Google Summer of Code 2021

CODING PROJECT PROPOSAL

Monte Carlo integration

Name: Arnab Suklabaidya

Affiliation: National Institute of Technology Agartala

Program: B.tech UG student first participation in GSoC

Mentors: Vissarion Fisikopoulos, Pedro Zuidberg Dos Martires

Email: arnabsuklabaidya@gmail.com

Github: https://github.com/noobscript69

Address: Agartala, Tripura, India. 799001

Phone: +91 7085830711

Contents

1 Synopsis 2

2 The Project 3

2.1 Understanding the code structure and design of VolEsti. 4

2.2 Extent VolEsti's functionality to handle multi-dimensional integrals 7

2.3 Implementation 14

2.4 Methodology 23

3 Benefits to the Community 24

4 Deliverables 24

4.1 Bonding period (6th May - 26th May) 25

4.2 Coding period (27th May - 26th August) 26

4.3 Schedule Conflicts 27

5 Related work 28

6 Tests 28

7 Biographical Information 31

7.1 Academic 31

7.2 Programming 31

7.3 Personal motivation 32

Synopsis

Sampling from high dimensional distributions and volume approximation of convex bodies are fundamental operations that appear in optimization, finance, engineering and machine learning. The package **VolEsti** , a C++ package with an R interface that provides efficient, scalable algorithms for volume estimation, uniform and Gaussian sampling from convex polytopes. VolEsti scales to hundreds of dimensions, handles efficiently three different types of polyhedra and provides non existing sampling routines to R. Thus it could be an essential tool for a quite large number of scientific applications in convex analysis, economics, biology or statistics, that need fast volume approximation or sampling in high dimensions. However, the possibility to scale from a few hundred to a few thousand dimensions was considered as a very far-reaching goal for many years.

The goal of this project is to extent VolEsti's functionality to handle multi-dimensional integrals. I exploit some very recent theoretical results that guarantee fast convergence and numerical stability in order to propose an efficient implementation of the **Monte Carlo Integration**. The proposed implementations will be a decisive contribution to other scientific fields as computational geometry, finance and optimization, research(see section 3). For example researchers in artificial intelligence have used a home-brew simple versions of the above algorithms for weighted model integration.

In section 4 we give a week time schedule for the coding project. We hope this project will be a decisive contribution towards the first complete and efficient tool for sampling, volume estimation and geometrical statistics in general and thus, help educational programs, research or even serve as a building block towards an international, interdisciplinary community in geometrical statistics.

The Project

The goal of this project is to extent VolEsti's functionality to handle multi-dimensional integrals. Under this project we are targeting to provide efficient implementations for various geometric random walk algorithms to sample points from convex polytopes and volume computation in high dimensions. In particular I have to implement a more advanced Monte Carlo Integration . It has been proved that the hit-and-run random walk is rapidly mixing for an arbitrary logconcave distribution starting from any point in the support. From this result, we derive asymptotically faster algorithms in the general oracle model for sampling, rounding, integration, volume estimation and maximization of logconcave functions. Also I have to Check-compare or implement well known MC integration algorithms e.g. MISER, VEGAS . The MISER algorithm is based on recursive stratified sampling. This technique aims to reduce the overall integration error by concentrating integration points in the regions of highest variance. The VEGAS algorithm approximates the exact distribution by making a number of passes over the integration region which creates the histogram of the function *f*. Each histogram is used to define a sampling distribution for the next pass.

We have to Examine possible integrations with stan a state-of-the-art platform for statistical modelling and high-performance statistical computation. Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. Hence the implementation of these methods will result the most complete package for sampling and volume estimation in a few thousand dimensions and also extent VolEsti's functionality to handle multi-dimensional integrals .

The main part of the implementation will be in C++ given as a natural extension of the current software of VolEsti and will be easily accessible through the Rcpp wrapping. I am going to use some of the geometrical concepts that are already implemented, e.g., points and convex polytopes. Packages like cubature, SI, Stan will be used in this coding project.

Proposal structure: The rest of the section presents the structure of the package, an overview of the methods that I will implemented and all the technical details and the methodology of the implementation. The section 3 presents the benefits for the scientific and business communities. The section 4 presents a detailed time schedule starting from the bonding period until the end of GSoC 2019. The section 5 presents some related work I have done for the proposed project. The section 6 presents the implemented solutions of the tree tests that are requested from the mentors. The section 7 contains biographical information and personal motivation.

2.1 Understanding the code structure and design of VolEsti.

The R package VolEsti combines the efficiency of a C++ implementation and the friendly interface of R. The package consists of more than 3 000 lines of C++ code. To call C++ code there is one Rcpp function for each procedure (i.e., sampling, volume estimation etc.) and it is exported as an R function. To export C++ classes that represent convex polytopes we use Rcpp modules to avoid useless R code and to maintain the code easily.

We use continuous integration practice to maintain and test the C++ part of VolEsti . Additionally, there is a test suite for the functions of the R package. The package provides 8 functions, 4 exposed classes for convex polytopes and 8 polytope generators. Table 1 presents the most important functions of the package for our project. VolEsti provides 4 different representations for a convex polytope with a corresponding exposed C++ class for each representation. The HMC walk family and the implemented random walks in can be applied only for the H-representation, i.e., when the polytope is given by a set of linear inequalities

The package provides the following three random walk methods for sampling from the input polytope: a) Coordinate Directions Hit-and-Run (CDHR), b) Random Directions Hit-ad-Run (RDHR), and c) Ball walk. Moreover the user could choose between uniform and multidimensional spherical Gaussian target distribution. The CDHR it is shown to be the most efficient random walk in practice for many applications among many random walk algorithms. However, it does not scale beyond a few hundred dimensions, regardless the application. The plots in Figure

1 demonstrate two samples, one for each target distribution that VolEsti provides. Both methods (SeqOf Balls and Cooling Gaussian) for volume estimation are based on sampling using a random walk while CDHR is the default choice. Seq Of Balls uses uniform sampling and Cooling Gaussian uses Gaussian, while the first is statistically more accurate and Cooling Gaussian is faster.

