

Stochastic simulation (Monte Carlo) Techniques – Basic Concepts and Methods

S. JADACH

IFJ-PAN, Kraków, Poland

Partly supported by Polish Government grant
Narodowe Centrum Nauki DEC-2011/03/B/ST2/02632

Presented at LHCpheno School, Krakow, PL
September, 2013

More material on <http://jadach.web.cern.ch/>

Abstract (plan of the talk)

I shall talk on elementary methods of Stochastic Simulation, commonly referred to as Monte Carlo (MC) techniques,

for simulating and/or integrating many-dimensional distributions, using random numbers:

- pointing out, that all known SS/MC techniques are a superposition of a small subset of 3-4 elementary methods,
- reviewing also briefly “general purpose” SS/MC tools like FOAM and VEGAS – their algorithms, strength's and weaknesses.

Several C++ programs used for numerical illustrations.

Also to be used in the practical exercises.

With CERN library **ROOT** for histogramming and graphics.

Dont be afraid of ROOT:))

Booking histogram:

```
double r,z,zmin = -1, zmax = 1; int nb=100;  
TH1D *Hist1= new TH1D("Hist1","p(z)=(3/8) (1+z)^{2}",nb,zmin,zmax);
```

Filling histogram:

```
Hist1->Fill(z ,WtNorm);
```

Drawing histogram on the screen:

```
Hist1->DrawCopy();
```

README file will guide you in the practical exercises:

1. Mapping MC Method

C++ program to be compiled and run in the interpreter mode by ROOT:

```
root ./examp_mapping1.C
```

a) increase no. of MC events and run it again!

b) identify essential part (3 lines) of the MC algorithm

The same C++ program run in the compile/link/run mode either like that

```
make -f Makefile examp_mapping1
```

or using command in top lines of the code (cat examp_mapping1.cxx)

a) increase no. of MC. events

b) play with it! try to invent what may go wrong and modify/spoil accordingly

2. MC integration, illustration of MC statistical error σ/\sqrt{N}

```
make -f Makefile examp_integr1
```

a) increase no. of trials to 100, 1000 etc

3. Study on the convergence of the MC integration method when increasing N

```
make -f Makefile examp_integr2
```

a) change vertical scale to see better convergence at higher N

4. Central Limit Theorem at work!

What is Monte Carlo, stochastic simulation/integration method?

In a nutshell...

In a nutshell, it is numerical method of obtaining *EXACT* numerical results involving integration/averaging in many dimensions ($n > 5$, $n \sim 10^3$, $\sim 10^6$) using *random numbers*.

These problems are **INSOLVABLE (!)** with any other numerical methods like quadrature (Gauss integration) or analytical methods.

Extremely wide variety of practical applications!

Mostly Markovian type...

Many “industrial applications” from the very beginning of MC era (Los Alamos) are dealing with some Markovian process:

- Neutron in the volume of the matter bouncing from atoms with some elastic cross section, sometimes absorbed, sometimes leaking outside.
- Photon coming from the sun, bouncing from atmospheric molecules, sometimes reaching earth, getting reflected (or absorbed) and finally reaching to the eye camera of the satellite monitoring ozone levels.
- Time evolution of the thousands of financial assets in tens of interacting stock/money markets with complicated interaction through transactions, insurances, options, external parameters like natural disaster, stock closures at different time zones etc.
- Calculating hadronic bound state masses in QCD on the time-space lattice, by means of the random walk in the configuration space (Metropolis algorithm).
- Parton shower Monte Carlo's for LHC!

MY LIMITED SCOPE:

I shall focus on non-Markovian Monte Carlo methods of simulating or integrating efficiently distribution in a small (usually fixed) number of dimension $n < 100$.

However, the modeling a *single step forward* in the Markovian MC world ($n \sim 10^3, \sim 10^6$), quite often requires solving exactly the same above problem, in $n < 100$ dimensions.

NB. Markovian MC can also be formulated in terms of four elementary method elaborated in the following.
(Acta Phys.Polon.B39:115,2008, <http://arxiv.org/abs/arXiv:0708.1906>)

Prehistory and history of MC/SS

- 19th cent. precursor mathematicians: Buffon, Kelvin, others, but...
- Modern Monte Carlo is a child of Los Alamos atomic bomb project (neutron transport), with many contributors (Fermi, Feynman, Ulam, Von Neuman...)
- N. Metropolis and S. Ulam, “The Monte Carlo method”, J. Amer. Stat. Assoc (1949) 44, **247**, 337-341.
- MC in Particle phys. precursor: I. Kopylov, JETP **35** (1958) 1426.
- G.R. Lynch, FAKE, UCRL-10335 (1962), Berkeley.
- F. James, FOWL, CERLNLIB W-505, 1966-70, CERN.
- C.J. Everet and E.D. Cashwell, “A Monte Carlo Sampler”, Los Alamos internal report, LA-5081-MS, October 1972.
- G. Peter Lepage, “A new Algorithm for adaptive Multidimensional Integration”, Journ. of Comp. Phys. **27** (1978) 192.

- **Stefan Weinzierl**,
“Introduction to Monte Carlo methods” 47 pages (2000)
<http://arxiv.org/abs/hep-ph/0006269>
- **Henryk Czyż**,
“Monte Carlo Methods”, slides of four lectures (2010)
http://czyz.phys.us.edu.pl/czyz/MC_Mainz_1.pdf
- **S. Jadach**,
“Practical Guide to Monte Carlo”, 20 pages (1999)
<http://arxiv.org/abs/physics/9906056>

MC simulation \neq MC integration

MC simulation more difficult than MC integration

**The integral value is always there as a byproduct of the MC simulation,
hence MC int. \in MC sim.**

MC Simulation versus MC Integration

What is Monte Carlo simulation?

Large number (list) of the **MC events** $x = (x_1, x_2, \dots, x_n)$ is “fabricated” randomly, independently, **exactly** according to predefined probability distribution $p(x)$.

Generated MC events are stored or used to calculate all kind of averages and distributions.

For example we may obtain 1-dim. distribution of an “**observable**” $G(x_1, x_2, \dots, x_n)$:

$$\frac{dp}{dg} = \int d^n x \, \delta(g - G(x_1, x_2, \dots, x_n)) p(x).$$

This leads to 1-dim. “**histogram**” with, say, 100 bins.

