

PARNI FOR IMPORTANCE SAMPLING
AND DENSITY ESTIMATION*ANDRÉ VAN HAMEREN[†]H. Niewodniczański Institute of Nuclear Physics Polish Academy of Sciences
Radzikowskiego 152, 31-342 Kraków, Poland*(Received October 28, 2008)*

We present an aid for importance sampling in Monte Carlo integration, which is of the general-purpose type in the sense that it in principle deals with any quadratically integrable integrand on a unit hyper-cube of arbitrary dimension. In contrast to most existing systems of this type, it does not ask for the integrand as an input variable, but provides a number of routines which can be plugged into a given Monte Carlo program in order to improve its efficiency “on the fly” while running. Due to the nature of its design, it can also be used for density estimation, *i.e.*, for the analysis of data points coming from an external source.

PACS numbers: 02.70.Rr

1. Introduction

Numerical integration often is the only solution to integration problems encountered in scientific research. If the space over which the integral has to be performed, the *integration space*, has many dimensions, the Monte Carlo method usually appears to be the only feasible option. It has the advantage that it works for any quadratically integrable function, but the disadvantage that the convergence rate may be low, *i.e.*, that many function evaluations may be needed to obtain a result to acceptable accuracy. Via the method of *importance sampling*, however, it is possible to translate information about the integrand into an enhancement of the rate of convergence, in principle up to the point where only one function evaluation is needed to complete the integral. In the latter case, the integration problem has essentially been solved analytically.

* Supported in part by the EU RTN European Programme, MRTN-CT-2006-035505 (HEPTOOLS, Tools and Precision Calculations for Physics Discoveries at Colliders) and by the Polish Ministry of Scientific Research and Information Technology grant No 153/6 PR UE/2007/7 2007–2010.

[†] Andre.Hameren@ifj.edu.pl

Obviously, information about the integrand is obtained during the Monte Carlo integration process itself through the evaluation of the integrand at the integration points. Importance sampling based solely on this information is called *adaptive* importance sampling. It is the last resort to improve the rate of convergence, possibly only applied to a sub-region of the integration space. In particular, it can be useful for so called *general-purpose* integration systems which are supposed to be able to deal with many different integrands. Several such systems have been developed and are being used extensively in the field of elementary particle phenomenology [1–5]. Most of these use the evaluated integration points to adapt the treatment of sub-spaces in which they are generated during the integration process.

The ability to apply importance sampling successfully implies the availability of a probability density which has a shape comparable to the shape of the absolute value of the integrand. An attempt to use importance sampling is essentially the search and construction of such a probability density. Consequently, given such a density, one can generate points in the integration space which are distributed following the shape of the absolute value of the integrand, or at least one has a tool to increase the efficiency of such generation process using straightforward methods like rejection. This is an important issue if one wants to perform Monte Carlo *simulations*, and in fact, many of the aforementioned general-purpose systems have been designed also with this goal in mind.

One technical item most of the mentioned general-purpose systems have in common is that they are *integrators*, possibly providing the probability density to be used for simulation after the integration process. More specifically, the integrand is an input variable, and the integration process is performed by the system itself. Sometimes, however, one has his own Monte Carlo program, and one would like to improve parts of it with adaptive importance sampling “on the fly”, without having to turn it into a function that can be used as the input in an initial “integration phase”. In this write-up, we present the program **PARNI**, which has been designed with exactly this in mind. It consists, apart of a trivial initialization, essentially of three routines which can be called inside a Monte Carlo program; one for the generation of integration points, one to return their weights, and one to collect the evaluated function values at those points. The collection routine builds the probability density with which the integration points are generated, and works independently from the generation routine. This means it can take points from another generation process, and estimate the density following which they are generated. In particular, the function evaluation may be absent and all points may have weight one, so that **PARNI** can serve as a pure density estimator.

The outline of the paper is as follows. In Section 2 an issue concerning the possible overestimation of integration errors when using adaptive importance sampling is addressed. Section 3 explains how the algorithms work on which PARNI is based. In Section 4 a technical detail concerning its architecture is addressed. Section 5 explains how the program is used, and Section 6 gives some examples. Section 7 finally contains the summary.

The program, finally, can be obtained upon request from the author.