Name	Input	Description	Parameterizations
sample_points()	A convex Polytope	Samples from the input polytope using random walks or perfect uniform sampling from some specific bodies	i) Random walk method ii) Walk step iii) Target distribution iv) Parameters for the exact uniform sampling
volume()	A convex Polytope	Estimates the volume of a convex polytope	i) Random walk method ii) Walk step iii) Requested error iv) Other parameters for the two methods
rounding()	A convex Polytope	Brings the polytope to an approximated well-rounded position	i) Random walk method ii) Walk step
rotating()	A convex Polytope	Rotates randomly the input polytope	

Table 1: Overview of the most important functions of package VolEsti.

We use continuous integration practice to maintain and test the C++ part of VolEsti 2 . Additionally, there is a test suite for the functions of the R package. The package provides 8 functions, 4 exposed classes for convex polytopes and 8

polytope generators. Table 1 presents the most important functions of the package for our project. VolEsti provides 4 different representations for a convex polytope with a corresponding exposed C++ class for each representation. The HMC walk family and the implemented random walks in can be applied only for the H-representation, i.e., when the polytope is given by a set of linear inequalities. The package provides the following three random walk methods for sampling from the input polytope: a) Coordinate Directions Hit-and-Run (CDHR), b) Random Directions Hit-ad-Run (RDHR), and c) Ball walk. Moreover the user could choose between uniform and multidimensional spherical Gaussian target distribution. The CDHR it is shown to be the most efficient random walk in practice for many applications among many random walk algorithms. However, it does not scale beyond a few hundred dimensions, regardless the application. The plots in Figure 1 demonstrate two samples, one for each target distribution that VolEsti provides. Both methods (SeqOfBalls and CoolingGaussian) for volume estimation are based on sampling using a random walk while CDHR is the default choice. SeqOfBalls uses uniform sampling and CoolingGaussian uses Gaussian, while the first is statistically more accurate and CoolingGaussian is faster.

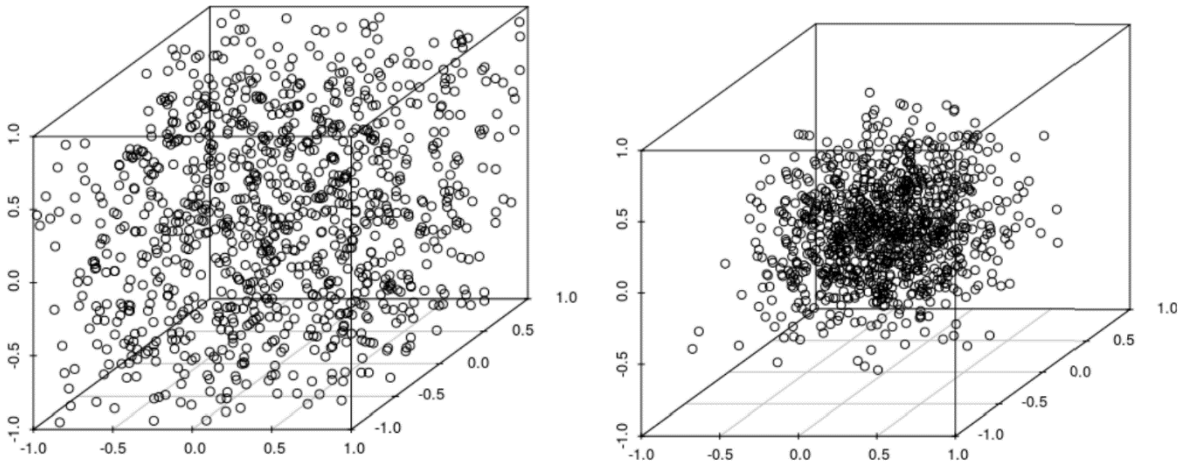


Figure1: Sampling from 3D unit cube using VolEsti. Uniform target distribution (Left) and spherical Gaussian with $\sigma^2 = 0.1$ centered at the origin (Right).

2.2 Extent VolEsti's functionality to handle multi-dimensional integrals.

Table 2 presents the methods I propose for implementation., so during the project I will make the necessary adjustments to extent VolEsti functionality. The project is splited in many parts for the better understanding and implementation.

A. Simple MC integration:

In numerical integration, methods such as the trapezoidal rule use a deterministic approach. Monte Carlo integration, on the other hand, employs a non-deterministic approach: each realization provides a different outcome. In Monte Carlo, the final outcome is an approximation of the correct value with respective error bars, and the correct value is likely to be within those error bars.

The problem Monte Carlo integration addresses is the computation of a **multidimensional definite integral**.

$$I = \int_{\Omega} f(\bar{x}) d\bar{x}$$

Where Ω, a subset of $\boldsymbol{R^m}$, has volume

$$\boldsymbol{V = \int_{\Omega} d\bar{x}}$$

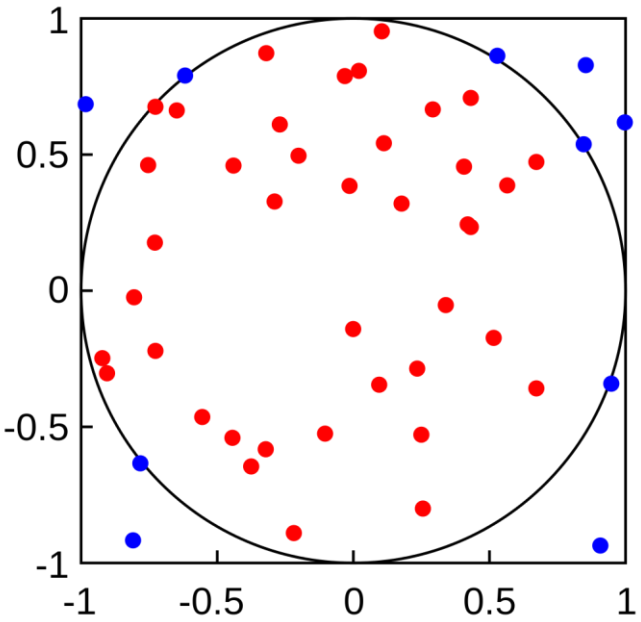


Figure 2: An illustration of Monte Carlo integration. In this example, the domain *D* is the inner circle and the domain E is the square. Because the square's area (4) can be easily calculated, the area of the circle ($\pi * 1.0^2$) can be estimated by the ratio (0.8) of the points inside the circle (40) to the total number of points (50), yielding an approximation for the circle's area of $4 * 0.8 = 3.2 \approx \pi$.

