Multi-Dimensional Integration – A Numerical Study

by

G. Dheeraj Simhachalam Naidu

(210121023)

under guidance of

Associate Prof. M C Kumar

To the

DEPARTMENT OF PHYSICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

SOURCE CODE

The source code for the implementation algorithm presented in this report are available at the Github and can be accessed through this hyperlink.

CHAPTER 1

INTRODUCTION

1.1 Numerical Methods

Numerical methods are mathematical techniques used to solve problems that are difficult or impossible to solve analytically. These problems often arise in various fields such as physics, engineering, finance, and computer science. Numerical methods involve approximating solutions using computations, algorithms, and iterative processes. They require careful consideration of factors such as accuracy, stability, and computational efficiency. Approximations are necessary when we encounter procedures that cannot be solved analytically, like the standard normal cumulative distribution function.

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt$$

In this case, an exact solution is not available for $\Phi(x)$ other than for $x=0$ and $x \to \infty$. For other values of x where an exact solution is not available, one may solve the problem by using approximate techniques.

1.2 Examples of Numerical Methods

In this section, we'll explore several numerical methods like Fourth-Order Runge-Kutta (RK4) method and The Simpson's 1/3 rule method that offer solutions to various mathematical problems, providing insights into their applications and computational approaches.

1.2.1 The Fourth-Order Runge-Kutta (RK4) method:

RK4, or the Fourth-Order Runge-Kutta method, is a widely used numerical technique for solving ordinary differential equations (ODEs). It's an iterative method that provides a balance between accuracy and computational efficiency. Here's how it works:

1. **Initial Conditions**: Given an initial value problem for an ODE:

   $$\frac{dy}{dx} = f(x, y)$$

   with an initial condition $y(x_0) = y_0$, where $f(x, y)$ is a given function representing the rate of change of y with respect to x, and (x_0, y_0) is the initial point.

2. **Discretization of the Interval**: The interval over which the solution is sought ($x_0 \leq x \leq x_n$) is divided into smaller steps of length h (step size).
3. **Iteration**: At each step x_n, the RK4 method computes the slope of the function $f$at several points to estimate the change in y over the interval h. It then takes a weighted average of these slopes to update the value of y at the next step x_{n+1}.
   - **Step 1**: Compute $k_1 = h.f(x_n, y_n)$, where $k_1$is the slope at the beginning of the interval.
   - **Step 2**: Compute $k_2 = h.f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$, where k_2 is the slope at the midpoint of the interval using the Euler method with k_1.
   - **Step 3**: Compute $k_3 = h.f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$, similar to step 2 but using k_2.
   - **Step 4**: Compute $k_4 = h.f(x_n + h, y_n + k_3)$, where k_4 is the slope at the end of the interval.
4. **Update the Solution**: Use a weighted sum of these slopes to update the value of y:

   $$y_{n+1} = \frac{1}{6}(y_n + k_1 + 2k_2 + 2k_3 + k_4)$$

5. **Repeat**: Repeat steps 3 and 4 until the desired endpoint x_f is reached.

RK4 is called a fourth-order method because the local truncation error (the error incurred at each step) is O(h^5), meaning it's proportional to the fifth power of the step size$h$. This makes it more accurate than simpler

methods like Euler's method, but it's also more computationally intensive. However, RK4 strikes a good balance between accuracy and computational cost, which is why it's widely used in practice for solving ODEs.

1.2.2 Simpson's 1/3 Rule Method

The Simpson's 1/3 rule method, also known as Simpson's rule, is a numerical technique used for approximating the definite integral of a function over a given interval. It's an improvement over simpler methods like the trapezoidal rule and provides more accurate results.

Here's how Simpson's 1/3 rule method works:

1. **Dividing the Interval**: The method starts by dividing the interval $[a, b]$ into an even number of subintervals. Let's say we divide it into 'n' subintervals, where 'n' is an even number.
2. **Approximating the Curve**: Within each subinterval, Simpson's rule approximates the curve by a second-degree polynomial (a quadratic function). This is done by fitting a parabola to three points: the endpoints of the subinterval and its midpoint.
3. **Calculating the Approximation**: Once the quadratic polynomials are determined for each subinterval, Simpson's rule calculates the area under each parabola. The total area under the curve is then approximated by summing up these individual areas.
4. **Weighted Average**: Simpson's rule uses a weighted average to compute the overall approximation. It gives more weight to the endpoints of the interval and less weight to the midpoint, reflecting the fact that the function varies more rapidly near the endpoints.

Mathematically, the formula for Simpson's 1/3 rule can be expressed as:

$$\int_a^b f(x)dx \approx \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

Where:

- $h = \frac{b-a}{n}$ is the width of each subinterval,
- x_i represents the endpoints of the subintervals, and
- $f(x_i)$ denotes the function value at each endpoint.

It's important to note that Simpson's rule requires the number of intervals to be even, and for certain functions, it may be necessary to use more advanced numerical integration techniques for accurate results.

1.3 Challenges of Multidimensional Integration

While methods like Simpson's rule and the trapezoidal rule are effective for single-dimensional integration, they become significantly more complex when extended to higher dimensions. Several challenges arise when attempting to apply these methods to multidimensional integration like increased computational complexity, curse of dimensionality, sampling issues, integration boundary complexity. Due to these challenges, multidimensional integration typically requires the use of more advanced numerical methods specifically designed for higher-dimensional spaces. Techniques such as Monte Carlo integration, quasi-Monte Carlo methods, sparse grid methods, and adaptive integration algorithms are commonly employed for multidimensional integration tasks. These methods offer more efficient and accurate solutions compared to extending single-dimensional integration techniques to higher dimensions. In this chapter, we will focus exclusively on the exploration of Monte Carlo Integration.

CHAPTER 2

Multidimensional Integration

2.1 Introduction to Multidimensional Integration

Multidimensional integration refers to the process of computing integrals of functions defined over multiple dimensions or variables. In simpler terms, it involves finding the volume, area, or hypervolume under a function of two or more variables within a given region in space.

In single-variable calculus, you integrate functions over one-dimensional intervals. However, many real-world problems involve functions that depend on multiple variables. For example, in physics, you might encounter equations representing the density of a fluid in three-dimensional space, or in economics, you might have functions describing the interactions of multiple variables in a complex system.

