Contents

1 Exercise 1. Integration.

1.1 Structure of the solution.

The solution is split into two files, according to the theme of the exercise. The file `monte_carlo_integrator.py` contains everything related to monte carlo integration, including the answers to core task 1 and supplementary task 1, and vice versa.

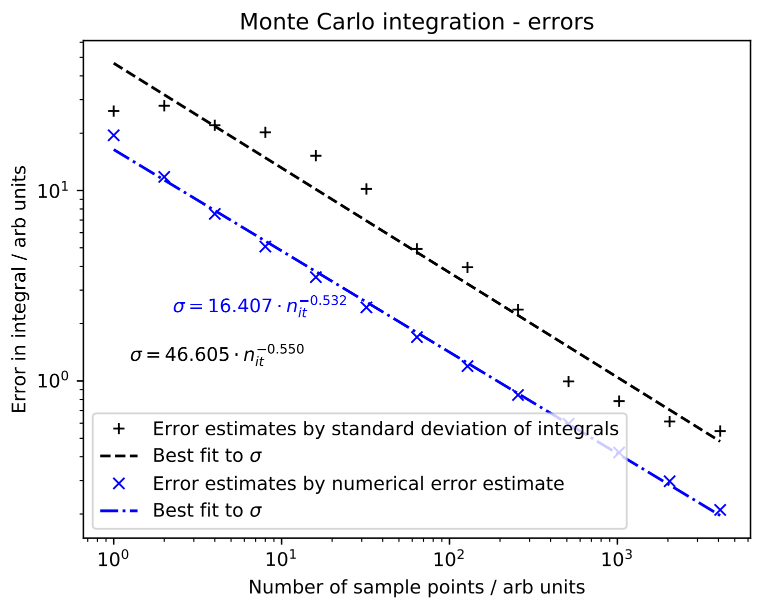
The scripts, contain a main function so can be compiled if need be, and automatically generate all of the plots. Data is not being saved to disk by default, even though it can be done using the provided `data_dump()` function.

The plots generated by both scripts can be found in the subdirectory aptly named `'figures/'`.

Monte Carlo integration.

1.1.1 Results:

The plot aptly named `Mc-error-bar-plots.pdf`



shows the dependence of the estimated error on the number of iterations of the algorithm.

The errors were estimated in two ways:

- The black series, is done via running the algorithm 25 (by default) times, and taking the standard deviation of the estimates.
- The Blue series, was estimated by using the error estimate result given in the booklet.

This result is averaged over the aforementioned 25 iterations, and reduced by a factor of

$$\sqrt{24}$$

(because we've used the data to find the average estimate), as we would expect from theory to reduce the error in a process as the standard error in the mean.

1.1.2 Discussion

First, note that the plot is on a logarithmic scale, thus implying a power-law dependence for both error estimates.

Although the exact dependence of the error on the number of iterations varies from run to run, the linear regression analysis shows that the exponent of the power law dependence is $0.5 \pm 5\%$, as we expect from theory. (Core task 1)

We can also compare the theoretical error estimate to the one obtained from Monte-Carlo integration. As expected, the theoretical error is smaller, by a constant factor.

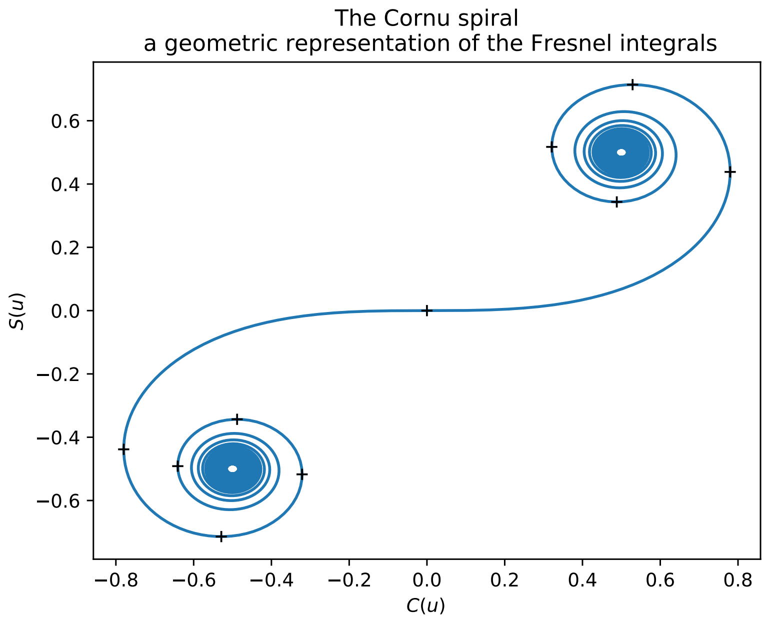
More interestingly, the theoretical error estimate seems to be more consistent with a power law dependence as well, which we might attribute to this being a more accurate estimate of the error in numerical integration.

