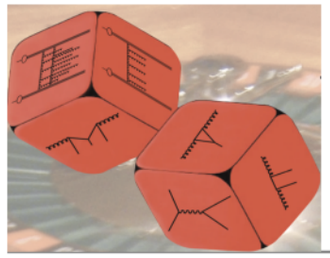
- Structure:
  - introductory lecture (20-30 min)
  - 3 working groups (in breakout-rooms) with tutors:
    - Sara Taheri Monfared
    - Qun Wang
    - Armando Bermudez Martinez
    - Luis Ignacio Estevez Banos
    - Mikel Mendizabal
    - Patrick Connor
  - ask questions
  - share screen to get help



Environment

- Virtual Machine
  - running full C++ code
  - compilation, execute program

- New:
  - running python on Jupyter Notebook
  - only need web browser
  - Login with School account on Computer-Farm at DESY
    - account will be valid for 2 weeks
    - Accounts on Google Doc
      - please pick an account and mark it (and remember your account and passwd)



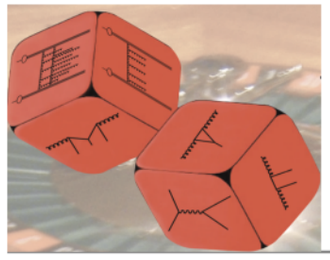
How to get started

- start VM
  - in case – passwd: terascale1234
  - perhaps change screen resolution

- compiling and running:

```
cd exercise-1
make example-1
./example-1
```

- templates are provided which include the general structure – you only have to fill the interesting – important parts

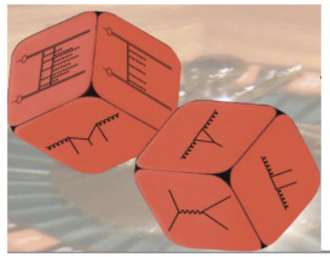


How to get started: Jupyter notebook

- open browser, and login to: https://naf-jhub.desy.de/
  - we have school accounts, valid for 2 weeks
    - Accounts on Google Doc
      - please pick an account and mark it (and remember your account and passwd)
    - (see https://confluence.desy.de/display/IS/Jupyter+on+NAF)

- to get started, logon jupyter cluster on naf
  - open terminal and copy templates and solutions:

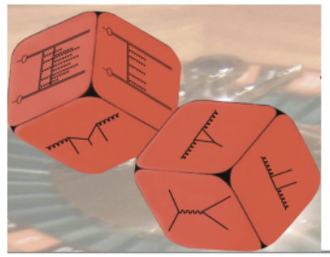
    wget
    https://www.desy.de/~jung/Terascale_SummerSchool_2020_Tutorial/Terascale_SummerSchool_2020_Tutorial_exericises_files/jupyter-python-all.tgz