Multidimensional integration plays a crucial role in various fields such as physics, engineering, economics, and computer science. It enables the analysis of complex systems and the computation of quantities like volumes, probabilities, and expectations in higher-dimensional spaces. However, multidimensional integration can be challenging due to the increased complexity and computational cost compared to single-variable integration. Therefore, selecting appropriate integration techniques and strategies is essential for obtaining accurate and efficient solutions to multidimensional integration problems. There are several approaches to performing multidimensional integration like Monte Carlo Integration, Adaptive Methods, Sparse Grid Methods, etc. . In this chapter, we will focus exclusively on the exploration of Monte Carlo Integration.

2.2 Monte Carlo Integration

Monte Carlo integration works by randomly sampling a function to get a rough estimate of its integral. Lets say we want to integrate a one-dimensional function $f(x)$ from a to b :

$$F = \int_a^b f(x)dx$$

We can estimate this integral by averaging the function f sampled at random points uniformly distributed within the interval. Given a set of N uniform random variables Xi ∈ [a, b) with a PDF of $\frac{1}{b-a}$, the Monte Carlo estimator for calculating$F$ is

$$\langle F^N \rangle = (b-a)\frac{1}{N-1}\sum_{i=0}^{N} f(X_i| \quad)$$

The random variable X_i in the interval can be generated by transforming a standard random number evenly distributed between zero and one, denoted as ξ_i, within the interval :

$$X_i = a + \xi_i(b-a)$$

Using this construction, we can expand the estimator to:

$$\langle F^N \rangle = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(a + \xi_i(b-a)| \quad)$$

153

Since $\langle F^N \rangle$ is a function of X_i , it is itself a random variable. We use this notation to clarify that $\langle F^N \rangle$ is an approximation of F using N samples. Intuitively, the Monte Carlo estimator computes the mean value of the function $f(x)$ over the interval a to b, and then multiplies this mean by the length of the interval $(b-a)$. By moving $(b-a)$ into the summation, you can think of the estimator as randomly selecting a height by evaluating the function and then averaging the areas of rectangles calculated by multiplying this height by the length of the interval $(b-a)$.

These two interpretations are illustrated in the following figure.

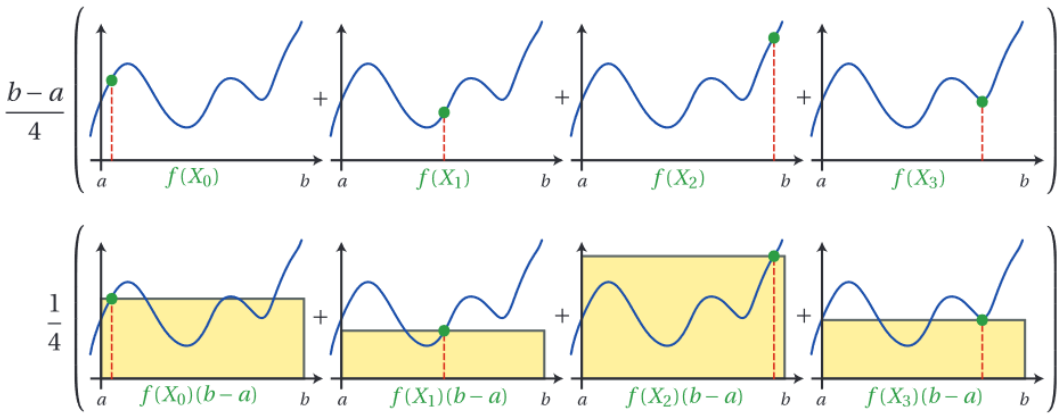


Figure 1: An illustration of the two interpretations of the fundamental Monte Carlo estimator with four samples: either by finding the mean value, or height, of the function and then multiplying by the interval length (top), or by calculating the average of several rectangular areas (bottom).[1]

Furthermore, as we increase the number of samples N, the estimator $\langle F^N \rangle$ becomes a closer and closer approximation of F .

2.2.1 Multidimensional Integration

Monte Carlo integration can be extended to utilize random variables sampled from arbitrary Probability Density Functions (PDFs) and to compute integrals over multiple dimensions such as

$$F = \int f(x) d\mu(x),$$

with the following modification:

$$\langle F^N \rangle = \frac{1}{N} \sum_{i=0}^{N} \frac{f(X_i)}{pdf(X_i)}$$

Apart from the convergence rate, another advantage of Monte Carlo integration compared to traditional numerical integration methods is how easily it can be extended to higher dimensions. Deterministic quadrature techniques require using N^dsamples for a d-dimensional integral. In contrast, Monte Carlo techniques offer the flexibility of selecting any desired number of samples. To enhance the accuracy of Monte Carlo integration, it's essential to minimize variance. For this purpose, we'll utilize the Cuba library to perform various multidimensional integrations.

CHAPTER 3

Cuba – a library for multidimensional numerical integration

In this chapter, we offer a brief introduction to the Cuba library. The Cuba library presents modernized versions of four versatile multidimensional integration algorithms: Vegas, Suave, Divonne, and Cuhre. Suave introduces an innovative algorithm, while Divonne refines an existing one with important details. Vegas and Cuhre are enhanced editions of existing algorithms, featuring minor improvements over their original versions. All four algorithms facilitate the integration of vector integrands and exhibit similar interfaces across Fortran, C/C++, and Mathematica. For a comprehensive understanding, please refer [2].

3.1 Vegas

Vegas[] is a Monte Carlo algorithm that utilizes importance sampling to reduce variance. It incrementally constructs a piecewise constant weight function on a rectangular grid through iterative sampling and grid refinements. In this updated version of Vegas, Sobol quasi-random numbers are used by default instead of pseudo-random numbers. This change has been observed to notably accelerate convergence, particularly in the initial phases of integration. The comparison between these sequences is depicted in Figure 2.

Vegas's main limitation arises from its utilization of a separable (product) weight function. Consequently, Vegas can only provide substantial enhancements when the integrand's characteristic regions align with the coordinate axes.