MC Simulation versus MC Integration

What is Monte Carlo integration?

The aim is to calculate a single number,
the integral $R = \int_{\Omega} \rho(x) d^n x$,
exploiting MC events $x_l, l = 1, 2, \dots, N$
generated according to some $p(x)$.

The **integral estimate** is

$$R = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{l=1}^N \frac{\rho(x_l)}{p(x_l)}.$$

The ratio $w(x) = \frac{\rho(x)}{p(x)}$ is the **MC weight** of event x .

For finite N **statistical error** of the integral estimate R is

$$\delta R = \frac{\sigma}{\sqrt{N}} = \frac{\sqrt{\langle (w - \langle w \rangle)^2 \rangle}}{\sqrt{N}},$$

according to **Central Limit Theorem**, see next slide...

(http://en.wikipedia.org/wiki/Central_limit_theorem)

Simple example of MC SIMULATION (mapping)

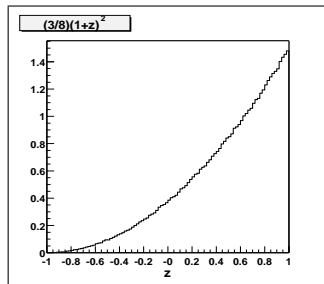
Generating z according to $p(z) = \frac{3}{8}(1+z)^2, z \in [-1, 1]$.

1. Take **cumulative funct.** $r(z) = \int_{-1}^z p(x)dx = (1/8)(1+z)^3$ as an integration variable:

$$\int p(z)dz = \int_0^1 dr \frac{dx}{dr} p(x(r)) = \int_0^1 dr \frac{1}{\frac{dr}{dx}} p(x(u)) = \int_0^1 dr 1.$$

2. Take uniformly distributed $r \in [0, 1]$ from any RN generator.

3. Map $r \rightarrow z$ using **inverse** of $r(z)$: $z = -1 + (8r)^{1/3}$. Et voilà!



```
Double_t RndCthe(TRandom *RNgen ){  
  // p(z)=(3/8)(1+z)^2 distribution  
  Double_t r = RNgen->Rndm(0);  
  Double_t z = -1.0+pow( 8.0*r, 1.0/3.0 );  
  return z;  
}
```

Simple example of MC SIMULATION (mapping)

Complete program to be run with: "root examp_mapping1.C"

```
/// root examp_mapping1.C
/// Mapping method to generate  $(1+z)^2$  distribution in  $(-1,1)$  range.
{ gROOT->Reset();
  /// booking Histogram
  double r,z,zmin = -1, zmax = 1; int nb=100;
  TH1D *Hist1 = new TH1D("Hist1","p(z)=(3/8)(1+z)^{2} Mapping method",nb,zmin,zmax);
  Hist1->GetXaxis()->SetTitle("z");
  /// random number generator object
  TRandom3 *RNg3 = new TRandom3();
  int nevt=100000;
  double WtNorm= (1.0/nevt)/((zmax-zmin)/nb); //  $dP/dx = (dN/N) / (dx)$ 
  /// loop over MC events
  for(int iev=0; iev<nevt; iev++){
    r = RNg3->Rndm(0);
    z = -1.0+pow( 8.0*r, 1.0/3.0 );
    Hist1->Fill(z ,WtNorm);
  }
  /// drawing histogram on screen
  TCanvas *cGener1 = new TCanvas("cGener1");
  cGener1->Draw();
  cGener1->cd();
  Hist1->SetStats(0);
  Hist1->DrawCopy();
}
```

Another version of the same: examp_mapping1.cxx

```
/// g++ -O -I/usr/include/root/ -c examp_mapping1.cxx
/// g++ -O examp_mapping1.o -L/usr/lib64/root -lGraf3d -o examp_mapping1.exe
/// ./examp_mapping1.exe
#include <iomanip>
#include <math.h>
#include "TCanvas.h"
#include "TH2.h"
#include "TRandom3.h"
#include "TApplication.h"
Double_t RndCthe(TRandom *RNGen ){
    /// generating  $p(z)=(3/8)(1+z)^2$  distribution
    Double_t r = RNGen->Rndm(0);
    Double_t z = -1.0+pow( 8.0*r, 1.0/3.0 );
    return z;
}
void test2(){
    Long_t nevt;
    nevt=1000;
    TRandom3 *RNg3 = new TRandom3();
    Double_t x;
    Double_t xmin = -1;
    Double_t xmax = 1;
    Int_t nb=10;
    TH1D *Hist2 = new TH1D("Hist2", "(3/8)(1+z)^2", nb, xmin, xmax);
    Hist2->Sumw2();
    Hist2->GetXaxis()->SetTitle("z");
    Hist2->GetXaxis()->CenterTitle();
    Hist2->GetXaxis()->SetTitleSize(0.05);
    Double_t WtNorm= (1.0/nevt)/((xmax-xmin)/nb); //dP/dx=dN/N/dx
    for(int iev=0; iev<nevt; iev++){
        x = RndCthe(RNg3);
        Hist2->Fill(x ,WtNorm);
    }
    Int_t WidPix, HeiPix;
    WidPix = 500; HeiPix = 500;
    TCanvas *cGener2 =
        new TCanvas("cGener2", "Gener2",420,70,WidPix,HeiPix);
    cGener2->SetFillColor(10);
    cGener2->Draw();
    cGener2->cd();
    Hist2->Draw("h");
}
int main(int argc, char **argv){
    TApplication theApp("theApp", &argc, argv);
    test2();
}
```

Numerical illustration of "random error" in MC integration

Simple example code: `examp_integr1.cxx`

$$\rho(z) = (1+z)^2 \theta(1-z) \theta(1+z)$$

$$p(z) = 1/2$$

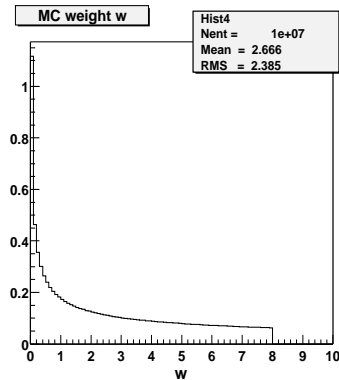
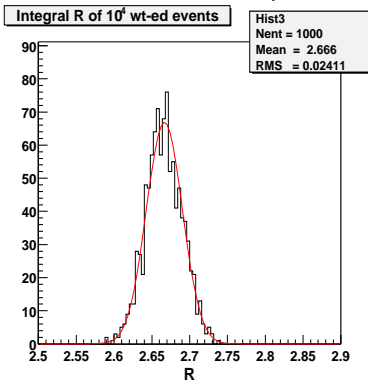
$$w = \frac{\rho(z)}{p(z)} = 2\rho(z)$$

Known integral $R = \int \rho(z) dz = 8/3 = 2.6666\dots$ calculated 1000

times using MC method: $R \simeq \langle w \rangle = \frac{1}{N} \sum_{l=1}^N w(z_l)$.

