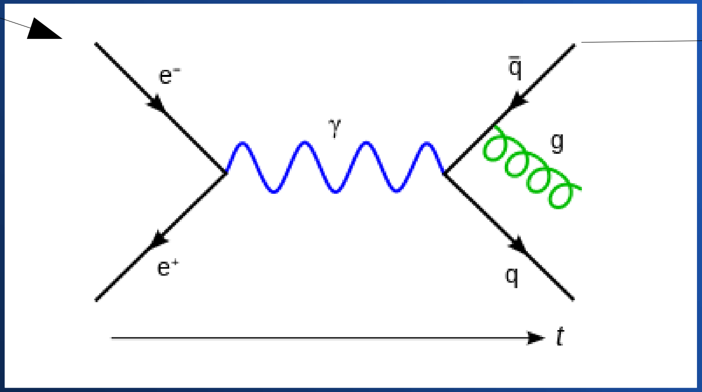
Final state radiation

Arthur B., Michael K.,
Anja B., Ludwig R.

Final state radiation

- Group 2: calculate quark momenta



- Group 4: Apply detector resolution

- Group 1: Integration

How to compute splitting functions

$$\overline{|\mathcal{M}_{n+1}|^2} \equiv \frac{2g_s^2}{p_a^2}\hat{P}_{q\leftarrow g}(z)\,\overline{|\mathcal{M}_n|^2}$$

$$P_{q\leftarrow g}(z) \equiv \hat{P}_{q\leftarrow g}(z) = T_R\left[z^2 + (1-z)^2\right]$$

$$P_{g\leftarrow q}(z) \equiv \hat{P}_{g\leftarrow q}(z) = C_F\frac{1+(1-z)^2}{z}$$

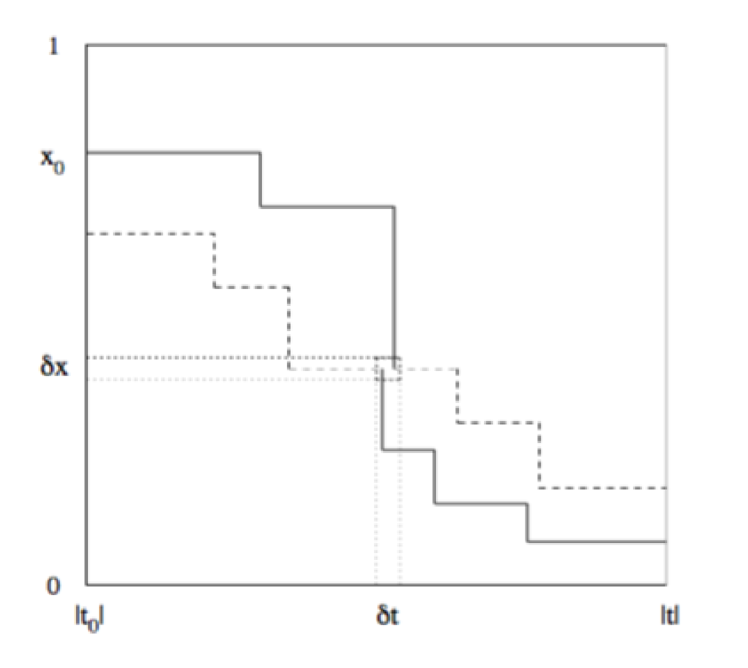
- Probability for final state radiation can be computed by the matrix elements (or splitting kernels)

→ equations for each probability

$$P_{g\leftarrow g}(z) = 2C_A\left(\frac{z}{(1-z)_+} + \frac{1-z}{z} + z(1-z)\right) + \frac{11}{6}\,C_A\,\delta(1-z) - \frac{2}{3}\,n_f\,T_R\,\delta(1-z)$$

Multiple splittings

- Virtuality t = pT^2
- Momentum x
- Order in t



Sudakov factors

- Probability of describing the splitting of a parton i into any of the partons j
- Assuming a Poisson process