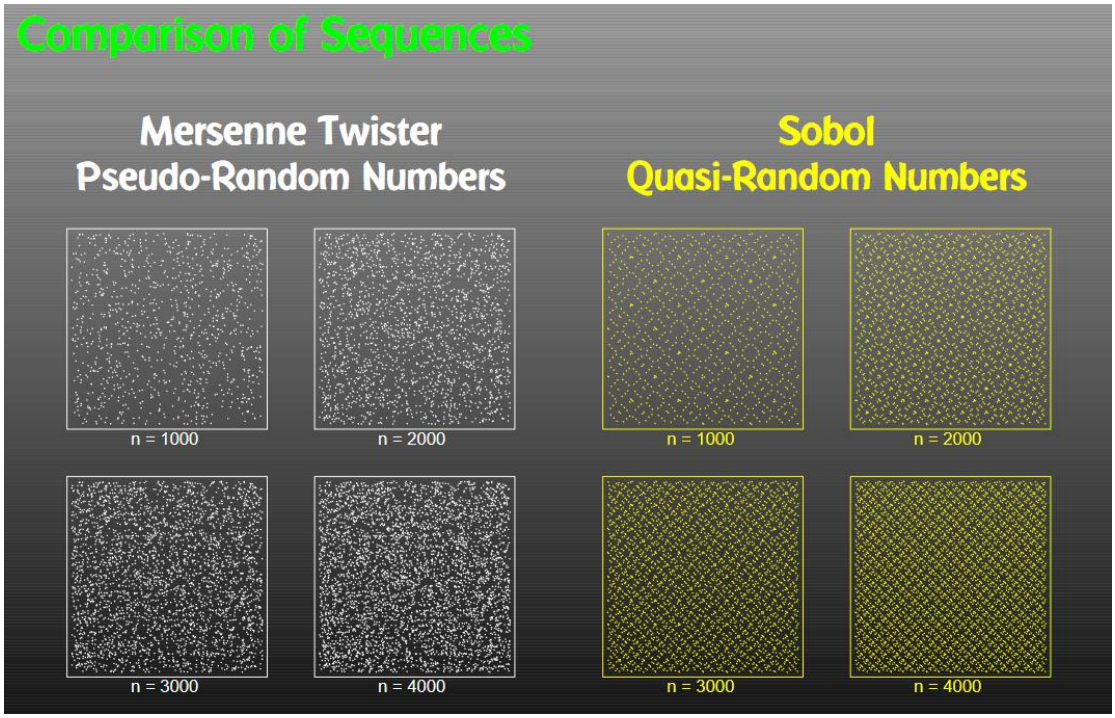


Figure 2. Comparision of Pseudo-Random Numbers and Sobol Quasi-Random Numbers[2]

3.2 Suave

Suave[], which stands for Subregion-Adaptive Vegas, employs importance sampling similar to Vegas, along with a globally adaptive subdivision approach. It continually subdivides the region with the highest error, focusing on the dimension where fluctuations in the integrand are minimized, until the desired accuracy is achieved. The allocation of new samples in each subregion is adjusted based on the fluctuations within that subregion.

Suave initially samples the integration region using a Vegas-like approach, employing importance sampling with a separable weight function. It subsequently divides the integration region into two sections. However, Suave differs in that it does not immediately recurse on these subregions. Instead, it maintains a list of all subregions and selects the one with the largest absolute error for the next cycle of sampling and subdivision. In essence, Suave utilizes global error estimation and concludes once the requested relative or absolute accuracy is achieved. The data regarding the weight function obtained in a single Vegas step isn't discarded. Instead, the grid used to compute the weight function is

stretched and reused across the subregions. Therefore, a region resulting from $m - 1$subdivisions has undergone$m$ Vegas iterations.

In Suave, there's a main loop that handles a sampling process similar to Vegas. This loop divides subregions and keeps track of the total values. At the same time, the sampling step actually collects data within these subregions and calculates results for each area.

3.3 Divonne

Divonne[] employs stratified sampling as a variance reduction technique, dividing the integration region into subregions so that each subregion has approximately equal values of a certain parameter known as the spread $\vec{s}$, defined as

$$\vec{s}(r) = \frac{1}{2} V(r) \left(\max_{x \epsilon r} \vec{f}(x) - \min_{x \epsilon r} \vec{f}(x) \right)$$

where $V(r)$ is the volume of region r. What sets Divonne apart from Suave is its method of finding the minimum and maximum values of the integrand, which involves employing numerical optimization techniques. Especially in high-dimensional scenarios, the chance of a previously sampled point being located at or near the true extremum is quite low.

On the other hand, the process of numerical minimization has its challenges. For example, when starting from the lowest sampled point, Divonne might head straight for the nearest local minimum, which might not be the absolute minimum. Divonne is more complicated than Suave and Vegas, but it's also much faster for various types of integrals.

3.4 Cuhre

Cuhre[] is a deterministic algorithm that employs one of several cubature rules with polynomial degrees in a globally adaptive subdivision approach. The subdivision process resembles Suave's and operates as follows:

When the total estimated error surpasses the specified bounds:

1. Select the region with the highest estimated error.
2. Bisect this region along the axis with the largest fourth difference.
3. Apply the cubature rule to the resulting two subregions.
4. Merge the subregions back into the list of regions and update the totals.

The current implementation offers only minor enhancements, such as an interface that aligns with other Cuba routines and a slightly simplified invocation. For instance, there's no need to allocate workspace.

In moderate dimensions, Cuhre proves to be quite competitive, especially when the integrand can be well approximated by polynomials. However, as the dimensionality increases, the number of points sampled by the cubature rules significantly rises, leading to a decline in usefulness. The table below lists the actual number of points utilized per invocation of the basic integration rule for lower dimensions.

number of dimensions	4	5	6	7	8	9	10	11	12
points in degree-7 rule	65	103	161	255	417	711	1265	2335	4433
points in degree-9 rule	153	273	453	717	1105	1689	2605	4117	6745

