Integration and Random Numbers

Takis Angelides

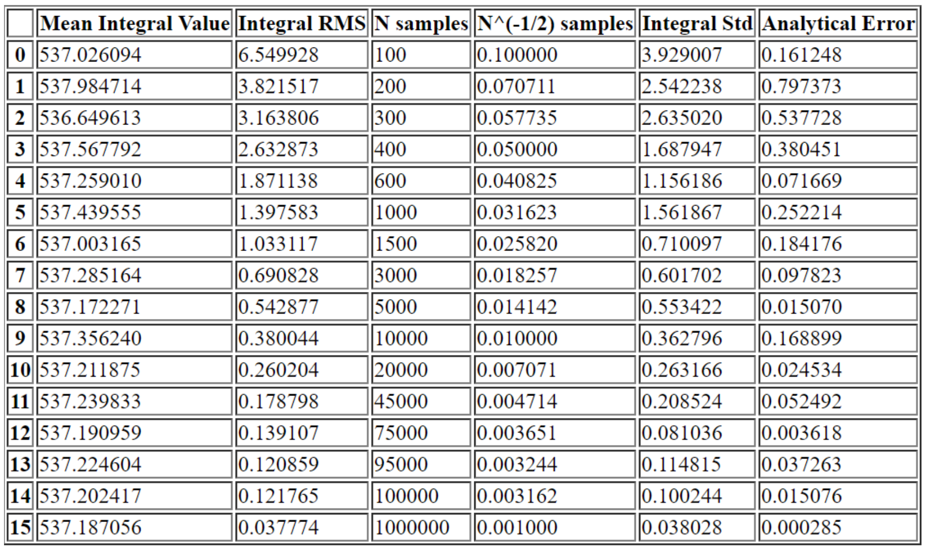
Janurary 2020

1 Core Task 1: Monte Carlo integration (monte_carlo_integration.py)

This program generates uniformly distributed random numbers in the form of a list for vectorised implementation and passes it as the argument to the integrand of the integral to be evaluated. The integral is evaluated using N samples and for each N value, 25 calls are made to the Monte Carlo integrator so that a mean value is found, along with a standard deviation of those 25 values and a root mean square error. Using the analytical answer provided, the program also calculates an analytical error for each N value. All the latter data are summarised using a pandas dataframe that is converted to a table in Figure 1.

The program and all subsequent programs respect the single responsibility principle that makes the code more readable and easy to debug. Graphs of log(RMS) and log(std) vs log($N^{-1/2}$) are also generated to test the theoretical statement that the error is proportional to $N^{-1/2}$. These graphs are shown in Figures 2 and 3. A straight line is also fitted to guide the eye and indeed these graphs support the theoretical statement.

The mean integral value is also plotted against log(N) with error bars from the RMS. In Figure 4 the analytical answer is also included to guide the eye to the process's convergence. Finally, plots of RMS and standard deviation vs N are also shown in Figures 5 and 6, along with the analytical error. The first 7 points are not included in the plot due to their high error which hinders clarity. The best estimate for the integral is 537.19 $\pm$ 0.03 using N = 10^6.



	Mean Integral Value	Integral RMS	N samples	N^(-1/2) samples	Integral Std	Analytical Error
0	537.026094	6.549928	100	0.100000	3.929007	0.161248
1	537.984714	3.821517	200	0.070711	2.542238	0.797373
2	536.649613	3.163806	300	0.057735	2.635020	0.537728
3	537.567792	2.632873	400	0.050000	1.687947	0.380451
4	537.259010	1.871138	600	0.040825	1.156186	0.071669
5	537.439555	1.397583	1000	0.031623	1.561867	0.252214
6	537.003165	1.033117	1500	0.025820	0.710097	0.184176
7	537.285164	0.690828	3000	0.018257	0.601702	0.097823
8	537.172271	0.542877	5000	0.014142	0.553422	0.015070
9	537.356240	0.380044	10000	0.010000	0.362796	0.168899
10	537.211875	0.260204	20000	0.007071	0.263166	0.024534
11	537.239833	0.178798	45000	0.004714	0.208524	0.052492
12	537.190959	0.139107	75000	0.003651	0.081036	0.003618
13	537.224604	0.120859	95000	0.003244	0.114815	0.037263
14	537.202417	0.121765	100000	0.003162	0.100244	0.015076
15	537.187056	0.037774	1000000	0.001000	0.038028	0.000285

Figure 1: Data from the Monte Carlo intagration summarised in a dataframe table. The table includes 15 different values of N samples tested.

