Adaptive Monte Carlo Integration Methods

Marcin Chrząszcz
mchrzasz@cern.ch

University of Zurich[UZH]

Monte Carlo methods,
10 March, 2016

Classical methods of variance reduction

⇛ In Monte Carlo methods the statistical uncertainty is defined as:

$$\sigma = \frac{1}{\sqrt{N}} \sqrt{V(f)}$$

⇛ Obvious conclusion:

- To reduce the uncertainty one needs to increase N.
  ⇉ Slow convergence. In order to reduce the error by factor of 10 one needs to simulate factor of 100 more points!

⇛ How ever the other handle ($V(f)$) can be changed! ⟶ Lot's of theoretical effort goes into reducing this factor.

⇛ We will discuss four classical methods of variance reduction:

1. Stratified sampling.
2. Importance sampling.
3. Control variates.
4. Antithetic variates.

Disadvantages of classical variance reduction methods

⇛ All aforementioned methods(beside the Stratified sampling) require knowledge of the integration function!
⇛ If you use the method in the incorrect way, you can easily get the opposite effect than intendant.
⇛ Successful application of then require non negligible effort before running the program.
⇛ A natural solution would be that our program is ”smart” enough that on his won he will learn something about our function while he is trying to calculate the integral.
⇛ Similar techniques already were created for numerical integration!
⇛ Truly adaptive methods are nontrivial to code but are widely available in external packages as we will learn.
⇛ Naming conventions:

- Integration MC- software that is able to compute JUST! integrals.
- Generator MC- software that BESIDES beeing able to perform the integration is also capable of performing a generation of points accordingly to the integration function.

Schematic of running this kind of methods

1. Function probing (exploration):
   - Recursive algorithm that searches for hipper-surfaces in which the function is approximately close. For evaluation of an integral in a given hipper-surface normally one uses numerical or MCcrude methods. In general it is not a easy task!
   - Often the function is approximated by a given set of elementary functions.
2. Calculation phase
   - The integral is calculated using mostly using Stratified Sampling and Importance Sampling, depending on exploration phase.
   - If a MCprogram has capability to generated distributions accordingly to the function of which we want to calculate the integral, it's in this place where it happens.

$\Rightarrow$ There are algorithms where the exploration phase is linked with calculation phase. For each of the optimisation phase the integral is calculated as well. The result will be weighted average of those integrals!

This method might be bias! if in the extrapolation phase the function picks up a function peaks to late the whole method will lead to systematically bias results.

RIWIAD algorithm

$\Rightarrow$ The first algorithm of this kind RIWIAD was proposed by Sheppeya & Lautrupa in 1970s. It was used to calculate integrals in cube $(0,1)^n$.

$\Rightarrow$ It worked as follows:

- At the begging the hipper-cube is divided in equal size sub cubes. In each of them the integral is calculated.
- Based on the calculated integrals programs moves the boundaries to make the hipper-cubes smaller in the places where the function is greater and smaller where the function is smaller.
- The process starts over and continues over and over again. At each step the integral estimator and it's standard deviation is calculated. Form those a weighted average is constructed and it's standard deviation is constructed and its standard deviation.
- The process stops when the standard deviation reaches our desired sensitivity.

$\Rightarrow$ Disadvantages:

- Hipper-cubes are always parallel to the coordinate axis.
- Some are are divided even thought they didn't have to.
- The weighted average might be a bias estimator.

Friedmanns algorithm

$\Rightarrow$ In the 1970s J.Friedmann has also developed an adaptive MCintegration algorithm.

$\Rightarrow$ The algorithm was as follows:

- A probe function is constructed using a combination of Cauchy functions (Briet-Wigner), in which the peaks correspond to the local maxima of the integration function. In order to do so one needs to study the eigen functions in the neighbourhood of each peak (nasty thing...).
- The Briet-Wigner is it falls down to 0 slower then a Gauss distribution.
- The integral and the standard deviation is calculated based on the weighted averaged based on the probe function.

Disadvantage:

Cannot be applied to functions that cannot be approximated with small number of Briet-Wigner functions.

DIVIONNE2 algorithm

⇛ J.Friedmann (1977): adaptive algorithm for MC integration based on recursive division of the integration area(available in the CERBLIB package).

⇛ The algorithm:

- Multidimensional division of the hipper-cube. We divide each of the initial sub cubes to minimalise the spread of the function.
- After this the integral is calculated using Stratified Sampling.
- We can generate a events accordingly to this function with this method.

⇛ Disadvantages:

- Hipper-cubes are always parallel to the coordinate axis.

⇛ Advantages:

- Because we divide only one hipper-cube at the time, the procedure doesn't get bias as easily the RIWID does.

VEGAS algorithm

$\Rightarrow$ J. G. P. Lepage (1978): adaptive algorithm for MC integration based on iterative division of the integration area(similar to `RIWID`).

$\Rightarrow$ Let's calculate: $\int_0^1 f(x)dx$.