B. MISER Algorithm:

The MISER algorithm is based on recursive stratified sampling. This technique aims to reduce the overall integration error by concentrating integration points in the regions of highest variance.

The idea of stratified sampling begins with the observation that for two disjoint regions a and b with Monte Carlo estimates of the integral $\sum_a(f)$ and $\sum_b(f)$ and variances $\sigma_a^2(f)$ and $\sigma_b^2(f)$, the variance of $\mathrm{Var}(f)$ of the combined estimate.

$$E(f) = \frac{1}{2}\left(E_a(f) + E_b(f)\right)$$

Is given by,

$$\mathrm{Var}(f) = \frac{\sigma_a^2(f)}{4N_a} + \frac{\sigma_b^2(f)}{4N_b}$$

It can be shown that this variance is minimized by distributing the points such that,

$$\frac{N_a}{N_a + N_b} = \frac{\sigma_a}{\sigma_a + \sigma_b}$$

Hence the smallest error estimate is obtained by allocating sample points in proportion to the standard deviation of the function in each sub-region.

The MISER algorithm proceeds by bisecting the integration region along one coordinate axis to give two sub-regions at each step. The direction is chosen by examining all d possible bisections and selecting the one which will minimize the combined variance of the two sub-regions. The variance in the sub-regions is estimated by sampling with a fraction of the total number of points available to the current step. The same procedure is then repeated recursively for each of the two half-spaces from the best bisection. The remaining sample points are allocated to the sub-regions using the formula for N_a and N_b. This recursive allocation of integration points continues down to a user-specified depth where each sub-region

is integrated using a plain Monte Carlo estimate. These individual values and their error estimates are then combined upwards to give an overall result and an estimate of its error.

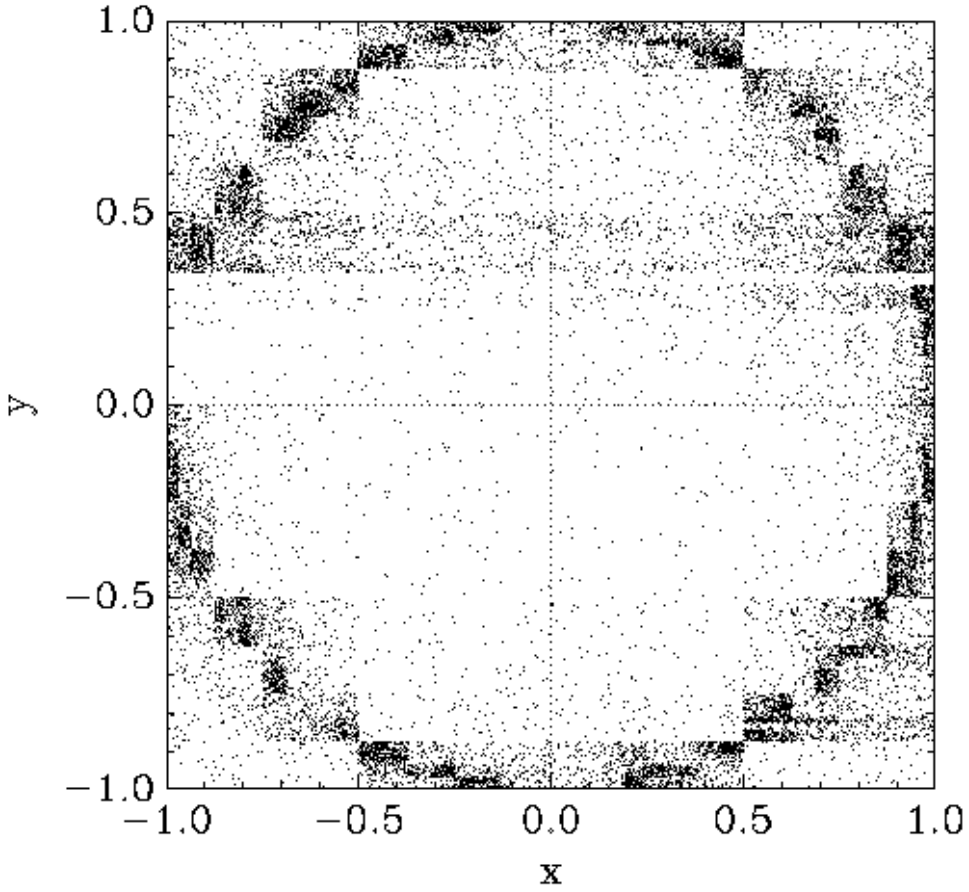


Figure 3: An illustration of Recursive Stratified Sampling. In this example, the function: from the above illustration was integrated within a unit square using the suggested algorithm. The sampled points were recorded and plotted. Clearly stratified sampling algorithm concentrates the points in the regions where the variation of the function is largest.

C. VEGAS Algorithm:

The VEGAS algorithm is based on importance sampling. It samples points from the probability distribution described by the function $|f|$, so that the points are concentrated in the regions that make the largest contribution to the integral.

In general, if Monte Carlo Integral of f over a volume Ω is sampled with points distributed according to a probability distribution described by the function g, we obtain an estimate $\sum_g(f;N)$,

$$\mathrm{E}_g(f;N) = \frac{1}{N}\sum_{i}^{N} f(x_i)/g(x_i).$$

The variance of the new estimate is then

$$\mathrm{Var}_g(f;N) = \mathrm{Var}(f/g;N)$$

Where $\boldsymbol{Var(f,N)}$ is the variance of the original estimate

$$\mathrm{Var}(f;N) = \mathrm{E}(f^2;N) - (\mathrm{E}(f;N))^2$$

If the probability distribution is chosen as $g = |f|/\int_\Omega |f(x)|\,dx$ then it can be shown that the variance $Var_g\,(f,N)$ vanishes, and the error in the estimate will be zero. In practice it is not possible to sample from the exact distribution g for an arbitrary function, so importance sampling algorithms aim to produce efficient approximations to the desired distribution.