2. Adaptive importance sampling

In this section we address an issue in the application of adaptive importance sampling in Monte Carlo integration concerning possible overestimates of the integration error. First of all, we assume that the integration/generation problem has been formulated such that the integration space is a hyper-cube $I_D = [0, 1]^D$ of a certain dimension D . Monte Carlo integration is based on the fact that for a quadratically integrable function f and a sequence of points $(x_i)_{i=1}^n$ in the hyper-cube distributed independently following a probability density g , the distribution of the average of the ratio f/g over the sequence converges, following the Central Limit Theorem, to a Gaussian distribution with expectation value

$$\mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{g(x_i)}\right) = \int_{I_D} f(x) d^D x, \quad (1)$$

and variance

$$\mathbb{V}\left(\frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{g(x_i)}\right) = \frac{1}{n} \left[\int_{I_D} \frac{f(x)^2}{g(x)} d^D x - \left(\int_{I_D} f(x) d^D x \right)^2 \right]. \quad (2)$$

Therefore, the average can be interpreted as an estimate of the integral, with a possible integration error given by the square root of its variance. The variance can be estimated by

$$\frac{1}{n-1} \left[\frac{1}{n} \sum_{i=1}^n \frac{f(x_i)^2}{g(x_i)} - \left(\frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{g(x_i)} \right)^2 \right], \quad (3)$$

and approaches zero for large values of n . The only restrictions on the density g are that it is non-zero on the support of f , and that there is an algorithm available to generate the sequence of points distributed following this density which is not too complex. Of course, the evaluation of the density at the integration points should be feasible, or say, should not be much more complex than the evaluation of the integrand. A first candidate

for the density g is unity on the hyper-cube. Importance sampling is the attempt to construct a density g such that the variance (2) is, for a given value of n , as small as possible.

In adaptive importance sampling, the evaluated integration points are used to update the density during the integration process. In this case, the *Lévi Central Limit Theorem*, which is the basis for the version of Monte Carlo integration presented above, does not hold anymore since it applies only in case the points are *independently identically* distributed. Theoretically, this does not need to be problem since the *Martingale Central Limit Theorem* may still apply [6]. In practice, however, it means that the integration error is possibly unnecessarily overestimated. To see how this happens, assume that a batch of n points has been generated with density g_1 , and another batch with density g_2 which possibly depends on the first batch of data. The straightforward Monte Carlo estimate of the integral would be

$$Y = \frac{X_1 + X_2}{2} \quad \text{with} \quad X_1 = \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{g_1(x_i)}$$

$$\text{and} \quad X_2 = \frac{1}{n} \sum_{i=n+1}^{2n} \frac{f(x_i)}{g_2(x_i)}. \quad (4)$$

The variance of the estimate is given by¹

$$V(Y) = \frac{V(X_1) + V(X_2)}{4}. \quad (5)$$

If the process by which g_2 is updated using the first batch of data is efficient enough, it may happen that $V(X_2) < \frac{1}{3}V(X_1)$, and the result for the integral using both batches may be worse than the result using only the second batch.

The straightforward solution to this problem is to introduce an “optimization phase” in the integration process, in which points are generated and the integrand is evaluated to adapt the density, but do not contribute to the final estimate of the integral. The integral is estimated in a second “integration phase”, in which the optimized density is not changed anymore, and the Lévi Central Limit Theorem applies.

The optimal solution would be to weigh the contributions with the inverse of the variance, *i.e.*, to take

$$Y = \frac{V(X_1)V(X_2)}{V(X_1) + V(X_2)} \left(\frac{X_1}{V(X_1)} + \frac{X_2}{V(X_2)} \right). \quad (6)$$

¹ The correlation $E(X_1X_2) - E(X_1)E(X_2)$ is zero. This is an indication for the martingale structure of a sequence of Monte Carlo estimates using adaptive importance sampling.

This choice gives the minimum variance for Y , however, the variances have to be estimated too, and replacing them by their estimates in the above formula would lead to a biased estimator for the integral of f .

A third option is to weigh contributions of equal sized batches of integration points, which are generated with the same density, with pre-determined weights. **PARNI** in particular generates batches of constant size with the same density, and uses the information from a whole batch at once to update the density, with which the next batch is generated. Weighing the contributions with the order in the sequence, such that the first batch gets a relative weight equal to 1, the second batch gets a relative weight equal to 2 *etc.*, appears to be a safe choice in the sense that the contribution of the early batches is relatively low and the problem sketched above is avoided.

