Computing report

Introduction

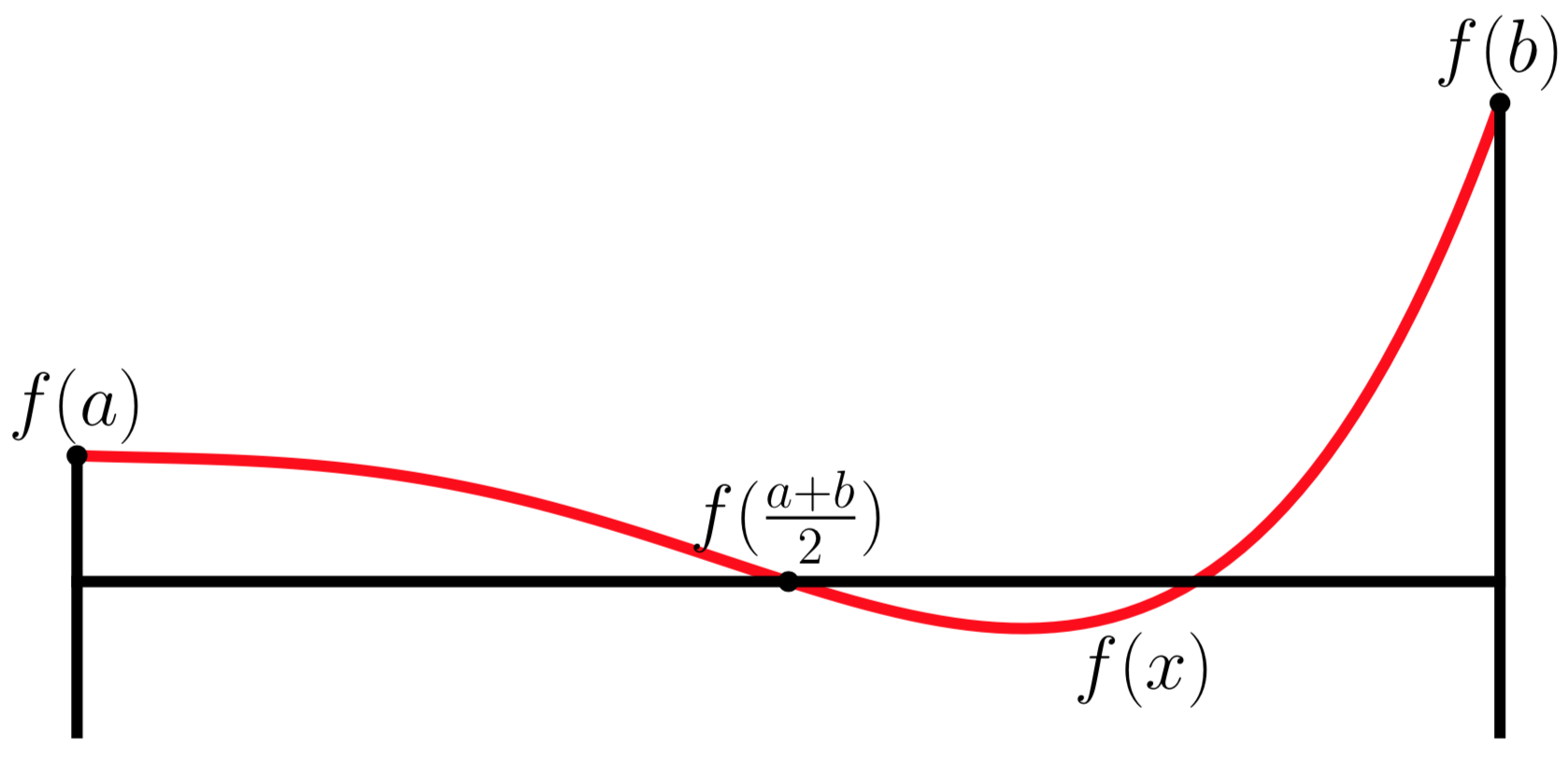
Determining the cross section of a particular interaction is of the upmost importance in several particle physics experiments, as the cross section is an indication of the probability of an interaction occurring. Calculating the cross section of a two-to-two interaction, such as electron scattering, requires the integration of two variables; a two-to-n interaction requires integrating over 3n-4 variables. Given that a typical LHC process produces hundreds of particles, a typical integration problem would require integrating hundreds of variables. This is practically impossible to perform analytically; therefore, numerical integration methods must be used.

