

Can one do better than VEGAS ?

Foam: Multi-dimensional General Purpose
Monte Carlo Generator With Self-adapting
Simplicial Grid

S. Jadach

Institute of Nuclear Physics, Kraków, Poland
and
University of Tennessee, Knoxville

Outline:

- Introduction: what is the problem?
- Foam algorithm and its implementation
- Tests and an application (beamstrahlung in $\mathcal{K}\mathcal{K}$ MC)
- Conclusions.

e-Print: physics/9910004, by S. Jadach

Program source: <http://home.cern.ch/jadach>

Simulation \neq Generation \neq MC-Integration

Distribution $\rho(x_1, x_2, \dots, x_n)$, integral $I = \int_{\Omega} d^n x \rho(\vec{x})$.

MC Integration (MC Integrator):

Provides **integral I** , but **NO MC events!**

$$\rho(\vec{x}) < 0 \text{ allowed! Efficiency} = \frac{\langle WT \rangle}{\sqrt{\langle WT^2 \rangle}}; \quad \Delta I = \frac{\sigma(WT)}{\sqrt{N}}.$$

Special (fractal) random numbers allowed and desirable.

MC Generation (MC Generator):

Provides **WT-ed MC events (\vec{x}, WT)** and the integral I ,

$$\rho(\vec{x}) < 0 \text{ allowed! Efficiency} = \frac{\langle WT \rangle}{\sqrt{\langle WT^2 \rangle}} = \frac{\langle WT \rangle}{\sqrt{\langle WT \rangle^2 + \sigma(WT)^2}}.$$

Random numbers true or special.

MC Simulation (MC Simulator):

Provides **WT=1 MC events \vec{x}** and the integral I ,

$$\rho(\vec{x}) \geq 0 \text{ !!! Efficiency} = \frac{\langle wt \rangle}{wt_{\max}}, \quad wt = \text{internal (rejection) weight},$$

Only true (pseudo)random numbers.

MC Simulator is the most difficult to construct.

MC generator \Rightarrow MC simulator unfeasible,

if crazy wt_{\max} (the usual case).

Original VEGAS = MC-Integrator, PYTHIA = MC-Simulator,

MC-Generators: customized VEGAS, FOWL (1968)

How to improve the MC efficiency?

The answer depends on what the efficiency is!

For MC events $(\vec{x}_j, j = 1, N)$ generated primarily with $\rho'(\vec{x})$
MC weight is $w_j = \frac{\rho(\vec{x}_j)}{\rho'(\vec{x}_j)}$ and integral $I = \langle w \rangle I'$.

MC Integration (MC Integrator):

Variance-Reduction: $\frac{\sigma(w)}{\langle w \rangle} \rightarrow 0$.

MC Generation (MC Generator):

Variance-Reduction: once again.

MC Simulation (MC Simulator):

WTmax-Reduction: Try hard $\frac{\langle wt \rangle}{wt_{\max}} \rightarrow 1 !!!$

In every case the method is to make $\rho' \rightarrow |\rho|$.

However, a given method for $\rho' \rightarrow |\rho|$ which is good for
Variance-Reduction is not necessarily so good for
WTmax-Reduction, and vice-versa.

Definition of WTmax

Naive def. $w_{\max} = \text{Max}_j w(\vec{x}_j)$ is not good.
Problems in case of w with “weak tail” due to:
(1) seldom numerical instabilities or
(2) “weak singularity” like $x^{-0.1}$ or $\ln 1/x$ in ρ .

The remedy is to define w_{\max}^ε as follows:

For a given precision level $\varepsilon \ll 1$, the w_{\max}^ε is determined from the weight distribution in such a way, that the relative contribution to the $\langle w \rangle$ (that is to the total integral) from “under-rejected” (“over-weighted”) events with $w > w_{\max}^\varepsilon$ is equal ε .

In practice it is a little bit of effort to determine the w_{\max}^ε from the weight distribution. One has to create a histogram for the weight distribution with at least ~ 1000 bins, in order to determine the w_{\max}^ε with at least 2-digit precision.