Each of 1000 MC runs with $N = 10^4$ MC events.

Statistical fluctuations/spread of R is examined:



Simple example code: examp_integr1.cxx

```
void GenCthe(TRandom *RNg3, Double_t &z, Double_t &w){
// p(z)=(3/8)*(1+z)^2 distribution, weighted events
  Double_t r = RNg3->Rndm(0);
  z = -1.0+2.0*r;
  Double_t p = 0.5;
  Double_t rho = (1+z)*(1+z);
  w = rho/p;
}
void test3(){
  Long_t nevt =10000, ntrial=1000;
  TRandom3 *RNg3 = new TRandom3();
  Double_t R, Rmin = 2.5, Rmax = 2.9, Wmax = 10.0;
  Int_t nb=100;
  TH1D *Hist3 = new TH1D("Hist3","Integral R for 10^{4} wt-ed events",nb,Rmin,Rmax);
  TH1D *Hist4 = new TH1D("Hist4","MC weight w", nb,0.0,Wmax);
  ...
  Double_t WtNorm= (1.0/nevt/ntrial)/((Wmax)/nb); // dP/dx=(dN/N)/(dx)
  Double_t z,w;
  for(int itr=0; itr<ntrial; itr++){
    Double_t sum = 0;
    for(int iev=0; iev<nevt; iev++){
      GenCthe(RNg3,z,w);
      sum += w;
      Hist4->Fill(w,WtNorm);
    }
    R = sum/nevt;
    Hist3->Fill(R,1.0);
  }
  TF1 *fGaus3=new TF1("fGaus3", "[1]/sqrt(2.0*pi)/[0]*exp(-sq((x-[2])/[0])/2)",Rmin,Rmax);
  fGaus3->SetParameter(1,ntrial*((Rmax-Rmin)/nb));
  fGaus3->SetParameter(2, 2.666666);
  Int_t WidPix = 1000, HeiPix = 500;
  TCanvas *cGener3 = new TCanvas("cGener3","Gener2",440,90, WidPix,HeiPix);
  cGener3->Divide(2, 1, 0.0, 0);
  cGener3->Draw();
  cGener3->cd(1);
  Double_t Ymax = Hist3->GetMaximum(); Hist3->SetMaximum(Ymax*1.2);
  Hist3->Draw();
  /// Gauss funtion on top of histogram
  Double_t sigma = 2.385*sqrt(1.0/nevt);
  fGaus3->SetParameter(0,sigma);
  fGaus3->DrawCopy("same");
  cGener3->cd(2);
  Hist4->Draw("h");
}
int main(int argc, char **argv){
  TApplication theApp("theApp", &argc, argv);
  test3();
}
```

Convergence study in MC integration

Example code: `examp_integr2.cxx`

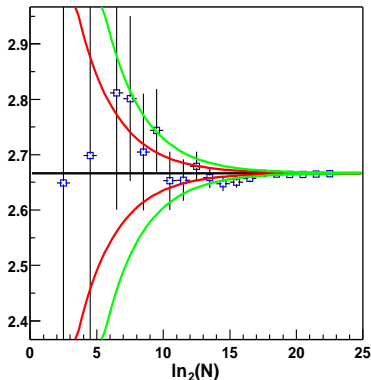
MC error of integral estimator falls slowly, $\delta R_N = \frac{\sigma}{\sqrt{N}}$, but firmly!

Contrary to other integration methods (Gauss, adaptive, etc.),

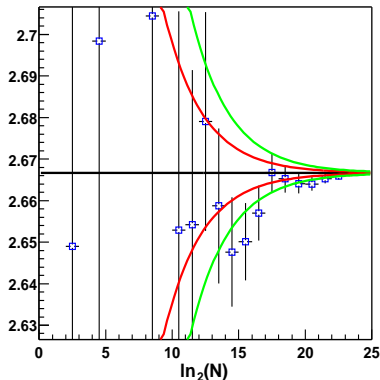
error estimate in MC is extremely **RELIABLE** and **STABLE**!

Green band: probability $|R - R_N| < 2\delta R_N$ is 96%.

Convergence of Integral R for 10^N events



Convergence of Integral R for 10^N events



Red band: $\pm\delta R_N$, probab. of $|R - R_N| < \delta R_N$ is 68%.

Central Limit Theorem (Law of big numbers)

Example code: `examp_centrl.cxx`

Define $p_N(x) = \int \prod_1^N dx_i p(x_i) \delta(x - \sum_1^N x_i)$, being the sum of independent trials, where distribution $p(x)$ is characterized by the mean μ_1 and variance σ .

Theorem: For $N \rightarrow \infty$, for ANY input distribution $p(x_i)$, with this μ_1 and σ , $p_N(x)$ will always converge to a normal (Gaussian) distribution, with the average $= N\mu_1$ and variance $= \sigma\sqrt{N}$