→ General equation:

$$\Delta_i(t) \equiv \Delta_i(t, t_0) = \exp\left(-\sum_j \int_{t_0}^{t} \frac{dt'}{t'} \int_0^1 dy\, \frac{\alpha_s}{2\pi} \hat{P}_{j\leftarrow i}(y)\right)$$

→ Example for radiating a quark or gluon:

$$\Delta_q(t) = \exp\left(-\int_{t_0}^{t} dt'\, \Gamma_{q\leftarrow q}(t, t')\right)$$

$$\Delta_g(t) = \exp\left(-\int_{t_0}^{t} dt'\, \left[\Gamma_{g\leftarrow g}(t, t') + \Gamma_{q\leftarrow g}(t')\right]\right)$$

MC procedure

$$\frac{\int_0^{x_2/x_1} dy \frac{\alpha_s}{2\pi} \hat{P}(y)}{\int_0^1 dy \frac{\alpha_s}{2\pi} \hat{P}(y)} = r_x \, \epsilon \, [0,1]$$

$$\frac{\Delta(t_1)}{\Delta(t_2)} = r_t \, \epsilon \, [0,1]$$

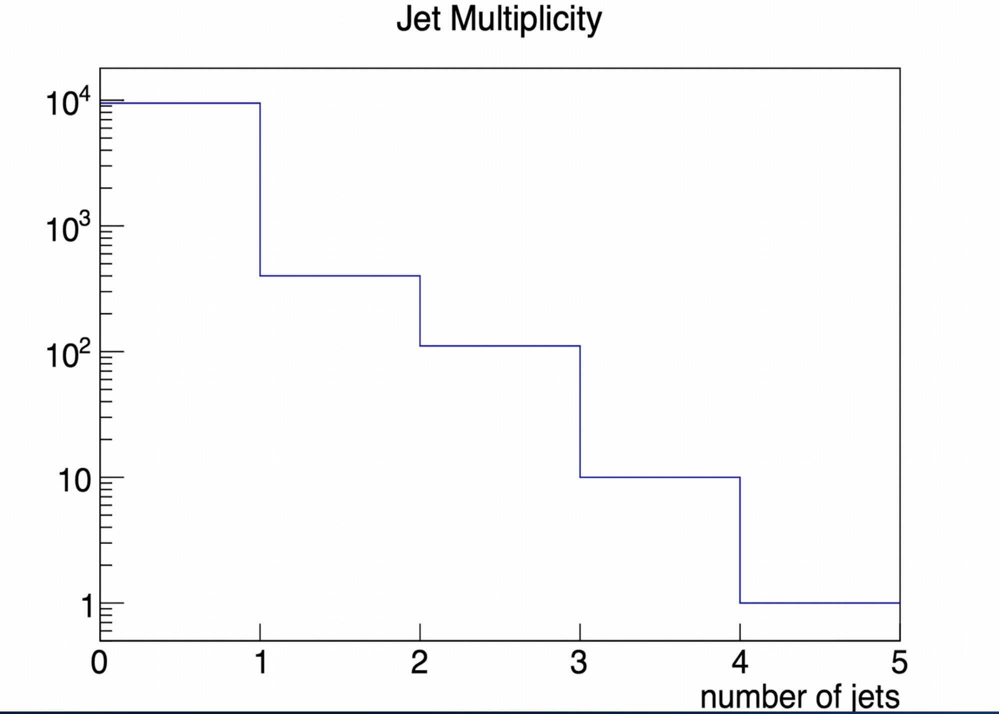
- Generate a r with MC
- Calculate t1 and x1 (initial conditions)
- Solve for t2 and x2
- Repeat until cutoff scale

