

**MONTE CARLO METHODS
FOR HIGH ENERGY PHYSICS**

by S. Jadach

Institute of Nucl. Physics, Kraków, Poland

Lecture 2

**Mapping, Rejection, Branching
and Their Combinations**

Slides, program sources: <http://home.cern.ch/jadach>

Numerical examples exploit ROOT package: <http://root.cern.ch/>

Outline:

1. Introduction
2. Rejection
3. Branching
4. Combining Branching and Rejection
5. Mapping
6. Combining Mapping, Branching and Rejection

Do we need to talk about MC methods?

“The elementary MC methods like rejection according to weight, branching (multichannel method) or mapping of variables are so simple and intuitive that it seems to be not worth to write anything on them. On the other hand in the practical MC applications these methods are often combined in such a complicated and baroque way that sometimes one may wonder if the author is really controlling what he is doing, especially if the documentation is incomplete/sparse and we lack commonly accepted terminology or graphical notation for describing MC algorithms.” (S.J. in LANL e-Print Physics/9906056)

Limiting our scope

In order not to get lost in the “rainforest of the MC methodology” we shall:

- define clearly our aims,
- restrict discussion to any combination of the three elementary methods:
mapping, rejection and branching.

Anything beyond this scope will be marked as ‘**digression**’.

Terminology and AIMS

Let us **RESTRICT** ourselves to multi-dim. integrable **positive unnormalised** distributions $\rho(x) = \frac{d^n R}{dx^n}(x)$, $\rho \geq 0$, $R = \int \rho(x) dx^n$. It can easily be turned into probability distribution $p(x) = \frac{1}{R}\rho(x)$, provided we know the integral R . Usually R is not known beforehand.

Our simultaneous AIMS are the following:

1. Generate randomly points/events $x = (x_1, x_2, \dots, x_n)$ exactly according to $p(x) = \rho(x)/R$ (i.e. unweighted events).
2. Calculate the integral $R = \int \rho(x) dx^n$.

We do not need to worry much about point 2.

The value of R comes as a byproduct of point 1. (The opposite not true.)

The above specifies **MC simulator**, which provides integral as well.

Rejection method

The most important method providing unweighted events (MC simulation) according to $\rho(x)$ is rejection of a subset of events generated (simulated) primarily according to an auxiliary distribution $\rho^{(0)}$, more primitive than ρ , easier to simulate/integrate.

Assume that a **MC simulator** for $\rho^{(0)}(x)$ is AVAILABLE. It provides (unweighted) events x and the integral $R^{(0)} = \int \rho^{(0)}(x) dx^n$ (At least at the end of simulation).

Integral R is then obtained from $R = \langle w \rangle_{\rho^{(0)}} R^{(0)}$, where $w(x) = \rho(x)/\rho^{(0)}(x)$ is the MC weight. Subscript in average $\langle \dots \rangle_{\rho^{(0)}}$ tells us that average is over ALL events generated according to $\rho^{(0)}$.

REJECTION can be applied if weight $w = \rho/\rho^{(0)}$ features finite maximum w_{\max} :

1. Generate randomly event $x = (x_1, x_2, \dots, x_n)$ according to $p^{(0)}(x) = \rho^{(0)}(x)/R^{(0)}$
2. Calculate weight $w(x) = \rho(x)/\rho^{(0)}(x)$
3. Generate uniform random number $r \in (0, 1)$
4. If $w < rw_{\max}$ accept event, otherwise reject (trash) it and go back to point 1.

Nothing depends on w_{\max} except acceptance rate, provided w_{\max} is big enough.

Rejection method; Another formulation of the same

AIM: Simulate events according to $\rho(x)$ and calculate integral $R = \int \rho$

ASSUMPTION: There exist $\bar{\rho}^{(0)}(x) \geq \rho(x)$ for which MC simulator is AVAILABLE. It provides (unweighted) events x and integral $\bar{R}^{(0)} = \int \bar{\rho}^{(0)}(x) dx^n$.

METHOD:

1. Generate randomly event x according to $p^{(0)}(x) = \bar{\rho}^{(0)}(x) / \bar{R}^{(0)}$
2. Calculate weight $\bar{w}(x) = \rho(x) / \bar{\rho}^{(0)}(x)$
3. Generate uniform random number $r \in (0, 1)$
4. If $r < \bar{w}$ accept event, otherwise reject (trash) it and go back to point 1.

RESULT: Accepted events are generated according to $\rho(x)$ and integral R is given again by

$R = \langle \bar{w} \rangle_{\bar{\rho}^{(0)}} \bar{R}^{(0)}$ where $\bar{w}(x) = \rho(x) / \bar{\rho}^{(0)}(x)$ is MC weight and average $\langle \dots \rangle_{\bar{\rho}^{(0)}}$ is taken over ALL events (accepted and rejected) generated according to $\bar{\rho}^{(0)}$.

NOTE: In this variant maximum weight w_{\max} is simply incorporated into the normalization of the lower level distribution $\bar{\rho}^{(0)} = w_{\max} \rho^{(0)}$.

Rejection method: Primitive case

In some cases one may (trivially) start from **flat/uniform distribution**:

$$\rho^{(0)}(x) = 1, \quad x \in \Omega, \quad R^{(0)} = \int_{\Omega} 1 \, dx^n = V_{\Omega}$$

Then, assuming $\rho(x) \leq \rho_{\max}$,

we have: $w = \rho(x)$ and $R = \langle w \rangle R^{(0)} = \langle \rho(x) \rangle V_{\Omega}$.

The rejection loop is testing for $\rho(x) < r \rho_{\max}$.

The volume of integration/simulation domain V_{Ω} and maximum ρ_{\max} has to be known **in advance** analytically, or from additional MC exercise (inconvenient).

This method applies only for mildly peaked distributions.

Rejection method: Check of correctness

At point x acceptance probability is $p_{accept} = w(x)/w_{\max}$.

Probability density of accepted events p_{accept} times the original density $\rho^{(0)}(x)$.

$$\frac{dp_{accepted}}{dx^n} = A p_{accept}(x) \rho^{(0)}(x) = A \frac{\rho(x)}{\rho^{(0)}(x)w_{\max}} \rho^{(0)}(x) = const \times \rho(x)$$

Calculating integral R using no. of accepted events (instead of $\langle w \rangle$)

What is total number of accepted events $d^n N_{accept}$ in the volume dx^n ?

We generate $d^n N^{(0)} = N^{(0)} p^{(0)} dx^n$ of primary events and we accept

$d^n N_{accept} = p_{accept} d^n N^{(0)}$ of them. In the entire space we get:

$$N_{accep.} = \int d^n N_{accep.} = N^{(0)} \int dx^n \frac{\rho(x)}{\rho^{(0)}(x)w_{\max}} p^{(0)}(x) = \frac{N^{(0)} R}{w_{\max} R^{(0)}}.$$

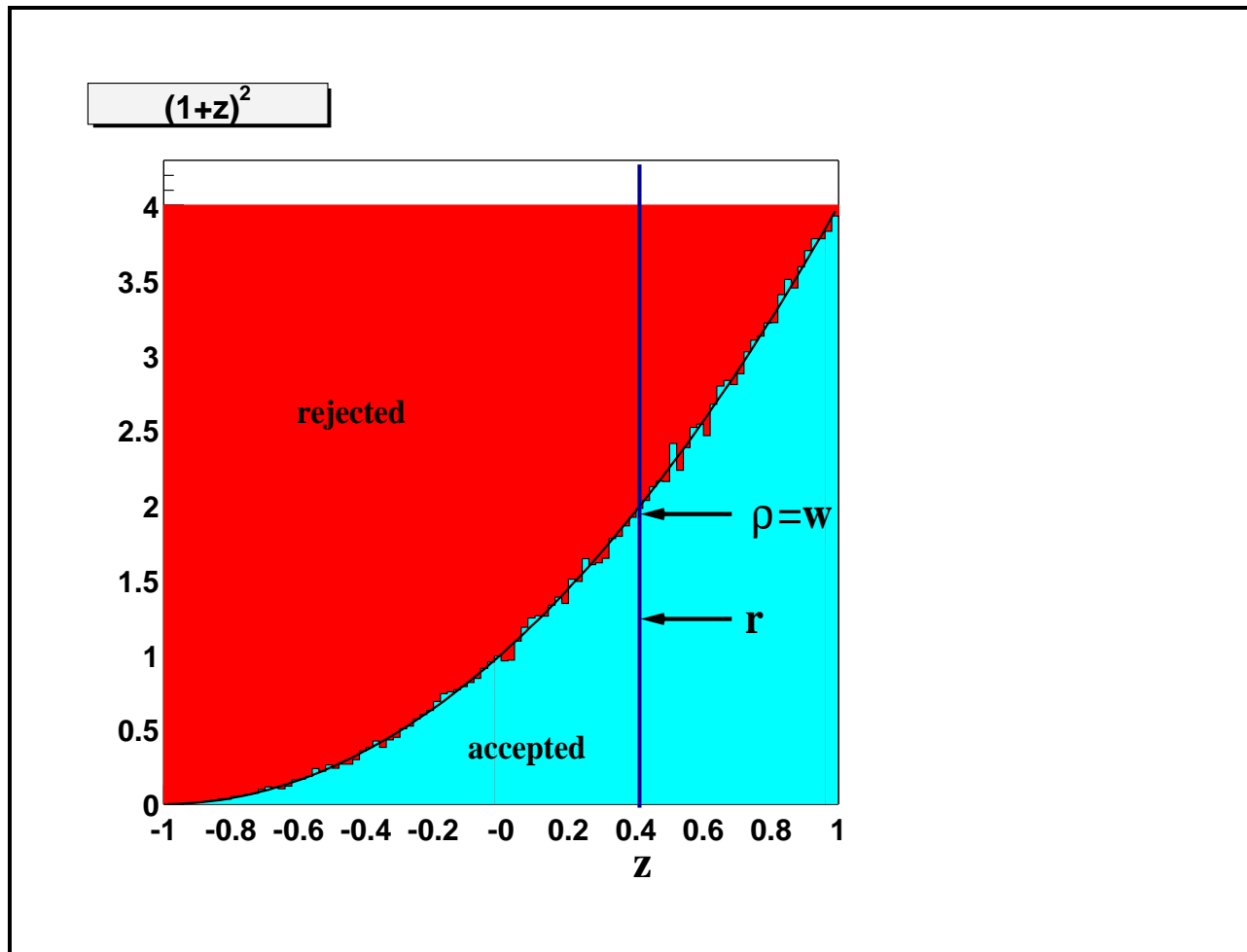
Inverting the above we get $R = w_{\max} R^{(0)} \frac{N_{acc.}}{N^{(0)}} = \bar{R}^{(0)} \frac{N_{acc.}}{N^{(0)}}.$

Contrary to previous case w_{\max} enters explicitly.

