UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Fisica

NEW MONTE CARLO ALGORITHMS FOR MULTI-DIMENSIONAL INTEGRATION WITH HARDWARE ACCELERATION

Relatore:
Dott. Stefano CARRAZZA
Correlatore:
Dott. Juan M. CRUZ-MARTINEZ

Elaborato finale di:
Andrea PASQUALE
Matricola: 960202

Codice PACS:
02.70.-c

Anno Accademico 2020/2021

Contents

Introduction

Chapter 1

Multi-dimensional integration in HEP

In this chapter we focus at first on Monte Carlo techniques applied to the problem of multi-dimensional numerical integration. We discuss the two main methods which involve importance sampling and stratified sampling. Secondly we give a brief overview on High Energy Physics (HEP) arguments, with particular attention on the computation of physical observables as a series of perturbative terms which involve high-dimensional integrals. Finally we present the state-of-art of MC integration applied to HEP discussing the current problems and limitations which is facing the High-Luminosity LHC programme.

1.1 Monte Carlo methods for multi-dimensional integration

Monte Carlo (MC) methods are a powerful tool which can provide the answer to a problem by simply running a simulation on the system studied. In the field of multi-dimensional integration techniques MC methods are the solution of choice, since contrary to the standard numerical integration formulas the error on the integral does not scale with the dimension [1].

1.1.1 A first example of Monte Carlo integration

In the case of multi-dimensional integration we are interested in performing the following integral I:

$$I = \int_V f(\mathbf{x}) d\mathbf{x} \tag{1.1}$$

where V is the domain of the integration and f is a function of n variables $\mathbf{x} = (x_1, x_2, ..., x_n)$.

When performing the integral I the simulation comes down to a sampling of the integrand function. First we need to generate a set of random points $\mathbf{x}_i$ which belong to the integration domain V. The simplest way to performe the sampling is to pick random points uniformly distributed in the volume V.

An estimate of the integral using N random points can be given as:

$$I \approx I_{\mathrm{MC}} = V \frac{1}{N} \sum_{\mathbf{x}_i \in V} f(\mathbf{x}_i) = V \langle f \rangle \tag{1.2}$$

where $\langle f \rangle$ denotes the arithmetic average of the function f.

I_{MC} is a random number, whose value depends on the sampled points, whose mean is given by the exact value of the integral I and the variance is given by:

$$\sigma_I^2 = \frac{1}{N}\Big[V \int_V f^2(\mathbf{x}) d\mathbf{x} - I^2\Big] \tag{1.3}$$

This variance is asymptotically related to the variance of the random value I_{MC}, therefore we can estimate the value of σ_I^2 in the limit of large N as:

$$\sigma_I^2 \approx \sigma_{\text{MC}}^2 = \frac{1}{N-1}\Big[V^2\langle f^2\rangle - I_{MC}^2\Big] \tag{1.4}$$

In both cases we can observe that the standard deviation decreases as the sample increased as $N^{-\frac{1}{2}}$ regardless of the dimension of the volume V. This is a remarkable feature typical of MC integration which differs from the standard quadrature techniques where the error increases with the dimension.

MC integration can also deal with another problem of quadrature integration: complicated boundaries. Suppose that we need to integrate a function h over some complicated region H, for which the random sampling becomes challenging. In this particular case we can easily overcome such problem by replacing the region H with a new volume G that includes H more suitable for the process of sampling. After that all we need to do is to replace also the function h with a new function g defined as:

$$g(\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{if } \mathbf{x} \in H \\ 0 & \text{if } \mathbf{x} \in G - H \end{cases} \tag{1.5}$$

Obviously one should try to make the new region G not too oversized with respect to H, because every point $\mathbf{x} \in G - H$ will contain no information about the integrand. Therefore the number of effective points used during the sampling N will reduce raising the error in Eq.(1.4).

This first MC integrator has a few disadvantages.

Firstly we have already observed that the error decreases as the square root of the number of sampled points, this implies that if the accuracy requirements are high we will need to increase the size of the sampling. Dealing with huge sizes of sampled points could be, even for a computer with large memory and a fast processor, challenging and we expect that the process of sampling could take a significant amount of time.

Secondly one will probably have to work with large samples, even if the accuracy requirements are modest, when the dimensionality of the integration domain is high. The reason being that if the integrand function is peaked in a small region compared to the volume V, we will need to generate more random points to make sure that the peak is correctly identified; such process of finding the peaks becomes more challenging in large number of dimensions. This can be seen for example considering the ratio of the volume of a D dimensional hypersphere with unity radius to the D dimensional hypercube with a side of twice the unity radius , which vanishes as the D goes to infinity as:

$$\frac{V_{\text{hypersphere}}}{V_{\text{hypercube}}} = \frac{1}{2^D}\frac{\pi^{\frac{D}{2}}}{\Gamma\left(\frac{D}{2}+1\right)} \approx \left(\frac{\sqrt{\pi}}{2}\right)^D \xrightarrow{D\to\infty} 0 \tag{1.6}$$

where $\Gamma(x)$ denotes the famous Gamma function.

In literature this phenomenon known as the *curse of dimensionality.*

1.1.2 Reducing the variance

In the field of MC integrators the main focus is to improve the error estimate in order to achieve more precise results using less number of events. Over the years and also lately with the advent of Machine Learning, several techniques have been proposed, implemented and applied to solve complex multi-dimensional integrals [4, 3, 2].

In literature there are two main ways of reducing the variance: importance sampling and stratified sampling.

Importance Sampling

In the naive MC integrator the sampling was performed by picking random points uniformly distributed in the integration volume. We have already discussed that if the integrand has a peak in a small region it will be challenging to find it especially because we are selecting points from a uniform distribution.

Suppose that the points $\mathbf{x}_i$ are chosen within the integration volume V with a probability density p correctly normalized:

$$\int_V p(\mathbf{x}) d\mathbf{x} = 1 \tag{1.7}$$

We can calculate the integral I as:

$$I = \int_V f(\mathbf{x}) d\mathbf{x} = \int_V \frac{f(\mathbf{x})}{p(\mathbf{x})} p(\mathbf{x}) d\mathbf{x} = \int_V \frac{f(\mathbf{x})}{p(\mathbf{x})} dP(\mathbf{x}) \tag{1.8}$$

where in the last step we used the transformation $d\mathbf{x} = dP(\mathbf{x})/p(\mathbf{x})$, with $P(\mathbf{x})$ the cumulative distribution of $p(\mathbf{x})$.
From Eq.(1.8) we can deduce that to compute the integral I, instead of using a uniform sampling of the function f, we can also perform a non-uniform sampling of the function f/p in the same integration volume V.
The integral and the relative error will be given by [2]:

$$I \approx I_{\mathrm{MC}} = V \langle f/p \rangle_P \tag{1.9}$$

$$\sigma_I^2 \approx \sigma_{\mathrm{MC}}^2 = \frac{V^2}{N-1} \left[\langle (f/p)^2 \rangle_P - \langle f/p \rangle_P^2 \right] \tag{1.10}$$

where $\langle \rangle_P$ denotes the average taken with respect to the non-uniform distribution $p(\mathbf{x})$.

This is the concept of *importance sampling*: by changing the distribution of the sampling we can reduce the variance by choosing a suitable $p(\mathbf{x})$ [5, 4, 2].

What is the best choice for the sampling density p? It can be shown that by minimizing the variance, as a functional of sampling density p, the optimal choice for p is to be proportional to $|f|$. This choice is not surprising, in fact by replacing p with $|f|$ in Eq.(1.10) we get a vanishing variance. Moreover, in order to satisfy the normalization requirement in Eq.(1.7) the exact solution for p is:

$$p = \frac{|f|}{\int_V |f| d\mathbf{x}} \tag{1.11}$$

As we can see we arrive at a paradox since the optimal sampling density requires

the knowledge of $\int_V |f(\mathbf{x})| d\mathbf{x}$ which is the integral that we are trying to compute! Therefore to minimize the variance the aim is to find a function p that resemble the shape of the integrand function f, this is usually done using adaptive recursive strategies which can provide a better sampling distribution using the sampled points.

Stratified Sampling

Another technique which is a standard one in literature is based on the idea of "stratified sampling" [6, 1]. We have already observed that we can estimate the variance of our integral by computing the variance of the random variable I_{MC}; we can exploit this relation by focusing on the average value of the function f over the domain V, denoted by $\langle\langle f \rangle\rangle$, and the correspoding MC estimate using a uniform sampling $\langle f \rangle$:

$$\langle\langle f \rangle\rangle = \frac{1}{V} \int_V f(\mathbf{x}) d\mathbf{x} \qquad \langle f \rangle = \frac{1}{N} \sum_{\mathbf{x}_i} f(\mathbf{x}_i) \tag{1.12}$$

We can now rewrite Eq.(1.4) as

$$\mathrm{Var}(f) \equiv \langle\langle f^2 \rangle\rangle - \langle\langle f \rangle\rangle^2 = \frac{\langle f^2 \rangle - \langle f \rangle^2}{N} \equiv \frac{\mathrm{Var}(\langle f \rangle)}{N} \tag{1.13}$$

Suppose we divide the volume V into two subvolumes V_a and V_b, chosen equal and disjoint, and we sample exactly $N/2$ points in each subvolume. We can formulate another estimator for the mean value of the function, $\langle\langle f \rangle\rangle$ as the arithmetic mean between the sample average in the two half -regions:

$$\langle f \rangle' \equiv \frac{1}{2}(\langle f \rangle_a + \langle f \rangle_b) \tag{1.14}$$

where $\langle \rangle_a$ denotes the MC estimate of the function for the $N/2$ points belonging to the subvolume V_a and similarly for $\langle \rangle_b$. The variance of the estimator in Eq.(1.14) can be easily computed as:

$$\mathrm{Var}(\langle f \rangle') = \frac{1}{4}\big[\mathrm{Var}(\langle f \rangle_a) + \mathrm{Var}(\langle f \rangle_b)\big] \tag{1.15}$$

$$= \frac{1}{4}\left(\frac{\mathrm{Var}_a(f)}{N/2} + \frac{\mathrm{Var}_b(f)}{N/2}\right) \tag{1.16}$$

$$= \frac{1}{2N}\big[\mathrm{Var}_a(f) + \mathrm{Var}_b(f)\big] \tag{1.17}$$

where $\mathrm{Var}(\langle f \rangle)_i = \langle\langle f^2 \rangle\rangle_i - = \langle\langle f \rangle\rangle_i^2$ denotes the variance of f limited to the subvolume V_i.

One could ask how the variance in subvolumes V_i is related to the variance of f in the whole integration domain V. This can be easily done using the additivity of the integral, the result is known in literature as the parallel axis theorem:

$$\mathrm{Var}(f) = \frac{1}{2}[\mathrm{Var}_a(f) + \mathrm{Var}_b(f)] + \frac{1}{4}(\langle\langle f \rangle\rangle_a - \langle\langle f \rangle\rangle_b)^2 \tag{1.18}$$

As we can see the stratified sampling gives a variance which is always equal or smaller than the variance of the naive MC integrator. In particular whenever the function f behaves differently in the two regions V_a and V_b, the variance of f increases as $(\langle\langle f \rangle\rangle_a - \langle\langle f \rangle\rangle_b)^2$ and such difference doesn't affect the $\mathrm{Var}(\langle f \rangle')$ given by Eq.(1.17). We can generalize the previous formulas by adding the possibility to sample a specific

number of points from each subvolume. Considering again the subvolumes V_a and V_b we can suppose to sample the first one with N_a points and the second one with N_b with the constraint that $N = N_a + N_b$.
It can be shown that in this case the variance is minimized when [6]:

$$\frac{N_a}{N} = \frac{\sqrt{\mathrm{Var}_a(f)}}{\sqrt{\mathrm{Var}_a(f)} + \sqrt{\mathrm{Var}_b(f)}} \tag{1.19}$$

Consequently we can see that the number of sample in a particular subvolume is proportional to the standard deviation of the samples in that subregion, which is quite expected. If there is a region with high variance we will need more points in order to have a better knowledge of the behaviour of the integrand function.
With the aim of taking full advantage from stratified sampling, the standard procedure is to divide the integration domain in several subvolumes each one with a different number of samples. Again by minimizing the variance we obtain that the number of samples in each subregion must be proportional to the standard deviation in that particular subregion.
Stratified sampling can struggle in higher dimensions due to the large number of subregions. In fact if we divide the integration volume in M subvolumes for each dimension we will end up with a total of M^d total subregions. Considering that in order to compute the variance in each region we need at least two points, we will need a total of $2M^d$ sampled points., which can become challenging.

1.2 HEP and multi-dimensional integration

One of the main fields of physics which requires computing multi-dimensional integrals is High Energy Physics (HEP). In this section we give a brief overview on some aspects of quantum field theory [7, 8], in particular we show that a prediction for a physical observable can be made as a series of perturbative terms which involve the evaluation of high dimensional integrals.

Finally we present shortly the quantum theory of strong interactions known as Quantum Chromodynamics (QCD) [9, 10, 11, 12].

1.2.1 Cross Section and Decay Rates

The experiments performed at LHC are aimed at probing the behaviour of elementary particles, which can be considered in a relativistic regime due to the high energies required. These are usually scattering experiments in which two beam of particles with well defined momenta collide creating new particles which are detected and measured.

The probability of finding a particular final state can be expressed in terms of a physical observable, the cross section. Suppose that we have two species of particle, $\mathcal{A}$, at rest with density $\rho_{\mathcal{A}}$, and $\mathcal{B}$ moving at velocity v towards $\mathcal{A}$ with density $\rho_{\mathcal{B}}$. Imagine also that we can measure the length of the two beam of particles $l_{\mathcal{A}}$ and $l_{\mathcal{B}}$.

We expect that the total number of scattering events must be proportional to the cross-sectional area A common to the two beams and all the previous quantities; the cross section, σ, is defined indeed as:

$$\sigma \equiv \frac{\text{Number of scattering events}}{A\rho_\alpha \rho_\beta l_\alpha l_\beta} \tag{1.20}$$

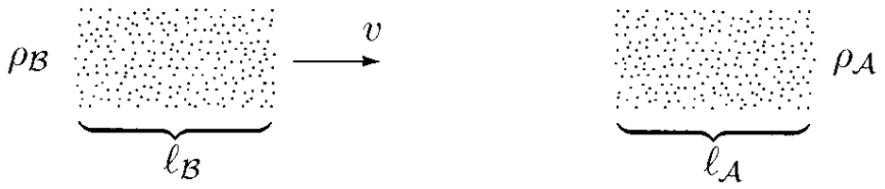

Figure 1.1: Symbolic representation of a scattering process. Image from Ref[7].

We also wish to measure the momenta of the outgoing particles, which are expected to be infinitesimal. In order to do so we can define a differential cross section $d\sigma/(d^3p_1...d^3p_n)$ such that:

$$\int_\Omega \frac{d\sigma}{d^3p_1...d^3p_n} d^3p_1...d^3p_n = \sigma|_\Omega \tag{1.21}$$

where $\sigma|_\Omega$ gives the cross section for scattering in the region of final-state momentum space Ω. Obviously the final state momenta are not all independent.

Firstly the total four-momentum of the incoming particles must be equal to the total momentum of the outgoing particles by four-momentum conservation. Secondly each particle detected will have a given mass (or zero for massless ones) which fixes the four momentum components, $p_i^2 = m_i^2$.

The majority of the particles produced during the collision are unstable, that is the lifetime τ of such particles is so short that they cannot be detected by the experimental apparatus. Nevertheless we know that an unstable particle decays in other particles species some of which can be detected. For an unstable particle δ we can define a new observable, the decay rate Γ defined as

$$\Gamma = \frac{\text{Number of decays per unit time}}{\text{Number of } \delta \text{ particles present}} \tag{1.22}$$

The lifetime can be computed as the reciprocal of the sum of its decay rates into all possible final states.

This two physical observables can be computed theoretically by using the scattering matrix S firstly introduced by Heisenberg.

1.2.2 The S-matrix formalism

When particles collides during a scattering experiments they interact according to the fundamental interactions describes by the Standard Model (SM). In particular the SM is a quantum field theory that explains how three of the four fundamental forces (strong, weak and electromagnetic) can be described through the exchange of particles called bosons.