Custom versus General purpose MC's

Custom-made MC is for a narrow class of distributions.

For a well defined family of $\rho(\vec{x})$. For single physical problem.

Made by combining cleverly a handful of elementary methods:

- (a) Mapping of variables,
- (b) Rejection according to a certain weight,
- (c) Multi-branching (multi-channelling).

Variance-Reduction and WT-Reduction “by hand” and v. good!

Custom-Made are all (?) MC Simulators, most of MC Generators.

General Purpose MC's have built-in ability of adjusting to ρ .

For a wide class of ρ 's. Practically only one type exists:

- (1) Divide integr. domain Ω into many small cells $\omega_i, i = 1 \dots l$
- (2) Generation: choose i and generate $\vec{x} \in \omega_i$ uniformly.
- (3) σ & WT-Reduction automatic, due to clever division into ω_i .

VEGAS family: cells are cubical.

Factorisability $\rho(\vec{x}) \simeq \prod_{j=1}^n f_j(x_j)$ is required.

FOAM: cells are simplicial. Factorisability not required.

Hybrid: Custom-made, but some variables General-Purp.

VEGAS

- G.P. Lepage, J. Comput. Phys. **27**, 195 (1978).

Recent activity in the subject:

- 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).
- 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).
- R. Kleiss and R. Pittau, Comput. Phys. Commun. **83**, 141 (1994).

This work:

- S. Jadach, Comput. Phys. Commun. in press, e-Print: physics/9910004.

VEGAS algorithm

The integrand function in n dimensions is assumed to be fairly well approximated by a product of functions, 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. The result of repeated MC exploration runs is used to improve the binning.

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

VEGAS customization

VEGAS is designed as MC *integrator*. With almost no modification it can produce MC weighted events (*MC generator*).

More effort is required to produce constant weight events (*MC simulator*). It requires recording maximum weight during the last iteration, for each integration variable, a maximum weight in each of the n bins. The multichannel generation of the sub-rectangles done using the product of the maximum weights.

VEGAS deficiency

The well known deficiency of VEGAS is that for a given $\rho(\vec{x})$ there is certain limiting value of the MC efficiency which cannot be overcome by further enlarging the grid and/or number of the MC trials.

This is due to factorizability requirement.

The efficiency of the MC Simulation $\frac{\langle wt \rangle}{wt_{\max}}$ is usually even worse than the efficiency of the MC generation $\frac{\langle WT \rangle}{\sqrt{\langle WT^2 \rangle}}$.

The principal aim of Foam is to overcome completely the above limitation.

Other General-Purpose MC efforts:

- **VAMP (T. Ohl, 1999)**
Improved VEGAS. Integrand approximated not with a single product $\prod f_i(x_i)$ like in VEGAS but with the linear combination of them.
- **BASES (S. Kawabata, 1995)**
Variables with the strongest singularities (wild) treated as in VEGAS, while the other variables are “averaged over”. Number of bins is not the same for all variables as in VEGAS.
- **MILXy way (G. I. Manankova et.al. 1995)**
With simplicial cells like Foam. Lack of implementation.

Assumptions and Aims:

- **Take care of several ($n < 10$) “wildest variables”.**

These with the strongest singularities. The other to be dealt with the VEGAS, or “averaged over” like in BASES.

- **The integrand completely arbitrary.**

No factorizability assumption. Singularities on arbitrarily shaped hyperspaces like ridges along diagonals, and “thin” hypersurfaces like sphere etc. Also big voids.
(For extremely narrow peaks, it always make sense to map variables.)

- **Initialization: Grid of vertices forming a “foam of cells” built, adapting automatically to the integrand.**

The resulting ratio $\langle w \rangle / w_{\max}$, i.e. the efficiency, is *arbitrarily* good. In the MC generation one cell is chosen and a point within this cell.