  → e.g. min p_t in detector
- Do x times to simulate many collisions

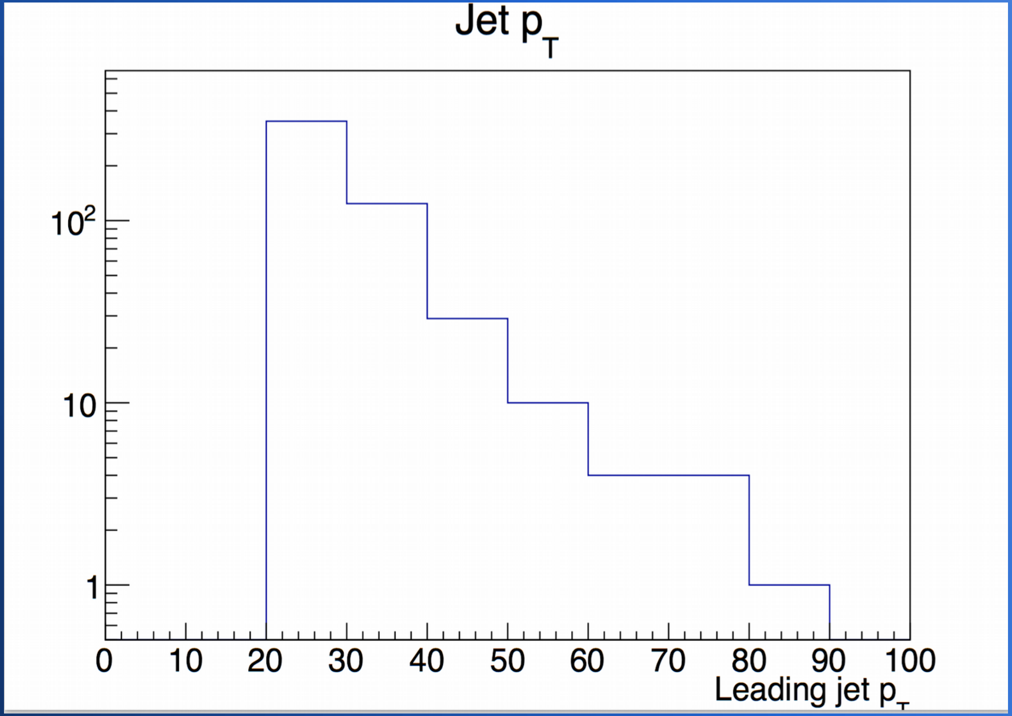
Cutoffs and parameters

- Minimum transversal momentum (= small angles can not be resolved)
- Numerical Cutoff (~1/z, ~1/(1-z))

Pythia



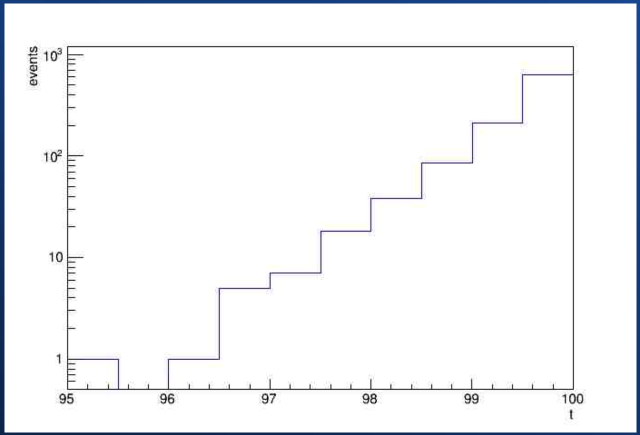
Jet Multiplicity
10^4
10^3
10^2
10
1
0
1
2
3
4
5
number of jets



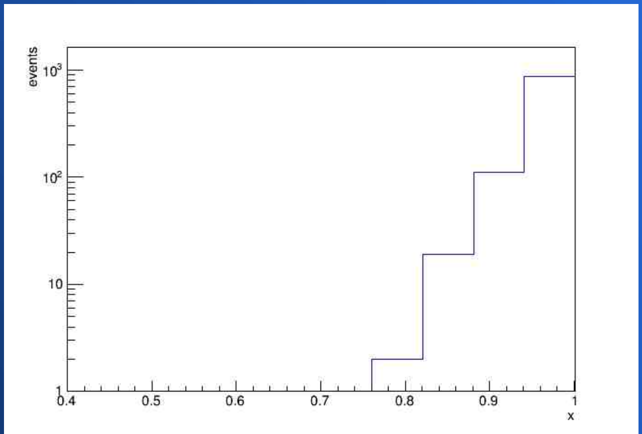
Jet p_T
10^2
10
1
0
10
20
30
40
50
60
70
80
90
100
Leading jet p_T