Convolution

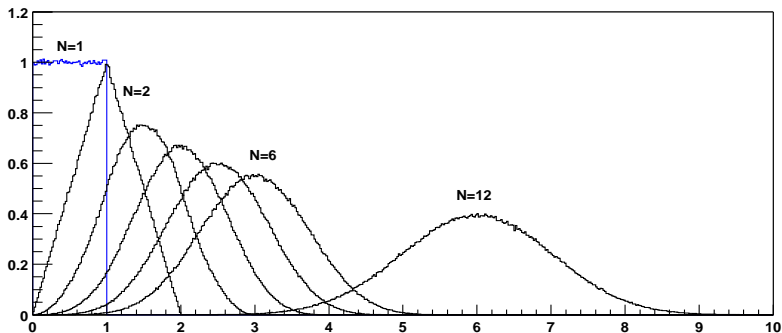
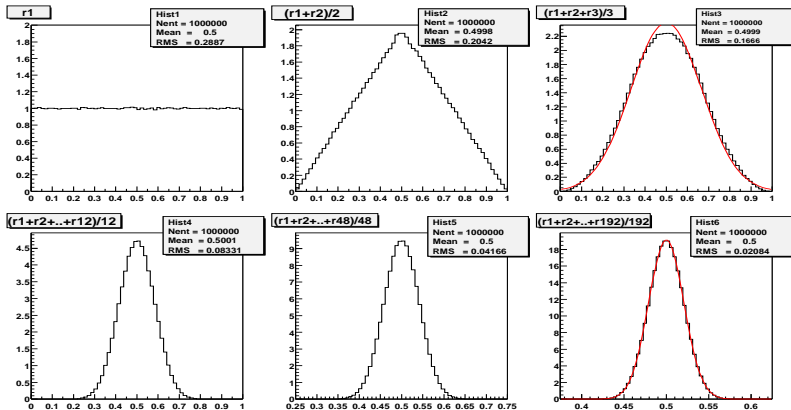


Illustration for $p(x)$ being flat distribution in $(0,1)$ range.

Central Limit Theorem – Numerical illustration

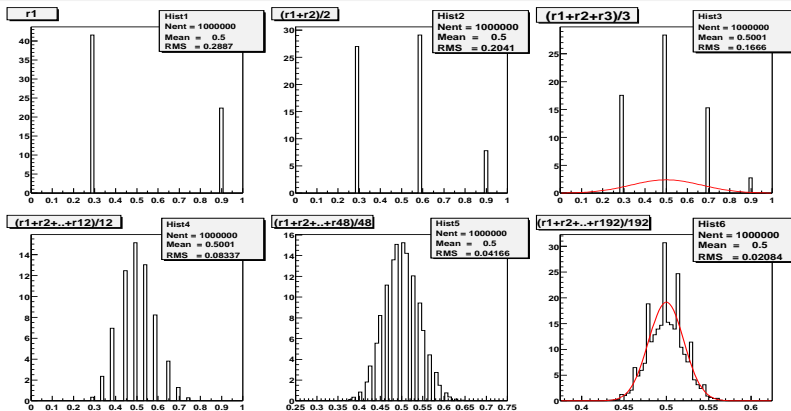
Example code: `examp_cent2.cxx`



$$p_N(r) = \int \prod_1^N dx_i p(r_i) \delta(r - \frac{1}{N}(\sum_1^N r_i))$$
, for $N = 1, 2, 3, 12, 48, 192$,
for continuous uniform distribution: $p(r) = 1$, $r \in [0, 1]$,
 $\mu_1 = \langle r \rangle = \frac{1}{2}$, $\sigma^2 = \frac{1}{12} = 0.08333$, $\sigma = \frac{1}{\sqrt{12}} = 0.288675$

Central Limit Theorem works for ANY $p(x)$!!!

Example code: `examp_centr2.cxx`



$$p_N(r) = \int \prod_1^N dx_i p(r_i) \delta(r - \frac{1}{N}(\sum_1^N r_i)), \text{ for } N = 1, 2, 3, 12, 48, 192,$$

for discrete non-uniform distribution

$$p(r) = q\delta(r - a) + (1 - q)\delta(r - b),$$

with a, b and q adjusted to get the same:

$$\mu_1 = \langle r \rangle = \frac{1}{2}, \quad \sigma^2 = \frac{1}{12} = 0.288675.$$

Pseudo-RN generators: Practical hints

- Never trust, never use RNGens provided by compilers, operating systems etc.
- Always use two RN generators and for final calculation switch among them.
- If possible (speed) use RANLUX of Luscher (1993), because of very good quality and well understood features, now included in ROOT!
- Every 5 years or so there is another interesting new random number generator on the market, so far RANLUX is unbeatable.
- In the present small exercises I am using “Mersenne Twistor” generator of M. Matsumoto and T. Nishimura (1998) with bit shuffling, available in ROOT.

Why elementary MC methods are important?

Citation from LANL e-Print Physics/9906056, by S.J:

*“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.”*

Hence in the following we describe THREE elementary MC methods:

1. **Rejection** according to weight
 2. **Branching** (multichannel method)
 3. **Mapping** of variables,
- and how they can be **combined**.

REJECTION METHOD

Rejection MC 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 a primitive distribution $\rho^{(0)}$, easier to simulate than $\rho(x)$.

A MC simulator for primitive $\rho^{(0)}(x)$ provides (weight=1) MC events x . The algorithm for simulating $\rho(x)$ is:

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

The integral is also available: $R = \langle w \rangle_{\rho^{(0)}} R^{(0)}$, where

$R^{(0)} = \int \rho^{(0)}(x) dx^n$ is provided by the underlying MC simulator.

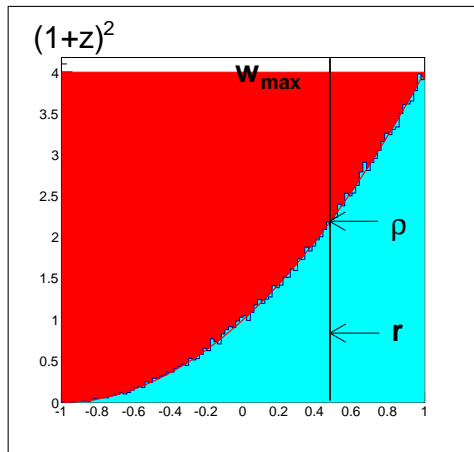
The average $\langle \dots \rangle_{\rho^{(0)}}$ is over ALL events generated according to $\rho^{(0)}$.