The error estimate $\frac{\Delta R}{R} = \sqrt{\frac{1}{N_{acc}} \left(1 - \frac{N_{acc}}{N^{(0)}}\right)}$ comes from the variance of the binomial distribution.

(It is slightly larger than the error from dispersion.)

Rejection method: Simple illustration



Rejection method: Simple illustration

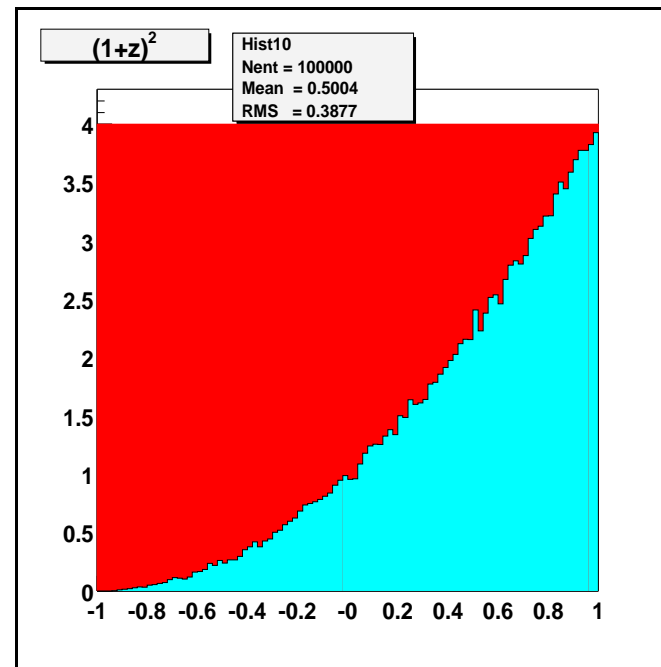
```

class SimuEvent{
// Mini simulator/integrator of (1+z)^2 distribution (rejection method)
private:
    long    m_Nevent;    // No of generated events
    long    m_Naccep;    // No of accepted events
    double  m_SumWt;     // Sum of wt
    double  m_SumWt2;    // Sum of wt^2
    double  m_WtWax;     // Maximum Wt for rejection
    double  m_R0;        // Primary integral = Volume
public:
    SimuEvent(double WtWax){
// constructor
        m_Nevent =0;    m_Naccep =0;
        m_SumWt  =0.0;  m_SumWt2 =0.0;
        m_WtWax  =WtWax; m_R0    =2.0;
    }
    double rho(double z){
// integrand function
        return (1+z)*(1+z);
    }
    void MakeEvent(TRandom *Rngen, double &z){
// generates single event
    RESTART:
        Double_t r1 = Rngen->Rndm(0);
        Double_t r2 = Rngen->Rndm(0);
        z = -1.0+2.0*r1;
        Double_t wt=rho(z);
        m_SumWt  += wt;
        m_SumWt2 += wt*wt;
        m_Nevent++;
        if( r2 > wt/m_WtWax ) goto RESTART;
        m_Naccep++;
    }
    void GetIntegral(double &R, double &delR){
// Provides Integral using average weight
        R    = m_SumWt/m_Nevent *m_R0;
        double sigma2= m_SumWt2/m_Nevent- sqr(m_SumWt/m_Nevent);
        delR = sqrt(sigma2)/sqrt(m_Nevent) *m_R0;
    }
    void GetIntegral2(double &R, double &delR){
// Provides Integral using no. of accepted events
        double p= (1.0*m_Naccep)/m_Nevent;
        R    = m_WtWax*m_R0 *p;
        delR = R *1/sqrt(m_Naccep)*sqrt(1-p);
    }
};

```

Rejection method: Simple illustration

Output distribution (properly normalized):



Output integral:

Generated events: 100000

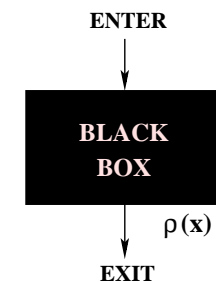
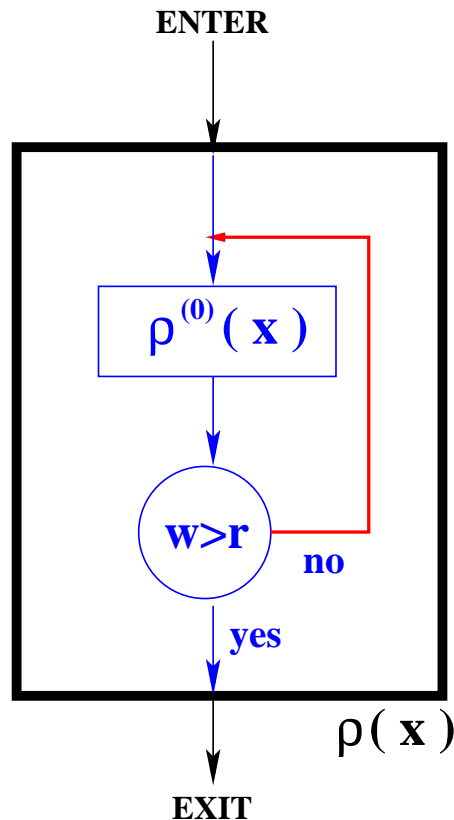
From N_{acc}/N_{tot} : $R, \Delta R = 2.66293 \pm 0.00687806$

From $\text{variance}(wt)$: $R, \Delta R = 2.66168 \pm 0.00435244$

Rejection method: Graphic representation

We need graphic representation in order to visualise: **control flow, information flow and the algorithm structure, ie. its decomposition into smaller simpler elements.**

This graph clearly visualizes control flow in the rejection method. Black rectangle marked $\rho(x)$ underlines the fact that its internal part can be treated as a “black box” part in another bigger MC algorithm.

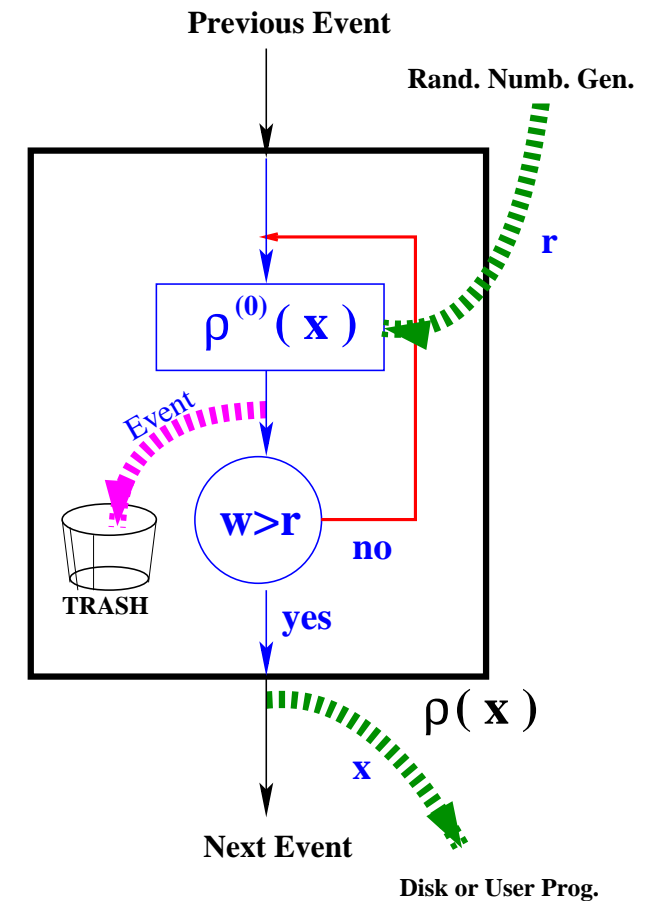


Information flow is not yet properly visualized, see next slide for possible solution.

Rejection method: Visualization

Information flow can be added, if necessary.

Here we stress that in the rejection method part of information is irreversibly lost (rejected events, rn. used for rejection) for the outside world (beyond the black rectangle).

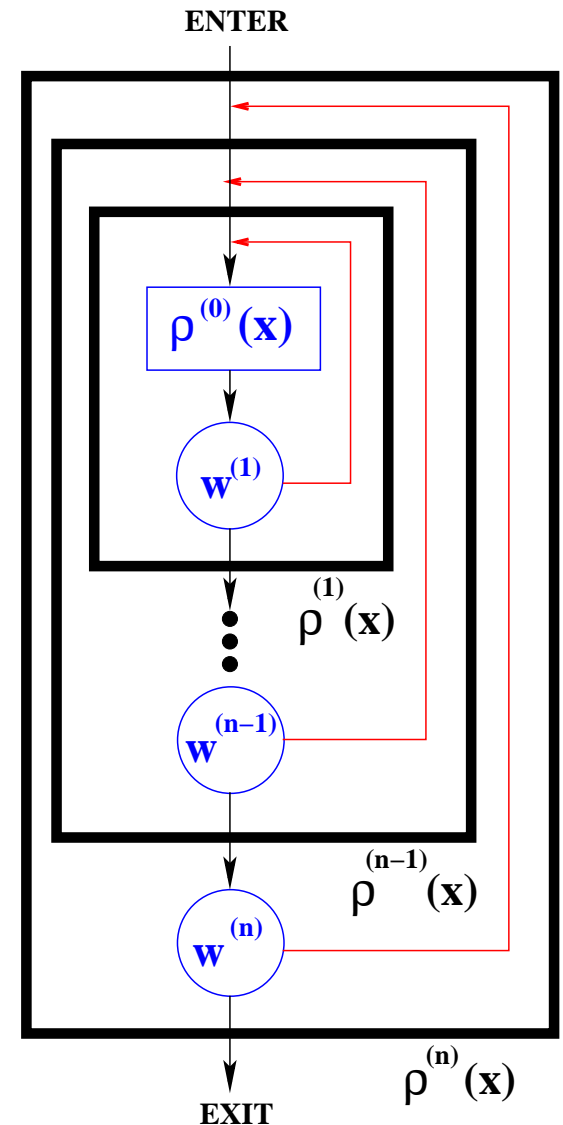


Rejection method: Nesting rejections

Very often rejection loops are nested. Why?

Advantages and disadvantages:

- ⊕ Inner loops may reject more but: unfinished events are cheaper (in CPU time), inner weights calculation is faster.
- ⊕ Outer-loops wt's add "fine details" into distributions; they are CPU time hungry, hence we profit from the fact that they reject little events.
- ⊕ The inner parts (black boxes) form self-contained reusable components of a program library.
- ⊖ Each loop has to have its own mechanism for the weight book-keeping, thus complicated programming.