The VEGAS algorithm approximates the exact distribution by making a number of passes over the integration region which creates the histogram of the function *f*. Each histogram is used to define a sampling distribution for the next pass. Asymptotically this procedure converges to the desired distribution.[9] In order to

avoid the number of histogram bins growing like K^d, the probability distribution is approximated by a separable function:

$$g(x_1, x_2, \ldots) = g_1(x_1)g_2(x_2)\ldots$$

so that the number of bins required is only Kd. This is equivalent to locating the peaks of the function from the projections of the integrand onto the coordinate axes. The efficiency of VEGAS depends on the validity of this assumption. It is most efficient when the peaks of the integrand are well-localized. If an integrand can be rewritten in a form which is approximately separable this will increase the efficiency of integration with VEGAS. VEGAS incorporates a number of additional features, and combines both stratified sampling and importance sampling

D. STAN (state-of-the-art platform for statistical modelling) :

In this project we have to Examine possible integrations with stan a state-of-the-art platform for statistical modelling and high-performance statistical computation. Stan provides a built-in mechanism to perform 1D integration of a function via quadrature methods. It operates similarly to the algebraic solver and the ordinary differential equations solver in that it allows as an argument a function. Like both of those utilities, some of the arguments are limited to data only expressions. These expressions must not contain variables other than those declared in the data or transformed data blocks.

As an example, the normalizing constant of a left-truncated normal distribution is

$$\int_a^\infty \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}$$

Performing a 1D integration requires the integrand to be specified somehow. This is done by defining a function in the Stan functions block with the special

signature:

```
real integrand(real x, real xc, real[] theta,
               real[] x_r, int[] x_i)
```

The function should return the value of the integrand evaluated at the point x.

The argument of this function are:

- *x*, the independent variable being integrated over
- *xc*, a high precision version of the distance from x to the nearest endpoint in a definite integral (for more into see section Precision Loss).
- *theta*, parameter values used to evaluate the integral
- *x_r*, data values used to evaluate the integral
- *x_i*, integer data used to evaluate the integral

Like algebraic solver and the differential equations solver, the 1D integrator separates parameter values, theta, from data values, x_r.

E. Weighted Model Integration :

Standard weighted model counting only supports discrete probability distributions. To repair this omission, WMC has recently been extended towards weighted model integration (WMI) (Belle, Passerini, and Van den Broeck 2015), supporting additionally continuous variables. However, the weight functions supported within current formulations of WMI (Belle, Passerini, and Van den Broeck 2015; Belle et al. 2016; Morettin, Passerini, and Sebastiani 2017; Kolb et al. 2018) allow only for piecewise polynomial functions. Moreover, none of these prior works has studied the applicability of knowledge compilation to WMI.

Compared to the well-known SAT problem, where the problem consists of deciding whether there is a satisfying assignment to a logical formula or not, an SMT problem generalizes SAT to additionally allowing the use of expressions formulated in a background theory.

Example 1. Consider the SMT theory broken:

broken ↔ (no cool ∧ (t > 20)) ∨ (t > 30) (1)

where no cool is a Boolean variable and t a real-valued variable. SMT then answers the question whether or not there is a satisfying assignment to the formula for the variables no cool and t.

2.3 Implementation :

I now give all the details about the implementations and how I plan to extent VolEsti's software. In the time schedule I propose to spend the 12th week for some general code optimizations.

C++ coding

Here I used the GNU Scientific Library (GSL) is a collection of routines for numerical computing

The library header files automatically define functions to have `extern "C"` linkage when included in C++ programs. This allows the functions to be called directly from C++.

To use C++ exception handling within user-defined functions passed to the library as parameters, the library must be built with the additional `CFLAGS` compilation option `-fexceptions`.

A. Simple MC Integration : There is lot of way to implement MC integration such as uniform sampling, stratified sampling, importance sampling, sequential Monte Carlo.

Here is a paradigmatic example of a Monte Carlo integration is the estimation of π. Consider the function.

$$H(x,y) = \begin{cases} 1 & \text{if } x^2 + y^2 \leq 1 \\ 0 & \text{else} \end{cases}$$

And the set $\Omega = [-1,1] \times [-1,1]$ with $V = 4$. Notice that

$$I_\pi = \int_\Omega H(x,y)dxdy = \pi.$$

Thus, a crude way of calculating the value of π with Monte Carlo integration is to pick N Random numbers on Ω and compute

$$Q_N = 4\frac{1}{N}\sum_{i=1}^{N} H(x_i, y_i)$$

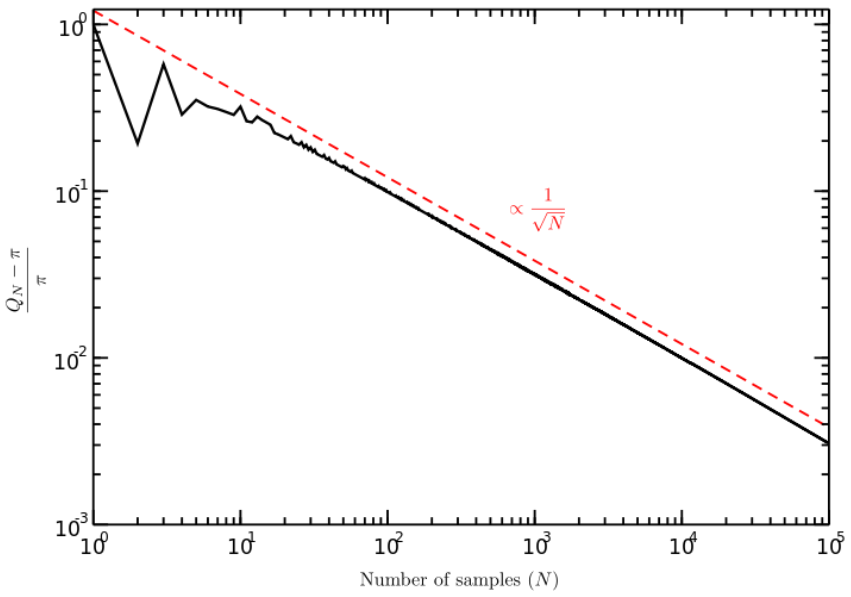


Figure 4: The relative error $\frac{q_{N-\pi}}{\pi}$ is measured as a function of N, confirming the $\frac{1}{\sqrt{N}}$

- The functions described in this section are declared in the header file `gsl_monte_plain.h`.