- **Iterative succession of “grows” and “collapses” of the foam to be available as an option.**

In order to remove/replace “unsuccessful” branch of the growth.

- **Integrand positive and integrable but weak integrable singularities should be allowed.**

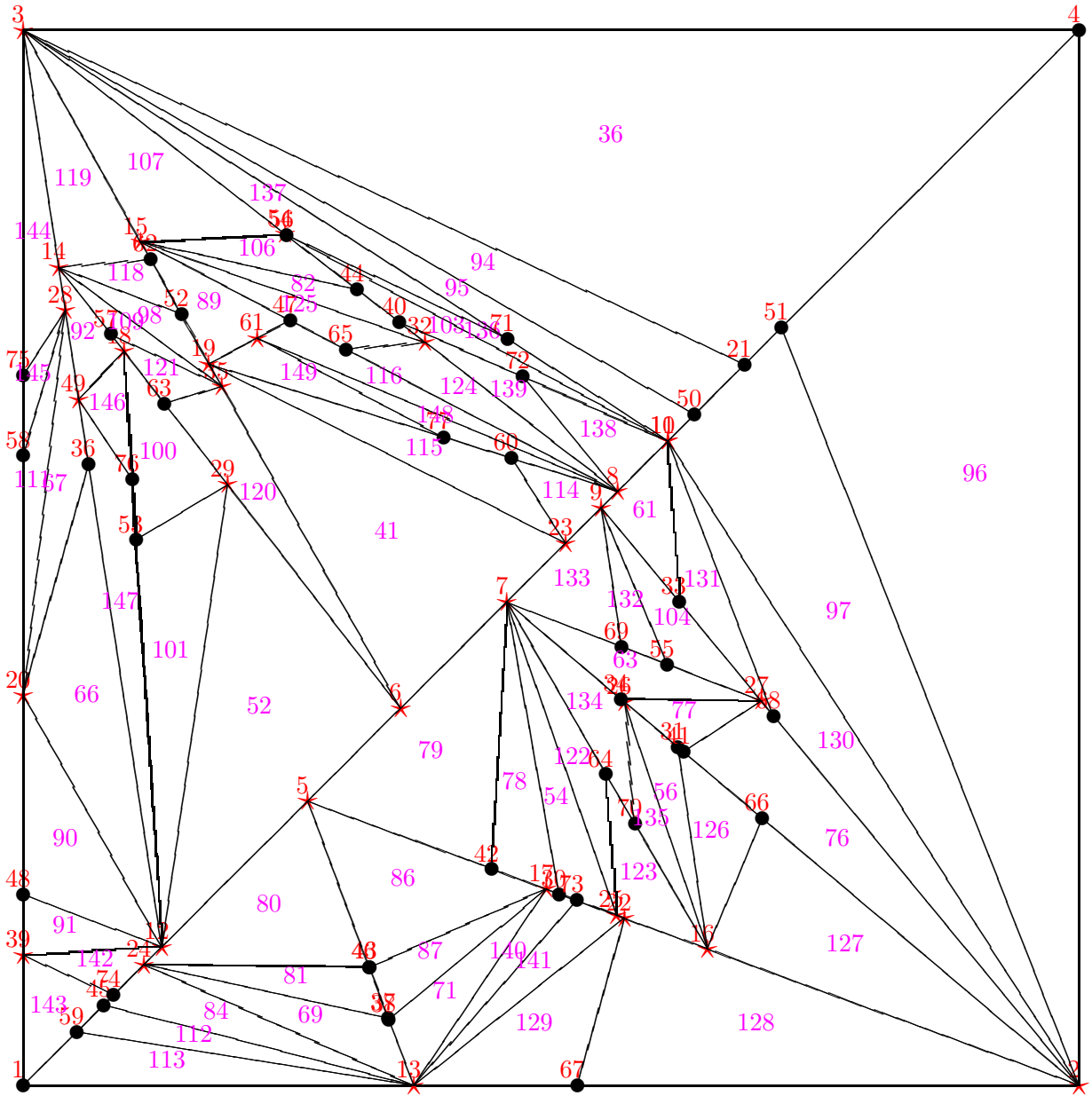
Singularities of the type \sqrt{x} or $\ln(1/x)$.