MC weight $w = \rho/\rho^{(0)}$ MUST have finite maximum w_{\max} :

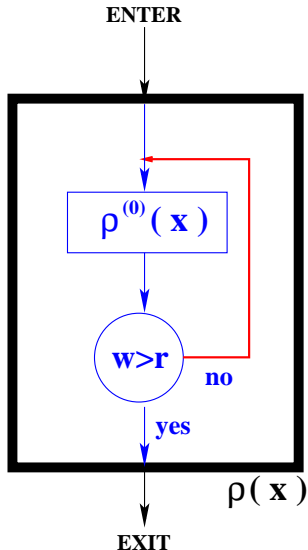
Rejection method: Simple illustration

c++ code: `examp_reject1.cxx`

Intuitively validity of the MC rejection method is obvious:

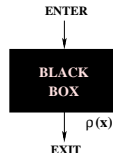


Rejection method: Graphic representation



We need graphic representation in order to visualize: **control flow**, **information flow** and the **algorithm structure**

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



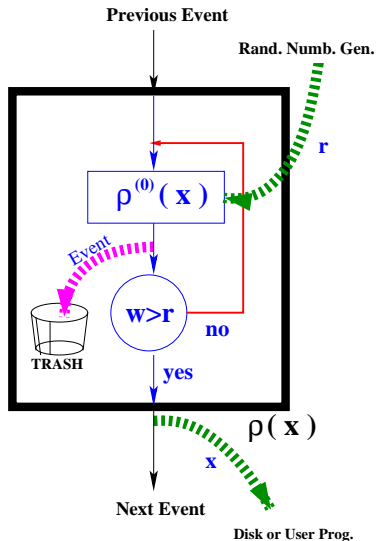
Information flow is not yet properly visualized, see next slide.

Rejection method: Information flow

INFORMATION FLOW

can be added to the graph.

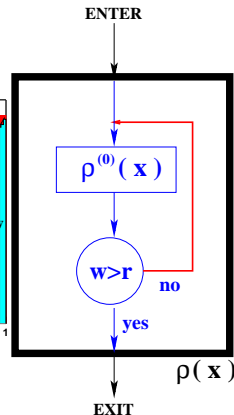
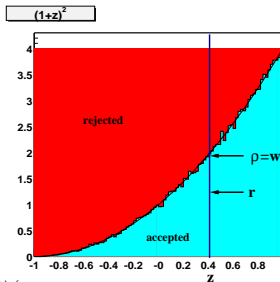
In the rejection method part of information is irreversibly lost (rejected events, RN's used for rejection) for the outside of the black rectangle.



Rejection method: C++ complete example

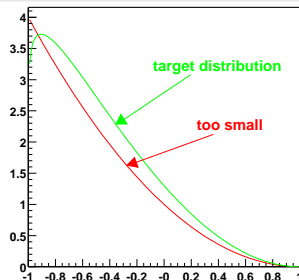
Typical functionality of MC simulator already implemented

```
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);
    }
}
```

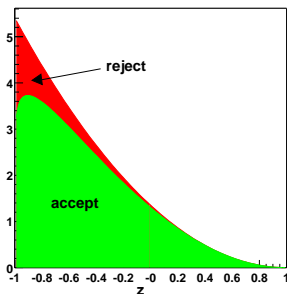


The problem of “maximum weight”

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! 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?

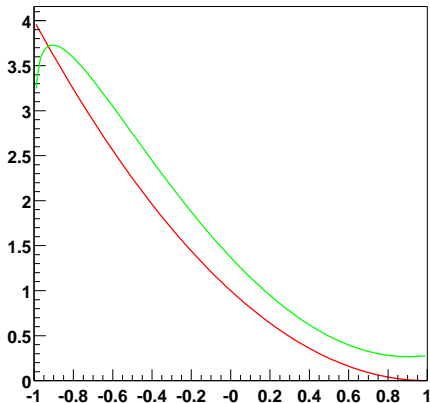


“Blind spots” in the rejection method

Rescaling not always possible!
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 not help!
Reject. weight has “infinite tail”,
average weight doesn't exist.



Still another limitations of the rejection method

Not only zeros, but first of all
narrow spikes are “fatal”
for the rejection method!

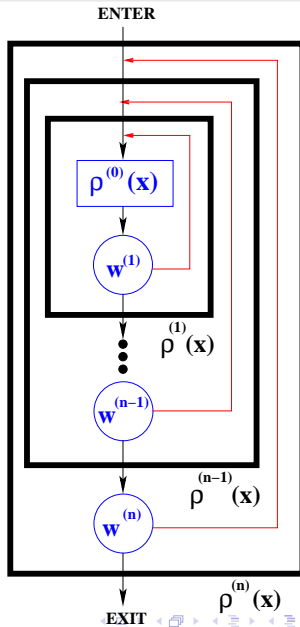
Here rejection can be done but
with **HUGE REJECTION** rate.



Multiple (nested) rejections

Nested rejection often used in order to gain on CPU time and modularity of the code:

- ⊕ 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.



BRANCHING METHOD

Alias Multi-Channel Method

Branching (multi-channel) method

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

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 next generate x according to $\rho(J; x)$, the simpler J -th component distr.

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

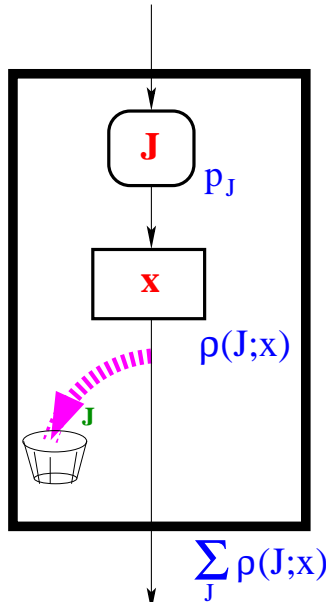
PROFITS: Each component $\rho^J(x)$ can be easier to simulate than the sum.

Better efficiency, smaller variance etc.

LIMITATIONS:

Sub-integrals R_J has to be known in advance!

Way out: Combine with rejection method.

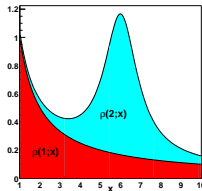
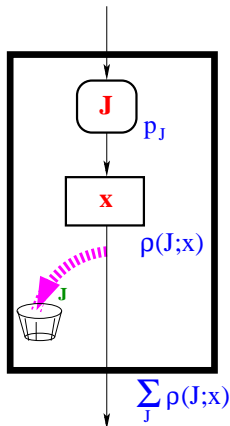


Branching method: Simple example 1

Let us try to simulate:

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

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



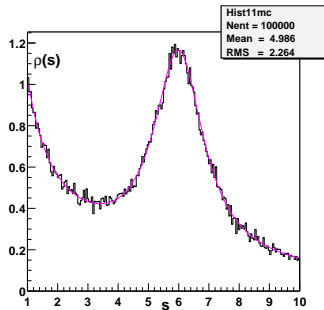
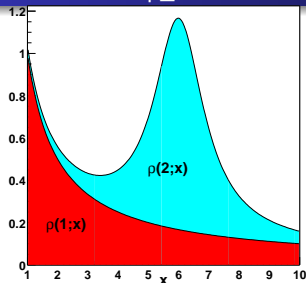
A kind of Breit-Wigner “resonance” (mass= $\sqrt{6}$, width =1) + non-resonant “background”, added incoherently (no interference).

