PHYS30009: Intro to Computational Physics Exercise 3

The deadline for this exercise is **Monday 13th February 2023** at 12:30 p.m. Your report and program (*.py) files should be uploaded into Blackboard at the appropriate point in the *PHYS_20032_30009_2022_TB-4: Introduction to Computational Physics (PHYS20032 + PHYS30009) 2022* course. *S. Hanna*

Objectives of the exercise

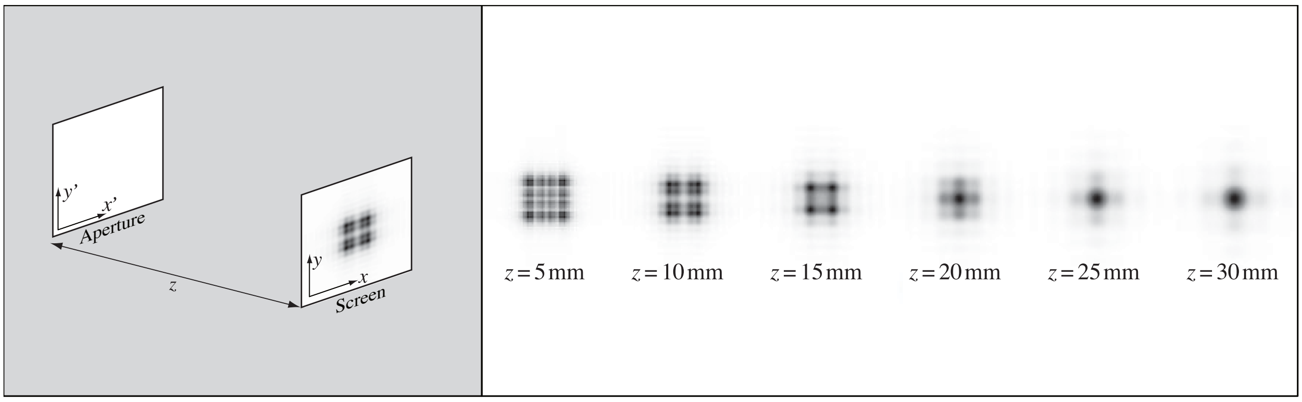
- To experiment with Simpson's rule for numerical integration;
- To get experience in the use of SciPy functions;
- To explore diffraction in the Fresnel and Fraunhofer limits;
- To explore the use of Monte Carlo methods for numerical integration;
- To gain more experience of plotting data.

Notes:

- For this exercise, you will be required to submit a program **and** a short report describing your results.
- We expect that you will need help as you develop your programs. Please consult the demonstrators as frequently as you need during the drop-in sessions. They are there to help.

Problem 1 (60% of marks): Fresnel diffraction from an aperture – Numerical integration using SciPy

In the computing lectures, you have been briefed about some basic types of numerical integration. Here we will apply numerical integration to an interesting diffraction problem. The general set-up for the diffraction experiment is shown below. Incident waves, travelling parallel to the z-axis, are diffracted by an aperture, and the scattering pattern observed on a screen. In the Fresnel approximation, the screen is near compared with the size of the aperture, and the diffraction pattern varies with the separation, z. Examples of this "near-field" diffraction are shown in the figure for a square aperture.



A general formula for Fresnel diffraction is given by:

$$E(x, y, z) = \frac{e^{ikz}}{i\lambda z} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} E(x', y') \exp\left\{ \frac{ik}{2z} \left[(x - x')^2 + (y - y')^2 \right] \right\} \mathrm{d}x' \, \mathrm{d}y' \tag{1}$$

where the origin ($z = 0$) is taken at the aperture and $k = 2\pi/\lambda$. The integral is performed over the $x' - y'$ plane of the aperture; the electric field of the incident light is taken to be zero everywhere *except* in the aperture, where it is a constant, i.e.:

$$E(x', y') = \begin{cases} E_0 & x', y' \text{ in aperture} \\ 0 & \text{otherwise} \end{cases}$$

This is most simply achieved by choosing the limits of integration to fit around the aperture and ignoring the constant amplitude phase factor of e^{ikz}/i:

$$E(x,y,z) = \frac{kE_0}{2\pi z} \iint_{Aperture} \exp\left\{\frac{ik}{2z}\left[(x-x')^2 + (y-y')^2\right]\right\} \mathrm{d}x'\,\mathrm{d}y' \tag{2}$$

$E(x,y,z)$ is interpreted as the electric field of the diffracted light at coordinates (x,y) on a screen distance z from the aperture. The observed diffraction intensity will be proportional to $|E^2|$:

$$I(x,y,z) = \epsilon_0 c E(x,y,z) E^*(x,y,z) \tag{3}$$

where E^* is the complex conjugate of E, c is the speed of light and ϵ_0 is the permittivity of free space.