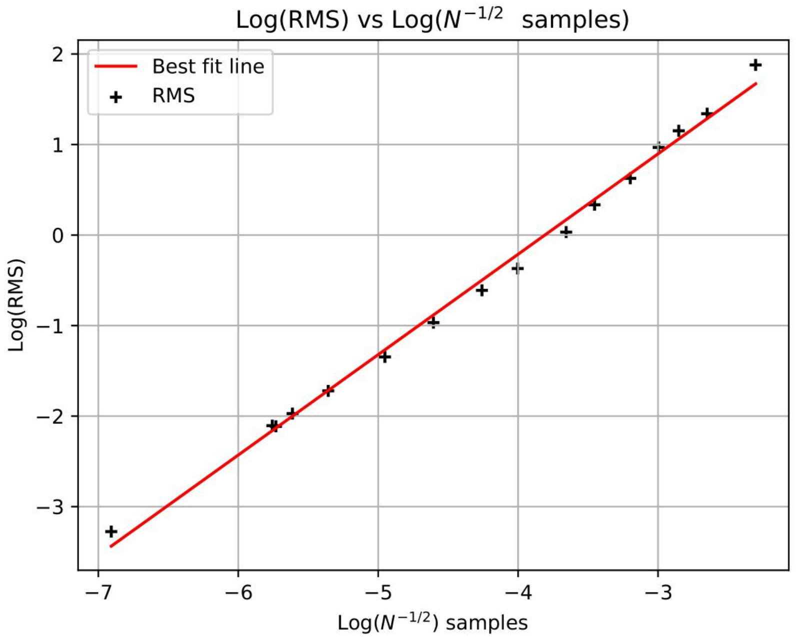


Figure 2: Log(Root mean square error) vs Log($N^{-1/2}$ samples) with a line of best fit which supports the theoretical prediction that the error is proportional to $N^{-1/2}$.

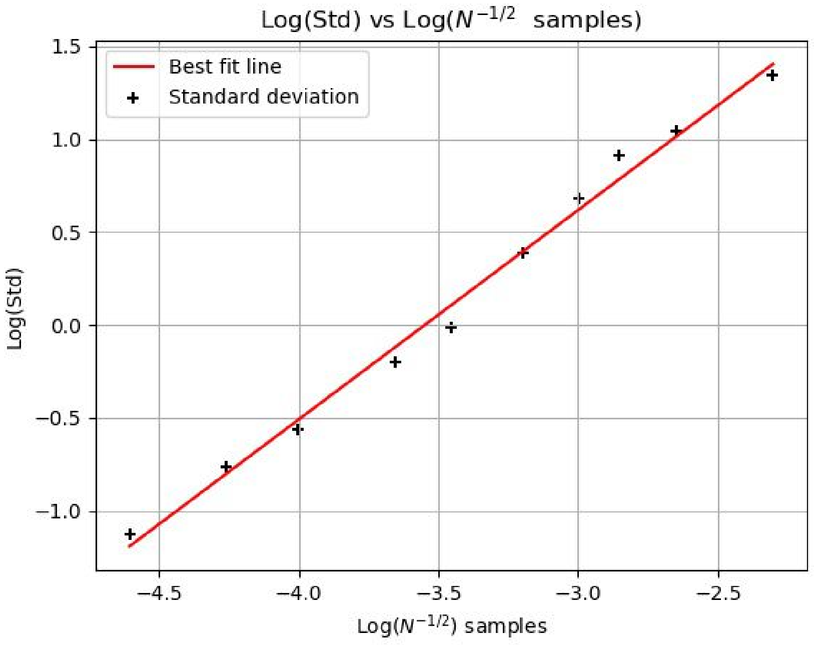


Figure 3: Log(standard deviation) vs Log($N^{-1/2}$ samples) with a line of best fit which supports the theoretical prediction that the error is proportional to $N^{-1/2}$, even for the standard deviation from the 25 calls to the MC integrator.