- **`gsl_monte_plain_state`**

  This is a workspace for plain Monte Carlo integration.

- **gsl_monte_plain_state * gsl_monte_plain_alloc(size_t *dim*)**

  This function allocates and initializes a workspace for Monte Carlo integration in **dim** dimensions.

- **int gsl_monte_plain_init(gsl_monte_plain_state* *s*)**

  This function initializes a previously allocated integration state. This allows an existing workspace to be reused for different integrations.

- **int gsl_monte_plain_integrate(gsl_monte_function * *f*, const double *xl[]*, const double *xu[ ]*, size_t *dim*, size_t *calls*, gsl_rng * *r*, gsl_monte_plain_state * *s*, double * *result*, double * *abserr*)**

  This routines uses the plain Monte Carlo algorithm to integrate the function **f** over the **dim**-dimensional hypercubic region defined by the lower and upper limits in the arrays **xl** and **xu**, each of size **dim**. The integration uses a fixed number of function calls **calls**, and obtains random sampling points using the random number generator **r**. A previously allocated workspace **s** must be supplied. The result of the integration is returned in **result**, with an estimated absolute error **abserr**.

- **void gsl_monte_plain_free(gsl_monte_plain_state * *s*)**

  This function frees the memory associated with the integrator state **s**.

B. MISER Algorithm :

The MISER algorithm proceeds by bisecting the integration region along one coordinate axis to give two sub-regions at each step. The direction is chosen by examining all d possible bisections and selecting the one which will minimize the combined variance of the two sub-regions. The variance in the sub-regions is estimated by sampling with a fraction of the total number of points available to the current step. The same procedure is then repeated recursively for each of the two half-spaces from the best bisection. The remaining sample points are allocated to the sub-regions using the formula for N_a and N_b.

The functions described in this section are declared in the header file gsl_monte_miser.h. All the MC Integration functions will be used. The MISER algorithm has several configurable parameters which can be changed using the following two functions .

- **void gsl_monte_miser_params_get(const gsl_monte_miser_state * *s*, gsl_monte_miser_params * *params*)**

  This function copies the parameters of the integrator state into the user-supplied **params** structure.

- **void gsl_monte_miser_params_set(gsl_monte_miser_state * *s*, const gsl_monte_miser_params * *params*)**

  This function sets the integrator parameters based on values provided in the **params** structure.

the necessary changes are made to the fields of the params structure, and the values are copied back into the integrator state using gsl_monte_miser_params_set(). The functions use the gsl_monte_miser_params structure which contains the following fields:

- `double estimate_frac` This parameter specifies the fraction of the currently

- `size_t min_calls` This parameter specifies the minimum number of function calls required for each estimate of the variance. If the number of function calls allocated to the estimate using **estimate_frac** falls below **min_calls** then **min_calls** are used instead. This ensures that each estimate maintains a reasonable level of accuracy. The default value of **min_calls** is **16 * dim**.

- `size_t min_calls_per_bisection` This parameter specifies the minimum number of function calls required to proceed with a bisection step. When a recursive step has fewer calls available than **min_calls_per_bisection** it performs a plain Monte Carlo estimate of the current sub-region and terminates its branch of the recursion. The default value of this parameter is ***32* * min_calls**.

C. VEGAS Algorithm :

The VEGAS algorithm approximates the exact distribution by making a number of passes over the integration region while histogramming the function f. Each histogram is used to define a sampling distribution for the next pass. Asymptotically this procedure converges to the desired distribution.

The VEGAS algorithm computes a number of independent estimates of the integral internally, according to the iterations parameter described below, and returns their weighted average. Random sampling of the integrand can occasionally produce an

estimate where the error is zero, particularly if the function is constant in some regions.

An estimate with zero error causes the weighted average to break down and must be handled separately.

The convergence of the algorithm can be tested using the overall chi-squared value of the results, which is available from the following function:

- **double gsl_monte_vegas_chisq(const gsl_monte_vegas_state * *s*)**

  This function returns the chi-squared per degree of freedom for the weighted estimate of the integral. The returned value should be close to 1. A value which differs significantly from 1 indicates that the values from different iterations are inconsistent

- **void gsl_monte_vegas_runval(const gsl_monte_vegas_state * *s*, double * *result*, double * *sigma*)**

  This function returns the raw (unaveraged) values of the integral **result** and its error **sigma** from the most recent iteration of the algorithm.

The VEGAS algorithm is highly configurable. Several parameters can be changed using the following two functions.

- **void gsl_monte_vegas_params_get(const gsl_monte_vegas_state * *s*, gsl_monte_vegas_params * *params*)**

  This function copies the parameters of the integrator state into the user-supplied **params** structure.

- **void gsl_monte_vegas_params_set(gsl_monte_vegas_state * *s*, const gsl_monte_vegas_params * *params*)**

  This function sets the integrator parameters based on values provided in the **params** structure.

Typically the values of the parameters are first read using **gsl_monte_vegas_params_get()**, the necessary changes are made to the fields of the **params** structure, and the values are copied back into the integrator state using **gsl_monte_vegas_params_set()**. The functions use the **gsl_monte_vegas_params** structure which contains the following fields:

- `double alpha`

  The parameter **alpha** controls the stiffness of the rebinning algorithm. It is typically set between one and two. A value of zero prevents rebinning of the grid. The default value is 1.5.