The beams of incident particles in a quantum field theory is treated as a quantum state $|\phi_A\phi_B\rangle_{\text{in}}$, in the same way the final state detected will be denoted by $|\phi_1\phi_2\dots\phi_n\rangle_{\text{out}}$. Both the initial and final states can be expressed as linear superpositions of eigenstates of the free theory, i.e. with definite momenta, constructed in the far past and in the far future respectively. For example for the final state we can write:

$$|\phi_1\phi_2...\phi_n\rangle_{\text{out}} = \left(\prod_{i=1}^{n} \int \frac{d^3p_i}{(2\pi)^3} \frac{\phi_i(\mathbf{p}_i)}{\sqrt{2E_i}}\right) |\mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n\rangle_{\text{out}} \tag{1.23}$$

The probability of scattering, which is connected to the overlap between the initial and the final state, can be related to the transition amplitudes between the eigenstates of momenta created in the far past *in* and in the far future *out*

$$ {}_{\text{out}}\langle \mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n|\mathbf{k}_1\mathbf{k}_2\rangle_{\text{in}} = \lim_{T\to\infty} \langle \mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n|e^{-iHT}|\mathbf{k}_1\mathbf{k}_2\rangle \equiv \langle \mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n|S|\mathbf{k}_1\mathbf{k}_2\rangle \quad (1.24) $$

where we have written explicitly the time evolution. As we can see the braket between the two asymptotic states can be rewritten as the braket between two momenta eigenstates by the limit of a sequence of unitary operators that is called *S-matrix*:

$$ {}_{\text{out}}\langle \mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n|\mathbf{k}_1\mathbf{k}_2\rangle_{\text{in}} = \langle \mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n|S|\mathbf{k}_1\mathbf{k}_2\rangle \quad (1.25) $$

If we are studying a non-interacting theory the S matrix is simply the identity matrix, since state of definite momentum are eigenstate of the free-field Hamiltonian, which are orthogonal. If we consider an interacting theory we can rewrite S as the identity matrix plus a non-trivial matrix T: $S = \mathbf{1} + iT$.
In literature the matrix element of the T matrix is written as:

$$ \langle \mathbf{p}_1\mathbf{p}_2...\mathbf{p}_n|iT|\mathbf{k}_1\mathbf{k}_2\rangle \equiv (2\pi)^4\delta^{(4)}\Big(k_1 + k_2 - \sum_{i=1}^{n} p_i\Big)\cdot i\mathcal{M} \quad (1.26) $$

where we introduced the invariant matrix element $\mathcal{M}$ by removing a factor that reflects the four-momentum conservation. The previous separation is useful since $\mathcal{M}$ contains all the informations regarding the interaction, while all the other factors depends merely on the kinematics of the process.

In particular, it can be shown that the differential cross section $d\sigma$ can computed using the square modulus of the invariant matrix element and the kinematic quantities of the particles involved in the collision:

$$ d\sigma = \frac{1}{4E_AE_B|v_A - v_B|}\int d\Pi_n \times |\mathcal{M}(k_A, k_B \to p_1, ..., p_n)|^2 \quad (1.27) $$

where $|v_A - v_B|$ is the relative velocity between the two beams and the integral over the final momenta is of the form:

$$ \int d\Pi_n = \left(\prod_{i=1}^{n}\int \frac{d^3p_i}{(2\pi)^3}\frac{1}{2E_i}\right)(2\pi)^4\delta^{(4)}\Big(k_A + k_B - \sum_{i=1}^{n} p_i\Big) \quad (1.28) $$

which corresponds to an integral involving $3n$ variables where n is the number of particles in the final state.

There is also a formula to compute the differential decay rate $d\Gamma$; it can be obtained considering an initial state with a single unstable particle in his rest frame that decays in n outgoing particles.

$$ d\Gamma = \frac{1}{2m_A}\left(\prod_{i=1}^{n}\int \frac{d^3p_i}{(2\pi)^3}\frac{1}{2E_i}\right)(2\pi)^4\delta^{(4)}\Big(k_A - \sum_{i=1}^{n} p_i\Big) \times |\mathcal{M}(k_A \to p_1, ..., p_n)|^2 \quad (1.29) $$

We have two equations to compute the differential cross section and the differential decay rate which involve the square modulus of the invariant matrix element $\mathcal{M}$. In the next subsection we show how $\mathcal{M}$ can be calculated pertubatively using Feynman diagrams.

1.2.3 Feynman diagrams

The invariant matrix element $\mathcal{M}$ was firstly introduced in Eq.(1.26) in the overlap between two momenta eigenstates with the S matrix. Such matrix has an exact solution which is expressed as a series of terms, known as Dyson expansion:

$$S = \sum_{n=0}^{\infty} S^{(n)} = \sum_{n=0}^{\infty} = \frac{(-i)^n}{n!} \int \cdots \int d^4x_1 ... d^4x_n \mathcal{T}\{\mathcal{H}_I(x_1) \dots \mathcal{H}_I(x_n)\} \tag{1.30}$$

where $\mathcal{T}\{\mathcal{H}_I(x_1) \dots \mathcal{H}_I(x_n)\}$ denotes the normal ordered product of the interacting Hamiltonian densities $\mathcal{H}_I(x_1) \dots \mathcal{H}_I(x_n)$ and the integral is over all space-time.

The expansion is reliable only if we can treat the interacting hamiltonian density, $\mathcal{H}_I$, as a perturbation which is the case for the majority of the theories studied. For example in QED the interaction is proportional to the fine structure constant $\alpha \approx 1/137$, thus the perturbative expansion makes sense.

Thanks to Wick's theorem it is possible to rewrite the time-ordered product in a different way which can be represented graphically using Feynman diagrams. To be more specific Feynman diagrams are graphs in which each part from the lines to the vertices is linked to a mathematical expression. The set of rules that allows us to pass from the pictorial representation to the correct mathematical formula are called Feynman rules, and are easily obtainable from the Lagrangian of the theory.

As for the matrix S also the matrix element $\mathcal{M}$ can be written as a perturbative expansion:

$$\mathcal{M} = \sum_{n=1}^{\infty} \mathcal{M}^{(n)} \tag{1.31}$$

where the contribution $\mathcal{M}^{(n)}$ comes from the nth order perturbation term $S^{(n)}$ and can be obtained by drawing all topologically different, connected Feynman diagrams which contain n vertices and the correct number of external lines.

Among the different diagrams there are some which may contains some loop lines, which correspond to quantum corrections. The Feynman rules tell us that for each loop line we must add an integration over the momentum k of that line, since such momentum is not fixed by the process.

The perturbative expansion of $\mathcal{M}$ can be seen as a sum of integral with increasing dimension. The first contribution comes from the diagrams that contain no loops, which is referred as *tree-level* since such diagram obeys the definition of a tree in graph theory. The second contribution contribution will come from the one-loop diagrams in which the dimension of the integral increases since we are also integrating over the loop momentum. By repeating this process we can express $\mathcal{M}$ as:

$$\mathcal{M} = \mathcal{M}^{\text{tree}} + \mathcal{M}^{\text{1-loop}} + \mathcal{M}^{\text{2-loop}} + \dots \tag{1.32}$$

It is well known in the field of quantum field theories that the diagrams which involves loop corrections can lead to divergent contributions, in the limit where the momenta of the loop particles become large. These divergences are known as ultra-violet (UV) divergences and in renormalizable theories, like QCD, they can be removed by a modification of the continuum limit, at least in perturbation theory. If the considered theory is *renormalizable* there are specific techniques to handle this type of divergences which belong to the field of *renormalization*.

Infra-red IR divergences also appear in perturbation theory for the S-matrix of theories such as QCD and QED that have massless fields. This type of infinities

arise when we consider radiative corrections to the tree-level diagram, i.e. when other particles with vanishing energy are emitted other than the final state particles.

For example in QED we can start from the process $e^-e^+ \to \mu^+\mu^-$ at tree-level and then consider a radiative correction which involves the emission of a photon from one of the two final-state quarks. The second order radiative correction will involve the emission of two photons and so on. Finally we can arrive at an expression to include all this radiative corrections of the form:

$$\mathcal{M}^{\text{inclusive}} = \mathcal{M}^{\text{tree}} + \mathcal{M}^{\text{1-leg}} + \mathcal{M}^{\text{2-leg}} + \dots \tag{1.33}$$

where $\mathcal{M}^{\text{1-leg}}$ corresponds to the emission of a single particle, $\mathcal{M}^{\text{2-legs}}$ emission of two particles and so on. We can observe that also these sequences involves integral of increasing dimensions since by adding one extra particle to the final states we go from an $3n$ dimensional space to a $3n+3$ space, always considering a final state of n particles.

IR singularities also appear in loop diagrams and thanks to the Kinoshita-Lee-Nauenberg (KLN) theorem it is possible to prove that the singularities of the real emissions and those one of the loop diagrams cancel each other out order by order in perturbation theory.

We have therefore shown that when computing a physical observable in quantum field theory we obtain a perturbative expansion which involves both loop diagrams and radiative correction which is finite. We can denote the first non vanishing term in this series as the *lowest order* (LO) term, the next term will be the *next-to-leading-order* term (NLO) and so no. For a physical observable $\mathcal{O}$ the perturbative series will be of the form:

$$\mathcal{O} = \mathcal{O}^{\text{LO}} + \mathcal{O}^{\text{NLO}} + \mathcal{O}^{\text{NNLO}} \tag{1.34}$$

where $\mathcal{O}^{\text{NNLO}}$ denotes the *next-to-next-to-leading-order* contribution.

1.2.4 Basics of QCD

At the Large Hadron Collider (LHC) we are particularly interested in processes which involves hadronic collisions. In the SM the theory which describe the hadronic interactions is known as Quantum Chromodynamics (QCD), which is a quantum field theory based on the gauge symmetry of the non-Abelian group $SU(3)$.

The hadrons, however, are not the fundamental quanta of the theory; they are described as bound states of subnuclear fermions known as quarks q and the relative anti-particles, the anti-quarks $\bar{q}$. The are two possible bound states observed: mesons, which are made by a quark-anti-quark couple $q\bar{q}$, and the baryons, which are described as a bound state of three quarks qqq.

In order to use the formalism of the S matrix and Feynman diagrams we need to be able to treat the interacting density Hamiltonian of QCD as a perturbation. The interacting term is proportional to the strong coupling constant α_s, thus a perturbative approach is reliable only if we are at scale μ s.t. $\alpha_S(\mu) \ll 1$. It has been proven both theoretically and experimentally that the strong coupling has the peculiar characteristic of decreasing at UV scales, which is known as asymptotic freedom.

If we consider an hard scattering process between two hadrons usually one compute firstly the differential cross section at a parton level, since due to the asymptotic freedom we can consider the partons almost as free particles. The differential cross

section , as we already saw in Eq.(1.27), will be proportional to the phase-space density $d\Phi_n$ and the squared matrix elements $|\mathcal{M}|^2$:

$$d\hat{\sigma}(p_1,\ldots,p_n;Q) \sim |\mathcal{M}(p_1,\ldots,p_n)|^2 d\Phi_n(p_1,\ldots,p_n;Q) \tag{1.35}$$

where Q denotes the renormalisation scale of the hard-process. Since we are at a hard-scale Q we can compute $d\hat{\sigma}$ as a perturbative series in the strong coupling $\alpha_s(Q)$

$$d\hat{\sigma} = d\hat{\sigma}^{\mathrm{LO}} + \alpha_s(Q) d\hat{\sigma}^{\mathrm{NLO}} + \alpha_s^2(Q) d\hat{\sigma}^{\mathrm{NNLO}} + \ldots \tag{1.36}$$

We can then relate the differential partonic cross section with the differential hadronic cross section by using the factorization theorem, when applicable, that allows us to subdivide the calculation of an observable into a short-distance part and an approximately universal long-distance part. The short-distance part in our case is the partonic cross section $d\hat{\sigma}$ while the long-distance part includes the Parton Density Functions (PDF) $f_{a/h}(x_a, \mu_F)$, which can be interpreted, to a first approximation, as the probability density of finding the parton a in the hadron h with a fraction x_a of the hadron's momentum when probed at a scale μ_F which is known as the *factorization scale*.

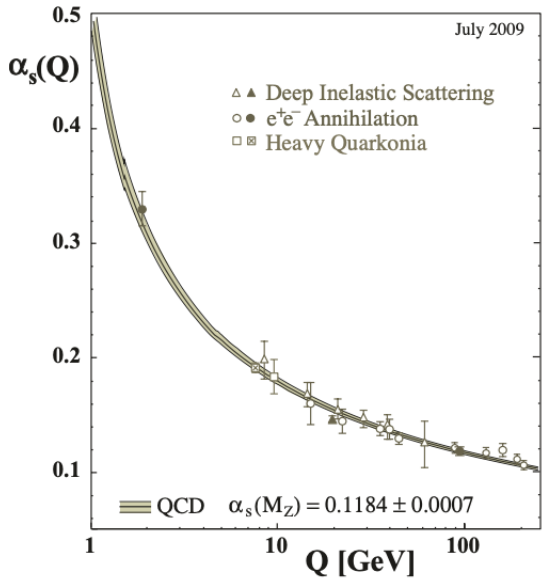

Figure 1.2: Measurements of the strong coupling constant α_s

$$d\sigma = \sum_{a,b} \int_0^1 dx_a dx_b \sum_F \int d\Phi_F f_{a/h_1}(x_a,\mu_F) f_{b/h_2}(x_b,\mu_F) d\hat{\sigma}_{ab\to F} \tag{1.37}$$

where the sum over a and b runs over all the partonic constituents of the two hadrons h_1 and h_2 and the inner sum over F runs over all the possible final states F

1.2.5 Need for MC integration in HEP

The theoretical framework presented in the previous sections shows us that to make a prediction we need to solve complicated integrals. The complexity of the integrals

will increase as we approach higher levels of precision since there will be more Feynman diagrams to consider. Moreover, the Feynman diagrams will be more intricate leading to difficult mathematical expressions.

The aforementioned integrals can be solved analytically only if we are dealing with the first perturbative terms of a simple process. Eventually one can use also tools such as the language program `Mathematica` [13] to solve such integrals analytically. However the majority of the calculations performed nowadays involves integrals that cannot be solved analytically or for which the analytical result is often not known.

This is a common problem also in other sectors of physics. The solution is to switch to a numerical approach in order to solve the integrals. This is why we need to use numerical techniques such as Monte Carlo integrators which are particularly suited for complex multi-dimensional integration problems.

1.3 Modern techniques and limitations

Thanks to the technological development at LHC we are able to obtain experimental data with a very high precision. To test whether the SM can explain and predict correctly these data we need to compare them with theoretical predictions that have the same accuracy. This leads us to the problem of increasing the precision of the theoretical results.

As we saw in the previous section, we can obtain an expression for physical observables such as the differential cross section as a series of terms which involves the calculation of high-dimensional integrals. The most relevant technique used to solve these integrals are MC methods, due to the high-dimensions as expected. The problem of reaching higher accuracy is therefore related to the possibility of reducing the MC estimate of the variance of the considered integral.

In section 1.1.2 we already discussed some MC techniques that enable us to reduce the variance, however, in the particular case of HEP integrands the task of variance reduction can become problematic.

1.3.1 Problems of HEP integration

One of the first problems when predicting physical observable based on a quantum field theory, such as QCD, is the fact we need to compute the observable beyond the LO term in order to match the experimental data.

Even if we start with a process with a relatively small dimensional phase space at LO, the next terms in the perturbation series will involve the computation of integrals with more complicated functions which are defined in a higher dimensional phase-space.

In particular the dimensionality of the integral will increase s.t.:

- if we consider a real emission the phase-space will change from a $3n$ integration volume to a $3(n+1)$ integration volume
- if we consider a virtual emission, we will need to compute an integral of dimension $3n+4$, where $3n$ comes from the phase-space integration and 4 from the integration over the loop-line

Secondly the squared matrix element $|\mathcal{M}^2|$ can become difficult to sample even for common low-dimensional SM processes, since in general is particularly peaked

in smaller region of the integration domain in the vicinity of kinematic divergences. These regions become even smaller for high-dimensional integrands due to the large number of parameters.

This tendency of having sharp peaks in limited regions is the cause of the terrible statistical convergence of naive MC integrators which perform uniform sampling. Thus importance sampling techniques are the solution of choice, since they enable us to perform a more effective sampling.

Once we have chosen a sampling technique we can reach better accuracies by simply increasing the size of the sample, since, as we have seen, the standard deviation for a MC simulation decreases with the size of the sample as $N^{-1/2}$.

Therefore we expect that for each physical process there will be a number of samples needed in order to reach the target accuracy required. In particular we will need large samples to reach high target accuracies when dealing with NLO terms due to the high-dimension and the complexity of the squared matrix element $|\mathcal{M}|^2$.

However this is only valid from a theoretical point of view, in practice these calculations are performed by computers which can have a hard time in sampling these complicated integrands as well as dealing with huge samples in order to reach the target accuracy. In particular the problem related to the CPU cost and the long computational times has been getting a lot of attention over the last years.