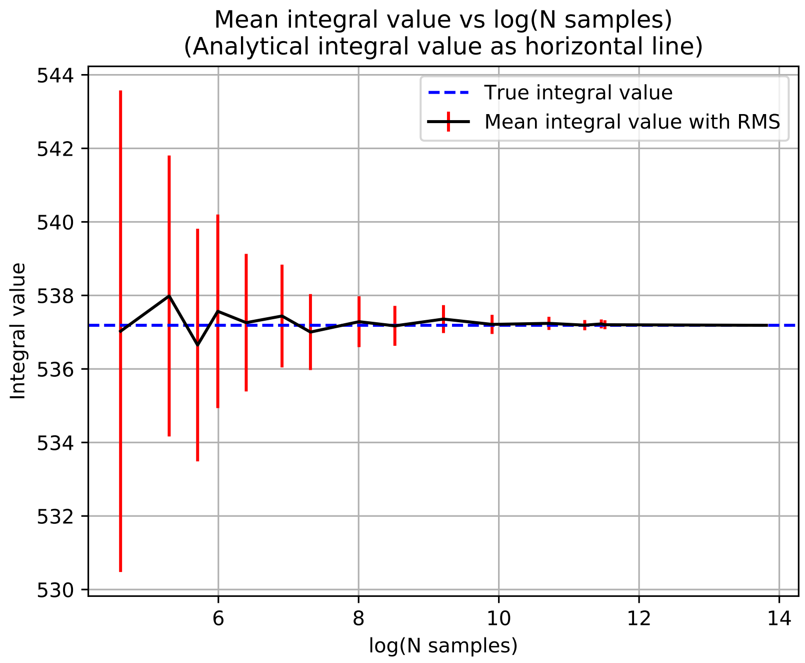


Figure 4: Mean integral value vs log(N samples) with error bars from the RMS. Satisfactory convergence is achieved for N > 1000. The blue dotted line represents the analytical answer provided with value 537.187.

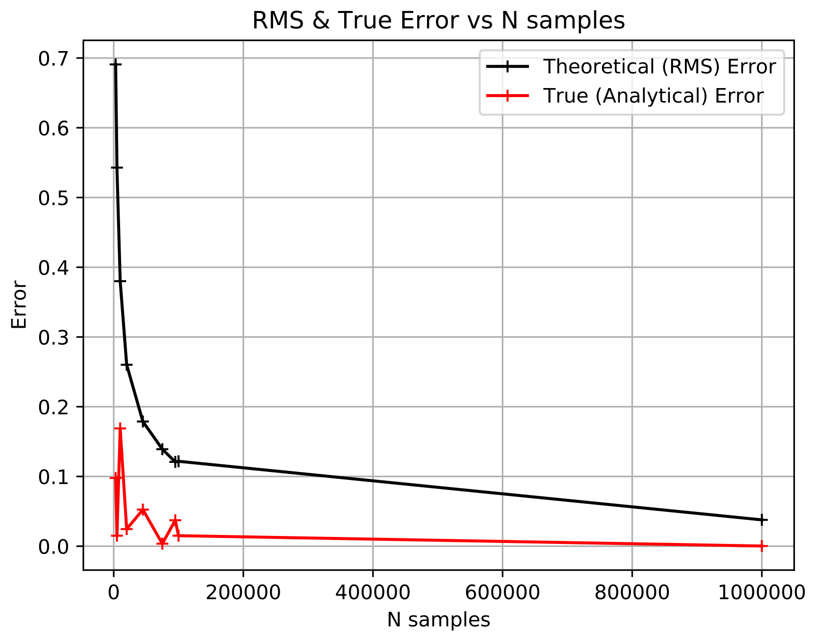


Figure 5: RMS vs N samples compared to the analytical error.

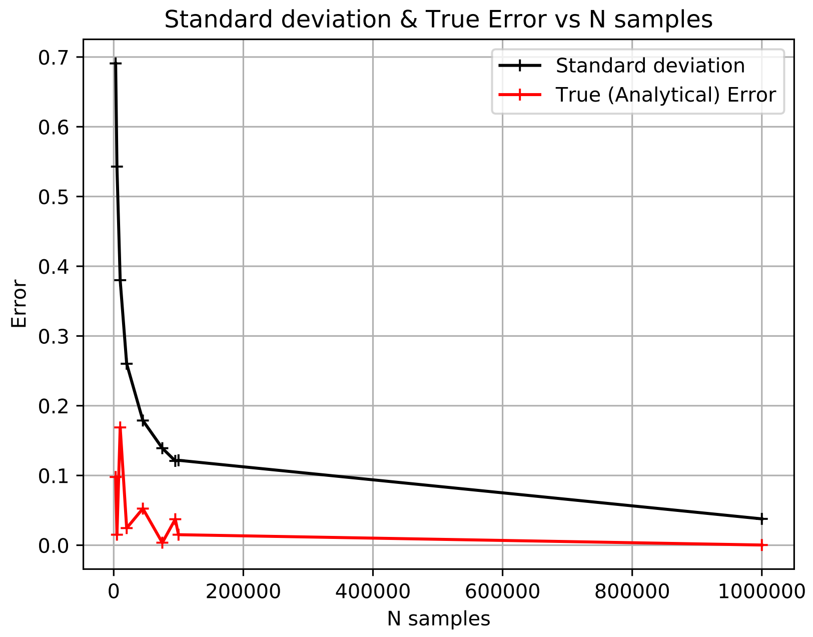


Figure 6: Standard deviation vs N samples compared to the analytical error.