3.5 Usage in C/C++

Being written in C, the algorithms can of course be used in C/C++ directly. The declarations are as follows:

```
typedef int (*integrand_t)(const int *ndim, const double x[],
  const int *ncomp, double f[], void *userdata);

typedef void (*peakfinder_t)(const int *ndim, const double b[],
  int *n, double x[], void *userdata);
```

```
void Suave(const int ndim, const int ncomp,
  integrand_t integrand, void *userdata, const int nvec,
  const double epsrel, const double epsabs,
  const int flags, const int seed,
  const int mineval, const int maxeval,
  const int nnew, const int nmin,
  const double flatness, const char *statefile, void *spin,
  int *nregions, int *neval, int *fail,
  double integral[], double error[], double prob[])

void Vegas(const int ndim, const int ncomp,
  integrand_t integrand, void *userdata, const int nvec,
  const double epsrel, const double epsabs,
  const int flags, const int seed,
  const int mineval, const int maxeval,
  const int nstart, const int nincrease, const int nbatch,
  const int gridno, const char *statefile, void *spin,
  int *neval, int *fail,
  double integral[], double error[], double prob[])


void Divonne(const int ndim, const int ncomp,
  integrand_t integrand, void *userdata, const int nvec,
  const double epsrel, const double epsabs,
  const int flags, const int seed,
  const int mineval, const int maxeval,
  const int key1, const int key2, const int key3,
  const int maxpass, const double border,
  const double maxchisq, const double mindeviation,
  const int ngiven, const int ldxgiven, double xgiven[],
  const int nextra, peakfinder_t peakfinder,
  const char *statefile, void *spin,
  int *nregions, int *neval, int *fail,
  double integral[], double error[], double prob[])

void Cuhre(const int ndim, const int ncomp,
  integrand_t integrand, void *userdata, const int nvec,
  const double epsrel, const double epsabs,
  const int flags,
  const int mineval, const int maxeval,
  const int key, const char *statefile, void *spin,
  int *nregions, int *neval, int *fail,
  double integral[], double error[], double prob[])
```

These prototypes are contained in cuba.h which should (in C) or must (in C++) be included when using the Cuba routines.

3.5.1 Common Options

- Precision Goal→ 3, the number of digits of relative accuracy to seek
- Accuracy Goal → 12, the number of digits of absolute accuracy to seek
- MinPoints -> 0, the minimum number of times the integrand needs to be evaluated.
- NStart -> 50000, the maximum number of integrand evaluations allowed
- Verbose -> the level of detail to print regarding intermediate results, ranging from 0 to 3.

3.5.2 Vegas-specific Options

- NStart -> 1000: Indicates the number of integrand evaluations per iteration to begin with.
- NIncrease -> 500: Represents the increase in the number of integrand evaluations per iteration.
- GridNo -> 0: Refers to the specific slot in the internal grid table.
- StateFile -> " " Denotes the file name used for storing the internal state.

3.5.1 Suave-specific Options

- NNew -> 1000: Specifies the number of new integrand evaluations in each subdivision.
- NMin -> 2: Indicates the minimum number of samples required from a previous pass to be considered in the compound integral value of a subregion.

- Flatness -> 50: Refers to the type of norm used to calculate the fluctuation of a sample.

3.5.3 Divonne-specific Options

- Key1 -> 47: An integer value that controls sampling during the partitioning phase. If Key1 is greater than 0, it indicates the use of a Korobov quasi-random sample. If Key1 is less than 0, it indicates the use of a "standard" sample (a Sobol quasi-random sample if PseudoRandom -> False, otherwise a pseudo-random sample).
- Key2 -> 47: An integer value that controls sampling during the partitioning phase.
- Key3 -> 1: An integer value that determines the strategy for the refinement phase.
- Border -> 0: Refers to the width of the border surrounding the integration region.
- MaxChisq -> 10: Represents the maximum $\chi 2$ value that a single subregion is permitted to have during the final integration phase.
- MinDeviation -> 0.25: Denotes a threshold, expressed as a fraction of the requested error of the entire integral, which determines whether further examination of a region that failed the $\chi 2$ test is worthwhile.

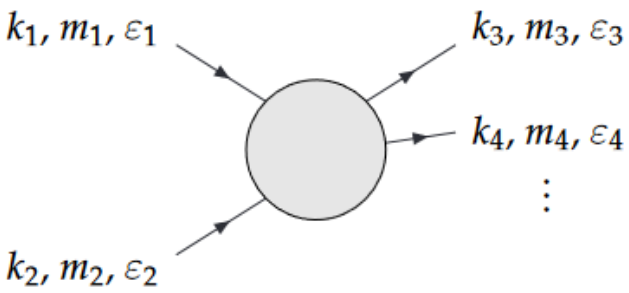
3.5.1 Cuhre-specific options

- Key -> 0: Selects the basic integration rule.
  If Key = 7, 9, 11, 13, it chooses the cubature rule of degree Key.
  Note that the degree-11 rule is only available in 3 dimensions, and the degree-13 rule is only available in 2 dimensions.

CHAPTER 4

RESULTS AND ANALYSIS

In this chapter, we present the computational results obtained from integrating the differential cross section area for kinematics of $1 \rightarrow 2$, $2 \rightarrow 2$ and $2 \rightarrow 3$. We begin by writing the differential cross section area for these processes, followed by comparison of results from all the four algorithms. In this chapter, the momenta, masses, and polarization vectors are numbered sequentially, as depicted in the following figure.



4.1 Kinematics for 2→2 Process

The $2 \rightarrow 2$ phase space can be parameterized using just one variable, typically denoted as the angle θ between the beam axis and the first outgoing particle. The expressions for the momenta and polarization

vectors are explicitly represented in terms of θ and $\sqrt{s}$, where $\sqrt{s}$ denotes the center-of-mass energy.

$$
\begin{aligned}
k_1 &= (E_1, 0, 0, p_{\text{in}})\,, & k_3 &= (E_3, p_{\text{out}}\sin\theta, 0, p_{\text{out}}\cos\theta)\,,\\
\varepsilon_1(0) &= (p_{\text{in}}, 0, 0, E_1)/m_1\,, & \varepsilon_3^*(0) &= (p_{\text{out}}, E_3\sin\theta, 0, E_3\cos\theta)/m_3\,,\\
\varepsilon_1(\pm) &= (0, 1, \pm\mathrm{i}, 0)/\sqrt{2}\,, & \varepsilon_3^*(\pm) &= (0, \cos\theta, \mp\mathrm{i}, -\sin\theta)/\sqrt{2}\,,
\end{aligned}
$$

