

**ADVANCED PRACTICAL MC METHODS**

**by S. Jadach**

**Institute of Nucl. Physics, Kraków, Poland**

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

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

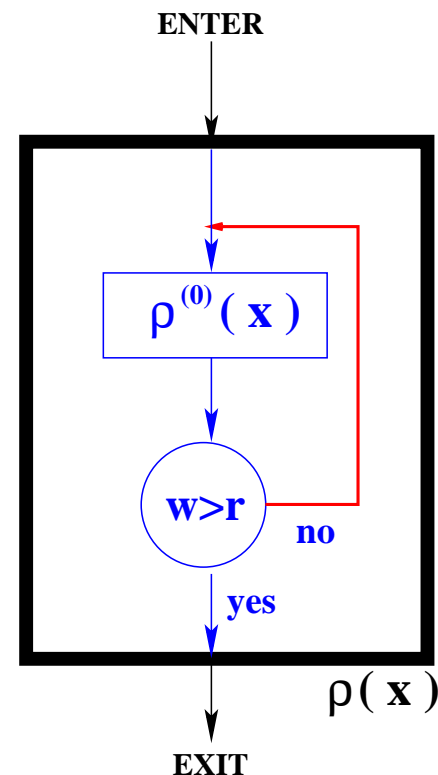
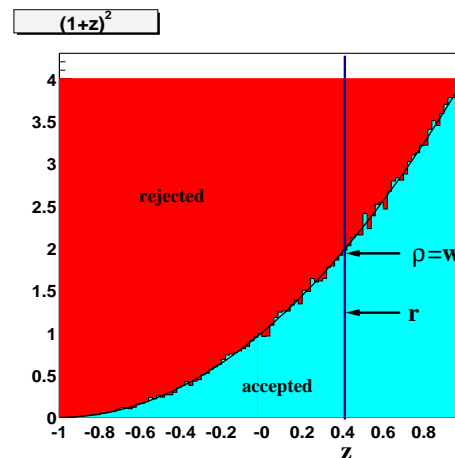
## Outline:

1. Illustrative examples of 3 elementary methods:  
rejection, branching and mapping
2. The big game – combining 3 elementary methods:  
rejection, branching and mapping
3. General Purpose MC methods:
  - Vegas algorithm
  - Cellular algorithms: Foam

