Computing report

Introduction

MENTION MISER MONTE CARLO AND HOW ITS RECURSIVE STRATIFIED SAMPLING

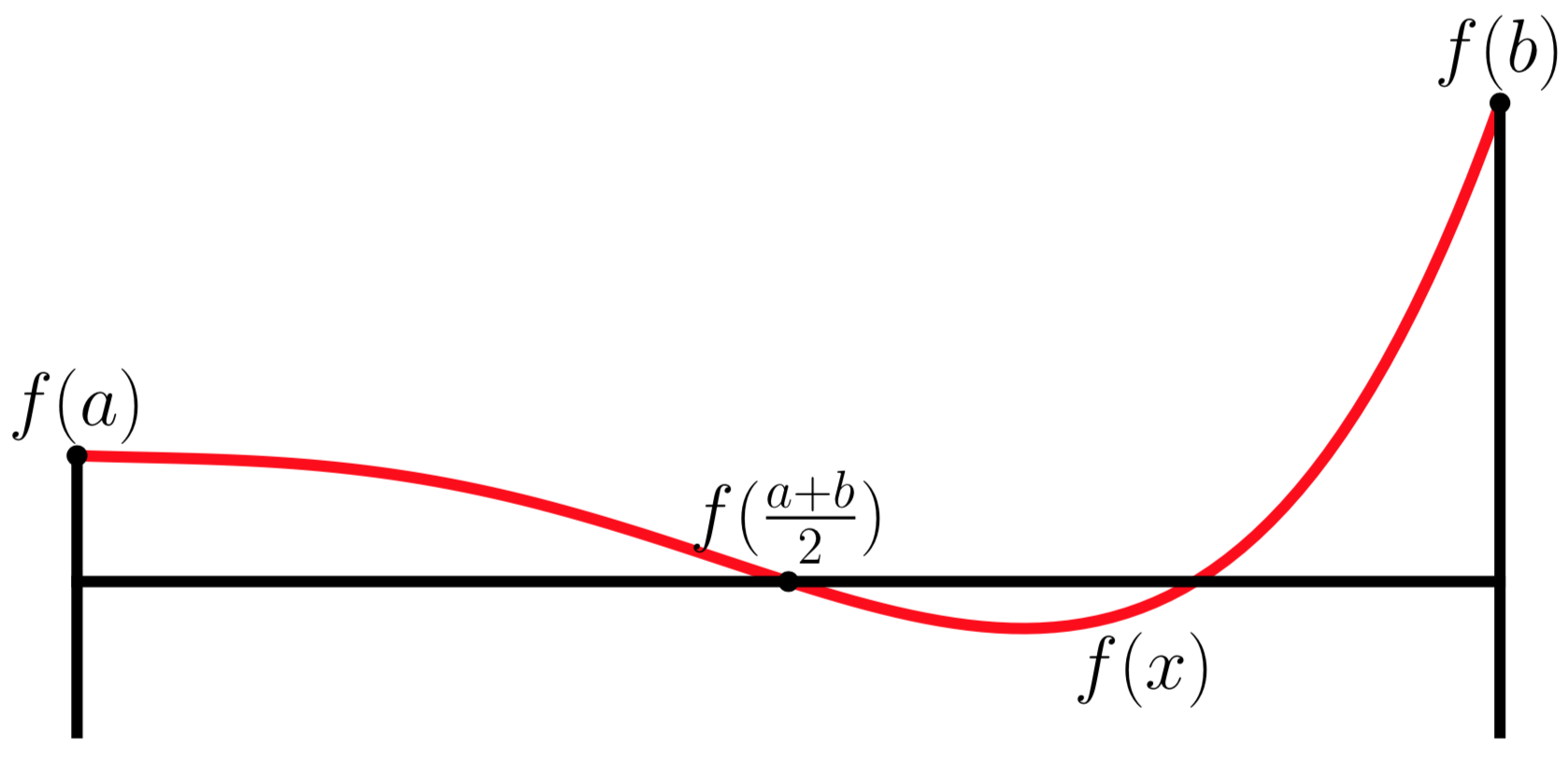
Determining the cross section of a particular interaction is of the upmost importance in several particle physics experiments, as the cross section is an indication of the probability of an interaction occurring. Calculating the cross section of a two-to-two interaction, such as electron scattering, 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