- `size_t iterations`

  The number of iterations to perform for each call to the routine. The default value is 5 iterations.

Rcpp Wrapping

D. STAN :

Stan is a C++ library for Bayesian modeling and inference. The R package **rstan** provides RStan, the R interface to Stan. The **rstan** package allows one to conveniently fit Stan models from R (R Core Team 2014) and access the output, including posterior inferences and intermediate quantities such as evaluations of the log posterior density and its gradients.

A C++ compiler, such as g++ or clang++, is required for use with Rstan. We have to install several packages on which Rstan package heavily depends

- **StanHeaders** (Stan C++ headers)
- **BH** (Boost C++ headers)
- **RcppEigen** (Eigen C++ headers)
- **Rcpp** (facilitates using C++ from R)
- **inline** (compiles C++ for use with R)

The Stan function wraps *3* steps:

- Translate a model in Stan code to C++ code
- Compile the C++ code to a dynamic shared object (DSO) and load the DSO
- Sample given some user-specified data and other settings.

A single call to stan performs all three steps, but they can also be executed one by one (see the help pages for stanc, stan_model, and sampling), which can be useful for debugging. The stan function returns a stanfit object, which is an S4 object of class "stanfit". For those who are not familiar with the concept of class and S4 class in R, refer to Chambers (2008). An S4 class consists of some attributes (data) to model an object and some methods to model the behavior of the object.

For class "stanfit", many methods such as print and plot are defined for working with the MCMC sample. For example, the following shows a summary of the parameters from the Eight Schools model using the print method:

```
print(fit1, pars=c("theta", "mu", "tau", "lp__"), probs=c(.1,.5,.9))
```

Arguments to the *stan* function :

The primary arguments for sampling (in functions stan and sampling) include data, initial values, and the options of the sampler such as chains, iter, and warmup. In particular, warmup specifies the number of iterations that are used by the NUTS sampler for the adaptation phase before sampling begins.

The optional init argument can be used to specify initial values for the Markov

chains. There are several ways to specify initial values, and the details can be found in the documentation of the stan function.

Stan uses a random number generator (RNG) that supports parallelism. The

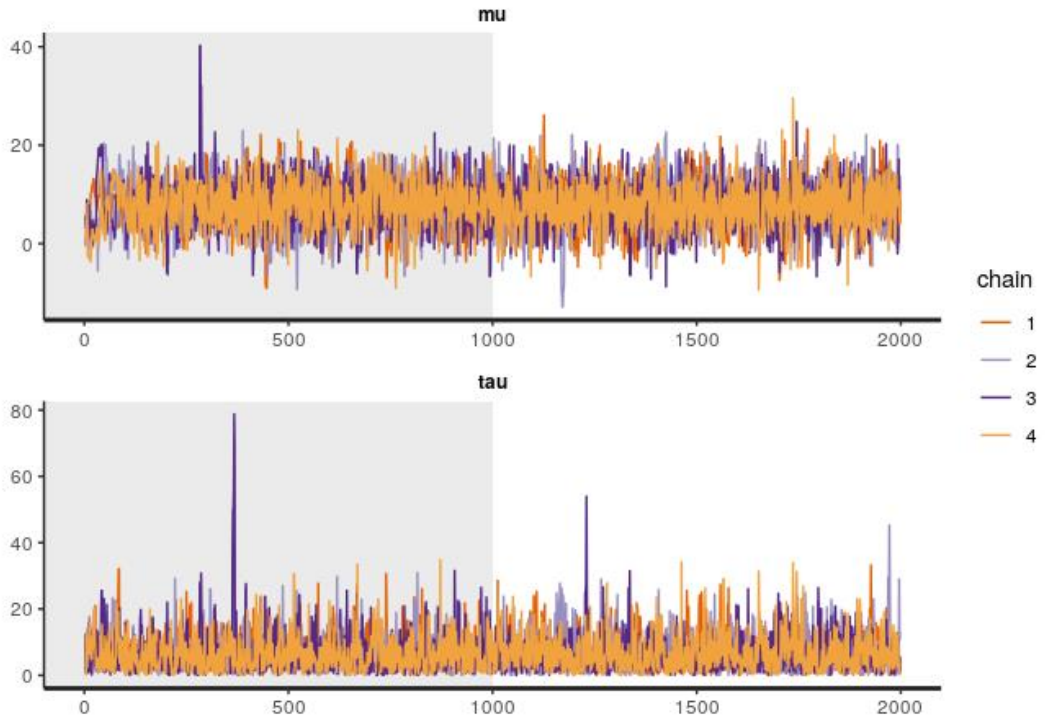
initialization of the RNG is determined by the arguments seed and chain_id. Even if we are running multiple chains from one call to the stan function we only need to specify one seed, which is randomly generated by R if not specified.

Methods for the "stanfit" Class :

The other vignette included with the **rstan** package discusses stanfit objects in greater detail and gives examples of accessing the most important content contained in the objects (e.g., posterior draws, diagnostic summaries). Also, a full list of available methods can be found in the documentation for the "stanfit" class at help("stanfit", "rstan").

The plot method for stanfit objects provides various graphical overviews of the output. The traceplot method is used to plot the time series of the posterior draws. If we include the warmup draws by setting inc_warmup=TRUE, the background color of the warmup area is different from the post-warmup phase:

```
traceplot(fit1, pars = c("mu", "tau"), inc_warmup = TRUE, nrow = 2)
```



2.4 Methodology

The implementations discussed in the previous subsection will have to be tested very carefully before I add them to the R package VolEsti. So every implementation will have to pass successfully some tests before I continue with the next one. In particular, I will follow the steps I describe in the sequel in order to complete the implementations

2. Create tests for the run time and the correctness of the output. Check if both of them verify the theoretical results.

3. If tests failed then improve the implementations and fix bugs until tests succeed.

antee efficiency and stability .