## Rejection method: Simple illustration

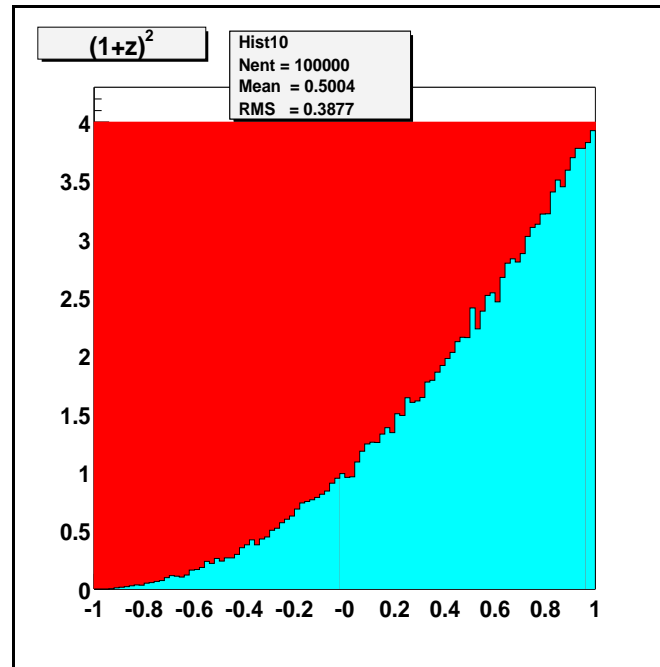
```

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  $N_{acc}/N_{tot}$ :  $R, \text{del}R = 2.66293 \pm 0.00687806$

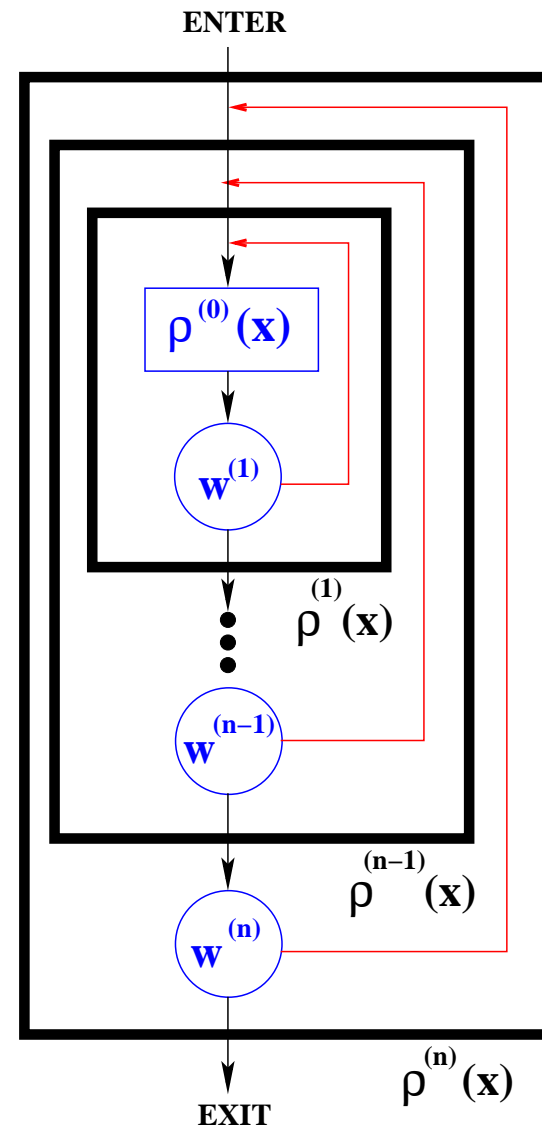
From  $\text{variance}(wt)$ :  $R, \text{del}R = 2.66168 \pm 0.00435244$

## Multiple (nested) 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.



## Opening rejection loops: weighted events

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

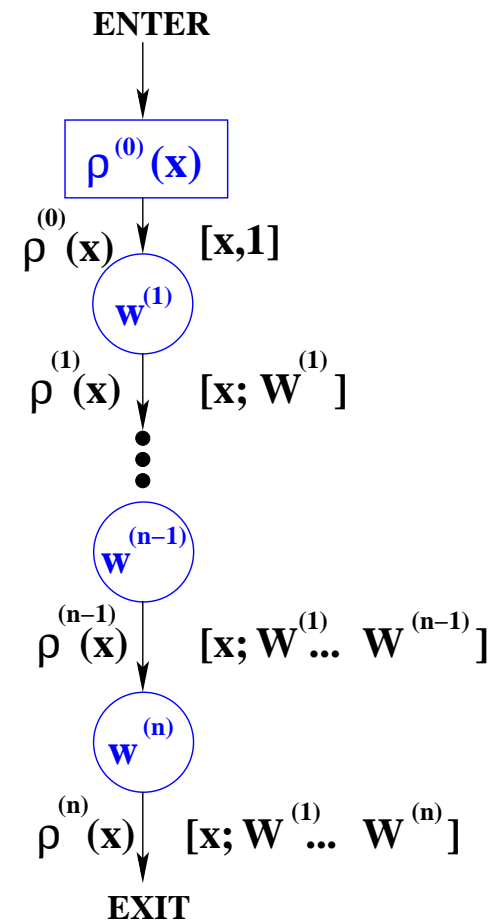
For nested loops the total weight is the product of all intermediate weights is:

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

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!



## 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”:

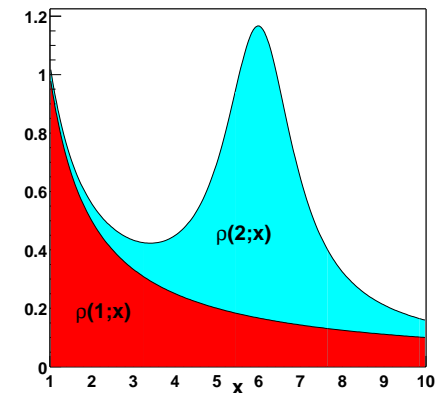
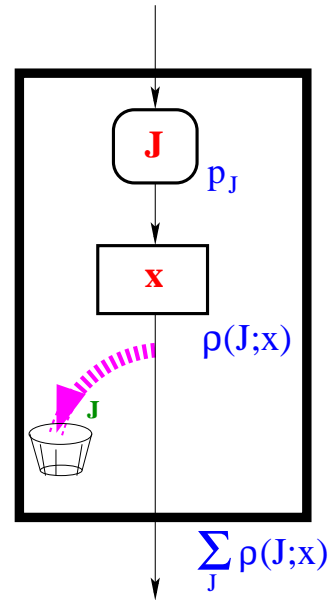
- 3-dim. Gaussian: simulate 4-dimen. Gaussian and trash 4-th component,
  - $p(x) = \frac{1}{4}x^3$ : out of 4 uniform rn. numbers, take the smallest, trash 3 others,
  - Uniform distr. of  $n$ -dim sphere: simulate  $n$ -dim. Gaussian and take  $\frac{x}{|x|}$ , (trash radius).
- and many other useful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

## Branching method: Simple example 1

Let us try to simulate:

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

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

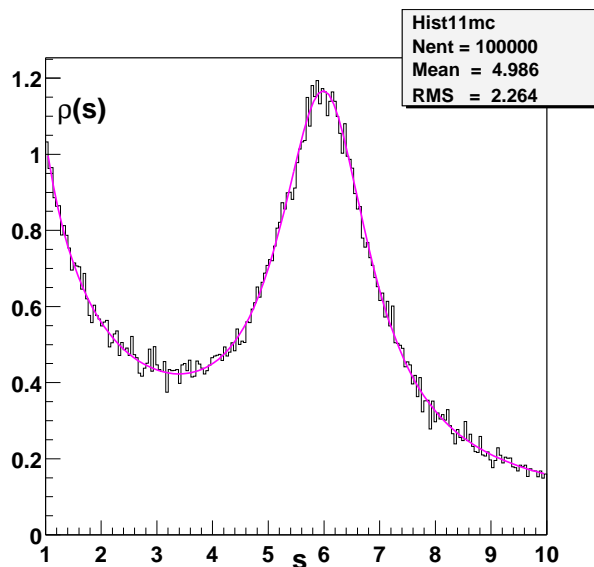
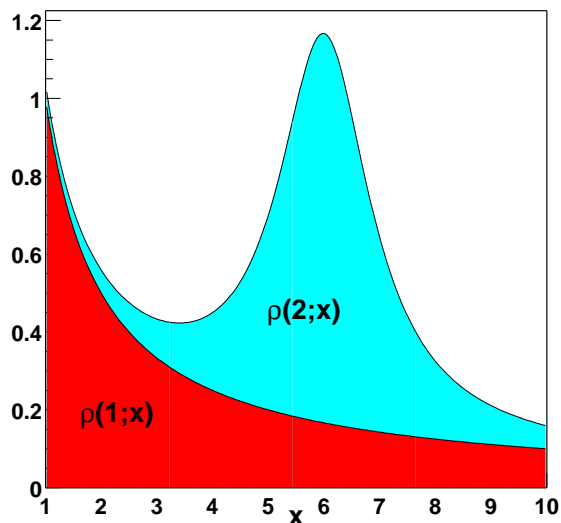


This is a kind of Breit-Wigner “resonance” (mass= $\sqrt{6}$ , width =1) plus non-resonant “background” distribution, added incoherently (no interference).

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

**NOTE: 1-dim. mapping involves analytical exact evaluation 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 in each branch

Explicit formulas:

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

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

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

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

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

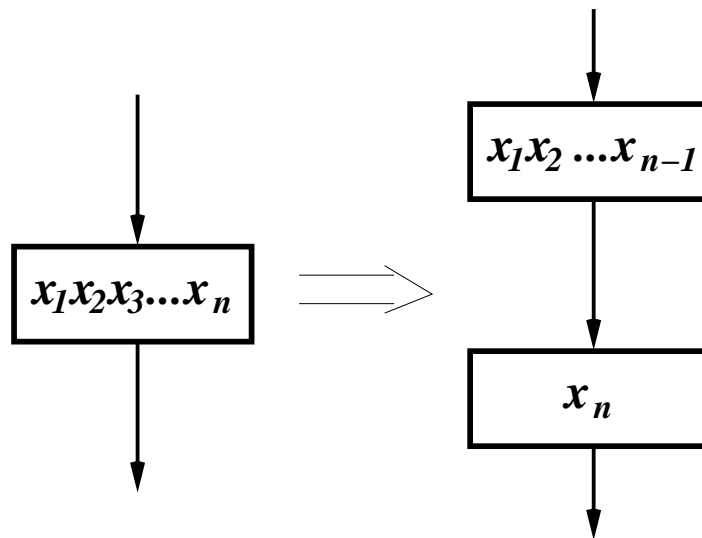
$$r = \frac{\arctan((x-a)/\gamma) - \arctan((x_1-a)/\gamma)}{\arctan((x_2-a)/\gamma) - \arctan((x_1-a)/\gamma)} \in (0, 1),$$