DIGRESSION: Rejection method as “projection”

Rejection can be regarded as a particular case of another class of “Projection method”, which we shall not discuss in this lecture.

Define random number r of the rejection as a next $(n + 1)$ -th integration variable:

$\tilde{x} = (x_1, x_2, x_3 \dots x_n, r)$ and the new distribution $\tilde{\rho} = \theta(\rho(x) - rw_{\max})$.

The integral is the same $R = \int \rho(x) dx^n = \int \tilde{\rho}(\tilde{x}) dx^{n+1}$, and we proceed to simulate new distribution $\tilde{\rho}$ with any known method, remembering that at the end of the procedure we are going to “trash” x^{n+1} , that is average/integrate over it.

Other (typical) examples of the “projection method”:

- 3-dim. Gaussian: simulate 4-dimen. Gaussian and trash 4-th component,
- $p(x) = \frac{1}{4}x^3$: out of 4 uniform rn. numbers, take the smallest, trash 3 others,
- Uniform distr. of n -dim sphere: simulate n -dim. Gaussian and take $\frac{x}{|x|}$, (trash radius).
and many other useful algorithms...

Opening rejection loops: weighted events

It is always possible to “open rejection loops” and turn simulation into variable-weight event generation.

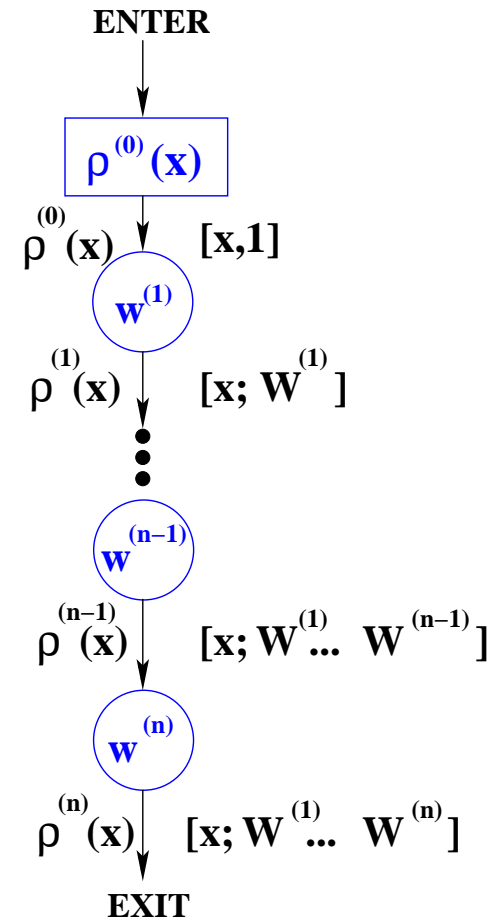
For nested loops the total weight is the product of all intermediate weights is:

$$W^{(n)} = w^{(1)} w^{(2)} w^{(3)} \dots w^{(n)}$$

Variable- w MC calculation can be optionally useful because:

- slightly faster convergence of the integration
- possibility of calculating several variants of the MC integral in a single MC run (also possible for $w=1$ events).
- super-fast evaluation of differences $\langle w - w' \rangle$ - very useful!

So keep always this as an option in the MC simulator!

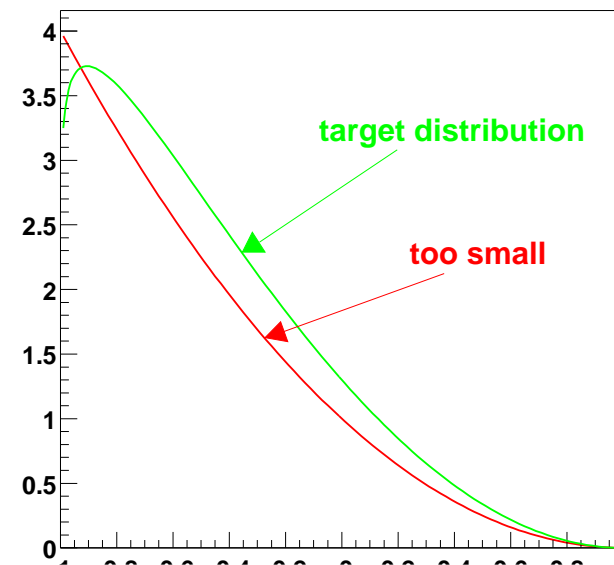


The problem of “maximum weight”

Does it exist?

Green curve is the desired target distribution $\rho(z)$.

Red curve represents candidate for $\bar{\rho}^{(1)}(z)$. It has right shape but it is **too small** to obey $\rho^{(1)} \geq \rho(z)$.

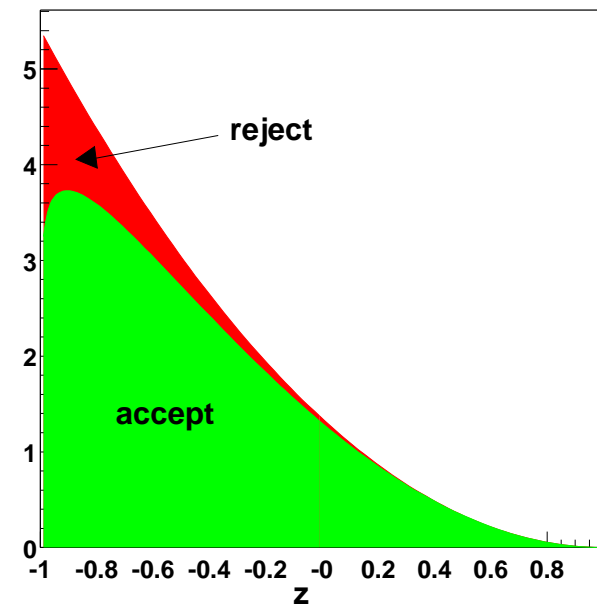


Never mind!

We multiply **red curve** by $\lambda = 1.3$ and we get $\bar{\rho}^{(1)} \geq \rho(z)$, that is $\bar{w} \leq 1$.

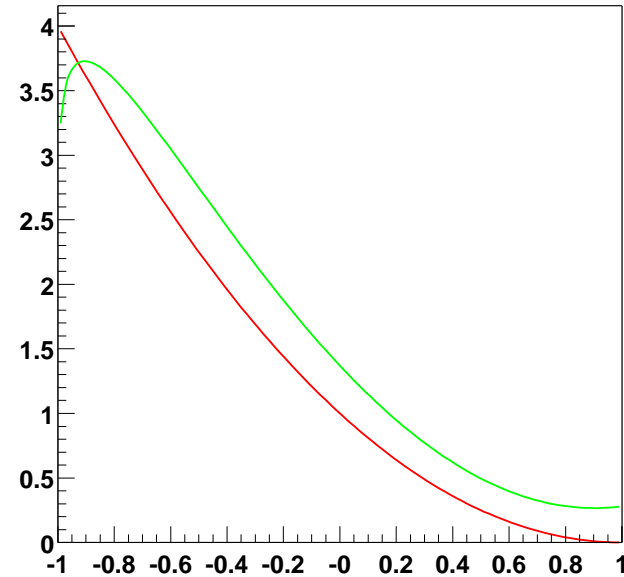
(Alternatively we could set $w_{\max} = 1.3$.)

Is such rescaling always possible?



Limitations of the rejection method

Not always! The candidate for $\bar{\rho}^{(1)}$ cannot have “blind spots”. Here, a candidate for $\bar{\rho}^{(1)}$ has zero at $z = 1$ while target ρ doesn't. Rescaling will never help! Rejection weight will have “infinite tail”, average weight will not exist.



Limitations of the rejection method

Not only zeros, but first of all
narrow spikes are “fatal”
for the rejection method!
Here rejection can be done but
HUGE REJECTION rate



Branching (multichannel) method

ASSUMPTION: $\rho = \sum_{J=1}^N \rho(J; x)$, $\rho^J(x) \geq 0$

and $p_J = \frac{R_J}{R} = \frac{\int \rho(J; x) dx^n}{\int \rho(x) dx^n}$, $\sum_J p_J = 1$.

METHOD: Generate component index J and then x according to single J -th component $\rho(J; x)$ at the time.

Branch Index J can be **trashed** or not. Assume that it is.

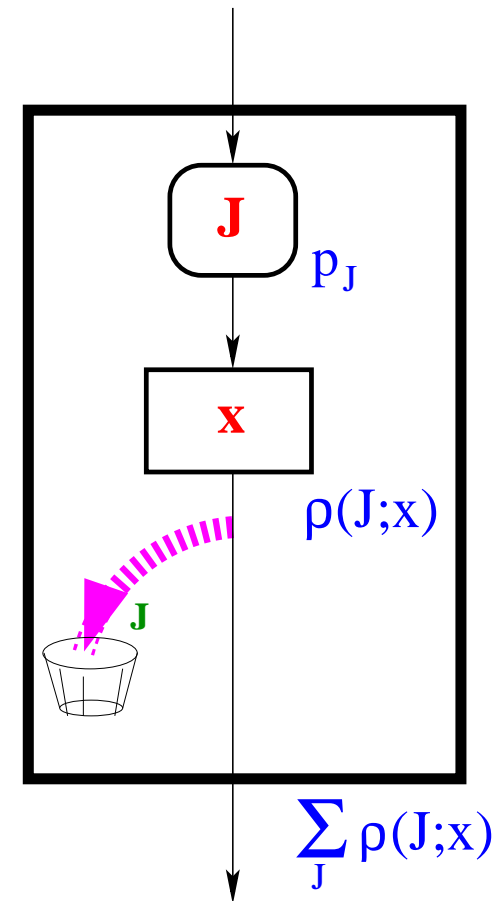
PROOF: Exit events are distributed as follows:

$$d^n p(x) = \sum_J \frac{R_J}{R} \frac{\rho(J; x) dx^n}{R_J} = \frac{\rho(x)}{R} dx^n.$$

PROFIT: Each component $\rho^J(x)$ can be easier to simulate than the sum. Better efficiency, smaller variance etc.

LIMITATIONS: Sub-integrals R_J has to be known in advance!

Way out: Combine with rejection method and/or iterate. See below.