Each of component distributions is analytically integrable, hence 1-dimensional “mapping method” in each branch.

NOTE: 1-dim. mapping provides also analytically the component integrals!

Branching method: complete example and MC result

c++ code: examp_branch2.cxx



```
// Mini simulator/integrator, 2 branches, Breit-Wigner+Ba
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 *RNgen, double &s){
        // generates single event, 2 branches
        Double_t 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);
        }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++;
    }
    void GetIntegral(double &R){
        // Provides total Integral
        R = m_R1+m_R2;
    }
};
```

MAPPING METHOD

In one and many dimensions

Mapping, 1-dim case, 2 examples

A very limited number of 1-dim. distributions $\rho(x)$ can be generated out of uniform random number r using simple “mapping” $x = H(r)$.

Two explicit examples:

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

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

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

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

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

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

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

Inverting cumulative function $F(y) = \int^y \rho(x) dx$

(a) is rarely feasible analytically;

(b) but one can also invert it numerically.

Mapping in 1-dim. c.d.

In 1-dim case only a finite number of the distributions can be generated using mapping within simple 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 simple distributions **cannot be** obtained by analyt. 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 brute 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”.

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

2-dimensional and n-dim. mapping

An example of clever 2-dim. mapping (Gaussian in radial variables):

$$\int dx dy \rho(x, y) = \int dx dy \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} = \\ \int_0^1 d\frac{\phi}{2\pi} \int_0^\infty \frac{1}{2\sigma^2} e^{-r^2/(2\sigma^2)} dr^2 = \int_0^1 d\frac{\phi}{2\pi} \int_1^0 de^{-r^2/(2\sigma^2)} = \int_0^1 dr_1 \int_0^1 dr_2 1.$$

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

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

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

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:

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

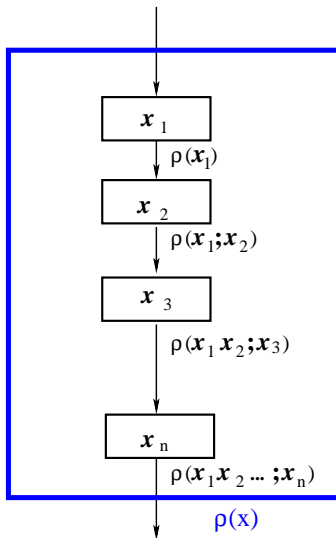
If a general numerically fast method for finding such a mapping was available, we could forget about all other MC methods!!!

Sequential mapping

Sequential mapping in n -dimensions usually boils down to brute force numerical pretabulation.

For instance Backward Evolution of Sjostrand in the MC parton shower.

In n -dimensions mapping the distribution of the next variable (down the tree) involves previously generated variable as a parameter, as indicated in the graph.



COMBINING ELEMENTARY METHODS

Combining Rejection, Branching and Mapping

Combining primitive methods leads quickly to very complicated scenarios with subtle issues not easy to explain and document!

You will get a flavour of that...

Combining Rejection and Branching (A)

Simple combination of Branching and Rejection

Combining rejection + branching.
Rejection 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.

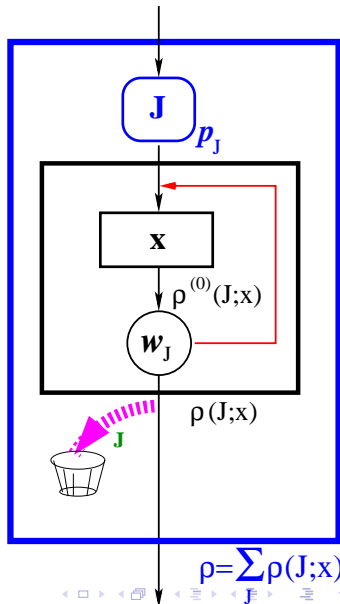
PROBLEM: p_J not known in advance.

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, requires $p_J \rightarrow p_J^{(0)}$.

So why not put J -generation inside rejection loop?



Combining Rejection and Branching (B)

Rejection return point before J -generation (see `examp_branch2w.cxx`)

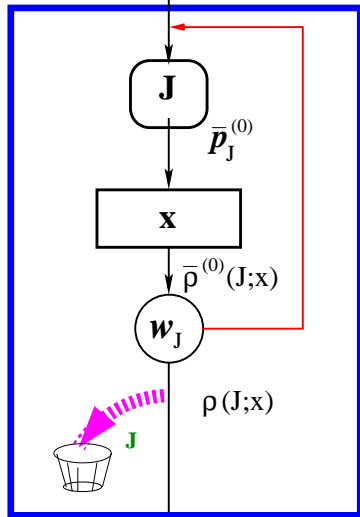
Rejection weight $\bar{w}_J = \frac{\rho(J;x)}{\bar{\rho}^{(0)}(J;x)}$

is aware of the J -branch index;
 J is “trashed” after.

PROFIT: Probabilities $\bar{\rho}_J^{(0)} = \frac{R_J^{(0)}}{\bar{R}^{(0)}}$ for simpler $\bar{\rho}^{(0)}$'s are possibly known analytically!!!

Normalizations controlled by “bared” probabilities and distributions, and 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 $\bar{\rho}_J$.