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

**More on mapping: Sequential mapping**

With a bit of luck we are able to do mapping for  $x_n$ , keeping  $x_1 \dots x_{n-1}$  as constant parameters (familiar integration technique):

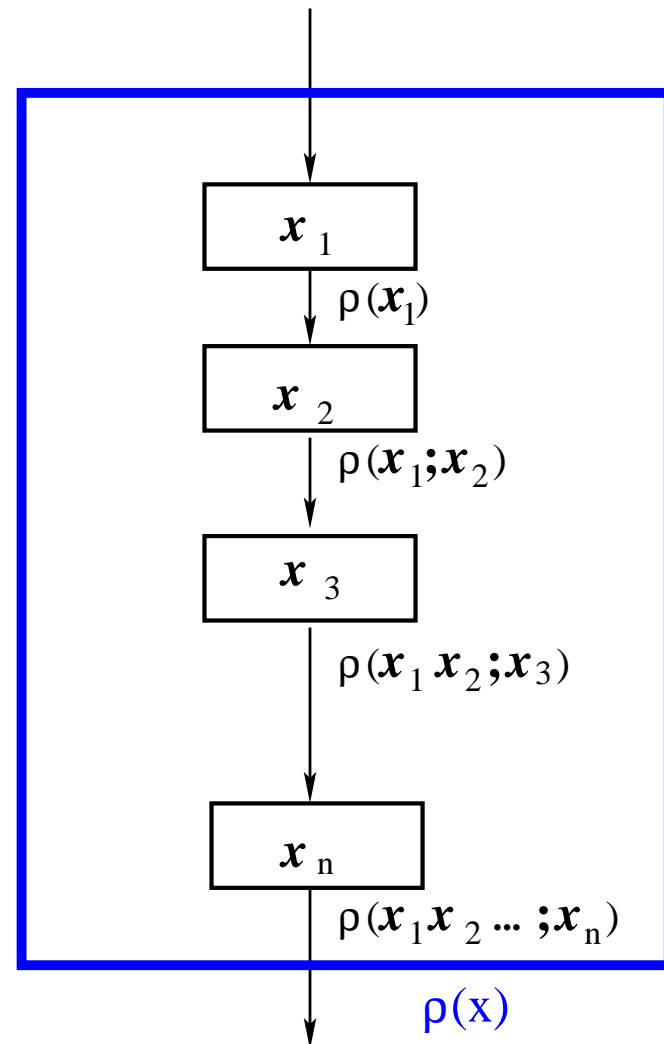
$\int dx_1 \dots dx_{n-1} \int dx_n \rho'(x_1 \dots x_{n-1}; x_n)$ , i.e. the integration over  $x_n$  can be done analytically, and its cumulative can be inverted.



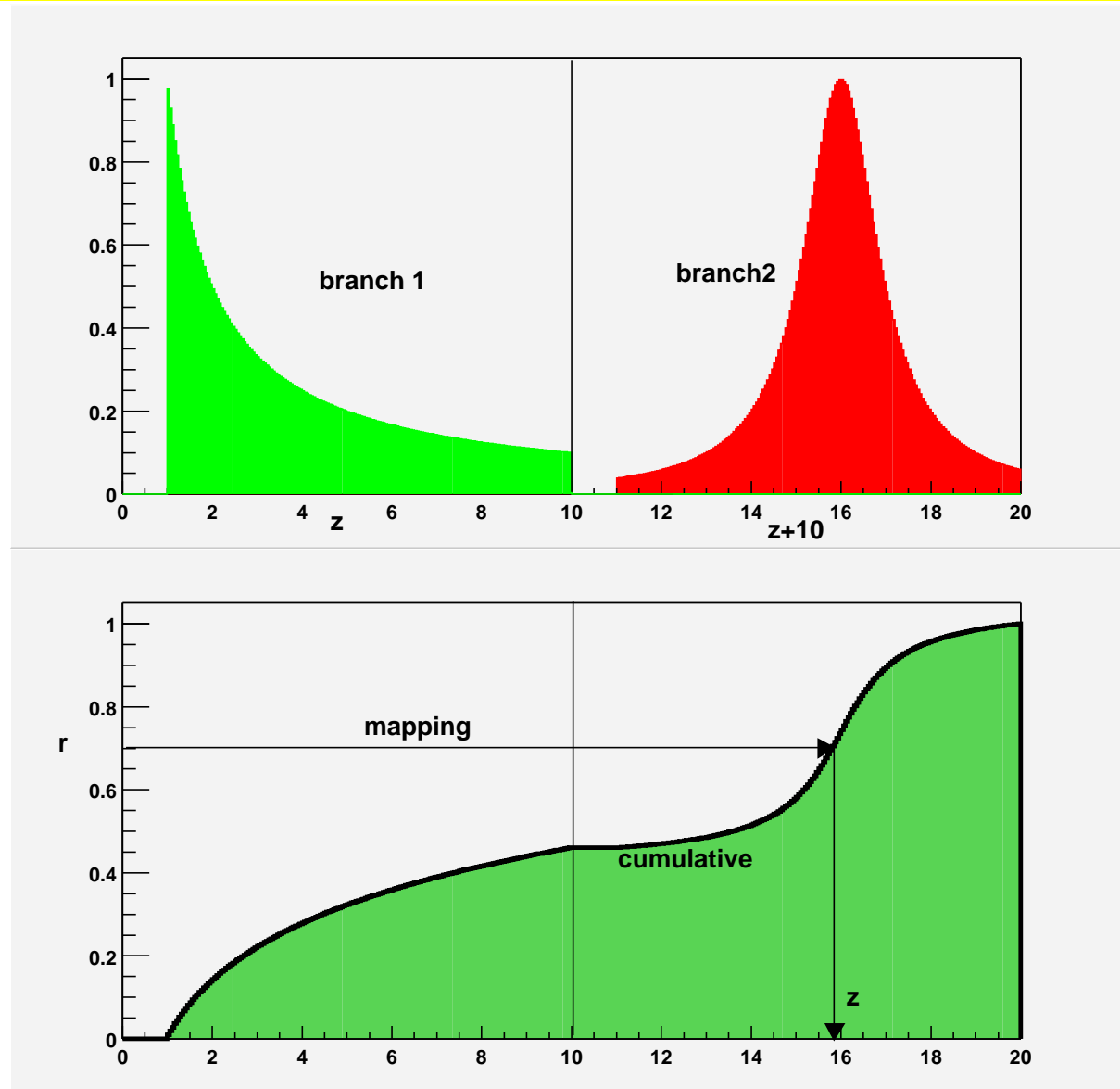
Hopefully, this procedure can be repeated  $n$  times, leading to... see next slide.

## Multiple sequential mapping

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.)



**DIGRESSION: Mapping may replace branching at 1-dimension**



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.

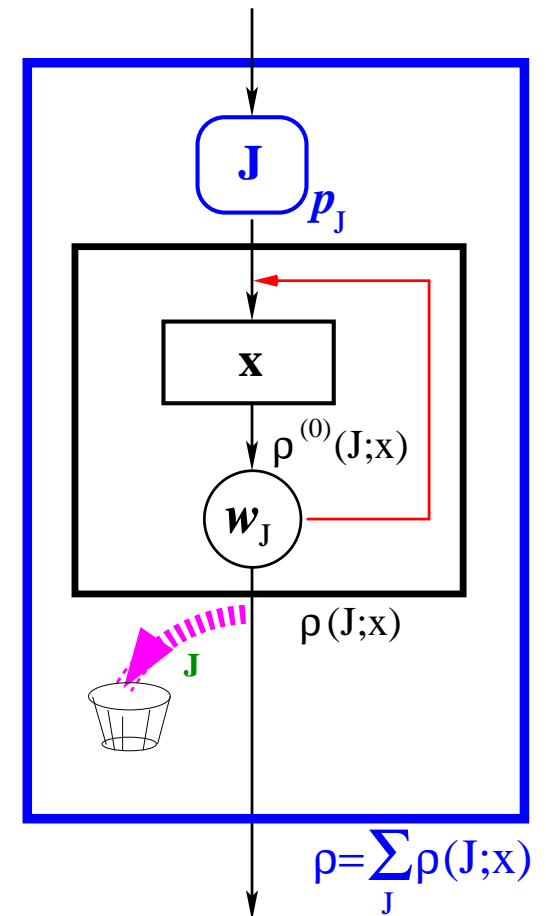
PROBLEM:  $p_J$  not known in advance.

$R_J$  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.

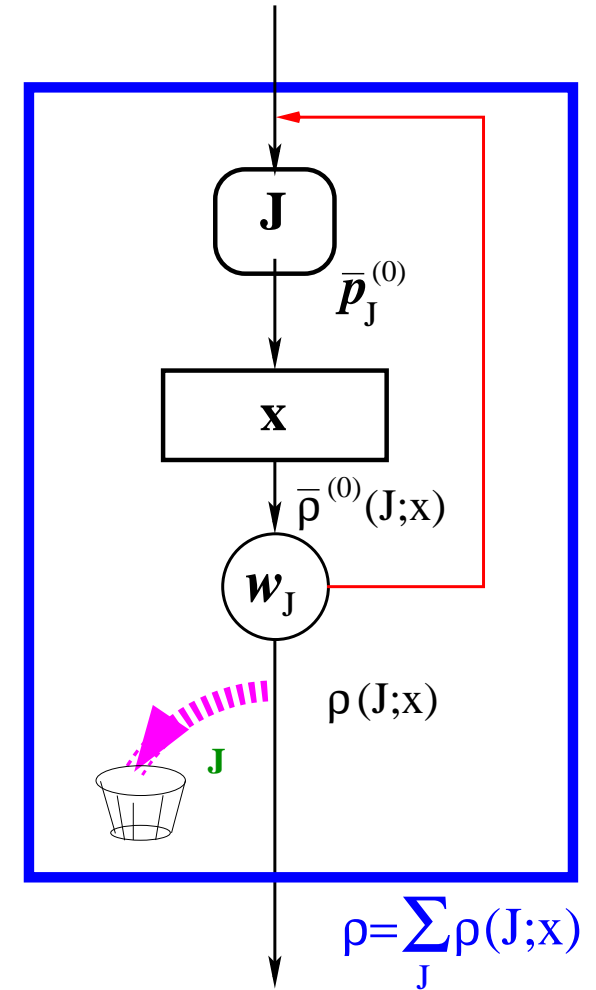
Rejection weight  $\bar{w}_J = \frac{\rho(J;x)}{\bar{\rho}^{(0)}(J;x)}$  is for the actual  $J$ -th branch and  $J$  is subsequently “trashed”.

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  $\bar{p}_J$ :

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

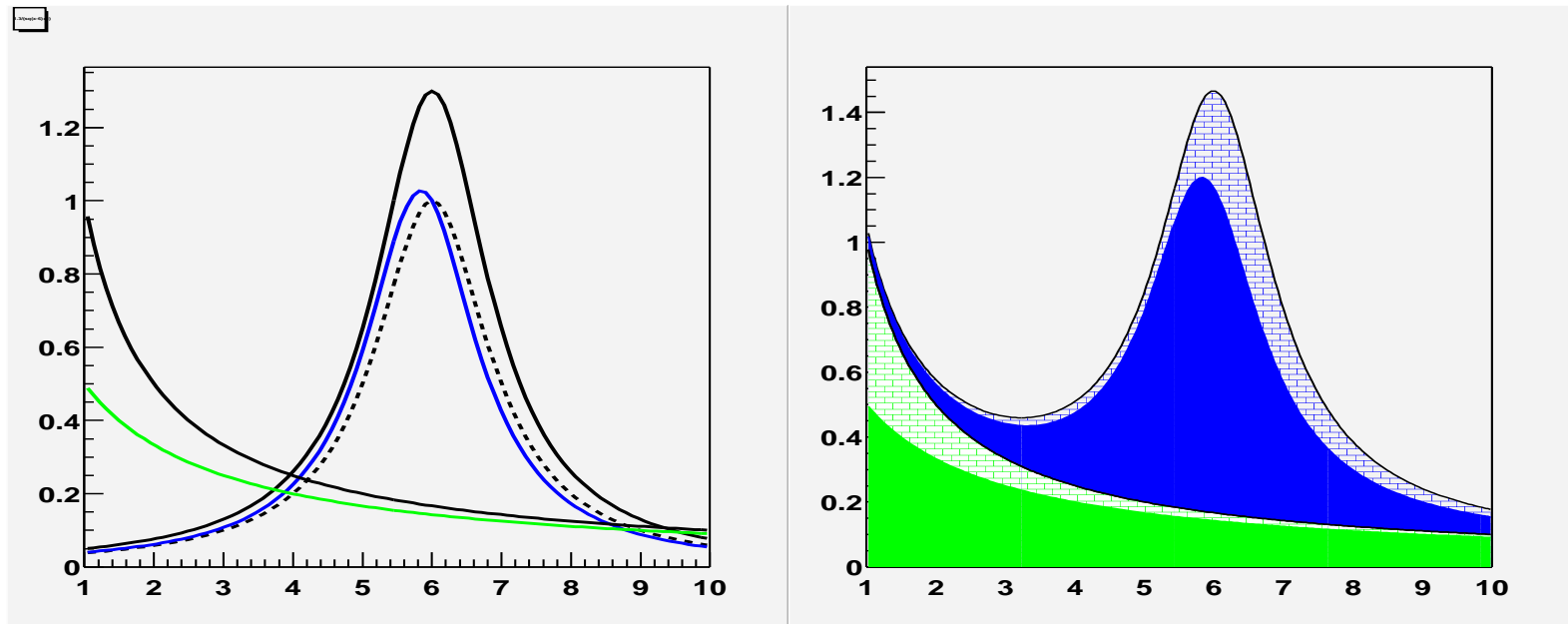


Example of method (B), details of distributions

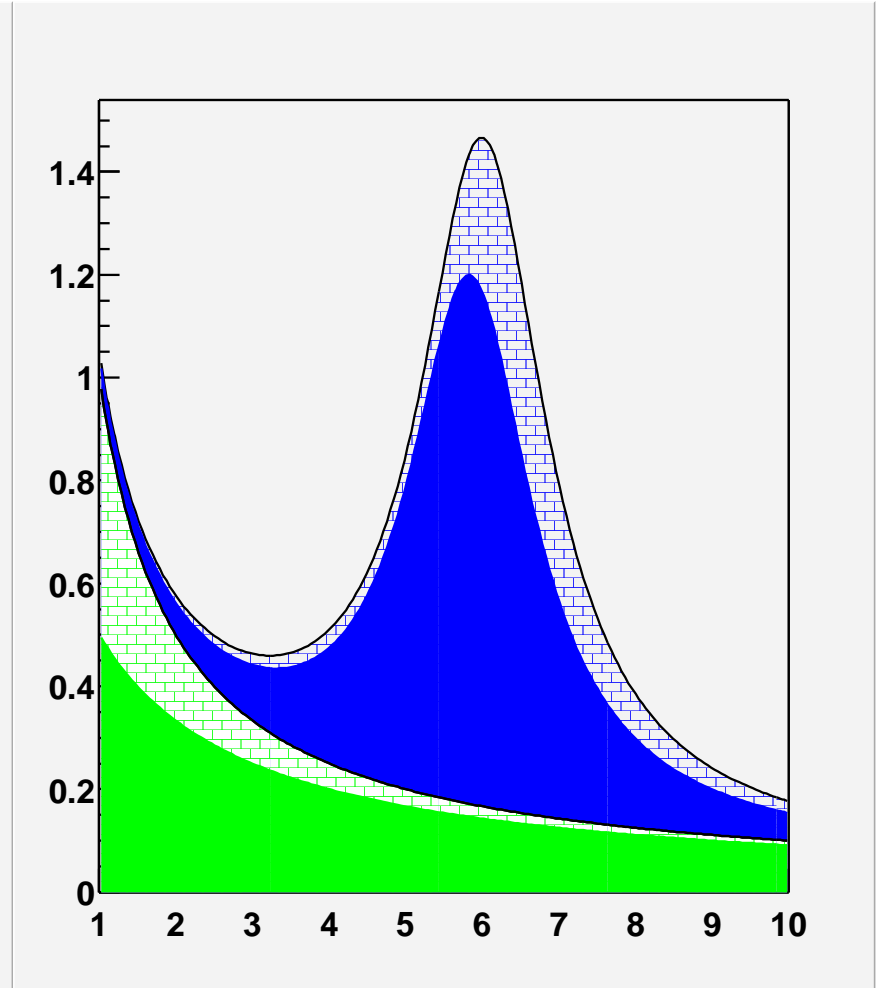
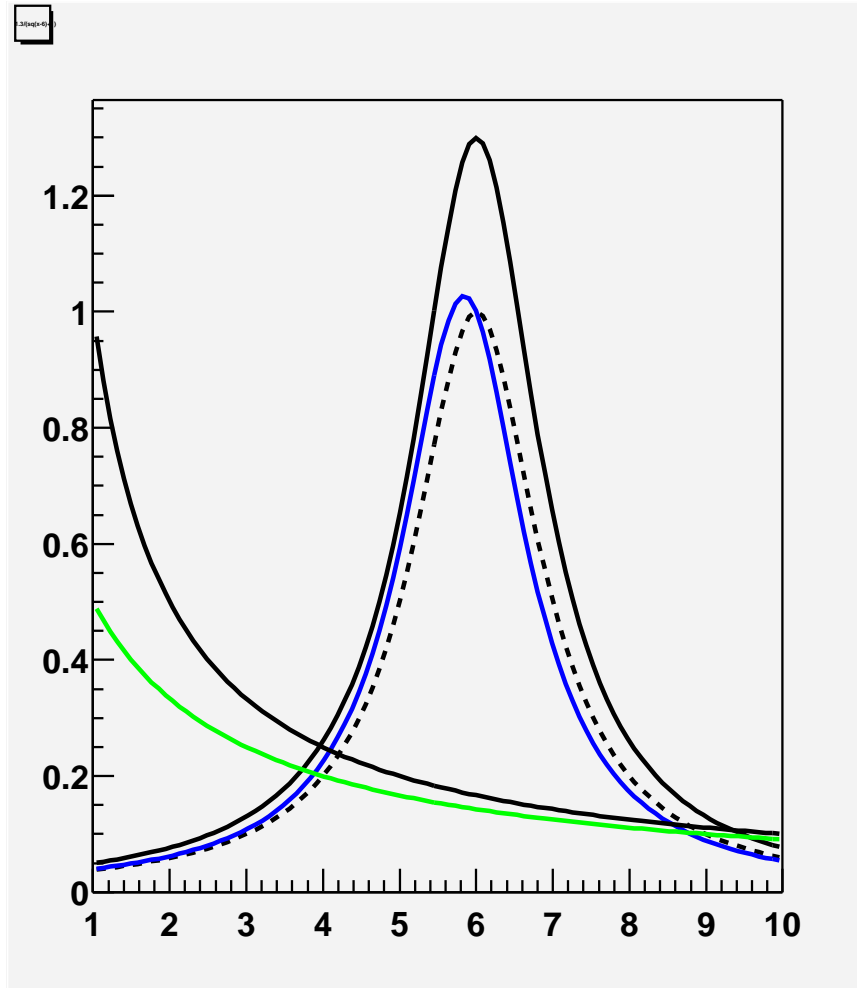
Two distributions in the branches are “background”  $B(s) = \left| \frac{-1}{(1+s)^{1/2}} \right|^2 = \frac{1}{1+s}$  and “resonance”  $R(s) = \left| \frac{1}{(s-6)+i(s/6)} \right|^2 = \frac{1}{(s-6)^2+(s/6)^2}$ . added incoherently.

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.

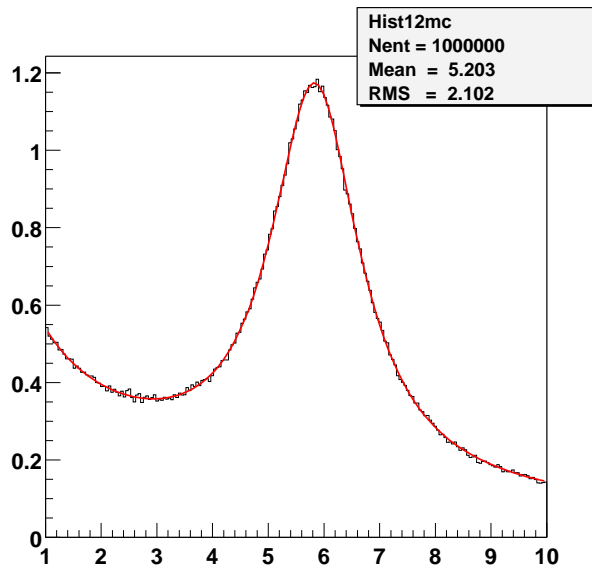
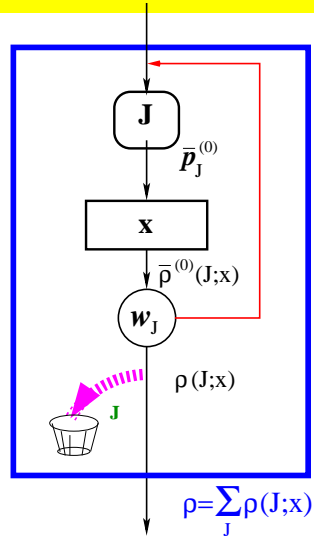


Combining Rejection and Branching (B) cont.



Two-branch example: "Brick-walled" part is rejected. "Solid-colour" part accepted.

## 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 *RNgen, double &s){
    // generates single event, 2 branches
    RESTART:
        Double_t wt,A1,A2;
        Double_t r1 = RNgen->Rndm(0);
        Double_t r2 = RNgen->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 = RNgen->Rndm(0);
        if( r3 > wt ) goto RESTART;
    }
}
    
```