Branching method: Simple example 1

Let us try to simulate:

$$\rho(x) = \frac{1}{x} + \frac{1}{(x-6)^2+1}$$

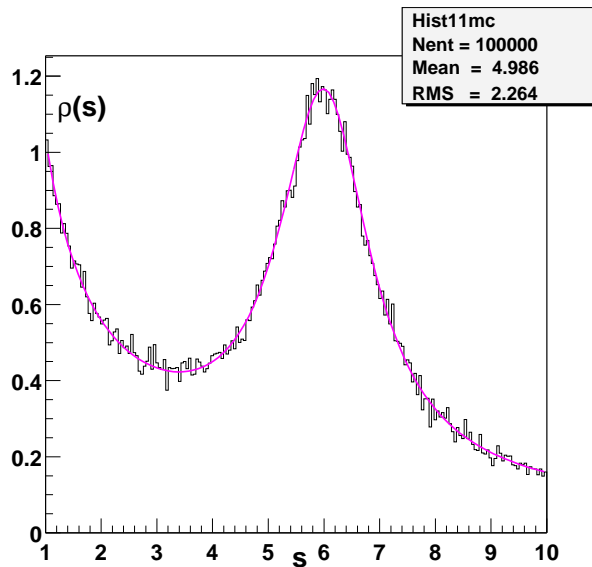
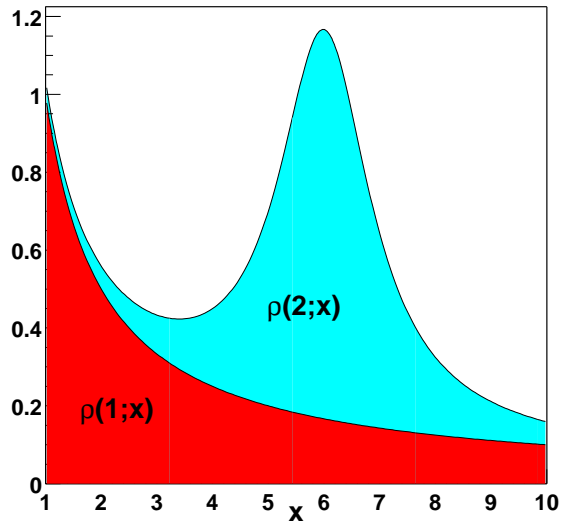
where $x = s \in (1, 10)$.

This is a kind of Breit-Wigner “resonance” (mass= $\sqrt{6}$, width =1) plus non-resonant “background” distribution, added incoherently (no interference).

Each of component distributions is analytically integrable. This example also demonstrates a 1-dimensional “mapping method” (inversion of cumulative).

Please note that 1-dim. mapping involves (analytical) calculation of the component integrals. This is not an accident.

Branching method: Simple example 1



```
// Mini simulator/integrator, 2 branches, Breit-Wigner+Background
private:
    long    m_Nevent;    // No of generated events
    double  m_s1;        // minimum
    double  m_s2;        // maximum
    double  m_R1;        // integral 1-st branch
    double  m_R2;        // integral 2-nd branch
    double  m_gam;       // BW width
    double  m_s0;        // BW center
public:
    SimuEven2(double s1, double s2 ){
    // constructor
        m_Nevent =0;
        m_s1=s1;
        m_s2=s2;
        m_gam=1.0; // BW width
        m_s0 =6.0; // BW center
        m_R1=log(m_s2/m_s1);
        m_R2= 1/m_gam*atan((m_s2-m_s0)/m_gam)
            -1/m_gam*atan((m_s1-m_s0)/m_gam);
    }

    void MakeEvent(TRandom *RNgem, double &s){
    // generates single event, 2 branches
        Double_t A1,A2;
        Double_t r1 = RNgem->Rndm(0);
        Double_t r2 = RNgem->Rndm(0);
        Double_t p=m_R1/(m_R1+m_R2);
        if(r2 < p){ // 1-st branch
            s = m_s1*pow((m_s2/m_s1),r1);
        }else{ // 2-nd branch
            A1 = 1/m_gam*atan((m_s1-m_s0)/m_gam);
            A2 = 1/m_gam*atan((m_s2-m_s0)/m_gam);
            s = m_s0+m_gam*m_gam*tan(A1+(A2-A1)*r1);
        }
        m_Nevent++;
    }

    m_Nevent++;
}

void GetIntegral(double &R){
    // Provides total Integral
    R = m_R1+m_R2;
}
};
```

Mapping, 1-dim case, 2 examples

A limited number of distributions $\rho(x)$ can be generated out of uniform random number r using simple “mapping” $x = H(r)$. Two examples in our small program:

$$\rho(x) = \frac{1}{x}, \quad x \in (x_1, x_2)$$

$$\int_{x_1}^{x_2} \frac{dx}{x} = (\ln x_2 - \ln x_1) \int_0^1 dr, \quad r = \frac{\ln x - \ln x_1}{\ln x_2 - \ln x_1} \in (0, 1),$$

$$x = \exp(\ln x_1 + r(\ln x_2 - \ln x_1)) = x_1 \left(\frac{x_2}{x_1}\right)^r \in (x_1, x_2)$$

$$\rho(x) = \frac{dx}{(x-a)^2 + \gamma^2}, \quad x \in (x_1, x_2)$$

$$\int_{x_1}^{x_2} \frac{1}{(x-a)^2 + \gamma^2} = \frac{\arctan((x_2-a)/\gamma) - \arctan((x_1-a)/\gamma)}{\gamma} \int_0^1 dr,$$

$$r = \frac{\arctan((x-a)/\gamma) - \arctan((x_1-a)/\gamma)}{\arctan((x_2-a)/\gamma) - \arctan((x_1-a)/\gamma)} \in (0, 1),$$

$$x = x_1 + \gamma \tan \left(\arctan \frac{x_1-a}{\gamma} + r \left[\arctan \frac{x_2-a}{\gamma} - \arctan \frac{x_1-a}{\gamma} \right] \right)$$

In general, it is possible if cumulative function $F(y) = \int^y \rho(x) dx$

(a) is known analytically and (b) can be inverted analytically.

Inversion of $F(y)$ can be done numerically rather easily (but rarely done).

Combining Rejection and Branching (A)

Trivial combination rejection + branching.

Rejection applied individually in all branches, or some:

$$w_J(x) = \frac{\rho(J;x)}{\rho^0(J;x)}, \quad p_J = \frac{R_J}{R} = \frac{\langle w_J \rangle}{\sum_L \langle w_L \rangle}$$

PROOF: no need, simple superposition.

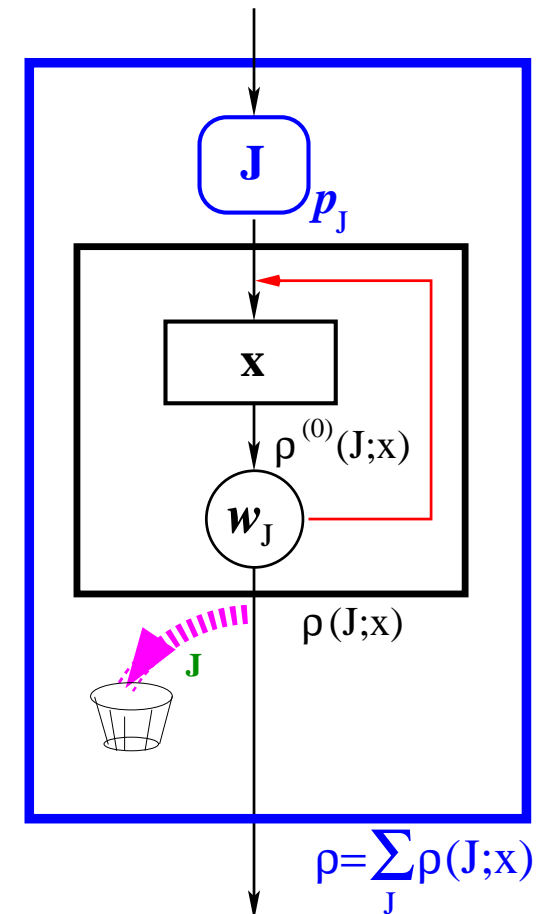
PROBLEM: p_J not known in advance.

R_J known at the end of the MC run - too late!

Because of that problem this arrangement is rarely used.

Also opening (temporarily) rejection loop not easy,
it requires $p_J \rightarrow p_J^{(0)}$.

So why not put J -generation inside rejection loop?



Combining Rejection and Branching (B)

Nontrivial combination of Rejection + Branching.

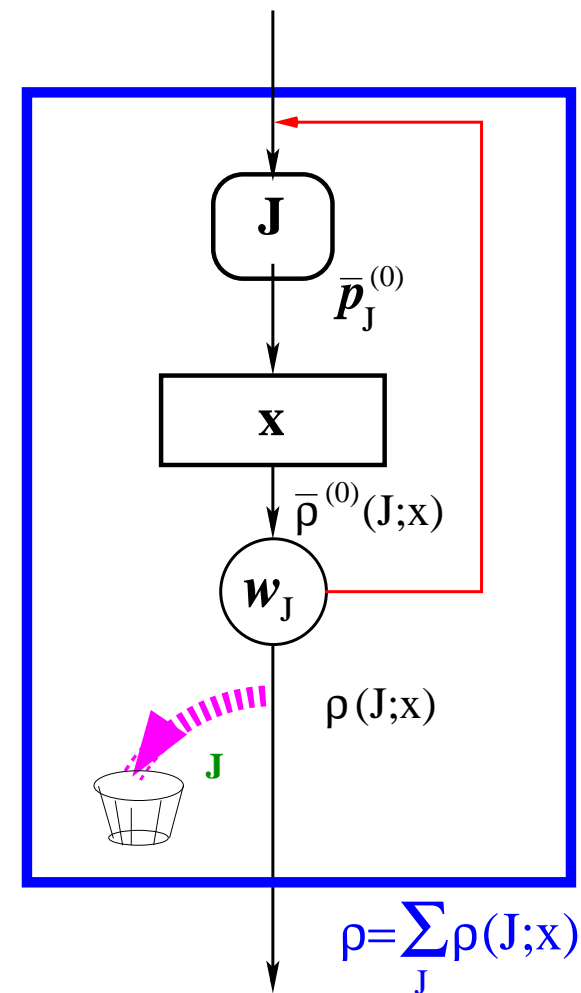
Rejection weight $\bar{w}_J = \frac{\rho(J;x)}{\bar{\rho}^{(0)}(J;x)}$ is for the actual J -th branch and J is subsequently “trashed”.

Probabilities $\bar{p}_J^{(0)} = \frac{R_J^{(0)}}{\bar{R}^{(0)}}$ are for simplified $\bar{\rho}^{(0)}$'s, hence possibly known in advance (analytically).