- We generate M random points from $\mathcal{U}(0,1)$. We calculate from them the integral and standard deviation.
- Now we divide the integration region in N equal subdivisions:

$$0 = x_0 < x_1 < x_2 < ... < X_N = 1,\ \Delta x = x_i - x_{i-1}$$

- Now each of this subdivisions we divide further into $m_1 + 1$ subsubdivisions.

$$m_i = K \frac{\overline{f}\Delta x_i}{\sum_j \overline{f}_j \Delta x_j},\ K = const.\ \text{typically}\ = 1000$$

and

$$\overline{f}_i \equiv \sum_{x \in [x_{i-1}, x_i)} |f(x)| \sim \frac{1}{\Delta x_i} \int_{x_{i-1}}^{x_i} |f(x)|dx$$

$\Rightarrow$ The new subsubareas will be "denser" where the function is greater and less dens where the function is smaller.

VEGAS algorithm

- We are are retrieving back the original number (N) of the subdivisions by glueing together equal amount subsubdivisions.
  $\Rightarrow$ The new subdivisions will be larger where the function is larger and vice versa.
- We generate the M points accordingly to the stop function probability:

$$p(x) = \frac{1}{N\Delta x_i}$$

  and calculate the integral Stratified sampling.
- We repeat the procedure until we find an optimum division:

$$m_i \approx m_j \;\; i, j = 1, ..., N.$$

- In each iteration we calculate the weighted average:

$$\sum_k \frac{I_k}{\sigma_k^2},$$

  where I_k and σ_k are the integral and error in the k interaction.
- After the procedure stop we calculate the final results:

$$\hat{I} = \sigma_I^2 \sum_k \frac{I_k}{\sigma_k^2} \quad \sigma_I = \left[\sum_k \frac{1}{\sigma_k^2}\right]^{-\frac{1}{2}}$$

VEGAS algorithm - futher improvements

$\Rightarrow$ In order to make the integrating area more stable(can happen that the division jumps around very rapidity). We can modify the algorithm:

$$m_i = K \left[\left[\frac{\overline{f}\Delta x_i}{\sum_j \overline{f}_j \Delta x_j} - 1 \right] \frac{1}{\log \left[\overline{f}_i \Delta x_i / \sum_j \overline{f}_j \Delta x_j \right]} \right]^{\alpha},$$

where $\alpha \in [1, 2]$ sets the convergence speed. $\Rightarrow$ When function has narrow peaks the I_k and σ_k might be wrongly calculated in early stages of iteraction. To fix this we can:

$$I = \left[\sum_k \frac{I_k^2}{\sigma_k^2} \right]^{-1} \sum_k I_k \left(\frac{I_k^2}{\sigma_k^2} \right), \quad \sigma_I = I \left[\sum_k \frac{I_k^2}{\sigma_k^2} \right]^{-0.5}$$

$\Rightarrow$ If the number of interactions is to large then you cannot trust the algorithm!

VEGAS algorithm - 2D case

⇛ Lets take for example $\int_0^1 dx \int_0^1 dy f(x,y)$.

⇛ We can do a trick:

$$p(x,y) = p_x(x)p_y(y)$$

⇛ One can show that using Lagrange multipliers that the optimum density has the form of:

$$p_x(x) = \frac{\sqrt{\int_0^1 dy \frac{f^2(x,y)}{p_y(y)}}}{\int_0^1 dx \sqrt{\int_0^1 dy \frac{f^2(x,y)}{p_y(y)}}}$$

⇛ So our 1D algorithm can be used to each of the axis (for x axis):

$$(f_i)^2 = \sum_{x \in [x_{i-1}, x_i)} \sum_y \frac{f^2(x,y)}{p_y(y)} \sim \frac{1}{\Delta x_i} \int_{x_{i-1}}^{x_i} dx \int_0^1 dy \frac{f^2(x,y)}{p_y(y)}$$

⇛ In analogous you do it for y axis.

VEGAS algorithm - an example

⇛ An example of usage; let's calculate:

$$I_n = (\frac{1}{a\sqrt{\pi}})^n \int_0^1 \exp\left[\frac{(x_n - 0.5)^2}{a^2}\right] d^n x$$

⇛ For the $n = 9$, $a = 0.1$ and $\alpha = 1$

Iteration	I_k	σ_k	I	$\sigma(I)$	Number of calculations
1	0.007	0.005	0.007	0.005	10^4
3	0.643	0.070	0.612	0.064	$3 \cdot 10^4$
5	1.009	0.041	0.963	0.034	$5 \cdot 10^4$
10	1.003	0.041	1.003	0.005	10^5
Crude MC method			0.843	0.360	10^5

VEGAS algorithm - comparison to numerical methods

⇛ An example of usage; let's calculate:

$$I_n = (\frac{1}{a\sqrt{\pi}})^n \int_0^1 \exp\left[\frac{(x_n - 0.5)^2}{a^2}\right] d^n x$$

⇛ For the $n = 9$, $a = 0.1$ and $\alpha = 1$.

Number of points on axis	Integral value	Number of calculations
5	71.364	$2 \cdot 10^6$
6	0.017	10^7
10	0.774	10^9
15	1.002	$3.8 \cdot 10^9$

Backup