1. Implement the method in C++ following the corresponding papers and the steps we described in subsection 2.3.

4. If tests succeed then implement the Rcpp wrappers and modify the Rcpp functions of VolEsti.

4. Benefits to the Community:

The project proposal will significantly update VolEsti package to the first all-in-one package for different types of MC Integration. Hence it is very important for researchers and users working on applications like statistics, optimization and economics that need multi-dimensional integrations. Moreover our project's implementations will give the potential for some very interesting future extensions. We hope this project will be a decisive contribution for the first complete tool for high-performance statistical computation which could be used in many scientific and business applications

The project is expected to be beneficial to both (a) educational and (b) research open source communities. For (a) university students studying computational geometry, statistics or sampling will have access to an efficient library that will help them build intuition as well as it can serve as a platform for student projects. About (b) researchers could use the library in their research projects and also extend it according to their needs. Ultimately, I hope that this project will help building an international and interdisciplinary community in the intersection of the scientific fields of computational convex and discrete geometry, statistics, optimization, economics and other scientific applications.

4. Deliverables

The total number of weeks is thirteen. Thus I give an exact time schedule for the coding project starting at 17th May. At the end of the time schedule I will have completed all the implementations described in section 2. I divide the schedule to bonding and coding periods. In bonding period we are going to prepare all the technical details and environments in order to be ready to begin the coding right away at 7th June.

4.1 Bonding period (17th May - 6th June)

We will have three weeks to prepare everything for the coding period. I give a week time schedule in order to start coding at 27th May in the worst case.

- **1st week (17th May – 23rd May)**

1. I have to create a branch of the the current VolEsti project at [15].

2. I am going to create a blog in order to report our progress in a weekly basis.

3. Work with mentors on studding additional papers on MC integration in order to optimize the coding plan.

- **2nd week (24th May - 30th May)**

1. Work with mentors in the current code structure of VolEsti in order to set all the necessary environments and minor implementations. In particular, I will replace the std::vector with Eigen vector in ./include/cartesian_geom/point.h for the representation of points and modify the rest of the implementations in the package accordingly. This improvement will help us to apply linear transformation to samples more easily and will result to a more consistent and efficient - implementation.

- **3rd week (31th May - 6th June)**

1. Probably start coding. (Of course I do not count this week as a coding week, but ideally I will be ready to start).

4.2 Coding period (7th June - 16th August)

• **1st & 2nd week.** Implement Simple Monte Carlo Integration. (27th May - 8th June)

I am going to implement the steps I described previously.

• **3rd & 4th week.** Implement Monte Carlo Integration in more advanced level.

• **5th week.** Documentation, tests, code development for Monte Carlo Integration.

1. Implementing GSL(C++ Package) to implement the MC Integration.

2. Add comments to C++ code.

3. Finding the possible extension of VolEsti's functionality using the C++ package.

4. Write R tests in the script for Advanced MC Integration.

5. Run experiments to compare runtimes between multi-dimensional integrals.

• **6th & 7th week.** Implement MISER an VEGAS algorithm with the help of GSL.

For MISER create the new header file gsl_monte_miser.h and implement all the functions described Previously for the annealing schedule that constructs the sequence of recursive stratified sampling. This technique aims to reduce the overall integration error by concentrating integration points in the regions of highest variance.

For VEGAS use the function `gsl_monte_vegas_params` for the implementation of different functions for computing importance sampling.

• **8th & 9th week**. Complete implementation of MISER an VEGAS algorithm.

Implement the rest of the functions we described Previously. After this I will try

to extent VolEsti using the efficient C++ package for the easy computation of importance sampling and recursive stratified sampling .At the end of this week we would be able to complete implementation of MISER and VEGAS algorithm.

• **10th week**. Documentation, tests, code development of STAN .

1. Make a perfect environment for stan using packages mentioned previously.
2. Complete the steps for implementing stan function using C++.
3. Test the available integrations and existing methods of RStan package.
4. Examine possible integrations with Stan.

• **11th week.** Implementations of Applications to weighted model integration

• **12th week**. C++ code optimizations , write tests and Documentations.

4.3 Schedule Conflicts:

There are not schedule conflicts during the summer. This GCoC project will be a full time job in the summer for me and I am going to commit in a daily basis.

5. Related work:

This is my first attempt of GSoC . I have on some packages on VolEsti before the submission of my coding proposal to complete the tests provided by the mentors Vissarion Fisikopoulos , Pedro Zuidberg Dos Martires and Samuel Kolb .

Related software package exist starting from SI and cubature R packages to C++ package latte-integrale. I have worked on cubature package while doing the test. In these packages the first can scale to high dimensions but are limited to cubical domains.

domains (i.e. high-dimensional cubes) while the later is for more general convex domains (i.e. polytopes). Since latte-integrale is exact it cannot scale to high-dimensions (typically more than 15 dimensions). Therefore there is a need for efficient software that scales to high-dimensions for general convex domains.

5. Tests :

Easy

Test: Easy: compile and run volesti. Read the CRAN package documentation, generate a random H-polytope and compute its volume.

- Compiled and ran volesti tests in C++ interface using cmake, make and ctest. Also installed all package dependencies like Rcpp, RcppEigen, BH etc and used devtools to install volesti Rcpp Package
- Generated a random H-polytope and computed its volume

Code

```
1.   library(volesti)
2. P = gen_rand_hpoly(10, 50, generator = list(constants = "sphere"))
3. volumes <- list()
4. for (i in 1:10) {
5. volumes[[i]] <- volume(P) # default parameters
6. }
7. options(digits=10)
8. summary(as.numeric(volumes))
```

Medium

Test: Use the R package cubature to compute the integral of f(x) = exp^{-a||x||^2} over the cube [-1,1]^n, for various values of a and dimension n until it crashes.

- Installed R package cubeture and used cubeture library to compute the integral of f(x) = exp^{-a||x||^2} over the cube [-1,1]^n.
- I tried varios values of a=[0,1,2,3] and the values of n incrementing by 1 stating from the number 1
- At the value of n=24 the program got crashed due to the higher dimention used showing the result 0.
- Ovserving this we can say this package is way more less efficient than volesti package. using volesti approximation we can compute up to a few hundreds of dimensions.