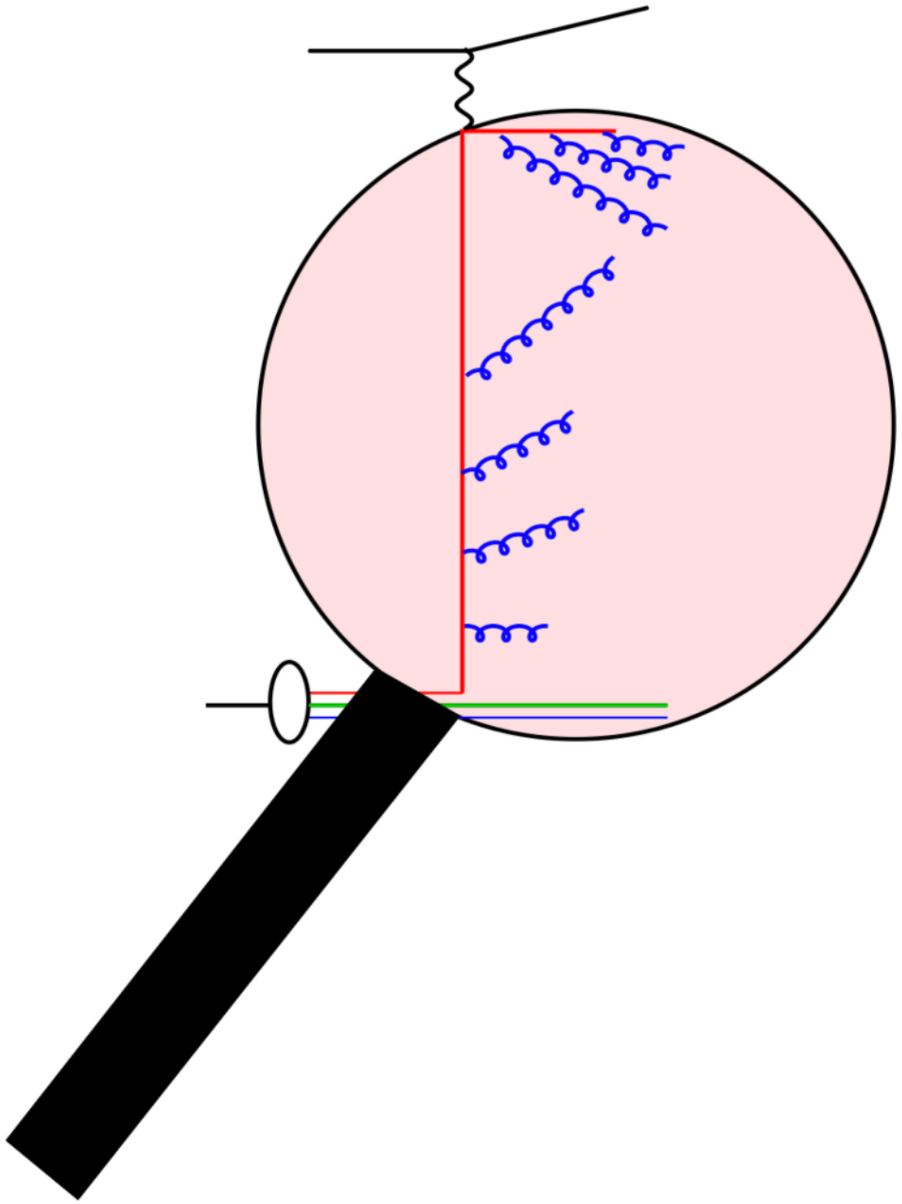
Exercise 1 - Introduction

Terascale Summer School: Tutorial/Exercises - QCD and Monte Carlo techniques

- Thursday 13. Aug 2020 - Exercise 1:
  - Random numbers
  - MC method
  - MC integration
- Monday 17. Aug 2020 – Exercise 1:
  - MC method
  - MC integration
- Tuesday 18. 2020 Aug - Exercise 2:
  - Sudakov form factor
- Wednesday 19 Aug. 2020 – Exercise 2 (cont'd)
  - MC solution of evolution equation
- Monday 24. Aug 2020 – Exercise 2 (cont'd)
  - MC solution of evolution equation
- Tuesday 25 Aug 2020 Exercise 3
  - Calculation & simulation of Higgs production
  - Using MC solution of evolution equation → calculation of pt spectrum of Higgs at LHC
- Wednesday 26 Aug – no Tutorial
- Thursday 27 Aug 2020 – Wrap up



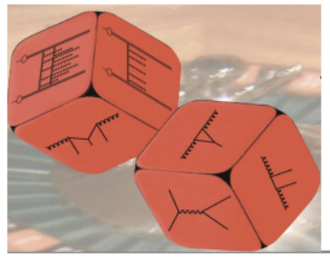
A proton in the initial state



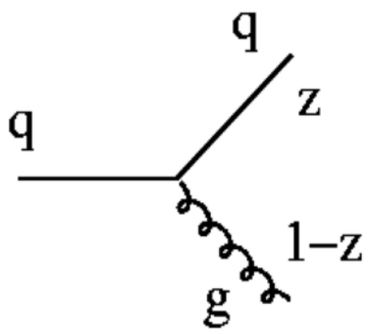
- Deep Inelastic Scattering is a incoherent sum of

  $e^+ q \to e + q$
- only 50 % of p momentum carried by quarks
- need a large gluon component
- partonic part convoluted with parton density function $f_i(x)$
- BUT we know, PDF depends on resolution scale Q^2

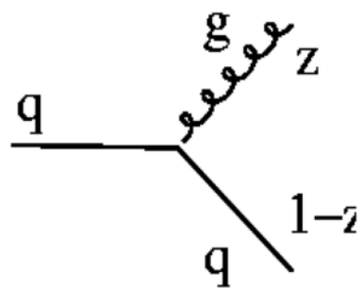
$$\sigma(e^+ p \to e^+ X) = \sum_i f_i(x, Q^2) \sigma(e^+ q_i \to e^+ q_i)$$



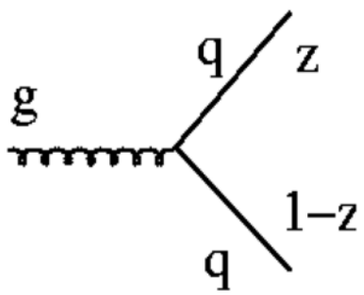
Splitting functions



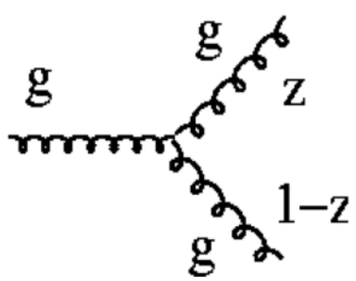
$$P_{qq} = \frac{4}{3}\left(\frac{1+z^2}{1-z}\right)$$



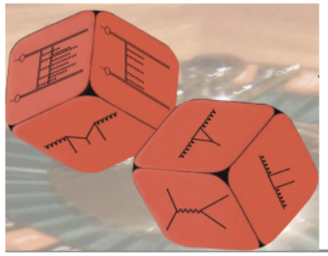
$$P_{gq} = \frac{4}{3}\left(\frac{1+(1-z)^2}{z}\right)$$