$$
\begin{aligned}
k_2 &= (E_2, 0, 0, -p_{\text{in}})\,, & k_4 &= (E_4, -p_{\text{out}}\sin\theta, 0, -p_{\text{out}}\cos\theta)\,,\\
\varepsilon_2(0) &= (p_{\text{in}}, 0, 0, -E_2)/m_2\,, & \varepsilon_4^*(0) &= (p_{\text{out}}, -E_4\sin\theta, 0, -E_4\cos\theta)/m_4\,,\\
\varepsilon_2(\pm) &= (0, 1, \mp\mathrm{i}, 0)/\sqrt{2}\,, & \varepsilon_4^*(\pm) &= (0, \cos\theta, \pm\mathrm{i}, -\sin\theta)/\sqrt{2}\,,
\end{aligned}
$$

with

$$
\begin{aligned}
E_1^2 &= p_{\text{in}}^2 + m_1^2\,, & E_3^2 &= p_{\text{out}}^2 + m_3^2\,,\\
E_2^2 &= p_{\text{in}}^2 + m_2^2\,, & E_4^2 &= p_{\text{out}}^2 + m_4^2\,,\\
p_{\text{in}}^2 &= \frac{(s + m_2^2 - m_1^2)^2}{4s} - m_2^2\,, & p_{\text{out}}^2 &= \frac{(s + m_4^2 - m_3^2)^2}{4s} - m_4^2\,.
\end{aligned}
$$

The phase-space element is given by

$$
\mathrm{d}\Gamma_2 = \frac{\mathrm{d}^3k_3}{(2\pi)^3 2k_3^0}\,\frac{\mathrm{d}^3k_4}{(2\pi)^3 2k_4^0}\,(2\pi)^4\delta^{(4)}(k_1 + k_2 - k_3 - k_4)
$$

from which the following formula for the differential cross-section is obtained:

$$
\mathrm{d}\sigma = \frac{|\mathcal{M}|^2\,\mathrm{d}\Gamma_2}{\Phi} = \frac{1}{4(2\pi)^2}\,\frac{1}{\Phi}\,\frac{p_{\text{out}}}{\sqrt{s}}\,|\mathcal{M}|^2\,\mathrm{d}\Omega_3
$$

with the flux factor Φ given by

$$
\Phi = 2\sqrt{\lambda(s, m_1^2, m_2^2)} = 4p_{\text{in}}\sqrt{s}\,.
$$

We write $d\Omega_3$ as

$$
d\Omega_3 = sin\,(\theta)d\theta d\phi
$$

Now for the integration purpose we transform the limits of θ and ϕ (as discussed in 2.2) to the interval [0,1], thus resulting in factors of π and 2π being multiplied in $d\sigma$. The function $sin\,(\theta)$ will be changed to

$$sin\theta d\theta = \pi sin(\pi x)dx$$

where x is a random number that is generated in the interval [0,1].

For the purpose of simplification, we assume the values of $|M|^2, \Phi$ to be unity. This will be followed in all the processes that will be discussed further.

4.2 Kinematics for 1→2 Process

When considering scenarios with the same number of outgoing particles, the transition from the 1 → 2 case to the 2 → 2 case is inherently similar. On the surface, particles 3 and 4 in the 2 → 2 case correspond to particles 2 and 3 in the 1 → 2 case, and the invariant mass $\sqrt{s}$ is replaced by the mass of the decaying particle.

However, unlike in scattering scenarios, there isn't a designated beam axis that establishes a preferred direction. Nevertheless, the matrix element of a decaying non-scalar particle displays a non-trivial reliance on a decay angle θ,which must then be measured relative to an arbitrarily chosen direction. In the program, this direction is set as the positive z-axis, resulting in the following momentum representations:

$$\begin{aligned}
k_1 &= (m_1, 0, 0, 0)\,, & k_2 &= (E_2, p_{\text{out}} \sin\theta, 0, p_{\text{out}} \cos\theta)\,,\\
\varepsilon_1(0) &= (0, 0, 0, 1)\,, & \varepsilon_2^*(0) &= (p_{\text{out}}, E_2 \sin\theta, 0, E_2 \cos\theta)/m_2\,,\\
\varepsilon_1(\pm) &= (0, 1, \pm\mathrm{i}, 0)/\sqrt{2}\,, & \varepsilon_2^*(\pm) &= (0, \cos\theta, \mp\mathrm{i}, -\sin\theta)/\sqrt{2}\,,\\
& & k_3 &= (E_3, -p_{\text{out}} \sin\theta, 0, -p_{\text{out}} \cos\theta)\,,\\
& & \varepsilon_3^*(0) &= (p_{\text{out}}, -E_3 \sin\theta, 0, -E_3 \cos\theta)/m_3\,,\\
& & \varepsilon_3^*(\pm) &= (0, \cos\theta, \pm\mathrm{i}, -\sin\theta)/\sqrt{2}\,.
\end{aligned}$$

where

$$p_{\text{out}} = \frac{(m_1^2 + m_3^2 - m_2^2)^2}{4m_1^2} - m_3^2\,.$$

The differential cross-section is

$$\mathrm{d}\sigma = \frac{1}{4(2\pi)^2}\frac{1}{\Phi}\frac{p_{\mathrm{out}}}{\sqrt{s}}|\mathcal{M}|^2\,\mathrm{d}\Omega_2$$

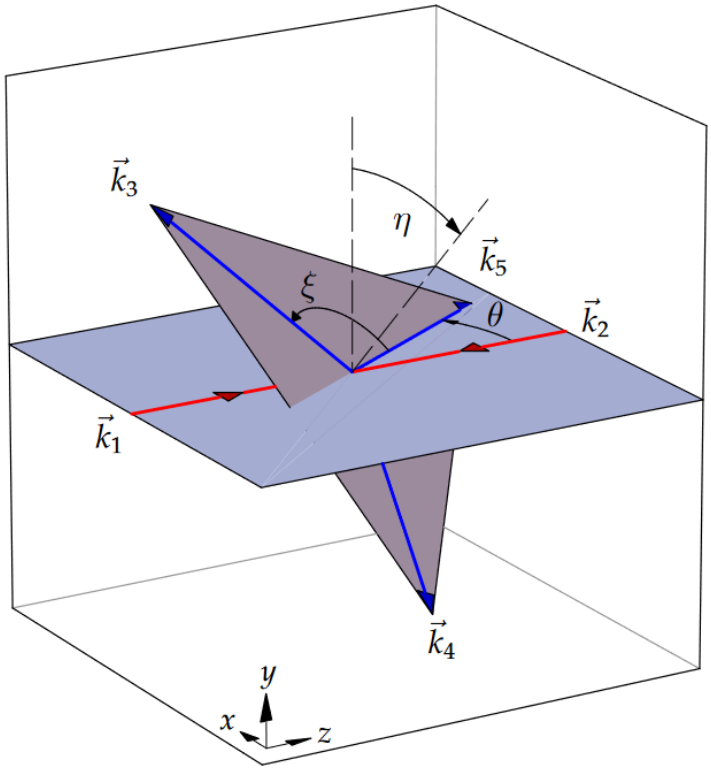
The integration over the scattering angle is done precisely as in the $2 \rightarrow 2$ case.