1.3.2 CPU costs and computational times

The CPU cost of MC methods is one of the biggest problem that is driving the budget of big experiments such as ATLAS or CSM. Since 2010, MC integration has gone from being a trivial element of an experiment's CPU budget to, particularly in the case of top quark production processes, an important consumer up to 20% of the experiment's CPU budget [14]. The main driver for this CPU usage has been the availability of the complex multileg and NLO QCD processes.

This trend of high CPU resources is in contrast with the CPU budgets currently available at the LHC experiments. From Figure 1.3 we can see that the annual CPU consumption will overcome the CPU budget especially in Run 4 and Run 5 of the ATLAS experiment. The main problem for the ATLAS experiment is the heavy use of the `SHERPA` event generator which is demanding more CPU by comparison with `MadGraph5` which dominates the CSM simulation budget.

Another problem is related to the long computational times.

During a simulation we expect to perform various iterations of our MC routine and the final result will be expressed as a weighted average of all the previous outputs. In particular in the case of integration we need to repeat the process of sampling and the following instructions to compute the integral according to the technique used. When dealing with complex integrands, such as the NLO contribution for a QCD process, sampling could require long computational times.

The current importance sampling techniques can provide an approximate estimate for the sampling distribution by discarding or re-weighting the samples from such distribution, in particular the aggregation of these imperfect phase-space mappings is one of the major cause for the poor efficiencies of MC event sampling at high fixed order in the strong coupling α_s .

The ideal distribution would by definition always have unit weights, i.e. if the sampling density matches the optimal one, but in practice the sample weights have a tail to lower values, since the proposal density includes phase-space points which

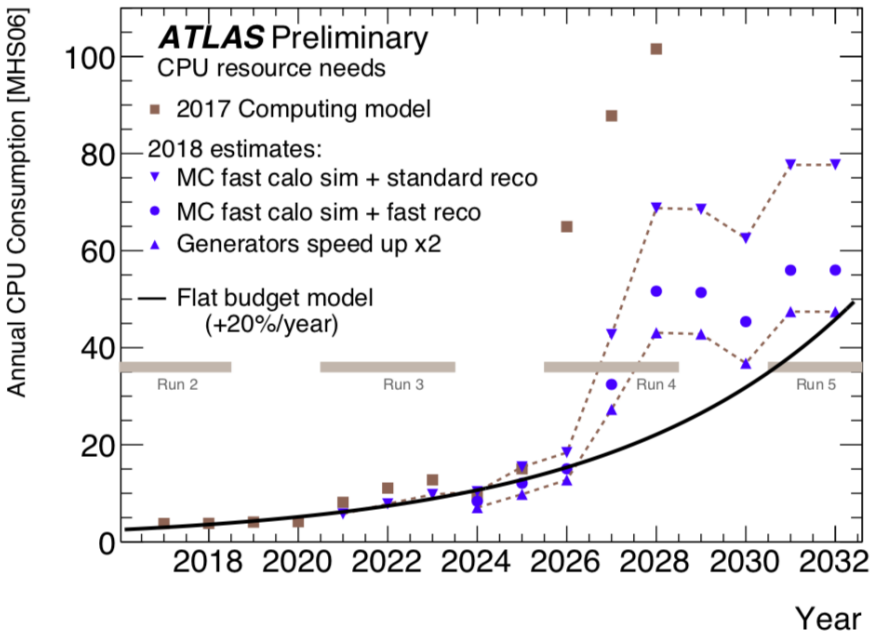

Figure 1.3: ATLAS CPU resource needs 2018 estimates. Image from Ref[15].

are not relevant for the integration. These lower weights can lead to poor statistical convergence in our computations.

We can also have greater than unit weights , these usually occur when the proposal density underestimated the maximum due to a failure in the previous sampling, with consequently single-event spikes.

In particular this sample rejection from broad distribution of weights combined with an already CPU-intensive computation of the matrix element value for each sample, can explain the huge CPU cost needed in the latest HEP experiments. From Figure 1.4 we can see that the CPU-time per event can become particularly large, for some current processes it can take up to 24 hours for a single event.

1.3.3 Possible solutions and aim of the thesis

The trend of high CPU requirements of the High-Luminosity LHC programme cannot continue, especially because in the future we will need more precision in the form of 1-loop NLO and 2-loop NNLO QCD calculation that may come at unacceptable CPU costs.

The question is therefore: how can we lower the CPU usage while still achieving high accuracy predictions? We can work in two different directions in order to achieve our goal.

Firstly we can develop new algorithms for multi-dimensional integration. In particular new techniques which are able to reach the target accuracy with a lower number of events thus reducing both the computational times and the CPU usage. These new techniques may include new numerical MC algorithms as well as ML techniques such as learning the integration phase-space using boosted decision trees [16] or deep neural networks [2].

Secondly we can lower the CPU usage by looking at new computer architecture such as GPUs or multi-threading CPUs. For the purpose of MC integration of the squared matrix element, which comes down to the event sampling and adaptive

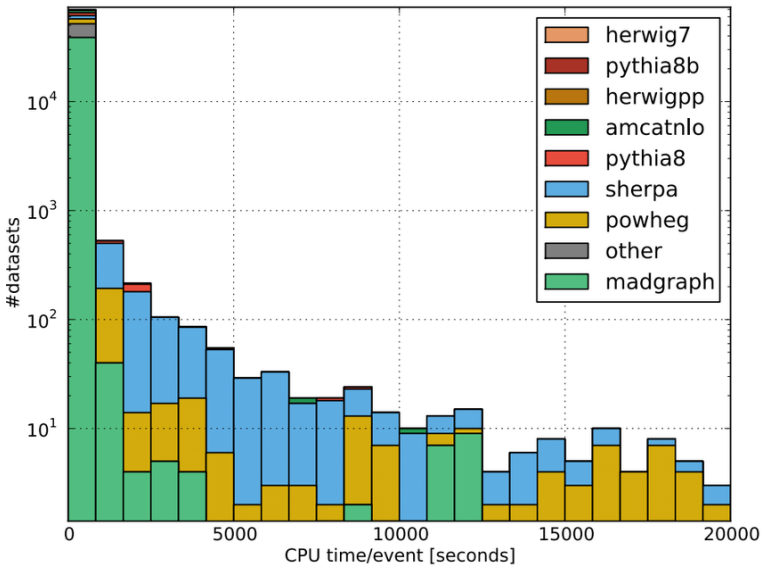

Figure 1.4: CPU time per event using different MC generators. Image from Ref[14].

strategies to compute the integral, the use of hardware acceleration devices is particularly appealing. In fact the sampling process is embarrassing parallel since we can just use a different random-number generator seeds for each run. Also the other operations used in MC techniques such as stratified sampling may take advantage of parallelizability due to the fact that we can express the integral as a sum of the MC integration in subregions of the integration volume.

The aim of the thesis is to study and implement new MC integration algorithms with the aim of achieving the target accuracies and at the same time overcoming the computational limitations by taking advantage of hardware acceleration devices.

Chapter 2

Algorithms and implementation

In this chapter we focus our attention on specific integration algorithms. The first algorithm considered is the classic VEGAS algorithm [5]. We discuss both the techniques used: importance sampling and stratified sampling. We also highlight its limitations.

Secondly we present in detail VEGAS+ [4], a modification of the classic VEGAS algorithm which includes an adaptive stratified sampling technique.

After that we focus on a possible implementation of these algorithms based on hardware acceleration devices. We analyze briefly an implementation of the VEGAS importance sampling algorithm using the TensorFlow library, `VegasFlow`. We highlight the role of TensorFlow as the back-end development framework and we discuss the most relevant achievements from Ref[3].

Finally we present a novel implementation of the VEGAS+ algorithm within the `VegasFlow` library.

2.1 Algorithms

Over the years have been proposed several integration algorithms based on Monte Carlo methods. These algorithms usually implement the techniques of variance reduction presented in section 1.1.2: importance sampling and stratified sampling.

We will focus on two algorithms which employ both importance and stratified sampling: VEGAS and VEGAS+.

2.1.1 VEGAS

VEGAS is an algorithm for adaptive multi-dimensional MC integration formulated by Peter Lepage in 1977 in Ref[5]. Since then it has been used in numerous fields including chemistry, applied finance and physics.

VEGAS is widely used especially in HEP both as a MC event generator as well as to evaluate Feynman diagrams numerically. In particular is the main driver for programs which perform QCD fixed-order calculations such as MCFM [17, 18], NNLOJET [19]. It also used for more general tools such as MG5_aMC@NLO [20] and Sherpa [21]

Importance sampling

VEGAS is primarily based on importance sampling but also features some stratified sampling techniques. Using importance sampling, as we know from the previous

chapter, the aim is to find a function p that resembles the integrand f which can easily be sampled. The sampling density implemented in the algorithm is a *separable* multi-dimensional function:

$$p \propto g(x_1, x_2, x_3, \ldots, x_n) = g_1(x_1)g_2(x_2)g_3(x_3)\ldots g_n(x_n) \tag{2.1}$$

The sampling of a n-dimensional vector $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$ is performed sampling the n-one dimensional sampling densities g_i to obtain the coordinates x_i of the vector, which is far more simple than sampling a complex multi-dimensional function.

Moreover it can be shown [5, 6] that optimal weight functions are:

$$g_1(x_1) = \frac{\left[\int dx_2 \ldots \int dx_n \frac{f^2(x_1,\ldots,x_n)}{g_2(x_2)\ldots g_n(x_n)}\right]^{\frac{1}{2}}}{\int dx_1 \left[\int dx_2 \ldots \int dx_n \frac{f^2(x_1,\ldots,x_n)}{g_2(x_2)\ldots g_n(x_n)}\right]^{\frac{1}{2}}} \tag{2.2}$$

which suggests what will be VEGAS adaptive strategy: starting from a set of g-functions, when sampling the function f, we can accumulate the squared value of the function in each sampled point $\tilde{\mathbf{x}}$, i.e. $f^2(\tilde{\mathbf{x}})$, and then use these informations to determine the improved g_i functions iteratively based on Eq.(2.2).

The algorithm uses as sampling densities step functions with a number of step N fixed, that is divides each one-dimensional domain of integration which can be taken as $[0,1]$ [1] in N subintervals Δx_i with the constraint:

$$\sum_{i=1}^{N} \Delta x_i = 1 \tag{2.4}$$

The one-dimensional probability density of a random number being chosen from any given step Δx_i is defined to be a constant equal to:

$$g_i(x) = \frac{1}{N\Delta x_i} \text{if} \tag{2.5}$$

if x is in the interval $\{x_i - \Delta x_i, x_i\}$.

The probability distribution for each dimension is then modified in each iteration of the simulation by simply adjusting the increment sizes Δx_i. It can be shown that the variance is minimized when the average value of $f^2(\mathbf{x})$ in each interval is the same for every interval:

$$\Delta x_i \int_{x_i - \Delta x_i}^{x_i} d\mathbf{x} f^2(\mathbf{x}) = \text{constant} \tag{2.6}$$

This approach is particularly effective for integrands with strong peaks since the algorithm will shorten the intervals where the function is peaked in order to achieve the optimal partition of intervals given by Eq.(2.6).

[1] If the integral is defined between two generic integers a and b we can perform a change of variable with Jacobian $J(y)$ to simply change the boundaries from $[a,b]$ to $[0,1]$

$$I = \int_a^b dx f(x) = \int_0^1 dy J(y) f(x(y)) \tag{2.3}$$

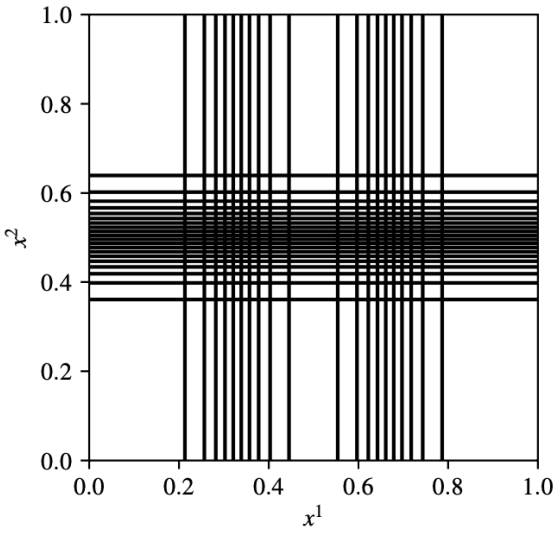

Figure 2.1: Vegas grid for the integral defined in Eq.(2.8). Image from Ref[4].

The variation of all the intervals Δx_i is done iteratively. VEGAS initially estimates the integral with a uniform grid (all the intervals have the same length). Simultaneously the algorithm accumulates the quantities d_is which are defined as:

$$d_i \equiv \frac{1}{n_i} \sum_{x_j \in [x_i - \Delta x_i, x_i]} f^2(\mathbf{x}) \approx \Delta x_i \int_{x_i - \Delta x_i}^{x_i} d\mathbf{x} f^2(\mathbf{x}) \tag{2.7}$$

Using these parameters the algorithm presented in Ref[4] is able to generate new intervals $\{x_i' - \Delta x_i', x_i'\}$ that contains an equal fraction of the total $d = \sum_i d_i$ thus fulfilling the constraint of Eq.(2.6).

To better understand the importance sampling used in VEGAS we show an example of an optimize grid. Suppose that we need to compute the following integral:

$$\int_0^1 d^4x \left(e^{-100(\mathbf{x}-\mathbf{r}_1)^2} + e^{-100(\mathbf{x}-\mathbf{r}_2)^2} \right) \tag{2.8}$$

where $\mathbf{x} = (x_1, x_2, x_3, x_4)$, $\mathbf{r}_1 = (0.33, 0.5, 0.5, 0.5)$ and $\mathbf{r}_2 = (0.67, 0.5, 0.5, 0.5)$. We expect that the optimized grid should shorten the intervals near 0.33 and 0.67 in the direction x_1 and near 0.5 for all the other directions. This exactly what is shown in Fig.2.1.

The importance sampling proposed in VEGAS is particularly effective for integrals like that one in Eq.(2.8) since the integrand function being a sum of Gaussians can be separated into a product of one-dimensional integrals over each direction.

The weakness of this algorithm is the obvious one: not all integrands can be approximated by their projections onto individual coordinate directions. This is why VEGAS can struggle to converge with function whose geometry is non-separable or more in general if the integrand is concentrated along one-dimensional (or higher) curved trajectories (or hypersurfaces) unless these happen to be aligned with the coordinate directions. The simplest case corresponds to a Gaussian peaked along a generic body diagonal line.

Stratified Sampling

VEGAS also employs standard stratified sampling techniques to reduce the variance. Assuming to compute a D dimensional integral each axis is divided into a fixed number of stratifications N_{st}, in particular using a total of N_{ev} per iteration N_{st} is computed as:

$$N_{\text{st}} = \lfloor (N_{\text{ev}}/2)^{1/D} \rfloor \tag{2.9}$$

which corresponds to dividing the D dimensional volume into N_{st}^D hypercubes of side $1/N_{\text{st}}$.

After that a MC integration is performed in each hypercube using n_{ev} samples, the integral required I can be computed as:

$$I = \frac{V}{N_{\text{st}}^D} \sum_h \left(\frac{1}{n_{\text{ev}}} \sum_{\mathbf{x} \epsilon h} f(\mathbf{x}) \right) = \sum_h I_h \tag{2.10}$$

where I_h denotes the integral estimate limited to hypercube h.

The variance can be computed using Eq.(1.17) as the sum of the variance in each hypercube:

$$\sigma_I^2 = \sum_h \sigma_h^2 \tag{2.11}$$

where σ_h^2 denotes the variance in the hypercube h. In order to perform a MC integration in each hypercube we need to have at least 2 integrand samples per hypercube. This requirement is already satisfied since the number of stratifications was defined in Eq.(2.9) such that we will have at least 2 samples per hypercube.

The algorithm uses the same number of samples per hypercube n_{ev} defined as:

$$n_{\text{ev}} = \lfloor (N_{\text{ev}}/N_{\text{st}}^D) \rfloor \geq 2 \tag{2.12}$$

The addition to a stratified sampling techniques along side the importance sampling is surely useful. However, as already discussed in Sect.1.1.2, stratified sampling works better with low-dimensional integrals. For high-dimensional integrals huge samples are necessary to observe any considerable improvements.

2.1.2 A new algorithm: VEGAS+

VEGAS, as we already discussed, struggles to converge with integrands that have non-trivial correlations between the integration variables. An integrand concentrated close to a body diagonal line, for example one from $(0, 0, \ldots, 0)$ to $(1, 1, \ldots, 1)$ shows a slower convergence since his geometry is completely non-separable. Also functions with multiple peaks can become challenging for the current implementation of VEGAS.

A new algorithm has been formulated which has been proven to perform better than classic VEGAS in these particular instances [4]. This new method consists in a modification of the classic VEGAS by adding a second adaptive strategy in addition to the importance sampling, hence the name VEGAS+.