$$P_{qg} = \frac{1}{2}\left(z^2+(1-z)^2\right)$$

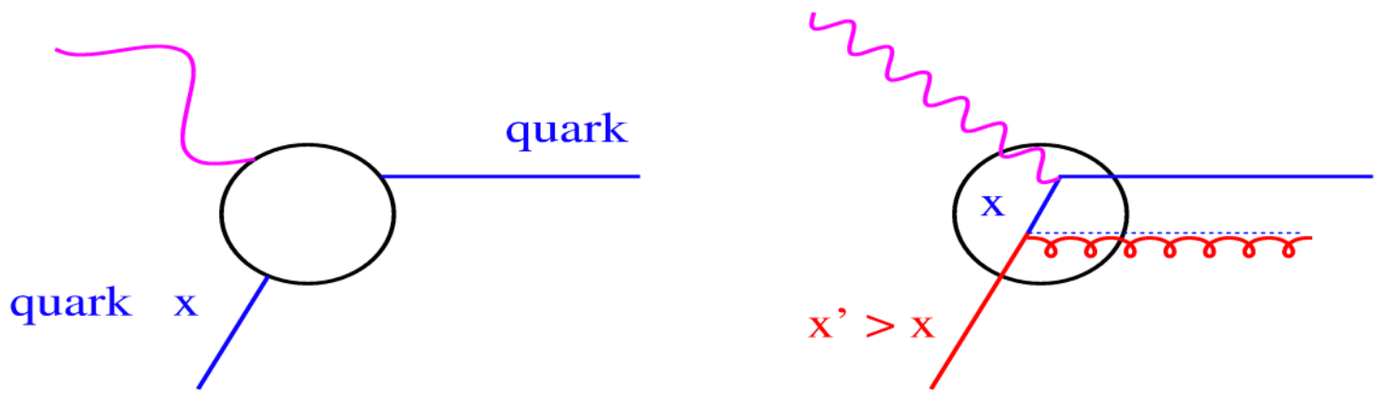


$$P_{gg} = 6\left(\frac{1-z}{z}+\frac{z}{1-z}+z(1-z)\right)$$

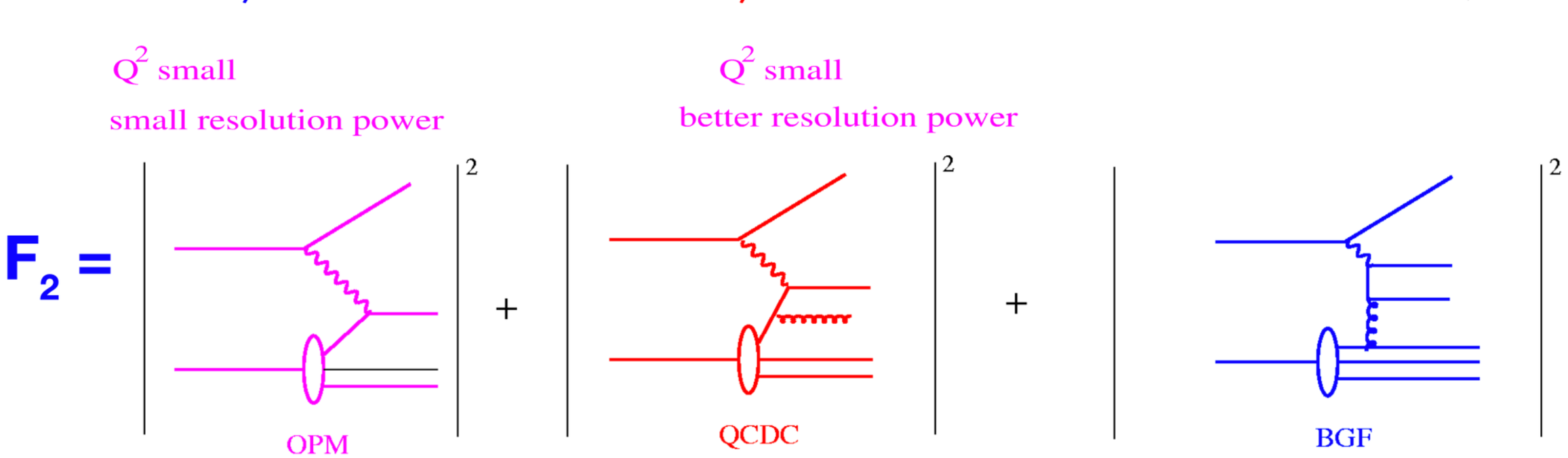


DGLAP evolution equation

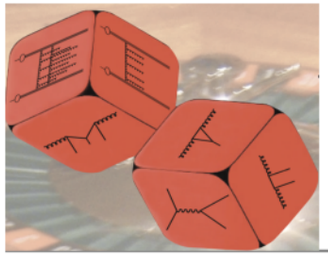
- QPM: F_2 is independent of Q^2
- Q^2 dependence of structure function: **D**okshitzer **G**ribov **L**ipatov **A**ltarelli **P**arisi



➔ Probability to find parton at small x increases with Q^2



➔ **Test of theory: Q^2 evolution of $F_2(x,Q^2)$!!!!!**



Collinear factorization: DGLAP

- introduce new scale $\mu^2 \gg \chi^2$ and include soft, non-perturbative physics into renormalized parton density:

- $$q_i(x,\mu^2) = q_i^0(x) + \frac{\alpha_s}{2\pi}\int_x^1 \frac{d\xi}{\xi}\left[q_i^0(\xi)P_{qq}\left(\frac{x}{\xi}\right) + g^0(\xi)P_{qg}\left(\frac{x}{\xi}\right)\right]\log\left(\frac{\mu^2}{\chi^2}\right)$$

- DokshitzerGribovLipatovAltarelliParisi equation:

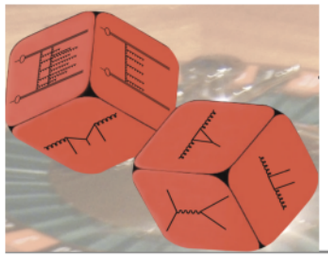
  V.V. Gribov and L.N. Lipatov Sov. J. Nucl. Phys. 438 and 675 (1972) 15, L.N. Lipatov Sov. J. Nucl. Phys 94 (1975) 20,
  G. Altarelli and G. Parisi Nucl.Phys.B 298 (1977) 126, Y.L. Dokshitser Sov. Phys. JETP 641 (1977) 46

$$\frac{dq_i(x,\mu^2)}{d\log\mu^2} = \frac{\alpha_s}{2\pi}\int_x^1 \frac{d\xi}{\xi}\left[q_i(\xi,\mu^2)P_{qq}\left(\frac{x}{\xi}\right) + g(\xi,\mu^2)P_{qg}\left(\frac{x}{\xi}\right)\right]$$

- BUT there are also gluons...

$$\frac{dg(x,\mu^2)}{d\log\mu^2} = \frac{\alpha_s}{2\pi}\int_x^1 \frac{d\xi}{\xi}\left[\sum_i q_i(\xi,\mu^2)P_{gq}\left(\frac{x}{\xi}\right) + g(\xi,\mu^2)P_{gg}\left(\frac{x}{\xi}\right)\right]$$

- DGLAP is the analogue to the beta function for running of the coupling



DGLAP evolution again....