4.3 Kinematics for 2→3 Process

This process employs a parameterization of the $2 \rightarrow 3$ phase space using the parameters

$$\theta, \eta, k_5^0 \wedge k_3^0$$

where k_3^0 and k_5^0 represent the energies of the third and fifth particles, respectively, while θ and η are two of the angles depicted in the following figure.



This set of parameters is especially useful in the context of hard bremsstrahlung. Specifically, if k_5 is selected as the momentum of the bremsstrahlung photon, it becomes easy to set boundaries on the phase-space variables to prevent common divergences: setting a minimum value for k_5^0 helps avoid soft-photonic infrared divergences, while setting a minimum value for θ helps avoid collinear divergences. The specific representations of the outgoing vectors, as implemented in this program, are:

$$k_5 = (k_5^0, |\vec{k}_5|\vec{e}_5)\,, \quad k_3 = (k_3^0, |\vec{k}_3|\vec{e}_3)\,, \quad \text{with} \quad |\vec{k}_i| = \sqrt{(k_i^0)^2 - m_i^2}\,.$$

From elementary geometry the unit vectors are found to be

$$\vec{e}_5 = \begin{pmatrix} \sin\theta \\ 0 \\ \cos\theta \end{pmatrix},$$

$$\vec{e}_3 = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\eta\sin\xi \\ \sin\eta\sin\xi \\ \cos\xi \end{pmatrix} = \begin{pmatrix} \cos\theta\cos\eta\sin\xi + \sin\theta\cos\xi \\ \sin\eta\sin\xi \\ \cos\theta\cos\xi - \sin\theta\cos\eta\sin\xi \end{pmatrix}.$$

In the center-of-mass (CMS) frame, $k_4$is determined by four-momentum conservation. This can be expressed as:

$$k_3^0 + k_4^0 + k_5^0 = \sqrt{s} \quad \text{and} \quad \vec{k}_3 + \vec{k}_4 + \vec{k}_5 = 0\,.$$

The three-particle phase-space element is

$$\mathrm{d}\Gamma_3 = \frac{\mathrm{d}^3k_3}{(2\pi)^3 2k_3^0}\,\frac{\mathrm{d}^3k_4}{(2\pi)^3 2k_4^0}\,\frac{\mathrm{d}^3k_5}{(2\pi)^3 2k_5^0}\,(2\pi)^4\delta^{(4)}(k_1 + k_2 - k_3 - k_4 - k_5)\,.$$

Rearranging the volume elements and utilizing the $\delta^{(4)}$function to eliminate the $k_4$integration, $d\Gamma_3$is simplified to:

$$\mathrm{d}\Gamma_3 = \frac{|\vec{k}_3||\vec{k}_5|}{4(2\pi)^5}\mathrm{d}k_3^0\,\mathrm{d}\Omega_3\,\mathrm{d}k_5^0\,\mathrm{d}\Omega_5\,\delta(k_4^2 - m_4^2)\,.$$

We allow the remaining δfunction to eliminate the $cos\xi$integration in $d\Omega_3 = dcos\xi d\eta$. After further simplification(refer [3]), the formula for the differential cross-section is

$$d\sigma = \frac{|\mathcal{M}|^2 \, d\Gamma_3}{\Phi} = \frac{1}{8(2\pi)^4} \frac{1}{\Phi} |\mathcal{M}|^2 \, dk_5^0 \, dk_3^0 \, d\cos\theta \, d\eta$$

with the flux factor given by

$$\Phi = 2\sqrt{\lambda(s, m_1^2, m_2^2)} = 4|\vec{k}_1|\sqrt{s}\,.$$

Regarding the integration limits, the angles θ and η, which determine the plane formed by the outgoing momenta relative to the x-z plane, can be integrated separately. The integration of the particle energies is constrained by the condition that$|cos\xi| \leq 1$, leading to the bounds:

$$\sigma = \int_{m_5}^{(k_5^0)^{\max}} dk_5^0 \int_{(k_3^0)^{\min}}^{(k_3^0)^{\max}} dk_3^0 \int_{-1}^{1} d\cos\theta \int_0^{2\pi} d\eta \, \frac{d\sigma}{dk_5^0 \, dk_3^0 \, d\cos\theta \, d\eta}$$

with

$$(k_5^0)^{\max} = \frac{\sqrt{s}}{2} - \frac{(m_3 + m_4)^2 - m_5^2}{2\sqrt{s}}$$

and

$$(k_3^0)^{\max,\min} = \frac{1}{2\tau}\left[\sigma(\tau + m_+ m_-) \pm |\vec{k}_5|\sqrt{(\tau - m_+^2)(\tau - m_-^2)}\right],$$
$$\sigma = \sqrt{s} - k_5^0\,, \qquad \tau = \sigma^2 - |\vec{k}_5|^2\,, \qquad m_\pm = m_3 \pm m_4\,.$$

Upon transforming the integral limits we get factors $(k_5^0)^{max} - m_5$, $(k_3^0)^{max} - (k_3^0)^{m\epsilon}$, 2 and 2π corresponding to $dk_5^0, dk_3^0, dcos\theta \; and \; d\eta$ respectively.