We saw that VEGAS also uses a stratified sampling by dividing the integration in N_{st}^D hypercubes then performing a MC estimates of the integral using at least 2 points per hypercube. In the classic implementation the number of samples per hypercube is the same for all the subvolumes.

VEGAS+ improves the stratified sampling techniques of VEGAS by allowing the number of integrand samples per hypercube to change from hypercube to hypercube. In particular these samples are redistributed iteratively in order to minimize the variance of the integral estimate using the samples of the previous iteration. We can therefore say that VEGAS+ employs an adaptive stratified sampling.

Moreover by allowing a redistribution of the samples this algorithm can include non-trivial correlations between the integration variables overcoming the separable-geometry approach of VEGAS.

The integral will be computed as before with the only difference that we need to define a new variable n_h which represents the samples used in the hypercube h:

$$I = \frac{V}{N_{\text{st}}^D} \sum_h \frac{1}{n_h} \sum_{\mathbf{x}\epsilon h} f(\mathbf{x}) = \sum_h I_h \tag{2.13}$$

Now the expression for the variance in Eq.(2.11) must be modified to include the different number of samples:

$$\sigma_I^2 = \sum_h \frac{\sigma_h^2}{n_h} \tag{2.14}$$

Let us show briefly that the variance is minimized when the number of samples n_h is proportional to the standard deviation of the hypercube σ_h. The samples per hypercube n_h are subject to the constraint that the sum of the points sampled in each hypercube must be equal to the total number of sampled points N_{ev}:

$$N_{\text{ev}} = \sum_h n_h \tag{2.15}$$

We can find the optimal n_h subject to the previous constraint using the method of Lagrange multipliers:

$$0 = \frac{\delta}{\delta n_h}\left(\sum_k \frac{\sigma_k^2}{n_k} + \lambda \sum_k n_k \right) = -\frac{\sigma_h^2}{n_h^2} + \lambda \tag{2.16}$$

Being λ constant we get that the previous equation is satisfied if

$$n_h \propto \sigma_h \tag{2.17}$$

The algorithm presented in Ref[4] redistribute the samples in the hypercubes as follows:

1. Choose as number of stratifications

$$N_{\text{st}} = \lfloor (N_{\text{ev}}/4)^{1/D} \rfloor \tag{2.18}$$

2. During each iteration estimate the variance of each hypercube with same samples used to compute the integral

$$\sigma_h^2 \approx \frac{V_h^2}{n_h} \sum_{\mathbf{x}\epsilon V_h} f^2(\mathbf{x}) - \left(\frac{V_h}{n_h} \sum_{\mathbf{x}\epsilon V_h} f(\mathbf{x}) \right)^2 \tag{2.19}$$

where V_h is the hypercube volume

3. Replace the variance by introducing a damping parameter $\beta \geq 0$ as:

$$d_h \equiv \sigma_h^\beta \tag{2.20}$$

with default value $\beta = 0.75$

4. Recalculate the number of samples for each hypercube to use in the next iteration:

$$n_h = \max\big(2, d_h / \sum_{h'} d_{h'}\big) \tag{2.21}$$

Let us now commentate briefly the algorithm.

Firstly we can observe that the number of stratifications chosen is smaller than the one used in VEGAS (Eq.(2.9)). This is done in order to have enough samples in each hypercube to have a better estimates of the variance and more significant variations for the n_hs.

Secondly the damping parameter β is being introduced to avoid overreactions to random fluctuations in the first steps of the simulation. The optimal choice will be $\beta = 1$, in the limit where $\beta = 0$ we have VEGAS usual stratified sampling without the redistribution of samples.

At the same time after each iteration the VEGAS map is updated according to the standard VEGAS importance sampling algorithm. The optimal VEGAS grid is independent of the allocation of samples, but the reallocation of samples can speed the convergence to the optimal map. This is due to the fact that by reallocating the samples we are able to find more quickly all the peaks of the integrand, even for diagonal structures. In fact Ref[4]considered the following diagonal-structured integral:

$$\int_0^1 d^8x \sum_{i=1}^{3} e^{-50|\mathbf{x}-\mathbf{r}_i|} \tag{2.22}$$

where the peaks are distributed along the diagonal:

$$\begin{aligned}
\mathbf{r}_1 &= (0.23, 0.23, 0.23, 0.23, 0.23, 0.23, 0.23, 0.23) \\
\mathbf{r}_2 &= (0.39, 0.39, 0.39, 0.39, 0.39, 0.39, 0.39, 0.39) \\
\mathbf{r}_3 &= (0.74, 0.74, 0.74, 0.74, 0.74, 0.74, 0.74, 0.74)
\end{aligned}$$

Fig.2.2 shows the speed of convergence of classic VEGAS and VEGAS+ for the previous integral. We can see that classic VEGAS becomes unstable below $N_{\text{ev}} = 3\times10^6$ while VEGAS+ is unstable below $N_{\text{ev}} = 1\times10^5$. Moreover VEGAS+ is overall the most accurate integrator for this particular integral.

Limitations

The adaptive stratified sampling suffers from the same limitations of the standard adaptive stratified sampling. We need a larger number of samples compared to the importance sampling to have significant effects for high-dimensional integral. From Eq.(2.18) we can observe that the number of stratifications N_{st} grows with the number of events as

$$N_{\text{st}} \approx N_{\text{ev}}^{1/D} \tag{2.23}$$

therefore the number of stratifications is suppressed as the dimension increases.

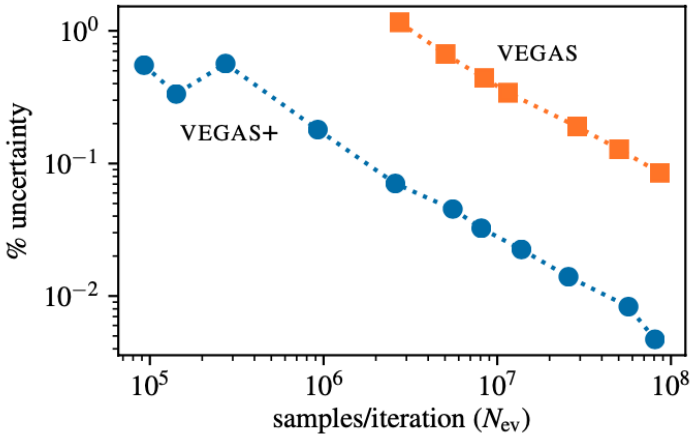

Figure 2.2: Image from Ref[4] showing the percent uncertainty in the integral estimates of Eq.(2.22) from 30 iterations of classic VEGAS and VEGAS+.

2.2 Implementation

In this section we discuss a novel implementation of the algorithms aforementioned. In particular the VEGAS algorithm has been available since the '80s and has been implemented in different programming languages. The new VEGAS+ algorithm is currently available at Ref[22]. The original implementations are usually written for a single CPU or at most they support multi-processor evaluation of integrands using MPI, via the python module `mpi4py`.

We have instead decided to focus on an implementation that can also run in hardware acceleration devices such as multi-threading CPUs and GPUs.

2.2.1 VegasFlow: a brief overview

`VegasFlow` [23] is the first implementation of the VEGAS algorithm that is able to run both on CPUs and GPUs.

The aim of the library is to exploit the paralelizability of MC computations thus lowering the long CPU-times usually required especially when dealing with HEP integrands. `VegasFlow` achieves this goal using Google's TensorFlow library, which is primarily used for deep learning applications, as the back-end development framework.

Why TensorFlow?

TensorFlow (TF) has a simple mechanism that enable us to write python code which can be distributed to hardware acceleration devices without complicated installation procedures.

In particular there is no need to write a different version of the code whether we run it on CPU or GPU. Once TF has found different devices, for example a CPU and a GPU, we can choose where to run a specific set of instructions by using the primitive `tf.device`. By default TF will run on GPU if available.

```
import tensorflow as tf

with tf.device('/CPU:0'):
        # these operations will run on CPU
```

```
        a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
        b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])

# this will run on GPU
c = tf.matmul(a, b)
```

Secondly TF has two execution modes one is the so called eager mode and the other one uses graphs to compile python code.

The eager execution, which is turned on by default, implements an imperative programming environment that evaluates operations immediately: each operation returns concrete values.

There is also the possibility to create and run a TF graph by adding the decorator `@tf.function` to the functions defined by the user. In particular the function defined with such decorator will be a python callable that builds TensorFlow graphs from the python function. A TF graph requires that his input must have a specified data and dimension type since it cannot contains all the statements of the eager program. The code is separated in two stages:

1. In the first step a graph is created by the function, all the python code runs normally while the statement expressed by TF primitives are deferred: they are simply stored in the graph. This first step is referred as *tracing*.

2. In the second step the graph created in the first one is run.

One of the main advantages is that the second step is much faster than the first one, due to the fact that the statements have been converted to a graph. Graphs are easily optimized and allow the compiler to do transformations like separating sub-parts of a computation that are independent and splitting them between threads or devices. If a function is called more than once with the same arguments the first step, the *tracing*, will be performed only the first time. In all the successive calls of the function the first step will be skipped since TF already as a graph available for that particular function. This possibility of skipping the tracing step is what enable TF to reach better performances when compared to the eager mode.

This feature can be exploited in MC simulation where we expect to call a function a different number of times depending on the accuracy required.

Design and algorithms

The library has an abstract class called `MonteCarloFlow` which implements the distribution of events across multiple devices with a job scheduling for multi-GPU synchronization using TF graph technology.

The proper MC integrators are implemented as derived class from the `MonteCarloFlow` class. The library provides two different integrators: one is a vey simple MC integrator in the class `PlainFlow` and the main one is an implementation of Vegas's importance sampling algorithm in the class `VegasFlow`. The ladder one is the first implementation of the importance sampling algorithm in VEGAS using TensorFlow as the back-end development framework. The two subclasses focus only on the integration algorithm chosen, in particular they need to specifies what the algorithm does to run one single event and secondly what to do in a full iteration of the MC simulation.

Therefore the library is designed such that new algorithms can easily be implemented as derived class from the `MonteCarloFlow` abstract class which handles all

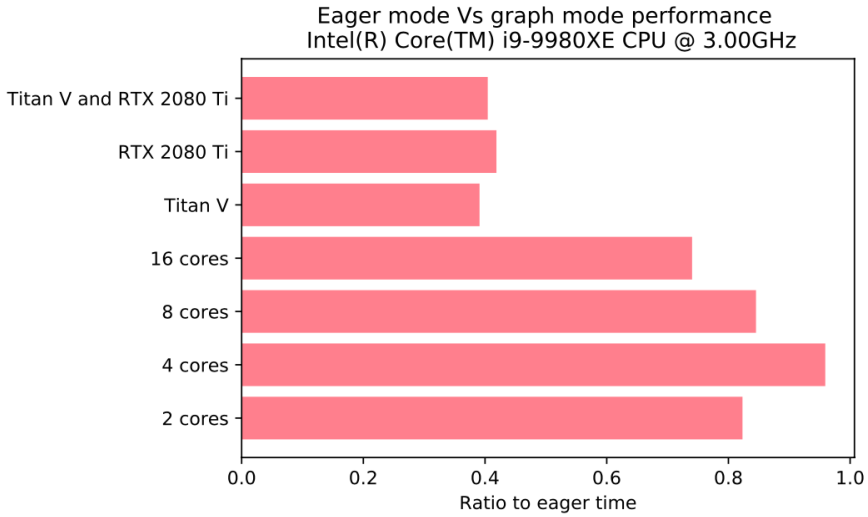

Figure 2.3: Comparison of performance between the eager and graph compilation TensorFlow mode. The results are shown as a ratio of the time it took the eager computation to complete one iteration. Image from Ref[3].

the technicalities such as GPU distribution, multi-threading or vectorization. The developer, as already shown in the integrators available, should only focus on what the integrators do for a single event and in a full iteration.

Results

We now comment some results of VegasFlow presented in Ref[3]. VegasFlow (running in both CPU and GPU) has been confronted with the current implementation of the Vegas importance sampling algorithm available in python [22]. The results show that in order to reach the same accuracy integrating a Gaussian distribution in 20 dimension the previous implementation of VEGAS takes 38 minutes, while VegasFlow running on CPU takes 26 minutes and only 5 minutes when running of GPU. Therefore as expected when `VegasFlow` runs on GPU there are some significant improvements in the computational times.

Moreover `VegasFlow` shows better performance thanks to the graph compilation mode when running on highly parallel scenarios such as multi-CPU computation or GPU as shown in Fig.2.3

Finally `VegasFlow` has been confronted with MadGraph5_aMC@NLO [20] for the computation of the single t-quark production at the partonic level at leading order. The results in Fig.2.4 show how `VegasFlow` can outperform MadGraph5_aMC@NLO while running on CPU and even more on GPU.

2.2.2 A new implementation: VegasFlowPlus

In this thesis we present a novel implementation of the VEGAS+ algorithm within the framework of the `VegasFlow` library.

Motivation

As we already observed in the `VegasFlow` library the primarily algorithm is the VEGAS importance sampling algorithm, which has been proven to be effective for multi-dimensional integration.

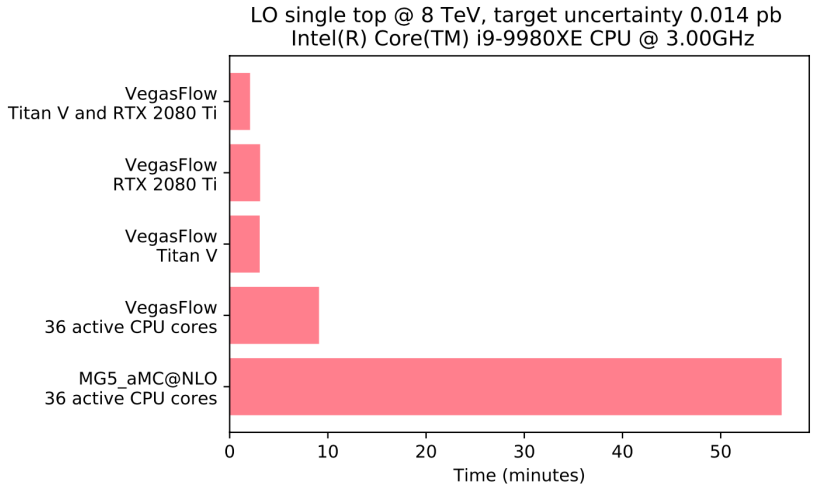

Figure 2.4: Comparison of a Leading Order calculation ran in both `VegasFlow` [23] and MG5_aMC@NLO [20]. For the same level of target accuracy `VegasFlow` is faster than MG5_aMC@NLO when using both CPUs and GPUs devices. Image from Ref[3].

`VegasFlow` has shown remarkable performances compared with the importance sampling implemented in python [22] thanks to the support of hardware acceleration devices . Obviously in order to reach the same accuracy level the number of iterations was roughly the same since the two integrators are based on the same algorithm.

In order to reach better performances we consider the possibility of implementing more accurate algorithms in the VegasFlow library, i.e. integrators that converge using less events per iteration or less iterations.

In particular we have considered the VEGAS+ algorithm as a possible candidate. As we already discussed in Sect.2.1.2 VEGAS+ employs a new technique called adaptive stratified sampling together with the importance sampling algorithm. We benchmarked the performances of the importance sampling compared to the new VEGAS+ algorithm and the adaptive stratified sampling technique alone to see whether one of them can outperform the importance sampling method.

The benchmark was performed using the python implementation of the VEGAS algorithm [22]. We first performed a dimensional comparison, i.e. we tested the three methods with integrands of different dimensions to see which integrators shows better accuracy the results are shown in Fig.2.5 and Fig.2.6.

We can observe that VEGAS+ seems to be the more accurate integrator in general. Moreover, we can see that for the Gaussian distribution the adaptive stratified sampling is by far the less precise integrator due to the sharp peak of the distribution. For the case of the RosenBrock function we can see that the adaptive stratified sampling can outperform the importance sampling only for dimension less than 5. This is expected since the stratified sampling struggles to converge in higher dimensions.

We also performed a comparison based on the number of samples which showed that in general using the same number of samples VEGAS+ converged more rapidly than the importance sampling. The adaptive stratified sampling converged more slowly and only with a large number of samples can sometimes reach the accuracy of the other two methods.