- differential form:

$$t\frac{\partial}{\partial t}f(x,t) = \int \frac{dz}{z}\,\frac{\alpha_s}{2\pi}P_+(z)\,f\left(\frac{x}{z},t\right)$$

- differential form using f/Δ_s with

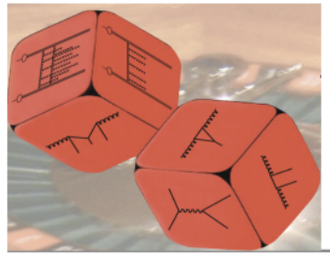
$$\Delta_s(t) = \exp\left(-\int_x^{z_{max}} dz \int_{t_0}^{t} \frac{\alpha_s}{2\pi}\frac{dt'}{t'}\tilde{P}(z)\right)$$

$$t\frac{\partial}{\partial t}\frac{f(x,t)}{\Delta_s(t)} = \int \frac{dz}{z}\,\frac{\alpha_s}{2\pi}\,\frac{\tilde{P}(z)}{\Delta_s(t)}\,f\left(\frac{x}{z},t\right)$$

- integral form

$$f(x,t) = f(x,t_0)\Delta_s(t) + \int \frac{dz}{z}\int \frac{dt'}{t'}\cdot\frac{\Delta_s(t)}{\Delta_s(t')}\tilde{P}(z)f\left(\frac{x}{z},t'\right)$$

no – branching probability from t_0 to t



Evolution equation and Monte Carlo

$$f(x,t) = f(x,t_0)\Delta_s(t) + \int \frac{dz}{z}\int \frac{dt'}{t'}\cdot\frac{\Delta_s(t)}{\Delta_s(t')}\frac{\alpha_s}{2\pi}\tilde{P}(z)f\left(\frac{x}{z},t'\right)$$

$$= f(x,t_0)\Delta_s(t) + \int dz\int \frac{dt'}{t'}\cdot\frac{\Delta_s(t)}{\Delta_s(t')}\frac{\alpha_s}{2\pi}\tilde{P}(z)f\left(x',t'\right)\delta(x-zx')dx'$$

- $f(x',t_0)\Delta_s(t',t_0)$ is probability from previous branching
- use Sudakov:

$$\Delta(t) = \exp\left[-\int_{t_0}^{t}\frac{dt'}{t'}\int^{z_{max}} dz\frac{\alpha_s}{2\pi}\tilde{P}(z)\right]$$

- generate *t* according to Sudakov

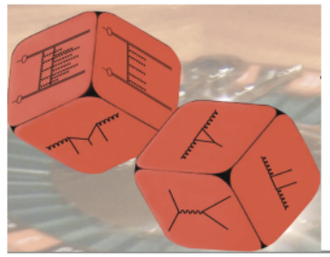
$$\frac{\partial}{\partial t'}\frac{\Delta_s(t)}{\Delta_s(t')} = \frac{\Delta_s(t)}{\Delta_s(t')}\left[\frac{1}{t'}\right]\int^{z_{max}} dz\frac{\alpha_s}{2\pi}\tilde{P}(z)$$

➔solve it for *t:*

$$\log\Delta_s(t,t') = \log R$$

- generate z according to

$$\int_{\epsilon}^{z} dz\frac{\alpha_s}{2\pi}P(z) = R\int_{\epsilon}^{1-\epsilon} dz\frac{\alpha_s}{2\pi}P(z)$$



Evolution equation and MC

$$f(x,t) = f(x,t_0)\Delta_s(t) + \int \frac{dz}{z} \int \frac{dt'}{t'} \cdot \frac{\Delta_s(t)}{\Delta_s(t')} \tilde{P}(z) f\left(\frac{x}{z}, t'\right)$$

- solve integral equation via iteration:

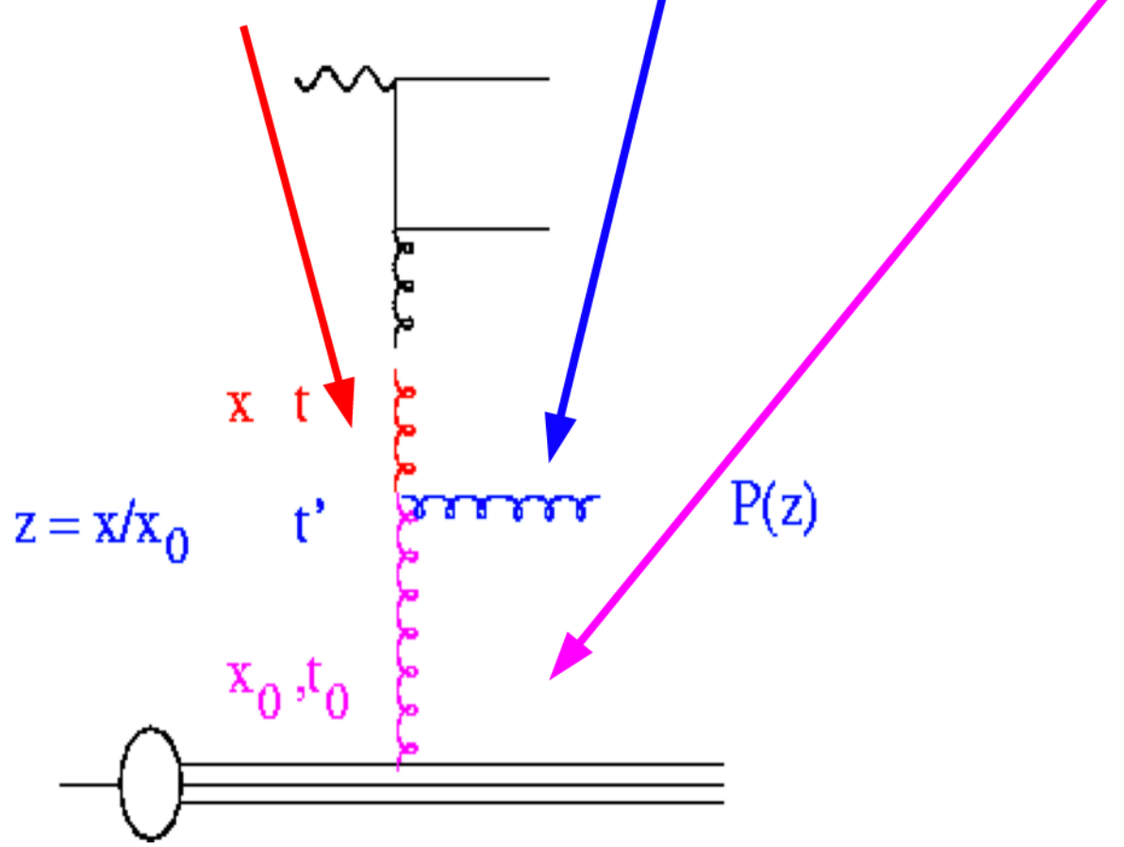
$$f_0(x,t) = f(x,t_0)\Delta(t)$$

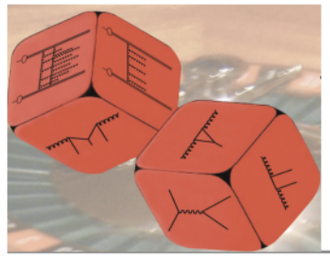
from t' to t w/o branching

$$f_1(x,t) = f(x,t_0)\Delta(t) + \int_{t_0}^{t} \frac{dt'}{t'} \frac{\Delta(t)}{\Delta(t')} \int \frac{dz}{z} \tilde{P}(z) f(x/z, t_0)\Delta(t')$$