2 Supplementary Task 1: Errors in Monte Carlo integration (monte_carlo_integration.py)

The code for this task can be found towards the end of the code for the core task 1. A bar chart is produced to compare the RMS values given by equation (3) to the standard deviation calculated for each N value tested. The standard deviation is calculated using np.std() on the array of 25 values generated for each N. The latter calculation implicitly assumes that the errors coming from each MC integration are Gaussian distributed. The RMS is calculated using RMS $=\sqrt{\frac{\sum_{n=1}^{25}\sigma_n}{25}}$, where σ_n is the error from equation (3). It is evident from Figure 7 that the standard deviation is consistently higher than the RMS values.

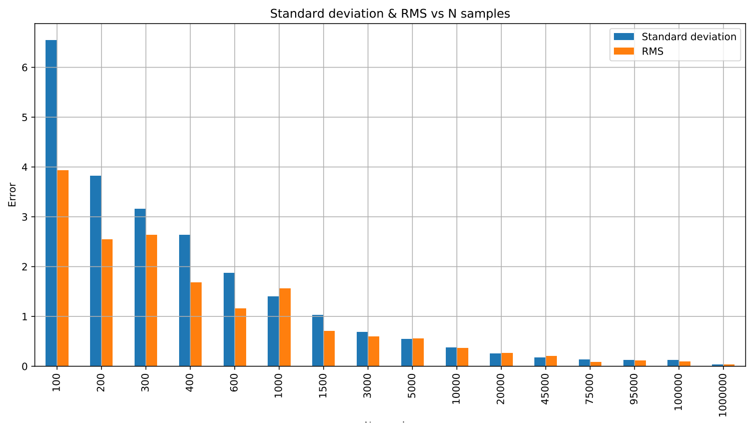


Figure 7: Bar chart of error vs N samples comparing the standard deviation errors to the theoretical RMS errors derived from equation (3). It is observed that RMS is consistently lower than the standard deviation.

3 Core Task 2: Fresnel integrals (fresnel_integrals.py)

This program calculates the Fresnel integrals using scipy's quad() and then plots the cornu spiral shown in Figure 8. It also creates a pandas dataframe and prints from that the first and last 5 points of the spiral's locus presented by Figures 9 and 10. The circular part of the spiral is also shown in Figure 11 to reveal the smoothness of the calculation. It was observed by extending the lower and upper limits of the integration that the circular parts of the spiral become denser.

