Computing report

Introduction

MENTION MISER MONTE CARLO AND HOW ITS RECURSIVE STRATIFIED SAMPLING

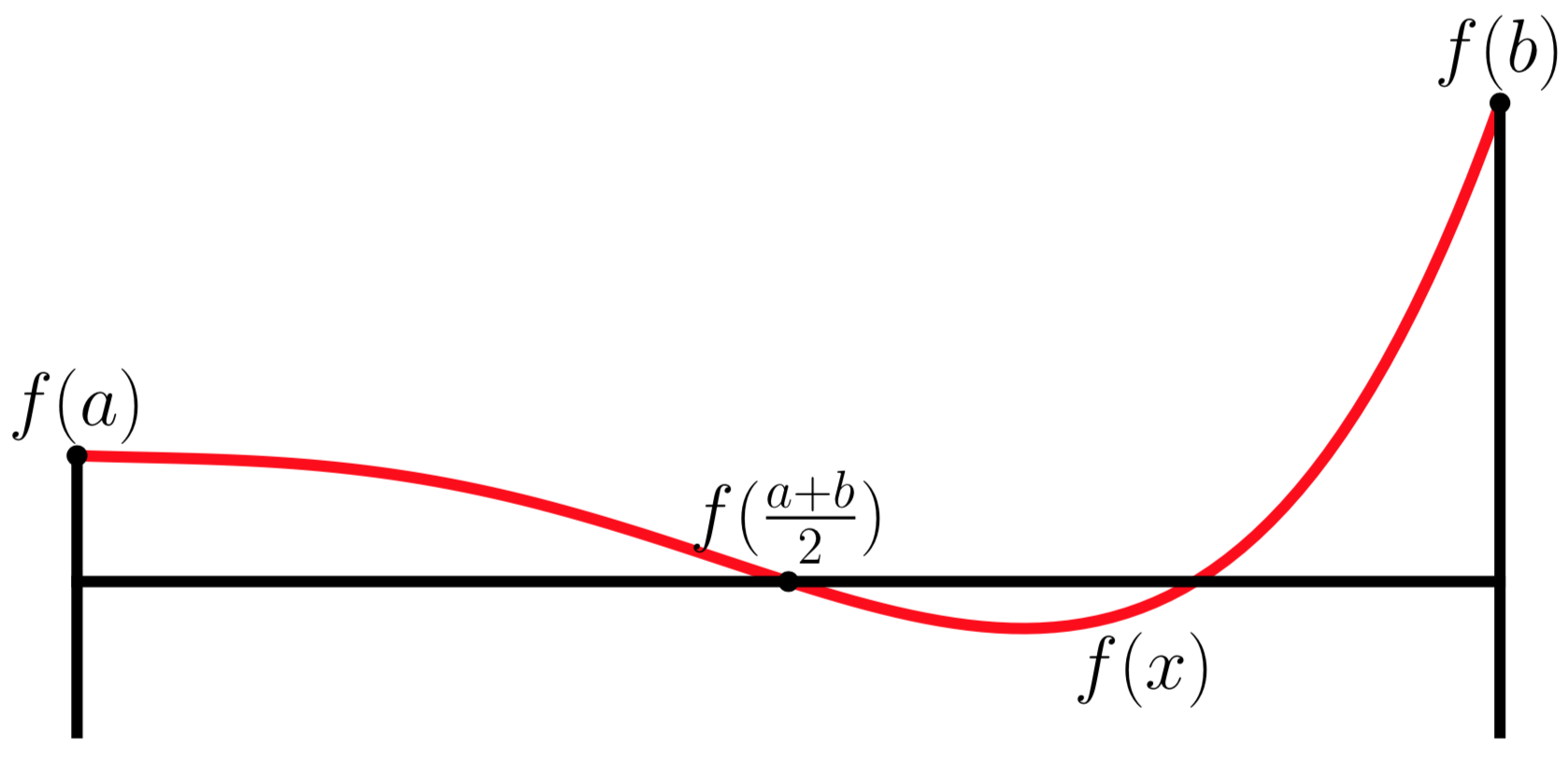
Determining the cross section of a particular interaction is of the upmost importance in several particle physics experiments, as the cross section is an indication of the probability of an interaction occurring. Calculating the cross section of a two-to-two interaction, such as electron scattering, requires the integration of two variables, while 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, thus requiring the use of numerical integration methods to perform such tasks.