Combining Rejection and Branching (B) cont.

**NORMALISATION:**

Total integral is a sum over integrals from all

branches  $R = \sum_J R_J$  where

$R_J = R_J^{(0)} \langle w_J \rangle = \bar{R}_J^{(0)} \langle \bar{w}_J \rangle$ . This yields:

$$R = \bar{R}^{(0)} \sum_J \bar{p}_k \langle \bar{w}_J \rangle = \bar{R}^{(0)} \langle \langle \bar{w} \rangle \rangle, \text{ where}$$

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

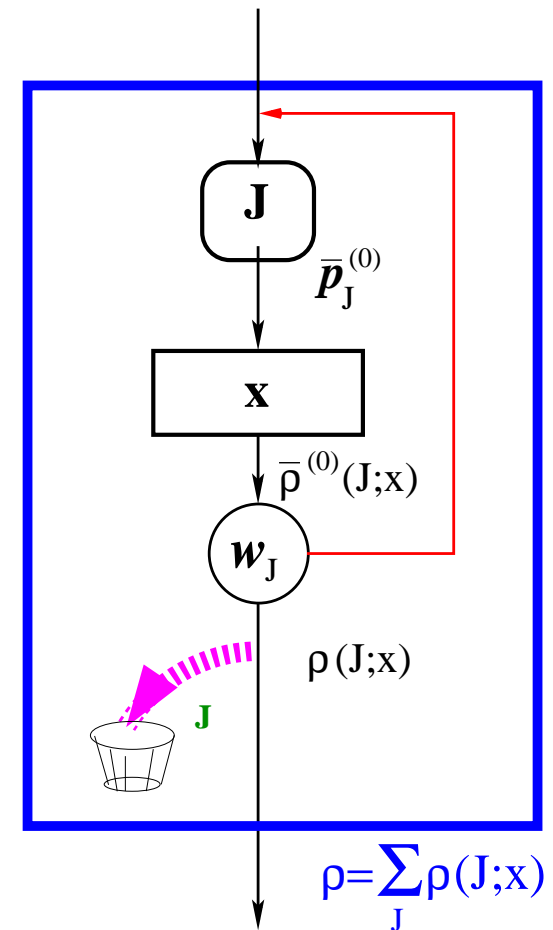
Formula for the total integral based on the number of accepted events  $\sigma = \bar{R}^{(0)} \frac{N}{N^{(0)}} = \bar{R}^{(0)} \frac{\sum_J N_J}{\sum_J N_J^{(0)}}$  is easily derivable.

```
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 the numerical value of the weight  $w_J$  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  $R = \bar{R}^{(0)} \langle w \rangle$ .

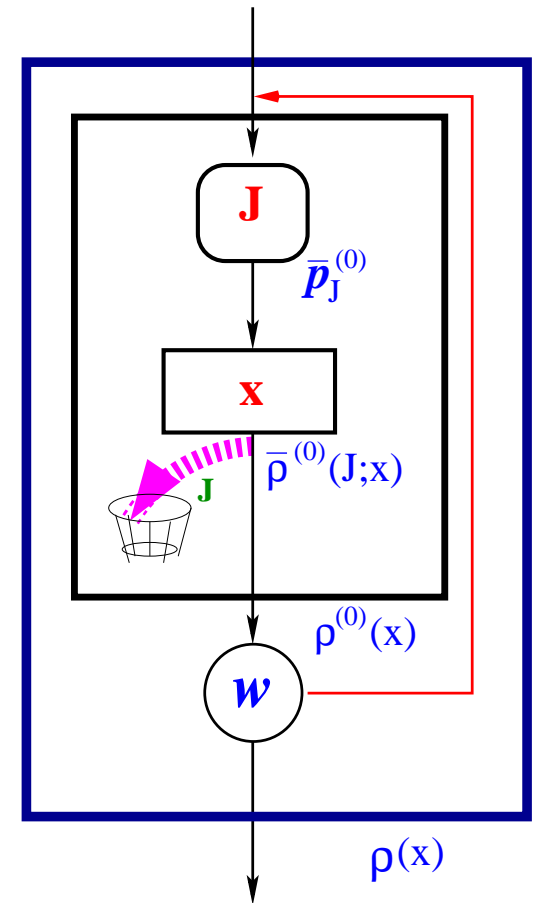
Disadvantage: sometimes, for large number of branches evaluation of the weights can be slow and complicated.

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 recipe of Kleiss and Pittau (Comput. Phys.

Commun. 83, 141-146, 1994).



Rejection + Branching, "Real Life example"

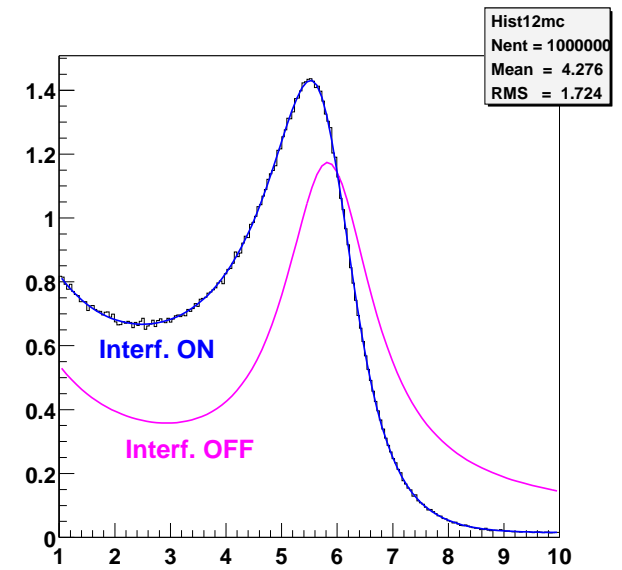
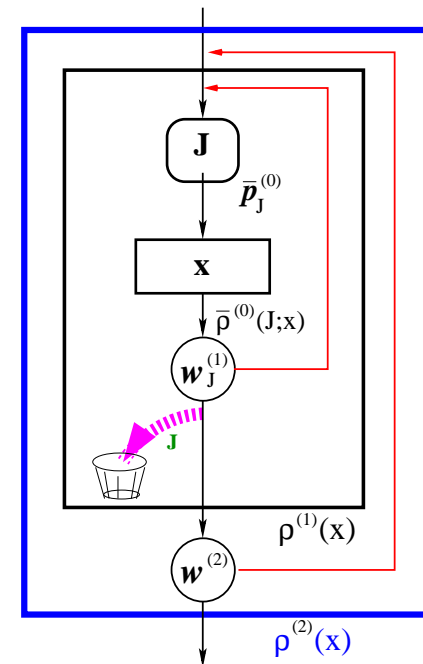
On top of branch-specific effects there might be effects and corresponding weights, which **cannot be attributed to any specific branch.**

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.



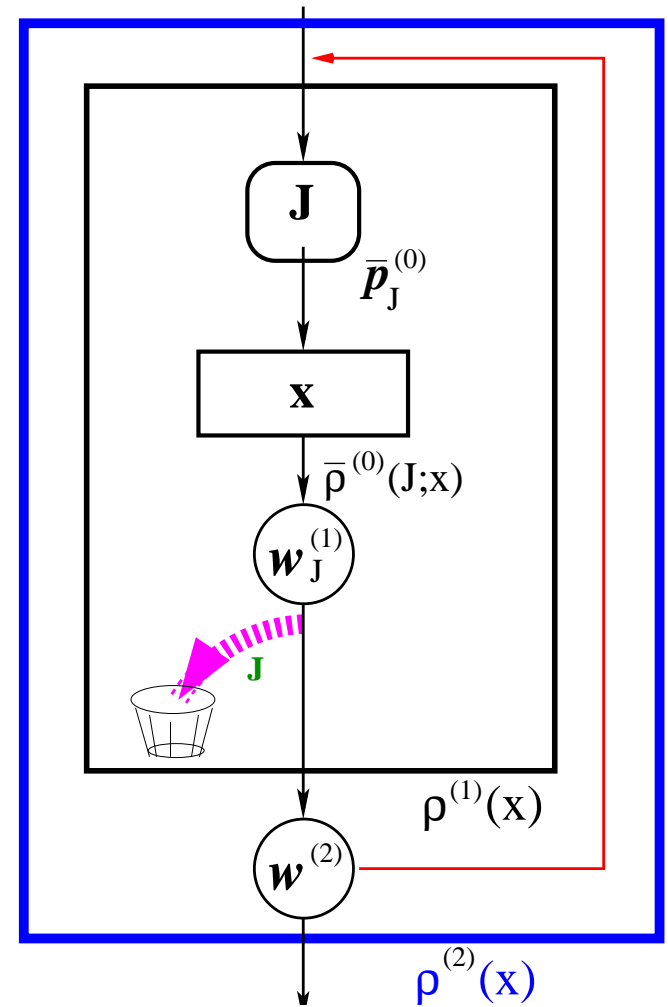
```

