Computing report

Introduction

TALK ABOUT VEGAS SOMEWHERE IN IT

Integrations play an important part in the study of particle physics. Calculating the cross section of a particular interaction is one of the goals in this study; determining the probability of a particular interaction will lead to more studies within it. Physicists often collide two particles together and calculate the probability of a particular interaction happening, the classic example being an electron-anti electron pair interacting to produce a muon-anti muon pair. This two-to-two process requires integrating over two variables, in general a two-to-n process requires integrating over 3n-4 variables. Given that a typical LHC process contains hundreds of particles, performing this multidimensional integral becomes challenging and practically impossible analytically.

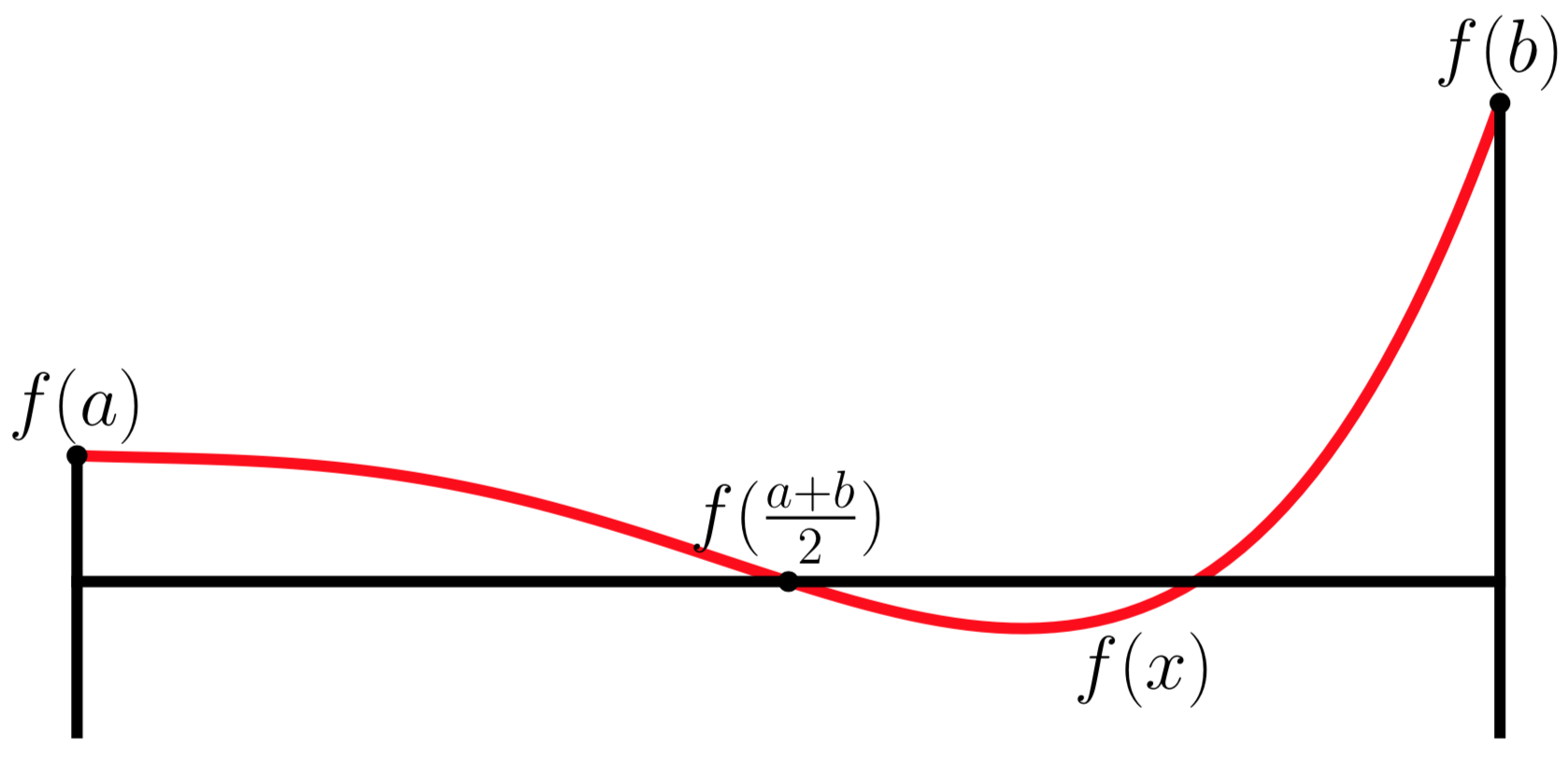
Numerical methods exist that would alleviate the problem. Traditional quadrature-based methods exist that can numerically integrate one dimensional problems with ease, as well as Monte Carlo integrations. Each integration method has its own convergence and error characteristics, the investigation of which would allow us to determine which integration method is best used for particle physics. Generalizing these methods in N-dimensions would be challenging and would also change the aforementioned characteristics of each method. The

The aim of this project was to investigate multiple numerical integration methods to determine which would be best suited for use in particle physics. The integration methods investigated were LIST THEM HERE. The main objectives were to implement these methods in python, generalize them to N dimensions, checking the uncertainty as a function of sampled points n, and perform timing tests, and demonstrating convergence.