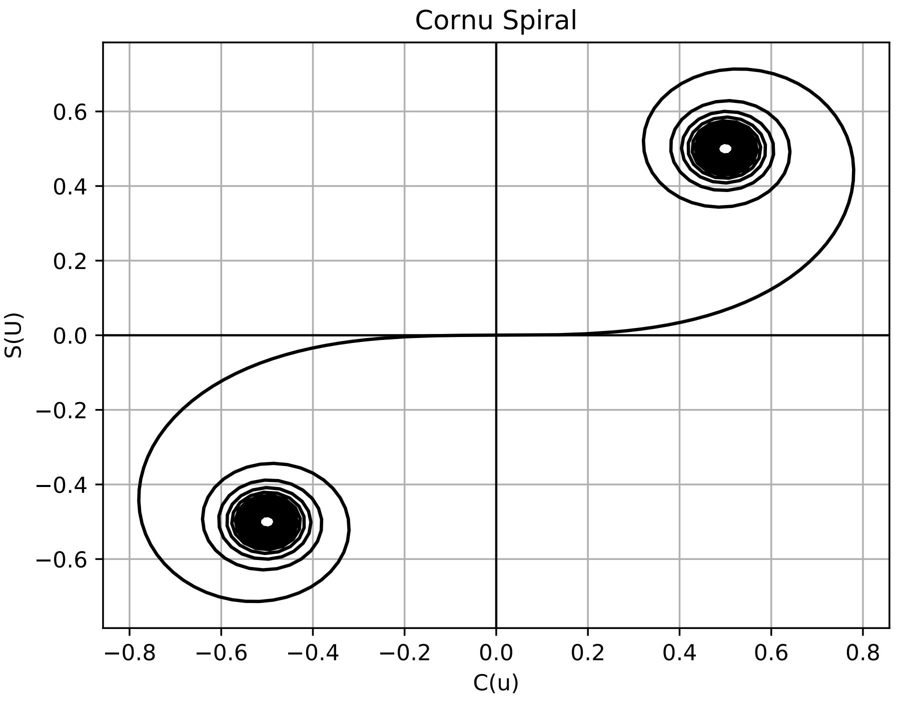
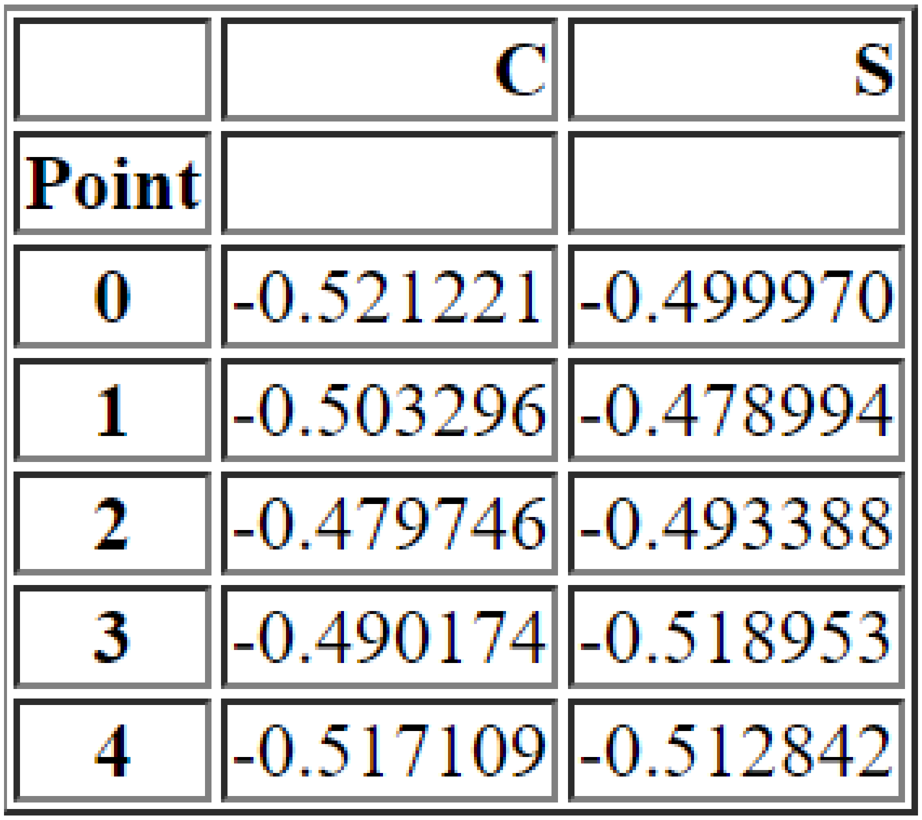
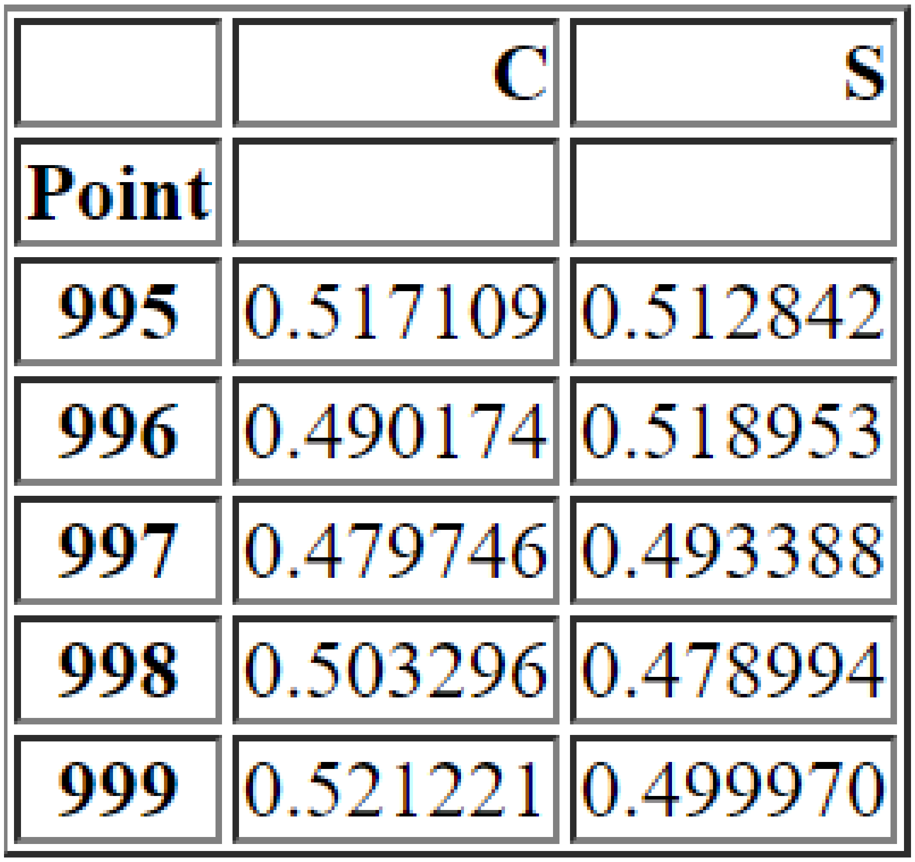


Figure 8: The Cornu spiral in the complex plane. Interestingly the spiral can also be obtained using a kitchen knife and an orange in interesting mapping applications as described in [1].