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 relies on the random sampling of the function to evaluate the integral relatively efficiently. Each of these methods could be enhanced by adapting which points are sampled to provide faster convergence. The VEGAS Algorithm is one such enhancement, using a modified form of Monte Carlo integration where the random number distribution is adapted over several iterations to achieve rapid convergence. Generalizing and implementing these methods in N dimensions is non-trivial 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 of which is the midpoint rule, where the function is approximated as a 0th order polynomial evaluated at the midpoint of the limits a and b, as shown in FIGURE SOMETHING. The midpoint rule is mathematically expressed as

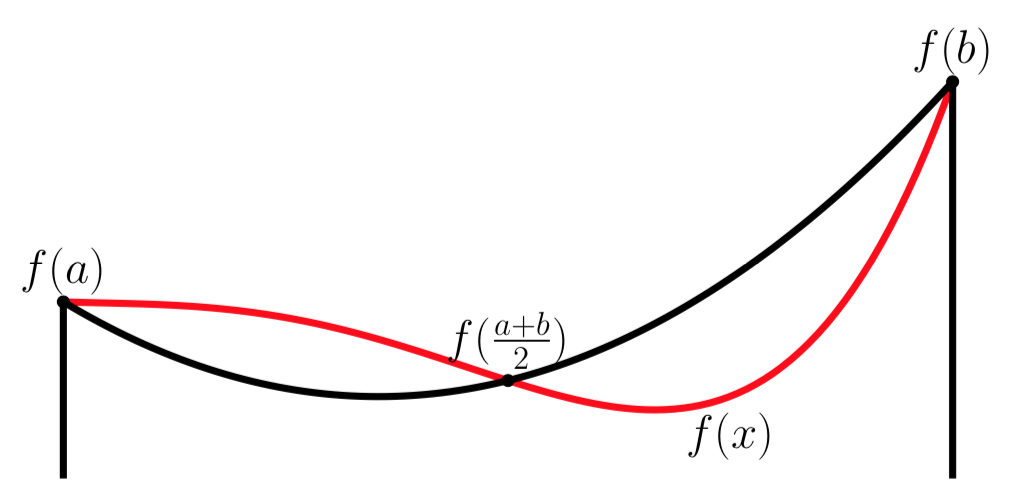


Instead of approximating the function as a 0th order polynomial, a 1st order polynomial could also be used. The trapezium rule is the NC quadrature rule that uses this approximation, illustrated in FIGURE SOMETHING. The mathematical form of the trapezium rule is

A close up of a logo

Description automatically generated

Simpson’s rule uses a higher 2nd order polynomial to approximate the function to increase the convergence and accuracy of the result. Simpson’s rule is illustrated in FIGURE SOMETHING, and represented mathematically as

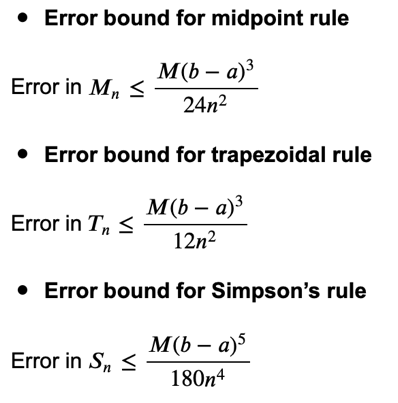


Higher order polynomial approximations for numerical integrations are available and would theoretically increase the accuracy, however are rarely used in particle physics due to the approximations possibly suffering from the Runge’s Phenomenon for certain functions. The approximation would vary wildly as the polynomial degree increases, as shown IN FIGURE SOMETHING. In this case, the error would tend towards infinity as the number of polynomials were increased. Due to the Runge Phenomenon, only NC quadratures of the 0th, 1st, and 2nd order were analysed for this project.