Physics review

* Midpoint rule
* Trapezium rule
* Simpsons rule
* ^make sure this is a really short discussion for this
* Talk about the convergences of the thingys
* Newton-cotes rule in general
* Adaptive integration
* Monte carlo
* General discussion about converting it to an n-dimensional integration

A simple way to determine the value of the integral is to evaluate the integrand along equally spaced intervals, which are called the Newton-Cotes quadrature rules. The simplest of these is the midpoint rule, where the function is approximated as a zero order polynomial.

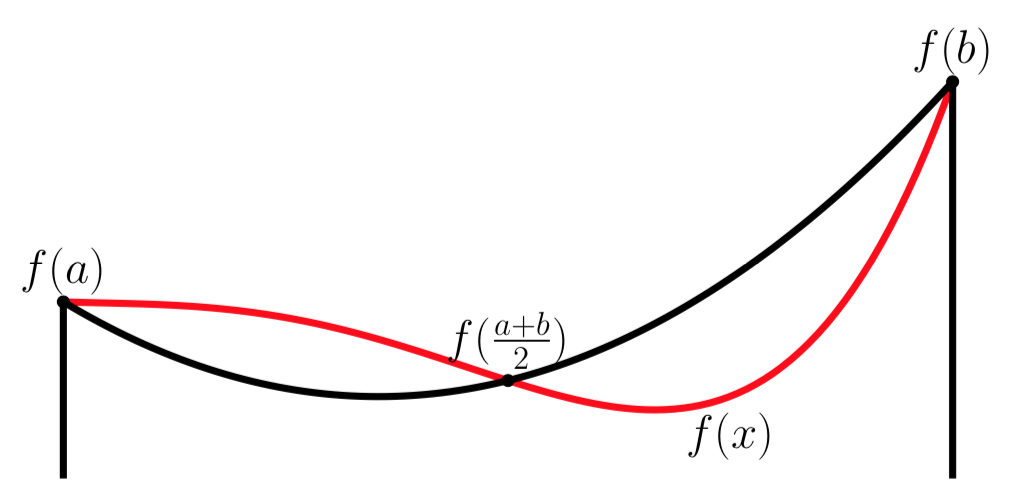


A more accurate method is to approximate the function as a one-dimensional polynomial, which is called the trapezium rule.

A close up of a logo

Description automatically generated

Going further from this, Simpson’s rule approximates the function as a two-dimensional polynomial.



In general, the Newton-Cotes approximations can be done for higher polynomial degrees, however this is impractical as higher polynomial approximations tend to fluctuate wildly due to the Runge’s phenomenon. Therefore, this project will only tackle Newton-Cotes methods up to Simpson’s rule.

As of right now, the quadrature rules have only been using the limits a and b, as well as the midpoint a+b/2. To increase the accuracy of the integration, the integral can be subdivided into smaller equally spaced intervals and evaluate the value of each interval using the quadrature rules. This method is called the composite integration method.

Generalizing the composite newton-cotes integration method is shown in equation this

Where and w\_i are the weights of the quadrature method. The table of weights are shown below

|  |  |
| --- | --- |
| 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] |
|  |  |

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

The convergence of these methods are 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.

Adaptive integration

Another way to increase the accuracy of the integration is through something called adaptive integration. Instead of using equally spaced intervals, the integral is divided into smaller divisions until some cutoff is reached. The integral in a particular division is calculated, and then the division is subdivided into two. The value of the integral in the subdivisions are then calculated. If the difference between the value of the original division subtracted with the sum of the values of the subdivisions is above an error cutoff, then the process repeats until the difference is less than the error cutoffs. This can significantly reduce the number of times and divisions it takes to converge.

Newton-Cotes formulas have only been shown in one dimensions so far, however they can easily be generalized to N dimensions. This is done simply by applying the quadrature method individually in each dimension, and multiplying them together. The two dimensional case is illustrated in FIGURE SOMETHING.