Data structures

- The basic data structure is the **foam**.
Foam = hierarchical linked list of **cells**.
- A simplex-shaped **cell** is uniquely defined by its **vertices**. Cells actually contain only pointers to vertices. Vertices are in a separate list of all vertices, being the n -component vectors.
- Each cell has also many other attributes. Such as pointers to parent and daughter cells, its volume, an estimate of the integral over the cell (the true integral) average weight, maximum weight etc.
- **Inactive cells** which underwent division, point to 2 daughter cells and **active cells** have no daughters.
Active cells actually cover entire integration area.
- The **root cell** = the entire integration region = a cube of **unit size**. It splits into $n!$ simplicial cell.

Foam Algorithm

The foam of cells



MC generation of the cell and the point

- In the MC simulation one **active cell** is chosen randomly according to its **crude integral** which is usually bigger than its proper integral.
- Each inactive cell knows the sum of crude integrals of all active cells it contains (all its daughter and granddaughter cells). It is done in such a way in order to make generation of the active cell as natural and simple as possible.
- Generation starts with inactive **root** cell at the top of the tree – one of the daughter cells is chosen randomly according to its crude integral.
- This process continues down the tree until an active cell (with no daughters) is randomly chosen.
- A point (vector) inside selected active cell is chosen with uniform probability.

Initialization: growth and collapse

The foam is constructed during the **initialization** phase in the subsequent *growths* and *collapses*.

The initial cube is divided into $n!$ equal simplices, daughter cells, each daughter cell is subjected to an **MC exploration** eg. a certain number of MC events is generated within the cell in order to calculate the average weight, dispersion, maximum weight, minimum weight, proper integral (MC estimator) and more.

In the **growth** phase each cell has a chance to get divided into 2 daughters. The chosen cell is tagged as inactive and divided into 2 daughter cells (active) and each daughter cell undergoes immediately MC exploration. The recipe for the **cell division** is the most important part of the algorithm.

In the case of a strongly peaked distribution, growth may go into a “wrong area”, so one is interested in a possibility of trimming/downsizing the foam, which is termed the **collapse** of the foam.

The growth is finished when the memory buffer is filled up.

Crude integral and the MC weight

In the initialization, the basic weight is $w = f(x)V_{Cart}$ where f is integrand function, and V_{Cart} is Cartesian volume of the cell. The proper integral is equal the average weight:

$$\langle w \rangle \equiv \int_{cell} f(x)d^n x, N_{MC} \rightarrow \infty.$$

The **crude integral** over the cell is an overestimated integral over the cell, used in the subsequent MC generation. MC weight w_{MC} will compensate for the fact that crude integral is not the true value of the integral over the cell. If decent maximum weight is main priority then crude integral has to be

$$I_{crude} = w_{max} = \text{Max}_X f(X) V_{Cart},$$

i.e. the maximum value of the integrand function times the volume of the cell. w_{max} is of course an estimate from the MC exploration of the cell. For this choice the condition $w_{MC} \leq 1$, essential for turning weighted events into $w_{MC} = 1$ events by means of the rejection, is not violated too often.

Optionally, for the purpose of the MC integration:

$$I_{crude} = \sqrt{\langle w \rangle^2 + \sigma^2} = \sqrt{\langle w^2 \rangle},$$

like in Vegas is available (it is either $\langle w \rangle$ or variance σ).

In any case, the compensating weight for the MC generation is always the same: $w_{MC} = f(x)V_{Cart}/I_{crude}$.

Cell division

Pair of vertices x_i and x_j is chosen and a new vertex Y is put somewhere on the line between:

$$Y = \lambda x_i + (1 - \lambda)x_j, \quad 0 < \lambda < 1$$

Two daughter simplices are defined with the list of vertices:

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

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

How do we choose (i, j) pair and the value of λ ?