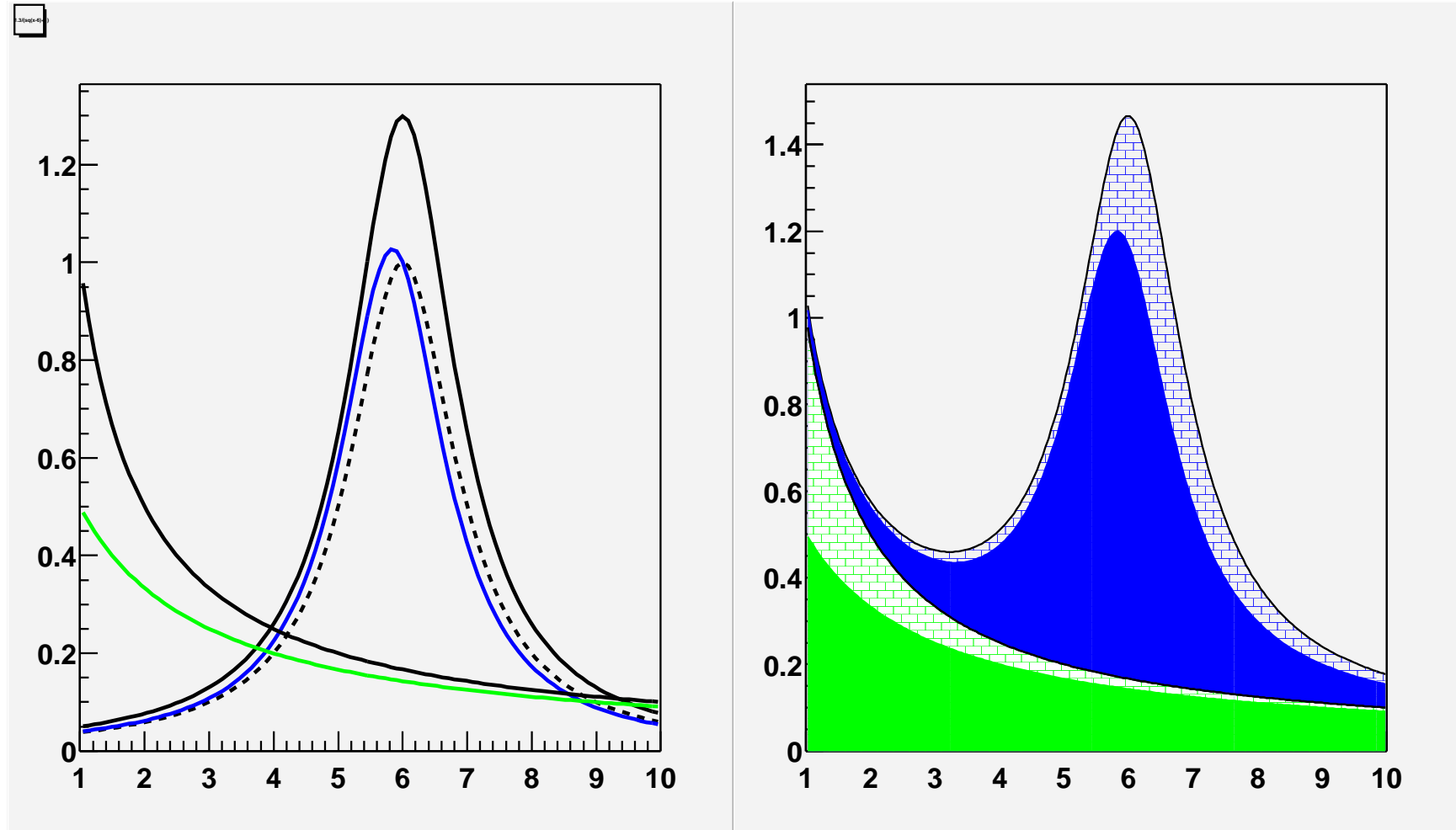
Accounting properly for normalisation requires using “bared” probabilities and distributions (including w_{\max}).

PROOF: Probability density $d^n p(x)$ at point x at the exit of the algorithm (graph) is proportional to product of probability density for the the J -th branch $d^n p_J^{(0)}(x) = \bar{\rho}^{(0)}(J;x) dx^n / \bar{R}_J^{(0)}$ times probability of accepting event $\bar{w}_J(x) = \rho(J;x) / \bar{\rho}^{(0)}(J;x)$, averaged over all branches with probabilities \bar{p}_J :

$$d^n p(x) = A \sum_J \bar{p}_J d^n p_J^{(0)}(x) \bar{w}_J(x) = A \sum_J \frac{\bar{R}_J^{(0)}}{\bar{R}^{(0)}} \frac{\bar{\rho}^{(0)}(J;x) dx^n}{\bar{R}_J^{(0)}} \frac{\rho(J;x)}{\bar{\rho}^{(0)}(J;x)} = A \frac{\rho(x) dx^n}{\bar{R}^{(0)}}$$



Combining Rejection and Branching (B) cont.



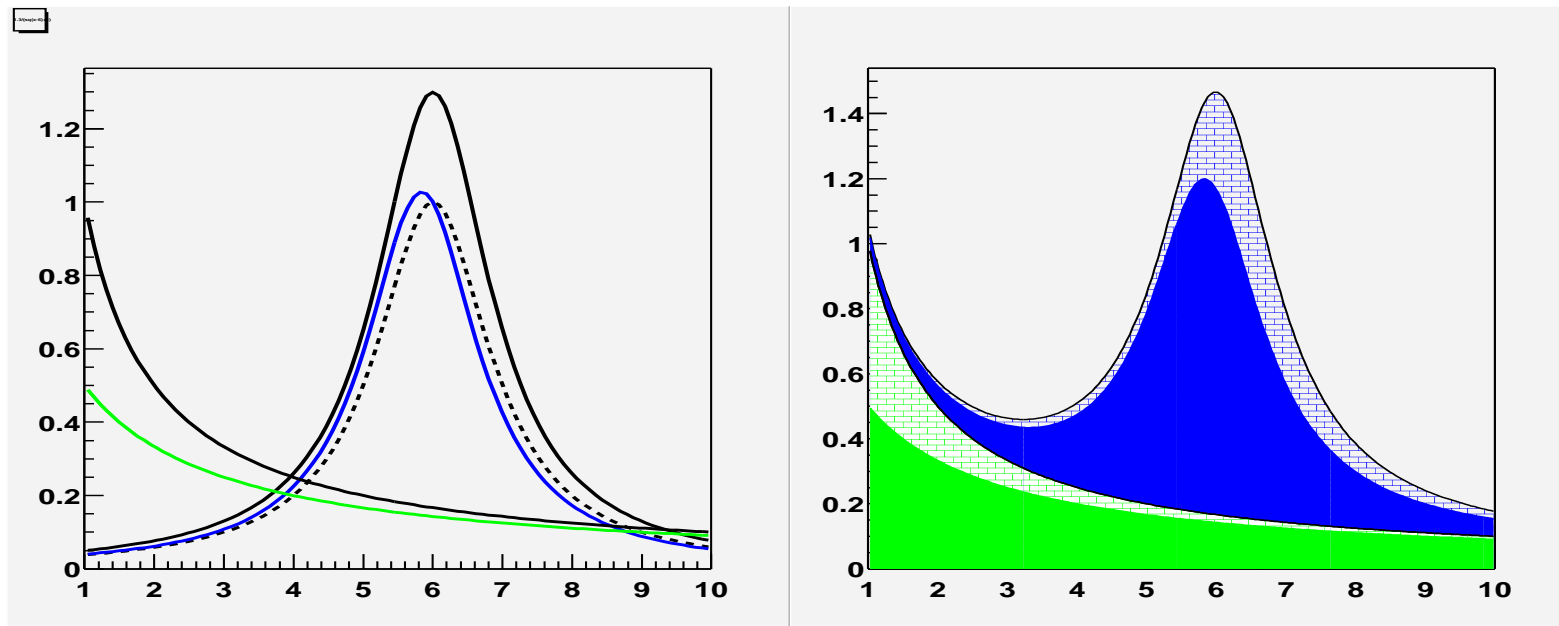
Two-branch example: “Brick-walled” part is rejected. “Solid-color” part accepted.

Example of method (B), details of distributions

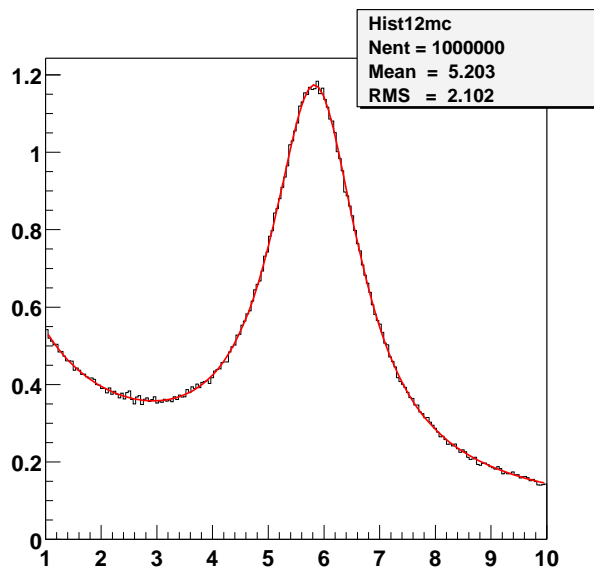
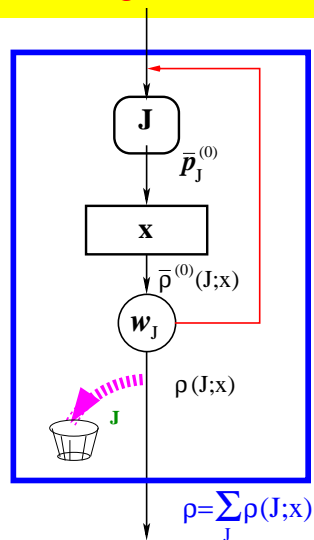
Two distributions in the branches are “background” $B(s) = \left| \frac{-1}{(1+s)^{1/2}} \right|^2 = \frac{1}{1+s}$ and “resonance” $R(s) = \left| \frac{1}{(s-6)+i(s/6)} \right|^2 = \frac{1}{(s-6)^2+(s/6)^2}$. added incoherently.

Resonance has “variable width” and we simplify it to standard Breit-Wigner form $R(s) \rightarrow R'(s) = \frac{1}{(s-6)^2+1}$ which can be generated with help of mapping. The corresponding compensating weight is $R(s)/R'(s)$.

The simplification $\frac{1}{1+s} \rightarrow \frac{1}{s}$ (corrected by the MC weight) in the background distribution is not really necessary, we do it for the illustration of the method.