branching at t'

from t_0 to t' w/o branching





Evolution equation and MC

$$f(x,t) = f(x,t_0)\Delta_s(t) + \int \frac{dz}{z} \int \frac{dt'}{t'} \cdot \frac{\Delta_s(t)}{\Delta_s(t')} \tilde{P}(z) f\left(\frac{x}{z}, t'\right)$$

- solve integral equation via iteration:

from t' to t w/o branching

branching at t'

from t_0 to t' w/o branching

$$f_0(x,t) = f(x,t_0)\Delta(t)$$

$$f_1(x,t) = f(x,t_0)\Delta(t) + \int_{t_0}^{t} \frac{dt'}{t'} \frac{\Delta(t)}{\Delta(t')} \int \frac{dz}{z} \tilde{P}(z) f(x/z,t_0)\Delta(t')$$

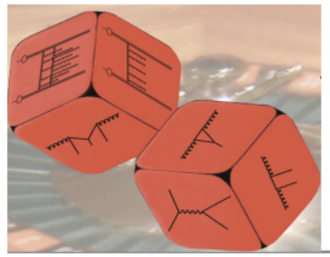
$$= f(x,t_0)\Delta(t) + \log\frac{t}{t_0} A \otimes \Delta(t) f(x/z,t_0)$$

$$f_2(x,t) = f(x,t_0)\Delta(t) + \log\frac{t}{t_0} A \otimes \Delta(t) f(x/z,t_0) +$$

$$\frac{1}{2}\log^2\frac{t}{t_0} A \otimes A \otimes \Delta(t) f(x/z,t_0)$$

$$f(x,t) = \lim_{n\to\infty} f_n(x,t) = \lim_{n\to\infty} \sum_n \frac{1}{n!} \log^n\left(\frac{t}{t_0}\right) A^n \otimes \Delta(t) f(x/z,t_0)$$

summing up all contribution up to t … advantage of importance sampling....

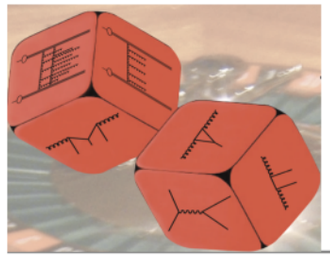


Exercise 2

7. write a program to evolve a parton density $g(x) = 3(1-x)^5/x$ from a starting scale $t_0 = 1$ GeV^2 to and higher scale $t = 100$ GeV^2. Do the evolution only with fixed $\alpha_s = 0.1$ and an approximate gluon splitting function $P_{gg} = 6(\frac{1}{z} + \frac{1}{1-z})$. To avoid the divergent regions use $z_{min} = \epsilon$ and $z_{max} = 1 - \epsilon$ with $\epsilon = 0.1$. Calculate the Sudakov form factor for evolving from t_1 to t_2 using only the $\frac{1}{(1-z)}$ part of the splitting function. Generate z according to P_{gg}. Repeat the branching until you reach the scale t.
   Plot the $xg(x)$ as a function of x for the starting distribution and for the evolved distribution. Repeat the same exercise but with $P_{qq} = \frac{4}{3}\frac{1+z^2}{1-z}$.

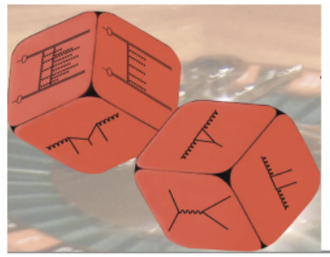
   Calculate and plot the transverse momentum of the parton after the evolution. At the starting scale the partons can have a intrinsic k_t, which is generated by a gauss distribution with $\mu = 0$ and $\sigma = 0.7$ (use generating a gauss distribution from Exercise 1).

   Compare the k_t distribution using P_{gg} and P_{qq}. What is different ?



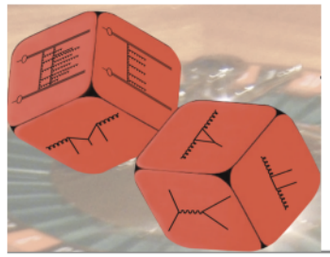
Exercise 2


```
int main(int argc,char **argv)
{
    TH1::SetDefaultSumw2();
    TApplication* gMyRootApp = new TApplication("My ROOT Application", &argc, argv);
    const int npoints = 1e6;

    const double xmin = 0.00001, q2=100*100;

    // initialise random number generator: rlxd_init( luxory level, seed )
    rlxd_init(2,32767);

    // book histograms: TH1D("label","title",nr of bins, xlow,xhigh )
    TH1D *histo1  = new TH1D("x0","x0",100, -5, 0.);
    TH1D *histo2  = new TH1D("kt0 ","kt0 ",100, 0, 10.);
    TH1D *histo3  = new TH1D("x","x",100, -5, 0.);
    TH1D *histo4  = new TH1D("kt ","kt ",1000, 0, 1000.);


    for (int n1 = 0; n1 < npoints; n1++ ) {

        double x , kx, ky, weightx ;
        double x0, kx0, ky0, weightx0 ;

        // generate starting distribution in x and kt
        get_starting_pdf(xmin, q2, weightx0, x0, kx0, ky0);
        // now do the evolution
        evolve_pdf(xmin, q2, x0, kx0, ky0, weightx, x, kx, ky);
        weightx = weightx0 * weightx;

        if (x < 1) {
            // weighting with 1/x0:
            // plot dxg(x)/dlogx *Jacobian, Jacobian dlogx/dx = 1/x
            // log(x) = 2.3026 log10(x)
            double kt0 = sqrt(kx0*kx0 + ky0*ky0);
            double kt  = sqrt(kx*kx + ky*ky);

            histo1->Fill(log10(x0), weightx0/log(10));
            histo2->Fill(kt0,weightx0);
            histo3->Fill(log10(x),  weightx/log(10));
            histo4->Fill(kt, weightx);
        }
    }
//
```