Finally we have also considered a performance comparison, i.e. at fixed samples per iteration we have considered the number of iterations needed in order to reach the

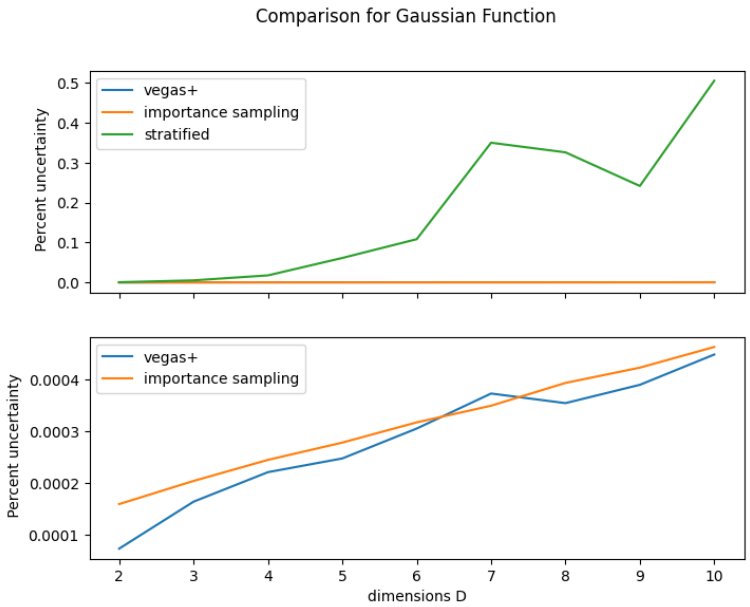

Figure 2.5: Comparison of percent uncertainty of Gaussian integral from 50 iterations with 12000 samples after a warmup of 5 iterations with 1000 samples.

accuracy required. The most relevant result was obtained using a physical integrand, MORE SPECIFIC . We noted that VEGAS+ was the integrator that performed better and was the only one that achieved a percent uncertainty of 0.0001 within a maximum of 100 iterations. The possibility of reaching the target accuracy using less than 20 iterations leads to significant improvements also in the computational time.

Overall all the tests performed show that VEGAS+ is the more efficient integration algorithm and can outperform the implementation of the importance sampling in VEGAS. The adaptive stratified sampling alone was effective only with low-dimensional integrands without sharp peaks, like the RosenBrock function. In general, it couldn't compete with the other two methods.

Moreover, the fact that for a physical integrand VEGAS+ converged way more rapidly than the importance sampling convinced us even more to exploit the VEGAS+ algorithm by running it on hardware acceleration devices.

Implementation

To implement the VEGAS+ algorithm within the `VegasFlow` library we exploit the possibility of adding new algorithms simply by adding classes derived from the `MonteCarloFlow` class. Moreover, since the VEGAS+ algorithm contains the same importance sampling method already implemented in the `VegasFlow` class we decided to create a derived class from the `VegasFlow` class called `VegasFlowPlus`

The next step was to implement all the machinery of the stratified sampling. One of the main difference is the process of sampling as expected. Using only importance sampling the points are sampled from all the integration domain accordingly to the sampling distribution which is a grid refined in every iterations. Using stratified sampling instead we need to divide the integration domain in a certain number of hypercubes and we need to extract a specific number of points from each hypercube. Thus we introduced a new function `generate_samples_in_hypercubes` that implements this different type of sampling.

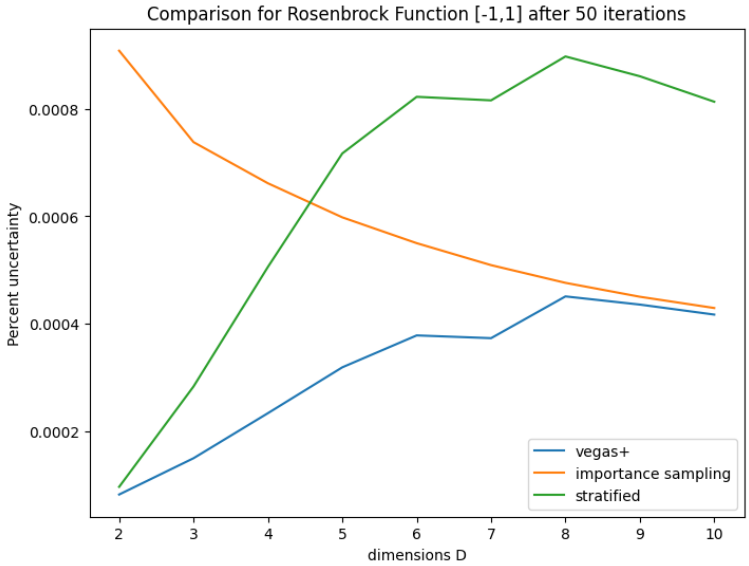

Figure 2.6: Comparison of percent uncertainty of Gaussian integral from 50 iterations with 12000 samples after a warmup of 5 iterations with 1000 samples.

Moreover, since we are interested in the VEGAS+ algorithm, in which the points sampled are no longer fixed in every hypercube but change accordingly to the variance in each one, we have also introduced a tensor `n_ev` which contains the number of samples in each hypercube. After each iteration this tensor is updated according to the VEGAS+ algorithm explain in section 2.1.2 by the method `redistribute_samples` that receives as input a tensor containing the variance computed in each hypercube.

Finally we also needed to overload the fundamental methods of the class that describe what the integrator should do for each event and for each iteration of the simulation. In particular since the integral is expressed as a contribution from a MC estimate of the integral in each hypercube both the integral estimate and the variance are computed by summing the partial results from all the hypercubes according to Eq.(2.10) and Eq. (2.11).

All the implementation is publicly available at the following GitHub repository: `https://github.com/N3PDF/vegasflow`.

Problems during the implementation

The process of implementing the VEGAS+ algorithm within the `VegasFlow` library has not been free of problems.

The choice of TensorFlow was motivated also by the fact that the graph implementation enable us to achieve better performances. In particular if a function is called more than once the program will refer to the same graph of the function if the type and the shape of the input tensors are the same. Both the importance sampling and the stratified sampling can benefits from this since all the tensors don't change size from one iteration to another. However the new feature of VEGAS+, i.e., the adaptive stratified sampling does not maintain the same shape for all the tensors. In fact when calling the method `redistribute_samples` the total number of events used in each iterations changes since the algorithm does not maintain exactly the same number of samples. Therefore when executing the program we will not be able to skip the tracing step, but for every iteration of the simulation we will need to

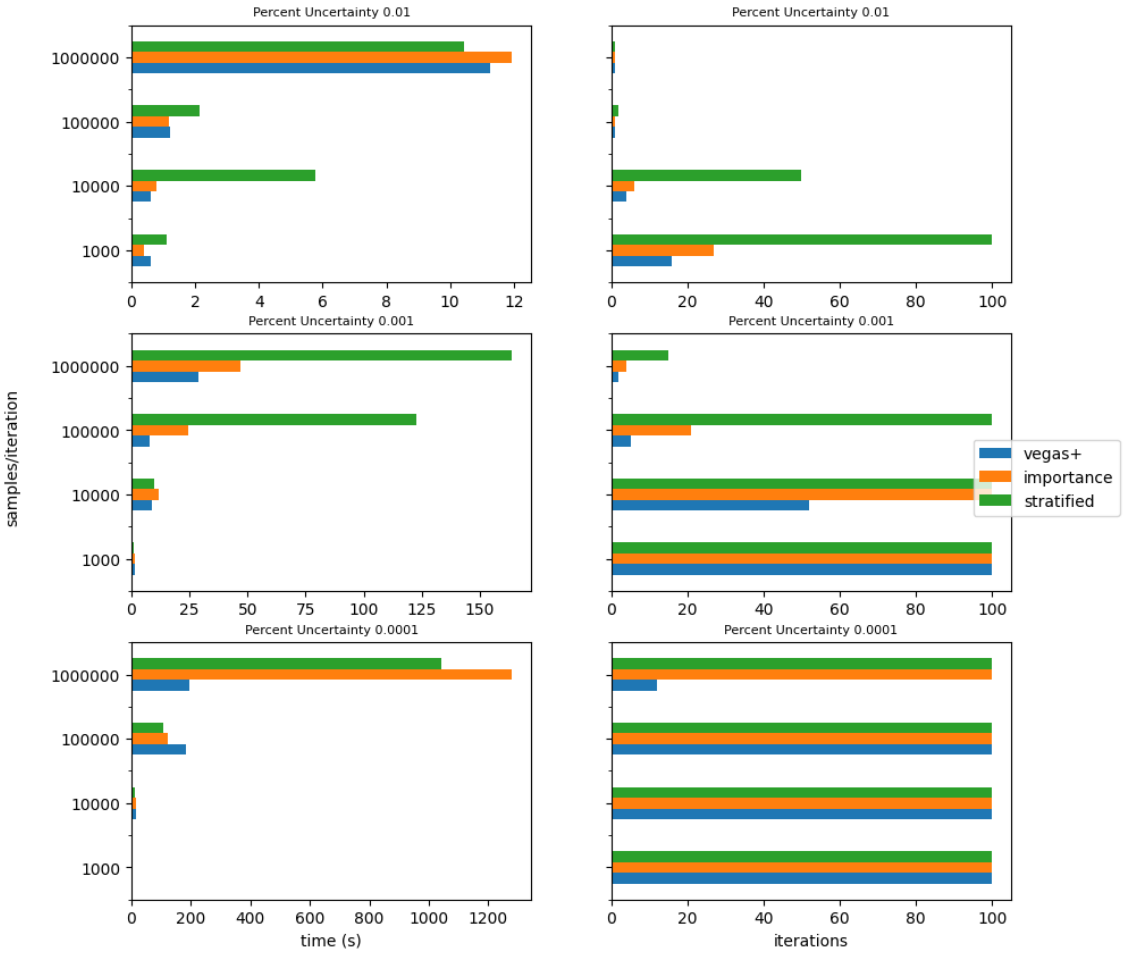

Figure 2.7: Comparison of perfomances between different integration methods at fixed percent uncertainty (10^{-2}, 10^{-3} or 10^{-4}) and at fixed samples per iteration (10^3,10^4,10^5 or 10^6) with 100 maximum iterations.

generate new graphs for all the functions that involve the total events per iteration. This problem is known as *retracing* and leads to worst performances especially if the function that causes the retracing is called a large of number of times.

The TensorFlow library offers a way around the *retracing* problem. In particular one can specify a `input_singnature` in a function defined with the decorator `@tf.function`. Through the `input_singnature` we can specify both the type and the shape of the input tensors, by setting one or more dimensions to `None` in the shape we can allow for flexibility in the trace reuse. Below we can see an example of the use of the `input_singnature` from the source code of the `VegasFlowPlus` class.

```
@tf.function(input_signature=3 * [tf.TensorSpec(shape=[None, None], dtype=DTYPE)])
def _compute_x(x_ini, xn, xdelta):
        """ Helper function for generate_samples_in_hypercubes """
        aux_rand = xn - tf.math.floor(xn)
        return x_ini + xdelta * aux_rand
```

Therefore we made all the programming environment flexible with respect to the number of events used in each iteration by specifying an input signature for each function.

There was a second problem the VEGAS+ algorithm. We experienced some crashes while the program was running due to a high CPU usage even for high performing machines. After some tests we found out that when generating the samples per hypercube there was a problem with the primitive of TensorFlow `tf.repeat`. In particular inside the function at some point a rank-3 tensor was initialized using the total number of hypercubes, the dimension of the integrand and the maximum number of samples in one hypercube. Due to some overreaction of the algorithm in the first iterations the redistribution of the samples can accumulate a large number of samples in one particular hypercube. This lead to rank-3 tensors such as `[65536,51749,8]` which are too big to handle even for professional-grade CPUs and GPUs.

To solve this issue we decided to modify the program in the following way. First of all we tried to reach the same output desired using alternatives to `tf.repeat`. These options could lower the CPU-usage however there were still some problems involving `tf.repeat`. So we decided to simply put a limit of 10000 maximum hypercubes because by limiting the number of hypercubes we could avoid the big tensors that caused the problems. Moreover, we decided to turn off the adaptive stratified sampling for high-dimensional integrands ($D > 13$) since it cannot be effective unless we use a bigger number of hypercubes.

Chapter 3

Result benchmark

In this chapter we present the benchmark between the integrator already available in `VegasFlow` and different variations of the VEGAS+ algorithm just implemented. We run the integrations both on CPU and GPU and quantify the benefits given from hardware acceleration.

We test different functions from the standard Gaussian distribution in various dimensions to integrands more relevant for HEP processes. In particular we consider three different computations: Drell-Yan, single top production and vector fusion boson Higgs production all at LO.

After a detailed analysis of the performance of each integrator, we compare all the results together. Our aim is to present to the reader which integrators works best depending on the integrand considered.

3.1 Setup and goals

For the benchmark we compare the importance sampling integrator in the `VegasFlow` library with the new integrator based on the VEGAS+ algorithm in the class `VegasFlowPlus`. We would like to answer the following questions:

1. which integrator is the most accurate depending on the integrand, his dimension and his complexity?

2. what are the benefits of hardware acceleration devices to the performance of these integrators?

To answer the first question we compare the number of iterations needed to reach different levels of accuracy for various type of integrands. Obviously we expect that the more accurate integrator will converge using less iterations of the MC simulation. Moreover, in the case where one or more integrators cannot achieve the target accuracy within the maximum number of iterations, we compare the accuracies reached at the end of the simulation.

To highlight the benefits of hardware acceleration we run all the benchmark in a professional-grade CPU (Intel i9-9980XE[1]) as well as a professional-grade GPU (NVIDIA Titan V[2]). In particular, we measure the average time per iteration to see if the new integrator exhibit the same speed-up already observed in `VegasFlow` when

[1] More specific about CPU
[2] More specific about GPU

Integrator Name	Class	$\beta_{\text{warm-up}}$	β	$\alpha_{\text{warm-up}}$ [3]	α [4]
Importance Sampling	`VegasFlow`	`None`	`None`	1.5	0
Classic VEGAS	`VegasFlowPlus`	0	0	1.5	0
VEGAS-VEGAS+	`VegasFlowPlus`	0.75	0	1.5	0
VEGAS+	`VegasFlowPlus`	0.75	0.75	1.5	0

Table 3.1: Integrators used in the benchmark.

running on high-parallel scenarios. Moreover, we want to observe if `VegasFlowPlus` can run faster than `VegasFlow`.

The integration is performed using 1M samples per each iteration. We also include a warm-up of 5 iterations using 1M sampled points in which the VEGAS grid is refined after each iteration. After that the VEGAS map is kept fixed for the rest of the simulation.

We consider four different configurations of the aforementioned integrators shown in Table 3.1:

- Importance Sampling: integrator already implemented in the class `VegasFlow`
- Classic VEGAS: integrator that implements exactly the VEGAS algorithm, i.e. importance sampling and stratified sampling without redistribution of samples ($\beta_{\text{warm-up}} = \beta = 0$)
- VEGAS-VEGAS+: integrator that implements the VEGAS+ algorithm in the warm-up phase, i.e. allowing a redistribution of samples between the hypercube, then performs a classic VEGAS integration
- VEGAS+: integrator that implements the VEGAS+ algorithm

3.2 Result for Gaussian integrals

Firstly we test the previous integrators with a symmetric Gaussian placed in the center of the integration region as it was also the first example shown in the original paper of Vegas [5]:

$$I_n = \left(\frac{1}{a\sqrt{\pi}}\right)^n \int_0^1 d^n x \exp\left(-\sum_{i=1}^{n} \frac{\left(x_i - \frac{1}{2}\right)^2}{a^2}\right) \tag{3.1}$$

We benchmark the previous integral for three different dimensions: 4, 8 and 12.

3.2.1 Dimension 4

In Fig.3.1 we can observe the benchmark results for dimension 4. For a percent uncertainty of 10^{-2} or 10^{-3} all the algorithms converge in only 1 iteration after the warm-up. Looking at the accuracy reached the importance sampling and the classic VEGAS are slightly more precise compared to the VEGAS+ algorithms.

In the last row of the plot we can see the results with a target accuracy of 10^{-4} percent uncertainty. This time all the algorithms converge within 40 iterations. The

[3] The parameter α is defined in Ref[4]. Setting $\alpha = 0$ implies no grid-refinement.
[4] See footnote 3

Figure 3.1: Benchmark for the integral I_4 in Eq.(3.1). From right to left are presented the average time per iteration (after the warm-up), the number of iterations needed to reach target accuracy and the percent uncertainty reached at the end of the simulation. From the first row to the last one the target accuracies (dashed in the last column) are at 10^{-2}, 10^{-3} and 10^{-4} percent uncertainty.

Figure 3.2: Benchmark for the integral I_8 in Eq.(3.1). From right to left are presented the average time per iteration (after the warm-up), the number of iterations needed to reach target accuracy and the percent uncertainty reached at the end of the simulation. From the first row to the last one the target accuracies (dashed in the last column) are at 10^{-2}, 10^{-3} and 10^{-4} percent uncertainty.

classic VEGAS integrator converges using a couple of iterations less compared to the importance sampling of `VegasFlow`. The VEGAS+ integrators need more iterations to reach the target accuracy.