3. The algorithm

The main ingredient of the algorithm **PARNI** uses to build the probability density is to divide the hyper-cube into sub-regions on which the density is constant, in a way similar to other systems, in particular the one in [3]. The second ingredient is to consider the probability density a weighted sum of probability densities restricted to those sub-regions. This is essentially an application of the *multi-channel* method [7]². It excellently suits the task of updating the probability density “on the fly”, by collecting batches of integration points of constant size during the generation of which the density is not changed. The disadvantage is that it requires extra memory during the computation, since the weights of all the sub-densities have to be stored, something which is avoided for example in [3].

To be more specific, at a given stage during the Monte Carlo computation, the density is given by

$$g(x) = \sum_{k=1}^m w_k g_k(x), \quad (7)$$

where each g_k is the constant probability density on a sub-region A_k of the hyper-cube, and the weights w_k are positive and sum up to 1

$$\sum_{k=1}^m w_k = 1. \quad (8)$$

The regions are non-overlapping hyper-rectangles, and their union is the hyper-cube:

$$\forall_{k \neq l} A_k \cap A_l = \emptyset \quad \text{and} \quad \bigcup_{k=1}^m A_k = I_D. \quad (9)$$

² Notice the difference with the application of the multi-channel method in [8], where each channel consists of a probability density on the full hyper-cube.

A triple (A_k, g_k, w_k) will be referred to as *channel* number k .

The creation of new channels is incorporated such that it aims at equidistribution, *i.e.*, it aims at moving the values of the channel weights towards the average $1/m$. Given a batch of evaluated integration points and a method how to update the weights, one of the updated weights, say w_k , will be the largest. Hyper-rectangle A_k is divided into two equal sized pieces, and g_k becomes a weighted sum of two probability densities. The weight of each of these new densities in the full sum of densities simply becomes half of the original updated weight. The division of the original rectangle happens such that it is cut in the middle perpendicularly to the dimension along which it has the longest edges. If it has several dimensions for which the edges are of equal length and the longest, one of them is chosen at random. This way, the hyper-rectangles tend to have the shape of hyper-cubes. After one division, we have a new set of $m' = m + 1$ channels. The division process is repeated until the weight efficiency

$$\frac{1}{m' \max_k w_k} \quad (10)$$

does not increase anymore (notice that m' increases while $\max_k w_k$ decreases with each division).

We have reserved a few options how to update the weights before the division process. In order to achieve variance minimization, the relative weights should be taken proportional to³

$$w_k \propto \sqrt{\text{vol}(A_k)^2 \int_{I_D} f(x)^2 g_k(x) d^D x}, \quad (11)$$

where $\text{vol}(A_k)$ is the volume of sub-region A_k . The integral is estimated by $F_k^{(2)} / F_k^{(0)}$, where

$$F_p^{(k)} = \sum_{x_i \in A_k} f(x_i)^p. \quad (12)$$

We do not calculate the numbers $F_p^{(k)}$ per batch, but calculate them incrementally during the whole adaptation process, by dividing them by 2 if the corresponding channel is divided into two pieces during the creation of channels.

Variance minimization might not lead to the optimal density to be used for simulation purposes; the density with which the simulation is most efficient. For this, a more point-wise recovery of the integrand may be desirable.

³ Realize that g_k is a *probability density*, so it is the indicator function of A_k divided by the volume of A_k .

Better results may be expected by putting the relative weights proportional to

$$w_k \propto \text{vol}(A_k) \int_{I_D} f(x) g_k(x) d^D x. \quad (13)$$

The integral is estimated by $F_k^{(1)}/F_k^{(0)}$.

In the case of **PARNI** used for the task of density estimation, implying that only the full weight coming with each data point is supplied, the channel weights are put proportional to

$$w_k \propto S_k^{(1)}, \quad (14)$$

where

$$S_k^{(p)} = \sum_{x_i \in A_k} s_i^p \quad (15)$$

and s_i is the weight coming with x_i . This way essentially a multi-dimensional histogram with non-equal sized bins is being built.