	C	S
Point		
0	-0.521221	-0.499970
1	-0.503296	-0.478994
2	-0.479746	-0.493388
3	-0.490174	-0.518953
4	-0.517109	-0.512842

Figure 9: The first 5 values evaluated for C and S of the spiral in the complex plane



	C	S
Point		
995	0.517109	0.512842
996	0.490174	0.518953
997	0.479746	0.493388
998	0.503296	0.478994
999	0.521221	0.499970

Figure 10: The last 5 values evaluated for C and S of the spiral in the complex plane

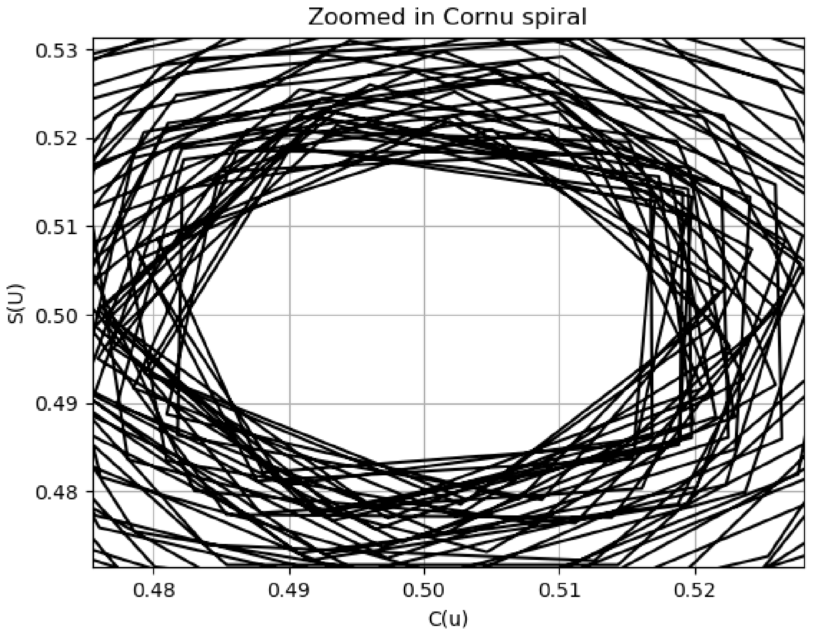


Figure 11: The circular part of the spiral, showing the degree of smoothness of the calculation.

4 Supplementary Task 2: Slit diffraction pattern (slit_fresnel_diffraction.py)

A list of points are created to evaluate the diffraction pattern on the screen caused by a slit in the Fresnel regime. The quad function is used to evaluate the Fresnel integrals and a dictionary is created for the 3 different values of D which is the aperture to screen distance. For each D key value, an array of arrays is stored and each nested array holds a tuple of position, magnitude and phase for each point in the initial list. Multiple plots of magnitude and phase - with different slit width and longer screen range - are then produced and are presented below along with their explanation in the figure captions. The relevant graphs are in Figures 12, 13, 14, 15, 16, 17.

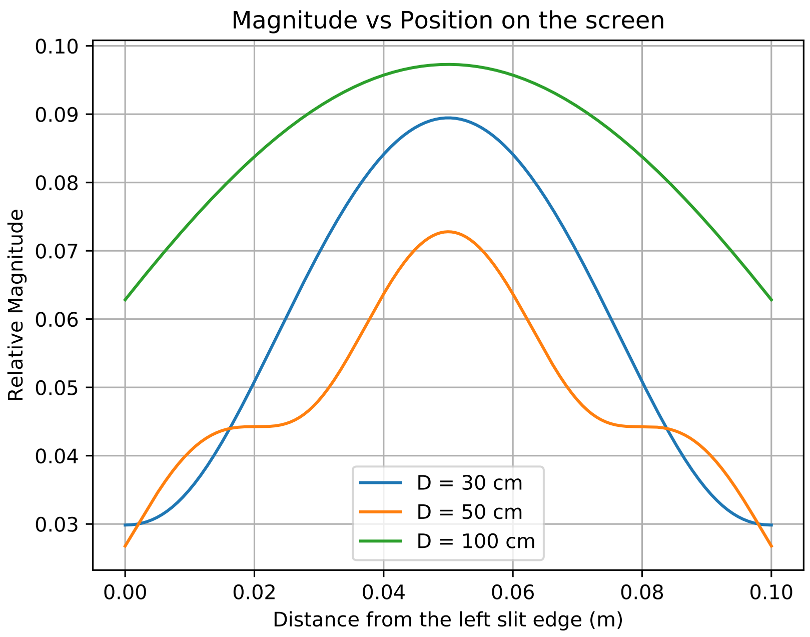


Figure 12: Magnitude of the diffraction pattern vs position from the left edge of the slit, with slit width d = 10 cm and various aperture-to-screen distances D in cm.