There are several numerical integration methods that could be used in particle physics, each with varying characteristics of accuracy and convergence. Traditional quadrature-based methods such as the Newton-Cotes method are simple to implement and converge quickly in one dimension. Monte Carlo integration utilizes random number generators to evaluate the integral with relative speeds. Each of these methods could be enhanced by adapting which points are sampled to provide faster convergence. The VEGAS Algorithm uses a modified form of Monte Carlo integration which alters the probability density function of the random number generator to increase the convergence quickly. Generalizing each of these methods in N dimensions is significantly more challenging and would also alter the characteristics of each method.

The aim of this project was to investigate and characterise multiple numerical integration methods and determine which would be best suited for use in particle physics. The integration methods investigated were the Newton-Cotes quadrature rules – the midpoint, trapezium, and Simpsons rule–, adaptive numerical integration using Newton-Cotes, Monte Carlo and its stratified sampling counterpart. The main objectives were to implement these methods in Python, generalize them to N dimensions, checking the uncertainty as a function of sampled points, perform timing tests, and demonstrate convergence.

Physics review

The Newton-Cotes (NC) quadrature rules are a set of rules used to calculate the integral by numerically interpolating the integrand as a polynomial through equally spaced intervals. The simplest NC rule is the midpoint rule, where the function is approximated as a constant evaluated at the centre of the limits, as shown IN FIGURE SOMETHING and mathematically expressed in EQUATION SOMETHING.

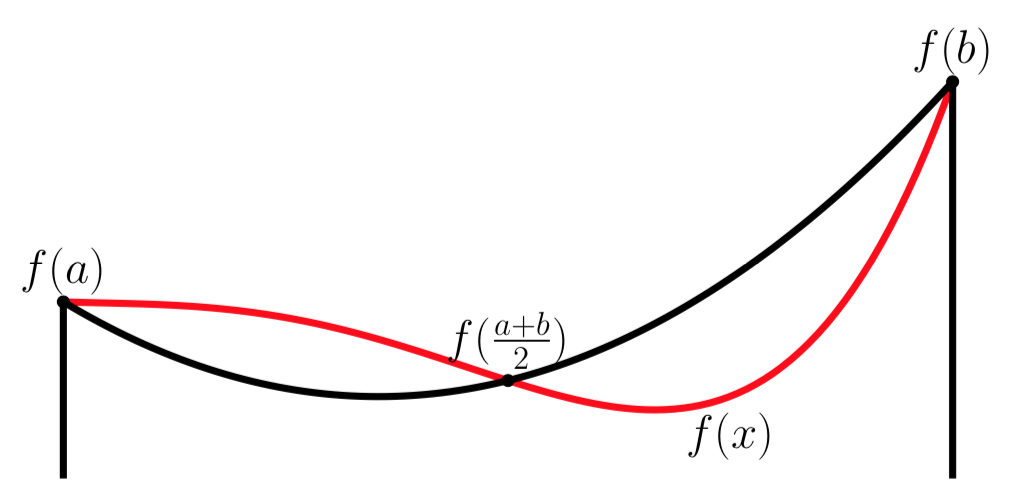


The trapezium rule increases the accuracy of the value by approximating the function as a one dimensional polynomial instead, shown IN FIGURE SOMETHING AND EQUATION SOMETHING.

A close up of a logo

Description automatically generated

Simpson’s rule builds upon this accuracy by now approximating the function as a two-dimensional polynomial, shown in FIGURE SOMETHING AND EQUATION SOMETHING.



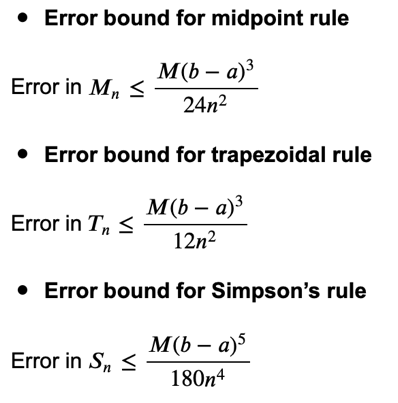
Higher order NC quadrature rules do exist and would theoretically increase the accuracy; however, these are impractical as higher order polynomial approximations could suffer from the Runge’s Phenomenon, fluctuating wildly as the polynomial degree increases. Due to the Runge Phenomenon, only NC quadrature rules up to the second order were analysed for this project.