We can easily explain this behaviour since the integral in Eq.(3.1) is separable. Moreover, there is a sharp peak at the center of the integration region which favours the importance sampling. VEGAS+ is not effective in this case since we are not dealing with diagonal-structured integrands.

The average time per iteration (first column of the plot) is fairly similar for all the integrators even if the classic VEGAS algorithm, which converge in less iterations, seem to be the slower one. All the newly implemented integrators show better performances when running on GPU compared to the CPU. In particular, for the classic VEGAS integrator we observe a 3x improvements when running on the Titan V.

3.2.2 Dimension 8

In Fig.3.2 are presented the result for the integral I_n of Eq.(3.1) with $n = 8$.

The trend is similar to the previous case studied. As expected the importance sampling and the classic VEGAS are the better performing integrators. In fact, they converge using less iterations particularly for a percent uncertainty of 10^{-4} where they converge using 50-60 iterations compared to the 98-100 iterations of VEGAS+

and VEGAS/VEGAS+ hybrid.

By looking at the average time per iteration we can observe that the VEGAS+ algorithms, i.e. the VEGAS-VEGAS+ hybrid and VEGAS+, are the fastest when running on CPU. We expect similar results for the running time, in fact, excluding the warm-up phase all the new integrators converge in around 8 seconds. The `VegasFlow` integrator converges to the required accuracy in 5 seconds since the average time per iterations is better compared to the classic VEGAS one.

In a GPU environment we can see great improvements especially when the simulation involves a larger number of iterations. This is expected since, as discuss in Sect.2.2.1, the graph-mode implementation of TensorFlow is more effective when a function is called a large number of times. The importance sampling and the classic VEGAS integrators exhibit a 10x and a 8x improvement respectively in the average time per iterations, while the VEGAS+ algorithms show a speed-up by a factor 6.5.

3.2.3 Dimension 12

Next we move to dimension 12 by considering the integral I_n of Eq.(3.1) with $n = 12$.

For lower accuracies we can see that all the integrators converge using roughly the same number of iterations. For the case of a target accuracy of 10^{-4} relative tolerance the integrators exhibit different behaviours. The importance sampling and the classic VEGAS algorithms reach the accuracy required using the same number of iterations. We observe that more iterations are required on GPU but this is somewhat expected since we are dealing with MC integrators, therefore the number of iterations for the same algorithm can vary.

The VEGAS/VEGAS+ hybrid and the VEGAS+ integrators have more difficulty reaching the desired accuracy. This is due to the fact that stratified techniques become ineffective at higher dimensions. The adaptive stratified sampling aims at exploring the integration domain faster by redistributing the samples. For the case of a 12-dim Gaussian, due to the large integration domain, the redistribution can move away points from the peak resulting in a slower converge of the integral estimate. For the VEGAS/VEGAS+ hybrid, where the redistribution is stopped after the 5 warm-up iterations, it can happen that during the last redistribution the points are distributed in a less optimal configuration. This may explain the fact when running on CPU the VEGAS/VEGAS+ hybrid cannot reach the desired accuracy after the maximum number of iterations set to 1000.

Looking at the average time per iteration the results show a pattern similar to the 8-dim Gaussian. The speed-up factors, for the higher precision case, are the following: importance sampling 11.5, classic VEGAS 10.1, VEGAS/VEGAS+ hybrid 14.8 and VEGAS+ 7.8. The significant improvement in VEGAS/VEGAS+ is mainly due to the odd result observed when running on CPU, the other ones are similar to the previous cases.

3.3 HEP integrands

We also benchmark the performance of the new algorithms on integrands taken from particle physics processes. The motivation behind this choice is that we are concerned by the computation cost of fixed order calculation involved in the simulation of important experiments such as ATLAS or CMS, as already discussed in Section 1.3.2.

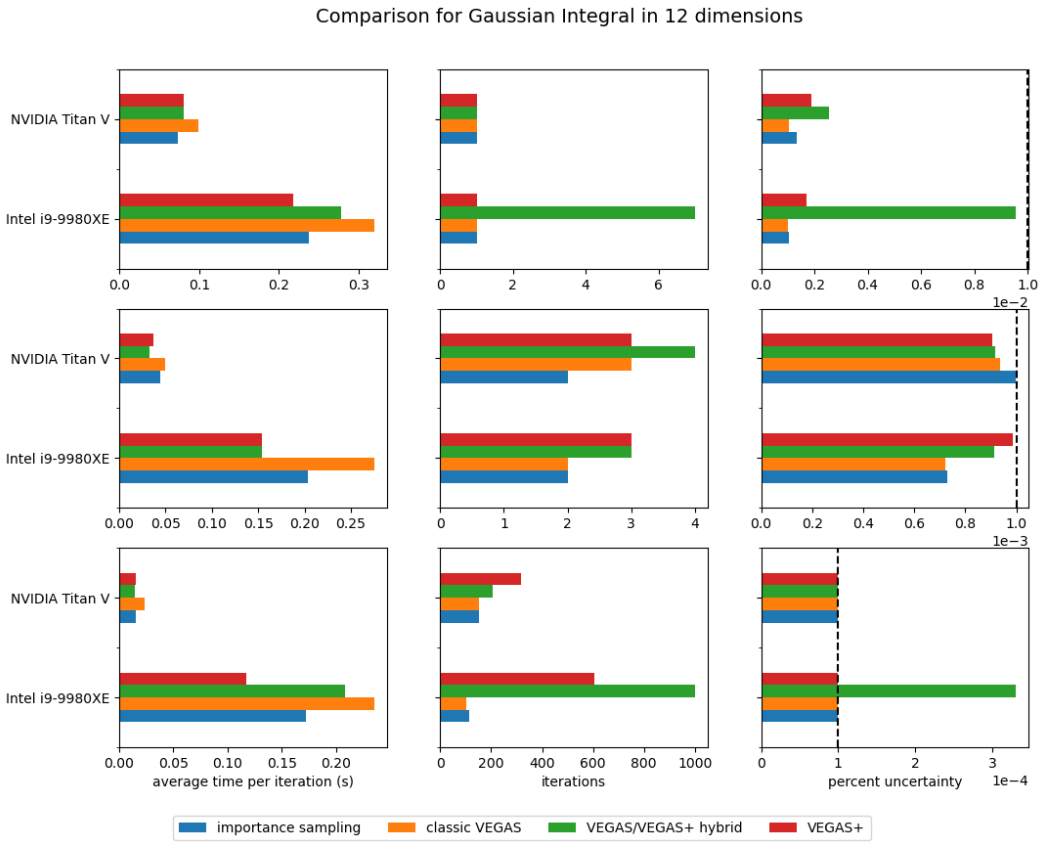

Figure 3.3: Benchmark for the integral I_{12} in Eq.(3.1). From right to left are presented the average time per iteration (after the warm-up), the number of iterations needed to reach target accuracy and the percent uncertainty reached at the end of the simulation. From the first row to the last one the target accuracies (dashed in the last column) are at 10^{-2}, 10^{-3} and 10^{-4} percent uncertainty.

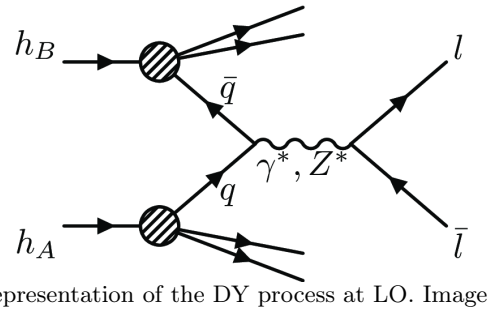

Figure 3.4: Representation of the DY process at LO. Image from Ref[25].

We are interested in seeing if any of the new implemented integrators can perform better than the importance sampling of `VegasFlow` in the computation of physical integrals.

In particular according to modern amplitude methods we use the spinor helicity formalism described in Ref[24]. We no longer consider the four-momenta of the particles k_i^μ, since the majority of the particle in the Standard Model have spin, but we work with a smaller representation of the Lorentz group, the spinor representation.

3.3.1 Drell-Yan process

The first example that we consider is the Drell-Yan (DY) process, which consists in an inclusive production of high-mass lepton pairs in hadron-hadron collision:

$$h_A + h_B \to (\gamma^* + l\bar{l}) + X \tag{3.2}$$

where X is an undetected final state, as shown in Fig.3.4.

In particular we consider the aforementioned process at the partonic level, this is, without considering the convolution with the parton density functions. We use the expression for the cross section computed at LO take from Ref[26].

The benchmark results are shown in Fig.3.5.

As for the previous cases, at lower accuracies all the integrators converges using less the 3 iterations. We can observe that the importance sampling and the classic VEGAS at a percent uncertainty of 0.001% reach the target accuracy using only one iteration after the warm-up. This trend continues for higher accuracies where there is a major different in the number of iterations needed to satisfy the precision requirements. Classic VEGAS is the better performing integrator compared to the importance sampling. The VEGAS+ algorithms have a more difficult time reaching the desired accuracy, they can need up to 5 times the number of iterations required to the classic VEGAS integrator.

Regarding the average time per iteration, the importance sampling is the fastest integrators followed by the classic VEGAS algorithm. The other integrators can take up to twice the time when running on GPU compared to the faster ones. All the integrators show a better performance when running on GPU compared to CPU. The importance sampling and the classic VEGAS methods have a speed-up factors of 7.7 and 6 respectively.

The worst performance of the VEGAS+ methods suggests that the integral doesn't have multiple peaks, but rather a sharp peak which is easily found by the VEGAS grid.

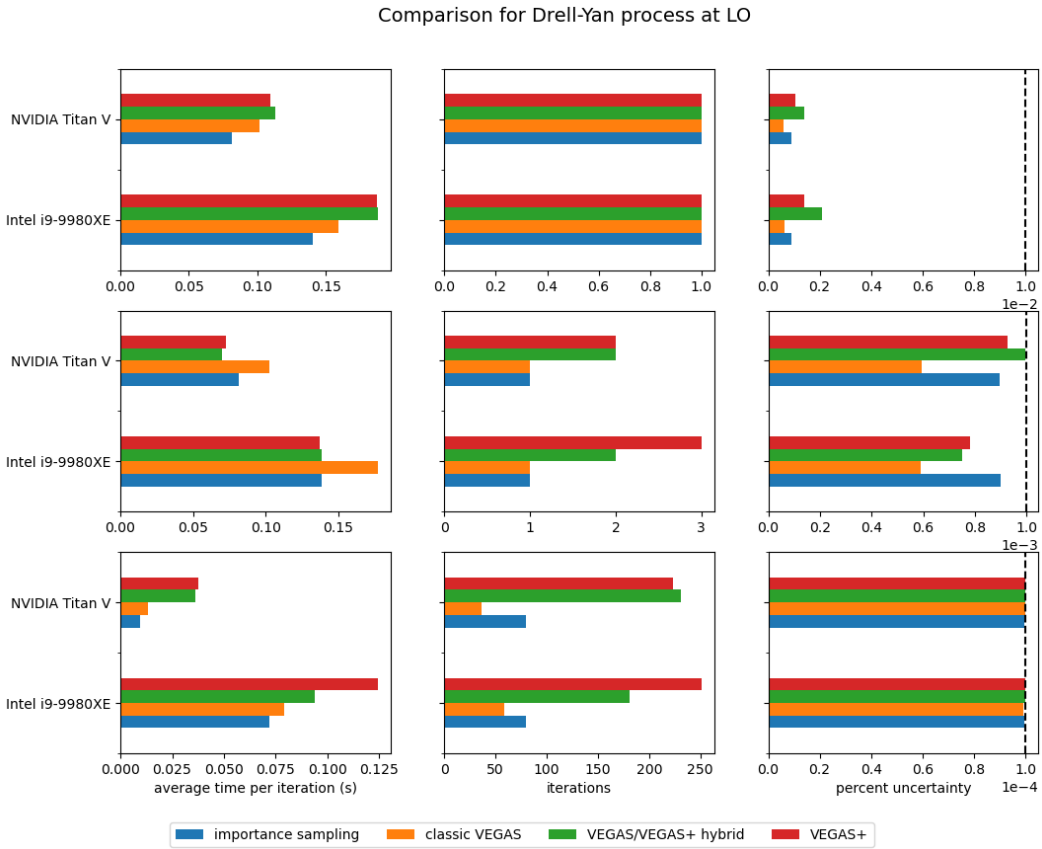

Figure 3.5: Benchmark for the partonic level cross section for the DY photon induced process at LO [26]. From right to left are presented the average time per iteration (after the warm-up), the number of iterations needed to reach target accuracy and the percent uncertainty reached at the end of the simulation. From the first row to the last one the target accuracies (dashed in the last column) are at 10^{-2}, 10^{-3} and 10^{-4} percent uncertainty.

3.3.2 Single Top Production

Another interesting process to consider is the production of the t-quark since it is important for understanding many properties such as the top quark mass, his Cabbibo-Kobayashi-Maskawa matrix element V_{tb} and many others.

Although the top quark is mainly produced via to the strong interactions, we consider his production governed by the weak interactions since it is relevant for several events at the LHC. There are several experimentally distinguishable ways to produce the single top quark, in this thesis we consider the t-channel process

$$W^* b \to t \tag{3.3}$$

which has the largest cross section with $\sqrt{s} = 8$ TeV at the LHC [27].

In particular we consider the process of Eq.(3.3) at LO at the partonic level. The physical parameters chosen are $m_t = 173.2$ GeV and $\sqrt{s} = 8$ TeV.

The results are shown in Fig.3.6. Even though this integral has the same dimension of the previous one we can see that the plots exhibit significant differences. In the first place, for lower accuracies even if all the integrators converge in only 1 iterations after the warm-up VEGAS+ seem to be the faster both on GPU and CPU.

This happens also at a higher accuracy on CPU, while on GPU the importance sampling of `VegasFlow` is the fastest methods looking at the average time per iterations. In fact, the importance sampling has a speed-up factor of 10.9 compared to the 4.32 of VEGAS+.

The most interesting aspect for this integral is the number of iterations needed to reach an accuracy of 0.0001% percent uncertainty. This is the first case where we see that the importance sampling alone is by far the less efficient method. Both on GPU and CPU `VegasFlow` needs at least 50 iterations to reach the required precision. On the other hand, all the new implemented algorithms converge using less than 10 iterations and VEGAS+ is the most accurate overall.

We can conclude that the adaptive stratified sampling technique of VEGAS+ seem to be particularly suited for the single-top production integrand.

3.3.3 Vector Fusion Boson Production Higgs at LO

Next we study a more complicated involving the Higgs boson H, which can be produced at the LHC through its Yukawa coupling to the top quark or through its coupling to electroweak bosons. In particular we consider the vector fusion boson process shown in Fig.3.7 which is the second-largest inclusive production mode for Higgs bosons, amounting to about 10% of the dominant gluon fusion process [28].

For this process we don't just compute the integral at the partonic level, but we also include the convolution with the PDFs. The parton density functions are including in the `VegasFlow` environment using the `PDFFlow` software [29, 30], which is implemented using the TensorFlow library and enable us to evaluate a generic set of PDFs on GPU. The integrand used is taken from Ref[30].

The results are shown in Fig.3.8.