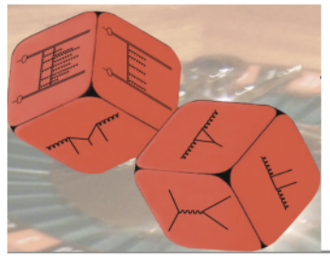
Exercise 2


```
void gauss2D (double sigma, double &kx, double &ky)
{
    double kT  = sigma * sqrt(-2*log(Rand()));
    double phi = 2*M_PI * Rand();
    kx = kT*cos(phi);
    ky = kT*sin(phi);
}

void get_starting_pdf (double xmin, double q2, double& weightx, double& x, double& kx, double& ky)
{
    const double xmax = 1;

    // get x value according to g(x)\sim 1/x
    x = xmin*pow(xmax/xmin, Rand());
    // use: xg(x) = 3 (1-x)**5

    double pdf     = pow(1-x, 5) * 3./x;
    weightx = x*log(xmax/xmin) * pdf ;

    // now generate instrinsic kt according to a gauss distribution
    gauss2D(0.7, kx, ky);
    // cout << "in getpdf:  x " << x << " kx " << kx <<" ky " << ky <<endl;
}

void sudakov (double t0, double& t)
{
```



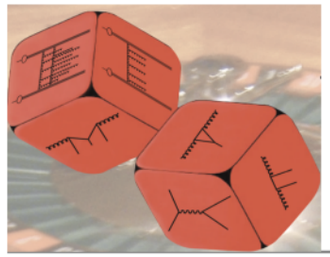
Exercise 2


```
void sudakov (double t0, double& t)
{
//  here we calculate  from the sudakov form factor the next t > t0
    double epsilon = 0.1;
    double as2pi = 0.1/2./M_PI, Ca=3.;
    double Pint;
// for Pgg use fact = 2*Ca

    double fac=2.*Ca;
    double t2;
    // use fixed alphas and only the 1/(1-z) term of the splitting fct

    double r1 = Rand();
    Pint=log((1.-epsilon)/epsilon); /* for 1/(1-z) splitting fct */
    t2 = -log(r1)/fac/as2pi/Pint;
    t2 = t0 * exp(t2);
    t = t2;
    // cout << " t0= "<<t0<< " t= "<<t2<< " epsilon= "<<epsilon <<endl;
    if (t2 < t0) {
        cout << "FATAL t0 > t:  t0= "<<t0<< "t= "<<t2<< "epsilon= "<<epsilon <<endl;
    }
}
```



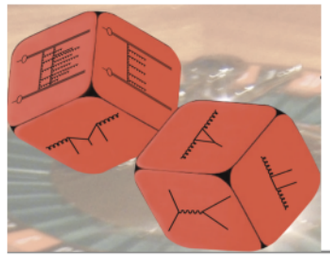
Exercise 2


```
void splitting(double& z, double& weightz)
{
    const double epsilon = 0.1;

    double as2pi = 0.1/2./M_PI;

//          here we calculate  the splitting variable z for 1/z and  1/(1-z)
//          use Pgg=6 (1./z + 1./(1.-z))

    double g0 = 6.*as2pi * log((1.-epsilon)/epsilon);
    double g1 = 6.*as2pi * log((1.-epsilon)/epsilon);
    double gtot = g0 + g1 ;

    double zmin = epsilon;
    double zmax = 1.-epsilon;

    double r1 = Rand();
    double r2 = Rand();

    z = zmin*pow((zmax/zmin),r2);
    if (r1 > g0/gtot)
        z = 1. - z;
//
    weightz = 1.;
}
```