We need to choose a suitable integration method, and we said in lectures that SciPy has many options available. The Simpson's method is often considered a workhorse method, which is generally useful. The implementation in SciPy is easy to use but requires a full set of sample points to be calculated in advance. The general quadrature approach, in which the name of a function is passed to the integrator together with the limits, is more flexible. Unfortunately the library of functions available in `scipy.integrate` derives from an old Fortran 77 library, which was not designed to take complex data types. (This is strange, because Fortran 77 definitely can handle complex numbers). Since neither approach is ideal, we will try them both!

We will explore the solution to Eq. (2) in stages. First we will solve the 1-d diffraction problem, using both `scipy.integrate.simpson()` and `scipy.integrate.quad()`. Then we will solve the full 2-d diffraction problem using `scipy.integrate.dblquad()`, which will allow us to calculate solutions for different shapes of aperture.

N.B. In carrying out the following stages, you should write a single Python program, based around a similar menu to that used in previous exercises. Write a separate piece of code for each section that requires it, making sensible use of functions.

a) **1-d diffraction:** We are going to evaluate the 1-dimensional Fresnel integral. The integral needed is shown in Eq. (4) both in its complex exponential form, and written out as real and imaginary parts.

Start by writing three functions to evaluate the kernels (inside parts) of the three integrals:

$$E(x,z) = \frac{kE_0}{2\pi z}\int_{x_1'}^{x_2'} \exp\left[\frac{ik}{2z}(x-x')^2\right]\mathrm{d}x' = \frac{kE_0}{2\pi z}\left\{\int_{x_1'}^{x_2'}\cos\left[\frac{k}{2z}(x-x')^2\right]\mathrm{d}x' + i\int_{x_1'}^{x_2'}\sin\left[\frac{k}{2z}(x-x')^2\right]\mathrm{d}x'\right\} \tag{4}$$

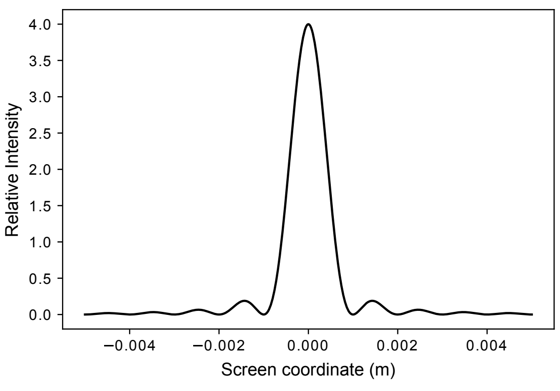
Each function should take the following arguments:

- the coordinate of a point in the aperture, x'
- the coordinate of a point on the screen, x
- the wavenumber, k
- the distance to the screen, z

Name	Variable	Value(s)
wavelength	λ	1×10^{-6} m
aperture width	$x_2' - x_1'$	2×10^{-5} m
screen distance	z	2 cm
screen coordinate	x	-5 to $+5$ mm
number of data points (Simpson only)	N	101

To create a diffraction pattern, you will need to consider each point on the screen, x, in turn. For each value of x (use a loop) you will need to perform the integration over all of the x' values. To do this you must do the following:

- For Simpson's method, create an array of x' values and another array containing the complex values of the kernel of the integral in the LH part of Eq. (4). Pass these arrays to `scipy.integrate.simps()`.
- For the quadrature method, separately evaluate the real and imaginary integrals in the RH part of Eq. (4), by passing the names of your functions and the values of x_1' and x_2' to `scipy.integrate.quad()`.



Test your program by plotting the intensity against the screen coordinate x. To observe far-field diffraction for a single slit, as shown in the figure on the right, use the parameters listed in the table. Remember, for the Simpson method you need an odd number of points, N, which implies an even number of intervals.