.....
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;
}
.....

```



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

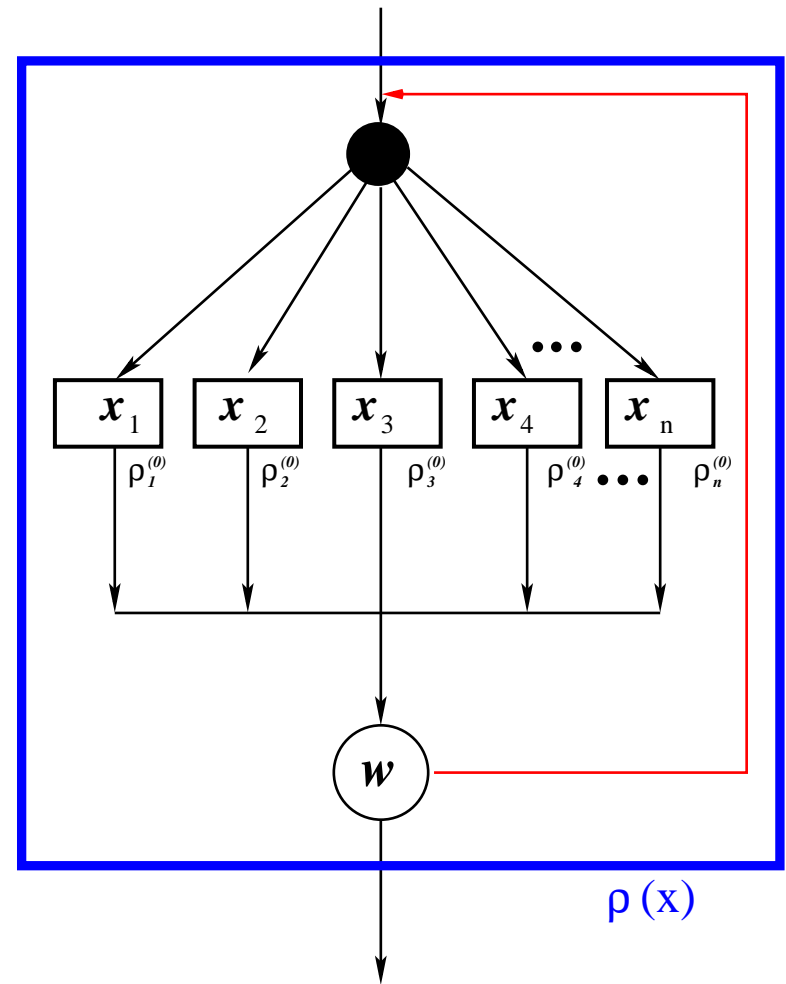
**Factorizing can also help!**

The basic method is to find simpler  $\rho$  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:

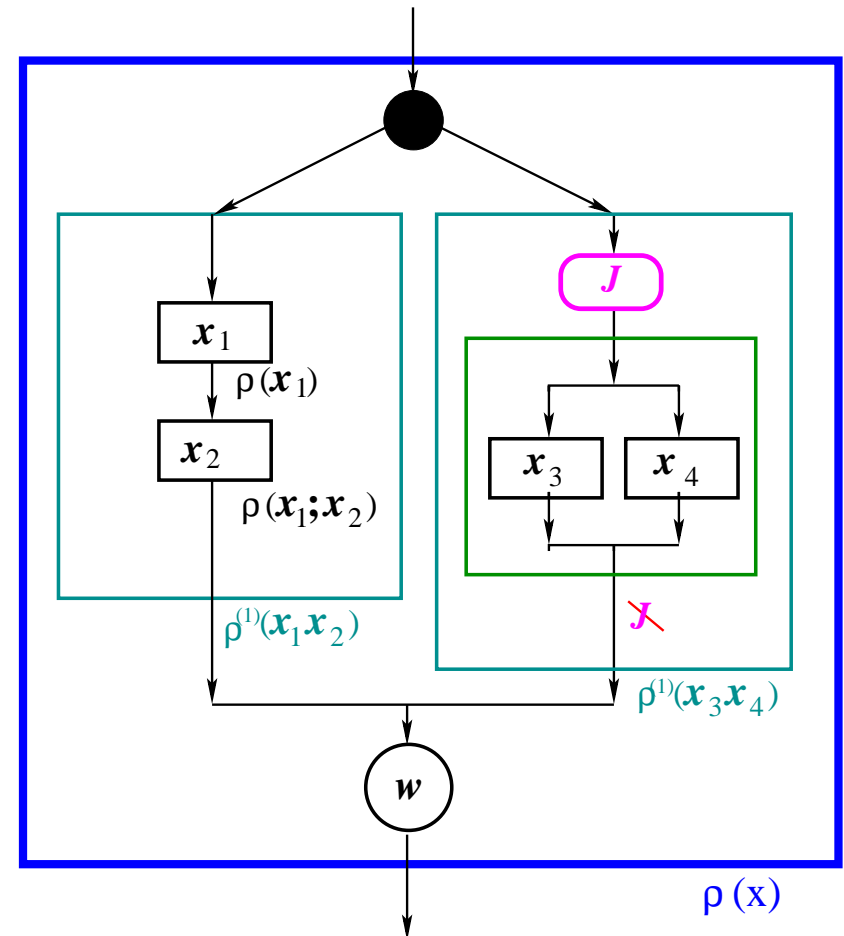
Our graph underlines parallel processing possibilities!



All methods together!

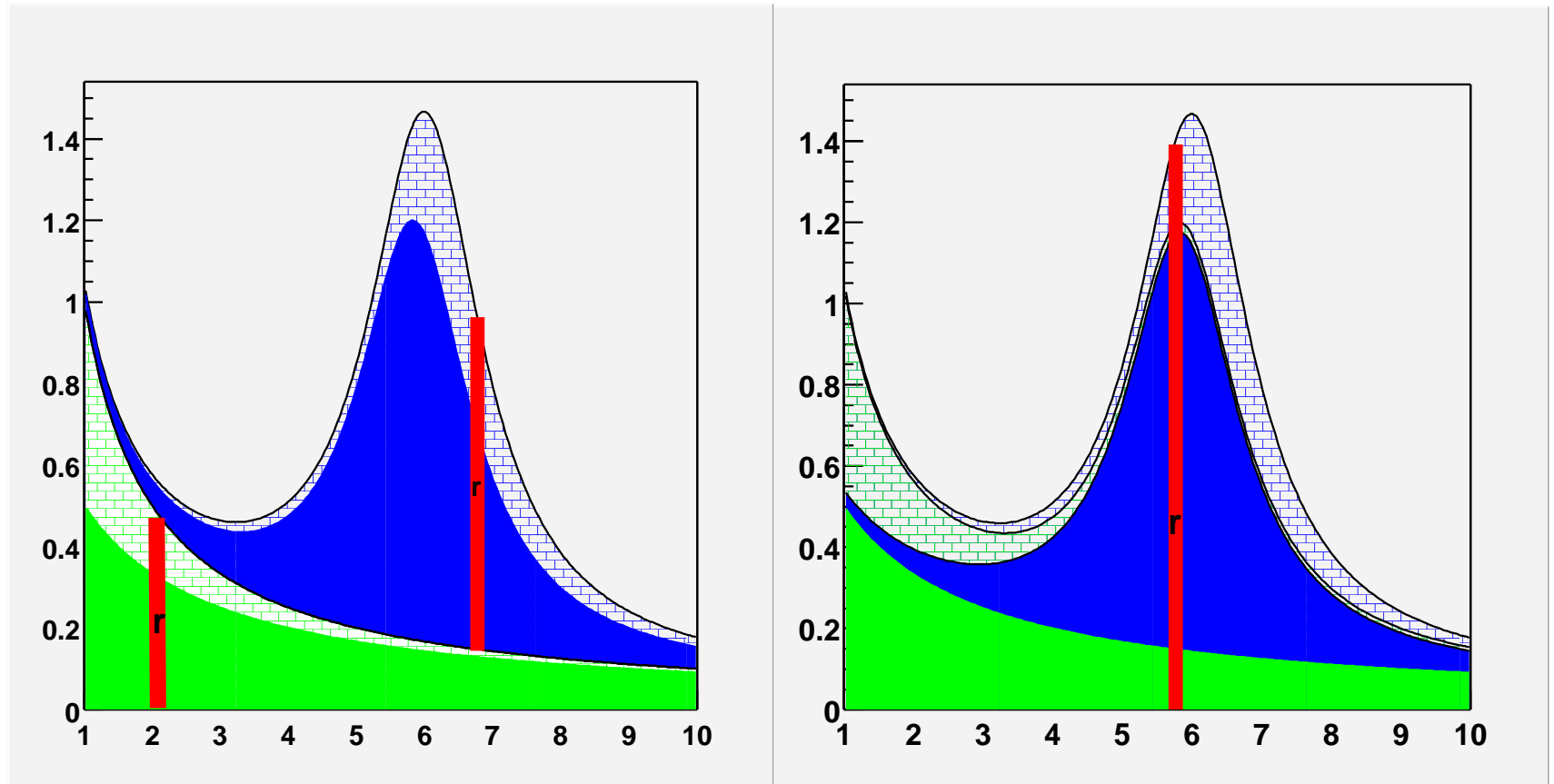
If you doubt that one may construct infinitely complicated MC using three simple elementary methods methods....:

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?



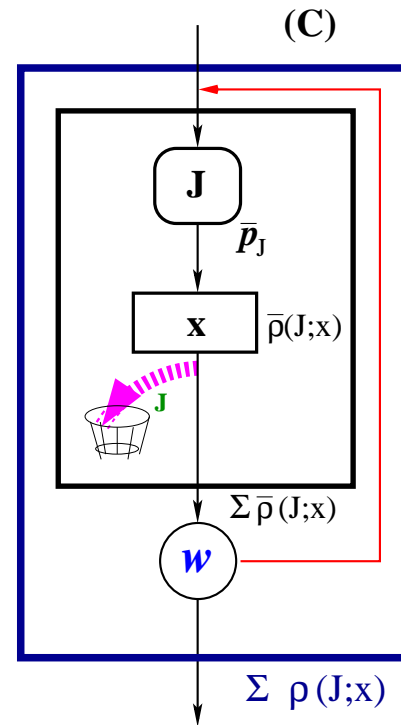
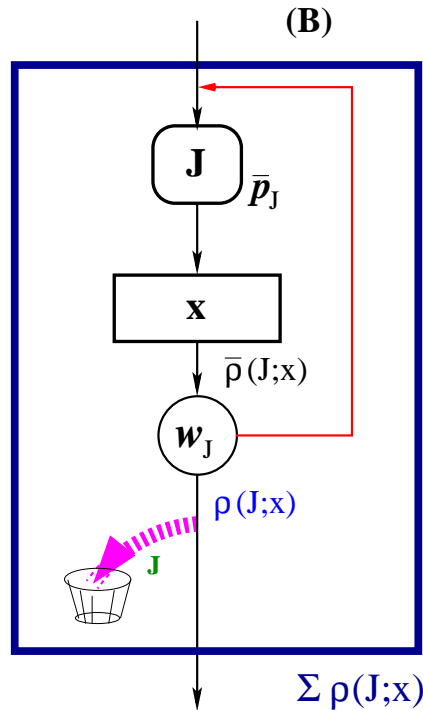
**Equivalent algorithms (pictorial)**

We may encounter “Equivalent algorithms” which provide the same distributions and integrals, use the same elementary methods and differ only in the efficiency. They may have the same average weight but different weight distribution. In the next slide there is an example of such 2 algorithms.



**Equivalent algorithms (formulas, graphs)**

Define:  $\langle\langle U \rangle\rangle \equiv \sum_J \bar{p}_J \int \frac{\bar{\rho}(J;x)}{R_J} U(J;x) dx^n \equiv \frac{1}{R} \sum_J \int \bar{\rho}(J;x) U(J;x) dx^n$



$$w_B(J;x) = \frac{\rho(J;x)}{\bar{\rho}(J;x)}$$

$$\langle\langle w_B \rangle\rangle = \int \sum_j \rho(J;x) = R$$

$$\langle\langle w_B^2 \rangle\rangle = \frac{1}{R} \int \sum_J \frac{\rho^2(J;x)}{\bar{\rho}(J;x)} dx^n$$

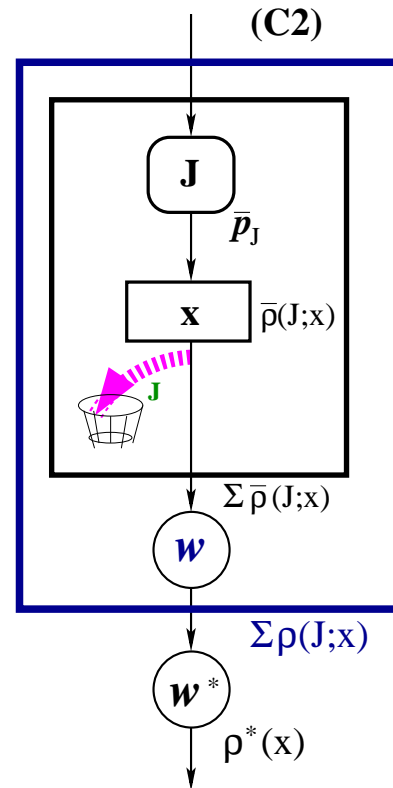
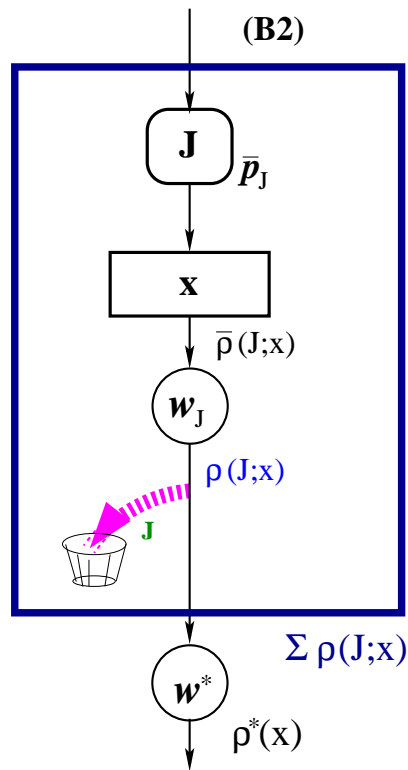
$$w_C(x) = \frac{\sum_J \rho(J;x)}{\sum_J \bar{\rho}(J;x)}$$

$$\langle\langle w_C \rangle\rangle = \int \sum_j \rho(J;x) = R$$

$$\langle\langle w_C^2 \rangle\rangle = \frac{1}{R} \int \frac{(\sum_J \rho(J;x))^2}{\sum_J \bar{\rho}(J;x)} dx^n$$

**Equivalent algorithms - real life**

The “equivalence” looks even more interesting with additional branch-independent weight  $w^*$  and open rejection loops:



$$w = \frac{\rho(J;x)}{\bar{\rho}(J;x)} \frac{\rho^*(x)}{\sum_J \rho(J;x)}$$

$$w = \frac{\sum_J \rho(J;x)}{\sum_J \bar{\rho}(J;x)} \frac{\rho^*(x)}{\sum_J \rho(J;x)} = \frac{\rho^*(x)}{\sum_J \bar{\rho}(J;x)}$$

**Branch optimization in case of wt-ed events (Kleiss&Pittau)**

Consider (C) with “opened rejection loop”. Target is  $\rho(x)$ .

Generated primarily:  $\bar{\rho}(x) = \sum \bar{\rho}(J; x)$  (mapping).

Normalization “anchor”:

$$\bar{R} = \sum_J \bar{R}_J = \sum_J \int \bar{\rho}(J; x) dx^n . \text{ Integral is}$$

$$R = \bar{R} \langle \langle w \rangle \rangle , \text{ where the average is over branches and}$$

over integration domain. The MC error is

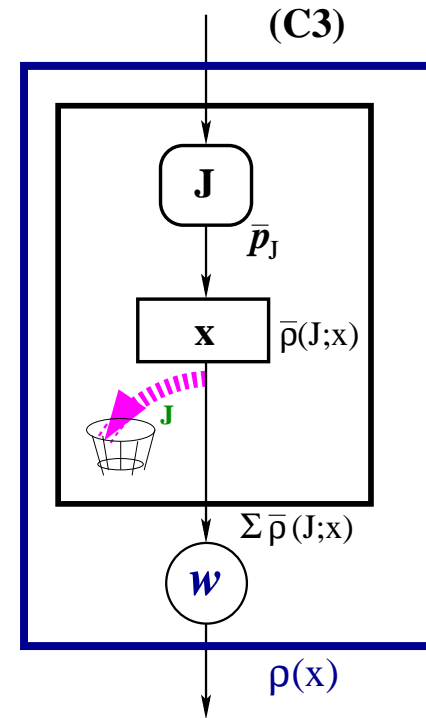
$$\delta R = \frac{\sigma}{\sqrt{N}} = \frac{\bar{G}}{\sqrt{N}} \sqrt{\langle \langle w^2 \rangle \rangle - \langle \langle w \rangle \rangle^2}$$

Define “MC inefficiency” as  $iE = N \frac{\delta R^2}{R^2} + 1 = \frac{\langle w^2 \rangle}{\langle w \rangle^2}$

$$\text{We find: } iE = \frac{1}{\int \rho(x) dx^n} \bar{R} \int \frac{(\rho(x))^2}{\sum_J \bar{\rho}(J; x)} dx^n$$

Our aim is to get **the smallest  $iE$**  by manipulating relative overall normalizations of the branch distributions

$$\bar{\rho}(J; x) \rightarrow \lambda_J \bar{\rho}(J; x) .$$



$$w = \frac{\rho(x)}{\sum_J \bar{\rho}(J; x)}$$

### Branch optimization in case of wt-ed events, cont

Our aim is to get **the smallest  $iE$**  by manipulating relative normalizations of the branch distributions  $\bar{\rho}(J; x) \rightarrow \lambda_J \bar{\rho}(J; x)$ . ( $\lambda_J$  are independent of  $x$ !)

We look for a minimum of  $E = A \int \frac{(\rho(x))^2}{\sum_J \bar{\rho}(J; x)} dx^n \bar{R}$  as a function of  $\lambda_J$ .

Obviously  $\bar{\rho}(J; x) \rightarrow \lambda \bar{\rho}(J; x)$  changes nothing.

Let us put constraint  $\bar{R} = \sum_J \lambda_J \bar{R}_J = \text{const}$ , to fix normalization.

By means of Lagrange method, differentiation  $\partial/\partial\lambda_J$  provides us local minimum condition:

$$W_K = \int \frac{(\rho(x))^2}{(\sum_J \bar{\rho}(J; x))^2} \frac{\bar{\rho}(K; x)}{\bar{R}_K} dx^n = B, \quad \text{all } W_K \text{ the same, independent of } K!$$

Kleiss&Pittau proposed iterative solution in which one starts from, say,  $\lambda_J = 1$ , then

$W_K = \langle \langle w^2 \frac{\bar{\rho}(K; x)}{\bar{R}_K} \frac{\bar{R}}{\bar{\rho}(x)} \rangle \rangle$  is calculated from the trial MC run and for the next trial run we

$$\text{substitute } \lambda_J \rightarrow \lambda_J \sqrt{W_J}.$$

This ansatz is based on the observation that for  $\rho(J; x)$  distributions which are vanishingly overlapping, we can solve the “minimum condition equations”, almost exactly, just after first iteration! (modulo statistical errors). See next slide....

### Branch optimization in case of wt-ed events, cont.

How to understand the origin of K&P ansatz?

Imagine that the target distribution  $\rho(x)$  has just two distinct peaks with almost zero overlap, located approximately in two subdomains (1) and (2).

We have at our disposal two “primitive” distributions  $\bar{\rho}(1; x)$  and  $\bar{\rho}(2; x)$  which fit very well  $\rho(x)$ , up to a normalization. Furthermore,  $\bar{\rho}(1; x)$  is nonzero in (1) and almost zero in (2), while  $\bar{\rho}(2; x)$  is nonzero in (2) and almost zero in (1).

Even more, let us assume that someone told us correct optimal normalization of  $\bar{\rho}(I; x)$ , so we know that the two integrals:

$$W_K = \int \frac{(\rho(x))^2}{(\sum_J \bar{\rho}(J; x))^2} \frac{\bar{\rho}(K; x)}{R_K} dx^n = B \text{ are equal (no } K \text{ dependence).}$$

Nevertheless we ignore this information and we run trial MC using “wrong”

$$\bar{\rho}'(J; x) = \lambda_J \bar{\rho}(J; x) \text{ and obtain “wrong” } W'_1 \text{ and } W'_2.$$

Since two integrals very well “localized” in the space, we get:

$$W'_1 \simeq \frac{1}{\lambda_1^2 \bar{R}_1} \int \frac{\rho^2(x)}{\bar{\rho}(1; x)} dx^n = \frac{B}{\lambda_1^2} \text{ and } W'_2 \simeq \frac{1}{\lambda_2^2 \bar{R}_2} \int \frac{\rho^2(x)}{\bar{\rho}(2; x)} dx^n = \frac{B}{\lambda_2^2}.$$

Obviously, to get back to original correct normalization,

we should to correct back our “wrong”  $\bar{\rho}'(I; x)$  back to original, using factor

$$\lambda_I^{-1} = (W'_I)^{1/2} \quad I = 1, 2. \text{ (Modulo constraint } \bar{R} = \bar{R}_1 + \bar{R}_2 = \text{const, of course.)}$$

## What is **General Purpose MC**?

For the problem of function minimalization one takes MINUIT or some other program and applies it to arbitrary user-function. One may find also **general purpose** programs for integration of “arbitrary” integrand. **General-purpose** means that these tools work, in principle, for a very wide range of user-functions.

For **multi-dimensional Monte Carlo simulation** problem, that is for the problem of generation randomly points according to a given  $n$ -dimensional distribution, there is precious little examples of the **General Purpose** Monte Carlo Simulators (GPMCS), that is programs which work (in principle) for arbitrary integrand – partly due to need of much CPU power and memory – only recently available/affordable.

***In the following we elaborate on the following GPMCSes:***

- ***VEGAS algorithm***
- ***Cellular algorithm of FOAM***

## References on GPMCSes

## VEGAS and the like

- G.P. Lepage, J. Comput. Phys. **27**, 195 (1978).
- T. Ohl, Vegas revisited: Adaptive Monte Carlo integration beyond factorization, Comput.Phys.Commun. **120** (1999)13, eprint: hep-ph/9806432.
- S. Kawabata, Comp. Phys. Commun. **88**, 309 (1995).

## Cellular:

- Earlier unpublished trials in 80's by S. Kawabata, R. Kleiss and S. Jadach.
- G. I. Manankova, A. F. Tatarchenko, and F. V. Tkachev, MILXy way: How much better than VEGAS can one integrate in many dimensions?, 1995, a Contribution to AINHEP-95, Pisa, Italy, Apr 3-8, 1995 (extended version).
- E. de Doncker and A. Gupta, "Multivariate Integration on Hypercubic and Mesh Networks", Parallel Computing 24 (1998) 1223.
- S. Jadach, Comput.Phys.Commun. **130**, 244 (2000); and  
"Foam: A general purpose cellular Monte Carlo event generator" e-Print: physics/0203033.

**VEGAS algorithm**

The integrand function in  $n$  dimensions is assumed to be fairly well approximated by a product of functions,  $\rho(x) = \prod_{i=1}^n \rho_i(x_i)$

each one depending just on one integration variable.

The integration range of each variable is divided into  $k_i$  bins of unequal width, with the binning (bin sizes) different for each variable.

The entire integration domain, that is an  $n$ -dimensional rectangle, is divided into  $k^n$  sub-rectangles.

The whole structure is explored by means of the MC generation of random points within each sub-rectangle, with a uniform distribution. Exploration repeated iteratively.

The binning is adjusted iteratively, such that the the ratio of the dispersion to the average weight is minimized – variance reduction.

The “driving function” which controls binning in each direction is the density  $\frac{\rho(x)^2}{p(x)}$ . The grid evolves iteratively such that projection of this function on each direction is as flat as possible. In each iteration bins are first subdivided, integral and projections onto each axis of  $\frac{\rho(x)^2}{p(x)}$  are calculated and the new binning is established.

### Peculiarities of VEGAS

- No published official version of the source code. You may get one from Peter Lepage. However, it is not exactly the same which produced tables in the article.
- Better check that you have got the right code. (Not spoiled by someone).
- To understand the algorithm one has to read carefully article and “decompile the code”. (FORTRAN IV, almost assembler by today’s standard).
- Important feature: oscillations of the binning, damping adjusted manually.
- Most of users treat it as a black box, of course.

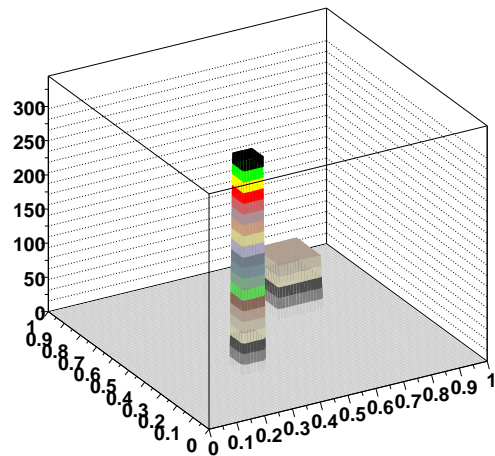
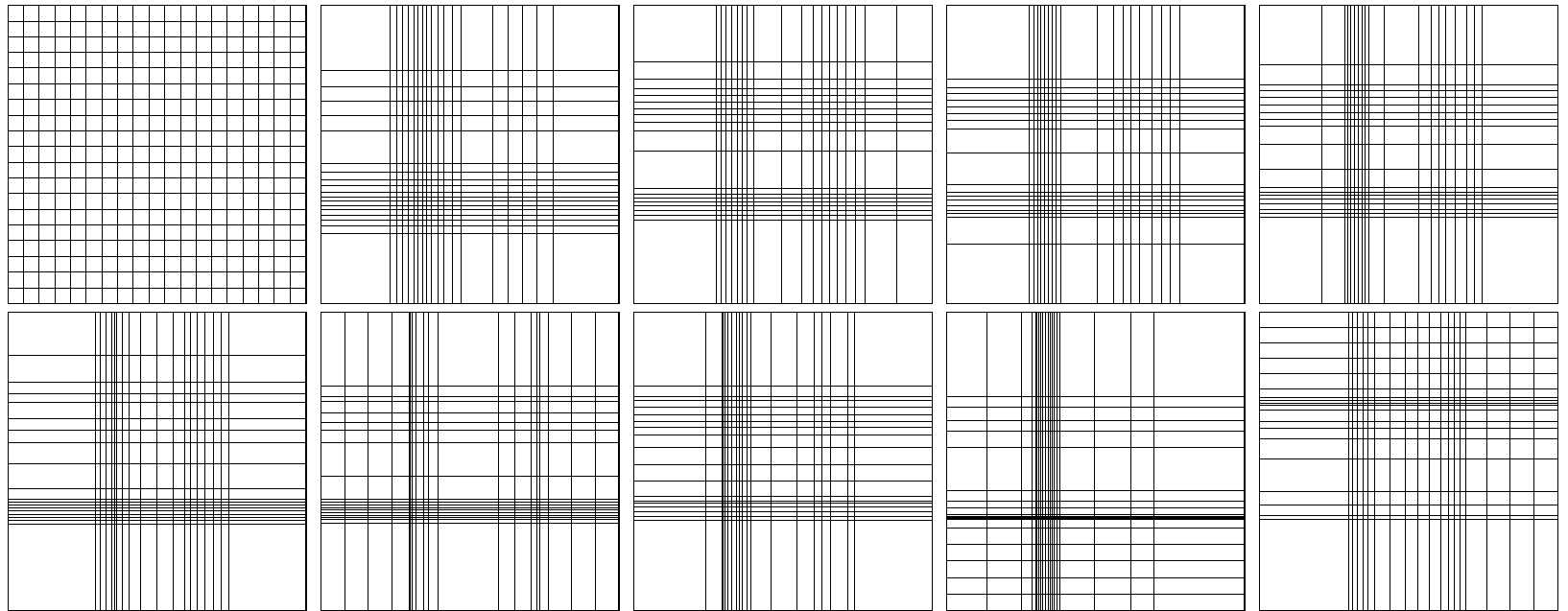
### VEGAS customization

VEGAS is designed as MC integrator. It can be turned into MC generator relatively easily. Remember about binning oscillations!

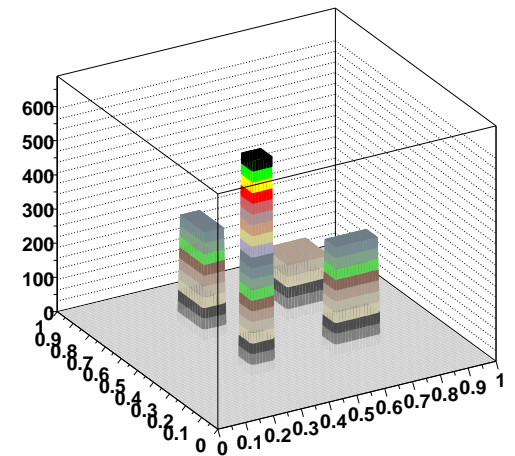
### VEGAS basic deficiency

For a given  $\rho(\vec{x})$  there is certain limiting value of the MC Simulation efficiency  $\frac{\langle W \rangle}{W_{\max}}$  which cannot be overcome by further enlarging the grid and/or number of the MC trials. This is due to factorizability requirement.

VEGAS grid oscillations, iterations 1-10:



True distr.



Vegas Approx.

## General features of General Purpose Monte Carlo Simulators (GPMCS)

Inevitably the GPMCS has to work in 2 stages: **exploration** and **generation**.

During **exploration** GPMCS is “digesting” the entire shape of the  $n$ -dimensional distribution  $\rho(x_1, x_2, \dots, x_n)$  to be generated and memorize it as efficiently as possible using all CPU power and memory available.

Obviously, for the memorized  $\rho'(x_1, x_2, \dots, x_n)$  a method of the MC generation of the points  $\vec{x}$  **exactly** according to  $\rho'(\vec{x})$ , has to be available.

The quality of the distribution of the weight  $w = \rho/\rho'$  for events in the **generation** (small variance, good ration of maximum to average, etc.) is determined by the algorithm of the **exploration**. In other words, the “target weight distribution” in the **generation** is determining the algorithm of **exploration**.

The GPMCS programs will be always limited to “small dimensions”.

With presently available computers “small” means in practice  $n < 10$  (up to  $n < 15$  for certain function). This is already not so bad!!!