Branching method: Simple example 2



```

class SimuEven3{ // Simulator/integrator, BreitWigner+background
private:
long    m_Nevent;           // No of generated events
double  m_SumWt,m_SumWt2;  // Sum of wt and wt^2
double  m_s1,m_s2;         // minimum, maximum s
double  m_R1,m_R2;         // integral \bar{R}^{(0)}_J, simplified
double  m_gam;             // BW width
double  m_s0;              // BW center
public:
SimuEven3(double s1, double s2 ){
// constructor
m_Nevent =0;
m_s1=s1; m_s2=s2;
m_gam=1.0; // BW width
m_s0 =6.0; // BW center
m_R1 = log(m_s2/m_s1);
m_R2 = 1/m_gam*atan((m_s2-m_s0)/m_gam)
      -1/m_gam*atan((m_s1-m_s0)/m_gam);
m_R2 = m_R2*1.3; // AMPLIFICATION !!!
cout<<"R1,R2= " <<m_R1<<" " <<m_R2<<endl;
}
double Weight(double s, int J){
if(J==1)
return (1/(s+1))/(1/s);
else
return ( 1.0/(sqr(s-m_s0)+m_gam*sqr(s/m_s0)))
      /(1.3/(sqr(s-m_s0)+m_gam));
}
void MakeEvent(TRandom *RNggen, double &s){
// generates single event, 2 branches
RESTART:
Double_t wt,A1,A2;
Double_t r1 = RNggen->Rndm(0);
Double_t r2 = RNggen->Rndm(0);
Double_t p=m_R1/(m_R1+m_R2);
if(r2 < p){ // 1-st branch
s = m_s1*pow((m_s2/m_s1),r1);
wt= Weight(s,1);
}else{ // 2-nd branch
A1 = 1/m_gam*atan((m_s1-m_s0)/m_gam);
A2 = 1/m_gam*atan((m_s2-m_s0)/m_gam);
s = m_s0+m_gam*m_gam*tan(A1+(A2-A1)*r1);
wt= Weight(s,2);
}
m_Nevent++;
m_SumWt += wt; m_SumWt2 += wt*wt;
Double_t r3 = RNggen->Rndm(0);
if( r3 > wt ) goto RESTART;
}
}

```

Combining Rejection and Branching (B) cont.

NORMALIZATION:

Total integral is a sum over integrals from all branches

$$R = \sum_J R_J \text{ where } R_J = R_J^{(0)} \langle w_J \rangle = \bar{R}_J^{(0)} \langle \bar{w}_J \rangle.$$

This yields:
$$R = \bar{R}^{(0)} \sum_J \bar{p}_k \langle \bar{w}_J \rangle = \bar{R}^{(0)} \langle \langle \bar{w} \rangle \rangle,$$

where average $\langle \langle \bar{w} \rangle \rangle$ is also over all branches.

Formula for the total integral based on the number of

$$\text{accepted events } \sigma = \bar{R}^{(0)} \frac{N}{N^{(0)}} = \bar{R}^{(0)} \frac{\sum_J N_J}{\sum_J N_J^{(0)}}$$

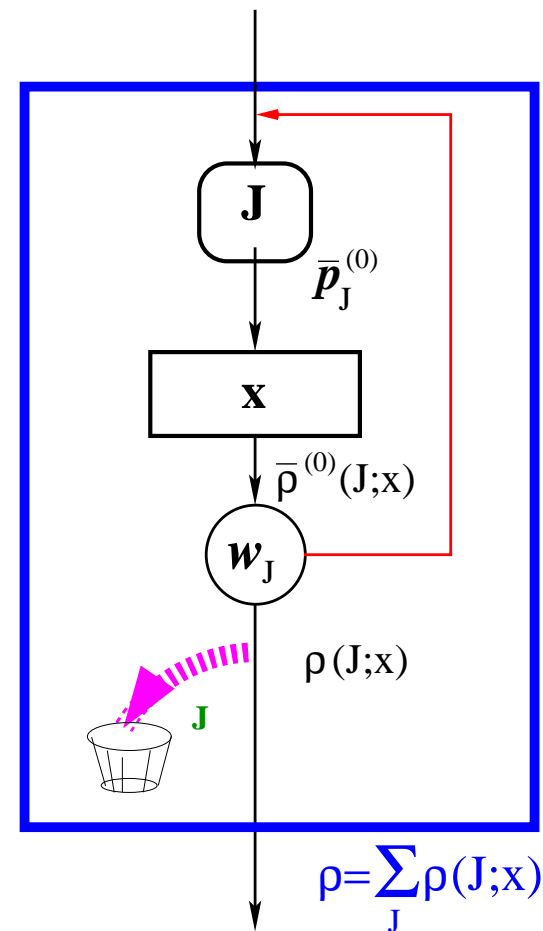
is easily derivable.

```
void GetIntegral(double &R, double &delR){
// Provides Integral using average weight
R = m_SumWt/m_Nevent *(m_R1+m_R2);
double sigma2= m_SumWt2/m_Nevent- sqr(m_SumWt/m_Nevent);
delR = sqrt(sigma2)/sqrt(m_Nevent) *(m_R1+m_R2);
}
```

Rejection loop, contrary to case (A), can be here opened immediately!

Quite remarkably the user of events outside blue-box does not need to know J !!!

He needs to know numerical value of the weight only.



Combining Rejection and Branching (C)

Third possible arrangement of rejection + branching.

The weight of the outer rejection loop does not (need to) know **the actual J** of a given event:

$$w = \frac{\rho(x)}{\sum_I \bar{\rho}^{(0)}(I;x)}$$

Contrary to previous case (B) weight includes sum over all branches. Again $R = \bar{R}^{(0)} \langle w \rangle$.

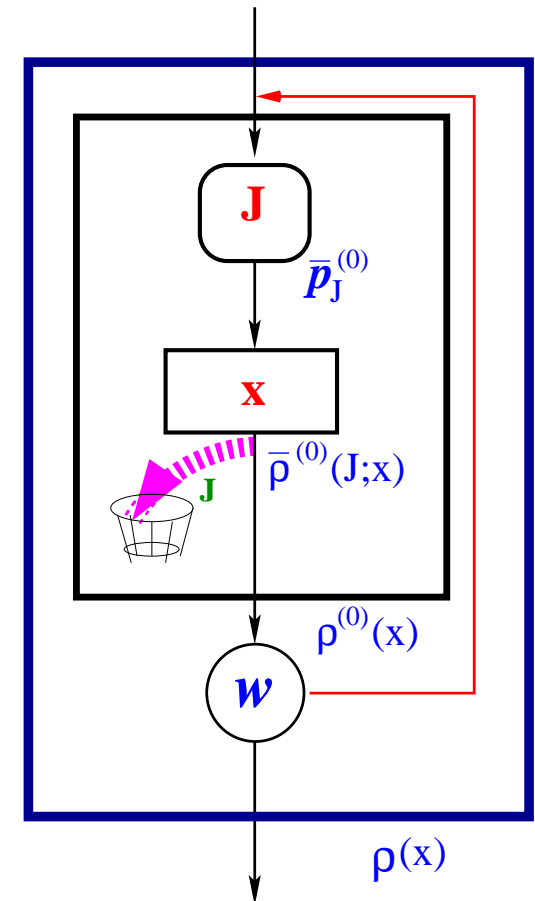
Disadvantage: sometimes, for large number of branches evaluation of the weights can be slow and complicated.

PROOF: No need, simple superposition of rejection and branching.

If rejection loop is opened and we deal with wt-ed events, then in this case we may adjust relative normalization of $\bar{\rho}^{(0)}(I;x)$

iteratively using powerful recipe of Kleiss and Pittau (Comput.

Phys. Commun. 83, 141-146, 1994).



Rejection + Branching, "Real Life example"

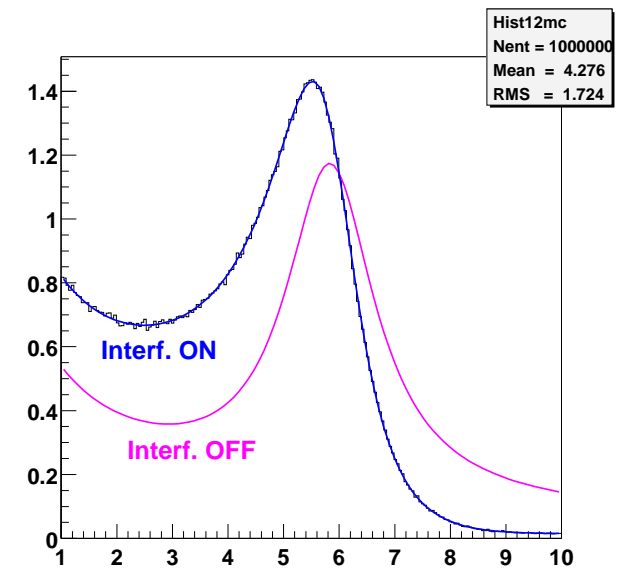
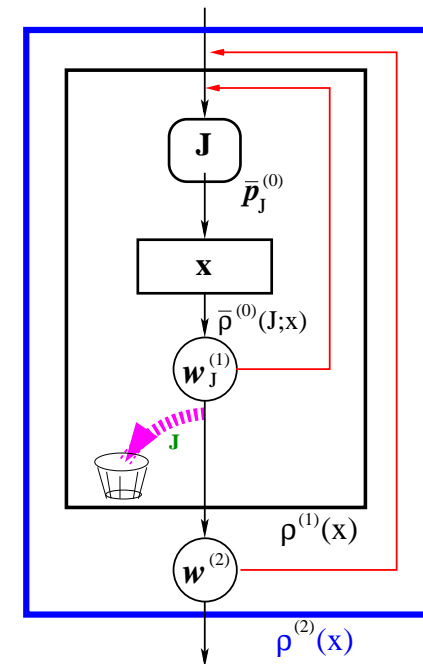
On top of branch-specific effects there might be effects and corresponding weights, which **cannot be attributed to any specific branch**.

They are entered with extra overall weight.

Quantum-mech. interferences are good examples.

NB. internal rejection loop can be opened and one gets single weight $w = w_J^{(1)} w^{(2)}$.

See next slide for another incarnation of our small simulator for resonance+background+interference.