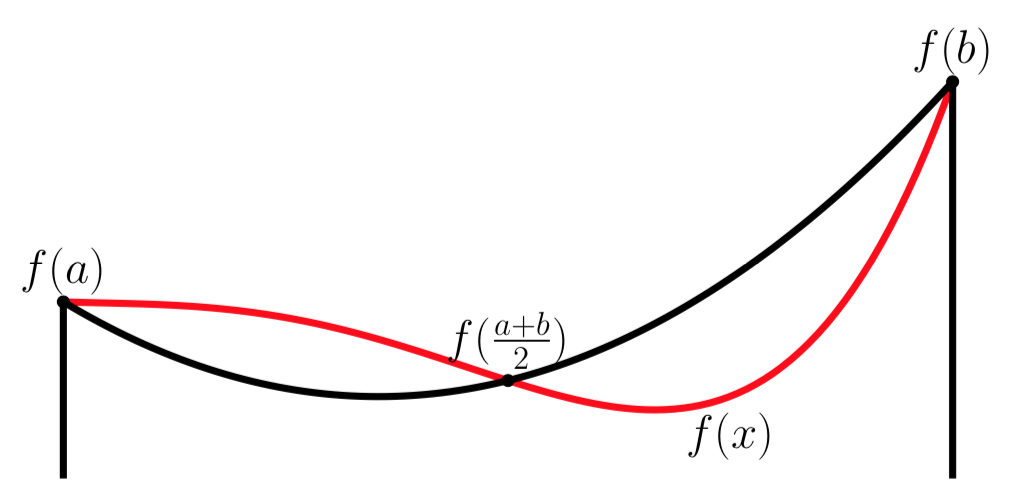


Instead of approximating the function as a 0th order polynomial, a 1st order polynomial could also be used. The trapezium rule is the NC quadrature rule that uses this approximation, illustrated in FIGURE SOMETHING. The mathematical form of the trapezium rule is

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

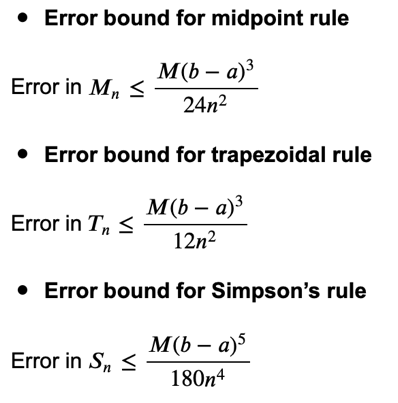


Higher order polynomial approximations for numerical integrations are available and would theoretically increase the accuracy, however are rarely used in particle physics due to the approximations possibly suffering from the Runge’s Phenomenon for certain functions. The approximation would vary wildly 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 the quadrature rule in each dimension throughout all sampling points. The sampling points are found from the cartesian product of the sampling points in each dimension. Summing over sampling points lead to the generalized mathematical equation

A close up of a logo

Description automatically generated

.

The two dimensional case is illustrated IN FIGURE SOMETHING, where the sampling points are the cartesian product P1xP2.

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 (SEE APPENDIX FOR DERIVATION OF THIS), 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

In order to perform the integration, the list of the indexes K were required. A generating function was thus was implemented in python to perform the cartesian product necessary for producing K. The index generated by this was combined with the two dimensional weights list, allowing the algorithm to loop through all sample points.

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

As expected, all NC quadrature rules converge quickly in well behaved polynomials F5F5F5F5. Simpson’s rule naturally converges faster than the others due to its higher order approximation. Both the trapezium rule and midpoint rule converge at the same rate, which was expected as shown in the previous discussion. There are certain cases in which Simpsons rule converges slower compared to the others. In FIGURE SOMETHING GGGGGGGGG, when integrating the gaussian function, Simpson’s rule was the last to converge. Analysing Simpsons rule further shows that this convergence is near but within the calculated limits in SECTION SOMETHING. The timing characteristics for all three quadrature methods were also extremely similar, as expected due to the way the algorithm was implemented. The midpoint rule seems to systematically perform slower than the others, however this was possibly due to the implementation of the algorithm. There were also some spikes in the timing test, but this was possibly due to the processor of the computer.

Adaptive integration was an improvement over traditional methods. Unlike traditional NC quadrature methods, adaptive integration converges quickly both in analytical and non-analytical functions. For the two tested functions, adaptive integration requires less than 20 iterations for it to converge to less than 1% error. However, due to the way it was implemented, adaptive integration is slower than standard NC quadrature, as seen FIGURE SOMETHING AND SOMETHING. The main advantage of this method is that it allows the user to set the error tolerance for the final value, which is a major advantage STICK THIS IN THE CODE BIT.