### Cellular exploration of the distribution

The most obvious method to minimize the variance (or maximum weight) of the target weight distribution in generation (proposed already some 40 years ago) is to split integration domain into many cells, such that  $\rho(\vec{x})$  is approximated by constant  $\rho'(\vec{x})$  within each cell. This is a **cellular class** of general purpose MC algorithms.

(I think that “stratified sampling”, used in the literature, has a narrower meaning.)

### FOAM: shape of cells, how to cover space with cells?

Three shapes of cells are used pure **simplices**, pure **hyperrectangles** and **Cartesian products of them**. They can be rather easily and efficiently parametrized.

The system of cells can be created all at once (like in Vegas) or in a more evolutionary way, by the “split process”. In the Foam algorithm we rely on the **binary split** of cells. )Choice of a cell to be split driven by target weight distribution.)

The binary split assures automatically **full coverage** of the space, simply because the primary “root cell” is the entire integration domain.

### Variance reduction versus maximum weight reduction

In construction of the FOAM algorithm I have put most effort on the minimization of the ratio of the maximum weight to the average weight  $w_{\max}/\langle w \rangle$ .

This parameter is essential, if we want to transform  $w$ -ted events into  $w = 1$  events, at the latter stage of the MC generation.

Minimizing maximum weight is not the same as minimizing variance

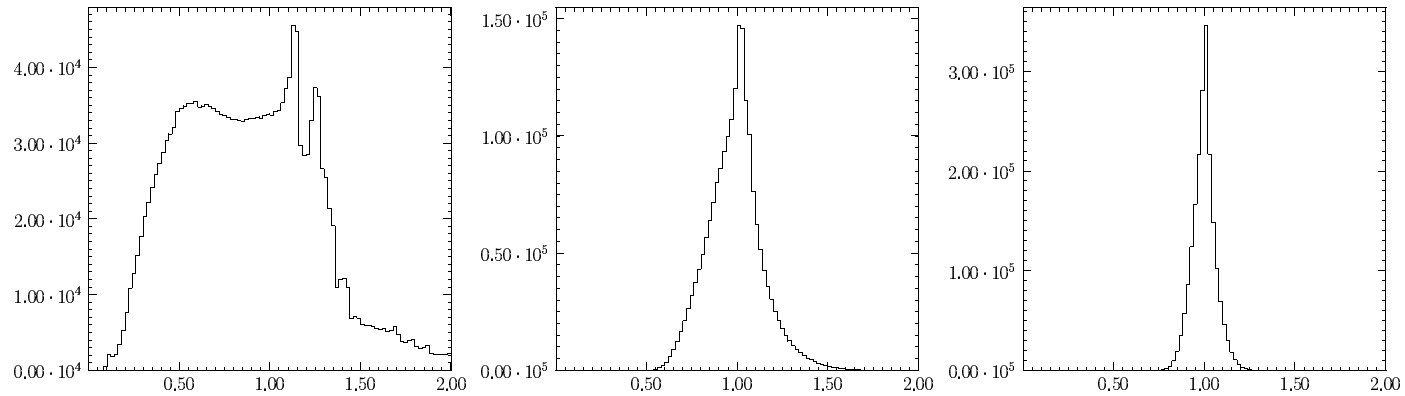
$\sigma = \sqrt{\langle w^2 \rangle - \langle w \rangle^2}$ . Usually minimizing  $w_{\max}$  is more difficult.

In FOAM minimizing variance is also implemented and optionally available.

It can be useful if case them  $w$ -ted events are acceptable.

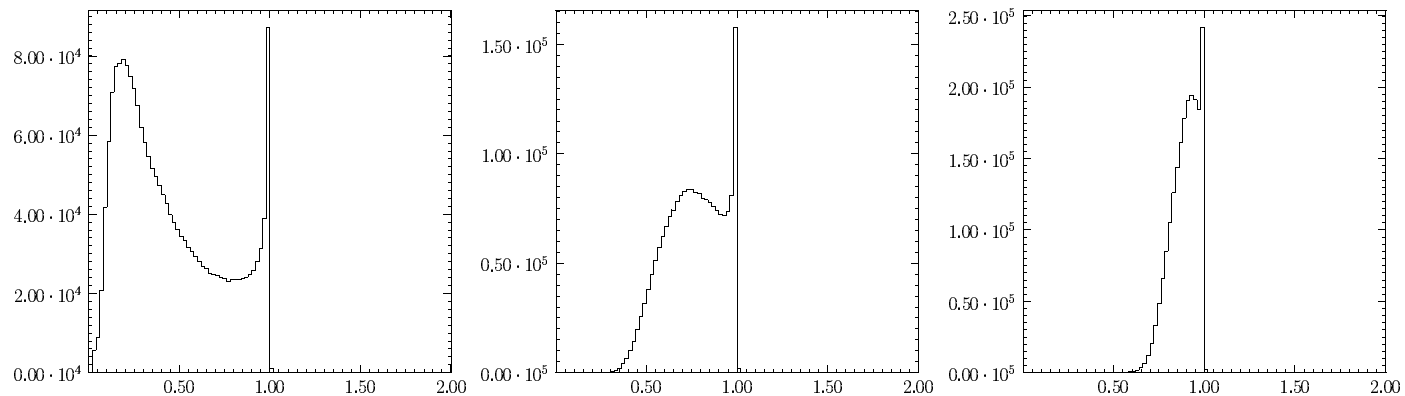
Next slide shows two examples of the weight distribution evolution in the Foam, when adding more and more cells.

Generation weight distribution: **minimization of variance**



Number of cells: 200, 2k, 20k

Generation weight distribution: **minimization of maximum weight**



Number of cells: 200, 2k, 20k

**Cell split algorithm: covers both (a)  $w_{\max}$  and (b) variance minimization**

We define two auxiliary distributions  $\rho'(x)$  and  $\rho_{loss}(x)$  related to integrand  $\rho(x)$ .

Both are constructed together with the foam of cells, in the exploration process.

**(1) EXPLORATION of  $\rho(x)$  and BUILD-UP of Foam of cells :**

$\rho_{loss}(x)$  and  $\rho'(x)$  are evolving in the process of the division of cells.

$R_{loss} = \int \rho_{loss} d^n x$  is minimized in the same process.

**(2) MC event generation:**

Events are generated according to  $\rho'(x)$ .  $R' = \int \rho' d^n x$  is known exactly.

$R = R' \langle w \rangle'$  where  $w = \rho/\rho'$ . The average  $\langle \dots \rangle'$  is over events generated according to  $\rho'$ .

**(a) Minimization of  $w_{\max}$**

$\rho'(x) \equiv \max_{y \in Cell_i} \rho(y)$ , for  $x \in Cell_i$ , the “ceiling function”.

$$R_{loss} = \int d^n x [\rho'(x) - \rho(x)] = \int d^n x \rho_{loss}(x),$$

Note that rejection rate in final MC run =  $R_{loss}/R$ .

**(b) Minimization of variance**

$\rho'(x) \equiv \sqrt{\langle \rho^2 \rangle_i}$ , for  $x \in Cell_i$ . The average  $\langle \dots \rangle_i$  is over  $i$ -th Cell assuming flat distribution.

$\rho_{loss}(x) \equiv \sqrt{\langle \rho^2 \rangle_i} - \langle \rho \rangle_i$ , for  $x \in Cell_i$ . Final MC variance is just  $\simeq R_{loss}$ .

### Two Rules governing binary split of a cell

**Each split of a Cell:**  $\omega \rightarrow \omega' + \omega''$  **should decrease total**  $R_{loss}$ .

$$R_{loss}^{\omega'} + R_{loss}^{\omega''} \ll R_{loss}^{\omega}$$

**How to get the best total decrease**  $\Delta R_{loss}$ ?

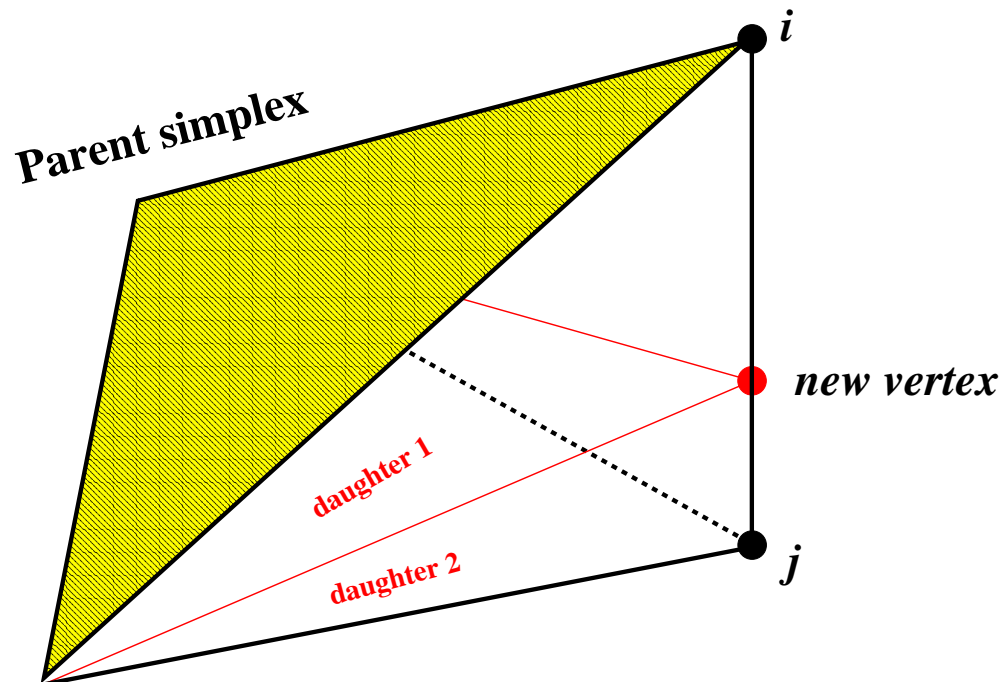
- [1] For each next cell split we choose a cell with the biggest  $R_{loss}$ .
- [2] Position/direction of a plane dividing a parent cell into two daughter cells is chosen to get the smallest total  $R_{loss}$ .

**How do we split a cell into two daughter cells?**

General method relies on the small MC exercise on which events are generated with flat distribution, weighted with  $\rho$  and projected onto  $n$  (simplicial case) or  $n(n+1)/2$  (h-rectangular case) of the cells.

Resulting histograms are analysed and the best “division geometry” found, for which the estimate of  $\Delta R_{loss}$  is calculated. See next slides...

### Geometry of $n$ -dim. Simplicial Cell division, 3-Dim. case



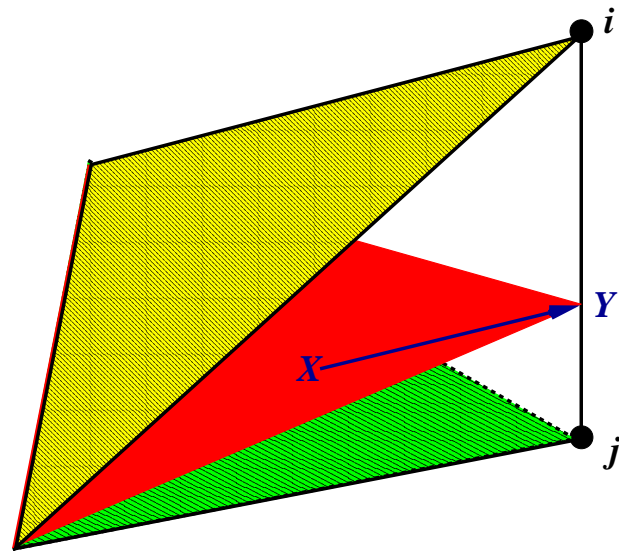
Pair of vertices  $x_i$  and  $x_j$  is chosen and a new vertex  $Y$  is put somewhere on the line in between:  $Y = \lambda x_i + (1 - \lambda)x_j$ ,  $0 < \lambda < 1$

Two daughter simplices are defined with the list of vertices:

$$(x_1, x_2, \dots, x_{i-1}, Y, x_{i+1}, \dots, x_{j-1}, x_j, x_{j+1}, \dots, x_n, x_{n+1}),$$

$$(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_{j-1}, Y, x_{j+1}, \dots, x_n, x_{n+1}).$$

### Geometry of $n$ -dim. Simplicial Cell division, 3-Dim. case



How do we choose  $(i, j)$  pair and the value of  $\lambda$ ?

Short sample of the MC events (100-1000) is generated  $\in$  cell.

Each MC point projected  $X \rightarrow Y$  onto a given edge  $(i, j), i \neq j$ :

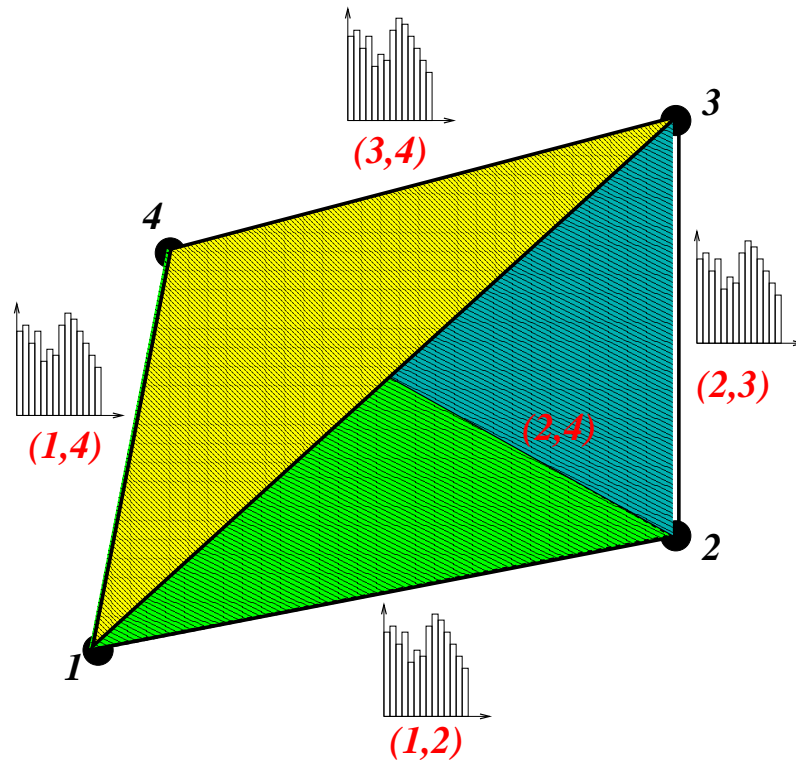
$$Y = \lambda_{ij}x_i + (1 - \lambda_{ij})x_j, \quad \lambda_{ij}(X) = \frac{|\text{Det}_i|}{|\text{Det}_i| + |\text{Det}_j|},$$

$$\text{Det}_i = \text{Det}(r_1, \dots, r_{i-1}, r_{i+1}, \dots, r_n, r_{n+1}),$$

$$\text{Det}_j = \text{Det}(r_1, \dots, r_{j-1}, r_{j+1}, \dots, r_n, r_{n+1}), \quad r_k = x_k - X,$$

where  $\text{Det}(x_1, x_2, \dots, x_n)$  determinant.

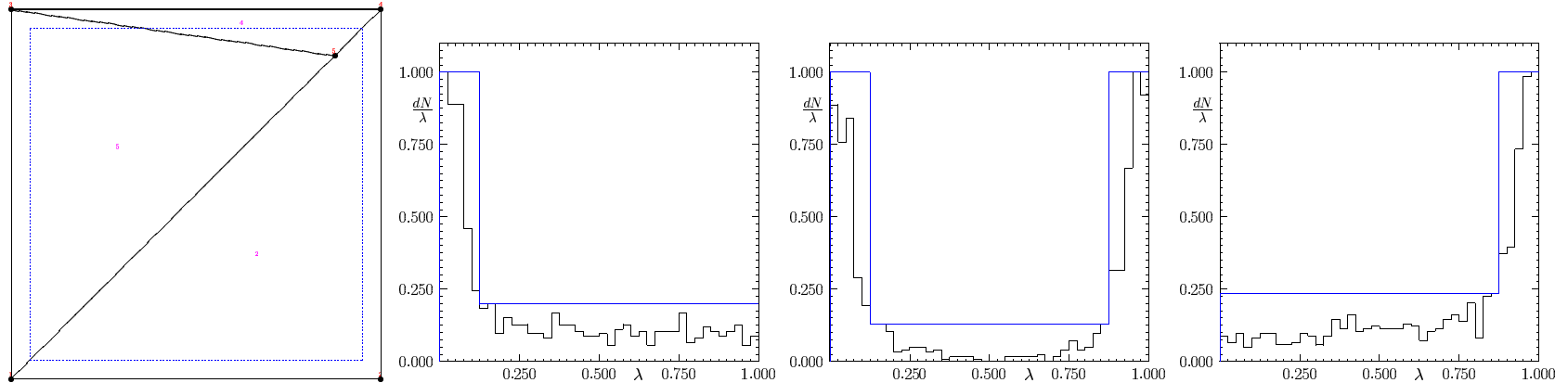
### Choice of the best division edge, Simplicial 3-Dim. case



How do we select  $(i, j)$ ? out of  $\frac{n(n-1)}{2}$  possibilities.

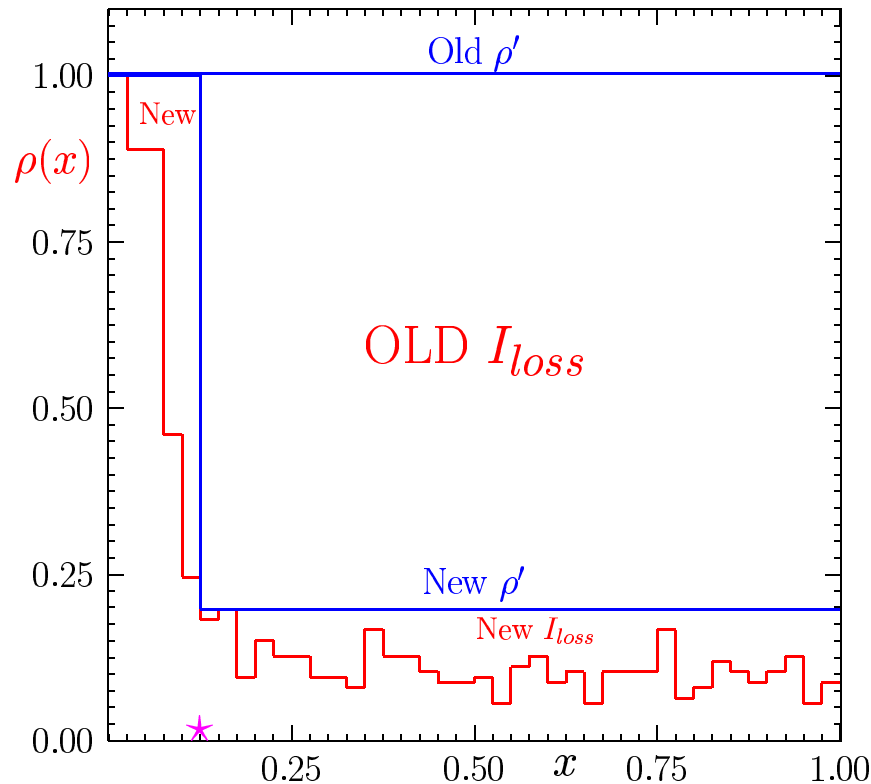