A close up of a logo

Description automatically generated

Where 

A close up of a logo

Description automatically generated

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

However, the main downside of using newton-cotes quadrature methonds in multiple dimensions is that the number of points required to perform the integral grows exponentially with d. As an example, where in 1 dimension an integral can be calculated using only 2 points, 2 dimensional integrals require 4 points, 3 dimensions require 8 points, and so on. This also means that the number of points required to converge also increases exponentially, as seen in table something.

Monte Carlo Integration differs from previous quadrature methods in that this one uses random number generators to evaluate the function at random points and use the average value of the function as its approximation.

The convergence of monte carlo scales with 1/√N, so in one dimension it would take more points for it to converge before the Newton-Cotes quadrature methods do. However, the main advantage of monte carlo is that the convergence of monte carlo remains 1/√N, regardless of the dimensionality of the integral. Therefore for particle physics monte carlo methods would be best suited for it.

While already good, monte carlo could be made even better by employing other methods to reduce the variance of the integral as quickly as possible. One variarnce reduction technique is called the stratified sampling model, where more points are sampled in areas where the variance is larger. This was done by dividing the integral to a series of bins, at which each bin the value of the monte carlo integral and the variance is calculated in there.

The variance is calculated via

If the variance in each bin is larger than a certain value, the bin is divided into two smaller bins, and the process is repeated. This is not unlike adaptive integration.

Algorithms and code structure

All mentioned integration methods were applied in one single class, with each of the integration methods being its own distinct method in the class. This allows it to be easily imported and used, as well as easy to maintain as everything is within one concrete package.

Each of the three newton-cotes method was implemented as its own method following the mathematical equations shown before. The composites of each equation was also implemented in each method. The Newton-cotes integration method for one dimension and multi dimensions were also implemented in separate methods, which required the weights w method to be defined as well. This was a conscious decision to separate out all these methods, such that the original and composites could be compared side by side as a verification, as well as the composites verifying the implementation of the general newton-cotes formulas.

Adaptive Integration talky talky

Implementing the adaptive integration required the use of recursion, as it was simply calling the function multiple times until a certain condition was met. Talk more about this. An example of this was using pseudocode is shown below.

1. **procedure** integrate ( f, a, b, τ )

2. Q≈∫abf(x)dx

3. ε≈|Q−∫abf(x)dx|

4. **if** ε>τ **then**

5. m = (a + b) / 2

6. Q = integrate(f, a, m, τ/2) + integrate(f, m, b, τ/2)

7. **endif**

8. **return** Q

Monte carlo talky talky

The monte carlo method required the use of the random module found in python to generate the pseudo-random numbers. The number of sample points was specified, after which the code performed the monte carlo integration as shown by the equation.

Stratified sampling was built upon the original monte carlo code, however required additional characteristics. The integral needs to be divided in to multiple bins, which each hold the value of the monte carlo integral as well as the value. The bins need to also be subdividable into two, to perform the stratified sampling. Due to this, a new class called Bins was developed that would serve as a support class to the stratified sampling method. The Bins class would have a monte carlo method that would calculate the value of that bin, as well as a Bisection method that would split the bins in two and perform the Monte carlo integrations in the new bins. With the new bins class, the stratified sampling algorithm can be performed as described in the pervious section.

Generalizing each method required modifying the code to accept multiple dimensions. The Newton-Cotes rules in multiple dimensions was relatively straightforward, aside from indexing the weights. Altering the method to multiple dimensions mostly involve converting the limits from floats to lists, where each element in the list represents the limits in a particular dimension.

Performing the w product rule is challenging, as we have to loop over all the points as shown in fIGURE SOMETHING. In a two dimensional case with only two points in each dimensions, the weight combinations are shown in the following table.

[0,0], [0,1], [ 1,0], [1,1]

[[w1, w2], [w3, w4]]

Wlist[d][V[d]]

A black sign with white text

Description automatically generated

This was solved by creating a Wlist, where each element is the wlist for each dimension. A python generator was used to create an indexer, which would loop through all the possible combinations of the points and add them together.

The monte carlo and stratified sampling algorithms was relatively straightforward to generalize to N dimensions, as they hold the same structure and use the same random generation technique. This basically involved generating random numbers in each dimension, and using these numbers to evaluate the function at this point. The algorithm to perform this is shown below.

A black and silver text on a screen

Description automatically generated

Performing the bisection in one dimension was trivial, however generalizing to N dimensions required more thought. In two dimensions, dividing each dimension by two would lead to 4 divisions, 3 dimensions to 8, and so on. Going through the different dimensional combinations required reusing the indexer algorithm that was previously developed.

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