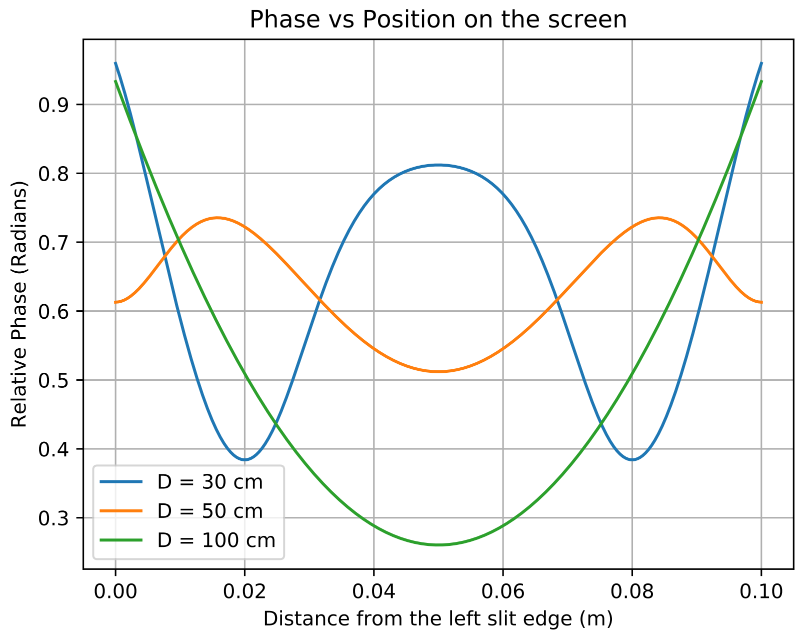


Figure 13: Phase of the diffraction pattern vs position from the left edge of the slit, with slit width d = 10 cm and various aperture-to-screen distances D in cm.

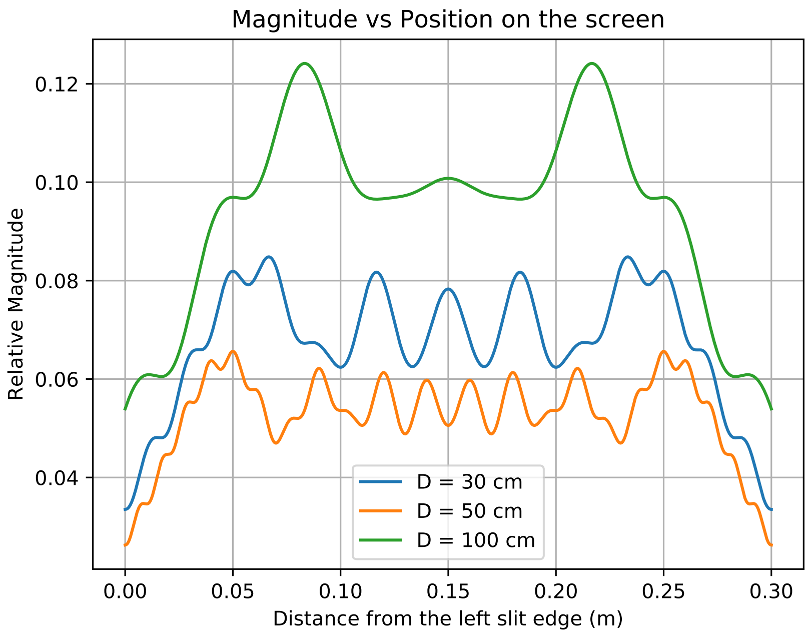


Figure 14: Magnitude of the diffraction pattern vs position from the left edge of the slit, with slit width d = 30 cm and multiple D values.