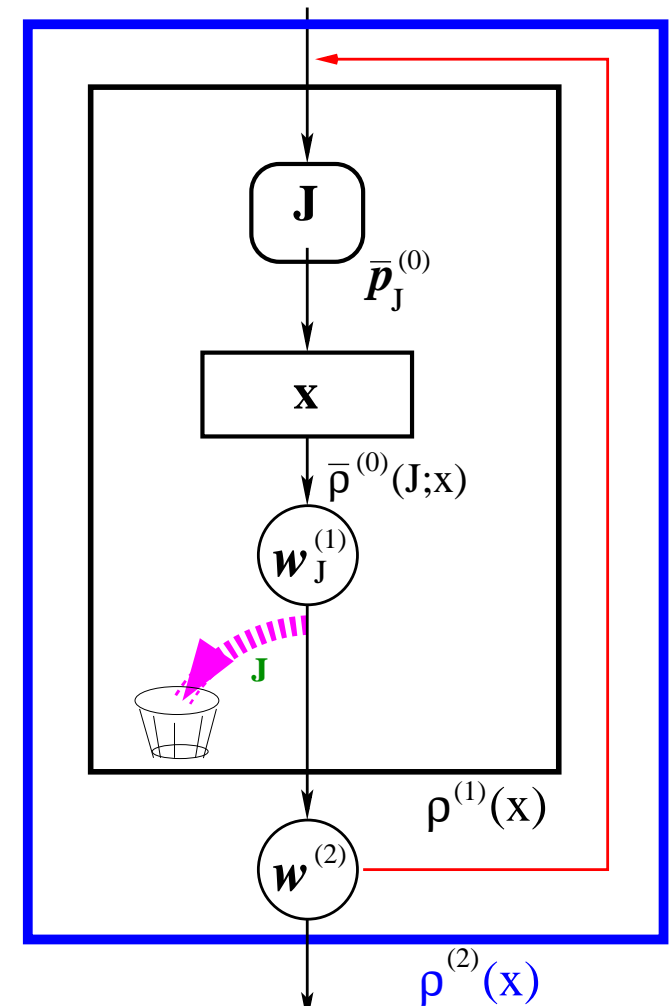
```

.....
double Weight1(double s, int J){
  // basic weight for each branch
  if(J==1)
    return (1/(s+1))/(1/s);
  else
    return ( 1.0/(sqr(s-m_s0)+sqr(m_gam*s/m_s0))
            /(1.3/(sqr(s-m_s0)+sqr(m_gam))));
}

double Weight2(double s){
  double_complex B =-1.0/double_complex(sqrt(1+s),0.0);
  double_complex R = 1.0/double_complex(s-m_s0, m_gam*s/m_s0);
  return sqr(abs(B+R))/(sqr(abs(B))+sqr(abs(R)));
}

void MakeEvent(TRandom *RNggen, double &s){
  // generates single event, 2 branches
  RESTART:
  Double_t wt,A1,A2;
  Double_t r1 = RNggen->Rndm(0);
  Double_t r2 = RNggen->Rndm(0);
  Double_t p=m_R1/(m_R1+m_R2);
  if(r2 < p){ // 1-st branch
    s = m_s1*pow((m_s2/m_s1),r1);
    wt= Weight1(s,1);
  }else{ // 2-nd branch
    A1 = 1/m_gam*atan((m_s1-m_s0)/m_gam);
    A2 = 1/m_gam*atan((m_s2-m_s0)/m_gam);
    s = m_s0+m_gam*m_gam*tan(A1+(A2-A1)*r1);
    wt= Weight1(s,2);
  }
  wt *= Weight2(s);
  m_Nevent++;
  m_SumWt += wt; m_SumWt2 += wt*wt;
  Double_t r3 = RNggen->Rndm(0);
  if( r3 > wt/2.0 ) goto RESTART;
}
.....

```



$$F(s) = \frac{-1}{(1+s)^{1/2}} + \frac{1}{(s-6)+i(s/6)}, \text{ Two branches modelled using } B = \left| \frac{-1}{(1+s)^{1/2}} \right|^2 \text{ and}$$

$$R = \left| \frac{1}{(s-6)+i(s/6)} \right|^2. \text{ Interf. added later with } w^{(2)} = \frac{|F|^2}{B+R} \leq 2.$$

Multidimensional and 1-dim. mapping

Example of 2-dimensional mapping:

$$\int dx dy \rho(x, y) = \int dx dy \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} =$$

$$\int_0^1 d\frac{\phi}{2\pi} \int_0^\infty \frac{1}{2\sigma^2} e^{-r^2/(2\sigma^2)} d r^2 = \int_0^1 d\frac{\phi}{2\pi} \int_1^0 d e^{-r^2/(2\sigma^2)} =$$

$$\int_0^1 dr_1 \int_0^1 dr_2 1.$$

The MAPPING $(r_1, r_2) \rightarrow (x, y)$, where $r_i \in (0, 1)$ are uniform r.n.'s is:

$$x(r_1, r_2) = (-2\sigma^2 \ln r_2)^{1/2} \cos(2\pi r_1),$$

$$y(r_1, r_2) = (-2\sigma^2 \ln r_2)^{1/2} \sin(2\pi r_1).$$

and the **Jacobian** of the mapping transformation

$$\frac{\partial(x, y)}{\partial(r_1, r_2)} = 2\pi\sigma^2 e^{+(x^2+y^2)/(2\sigma^2)} = \frac{1}{\rho(x, y)}$$

cancels exactly the distribution!

DREAM:

$$\int dx^n \rho(x) = \int_{(0,1)^n} dr^n \left| \frac{\partial(x)}{\partial(r)} \right| \rho(x) = \int_{(0,1)^n} dr^n 1, \quad x_i = x_i(r_1, \dots, r_n).$$

If there was a general numerically fast method for finding such a mapping (into Riemann unit hypercube) then this lecture should be removed from the program of the school!

Mapping

Even in 1-dim case there is only a finite number of the distributions which can be generated with help of mapping and elementary functions:-

$$\frac{1}{x}, \quad \frac{\ln^n(x)}{x}, \quad x^a |_{a \neq -1}, \quad e^{ax}, \quad \frac{1}{a^2+x^2}, \quad \frac{1}{a^2-x^2}, \quad \frac{1}{(a^2-x^2)^{1/2}}, \quad \cos(x), \quad \dots$$

Quite nasty-looking distributions can be generated by mapping:

$$\rho(x) = \frac{(1-e^{-x})^{\alpha-1} e^{-x}}{1-(1-e^{-x})^\alpha}, \quad x = -\ln(1 + (1 - e^{-r})^{1/\alpha}).$$

Quite simply-looking distributions cannot be obtained by analytical mapping:

$$e^{-ax^2}, \quad e^{-\alpha x} x^{\beta-1}, \quad c_0 + c_1 x + c_2 x^2 + c_3 x^3, \quad 1 + \sqrt{x}.$$

It is always possible to generate 1-dim distribution by brut force (memorizing, parametrizing distribution numerically), however one should know and use analytical mapping, because it is fast and allows to change parameters in the distribution “in flight”.

For e^{-x^2} parametrization of the inverse of cumulative in fact available: see Abramowitz & Stegun, eq. 26.2.22

How to get closer to distributions treatable by mapping method?

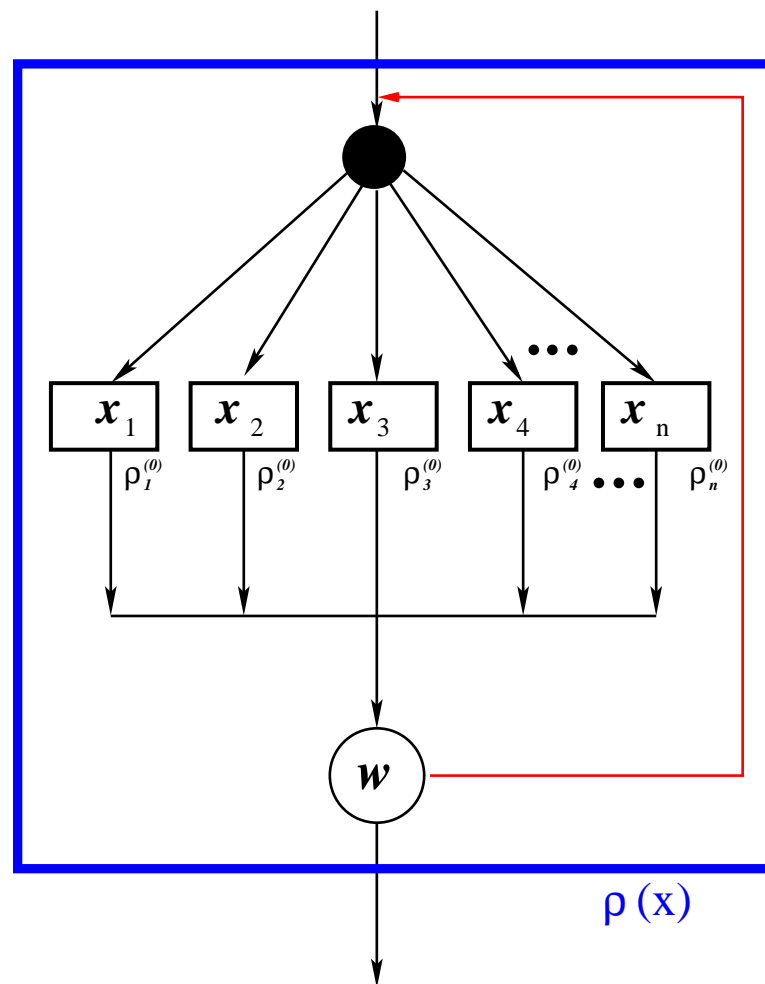
The basic method is to find simpler ρ which, for instance, factorizes into product of independent functions, $\rho^{(0)}(x) = \prod_{i=1}^n \rho_i^{(0)}(x_i)$, each of them “mappable” using elementary functions. Remember that integration limits have to be also independent!

The simplification $\rho(x) \rightarrow \rho^{(0)}(x)$ is “countered” by correcting weight $w = \frac{\rho(x)}{\rho^{(0)}(x)}$.

NOTE1:

Our graph underlines parallel processing possibilities!

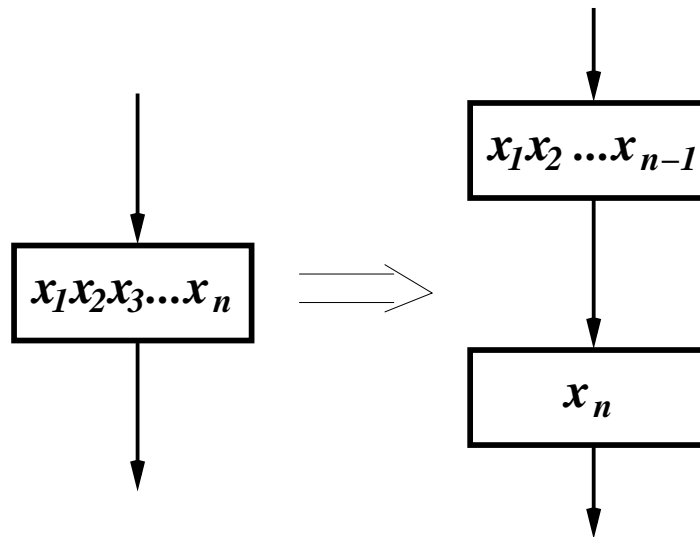
NOTE2: Vegas does mapping numerically (sort of!).