Short sample of the MC events (100-1000) is generated inside the cell – MC exploration is done. Each MC point X is “projected” into a point Y on a given edge $(i, j), i \neq j$:

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

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

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

$$r_k = x_k - X, \text{ and } \text{Det}(x_1, x_2, \dots, x_n) \text{ determinant.}$$

The division is made for an edge (i, j) along which the integrand is **varying most rapidly**. How to select (i, j) ?

For each (i, j) the distribution $dN/d\lambda$ is histogrammed, the value of the integral $R_{i,j} = \int \left| \frac{dN}{d\lambda_{i,j}} - N \right| d\lambda_{i,j}$ is calculated.

The edge (i, j) with the biggest $R_{i,j}$ is selected!

For the division $\lambda = \langle \lambda_{i,j} \rangle$ is taken.

Data

- Single **class common block /c_FoamA/** with all **class member** variables, placed in the header file **FoamA.h** . Variables **class members** have special prefix “m_” in their name, e.g. **m_Iterat** is number of iterations.
- Each subroutine in **FoamA.f** has **INCLUDE 'FoamA.h'** statement. The outside programs should **never** (!) include **/c_FoamA/** . All input/output communication is done with the help of dedicated, subroutines.

Categories of subprograms

- **Constructor** **FoamA_PreInitialize**: Pre-sets default values of many (input) variables. Invoked automatically.
- **Initializer** **FoamA_Initialize**: Performs initialization of the foam grid.
- **Finalizer** **FoamA_Finalize**: Summarizes the whole run, sets output values in **/c_FoamA/**, prints output etc.
- **Maker** **FoamA_MakeSomething**: Does the essential part of job, e.g. **FoamA_MakeEvent** generates single MC event.
- **Setter** **FoamA_SetVariable**, is called from the outside world to set **m_Variable** in **/c_FoamA/**. For example **CALL FoamA_SetIterat(5)** sets variable **m_Iterat=5**. Only certain privileged variables have a right to be served by their own setter, the other ones are in principle “private”. Most of setters should be called before initialization.
- **Getter** **FoamA_SetVariable**, is called from the outside world to get **m_Variable** from **/c_FoamA/**. It is a preferred way of sending output information to outside world. For example, with **CALL FoamA_GetMCwt(MCwt)** one gets MC weight **MCwt** in the user program.