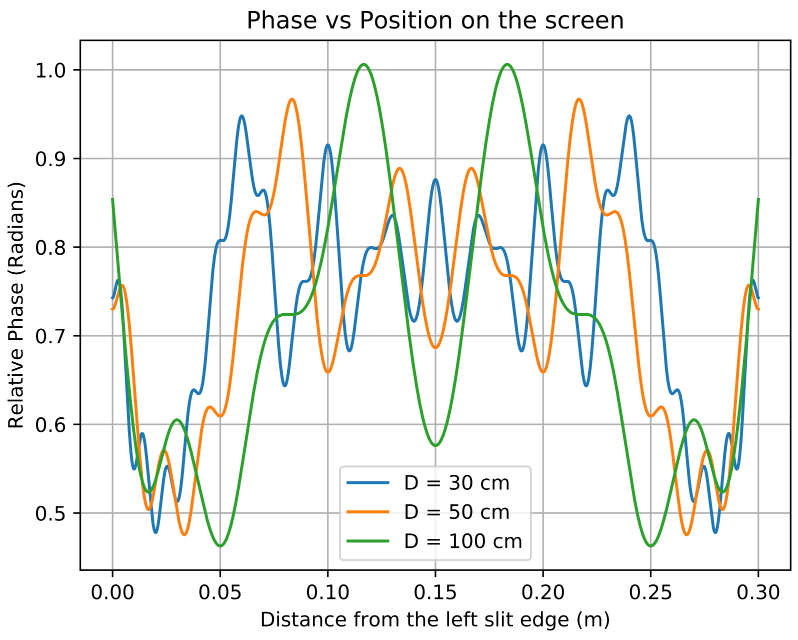


Figure 15: Phase of the diffraction pattern vs position from the left edge of the slit, with slit width d = 30 cm and multiple D values.

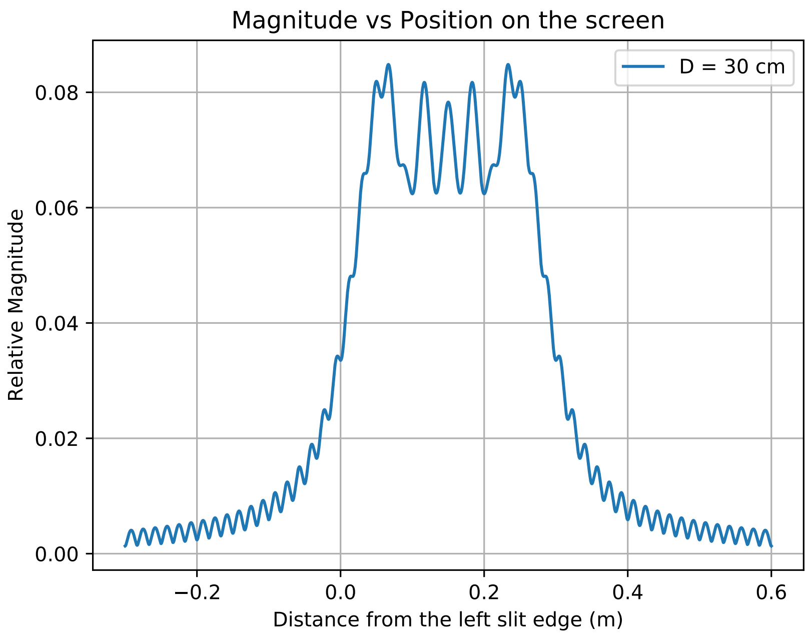


Figure 16: Magnitude of the diffraction pattern vs position from the left edge of the slit, with slit width d = 30 cm and D = 30. The positions now include a longer range on the screen.

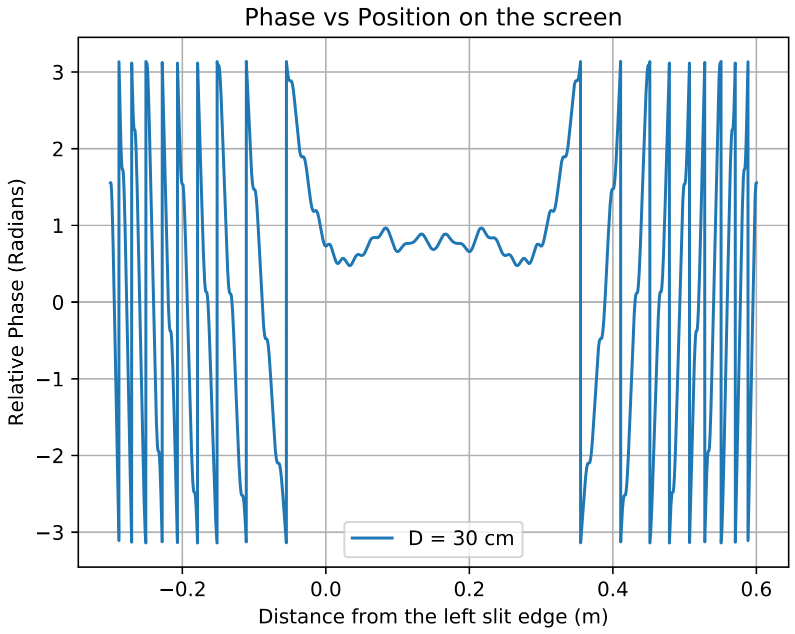


Figure 17: Phase of the diffraction pattern vs position from the left edge of the slit, with slit width d = 30 cm and D = 30. The positions now include a longer range on the screen.

References

[1] Andre G. Henriques Laurent Bartholdi. Orange peels and fresnel integrals. 2012.