As of right now, NC quadratures utilizing only the lower and upper limits have been introduced. These can be enhanced by dividing the integral into a series of equally spaced intervals, after which any NC quadrature method is used to evaluate the integral at every division. Naturally, using more intervals and more sampling points would lead to a more accurate evaluation, and its behaviour as more sampling points are used can be studied. Applying the division generally to each NC quadrature method leads to

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] |



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. As the midpoint rule and the trapezium rule both approximate the function as straight lines, the convergence of the two are of the same order O(1/n^2). However, the midpoint rule has a better error bound compared to the trapezium rule as the trapezium rule systematically overestimates the value of the integral when the function has a positive curvature, and underestimates the value when it has a negative curvature. Meanwhile, the midpoint rule’s estimate is partially averaged over the same interval. SEEN IN THE APPENDIX.

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

NCInt

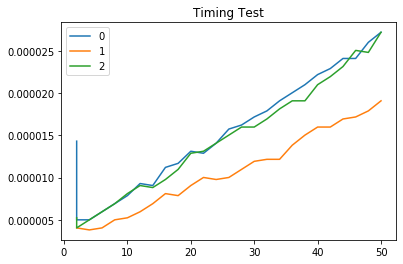
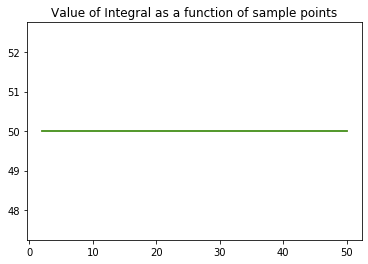
Sections

1. 1 D
   1. NCInt
      1. Midpoint, trapezium, simpsons
         1. F3, F5, G
      2. Adaptive
   2. Monte Carlo
      1. Crude
      2. Strartified
2. 4D
   1. Function is a hypersphere
   2. NC Int N
      1. Use only simpsons NC INT N
   3. Monte Carlo
      1. Crude and Stratified

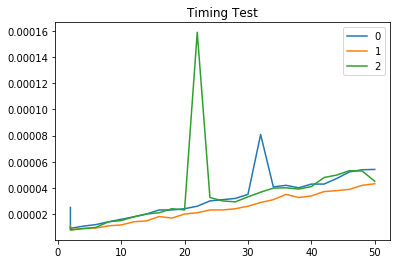
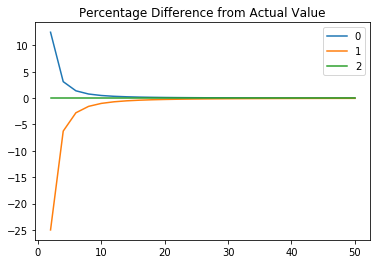
The newton-cotes rules produce the expected answer in one dimension relatively quickly, achieving the expected result to an uncertainty of 1% in less than 50 sampling points. As expected, Simpson’s rule converges the fastest when dealing with a well behaved polynomial in f3 and f5. It converges especially quickly in f3 due to it being a very simple function and that it is only a 3rd order polynomial.

The timing characteristics between the three are also unsurprisingly similar, due to the way the algorithm was designed means that each method has the exact same algorithm thingy. However, it was also interesting to note that simpsons rule converges poorly in relation to the midpoint and simpsons rule for the gaussian function, especially due to the large limits. It is possible that this is a case of Runge’s function,

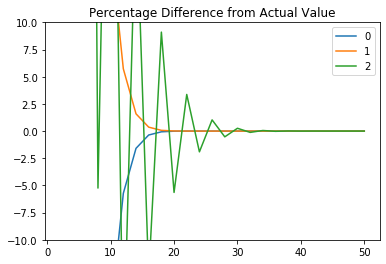
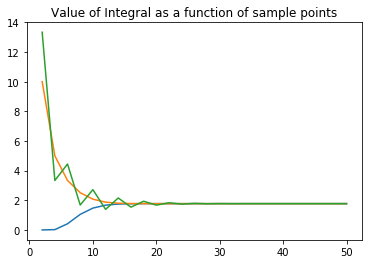
F1