Program usage

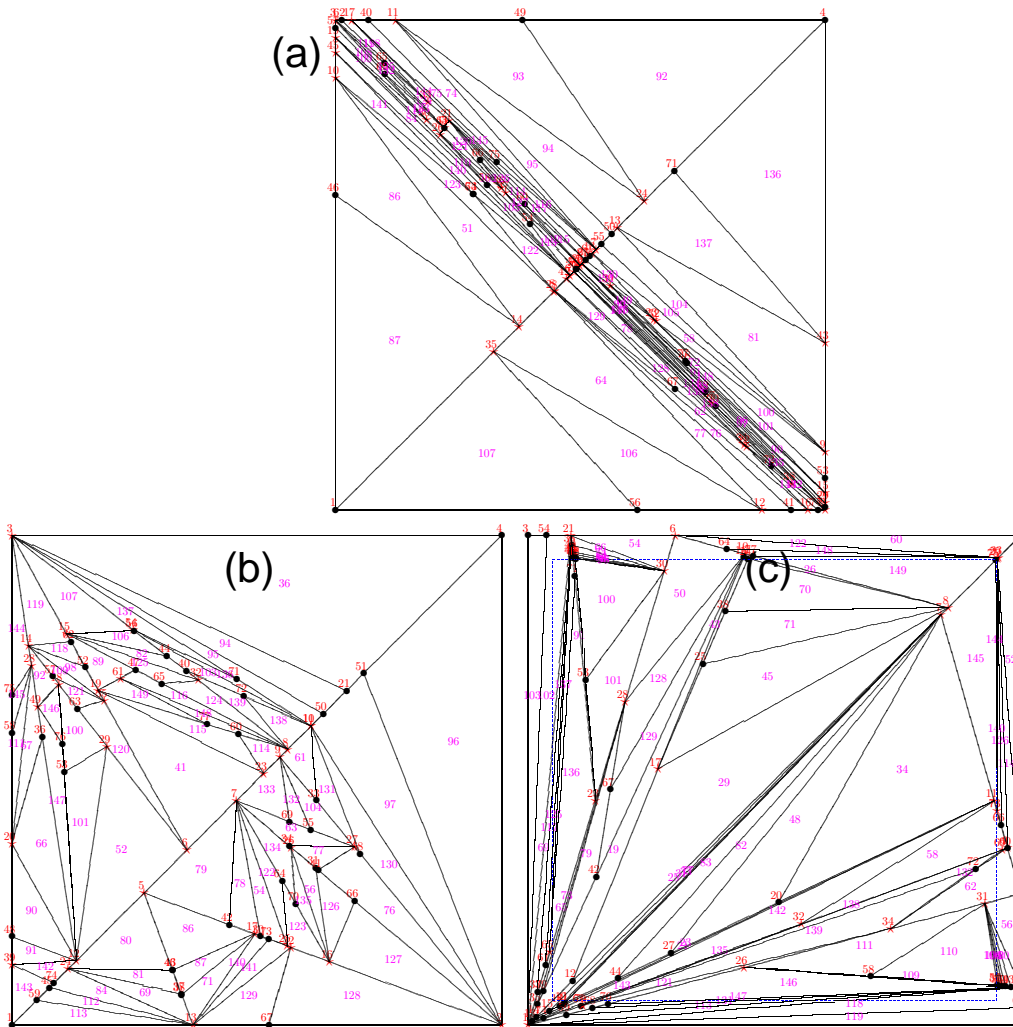
```

*-----
DOUBLE PRECISION   Density
EXTERNAL           Density
CALL FoamA_SetNdim(      3) ! number of dimensions
CALL FoamA_SetnBuf(    2000) ! length of buffer
CALL FoamA_SetIterat(   1) ! number of iterations
CALL FoamA_SetnSampl(   500) ! no. MC ev./cell (init.)
CALL FoamA_SetOptCrude(  2) ! type of crude, =2 default
CALL FoamA_SetOptEdge(  1) ! edge point incl.(=0 excl)
CALL FoamA_SetChat(     1) ! printout level
CALL FoamA_Initialize(Density) ! initialize foam grid
DO loop = 1, 200000
  CALL FoamA_MakeEvent(Density)      ! generate MC event
  CALL FoamA_GetMCvector(MCvector)  ! get MC event, vec.
  CALL FoamA_GetMCwt(MCwt)          ! get MC weight
  CALL GLK_Fill(1000, MCwt,1d0)     ! histogramming
ENDDO
CALL FoamA_Finalize(MCresult,MCerror) !get integr.+err.
CALL FindWtLimit(1000)      ! user prog. check MC efficiency
*-----

```

In fact user has to set only the number of dimensions **Ndim**. The other input variables **nBuf**, **Iterat**, **nSampl**, **OptCrude**, **OptEdge**, **Chat** are already preset for the user thus calling setters for them is optional. User needs to provide his own integrand function which in the example is **Density**.

2-dimens. foam of cells for 3 density functions



Density f_a peaks along one of diagonals of the square, f_b peaks along a 2% wide ring centered at (0.25,0.40) of radius 0.25, and f_c represents 5% wide band along four edges of the square. For f_c the boundary of the nonzero integrand is marked with a dashed line.