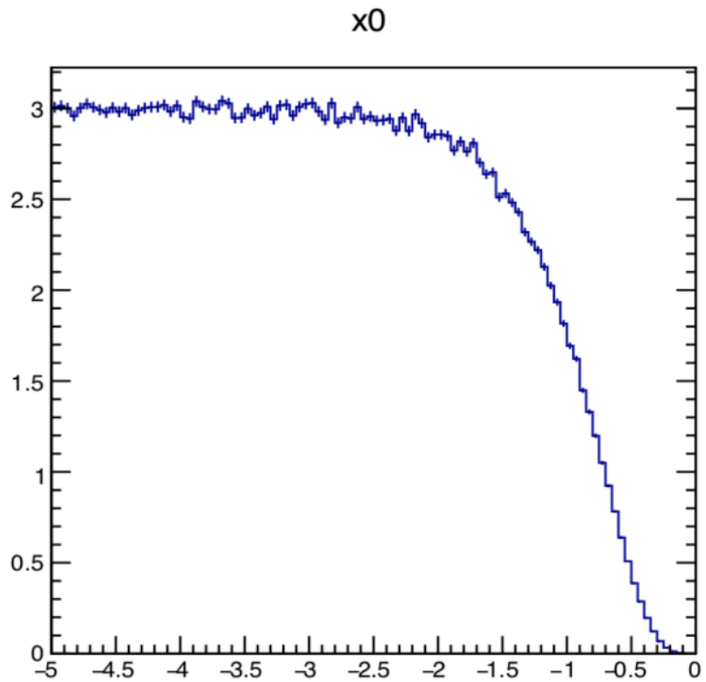


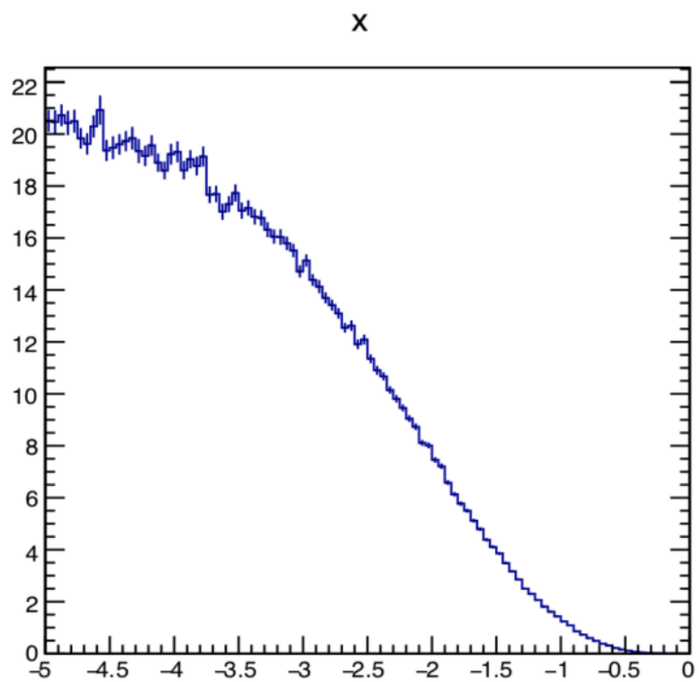
Exercise 2

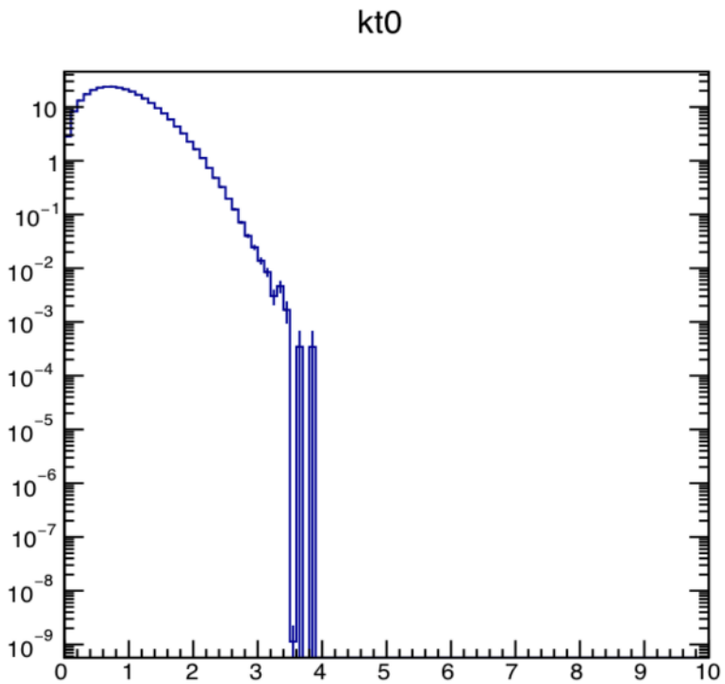

```
void evolve_pdf(double xmin, double q2, double x0, double kx0, double ky0, double& weightx, double& x, double&
kx, double& ky)
{
    double t0=1.,t1, tcut;
    double z=0, weightz;
    double ratio_splitting;
    x = x0;
    kx = kx0;
    ky = ky0;
    weightx = 1. ;
    t1 = t0 ;
    tcut = q2;
    while (t1 < tcut ) {
        // here we do now the evolution
        // first we generate t
        t0 = t1;
        sudakov(t0, t1) ;
        // now we generate z
        splitting(z, weightz);
        /*
           since the sudakov involves only the 1/(1-z) part
           of the splitting fct, we need to weight each branching
           by the ratio of the intgral of the full and
           approximate splitting fct
         */
        ratio_splitting = 2; /* for using Pgg*/

        if ( t1 < tcut ) {
            x = x*z;
            weightx = weightx *weightz*ratio_splitting;
            /*
               use here the angular ordering condition: sqrt(t1) = qt/(1-z)
               and apply this also to the propagator gluons
             */
            double phi = 2*M_PI*Rand();
            kx +=  sqrt(t1)*cos(phi)*(1.-z);
            ky +=  sqrt(t1)*sin(phi)*(1.-z);
            //                       kx = kx + sqrt(t1)*cos(phi);
            //                       ky = ky + sqrt(t1)*sin(phi);
        }
    }
    // cout << " evolve end  t= "<< t<< " q2 " <<q2 << " x0 = " <<x0 << "x =" << x <<endl;
}
```

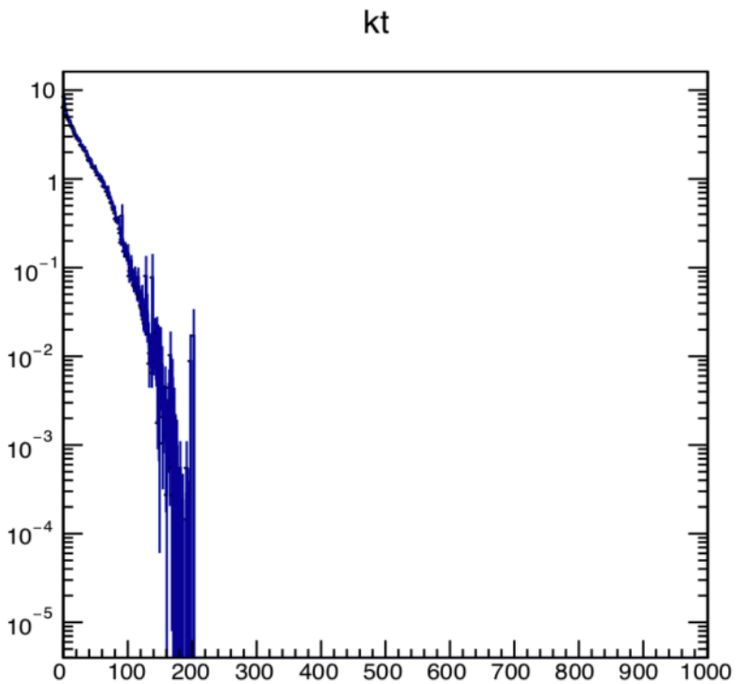


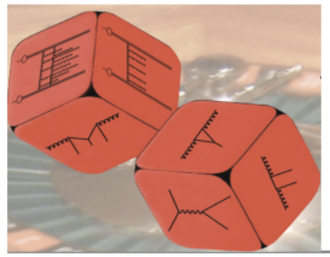
Exercise 2











Why is this relevant ?