HINT: To generate the plot, you will need to create two arrays, one for the $|E(x)|^2$ values and the other for the x values. Plot your result using matplotlib as in the previous exercise.

b) When you are happy that both of your 1-d integration methods are working correctly (they should give identical results with the parameters suggested), explore the effect of changing the aperture width (x'_1 and x'_2) and the screen distance, z, on the appearance of the plot. Describe and discuss this in your report. Near-field effects will be observed by increasing the size of the aperture and/or reducing z. You may need to change the screen coordinate range to accommodate this. You should also examine the effect of varying N on the appearance of your plot from Simpson's method. Examine the effect of very small N values. Generally, you will need larger N when you explore the near-field features, or when you look far from the axis on the screen. Can you suggest why?

c) **2-d diffraction:** Now you are going to evaluate the 2-d Fresnel integral given in Eq. (2). However, for general (non-reentrant) apertures this can be rewritten as:

$$E(x, y, z) = \frac{kE_0}{2\pi z} \int_{y'_1}^{y'_2} \int_{x'_1(y')}^{x'_2(y')} \exp\left\{\frac{ik}{2z}\left[(x - x')^2 + (y - y')^2\right]\right\} \mathrm{d}x'\, \mathrm{d}y' \tag{5}$$

Note that the order of integration is important here. We are taking the y' integral as the outer integral, with the limits on the inner integral, x'_1 and x'_2 being potentially functions of y'.

Proceed in a similar way to part (a), but use `scipy.integrate.dblquad()` in place of `scipy.integrate.quad()`, and don't use Simpson's method for this part. You will need to provide two functions to evaluate the real and imaginary parts of the kernel of Eq. (5). The ordering of arguments is important in these functions. The code snippets below indicate how the arguments should be arranged and shows how to call `scipy.integrate.dblquad()` for cases where the inner limits are constant or varying.

```
from scipy.integrate import dblquad

def Fresnel2dreal(yp, xp, y, x, k, z): # must have yp first, xp second
def Fresnel2dimag(yp, xp, y, x, k, z): # must have yp first, xp second

# Calling dblquad with fixed limits
realpart = dblquad(Fresnel2dreal,yp1,yp2,xp1,xp2,args=(y, x, k, z))
imagpart = dblquad(Fresnel2dimag,yp1,yp2,xp1,xp2,args=(y, x, k, z))
# yp1, yp2, xp1 and xp2 are fixed upper and lower limits
# the order in the list args=(y, x, k, z) must match the Fresnel function

# Calling dblquad with varying limits
def xp1func(yp): # functions for inner integral limits can only have single argument
def xp2func(yp): # and must return a floating point value

realpart = dblquad(Fresnel2dreal,yp1,yp2,xp1func,xp2func,args=(y, x, k, z))
imagpart = dblquad(Fresnel2dimag,yp1,yp2,xp1func,xp2func,args=(y, x, k, z))
# function call looks very similar to the fixed limit case
```

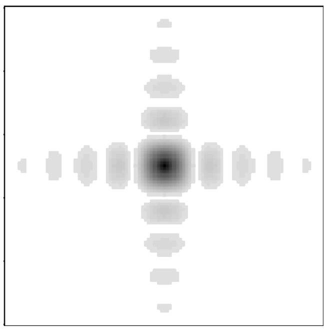
To test your approach, choose similar parameters to those in the table above, and calculate the diffraction intensity from a square aperture for a grid of (x, y) values on the screen. You will need to store these values in a 2-dimensional array and plot the array as an image. Here is an example of the use of a 2-d array to generate an image and, on the right, a typical far-field diffraction from a square aperture:

```
import numpy as np
import matplotlib.pyplot as plt

NumPoints = 200
delta = 4.0*np.pi / (NumPoints - 1)
intensity = np.zeros( (NumPoints,NumPoints) )

for i in range(NumPoints):
    x = i * delta
    for j in range(NumPoints):
        y = j * delta
        intensity[i,j] = np.sin(x) * np.cos(y)

plt.imshow(intensity)
plt.show()
```


Farfield diffraction from square aperture.

The 2-d diffraction calculations can be slow, so avoid taking too many points in the image; a 100×100 grid is reasonable for a final image, but will take over 30 s to compute, so you may find more helpful to work with 50×50 grids while testing. Repeat your calculation for different values of z and aperture size and shape (square or rectangular). Comment on any qualitative differences in your report.

d) Finally, adapt your code to allow your x' limits to be functions of your y' limits (see above), to allow you to calculate the diffraction from other shapes of apertures. Calculate the diffraction from a circular aperture and discuss what you find in your report.