Three 2-dimensional testing functions:

$$f_a = 1 - \Theta(0.5 - |x_1 - 0.5| - \gamma) \Theta(0.5 - |x_1 - 0.5 - \gamma|), \quad \gamma = 0.05,$$

$$f_b = \frac{1}{4\pi R^2} \frac{\gamma}{\pi[(R - \sqrt{(x_1 - 0.25)^2 + (x_2 - 0.40)^2})^2 + \gamma^2]},$$

$\gamma = 0.02, R = 0.35$

$$f_c = \frac{\gamma}{\pi[(x_1 + x_2)^2 + \gamma^2]}, \quad \gamma = 0.02.$$

Functions	Foam	VEGAS
$f_a(x_1, x_2)$ (diagonal ridge)	0.94	0.05
$f_b(x_1, x_2)$ (circular ridge)	0.83	0.15
$f_c(x_1, x_2)$ (edge of square)	0.57	0.53

Efficiency w_{\max}^ε , for $\varepsilon = 10^{-4}$, of Foam and VEGAS for 3 examples of the 2-dim. integrand function. After initialization, efficiency was determined from sample of 10^6 MC events. Results from Foam are for 5000 cells (about 2500 active cells) and cell exploration was based on 200 MC events/cell.

For a given precision level $\varepsilon \ll 1$, w_{\max}^ε is determined from the weight distribution in such a way that the contribution to $\langle w \rangle$ (that is to total integral) from “under-rejected” (“over-weighted”) events with $w > w_{\max}^\varepsilon$ is equal exactly ε . The rejection rate (efficiency) is $w_{\max}^\varepsilon / \langle w \rangle$.

3-dimensional test:

Functions	Foam	VEGAS
f_a (thin diagonal)	0.67	0.04
f_b (thin sphere)	0.36	0.10
f_c (surface of cube)	0.37	0.33

Efficiency w_{\max}^ε , for $\varepsilon = 10^{-4}$, of Foam and VEGAS for 3 examples of the 3-dimensional integrand functions, the analogs of the previous 2-dimensional distributions.

After initialization, efficiency was determined from sample of 10^6 MC events.

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

Again Foam is clearly superior, and again the efficiency of VEGAS already is at its limiting/asymptotic value while the efficiency of Foam can be still improved by adding more cells in the initialization phase.

In this 3-dimensional case the CPU time consumption of Foam is noticeably smaller than of VEGAS.

Confusion?

Is this the problem that simplices get excessively elongated?

No! For long and flat “ridge” they should.

Yes! The example with the “void” demonstrates this.

We need a bright idea how to limit excessive elongation in certain cases but not eliminate it.

Beamstrahlung as a MC problem

Most general form of the differential distributions in presence of beamstrahlung in $e^+e^- \rightarrow X$ reads:

$$d\sigma^{bst}(p_a, p_b; X) = \int dz_1 dz_2 \mathcal{D}(z_1, z_2, \sqrt{s}) d\sigma(z_1 p_a, z_2 p_b; X)$$

where $\mathcal{D}(z_1, z_2)$ is the double differential beamstrahlung “luminosity spectrum” for simultaneous beamstrahlung from both beams. The luminosity spectrum is assumed to be in the most general form

$$\mathcal{D}(z_1, z_2) = \delta(1 - z_1)\delta(1 - z_2)\rho_0 + \delta(1 - z_1)\rho_1(z_2) + \delta(1 - z_2)\rho_1(z_1) + \rho_2(z_1, z_2)$$

where ρ_0 is constant and $\rho_1(z)$ and $\rho_2(z_1, z_2)$ are analytical functions for $z \in [0, 1)$. We allow for power-like integrable singularities at $z_i = 1$:

$$\rho_1(1 - \epsilon) \sim \epsilon^\alpha, \quad \rho_2(1 - \epsilon, z_2) \sim \epsilon^\beta, \quad \rho_2(z_1, 1 - \epsilon) \sim \epsilon^\beta.$$

Otherwise the functions ρ_i are regarded as completely arbitrary and we require that:

the M.C. should cope efficiently with any beamstrahlung luminosity spectrum of the above most general type.