Previous representations of NC quadrature use only the lower and upper limits, providing a large uncertainty in the integration. The integral can instead be divided into a series of equally spaced intervals, after which any NC quadrature method can be applied in each interval to calculate the value of the integral. Applying this generally to each NC quadrature method leads to the NC formula,

where wi are the weights used for each point. The weights for each rule are found IN TABLE SOMETHING, which could be derived from the previous definitions (REFERENCE).

|  |  |
| --- | --- |
| Quadrature Method | Weights |
| Midpoint Rule | [0, h, h, …, h] |
| Trapezium Rule | [h/2, h, h, …, h/2 |
| Simpsons Rule | [h/3, 4h/3, 2h/3, 4h/3, 2h/3, … , h/3] |

The errors of each of the methods are well known, and are shown in TABLE SOMETHING. Due to the higher order approximation from Simpson’s rule,



https://math.libretexts.org/Courses/Mount\_Royal\_University/MATH\_2200%3A\_Calculus\_for\_Scientists\_II/2%3A\_Techniques\_of\_Integration/2.5%3A\_Numerical\_Integration\_-\_Midpoint%2C\_Trapezoid%2C\_Simpson%27s\_rule

The convergence of these methods is well known and are shown in TABLE SOMETHING. Naturally, due to the higher order estimation of Simpson’s rule, it was expected that Simpsons Rule would converge faster. Interestingly, the convergence of the midpoint rule and the trapezium rule are of the same order O(1/n^2), and unexpectedly the error in the Midpoint rule is 50% better than the trapezium rule. This was because the trapezium rule consistently overestimates the value when the integral has a positive curvature and underestimates the value when it has a negative curvature, while the midpoint rule partially averages the error by the same interval. APPENDIX SOMETHING PROVIDES A BETTER EXPLANATION.

Adaptive integration

The convergence of the quadrature rules could be improved using adaptive integration by systematically choosing the appropriate interval to integrate over rather than using equally spaced intervals. The value of the interval is first estimated using any NC quadrature rules, and the interval is subdivided into two divisions. The value of the integral in each division is calculated, and if the difference between the first estimation and the sum of the subdivisions is larger than an error tolerance, then then the algorithm stops. If its larger, the intervals are further divided, continuing until the error is less than the tolerance. Selectively choosing which intervals to divide over would lead to faster convergence, as intervals are calculated only when required.

The NC quadrature could be generalized to N dimensions by applying each quadrature individually in each dimension throughout all the sampling points. Using combinatory logic, the general equation of NC quadrature in N dimension is

A close up of a logo

Description automatically generated

.

The two dimensional case is illustrated IN FIGURE SOMETHING.

A close up of a logo

Description automatically generated

A problem rapidly develops when using NC quadrature in N dimensions. In FIGURE SOMETHING, a total of 12 points must be used even when only 3 and 4 points are used in each dimension. The number of points required to perform the integral scales exponentially with the number of dimensions, as shown in TABLE SOMETHING.

|  |  |
| --- | --- |
| Quadrature Method | Error Scales as |
| Midpoint Rule | 1/N^(2/d) |
| Trapezium Rule | 1/N^(2/d) |
| Simpsons Rule | 1/N^(4/d) |

Overall, while NC quadrature converges quickly in one dimension, performance decreases in multidimensional integrals and would therefore be ill suited for use in particle physics.

MONTE CARLOOO

Monte Carlo (MC) Integration is another numerical integration method, utilizing random numbers to sample the integrand within the limits to determine its average value. The equation form of Monte Carlo is given as

in one dimension, and is easily generalizable to N dimensions through.

The convergence of Monte Carlo Integration scales with 1/√N in any dimensions, which is its main advantage and disadvantage. In one dimension, any NC quadrature method converges faster at a minimum of 1/N^2 compared to Monte Carlo’s 1/√N. However, in multiple dimensions, the convergence of NC quadrature deteriorates while Monte Carlo’s does not, thereby making it the preferred choice of numerical integration method for multi-dimensional problems.