With the creation of more and more channels, more and more memory is needed, which may not be available anymore at some point. **PARNI** provides the option to set a maximum to the number of channels. If this number is reached, **PARNI** will continue to divide channels, but it will also start to merge channels in order to keep the total number close to the maximum set. Merging is performed with the channels which came from the same parent channel during the division process, and have the smallest sum of weights of all such pairs. The original parent hyper-rectangle is restored, and it gets a channel weight which is the sum of the weights of the pair of daughter channels. Also the values of the quantities $F_k^{(p)}$ and $S_k^{(p)}$ are obtained by adding those of the daughters together.

4. Binary-tree structure

One technical detail which is worth mentioning because it highly contributes to the computational efficiency of the program is that the structure of sub-spaces is, like in [3], organized in a binary-tree structure. So for every hyper-rectangle, the program keeps track of the parent it was created from by the division process, and the daughters created by its own division. This means that the program keeps track of twice as many hyper-rectangles as there are channels, but this disadvantage is fully compensated by the increase in computational efficiency. For example, in order to evaluate the built density at a given point in the hyper-cube, the hyper-rectangle has to be found in which the point is situated. Because of the binary-tree structure this can be done very quickly with a binary search. Also for the generation of a point in the hyper-cube following the built density a binary-tree

search is used. The channel weights are put in a row on the unit interval, and a random number is thrown in. Now the interval, corresponding to the weight in the row, has to be found in which this random number exactly fell. This interval then corresponds to the channel delivering the next integration point.

5. Use of the program

The program is written in **Fortran77** using long names, underscores in names, **enddo** and **do while** statements. It has been designed in such way that an arbitrary number of instances of the algorithm can work in parallel, dealing with completely different integrands living in different numbers of dimensions. Before **PARNI** can be used, some integer type global parameters have to be set. This happens in the header-file **avh_parni.h**. There is **avh_parni_size**, which is a measure of the total amount of memory one is willing to spend on all instances of **PARNI** together. The amount consists of the equivalent of this parameter times 4 double precisions plus 6 integers. Then there is **avh_parni_ncopy**, setting the maximal number of instances of **PARNI**, and finally there is **avh_parni_dim**, setting the largest dimension an instance of **PARNI** may be ordered to deal with.

Next, the user of the program has to implement the contents of the the routine

```
subroutine avh_parni_random(rho ,nn)
```

which is supposed to generate double precision arrays **rho** of arbitrary length **nn** consisting of uniformly distributed (pseudo) random numbers. In practice one line with a call to an external routine of this type will suffice.

An instance of **PARNI** has to be initialized before the Monte Carlo loop with

```
call avh_parni_init(ID ,itask ,ndimen ,nbatch ,nchmax)
```

All variables are input and integers, and the first one, **ID**, is the id of the instance of **PARNI**. This number should be larger than 0 and not larger than the value of **avh_parni_ncopy**. The second one, **itask**, specifies the way the channel weights are updated. The choice **itask=1** corresponds to Eq. (13) and **itask=2** corresponds to Eq. (11). For the task of density estimation, one has to put **itask=11**. The third input variable, **ndimen**, specifies the number of dimensions of the hyper-cube. The fourth input variable, **nbatch**, specifies the size of the batches of data points to be collected before a new adaptation step is performed. The larger this number, the more conservative the algorithm and the less channels will be created for a given total number of data points. From experience we find that this number should best be

of the order of the square-root of the total number of data points that is expected to be encountered. The fifth input variable, `nchmax`, is the maximal number of channels one wishes to reach. If its value is put to 0, the number of channels will grow indefinitely.

Inside the Monte Carlo loop, an integration point distributed following the density built so far is generated by

```
call avh_parni_generate(ID ,xx)
```

The first variable, `ID`, is input and is the id of the instance of **PARNI**. The second variable, `xx`, is output consisting of a double precision array of length `ndimen`, where the latter is the number given on input at the initialization of this instance of **PARNI**. This integration point is stored together with its weight, and these data are returned by

```
call avh_parni_weight(ID ,ww,xx)
```

Here, the weight `ww` and the point `xx` are output. One can obtain the value of the built probability density at any point inside the hyper-cube with the function

```
avh_parni_density(ID ,xx)
```

Here, `xx` is input. The weight coming with an integration point is given by 1 divided by the number obtained by evaluating this function at the generated point. A data point is collected for the adaptation of the density with

```
call avh_parni_adapt(ID ,value ,xx)
```

All variables are input. The third one, `xx`, is the data point in the hyper-cube, and the second one, `value`, is the full weight coming with this point, so including the weight from the generation of **PARNI** itself in the case of importance sampling.