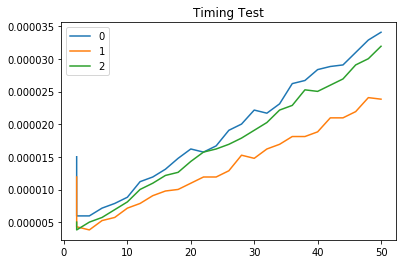


F3

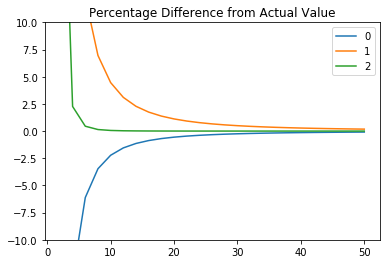


Gauss

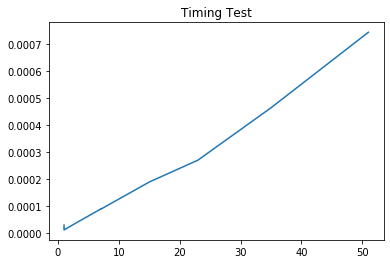
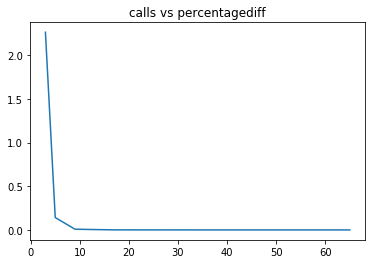
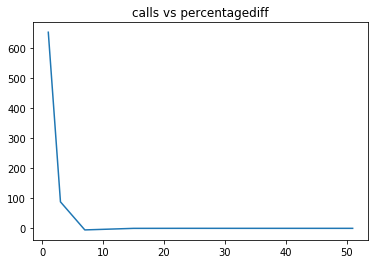




F5

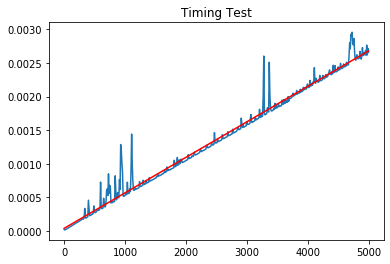
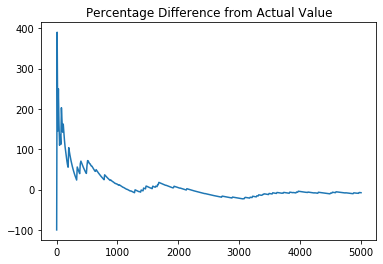


Adaptive integration was a massive improvement over the traditional NC quadratures. Both well behaved polynomials such as f5, as well as non-analytically integrable functions such as the gaussian, it converged to near 0 in less than 10 interations of the algorithm. Given that it uses the exact same NC Int algorithm, this program is extremely fast. However, the recurvise nature of it limits it speed, therefore taking approximately 10 times longer to run to achieve the same error. Overall, as a function of calls this is blazing fast, but as a function of time it is not as fast. However, it is expected in the long run that the less amount of calls it takes to run would win over the timing increaseements compared to others.

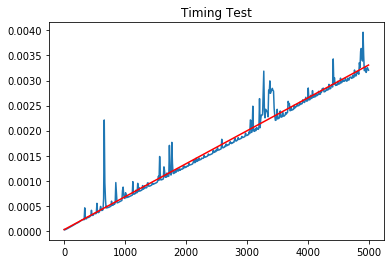
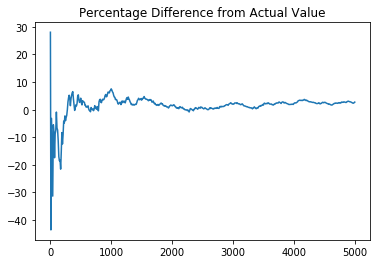


Monte Carlo

G



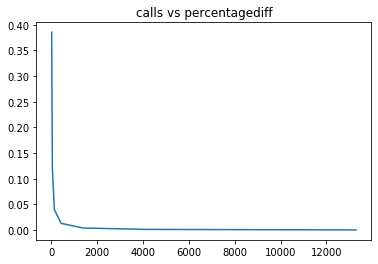
F5



Monte carlo simulations are slow to converge but fast to perform, 5000 function calls take only 0.004 seconds to do. Basically, on average each call takes roughly the same time as the NCInt counterpart. However, because monte carlo is based on randomness compared to NCInts deterministic properties, convergence is much slower. As seen there, the error of the thingy does indeed go down as 1/√N.