Sequential mapping

With a bit of luck we can simplify $\rho \rightarrow \rho'$ (compensating with the rejection) into new ρ' for which we are able to do mapping for x_n , keeping $x_1 \dots x_{n-1}$ as constant parameters:

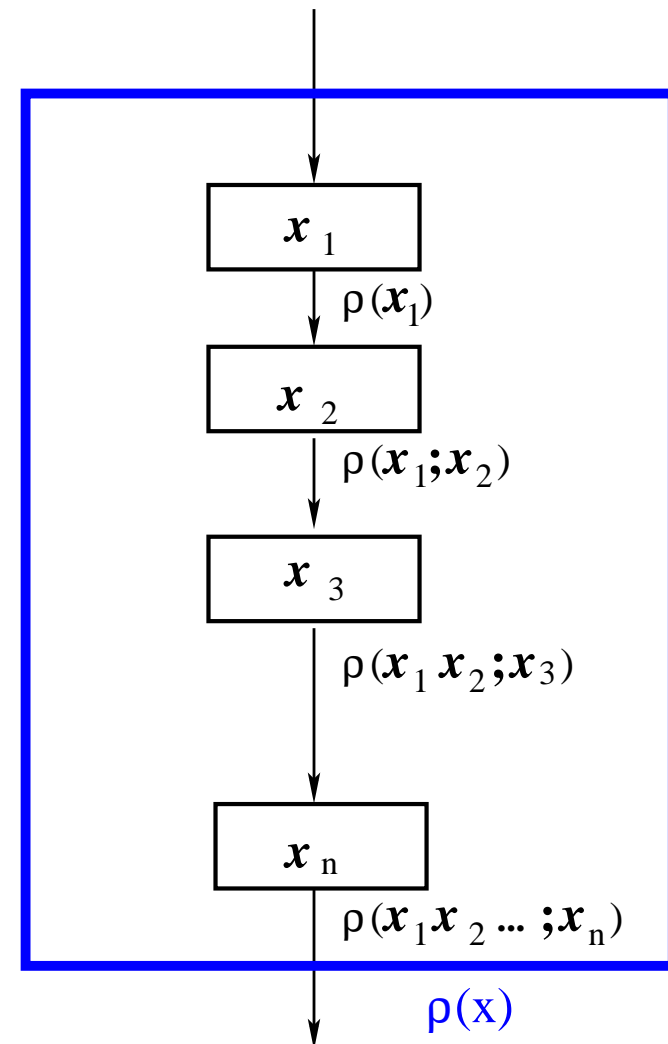
$\int dx_1 \dots dx_{n-1} \int dx_n \rho'(x_1 \dots x_{n-1}; x_n)$, i.e. the integration over x_n can be done analytically, and its cumulative can be inverted.



Hopefully, this procedure can be repeated n times, leading to... see next slide.

Sequential mapping (or for other methods)

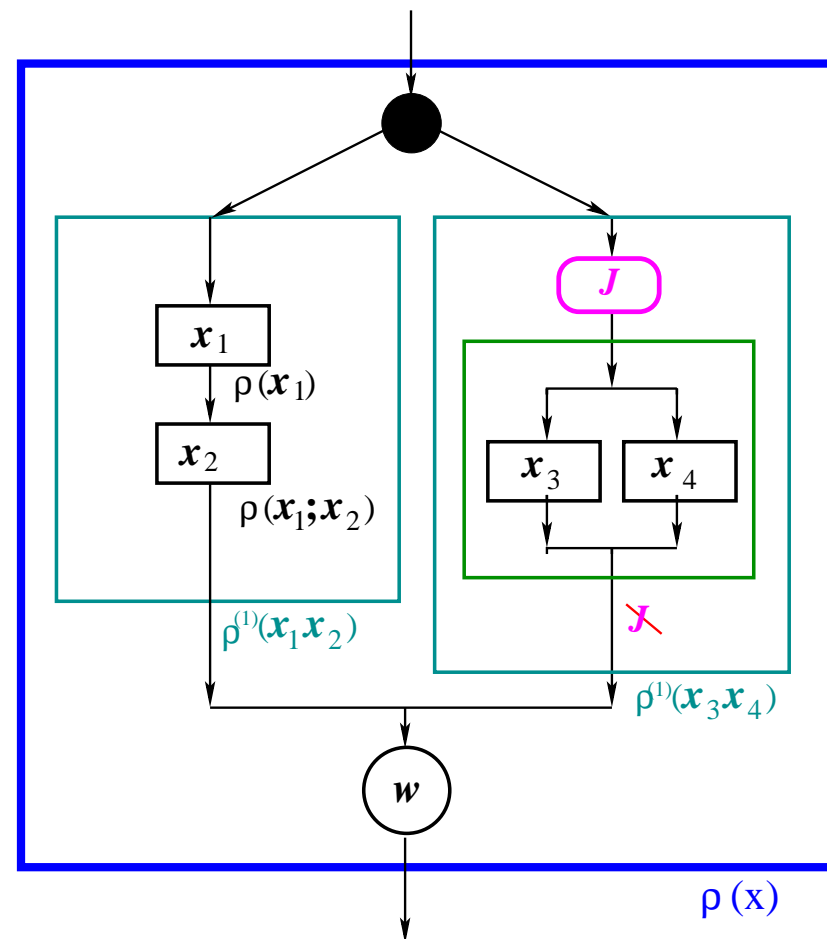
Quite often, the mapping (or other method) generation of variable, or groups of them, **is sequential**, that is, distribution of the next variable (down the three) involves previously generated variable as a parameter. Without entering into details, this may be indicated easily in the graph of the MC simulation algorithm. (Parallel processing inhibited.)



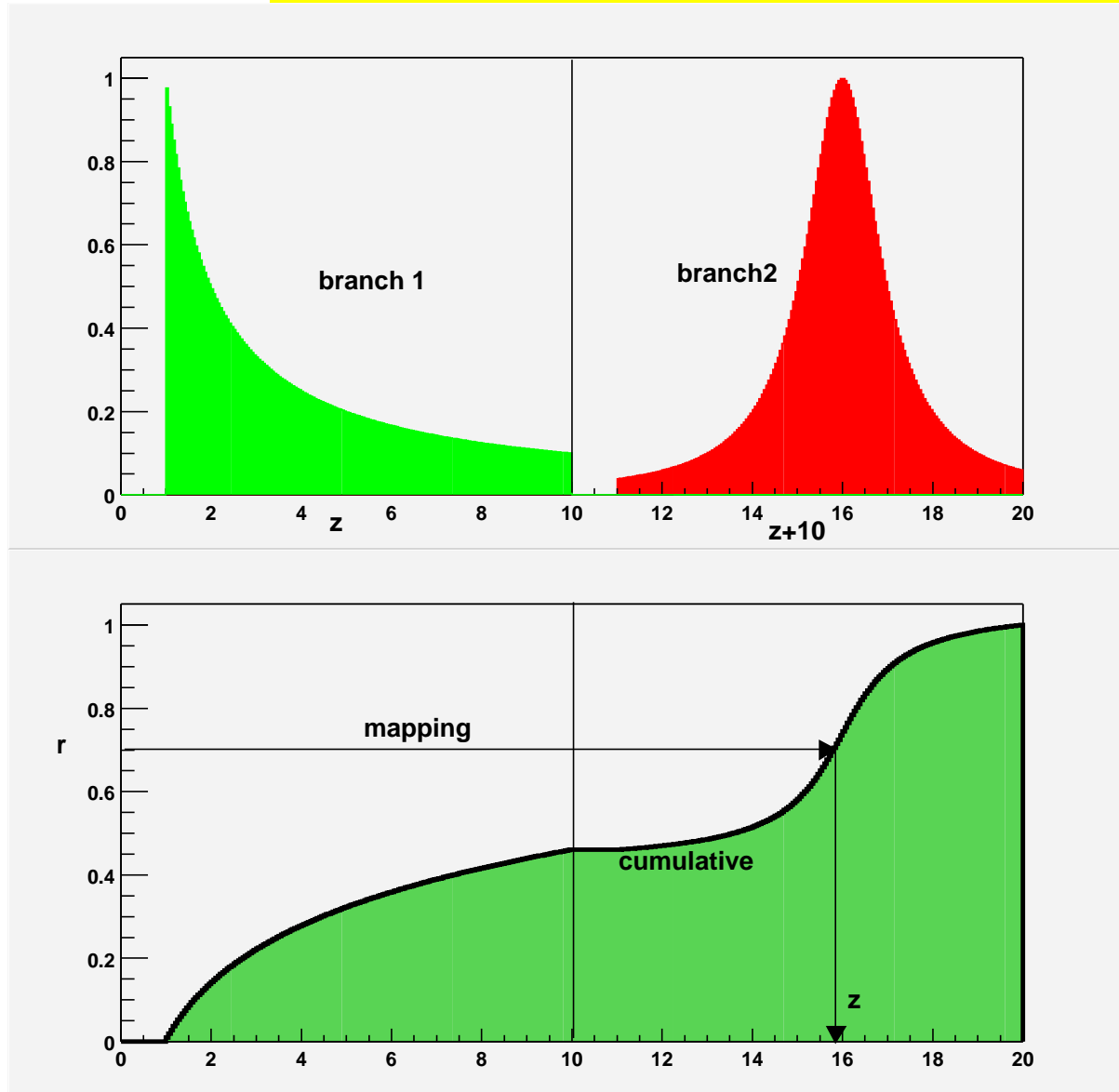
All together!

If you doubt that one may construct infinitely complicated MC using three simple elementary methods methods....:

First, we factorize $\rho(x_1, \dots, x_4) \rightarrow \rho^{(1)}(x_1, x_2) \times \rho^{(1)}(x_1, x_4)$ and compensate with rejection. Then, $\rho^{(1)}(x_1, x_2)$ is generated using “sequential mapping” and, in parallel, $\rho^{(1)}(x_1, x_4)$ is modelled using branching, where in each branch we get perfect factorization. Simple?



Note: Mapping may replace branching at 1-dimension



Conclusions

- **Rejection (and weighted events) is the king of MC methods.**
- **Branching (multichannel) can help a lot.**
- **There is at least 3 ways of combining branching and rejection.**
- **Mapping very useful but strongly limited to finite many of elementary functions.**
- **The real art is to combine them various methods.**
- **Don't cross out "general purpose" adaptive methods! See next lecture.**