Besides the necessary routines described above, there are some diagnostic routines available. With

```
call avh_parni_marg(ID ,nunit ,label)
```

for each dimension a file is created with two columns of data corresponding to the marginal density in that dimension. In other words, it gives the density obtained when all other dimensions are “integrated out”. The integer input variable `nunit` refers to the unit the file is written to, and `label` is, if it is chosen to be positive, an extra label to the file name. If, for example, `ID=13` and `label=25`, then the file name with the marginal density of dimension number 2 is `PARNI13d2_25`.

In the case when the integration space is 2-dimensional, the density can be visualized with the file created by

```
call avh_parni_plot(ID ,nunit ,label)
```

In the example above, the name of the file name will be `PARNI13p_25`. It is in a format to be used by `gnuplot` [9]:

```
gnuplot> plot 'PARNIp13_25' w l
```

will visualize the 2-dimensional structure of rectangles, whereas

```
gnuplot> splot 'PARNIp13_25' w l
```

will visualize a full 3-dimensional picture of the density.

The number of channels which finally has been reached is send to standard output by

```
call avh_parni_result(ID)
```

Also an integration estimate with error estimate are printed, but the latter may be an overestimation.

Finally we want to remark that the program is distributed over 3 source files with a total length of less than 1500 lines including comments, and that the names of all subroutines, functions and common blocks start with `avh_parni`.

6. Some examples

A typical problem in a general purpose Monte Carlo program for elementary particle phenomenology is that certain variables which have to be generated may be distributed very differently for different processes. An extreme example would be a distribution consisting of a sharp delta-like spike with an unknown position. In order to simulate this situation, we use `PARNI` to integrate a function consisting of a truncated Cauchy-density

$$f(x) = \frac{N(x_0, \varepsilon)}{(x - x_0)^2 + \varepsilon^2}, \quad (16)$$

where $N(x_0, \varepsilon)$ is the normalization such that the correct integral is equal to 1. We take $x_0 = 0.6$ and $\varepsilon = 10^{-5}$, so that relative to the unit interval the density becomes a sharp delta-like spike. In Fig. 1 we show the density built by `PARNI` after 10^4 integration points applied to the adaptation in batches of 10^2 . We did not restrict the number of channels, and `PARNI` created 202 of them. To give a quantitative measure of the result: integration of the integrand with importance sampling using this density (so after its creation, not during) is done with a crude efficiency⁴ of about 23%. Without any

⁴ The average weight divided by the maximal weight in the Monte Carlo process.

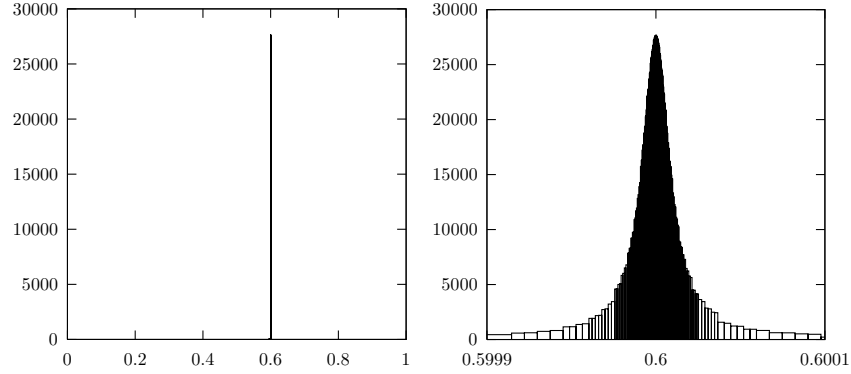


Fig. 1. The density built by PARNI during the integration of the function in Eq. (16) with $\varepsilon = 10^{-5}$. The picture on the right is a zoom-in.

importance sampling, the Monte Carlo integration of this integrand would be performed with a crude efficiency of 0.0037%.