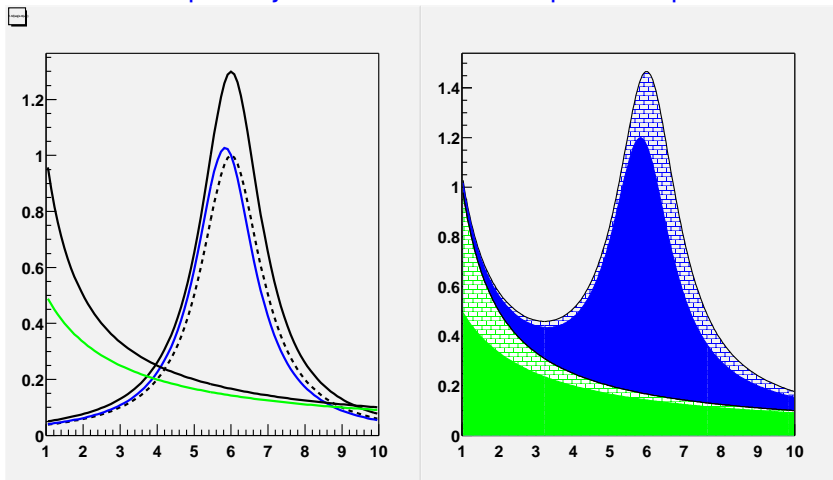


$$\rho = \sum \rho(J;x)$$

Combining Rejection and Branching (B) cont.

1-dim. example with two branches (see `examp_branch2w.cxx`)

Blue and Green areas represent distributions of two branches.
“Brick-walled” part rejected. “Solid-colour” part accepted.

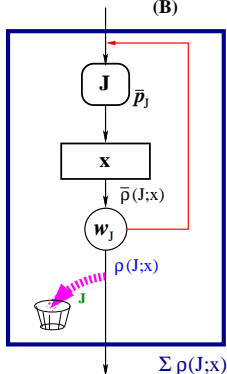


Equivalent algorithms (B) and (C)

Rejection decision point after J -trashing point, see `examp_branch2ow.cxx`

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

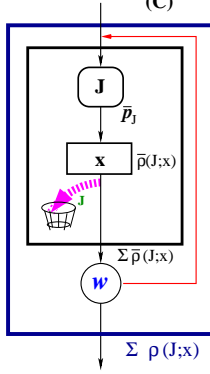
(B)
(C)



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

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

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



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

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

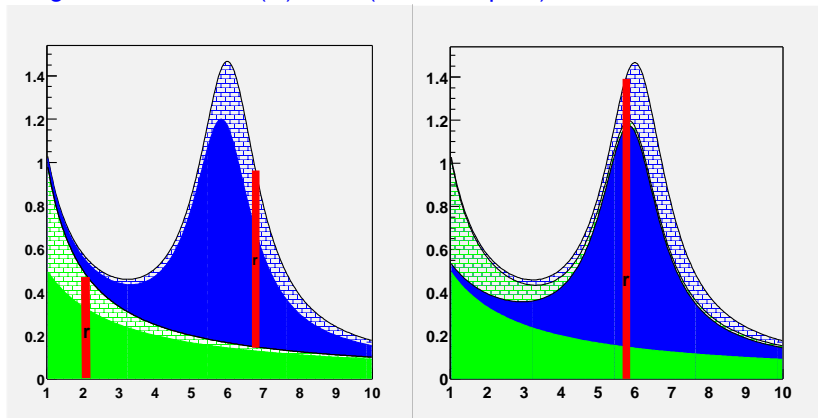
$$\langle\langle w_C^2 \rangle\rangle = \frac{1}{\bar{R}} \int \frac{(\sum_J \rho(J;x))^2}{\sum_J \bar{\rho}(J;x)} dx^n$$

Equivalent algorithms (B) and (C)

Rejection decision point after J -trashing point, see `examp_branch2ow.cxx`

Rejection (of brick-walled) can be executed either
(B) separately for each branch/layer (left) or
(C) at once for both branch/layer (right), weight J -averaged.

Weight distribution in (C) nicer (no blind spots); costs more CPU time.



Branch optimization in case of wt-ed events

Method of Kleiss&Pittau. A step towards self-adapting MCs.

In branching+rejection with “opened rejection loop” target dist. is $\rho(x)$. Generated primarily: $\bar{\rho}(x) = \sum \bar{\rho}(J; x)$ (mapping). Normalization “anchor”: $\bar{R} = \sum_J \bar{R}_J = \sum_J \int \bar{\rho}(J; x) dx^n$. Integral is $R = \bar{R} \langle \langle w \rangle \rangle$, where the average is over branches and over integration domain. The MC error is $\delta R = \frac{\sigma}{\sqrt{N}} = \frac{\bar{G}}{\sqrt{N}} \sqrt{\langle \langle w^2 \rangle \rangle - \langle \langle w \rangle \rangle^2}$

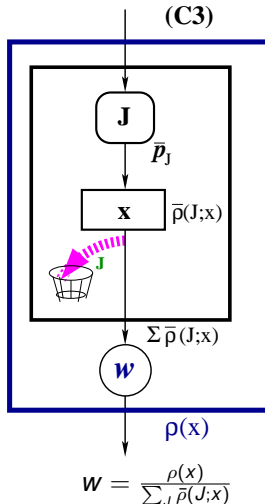
“MC inefficiency” is $iE = N \frac{\delta R^2}{R^2} + 1 = \frac{\langle w^2 \rangle}{\langle w \rangle^2}$

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

Get the smallest inefficiency iE by manipulating relative overall normalizations of the branch distributions $\bar{\rho}(J; x) \rightarrow \lambda_J \bar{\rho}(J; x)$.

PROCEDURE: Start from $\lambda_J = 1$, 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

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



GENERAL PURPOSE MC METHODS AND TOOLS

VEGAS and FOAM

VEGAS and FOAM compared

- Both are General Purpose MC simulators/integrators with automatic self-adapting to user-provided distribution.
- Both are for MC problems in $n < 100$ dimensions.
- Primary aim of FOAM is MC/Stochastic SIMULATION (Weight=1 events)
- Primary aim of VEGAS is MC INTEGRATION; but was customized to provide MC events (un)weighted.
- FOAM is adapting system of hyper-cells to shape of distr.
- VEGAS is adapting division of each of the variables into unequal intervals to user distr.
- FOAM is included in ROOT, VEGAS in GNU libraries.

General purpose self-adapting MC tools

SELECTED REFERENCES

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 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 $\frac{\rho(x)^2}{\rho(x)}$. 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 $\frac{\rho(x)^2}{\rho(x)}$ are calculated and the new binning is established.