Looking at the number of iterations needed to reach the target accuracy we can observe that the classic VEGAS algorithm is the more efficient integrator both at low and high precisions. The importance sampling method cannot outperform the other integration algorithms and in some cases struggle to converge to the required accuracy. This is interesting since we are computing a multi-dimensional integral of

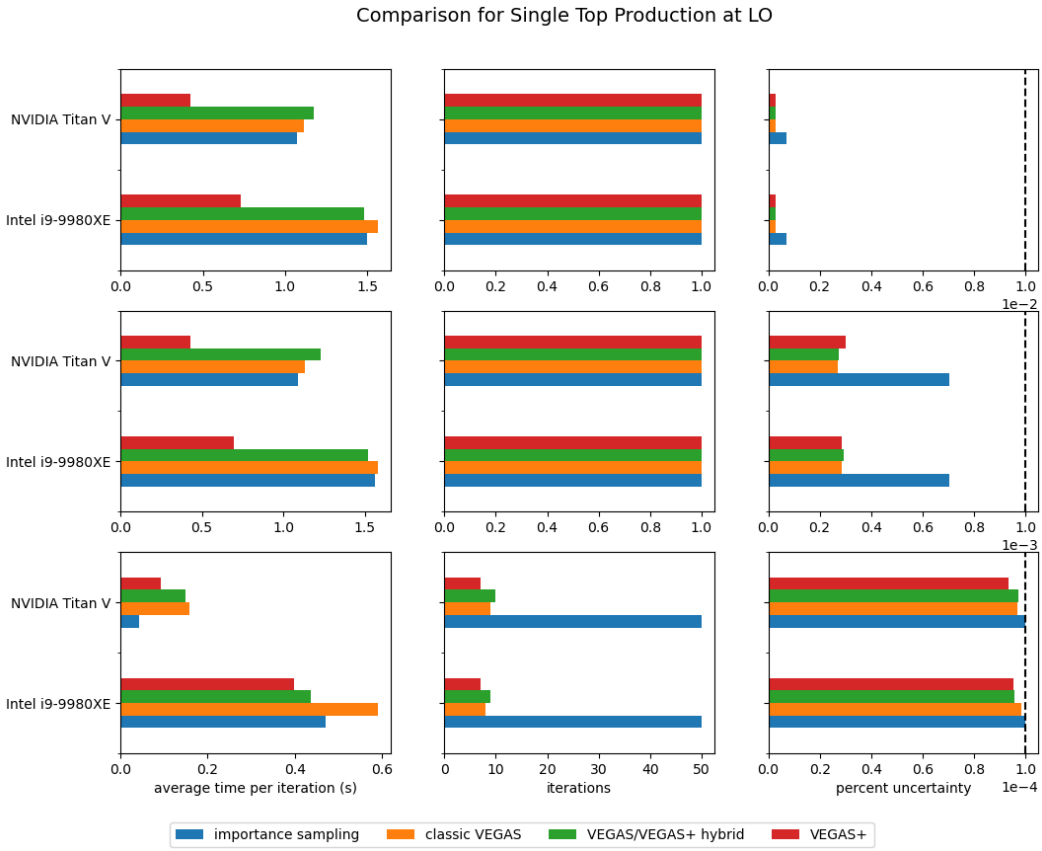

Figure 3.6: Benchmark for the partonic level cross section for the single t-quark production (t-channel) at LO [27]. From right to left are presented the average time per iteration (after the warm-up), the number of iterations needed to reach target accuracy and the percent uncertainty reached at the end of the simulation. From the first row to the last one the target accuracies (dashed in the last column) are at 10^{-2}, 10^{-3} and 10^{-4} percent uncertainty.

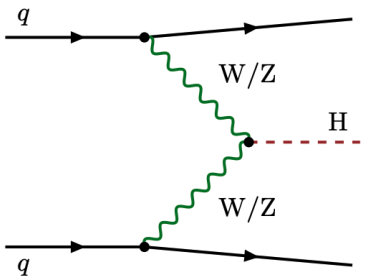

Figure 3.7: Born-level vector boson fusion process. Image from Ref[28].

dimension 6. Usually we expect that the (adaptive) stratified sampling techniques have only small effects for high-dimensional integrals.

For the computational times we can observe how the average time per iterations decreases both on GPU and CPU if we consider more iterations, i.e. by going for higher precisions. The first iteration can take up to 6 seconds and we can reduce the average time per iterations to a few tenths of a second thanks to hardware acceleration and to the graph mode implementation of TensorFlow.

Regarding the average time per iteration, we can see that VEGAS+ and VEGAS/VEGAS+ hybrid are the fastest integrators, followed by the importance sampling and the classic VEGAS algorithm. We also observed significant improvements when the integration run on GPU with speed-up factors between 6 and 4.

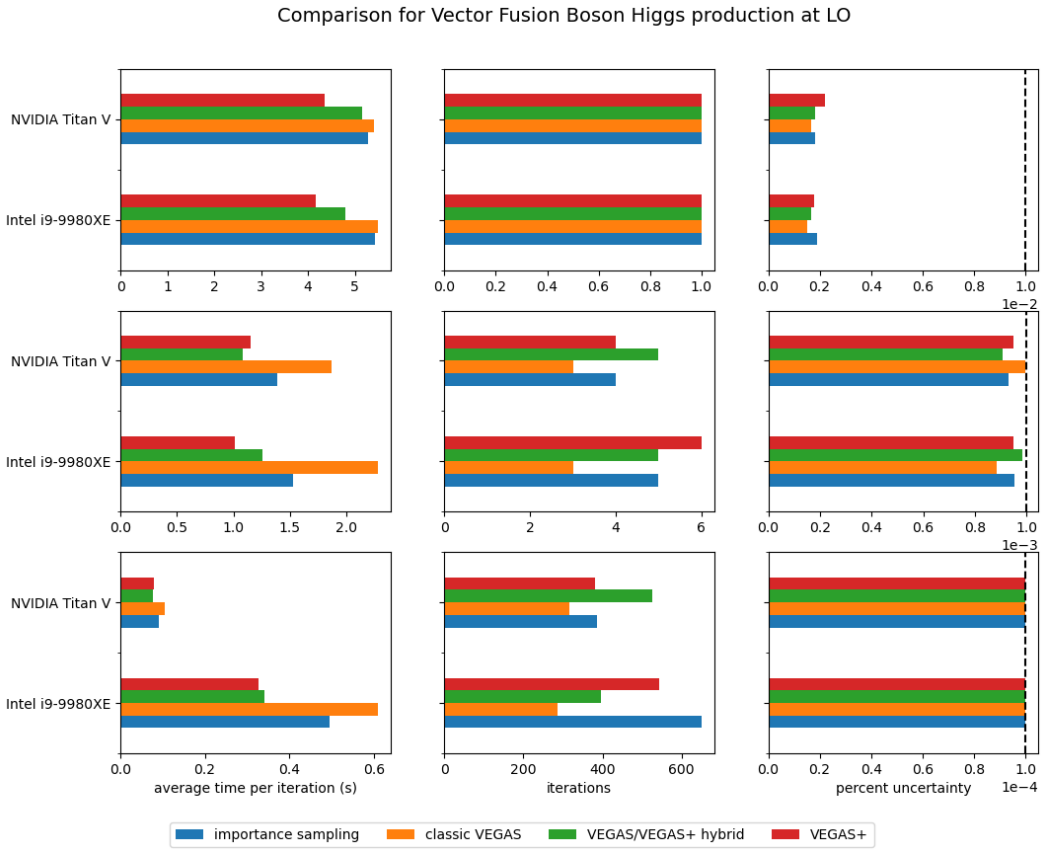

Figure 3.8: Benchmark for the cross section for the vector fusion boson (VFB) Higgs production at LO [27]. The set of PDFs is `NNPDF31_nnlo_as_0118/0` From right to left are presented the average time per iteration (after the warm-up), the number of iterations needed to reach target accuracy and the percent uncertainty reached at the end of the simulation. From the first row to the last one the target accuracies (dashed in the last column) are at 10^{-2}, 10^{-3} and 10^{-4} percent uncertainty.

3.4 Recipe for the reader

We now compare all the results obtained in the previous section. We would like to see if we can suggest to the user which integrators works best depending on the integral to compute. We base this analysis on the average time per iteration and on the number of iterations needed for the precision of 0.0001% percent uncertainty.

3.4.1 Average time per iteration

In Fig.3.9 we show the average time per iteration (after the warm-up) for all the integrands used in the benchmark when running on the Intel i9-9980XE.

As expected all the integrators requires more time per iteration when computing the single top and the higgs production due to the more complex integrands to evaluate. However, it is interesting to observe that for the previous cases the average time per iteration is shorter for the two VEGAS+ algorithms compared to the classic VEGAS and the importance sampling of `VegasFlow`. This is very important for our analysis since we are concerned by the CPU usage when integrating physical integrands.

The same cannot be said for the DY integral although we can observe that the time differences in all the integrators is quite limited due to the low-dimensional integration domain.

By looking at the standard gaussian integrals for all the integrators the time increases as the dimension increases which is expected. Furthermore, the plot shows that using the VEGAS+ integrator the average time per iteration is almost always shorter compared to all the other integrators. This is true even for the 12-dim gauss integral despite the high dimension.

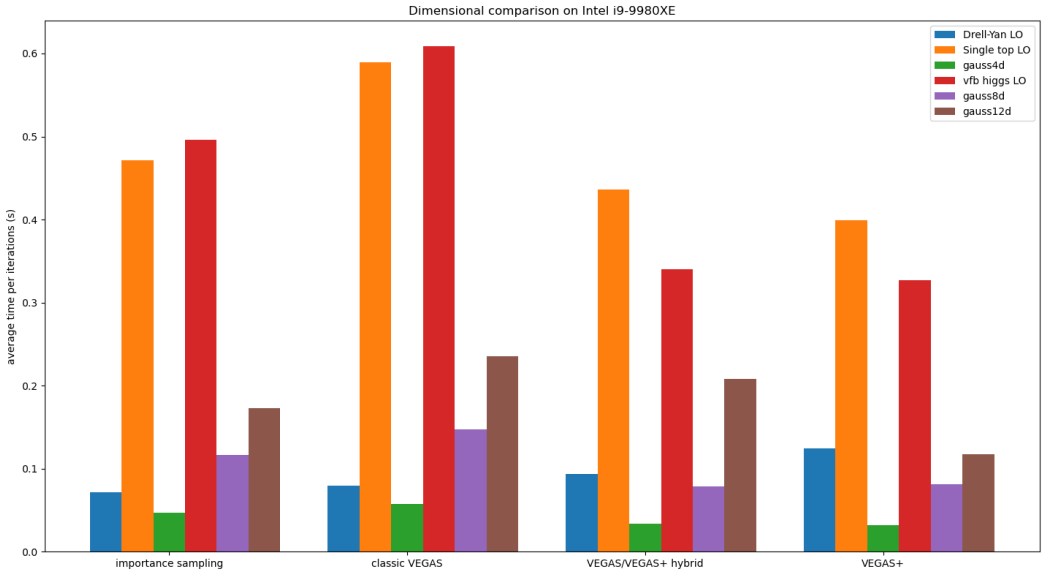

Figure 3.9: Average time per iterations (after the warm-up) for all the previously studied cases. The results are sorted by the integrand dimension for each integrator. All the integration are performed on the Intel i9-9980XE CPU.

In Fig.3.10 we present the same plot when all the integrations are performed on the NVIDIA Titan V GPU. We can see that all the times are shortened and have similar values except for the more complicated integrals (single top and higgs production).

Overall the fastest integrator seem to be the importance sampling. In fact, there is a great improvement on the computational time of the single top integral compared to the CPU case. That same time cannot be reached by the other integrators when running on GPU.

The case of the VFB Higgs production is also interesting. The VEGAS/VEGAS+ hybrid and the VEGAS+ algorithms are faster than the importance sampling as for

the CPU case. This means that the new integrators can provide a solid alternative in a GPU environment when computing complicated physical integrands.

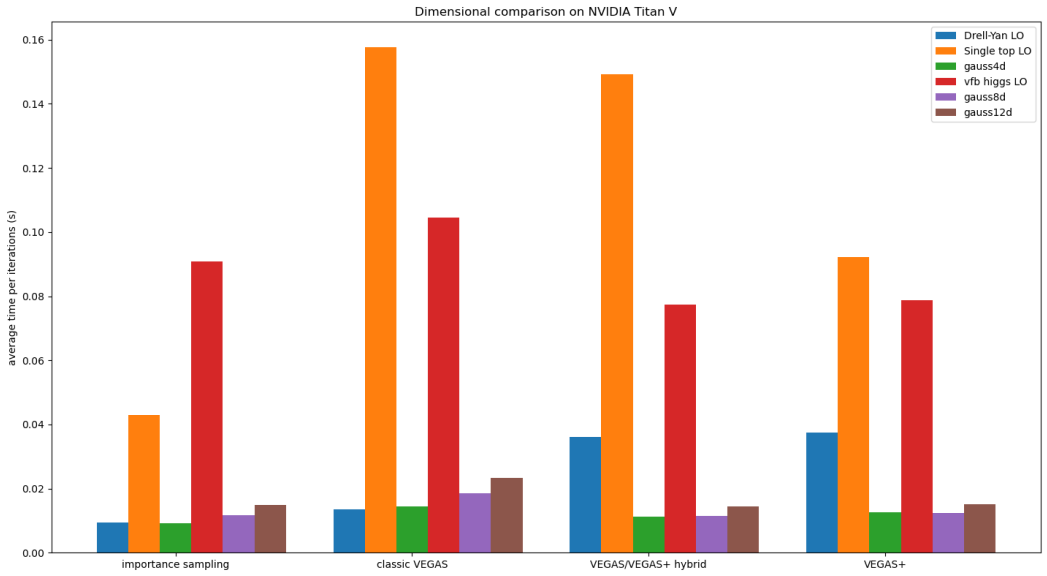

Figure 3.10: Average time per iterations (after the warm-up) for all the previously studied cases. The results are sorted by the integrand dimension for each integrator. All the integration are performed on the NVIDIA Titan V GPU.

3.4.2 Number of iterations

We consider now the number of iterations needed to reach the target accuracy of 0.0001 percent uncertainty for all the previous integrands. Since the number of iterations are independent on the hardware environment (CPU or GPU) we have decided to present the results from the integration on CPU.

The results are shown in Fig.3.11. The integrator that in general converge using less iterations is the classic VEGAS algorithm, i.e. without the redistribution of samples. It is more accurate than the importance sampling for all the physical integrands: DY, single top and Higgs production. In fact, for the Higgs production integrand classic VEGAS converges using less than half iterations used by the importance sampling.

For the single top production all the new implemented algorithms can outperform the importance sampling, as already observed in Section 3.3.2.

As expected the adaptive stratified sampling methods needs more iterations as the dimensionality of the integrands increases, especially if the peaks are easy to find by the VEGAS grid, as for the case of a Gaussian distribution with a single peak.

3.4.3 Final comments

The newly implemented integrators in the `VegasFlow` library provide solid alternatives to the importance sampling algorithm already implemented in the library.

By looking at the average time per iterations,we have shown that the integrators that employs the VEGAS+ algorithms can outperform the importance sampling method when dealing with high-dimensional and complicated integrals such as the VFB Higgs production. For the case of gaussian integration all the integrators

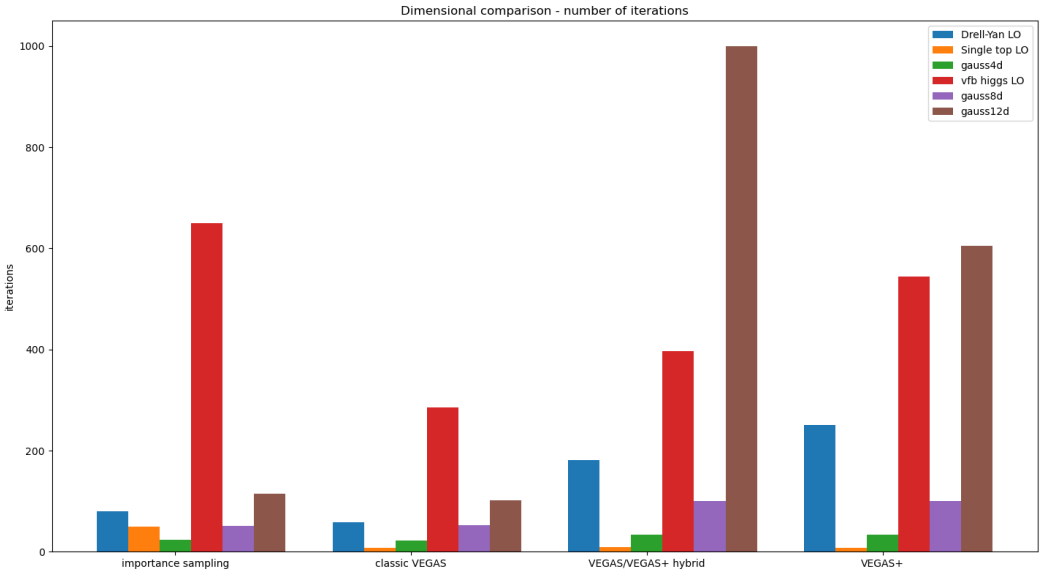

Figure 3.11: Iterations needed to reach the accuracy of 10^{-4} percent uncertainty (after the warm-up) for all the previously studied cases. The results are sorted by the integrand dimension for each integrator. All the integration are performed on the Intel i9-9980XE CPU.

exhibit the same performance, especially on GPU, with a few differences on CPU that seem to favour the VEGAS+ algorithms.

For low-dimensional physical integrands such as the DY and the single top production the importance sampling perform better especially in the case of the DY process both on CPU and GPU. In computing the single top integral the importance sampling performs better on GPU while on CPU the VEGAS+ algorithm seem to be the fastest.

When comparing the number of iterations we have observed that the classic VEGAS is the more efficient integrator in general. Therefore, even if his computational time are not as fast the other integrators, is still a valuable option since the simulation will require less iterations.

The VEGAS/VEGAS+ hybrid and the VEGAS+ algorithms converge using a small number of iterations when integrating physical integrands such as the single top or the Higgs production. This, combined with the short average times per iteration on CPU and on GPU, makes these methods a very useful tool when computing particle process related integrals.