The program **VEGAS** [1] works very well for multi-dimensional integration problems for which the integrand is factorisable over the dimensions, *i.e.*, when the integrand can be seen as a product of integrands with each of them depending on only one of the dimensions. In the second example, we visualize why it is very important to use this knowledge about the integrand when available. We choose an integrand which is the product of truncated Cauchy-densities

$$f(x, y) = \frac{N(x_0, a)}{(x - x_0)^2 + a^2} \cdot \frac{N(y_0, b)}{(y - y_0)^2 + b^2}, \quad (17)$$

with $(x_0, a) = (0.6, 0.02)$ and $(y_0, b) = (0.33, 0.04)$. Now we attack the integration problem in two different ways. In the first approach, we let one instance of PARNI deal with the full 2-dimensional problem at once. A rudimentary Monte Carlo program in this approach would look like

```
call avh_parni_init(1,itask,2,nbatch,nchmax) ! ndimen=2
result = 0d0
do iev=1,nev
    call avh_parni_generate(1,x)
    weight = integrand(x(1),x(2))
    &      / avh_parni_density(1,x)
    call avh_parni_adapt(1,weight,x)
    result = result+weight
enddo
result = result/nev
```

In the second approach, we let two instances of **PARNI** deal with the problem, each of them with one of the dimensions. A rudimentary Monte Carlo program in this approach would look like

```

call avh_parni_init(1,itask1,1,nbatch1,nchmax1) ! ndimen=1
call avh_parni_init(2,itask2,1,nbatch2,nchmax2) ! ndimen=1
result = 0d0
do iev=1,nev
  call avh_parni_generate(1,x1)
  call avh_parni_generate(2,x2)
  weight = integrand(x1,x2)
  &      / avh_parni_density(1,x1)
  &      / avh_parni_density(2,x2)
  call avh_parni_adapt(1,weight,x1)
  call avh_parni_adapt(2,weight,x2)
  result = result+weight
enddo
result = result/nev

```

In order to compare the two approaches in a fair way, we demand that the sum of the number of channels used by the two instances in the latter approach should be the same as the number of channels used by the one instance in the former approach. Fig. 2 depicts the density built using 10^5 integration points in batches of 316 in the first approach. The number of channels is 200. The picture on the right is the top-view of the picture on the left, and shows the structure of rectangles. The picture on the left of Fig. 3 shows the marginal densities in the two dimensions for this case. The picture on the right, on the other hand, shows the density built by each of the instances of **PARNI** in the second approach. In this picture, the densities look much nicer because more “bins” are available, namely 100 for each of

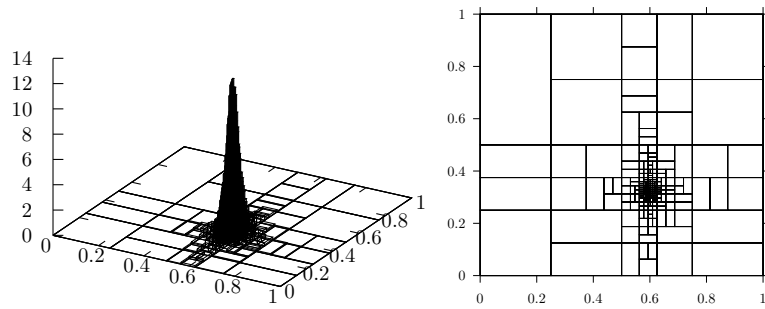


Fig. 2. Density (left) and structure of rectangles (right) built during the integration of the integrand in Eq. (17).

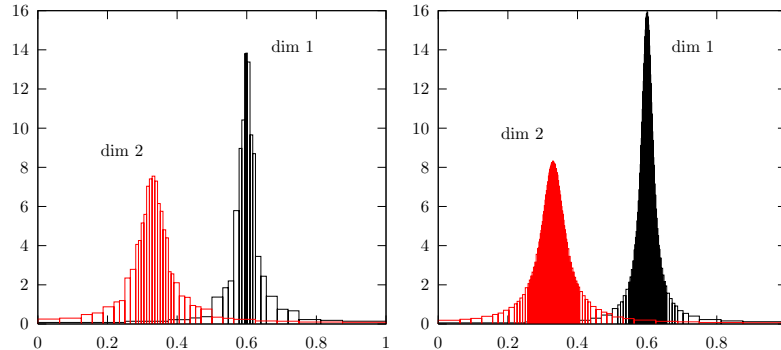


Fig. 3. Marginal densities obtained with one instance of **PARNI** dealing with the full problem (left) and with the **VEGAS**-approach (right), using the same total number of channels.