4.5 Results

The Cuba parameters used are:

#define NDIM 2

#define NCOMP 1

#define USERDATA NULL

#define NVEC 1

#define EPSREL 1e-3

#define EPSABS 1e-7

```
#define VERBOSE 1
#define LAST 4
#define SEED 0
#define MINEVAL 0
#define MAXEVAL 50000

/*----------------------For Vegas-----------------------*/

#define NSTART 1000
#define NINCREASE 500
#define NBATCH 1000
#define GRIDNO 0
#define STATEFILE NULL
#define SPIN NULL
#define RETAINSTATEFILE false

/*----------------------For Suave-----------------------*/

#define NNEW 1000
#define NMIN 2
#define FLATNESS 50

/*----------------------For Divonne-----------------------*/

#define KEY1 47
#define KEY2 1
#define KEY3 1
```

#define MAXPASS 5

#define BORDER 1e-6

#define MAXCHISQ 10

#define MINDEVIATION .25

#define NGIVEN 0

#define LDXGIVEN NDIM

#define NEXTRA 0

/*----------For Cuhre----------*/

#define KEY 0

The initial assumption of arbitraty values of mass and momentum for 1->2 process are m1 = 3, m2 = 2, m3 = 1 and pin = 9. The resulting cross-section area is:

1. Vegas:

   Iteration 1:
   0.67175118 +- 0.0102402 chisq 0.00000000
   Iteration 2:
   0.67106017 +- 0.00359279 chisq 0.00519278
   Iteration 3:
   0.67108690 +- 0.00153038 chisq 0.00526042
   Iteration 4:
   0.67104831 +- 0.000851905 chisq 0.00618208
   Iteration 5:
   0.67108247 +- 0.000594597 chisq 0.00931819
   Vegas Result:
   Number of Evaluations: 10000
   Number of Failed Evaluations: 0
   Final Result: 0.67108247 +- 0.00059460 chisq 0.00931819

2. Suave:

Iteration 1:
0.67175118 +- 0.0102402 chisq 0.00000000 (0 df)
Iteration 2:
0.67235076 +- 0.00438898 chisq 0.00296133 (2 df)
Iteration 3:
0.67194305 +- 0.00330312 chisq 0.08049297 (5 df)
Iteration 4:
0.67165944 +- 0.00157569 chisq 0.09493483 (8 df)
Iteration 5:
0.67172630 +- 0.00136095 chisq 2.67303177 (12 df)
Iteration 6:
0.67166431 +- 0.00113861 chisq 7.90177871 (16 df)
Iteration 7:
0.67100148 +- 0.00100225 chisq 13.79413093 (20 df)
Iteration 8:
0.67067990 +- 0.000779457 chisq 16.46208864 (24 df)
Iteration 9:
0.67074710 +- 0.000642083 chisq 16.96377576 (29 df)
Suave Result:
Number of Regions: 9
Number of Evaluations: 9000
Number of Failed Evaluations: 0
Final Result: 0.83569067 +- 0.84358194 chisq 8.48188788 (29 df)

3. Divonne:

Iteration 1:
0.67105705 +- 0.00149188
Iteration 2:
0.67105796 +- 0.00114246
Iteration 3:
0.67106499 +- 0.000976659
Iteration 4:
0.67105922 +- 0.000783266
Iteration 5:
0.67105385 +- 0.00052586
Iteration 6:

0.67105164 +- 0.000470653
Iteration 7:
0.67105325 +- 0.000450583
Iteration 8:
0.67105176 +- 0.000430127
Iteration 9:
0.67105028 +- 0.00040866
Iteration 10:
0.67104883 +- 0.000386241
Iteration 11:
0.67104742 +- 0.000362693
Divonne Result:
Number of Regions: 18
Samples per Regions: 97
Number of Evaluations: 2352
Final Result: 0.67105284 +- 0.00000003 p = 0.0000

4. Cuhre:

Iteration 1:
0.67105616 +- 7.49007e-11 chisq 0.00000000 (0 df)
Iteration 2:
0.67105616 +- 9.12614e-14 chisq 0.00097656 (1 df)
Cuhre Result:
Number of Regions: 2
Number of Evaluations: 195
Number of Failed Evaluations: 0
Final Result: 0.67105616 +- 0.00000000 chisq 0.00097656 p = 0.0249

The initial assumption of arbitraty values of mass and momentum for 2->2 process are m1 = 3, m2 = 2, m3 = 1, m4 = 4 and pin = 9. The resulting cross-section area is:

1. Vegas:

Iteration 1:
0.03562499 +- 0.000543068 chisq 0.00000000
Iteration 2:
0.03558834 +- 0.000190537 chisq 0.00519278
Iteration 3:

0.03558976 +- 8.11605e-05 chisq 0.00526042
Iteration 4:
0.03558771 +- 4.51791e-05 chisq 0.00618208
Iteration 5:
0.03558952 +- 3.15333e-05 chisq 0.00931819
Vegas Result:
Number of Evaluations: 10000
Number of Failed Evaluations: 0
Final Result: 0.03558952 +- 0.00003153 chisq 0.00931819

2. Suave:

Iteration 1:
0.03562499 +- 0.000543068 chisq 0.00000000 (0 df)
Iteration 2:
0.03565678 +- 0.000232761 chisq 0.00296133 (2 df)
Iteration 3:
0.03563516 +- 0.000175174 chisq 0.08049297 (5 df)
Iteration 4:
0.03562012 +- 8.35635e-05 chisq 0.09493483 (8 df)
Iteration 5:
0.03562367 +- 7.21752e-05 chisq 2.67303177 (12 df)
Iteration 6:
0.03562038 +- 6.03841e-05 chisq 7.90177871 (16 df)
Iteration 7:
0.03558523 +- 5.31521e-05 chisq 13.79413093 (20 df)
Iteration 8:
0.03556817 +- 4.13369e-05 chisq 16.46208864 (24 df)
Iteration 9:
0.03557174 +- 3.40516e-05 chisq 16.96377576 (29 df)
Suave Result:
Number of Regions: 9
Number of Evaluations: 9000
Number of Failed Evaluations: 0
Final Result: 0.83569067 +- 0.84358194 chisq 8.48188788 (29 df)

3. Divonne:

Iteration 1:

0.03558794 +- 5.6427e-05
Iteration 2:
0.03558768 +- 5.03606e-05
Iteration 3:
0.03558744 +- 4.35049e-05
Iteration 4:
0.03558792 +- 3.75187e-05
Iteration 5:
0.03558774 +- 3.08138e-05
Iteration 6:
0.03558786 +- 2.82815e-05
Iteration 7:
0.03558797 +- 2.55018e-05
Divonne Result:
Number of Regions: 14
Samples per Regions: 97
Number of Evaluations: 1891
Final Result: 0.03558807 +- 0.00000000 p = 0.0000