Chapter 4

Conclusions

In this thesis we have considered the problem of solving high-dimensional integrals in the context of HEP. These integrals are usually solved by means of MC integration techniques which requires long computational time as well as high CPU resources. The cost of these calculations is currently driving the budget of important experiments such as ATLAS or CMS [14].

To shorten the computational times we focus on implementing new integration algorithms, which can take advantage of hardware acceleration devices such as GPU or multi-thread CPUs. In particular we have considered a new algorithm proposed in Ref[4], called VEGAS+, which consists in a modification of the classic VEGAS algorithm [5], ubiquitous in the particle physics community. The algorithm has been implemented within the `VegasFlow` library [3], which enable us to run our computations using hardware acceleration thanks to Google's TensorFlow library [31].

We have compared the performance of different variations of the new algorithm with the importance sampling "à la" VEGAS already implemented in the `VegasFlow` library. The benchmark was performed by using classical integrands, such gaussian distributions, as well as integrands taken from common particle physics processes, such as the Drell-Yan process, the single top production and the vector fusion boson Higgs production. In particular we run the integrations on a professional-grade CPU and on a professional-grade GPU to quantify the benefits of hardware acceleration.

The results show that the new integrators benefits from highly parallel scenarios, with speed-up factors up to 10 when comparing the average time per iteration on CPU and GPU. The new integrators are also more accurate when dealing with particular HEP integrads, i.e. they converge using less iterations of the chosen algorithm. For example, in the case of the single top production at LO we observe that all the newly implemented integrators converge using only one fifth of the iterations needed by the importance sampling algorithm. Also for the case of the vector fusion boson Higgs production at LO, which involves the computation of a 6-dimensional integral, the new algorithms are more accurate than the importance sampling method.

We have also presented to the reader a full comparison of all the results obtained from each integral analysed. The aim is to suggest which integrator performs better by looking at the average time per iterations, as well as, the number of iterations needed to reach the target accuracy. On CPU, except for the Drell-Yan integrand, the integrator which implements the full VEGAS+ algorithm is the fastest. While on GPU, we have found that the importance sampling seem to be the more fast

integrator, except for the Higgs integrand. The comparison of the number of iterations shows that the classic VEGAS algorithm, here presented as a variation of the VEGAS+ method, is overall the most efficient integrator, despite his slower computational times.

All the new integrators implemented are publicly on github at `https://github.com/N3PDF/vegasflow`.

We believe that, based on the accuracy and the computational times, the new integrators implemented are a very useful tool when computing complicated integrals, especially in the field of HEP. Moreover, our implementation aims at empowering the new VEGAS+ algorithm by enabling to run the integrations in all kinds of hardware supported by the TensorFlow library.

For future developments it will be interesting to implement new algorithms in the `VegasFlow` library, by exploiting the design of the library which enable us to implement new integrators simply as derived class of the `MonteCarloFlow` class. In particular, there are several algorithms which implement an importance sampling based on machine-learning, such as neural network or boosted decision trees [16], which can outperform than the VEGAS algorithm as in Ref[32]. By implementing these techniques in the `VegasFlow` library we will combined the efficiency of these integrators with hardware acceleration, thus lowering the computational cost of these complex computations.

Appendix A

Integrating with VegasFlow

A.1 Basic Usage

In this appendix we present directly how to use the `VegasFlow` library and the new algorithms implemented in the `VegasFlowPlus` class. The integration of a function is done in three steps:

1. Create an instance of the integrator. At this step the user needs to set the dimension and the number of events per iterations. For the `VegasFlow` integrator this step can be performed as follows:

   ```
   from vegasflow import VegasFlow

   dims = 3
   n_calls = int(1e7)
   vegas_instance = VegasFlow(dims, n_calls)
   ```

2. Compile the integrand function. Obviously the user need to specify to the integrator which function will be integrated. The integrand can be implemented as a simple python function that accept three arguments: the number of dimensions `n_dim`, the weight of each sampled point `weight` and the an array of random numbers `xarr`.

   ```
   import tensorflow as tf

   def example_integrand(xarr, n_dim=None, weight=None):
           s = tf.reduce_sum(xarr, axis=1)
           result = tf.pow(0.1/s, 2)
           return result

   vegas_instance.compile(example_integrand)
   ```

   As we can see the function is defined using only `TensorFlow` primitives, since it allows for the usage of all the hardware `TensorFlow` is compatible with. In particular when is called the method `compile` the integrand is registered and it is compiled using the decorator `tf.function` that enable us to use the benefits of the graph-mode implementation discussed in Sect. 2.2.1.

3. Run the integration. After having completed the setup we just need to tell the integrators for how many iterations we need to run the simulation. The integration is performed using the method `run_integration(n_iter)` as follows:

```
n_iter = 5
result = vegas_instance.run_integration(n_iter)
```

The output variable `result` is a tuple variable where the first element is the result of the integration while the second element is the error of the integration.

4. The program will display the following output during the integrations.

```
[INFO] Result for iteration 0: 8.607e-03 +/- 4.039e-05(took 16.11334 s)
[INFO] Result for iteration 1: 8.628e-03 +/- 5.101e-06(took 9.36287 s)
[INFO] Result for iteration 2: 8.631e-03 +/- 2.715e-06(took 9.58453 s)
[INFO] Result for iteration 3: 8.633e-03 +/- 2.292e-06(took 9.56779 s)
[INFO] Result for iteration 4: 8.632e-03 +/- 2.161e-06(took 9.55856 s)
[INFO]  > Final results: 0.00863171 +/- 1.31392e-06
```

The final result is a weighted average of the results obtained in each iterations.

Since we are using the integrator `VegasFlow` the VEGAS grid will be refined after each iteration using the importance sampling algorithm presented in section 2.1.1.

A.2 How to use the `VegasFlowPlus` class

If we want to use the new integrators implemented we simply need to import the `VegasFlowPlus` class. The setup is quite similar with the only difference that we can set the parameter `adaptive` to `True` if we want to use the VEGAS+ algorithm [4] or to `False` if the we want to use the classic VEGAS algorithm [5]. By default the `adaptive` flag is set to `True`.

```
from vegasflow import VegasFlowPlus

dim = 4
n_calls = int(1e6)
# VEGAS+ integrator: importance sampling + adaptive stratified sampling
vegas+_instance =  VegasFlowPlus(dims, n_calls)
# VEGAS integrator: importance sampling + stratified sampling
vegas_instance = VegasFlowPlus(dims, n_calls, adaptive = False )
```

The integrand definition is identical to the `VegasFlow` case if we are using the VEGAS integrator. For the VEGAS+ integrator, as already discussed in Section 2.2.2, we need to specify an input signature due to the fact that the VEGAS+ integrator uses a different number of events in each iteration.

For the case of a Gaussian integral we show how to add this signature to the function:

```
@tf.function(input_signature=[
tf.TensorSpec(shape=[None,dim], dtype=tf.float32),
tf.TensorSpec(shape=[], dtype=tf.int32),
tf.TensorSpec(shape=[None], dtype=tf.float32)
]
)
def symgauss(xarr, n_dim=None, weight=None, **kwargs):
"""symgauss test function"""
if n_dim is None:
n_dim = xarr.shape[-1]
a = tf.constant(0.1, dtype=DTYPE)
n100 = tf.cast(100 * n_dim, dtype=DTYPE)
```

```
pref = tf.pow(1.0 / a / np.sqrt(np.pi), float_me(n_dim))
coef = tf.reduce_sum(tf.range(n100 + 1))
coef += tf.reduce_sum(tf.square((xarr - 1.0 / 2.0) / a), axis=1)
coef -= (n100 + 1) * n100 / 2.0
return pref * tf.exp(-coef)
```

By adding `None` to the signature we allow `TensorFlow` to build only one graph corresponding to the function `symgauss`, avoiding the creation of multiple graphs for the same function (*retracing*) .

After that the integration is performed exactly in the same way using the method `run_integration(n_iter)`

```
n_iter = 5
# VEGAS+ result
result_vegas+ = vegas+_instance.run_integration(n_iter)
# VEGAS result
result_vegas = vegas_instance.run_integration(n_iter)
```

Bibliography

[1] William H. Press et al. "Numerical Recipes in FORTRAN: The Art of Scientific Computing". In: (Sept. 1992).

[2] Christina Gao, Joshua Isaacson, and Claudius Krause. *i-flow: High-dimensional Integration and Sampling with Normalizing Flows*. Jan. 2020.

[3] Stefano Carrazza and Juan M. Cruz-Martinez. "VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms". In: *Comput. Phys. Commun.* 254 (2020), p. 107376. DOI: 10.1016/j.cpc.2020.107376. arXiv: 2002.12921 [physics.comp-ph].

[4] G. Peter Lepage. "Adaptive multidimensional integration: VEGAS enhanced". In: *J. Comput. Phys.* 439 (2021), p. 110386. DOI: 10.1016/j.jcp.2021.110386. arXiv: 2009.05112 [physics.comp-ph].

[5] G. Peter Lepage. "A New Algorithm for Adaptive Multidimensional Integration". In: *J. Comput. Phys.* 27 (1978), p. 192. DOI: 10.1016/0021-9991(78)90004-9.

[6] William H. Press and Glennys R. Farrar. "RECURSIVE STRATIFIED SAMPLING FOR MULTIDIMENSIONAL MONTE CARLO INTEGRATION". In: (Dec. 1989).

[7] Michael E. Peskin and Daniel V. Schroeder. *An Introduction to quantum field theory*. Reading, USA: Addison-Wesley, 1995. ISBN: 978-0-201-50397-5.

[8] Matthew D. Schwartz. "TASI Lectures on Collider Physics". In: *Proceedings, Theoretical Advanced Study Institute in Elementary Particle Physics : Anticipating the Next Discoveries in Particle Physics (TASI 2016): Boulder, CO, USA, June 6-July 1, 2016*. Ed. by Rouven Essig and Ian Low. 2018. DOI: 10.1142/9789813233348_0002. arXiv: 1709.04533 [hep-ph].

[9] John Collins. *Foundations of perturbative QCD*. Vol. 32. Cambridge University Press, Nov. 2013. ISBN: 978-1-107-64525-7, 978-1-107-64525-7, 978-0-521-85533-4, 978-1-139-09782-6.

[10] Taizo Muta. *Foundations of Quantum Chromodynamics: An Introduction to Perturbative Methods in Gauge Theories, (3rd ed.)* 3rd. Vol. 78. World scientific Lecture Notes in Physics. Hackensack, N.J.: World Scientific, 2010. ISBN: 978-981-279-353-9.

[11] R. Keith Ellis, W. James Stirling, and B. R. Webber. *QCD and collider physics*. Vol. 8. Cambridge University Press, Feb. 2011. ISBN: 978-0-511-82328-2, 978-0-521-54589-1.

[12] Peter Skands. "Introduction to QCD". In: *Theoretical Advanced Study Institute in Elementary Particle Physics: Searching for New Physics at Small and Large Scales*. July 2012. DOI: 10.1142/9789814525220_0008. arXiv: 1207.2389 [hep-ph].

[13] Inc. Wolfram Research. *Mathematica, Version 12.3*. Champaign, IL, 2021. URL: https://www.wolfram.com/mathematica.

[14] Andy Buckley. "Computational challenges for MC event generation". In: *J. Phys. Conf. Ser.* 1525.1 (2020), p. 012023. DOI: 10.1088/1742-6596/1525/1/012023. arXiv: 1908.00167 [hep-ph].

[15] *The ATLAS Collaboration Software and Firmware*. Tech. rep. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNO SOFT-PUB-2021-001. Geneva: CERN, May 2021. URL: https://cds.cern.ch/record/2767187.

[16] Joshua Bendavid. "Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks". In: (June 2017). arXiv: 1707.00028 [hep-ph].

[17] John M. Campbell, R. Keith Ellis, and Walter T. Giele. "A Multi-Threaded Version of MCFM". In: *Eur. Phys. J. C* 75.6 (2015), p. 246. DOI: 10.1140/epjc/s10052-015-3461-2. arXiv: 1503.06182 [physics.comp-ph].

[18] John Campbell and Tobias Neumann. "Precision Phenomenology with MCFM". In: *JHEP* 12 (2019), p. 034. DOI: 10.1007/JHEP12(2019)034. arXiv: 1909.09117 [hep-ph].

[19] Thomas Gehrmann et al. "Jet cross sections and transverse momentum distributions with NNLOJET". In: *PoS* RADCOR2017 (2018). Ed. by Andre Hoang and Carsten Schneider, p. 074. DOI: 10.22323/1.290.0074. arXiv: 1801.06415 [hep-ph].

[20] J. Alwall et al. "The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations". In: *JHEP* 07 (2014), p. 079. DOI: 10.1007/JHEP07(2014)079. arXiv: 1405.0301 [hep-ph].

[21] T. Gleisberg et al. "Event generation with SHERPA 1.1". In: *JHEP* 02 (2009), p. 007. DOI: 10.1088/1126-6708/2009/02/007. arXiv: 0811.4622 [hep-ph].

[22] Peter Lepage. *gplepage/vegas: vegas version 4.0.1*. Version v4.0.1. May 2021. DOI: 10.5281/zenodo.4746454. URL: https://doi.org/10.5281/zenodo.4746454.

[23] Juan Cruz-Martinez and Stefano Carrazza. *N3PDF/vegasflow: vegasflow v1.0*. Version v1.0. Feb. 2020. DOI: 10.5281/zenodo.3691926. URL: https://doi.org/10.5281/zenodo.3691926.

[24] Lance J. Dixon. "A brief introduction to modern amplitude methods". In: (Oct. 2013). Comments: 48 pages, 15 figures. Lectures presented at the 2012 European School of High-Energy Physics and at TASI 2013, 31–67. 48 p. DOI: 10.5170/CERN-2014-008.31. arXiv: 1310.5353. URL: https://cds.cern.ch/record/1613349.

[25] Florian Bechtel and Peter Schleper. "The Underlying Event in Proton-Proton Collisions". In: (June 2021).

[26] S. Carrazza et al. "PineAPPL: combining EW and QCD corrections for fast evaluation of LHC processes". In: *Journal of High Energy Physics* 2020.12 (Dec. 2020). ISSN: 1029-8479. DOI: 10.1007/jhep12(2020)108. URL: http://dx.doi.org/10.1007/JHEP12(2020)108.

[27] Mathias Brucherseifer, Fabrizio Caola, and Kirill Melnikov. "On the NNLO QCD corrections to single-top production at the LHC". In: *Physics Letters B* 736 (Sept. 2014), pp. 58–63. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2014.06.075. URL: http://dx.doi.org/10.1016/j.physletb.2014.06.075.

[28] J. Cruz-Martinez et al. "Second-order QCD effects in Higgs boson production through vector boson fusion". In: *Physics Letters B* 781 (June 2018), pp. 672–677. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2018.04.046. URL: http://dx.doi.org/10.1016/j.physletb.2018.04.046.

[29] Stefano Carrazza, Juan M. Cruz-Martinez, and Marco Rossi. "PDFFlow: Parton distribution functions on GPU". In: *Computer Physics Communications* 264 (July 2021), p. 107995. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.107995. URL: http://dx.doi.org/10.1016/j.cpc.2021.107995.

[30] Juan M. Cruz-Martinez, Marco Rossi, and Stefano Carrazza. *N3PDF/pdfflow: PDFFlow 1.2.1*. Version v1.2.1. June 2021. DOI: 10.5281/zenodo.4903010. URL: https://doi.org/10.5281/zenodo.4903010.

[31] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[32] Christina Gao, Joshua Isaacson, and Claudius Krause. "i-flow: High-dimensional Integration and Sampling with Normalizing Flows". In: *Mach. Learn. Sci. Tech.* 1.4 (2020), p. 045023. DOI: 10.1088/2632-2153/abab62. arXiv: 2001.05486 [physics.comp-ph].

[33] Andrea Dainese et al. *Report on the Physics at the HL-LHC, and Perspectives for the HE-LHC*. Tech. rep. Geneva, Switzerland, 2019. DOI: 10.23731/CYRM-2019-007. URL: http://cds.cern.ch/record/2703572.

[34] Stefano Carrazza, Juan M. Cruz-Martinez, and Marco Rossi. "PDFFlow: Parton distribution functions on GPU". In: *Comput. Phys. Commun.* 264 (2021), p. 107995. DOI: 10.1016/j.cpc.2021.107995. arXiv: 2009.06635 [hep-ph].

[35] Juan Cruz-Martinez, Marco Rossi, and Stefano Carrazza. *N3PDF/pdfflow*. Version v0.0.1b1. July 2020. DOI: 10.5281/zenodo.3964191. URL: https://doi.org/10.5281/zenodo.3964191.