Beamstrahlung in $\mathcal{K}\mathcal{K}$ MC

Generation of the z_i variables is done at the very beginning of the generation, together with the generation of the QED ISR total photon energy.

The technical problem to be solved is the following:

How to generate up to 3 variables, the two variables z_1, z_2 for beamstrahlung and one $z = 1 - v$ for ISR, according to a highly singular probability density distribution?

Not only the beamstrahlung spectrum and ISR photon distribution are strongly singular but in addition the M.C. algorithm has to deal efficiently with singularities due to resonances and thresholds in the reduced CMS energy variable $s'' = sz_1z_2z$.

In addition due to presence of the δ 's in the general beamstrahlung distribution 4 branches in the MC generation are present: one 1-dimensional, two 2-dim. and one 3-dim., corresponding to each term. In each branch the corresponding subset of the z_1, z_2 and z should be generated.

Beamstrahlung in $\mathcal{K}\mathcal{K}$ MC

In the present version we use Foam and the Vegas program, customised to our needs.

The second solution based on VEGAS is, however, much less efficient than the principal one.

Note that variables z_1, z_2 and z are generated in the beginning of the whole $\mathcal{K}\mathcal{K}$ MC algorithm, consequently they have to be generated with weight equal one (MC simulation), otherwise we would waste CPU time for calculating complicated QED matrix element for “unworthy” weighted events).

The new M.C. tool Foam is specialized exactly for the task of generating efficiently $WT=1$ events. The efficiency of the Foam solution in terms of the rejection rate (turning weighted into unweighted events) is about factor 10 better than of the VEGAS. (VEGAS will be probably removed in the future version)

Change of variables (mapping)

Take the worst case case with $\int dz_1 dz_2 dz$ where $z = s'/s$ and s' is “after ISR”.

By trial/error found that the best mapping for $\sqrt{s} < 1\text{TeV}$ is:

$$x_1 = 1 - z_1, \quad x_2 = 1 - z_2, \quad v = 1 - z$$

$$x_1 = r_1^{1/\alpha}, \quad x_2 = r_2^{1/\alpha}, \quad v = r_3^{1/\gamma} v_{\max}$$

$0 < r_i < 1$ are used by Foam, α is from CIRCE and for ISR we have $\gamma \sim 2(\alpha_{QED}/\pi) \ln(s/m_e^2)$.

The integral gets transformed into:

$$\begin{aligned} & \int dz_1 dz_2 dz F(z_1, z_2, s z_1 z_2 z) \Theta(1 - z_1 z_2 z - v_{\max}) \\ &= \int_0^1 dr_1 dr_2 dr_3 \frac{v v_{\max}}{r_3 \gamma} \frac{x_1}{r_1 \alpha} \frac{x_2}{r_2 \alpha} \Theta(1 - z_1 z_2 z - v_{\max}) F \\ &= \int_0^1 dr_1 dr_2 dr_3 \frac{v_{\max} \Theta(1 - z_1 z_2 z - v_{\max}) F}{(\alpha x_1^{\alpha-1})(\alpha x_2^{\alpha-1})(\gamma v^{\gamma-1})} \end{aligned}$$

Note that this mapping takes care of singularities at $z_i = 1$ only, and not of Z resonance peak, nor of the phase space limit $1 - z_1 z_2 z < v_{\max}$.

Surprisingly, more sophisticated mapping gives efficiency much worse!

(For both Foam and Vegas).

Summary

- A new general purpose Monte Carlo tool based on a symplectic self-adapting grid is available.
- Numerical tests show that for non-factorizable integrand functions it is more effective than the classical VEGAS solution.
- The main limitation is that program is adapted for relatively small dimensions, $n < 8$.
- The algorithm and the program should not be treated as a final solution – it is rather a beginning of a new development direction in MC methods.
- The first real application to beamstrahlung and ISR in the $\mathcal{K}\mathcal{K}$ Monte Carlo confirms maturity of the solution.

Postscript of the slides is on: <http://home.cern.ch/jadach>
As well as source code of the Foam and the paper.