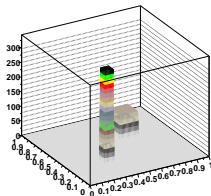
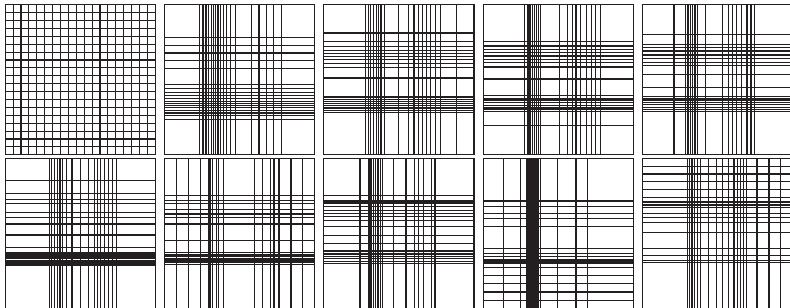
VEGAS basic deficiencies

VEGAS is FANTASTIC but...

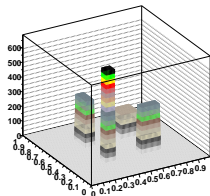
- For a given $\rho(\vec{x})$ there is certain limiting value of the MC Simulation efficiency $\frac{\langle W \rangle}{W_{\max}}$ which cannot be overcome by further enlarging the grid and/or number of the MC trials.
- The above failure and general inefficiency occurs for integrands badly approximated by the product $\rho(x) = \prod_{i=1}^n \rho_i(x_i)$.
(This limitation not present in FOAM).

VEGAS grid oscillations, iterations 1-10:

for a “non-factorisable” 2-dim. distribution



True distr.



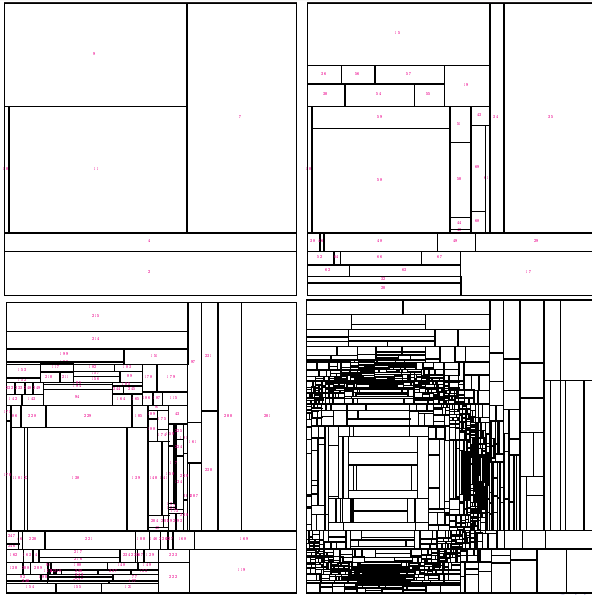
Vegas Approx.

FOAM algorithm

- FOAM operates in “two-stroke” mode:
 - Exploration phase; building system of cells adapted to the shape of the user-provided distribution
 - Generation $WT=1$ MC event series (optionally weighted) exactly according to user distribution.
The integral value is also calculated.
- In exploration cells are divided step by step into 2 daughter cells, such that projected final MC efficiency is minimized
- The choice of the division wall (position and direction) is done with help of a short MC exploratory MC run within the cell.
- Creating system of cells may take time, so there is a possibility to dump it into a disk file and reuse many times (persistency).

Evolution of the cell system

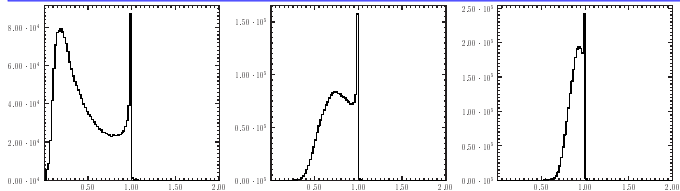
with the growing number of cells= 10, 70, 250, 2500.



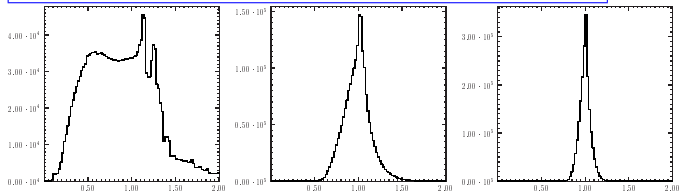
Evolution of the weight distribution

with the growing number of cells=200, 2000, 20k

Minimization of maximum weight, SIMULATION mode



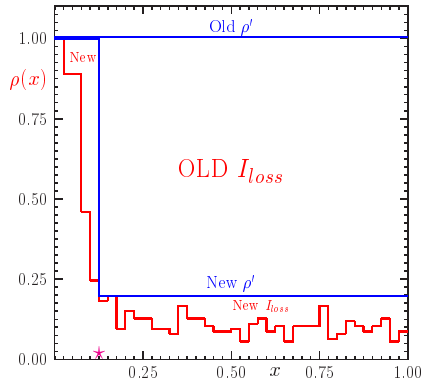
Minimization of variance, INTEGRATION mode



Cell division algorithm

is the most complicated part of FOAM algorithm

For each cell and each edge results of MC exploratory run are examined to find the best edge and the best point of the division into 2 daughter cells.



- $\rho(x)$ is from MC exploratory run
- Old majorizing ρ' for parent cell
- New ρ' for two daughter cells
- OLD R_{loss} all area above red line, for the parent cell
- New R_{loss} between red line and New ρ' , for 2 daughters
- Division point \star is chosen to MINIMIZE the INEFFICIENCY functional R_{loss} !

Comaring FOAM and VEGAS at low dimensions

Tabulated Efficiency = (AverageWeight)/(MaximumWeight)

For non-factorisable distributions FOAM is superior.

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 with 200 MC events/cell

Asking whether FOAM or VEGAS is better is pointless

The answer depends upon the type of the user-distribution and the type of the problem

SUMMARY

- Monte Carlo stochastic methods are great tools for all multi-dimensional problems!
- They are easy to learn and use.
- But sophisticated MC algorithms are not easy to construct and document.
- General purpose MC tools like FOAM and VEGAS make our live easier!