Monte Carlo integration can be improved by implementing variance reduction techniques in order to increase the speed of convergence. One such technique is using a modified form of stratified sampling, which behaves similarly to adaptive integration. The integral is divided into several bins, and each bin is sampled. If the variance in a particular bin, given as

is larger than a tolerance, the bin is subdivided into two smaller bins and the process is repeated. This has the effect of sampling more points in areas where the integrand changes quickly, thus ensuring a more accurate value. An illustration of this is SHOWN IN FIGURE SOMETHING

Fig something. There are more strata of smaller intervals in areas where the function changes dramatically, and less strata where the function is stable. The value in each interval is calculated and added together to determine the value of the integral.

Stratified sampling was chosen over other variance reduction techniques such as importance sampling or SOMETHING HYPERCUBES due to the simple concept and implementation. Due to time restrictions, only simple variance reduction techniques could be implemented in time. Furthermore, a similar technique was used by the VEGAS algorithm to perform its adaptive Monte Carlo integation, with the main difference being that the grid adjustment algorithm is more advanced than this one (REFERENCE).

Algorithms and code structure.

All mentioned integration methods were applied in a single class, with each integration method being its own distinct method in the class. This was an intentional choice, as object oriented programs tend to be easier to maintain and use. Furthermore, keeping all the methods in a single class would allow it to compare characteristics of different methods.

Each of the three NC methods and their composite counterparts were implemented in one dimension in different methods, following the mathematical description provided in SECTION SOMETHING. Once each method was implemented, the general NC quadrature method was implemented, and its performance was verified against the original implementations.

The mathematical description of adaptive integration was easily implemented in python due to its recursive nature. Pseudocode of the implementation is shown below to demonstrate how recursion was used to perform the integration.

Def AI(f, a, b, tolerance):

Midpoint = (a+b)/2

Value = NCIntegrate(f, a, b)

Value1 = NCIntegrate(f, a, midpoint)

Value2 = NCIntegrate(f, midpoint, b)

If |Value – (Value1+Value2)| > tolerance:

Value = AI(f, a, midpoint, tolerance/2) + AI(f, midpoint, b, tolerance/2)

Return value

The code for Monte Carlo integration simply followed the mathematical description mentioned before. Implementing the monte carlo method required the use of a pseudo-random number generator, therefore the random module was used. A seed was used in the code to induce reproducibility in order to easily detect if a problem arose and also to provide consistent results.

The algorithm for stratified sampling quite complex, given that the interval needs to be divided and evaluated using monte carlo. A new inner class was created to handle the strata and associated methods required in the algorithm, including performing monte carlo within its limits and subdividing the strata. The inner class allowed for a more streamlined code that is easy to develop, maintain, and use.

Generalizing each method to N dimension required modifying the code to accept multiple dimensions, achieved mostly by converting the upper and lower limits from floats to arrays with N elements. Generalizing NC methods were challenging due to the summations found in EQUATION SOMETHING requiring the algorithm to loop through all the possible combinations of indexes, a two dimensional example shown below.

A close up of a logo

Description automatically generated

A generating function in python was used to generate the list of index K required, such that k =[0,0]. This, combined with a two dimensional Weights list, allowed the algorithm to loop through all possible combinations

A black sign with white text

Description automatically generated

The monte carlo and stratified sampling algorithms was relatively straightforward to generalize to N dimensions, as they have the same structure as the one-dimensional counterparts. The challenge came by when dividing the bins in multiple dimensions, however the index generator developed before solved the problem by looping throughout all the required combinations easily.

A black and silver text on a screen

Description automatically generated

ADAPTIVE INTEGRATION.

Once all the integration methods were developed in python, each integration methods need to be tested to show their convergence and timing tests. Teah method was therefore tested against several functions, shown below

FUNCTIONS LIST

Results

1d

* NCINT each produce the same timing tests (to be expected)
* All of the NCInt methods converge instantaneously on f1, which was expected
* Monte carlo ‘converges’ slowly, compared to those guys
* Adaptive integration converges hella quickly
* Show for standard functions f1, f3, and Gaussian

4d

* Do the area of a hypersphere
* Results
  + Show that the newton cotes is superior in the one dimensional case
  + show that the one d convergers are accurate to the ones mentioned there
  + show that monte carlo is superior in the four dimensional case
  + validity? Probably compare it with known thingys
* Conclusion
  + Shruggie