them, whereas in the picture on the left this number is only of the order of $\sqrt{200}$. Since we are dealing with a factorisable integrand, we know that the marginal densities are sufficient to capture all information about the integrand, and we can thus confirm visually that the second, **VEGAS**-like, approach works much better. This is also shown more quantitatively by the crude efficiency of the integration of the integrand using the densities (again after their creation, not during): 66% for the **VEGAS**-approach against 15% for the other approach.

When the integrand is not factorisable, **VEGAS** does not work so well, and for example [3] has particularly been designed to deal with this problem. Also **PARNI** can deal with non-factorisable integrands, and as an example, we give some results for a 2-dimensional integration problem in which the support of the integrand is concentrated around a circle. It is proportional to

$$f(x, y) \propto \exp\left(-\frac{(r-c)^2}{d^2}\right) \quad \text{with} \quad r = \sqrt{(x-a)^2 + (y-b)^2}, \quad (18)$$

and we put $a = 0.57$, $b = 0.62$, $c = 0.3$ and $d = 0.01$. The structure of rectangles built by **PARNI** after 10^6 integration points in batches of 10^3 is shown in Fig. 4. We did not put a restriction on the number of channels to be created, and **PARNI** created 1819 of them. If we try to deal with this integrand using the **VEGAS**-like approach, around 2800 channels are created for either dimension. Still, the integration is less efficient: with the same number of integration points, the **VEGAS**-like approach reaches an estimate of the relative error of 0.24%, whereas the approach with one instance dealing with the full problem reaches an estimated 0.081%. In comparison, without any importance sampling, one reaches an estimated relative error of 0.43%.

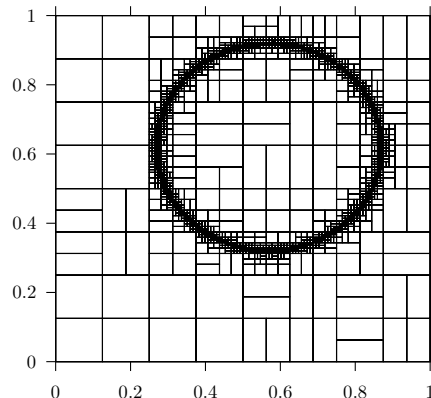


Fig. 4. Structure of rectangles built during the integration of the integrand in Eq. (18).

7. Summary

We presented the program **PARNI**, a practical aid for importance sampling in Monte Carlo integration. It adapts automatically to integrands on the unit hyper-cube of in principle any dimension, and can therefore be considered to be of the general-purpose type. However, it does not ask for the integrand as an input variable, but provides a number of routines which should be incorporated into an existing Monte Carlo program, so that the optimization happens “on the fly” while the Monte Carlo is running. The generation of integration points and the adaptation of the probability density following which they are generated happen independently, and the program can also be used only to adapt the probability density using data from another source. In other words, it can also be used as a pure density estimator. The program can be obtained upon request from the author.

REFERENCES

- [1] G.P. Lepage, *J. Comput. Phys.* **27**, 192 (1978).
- [2] S. Kawabata, *Comput. Phys. Commun.* **88**, 309 (1995).
- [3] S. Jadach, *Comput. Phys. Commun.* **152**, 55 (2003) [arXiv:physics/0203033].
- [4] T. Hahn, *Nucl. Instrum. Methods* **A559**, 273 (2006) [arXiv:hep-ph/0509016].
- [5] P. Nason, arXiv:0709.2085 [hep-ph].
- [6] A.F.W. van Hameren, arXiv:hep-ph/0101094.
- [7] R. Kleiss, R. Pittau, *Comput. Phys. Commun.* **83**, 141 (1994).
- [8] T. Ohl, *Comput. Phys. Commun.* **120**, 13 (1999) [arXiv:hep-ph/9806432].
- [9] <http://www.gnuplot.info/>