4. Cuhre:

Iteration 1:
0.03558813 +- 3.97221e-12 chisq 0.00000000 (0 df)
Iteration 2:
0.03558813 +- 4.81928e-15 chisq 0.00000000 (1 df)
Cuhre Result:
Number of Regions: 2
Number of Evaluations: 195
Number of Failed Evaluations: 0
Final Result: 0.03558813 +- 0.00000000 chisq 0.00000000 p = 0.0000

The initial assumption of arbitraty values of mass and momentum for 2->3 process are m3 = 2, m4 = 4, m5 = 5, k3 = 2, k4 =3 and k5 = 4. The resulting cross-section area is:

1. Vegas:

Iteration 1:
0.18879046 +- 0.00287956 chisq 0.00000000
Iteration 2:

0.18871250 +- 0.00102163 chisq 0.00083859
Iteration 3:
0.18857832 +- 0.000435749 chisq 0.02192540
Iteration 4:
0.18856484 +- 0.000250971 chisq 0.02335633
Iteration 5:
0.18854723 +- 0.000175504 chisq 0.03300118
Vegas Result:
Number of Evaluations: 10000
Number of Failed Evaluations: 0
Final Result: 0.18854723 +- 0.00017550 chisq 0.03300118

2. Suave:

Iteration 1:
0.18879046 +- 0.00287956 chisq 0.00000000 (0 df)
Iteration 2:
0.18854234 +- 0.00126507 chisq 0.05199129 (2 df)
Iteration 3:
0.18862076 +- 0.000965452 chisq 0.09649342 (5 df)
Iteration 4:
0.18852181 +- 0.000525154 chisq 0.15054081 (8 df)
Iteration 5:
0.18849587 +- 0.000477691 chisq 0.55362180 (12 df)
Iteration 6:
0.18856946 +- 0.00043161 chisq 3.32619670 (16 df)
Iteration 7:
0.18840871 +- 0.000399912 chisq 3.89365870 (20 df)
Iteration 8:
0.18834791 +- 0.000355583 chisq 4.54616865 (24 df)
Iteration 9:
0.18844302 +- 0.000335318 chisq 7.11546132 (29 df)
Iteration 10:
0.18857062 +- 0.000319167 chisq 11.04268851 (34 df)
Iteration 11:
0.18853951 +- 0.000305048 chisq 15.98967809 (39 df)
Iteration 12:
0.18855086 +- 0.00028799 chisq 17.91851851 (44 df)
Iteration 13:
0.18857087 +- 0.00027242 chisq 19.44248154 (49 df)

Iteration 14:
0.18861805 +- 0.000257164 chisq 20.86210338 (54 df)
Iteration 15:
0.18861874 +- 0.000240037 chisq 23.88922196 (59 df)
Iteration 16:
0.18859308 +- 0.000226414 chisq 30.60045838 (64 df)
Iteration 17:
0.18859758 +- 0.00021469 chisq 30.92395882 (70 df)
Iteration 18:
0.18854805 +- 0.000207793 chisq 39.06719538 (76 df)
Iteration 19:
0.18848642 +- 0.000203928 chisq 41.68827230 (82 df)
Iteration 20:
0.18849166 +- 0.000196855 chisq 47.70761603 (88 df)
Iteration 21:
0.18852864 +- 0.000193059 chisq 68.73752297 (94 df)
Iteration 22:
0.18849641 +- 0.000189805 chisq 74.13159165 (100 df)
Iteration 23:
0.18847845 +- 0.000185702 chisq 92.30855643 (106 df)
Suave Result:
Number of Regions: 23
Number of Evaluations: 23000
Number of Failed Evaluations: 0
Final Result: 0.83617078 +- 46.15427822 chisq 0.85932829 (106 df)

3. Divonne:

Iteration 1:
0.18850664 +- 0.00029889
Iteration 2:
0.18850528 +- 0.000266756
Iteration 3:
0.18850398 +- 0.000230442
Iteration 4:
0.18850654 +- 0.000198734
Iteration 5:
0.18850559 +- 0.000163218
Iteration 6:

0.18850619 +- 0.000149805
Iteration 7:
0.18850680 +- 0.000135081
Divonne Result:
Number of Regions: 14
Samples per Regions: 97
Number of Evaluations: 2303
Final Result: 0.18850731 +- 0.00000000 p = 0.0000

4. Cuhre:

Iteration 1:
0.18850763 +- 1.03521e-07 chisq 0.00000000 (0 df)
Iteration 2:
0.18850763 +- 3.0085e-09 chisq 0.00068441 (1 df)
Cuhre Result:
Number of Regions: 2
Number of Evaluations: 819
Number of Failed Evaluations: 0
Final Result: 0.18850763 +- 0.00000000 chisq 0.00068441 p = 0.0209

4.6 Conclusion

In conclusion, this paper has provided an in-depth exploration of multidimensional integration techniques facilitated by the Cuba library. Through our investigation, we have highlighted the versatility and efficiency of four general-purpose integration algorithms: Vegas, Suave, Divonne, and Cuhre. These algorithms offer robust solutions for computing integrals in various dimensions, accommodating both vector integrands and scalar functions.

By leveraging features such as importance sampling, stratified sampling, and adaptive subdivision, the Cuba library enables researchers to tackle challenging integration problems effectively.

Furthermore, we have demonstrated the ease of use and flexibility of the Cuba library, which provides consistent interfaces across different programming languages. This ensures accessibility and ease of integration into existing computational workflows.

Overall, the Cuba library stands as a valuable tool for researchers and practitioners in fields ranging from theoretical physics to computational biology. Its comprehensive suite of algorithms, coupled with its user-friendly interface, makes it a reliable choice for tackling complex multidimensional integration tasks with precision and efficiency.

References

[1]https://cs.dartmouth.edu/~wjarosz/publications/dissertation/appendixA.pdf

[2] Cuba – a library for multidimensional numerical integration, T. Hahn Max-Planck-Institut f¨ur Physik, F¨ohringer Ring 6, D–80805 Munich, Germany, January 26, 2005

[3] FormCalc 4 User’s Guide, Thomas Hahn, March 2, 2005