Our code:

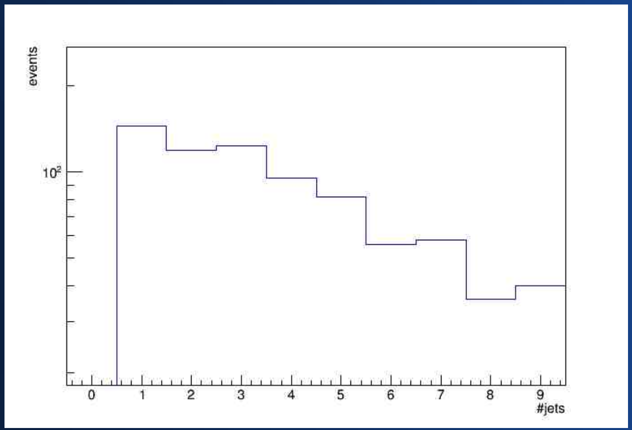
t = pT^2 (quark)



x (quark)



Jet multiplicity



Strongly dependent on non-physical parameters → Finetuning

ToDo:

- Parameter tuning
- Definition of virtuality

- Interface with Matrix-Element-Group: qq (started implementation, finetuning needed)
- Gluon Splitting
- ISR

Code Structure

Particle Interface Class

```
class Particle{

  public:

    // simplest constructor
    Particle(ParticleType type=undefined, double t=0, double x=0);
    // if another group works with e.g. TLorentzVector's we can add
    // something like this and then a getter function.
    Particle(ParticleType type, TLorentzVector v);
    ~Particle() {};

    void SetType(ParticleType type) { m_type = type; }
    void SetT(double t) { m_t = t;}
    void SetX(double x) { m_x = x;}

    ParticleType GetType(void) { return m_type; }
    double GetT(void) { return m_t; }
    double GetX(void) { return m_x; }


  private:

    ParticleType m_type;
    double m_t;
    double m_x;
};
```

```cpp
// final state radiation test class
class FSR{

  public:
    // constructor
    FSR(double t0);
    ~FSR();

    // produce a jets from a particle, call recursively
    void MakeJets(Particle p_in, vector< Particle >& jets);

    ///save event to file
    void save_events(std::string filename, const std::vector<event>& events);
    ///load events from file into vector
    std::vector<event> load_events(std::string filename, int neventsmax = -1);

    //debug
    void DrawTXPlot(char* pdf);
    void DebugPlots(char* pdf, double t_in=100);

  private:
    // check whether a particle can still radiate (eg t>t0
    bool CanRadiate(Particle p);
    bool Radiate(Particle p_in, Particle &p_out1, Particle &p_out2);

    double Delta_gg(double t0, double t1);
    double Delta_qg(double t0, double t1);
    double Delta_qq(double t0, double t1);

    double GetTFromDelta_gg(double t_low, double c);
    double GetTFromDelta_qg(double t_low, double c);
    double GetTFromDelta_qq(double t_low, double c);

    static double P_gg(double z);
    static double P_qg(double z);
    static double P_qq(double z); // static, so that can be used in Integrate();

    double IntP_gg(double z0, double z1);
    double IntP_qg(double z0, double z1);
    double IntP_qq(double z0, double z1);

    double GetXFromP_gg(double x1, double c);
    double GetXFromP_qg(double x1, double c);
    double GetXFromP_qq(double x1, double c);

    double Integrate(double (*func)(double), double z0, double z1);

    TRandom3* m_rand;
```