Monte Carlo Integration’s execution speeds are similar to NC quadrature methods, yet was much slower to converge. As discussed BEFORE PUT DOWN A SOURCE FOR THE √N THINGY, due to utilizing random numbers the value of the integration converges as O(1/√N). Convergence can be seen where the fluctuations in the value decreaasse as the number of sample points increases. From this, it can be concluded that the crude monte carlo method would never be as efficient as NC quadrature methods in one dimension.

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. Recurvive Stratified Sampling improves on monte Carlo and converges much faster than monte carlo. Each iteration of stratified sampling brings the value closer to the actual value, and is therefore much better than crude monte carlo. Due to the complexity of the algorithm, it does require a longer time to execute, however converges much faster and would therefore give a truer result. Comparing it with adaptive integration, the execution speed of adaptive integration is much faster compared to recurvise stratified sampling.

Overall, in one dimension, deterininstic quadrature rules perform better compared to monte carlo techniques. Not relying on random samples allow the quadrature rules to converge much faster by systematically choosing the best sampling points to use.

N Dimensions

- Converges Very slowly now

- Suffers from curse of dimensionality

- The time it takes increases polynomially (is that a word)

- Because there are much more points required

- Very bad, as adding an additional point increases it by a lot

Applying NC Quadrature rules in four dimensions lead to poor performance. The quadrature rule requires much more sampling points for it to converge, and each additional sampling point added to the function increases the execution speed exponentially. As shown in FIGURE SOMETHING, converging from an error of around -5% to an error of -4% requires using 50 sampling points per dimension instead of 40, which contributes to a total of 6.25 million sampling points required. This is due to the curse of dimensionality, where the number of points required for it to converge depends on the dimensions of the integral itself, as discussed in the previous section. Extrapolating this to hundreds of dimensions, NC Quadrature would perform extremely poorly due to the aforementioned characteristics. SOMETHING ABOUT ADAPTIVE INTEGRATION

Monte Carlo’s characteristics remain the same in 4 dimensions. The convergence rate remains of the order O(1/√N), which is now favourable over quadrature methods. A few thousand function calls bring the error of the evaluation to under 2.5%, and will converge even more as the number of sampling points increases. Given that the characteristics of monte carlo dont change when the dimensions increase, monte carlo based methods are the preferred methods when performing multi dimensional integrals.

The multidimensional form of recursive stratified sampling has much faster convergence compared to the crude form of monte carlo. A few iterations of the algorithm already returns a value with an error of LESS THAN X PERCENTAGE. It was noted that it does take a long time to run with more iterations due to the number of divisions it needs to do, however is still much faster than NC quadrature methods. It seems to suffer from the dimensional curse as well, however it seems that the effects are not as large as the NC quadratrues. The stratified sampling algorithm can be improved, as the stratified sampling algorithm here subdivides the bins in all dimension instead of only in one dimension. This unneededly increases the complexity of the calculation, and leads to the larger time use.

Conclusion

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.

In conclusion, the project was a success. The main objectives of implementing Newton-Cotes quadrature methods and Monte Carlo methods in Python were completed, and were successfully characterized using multiple functions. It was shown that Newton-Cotes quadrature methods, especially Simpson’s rule and Adaptive Integration, had superior performance compared to Monte Carlo methods in one dimensional due to its fast execution speeds and convergence. However, in multiple dimensions, Monte Carlo based methods performed better due to it having consistent characteristics regardless of the dimensionality. Based on these findings, Monte Carlo integration methods are recommended for use in particle physics.

Recursive stratified sampling was chosen over other variance techniques due to its simple concept and implementation, and that it was Monte Carlo’s implementation of adaptive integration. Due to time restrictions, only simple variance reduction techniques such as this one could be implemented in time

However, there were still factors in the project that could be improved. A maximum iteration condition was not implemented in the one dimensional adaptive integration algorithm, as there was a lack of time due to more work in polishing the stratified sampling code. Furthermore, the adaptive integration for multiple dimensions was not working properly, and could not be fixed for the project due to focus on monte carlo methods in multiple dimensions. Furthermore, the stratified sampling algorithm in multiple dimensions could be improved as it was highly inefficient. Proper variance reduction algorithms such as VEGAS and MISER adapt their grid spacings smartly, by subdividing into dimensions where the benefit would be larger. This has not been implemented yet, but is an expansion for future work.

Recommendations for this project would be to perform a comparison study between multiple variance reduction techniques in monte carlo. As this project has established, monte carlo methods performs best in multiple dimensions, especially with variance reduction techniques. Recommended methods to study are the Recursive Stratified Sampling Methods, Importance Sampling, SOMETHING Hypercubes, and others.

TODO:

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