Problem 2 (40% of marks): Monte Carlo integration

Some integrals are difficult to solve efficiently using conventional numerical integration. An alternative approach relies on the use of random numbers to 'sample' a function across its domain in order to estimate the integral. This approach (called Monte Carlo integration) has advantages and disadvantages compared to standard methods. In some cases, e.g. integrals over oddly-shaped regions, or in higher dimensions, it can provide a large increase in efficiency.

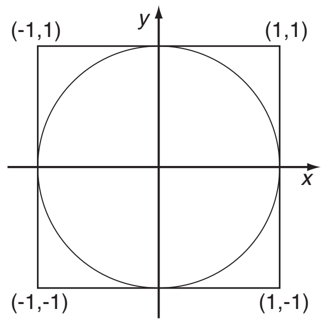
The Monte Carlo method is captured in the following equation:

$$\int f dV \approx V\left(\langle f\rangle \pm \sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}\right) \tag{6}$$

where angle brackets denote the average of N random samples. Quite often, we wish to integrate over a volume which is oddly shaped. This is possible, since we can choose random sample points from a regularly-shaped volume which includes the integration region, and then throw away those points that are not needed i.e. we assign the function the value zero for these.

As an example of Monte Carlo estimation, you will write a program to find the volume of a hypersphere. Conceptually, we draw a square with an inscribed circle (see figure). In the context of Eq. (6) you are performing the integral:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dxdy \qquad \begin{cases} f = 1, & x^2+y^2 \le 1 \\ f = 0, & x^2+y^2 > 1 \end{cases} \tag{7}$$



In practice, however, $\langle f\rangle \equiv 1$ and you need only count the points. Choosing points within the square at random, the ratio of the number of points lying within the circle to those outside is proportional to the ratio of areas:

$$\frac{n_{\text{inside}}}{n_{\text{total}}} \approx \frac{A_{circle}}{A_{square}} \qquad i.e. \quad A_{circle} \approx \frac{n_{\text{inside}}}{n_{\text{total}}} \times 2^2$$

for a circle of radius 1. This approach may be generalised to dimensions $d > 2$ (and unit radius):

$$\begin{aligned} V_{hypersphere} &= \int_{-1}^{1}\int_{-1}^{1}\cdots\int_{-1}^{1} f(x_1,x_2,\ldots,x_d)\,dx_1dx_2...dx_d \qquad \begin{cases} f = 1, & x_1^2+x_2^2+\ldots+x_d^2 \le 1 \\ f = 0, & x_1^2+x_2^2+\ldots+x_d^2 > 1 \end{cases} \\ &\approx \frac{n_{\text{inside}}}{n_{\text{total}}} \times 2^d \end{aligned}$$

N.B. In carrying out the following stages, you should add to your existing Python program, adding menu items as appropriate to correspond to the sections below.

e) Write a program to evaluate the integral in Eqn. (7) using the Monte Carlo approach.

f) Investigate the convergence of your integral on the 'correct' value of A_{circle} as a function of the number of sample points used, N. You would expect the error in A_{circle} to improve indefinitely as $1/\sqrt{N}$. Is this what you find?

HINT: to get a statistical measure of the error in the integral, you will need to repeat the integration several times for each N.

g) Now generalise your program to give the volume of a unit-radius hypersphere with dimension up to $d = 10$. Examine the convergence of your method with N for each d. Note: if $V_2 = \pi r^2$ and $V_3 = (4/3)\pi r^3$, the higher dimensional hypersphere volumes follow from:

$$V_d = 2\frac{\pi r^2}{d} V_{d-2}$$

Monte Carlo integration depends on reliable random numbers. You should read about random number generators and outline their potential deficiencies in your report.

Report

As previously, you should prepare a concise report outlining your methods and highlighting your findings with suitable graphs or tables. Credit will be given for a reasoned discussion of your findings.

Submitting your work

You should submit the following to Blackboard:

- A concise report, in MS Word or pdf format;
- Your program for 1-d and 2-d diffraction simulation, as well as the Monte Carlo integration;

Please note the following points:

- Please upload your “program.py" files.
- However, Turnitin won’t accept a file ending “.py" so please rename your file with a “.txt" extension.
- **Please note, there is only a single upload point for your program, so please don’t attempt to upload multiple files.**
- Please also give your programs sensible distinguishing names, including your name or userid e.g. “my_userid_ex3.txt".

If you have any problems submitting your work, please contact Dr. Hanna (s.hanna@bristol.ac.uk) or ask a demonstrator.