Terascale Summer School: Tutorial/Exercises - QCD and Monte Carlo techniques

Cipriano, P., Dooling, S., Grebenyuk, A., Gunnellini, P., Hautmann, F., Jung, H., and Katsas, P. (2013). Higgs boson as a gluon trigger, Phys. Rev., D88(9), 097501

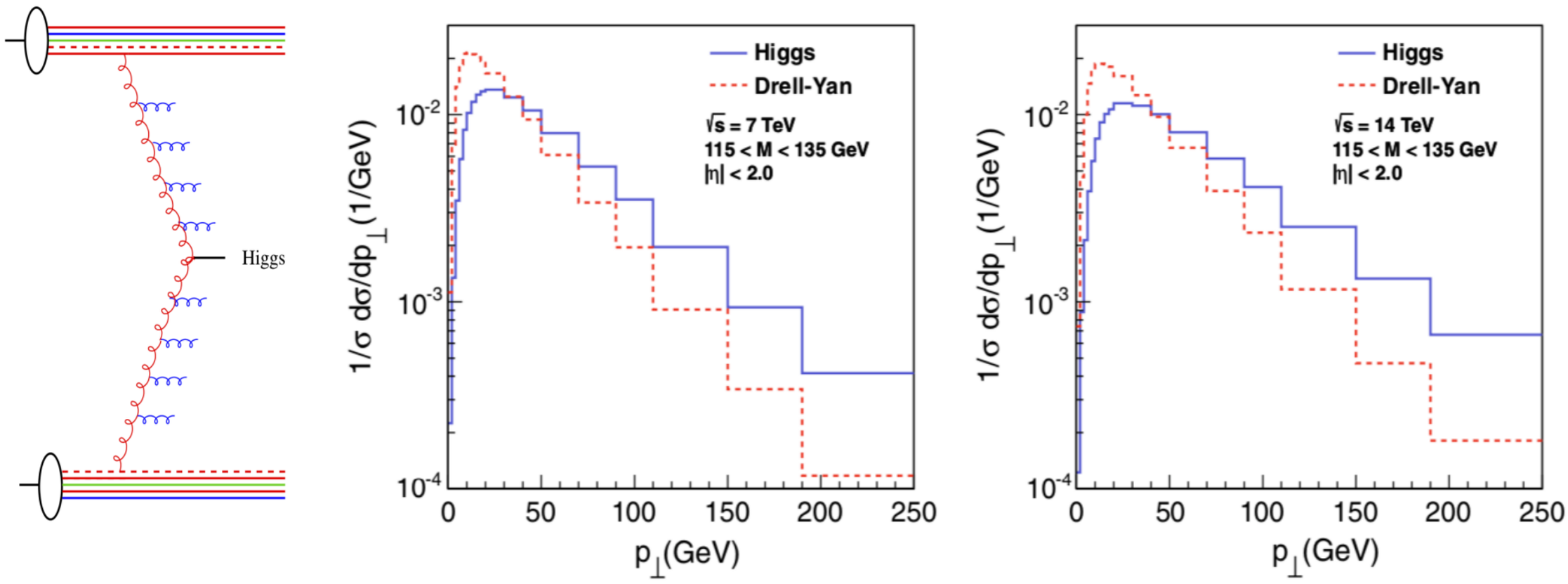
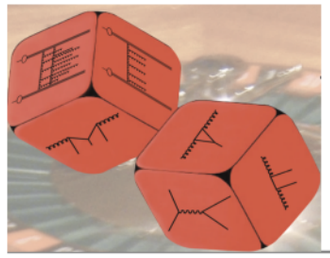
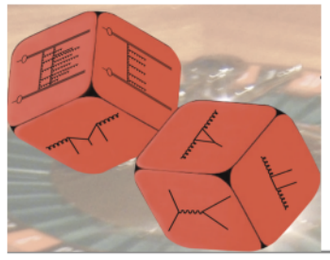


FIG. 1 (color online). Normalized transverse momentum spectra for Higgs bosons and for Drell-Yan pairs.



Breakout Room

- Now we have the tutorial for the second exercises: 30 min
  - work with your tutors in breakout rooms
    - Luis Ignacio: Tutorial with Jupyter
    - Sara: Tutorial with C++
    - Qun: Tutorial with C++



Literature for MC solution of evolution equation

- Monte Carlo and QCD lectures
  - https://www.desy.de/~jung/QCD_and_Monte_Carlo_lectures.html

- R.K. Ellis, W. J. Stirling, and B. R. Webber. QCD and collider physics. Camb. Monogr. Part. Phys. Nucl. Phys. Cosmol., 8:1–435, 1996.
- Bermudez Martinez, A. and others (2020). The transverse momentum spectrum of low mass Drell--Yan production at next-to-leading order in the parton branching method, Eur. Phys. J. C, 80(7), 598
- Bermudez Martinez, A. and others (2019). Production of Z-bosons in the parton branching method, Phys. Rev., D100(7), 074027
- Hautmann, F., Jung, H., Lelek, A., Radescu, V., and Zlebcik, R. (2018). Collinear and TMD quark and gluon densities from Parton Branching solution of QCD evolution equations, JHEP, 01, 070
- Bermudez Martinez, A., Connor, P., Hautmann, F., Jung, H., Lelek, A., Radescu, V., and Zlebcik, R. (2019). Collinear and TMD parton densities from fits to precision DIS measurements in the parton branching method, Phys. Rev., D99(7), 074008
- Hautmann, F., Jung, H., and Monfared, S. T. (2014). The CCFM uPDF evolution uPDFevolv, Eur. Phys. J., C74, 3082