Recursive Stratified sampling

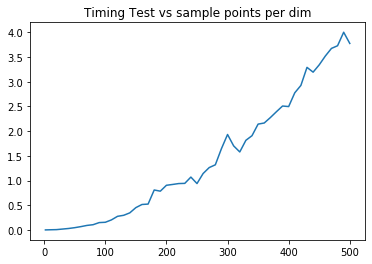
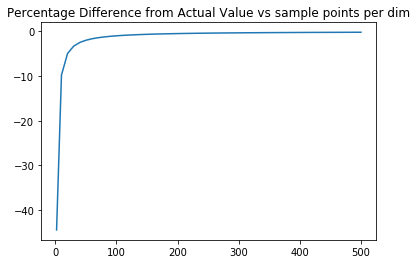
Stratified sampling enhances the crude monte carlo method and increases its convergence. As seen here, it only takes less than 2000 random numbers for it from that it takes less than 5 seconds to complete, meaning it is much slower than the thingy. However, even at very low calls, it is already at only 0.4% difference away from the actual value, making it very good.



Overall, in one dimension, deterministic quadrature rules are much better compared to monte carlo techniques. The adaptive NCInt converges extremely quickly, in only a few iterations of the algorithm. Meanwhile, crude monte carlo is slow due to the number of random number generations it needs. Stratified sampling is better, however it is still slow compared to that.

N Dimensions

NCInt



Because of the curse of dimensionality, the integration is not as good as in one dimension. As shown here, the integral converges very slowly, requiring more than 50 points per dimension for it to converge close enough to the actual value. 50 points per dimension means 2500 points for it to calculate, which is why it is extremely slow. For it to converge more, it would require additional points, and more points would slow down the algorithm. As seen in the timing test, there is a clear rise in the timing test as the number of points per dimension increases, a polynomial increase which is really bad.

On the other hand, both monte carlo and its stratified were still working well. Due to the uncomplicated algorithm of monte carlo, it can easily handle 2000 function calls without slowing down. Small amounts of points already lead it to a 5% error estimate, which is already really good, and further function calls make it a better error estimate.

Stratified sampling also works really well, going the closest to the hypersphere’s true value. It also only takes that superb accuracy, meaning that monte carlo is the best and so is stratified sampling

Conclusion

* Quadrature methods rule in one dimension
* Especially adaptive integration
* Monte carlo and fam wins in multiple dimension
* Especially stratified sampling, very accurate

Recommendations

* Make algorithms better, as there are some where its slow due to poor implementations
* Also this method was recursive stratified sampling, try to use another algorithm such as importance sampling similar to vegas
* Plot the graphs in one d and two d
* Implement the maximum iterations in the adaptive bit, done in strat

TODO:

* **Fix adaptive integration N**
* Clean up stuff for lab report V2
* Polish up code
* Generalize the plotmes
* Can do the RetAns for each thingy that’s fine done
* But the plotmes must be standardized progress
* What do I want in the plots?
* I want them in different plots
  + Value of Integral as function of sampled points
  + Percentage Difference from Actual Value
  + Timing tests as function of sampled points
* Good titles and legends
* X and y axis labelled
* I want the limiting factors in ncint (in each individual graph is faded)
* If possible also in monte carlo
* Also want the combined 0,1,2 ncint in one thing
* Get rid of the straight line in the graph, yg interpolate itu
* Adaptive Integration and Stratified Sampling should have their own plotmes
  + Calls vs percentage difference is important
* Combine the 1 dimension and N dimension plots
* Use the existence of the additional element in the RetAn list, use try except for N
* By the end of today have all the pretty plots ready for sticking in labs (insyaallah)
* Arrange new notebook similarly to your lab rerpot