For each  $(i, j)$  the  $dN/d\lambda$  is histogrammed, and its LOSS functional  $R_{loss}$  is estimated. The edge  $(i, j)$  with the biggest LOSS is selected! For the cell division  $\lambda$  is read from the histogram.  $\lambda$  is always a rational number,  $n/N_{bin}$ !

### Binary split 2-dim. example



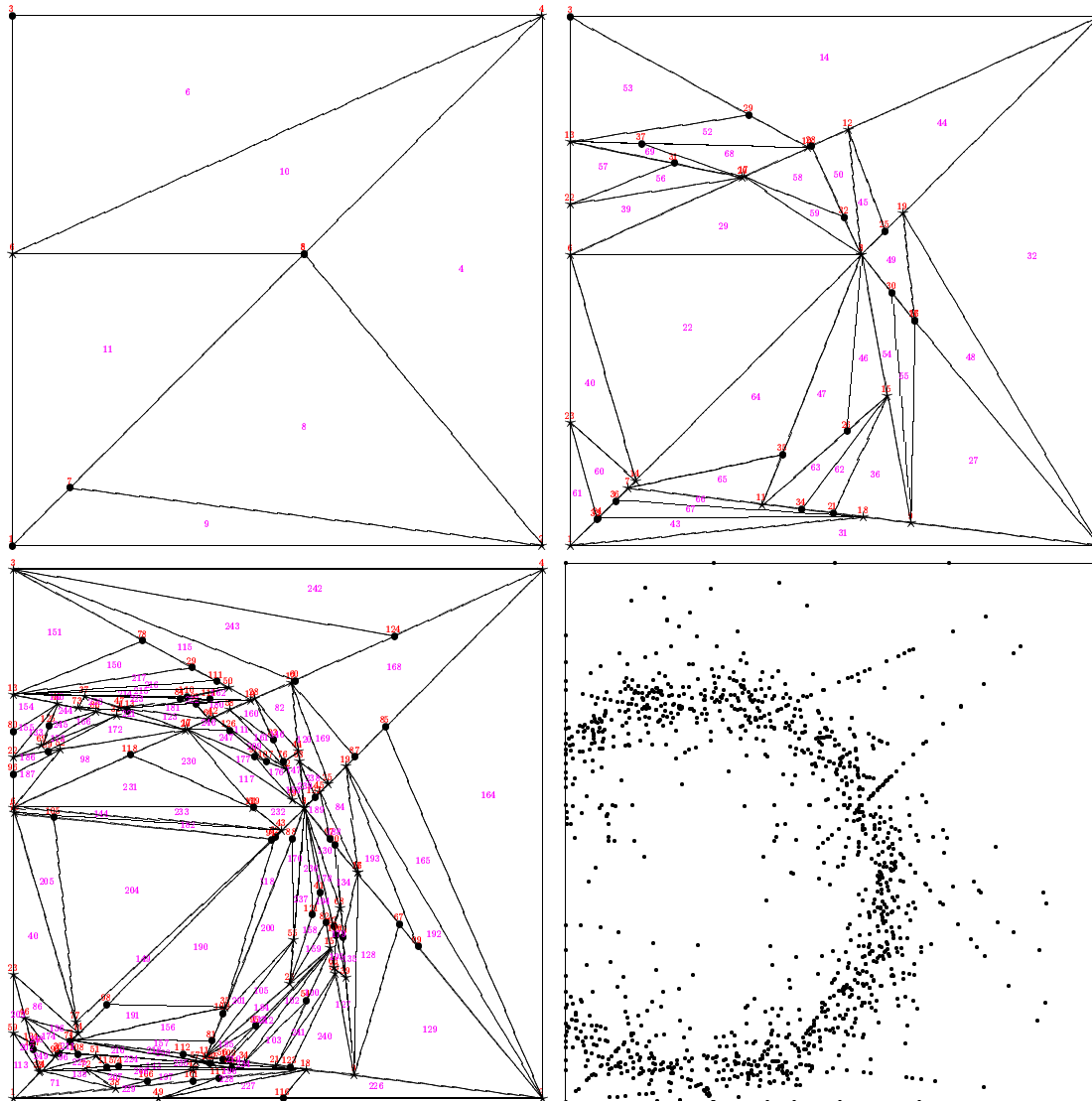
- **Integrand covers narrow strip along edges.**
- **We intend to split upper triangle.**
- **1000 w-ted events are generated and projected onto 3 sides of the parent triangle.**
- **3 Projections are analyzed.**
- **Chosen is the cell with the smallest  $R_{loss}$  (middle plot).**
- **Two resulting daughter triangles are shown at leftmost plot.**

## 2-dimensional example of binary split: projection on one of edges



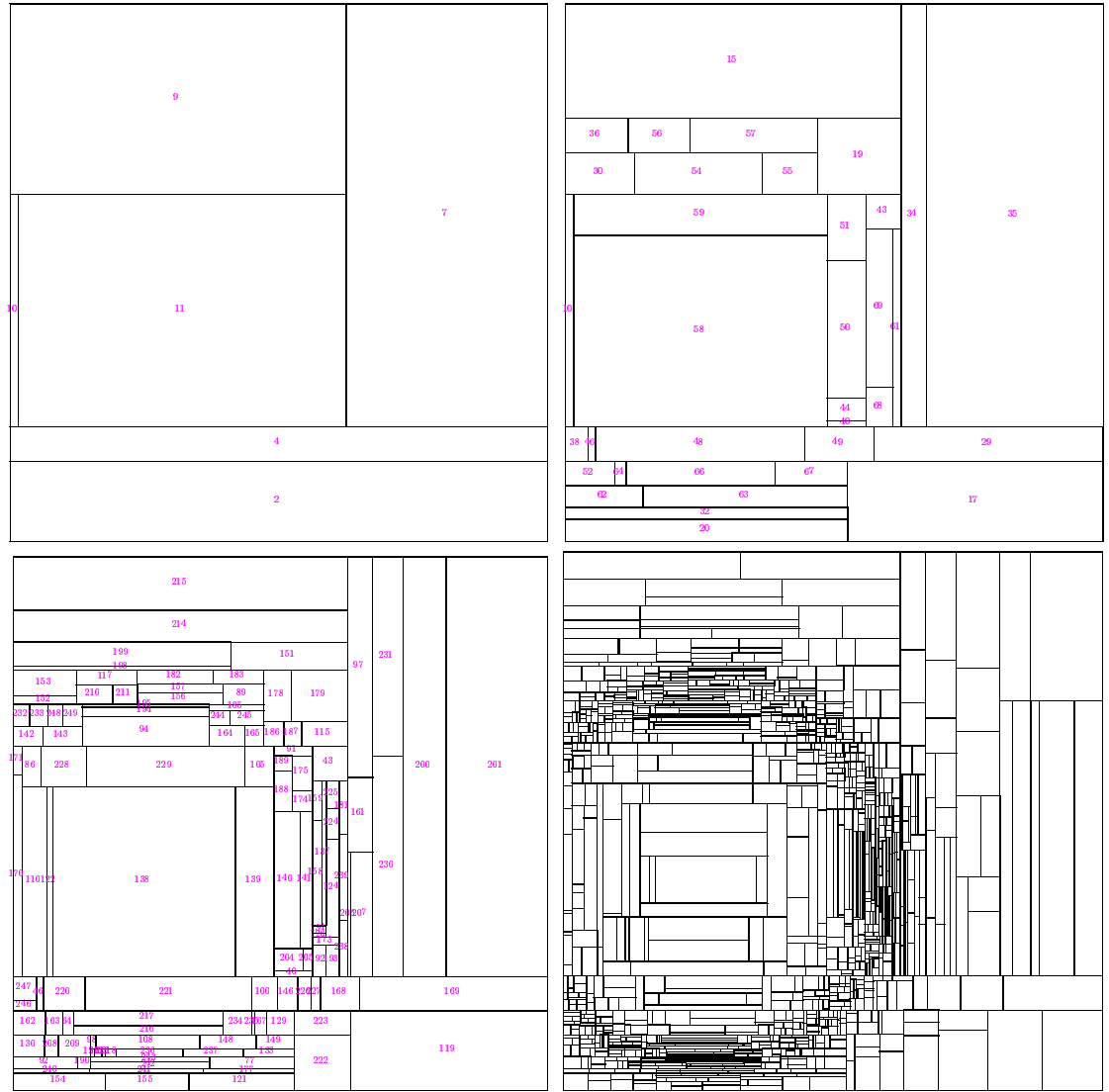
- Projected integrand  $\rho(x)$ .
- Old  $\rho'$  for parent cell (majorizing  $\rho(x)$ ).
- New  $\rho'$  for two daughter cells.
- OLD  $R_{loss}$  all area above red line, for the parent cell.
- New  $R_{loss}$  between red line and New  $\rho'$ , for 2 daughters.
- Obviously  $I'_{New} < I'_{Old}$ , the division point  $\star$  is chosen to MINIMIZE THE LOSS functional/integral  $R_{loss}$ !

**Evolution of simplicial foam at 2-dim.**



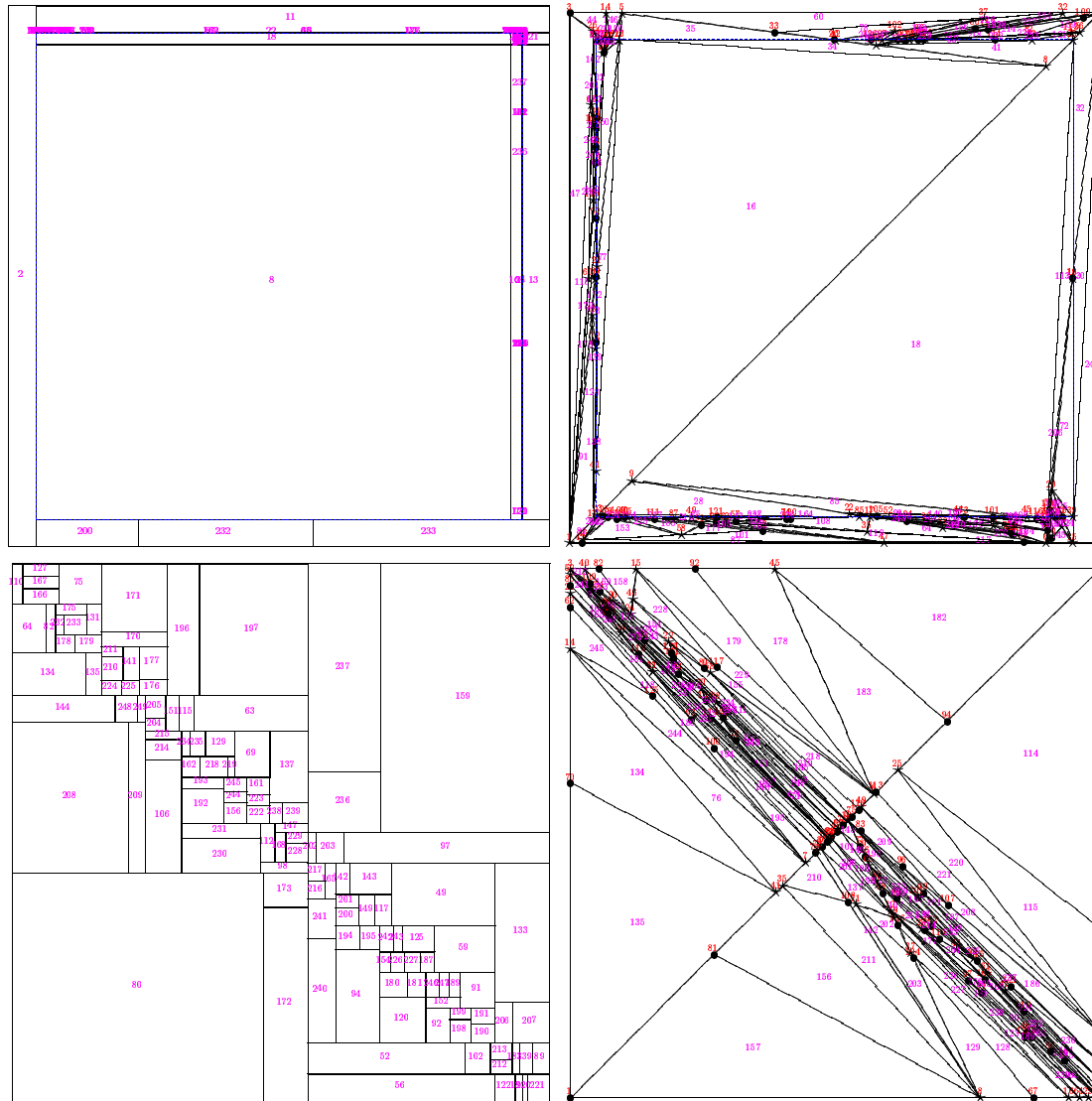
Number of cells= 10, 70,  
250, 2500.

Evolution of hyper-rect. foam at 2-dim.



Number of cells= 10, 70, 250, 2500.

Void and Diagonal at 2-dim.



Number of cells= 250.

### Hyperrectangles or simplices?

- **Simplices are limited to low dimensions  $n < 6$  because of  $n!$ , and because of many determinants heavily consuming CPU time (unless the integration domain is simplex itself).**
- **For simplices memory consumption is  $\sim 16n$  Bytes/Cell, this is a serious limitation.**
- **For hyperrectangles memory consumption is below 50Bytes/Cell independently of  $n$ . This is really great! How it is done? See [http://jadach.home.cern.ch/jadach/Krakow2001\\_generators\\_jadach.ps.gz](http://jadach.home.cern.ch/jadach/Krakow2001_generators_jadach.ps.gz)**
- **Experience with various about 10 testing functions shown that in most cases hyperrectangles provide better final MC efficiency.**

**CPU barrier: one quick fix is found**

Final MC efficiency is improved essentially by the increasing No. of cells  $N_c$ .

CPU time of exploration  $T \sim n \times N_c \times N_{samp}$  where  $N_{samp}$  is the number of MC events used in exploration of each newly created Cell.

**Can we limit  $N_{samp}$  somehow in order to increase  $N_c$ ?**

**SOLUTION:** During MC exploration of a new cell continuously monitor the no. of accumulated effective  $W = 1$  events:  $N_{eff} = \frac{(\sum w_i)^2}{\sum w_i^2}$  and stop when  $N_{eff}/n_{bin} > 25$ , where  $n_{bin}$  is the number of bins in each histogram used to estimate the best division direction/edge and parameter.

The increase of  $N_{samp}$  not wasted for cells in which integrand is varying very little.

### Tests of Foam at low dimensions

Functions at 2-dimens.	Foam 1.01	Simpl.	H-Rect.	VEGAS
$\rho_a(x)$ (diagonal ridge)	0.93	0.93	0.86	0.03
$\rho_b(x)$ (circular ridge)	0.82	0.82	0.82	0.16
$\rho_c(x)$ (edge of square)	0.57	1.00	1.00	0.53
Functions at 3-dimens.	Foam 1.01	Simpl.	H-Rect.	VEGAS
$\rho_a(x)$ (thin diagonal)	0.67	0.74	0.66	0.002
$\rho_b(x)$ (thin sphere)	0.36	0.47	0.53	0.11
$\rho_c(x)$ (surface of cube)	0.37	0.95	1.00	0.30

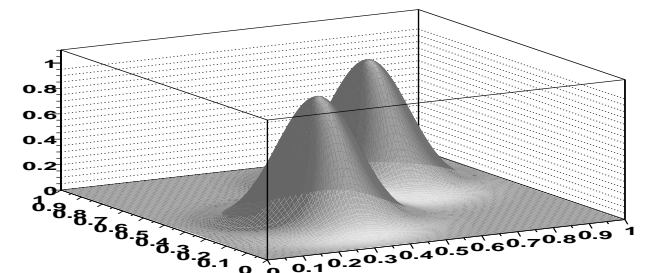
Results from Foam/MCell are for 5000 cells (2500 active cells) and cell exploration based on 200 MC events/cell.

Efficiencies are  $\langle W \rangle / W_{\max}^\varepsilon$  with  $\varepsilon=0.0005$ .

Tests of Foam at higher dimensions, Camel test-function of P. Lepage, normalized to one

nDim	kDim	nCalls	nCells	nSampl	$w_{\max}^e / \langle w \rangle$	$\sigma / \langle w \rangle$	$\Delta_{\text{statist. } R}$	$R$
0	1	206192	1000	1000	0.99147	0.014752	1.043e-05	0.99999962
0	1	206192	1000	10000	0.99147	0.014752	1.043e-05	0.99999962
0	3	435112	1000	1000	0.50886	0.54033	0.000382	1.00017104
0	3	834094	1000	10000	0.50359	0.54674	0.000386	1.00056316
0	3	1015157	1000	33333	0.51091	0.54035	0.000382	0.99983999
0	3	2675820	10000	1000	0.72677	0.27504	0.000194	0.99995080
0	3	3333479	10000	10000	0.72200	0.27720	0.000196	1.00008994
0	3	3575366	10000	33333	0.72243	0.27786	0.000196	0.99997875
0	4	3825046	10000	1000	0.50363	0.51168	0.000361	1.00013082
0	4	6559430	10000	10000	0.50297	0.51001	0.000360	0.99960319
2	2	4493961	10000	1000	0.43076	0.63185	0.000446	1.00072564
2	2	9374351	10000	10000	0.44922	0.60669	0.000429	1.00013171
4	0	6642202	10000	1000	0.21029	1.19420	0.000844	1.00072248
4	0	12337748	10000	10000	0.20817	1.20067	0.000849	1.00020405
0	6	2311881	1000	3333	0.04199	2.12091	0.001499	0.99856206
0	6	12844256	1000	33333	0.03279	2.61028	0.001845	0.99799089
0	6	12737314	10000	3333	0.15385	1.15211	0.000814	1.00039754
0	6	42827237	10000	33333	0.14168	1.22627	0.000867	0.99954178
0	6	42808972	100000	1000	0.30910	0.71250	0.000503	0.99972833
0	6	92531875	100000	10000	0.30905	0.71423	0.000505	0.99985093
0	9	78325890	100000	1000	0.03718	1.64608	0.001163	0.99367339
0	9	353943409	100000	10000	0.05196	1.80538	0.001276	1.00196909
0	9	272162624	400000	1000	0.08490	1.30193	0.000920	1.00065580
0	9	924011087	400000	10000	0.08853	1.38579	0.000979	1.00052122
0	12	261911066	100000	3333	0	5.83954	0.004129	0.97304842
0	12	671460574	100000	10000	0.00640	3.85823	0.002728	0.98878698
0	12	913072065	400000	3333	0.01285	2.73991	0.001937	0.98688299
0	12	2117963809	400000	10000	0.01235	2.92642	0.002069	0.99301117

Efficiency depends mainly on number of cells nCells  
 Number of MC trials per Cell cannot be too small.



**Conclusions**

- **Combining elementary MC method can be organized and documented in a systematic way.**
- **VEGAS is alive and well.**
- **Cellular algorithms like Foam come to state of art.**