Code

```
1. library(cubature) # load the package "cubature"
2. testFnWeb <- function(x) {
3.   exp(-a * sum(x^2))
4. }
5. hcubature(testFnWeb, rep(-1,n), rep(1,n), tol=1e-4)
```

Hard

Test: Use volesti to approximate the same integrals as in previous test by simple Monte Carlo based on uniform sampling and by Importance Sampling using multivariate spherical Gaussian. Comment on the accuracy and run-time.

- Used library volesti to compute integral f(x) = exp^{-a||x||^2} till 100th dimension using uniform and gaussian sampling.

Observation of f(x) = exp^{-a||x||^2}

- *Accuracy* : cubature is more accurate for small dimentations and it breaks easily for higher dimentions (n>=24). On otherside volesti perfectly works with a higher dimention with very less posibility of error.
- *Runtime* : Comparing upto 23rd dimention the runtime taken by volesti was definitely less then cubature.

Code

```
1. #installing volesti package
2. install.packages("volesti")
3. library(volesti)
4.
5. myfunction <- function(x) {
6.   y = exp(-a *sum(x^2))
7.
8. }
9.
10.## SIMPLE MONTE CARLO USING UNIFORM DISTRIBUTION
11.
12.P = gen_cube(d, 'H')
13.points = sample_points(P1, random_walk = list("walk" = "Baw"), n =
   10000,
14.                          seed = 5)
15.plot(points[1,],points[2,])
16.
17.
18.## SIMPLE MONTE CARLO USING GAUSSIAN DISTRIBUTION
19.
20.P <- gen_cube(d,"H")
21.
22.points = sample_points(P, n=10000 , distribution = list("density" =
   "gaussian", "variance" = 1)
23.
24.plot(points[1,],points[2,])
25.
26.
27.
```

Bonus

Task: Generate a 100-dimensional random H-polytope compute the largest inscribed ball (Chebychev ball) and let the center be the x0. Compute the integral of f(x) = exp^{-a||x-x0||^2} over the polytope for various values of a, 20 times each with both uniform and Gaussian sampling and take the average. Report the standard deviation for each experiment.

- Created HPolytope using gen_rand_hpoly() and computed centre x0 of the largest inscribed ball(Chebychev ball) along with radius using inner_ball().
- Computed the integral f(x) = exp^{-a||x-x0||^2} over the polytope for various values of a, 20 times using both uniform and Gaussian sampling and took the average and calculated standard deviation using standard R functions mean() and sd()

Code:

```
1. library(volesti)
2.
3. myfunction <- function(x,a=runif(1)) {
4.   y = exp(-a*(sum(x^2)))
5.   return(y)
6. }
7. cycles <- 20
8.
9. P = gen_rand_hpoly(100, 1000, generator = list(constants = "sphere"))
10.
11.z <- inner_ball(P) # center of the sphere coordinates is stored into the
   variable
12.print(paste("Radius of the inscribed ball = ",z[101]))
13.
14.points = sample_points(P, n=10000, distribution = list("density" =
   "gaussian", "variance" = 2))
15.
16.plot(points[1,],points[2,])
17.
```

7. Biographical Information :

7.1 Academic :

- B.tech UG at National Institute Of Technology Agartala, India
- Developer at Developer Students Club NITA

7.2 Programing :

- Working on R package VolEsti.
- Very Experienced in algorithm implementation and general programming.
- Experienced in programming with R and C/C++ .
- Completed a data science Project on R language .

7.3 Personal Motivation:

During my studies I got specially interested in algorithms and programming. I stayed active as a programmer by developing my programming skills and learning new programming languages. As I am a very active programmer especially in C++ and R communities.

Working on package VolEsti during Tests was a great experience. Working with mentors was superb. Google Summer of Code 2021 is a great opportunity significantly upgrade this work by providing R with new geometrical and statistical tools for the first time.

Coding was always one of my best activities. I enjoy to integrate algorithms that I study, to choose the proper programming language for each project and to learn new programming skills and languages. I think that the combination of C++ with R is very effective and practical as we could obtain speed and to provide nice interfaces to scientific or business communities that are not familiar with low level programming. So I think that the integration with open-source communities, that GSoC provides, is a great opportunity for me to participate and to contribute more.

References :

1. Café math : Monte Carlo Integration : A blog article describing Monte Carlo integration (principle, hypothesis, confidence interval)
2. Fast Algorithms for Logconcave Functions: Sampling, Rounding, Integration and Optimizationhttps://www.cc.gatech.edu/~vempala/acg/www/www/papers/integration.pdf
3. *Botev, Z.; Ridder, A. (2017).* "Variance Reduction". Wiley StatsRef: Statistics Reference Online: 1–6. doi:10.1002/9781118445112.stat07975. ISBN 9781118445112.
4. G. Peter Lepage, https://en.wikipedia.org/wiki/VEGAS_algorithm
5. *Ohl, T. (July 1999). "Vegas revisited: Adaptive Monte Carlo integration beyond factorization". Computer Physics Communications.* ***120*** *(1): 13–19. arXiv:hep-ph/9806432. Bibcode:1999CoPhC.120...13O. doi:10.1016/S0010-4655(99)00209-X.*
6. ARTICLES YOU MAY BE INTERESTED IN Monte Carlo methods for multidimensional integration for European option pricing AIP Conference Proceedings 1773, 100009 (2016); https://doi.org/10.1063/1.4965003
7. W.H. Press, G.R. Farrar, *Recursive Stratified Sampling for Multidimensional Monte Carlo Integration*, Computers in Physics, v4 (1990), pp190–195.
8. G.P. Lepage, *A New Algorithm for Adaptive Multidimensional Integration*, Journal of Computational Physics 27, 192–203, (1978)
9. GSL https://www.gnu.org/software/gsl/doc/html/montecarlo.html
10. RStan https://cran.rproject.org/web/packages/rstan/vignettes/rstan.html
11. Extending R with C++: A Brief Introduction to Rcpp https://cran.r-project.org/web/packages/Rcpp/vignettes/Rcpp-introduction.pdf