The plot Mc-values.pdf, shows how the integral estimates converge to the expected theoretical result. Here we can compare the estimated errors to the deviation from the known analytical result, and note that for a small number of iterations, e.g. 2500, the error estimates are an order of magnitude less than the actual discrepancy.

After such a demonstration one would be hesitant to think that these errors are representative of the deviation from the true value for a small sample size.

1.2 Fresnel integrals.

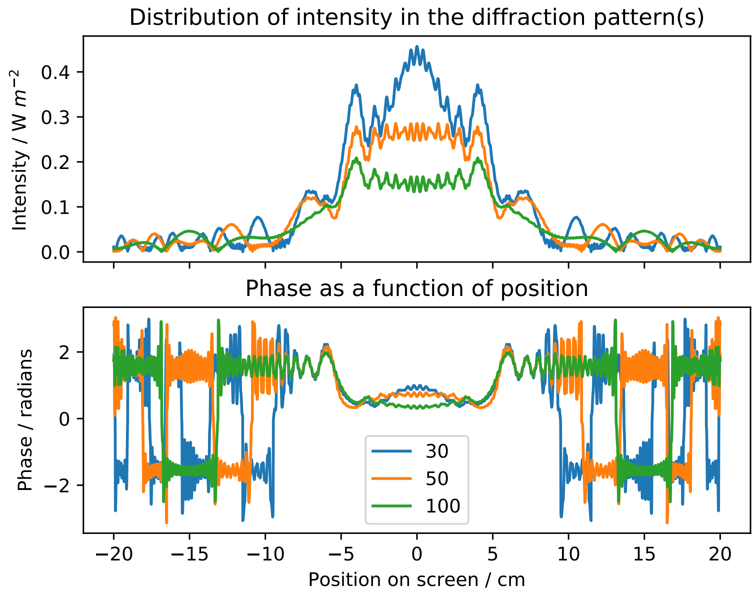
The plot `Cornu-spiral-plot.pdf`



, shows a nice picture of a particular case of an Euler spiral, that has special significance in Fresnel diffraction theory.

Tick marks indicate the positions where the integration variable is equal to a square root of an integer, and act as visual guides for performing numerical integration "by eye". As on plots from other sources, these occur at the points tangent to the spirals.

The plot `Fresnel-intensity.pdf`



, models the diffraction pattern generated using a slit of fixed width at different distances from the aperture.

The intensity distribution is as predicted by model plots, with a characteristic serration in the middle, and non-maximal intensity where the slit is supposed to be. Since the scaling was implemented both for the distribution of the intensity but also for the actual value thereof, the plots for different distances don't coincide.

On the subfigure showing the phase dependence, one can see model behaviour while within the bounds of the aperture. However, once reaching the boundary, the phase ceases to be well defined. Even by using the cmath atan2 function which computes the angle without overwrapping, some chaotic behaviour can be seen.

1.3 Notes on optimisations.

Wherever possible, vectorised application of functions was used, however in most cases proper vectorisation of used functions was not performed, for the following reasons.

The real gain in performance is observed if vectorising is done by writing a compiled c function that operates on python objects. While that can yield a significant performance gain, when the bottleneck is a specific mathematical operation that can be optimised on the hardware level, this is a significant time investment for a modest performance boost. Not to mention that the there exist optimising compilers for python that can do that automatically.

Instead I have chosen to parallelise the task between different CPU cores, using the multiprocessing library. This approach chunks the data for independent operation of CPU threads, and also separates the chunks in RAM, so as to minimise the interference of one task on another. This approach is better than multi-threading, because then the calculations can be spread across not only different cores of the same CPU, but also different computers. The additional memory overhead is negligible, and in my opinion well worth the improvement.

Also, the Fresnel integrals have produced several warnings for the integrator in quadrature, saying that the increase in precision doesn't yield a significant enough change in the value, indicating that some of the integrals may not have converged to the actual value. However, since the error message is a message and not an exception, it makes pinpointing such difficulties and removing them nigh impossible.