MATH3001 - Project in Mathematics

Computational Applied Mathematics

Ashleigh Shambrook - 201102747
Stephen Griffths
27/03/2020

Contents

Chapter 1

Using random numbers to integrate

1.1 Idea of Monte Carlo methods

Unlike in other numerical integration methods such as the Trapezium rule or Simpson's rule where the interval is split into smaller intervals of a set size, Monte Carlo methods select random numbers within the domain of the integral to estimate its value [1][pg 9]. Since the Python language has already got functions to create random samples over an interval using different distributions such as the normal distribution, using computational methods for Monte Carlo integration is more efficient than by hand.
When working in one dimension methods such as Trapezium rule usually give more accurate estimates where Monte Carlo methods can result in inconsistent results due to the random sampling involved. However when working in higher dimensions Monte Carlo becomes more effective as it samples over the entire area of the integral rather than first needing the split it into many 1-dimensional integrals.

1.2 Monte Carlo Integration for 1 Dimension

For most cases in the real world, performing integrals analytically is not possible, this is where numerical methods such as Monte Carlo are beneficial. They provide a way of approximating integrals as summations.

$$\int_a^b f(x)dx \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i) \tag{1.1}$$

Here equation (1.1) is a Monte Carlo approximation for the integral $\int_a^b f(x)dx$, done by selecting N random x values within the domain of integration and taking the average of the corresponding $f(x)$ values, then multiplying it by the width of the domain. This basically approximates the area under the line as a rectangle with the average of the $f(x)$'s being the height and width being $(b-a)$, this is shown by the cyan rectangle in figure (1.1) for the line $f(x) = -2x^2 + 1$ with $x \in (0, 1.75)$.

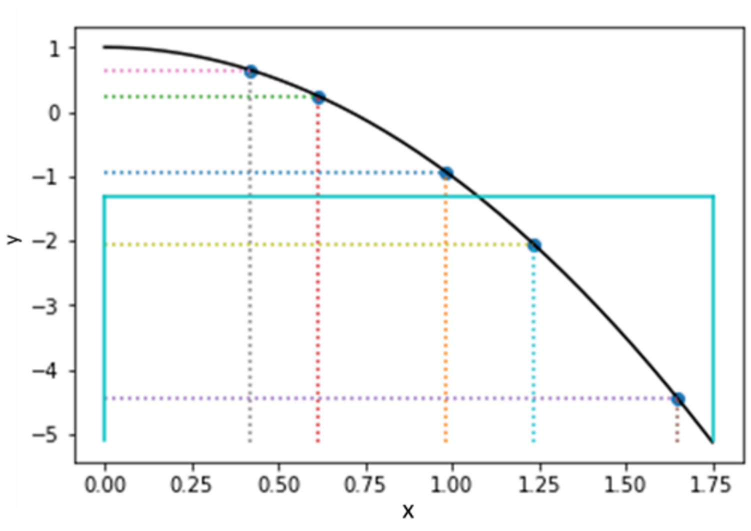

Figure 1.1: Example of Monte Carlo integral estimate in 1-Dimension

Comparing Monte Carlo methods against Trapezium rule in 1D. First consider the integral $\int_0^2 \frac{1}{2}\cos^2(x)$ as a test for some numerical methods. By applying equation (1.1) with $f = \frac{1}{2}\cos^2(x)$ and multiple sample sizes

$$N = [100, 200, 400, 800, 1600, 3200, 6400]$$

Comparing the estimates against the true integral value of 0.4054 we get a list of errors that when plotted in $\log_{10}$ result in a sequence which converge to the true value as $N \to \infty$.

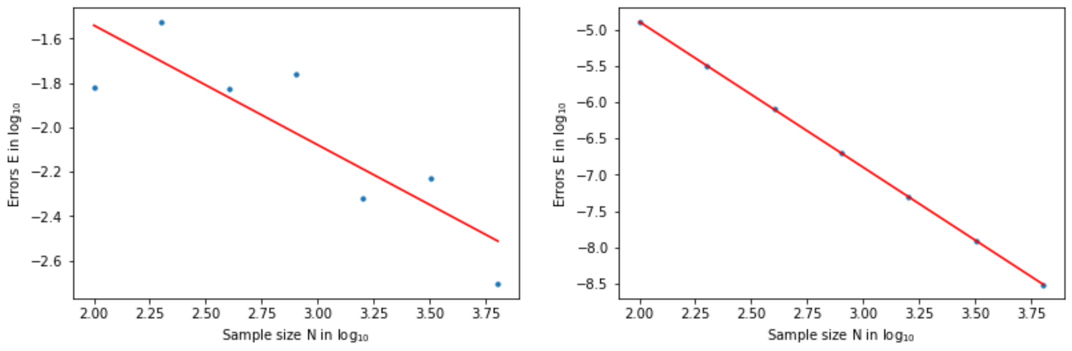

Figure 1.2: Monte Carlo estimation errors and Trapezium rule errors in $\log_{10}$

From figure (1.2), the Trapezium rule for 1-Dimension in figure has a more predictable trend than the Monte Carlo method. The rate of convergence of these sequences of errors is equal to the gradient of the line which, for Monte Carlo is 0.5378 implying the Errors are scaled approximately $N^{-0.5}$, and from the Trapezium method the rate of convergence is 2.0042 implying errors scaled as N^{-2}. So for 1-D the Trapezium Rule tends to the true value

4 times faster than the Monte Carlo and is therefore more efficient.

1.3 Monte Carlo Integration for the area of unit circle

For another example let us consider approximating the area of a unit circle using the function.

$$f = \begin{cases} 1, & x^2 + y^2 < 1 \\ 0, & \text{otherwise} \end{cases}, \quad \text{over the domain } V = \{(x, y) : |x| < 1, |y| < 1\} \tag{1.2}$$

For this consider the proportion of the area that a quarter circle covers from the area of a unit square. To do this sample (x, y) points from the unit square and test to see if they lie within the quarter circle or not,

$$\text{Area of circle } = 4\left(\frac{\text{Number of points within the quarter circle}}{\text{Number of points overall}}\right)$$

This method is demonstrated in Figure (1.3) with a sample size $N = 12800$.

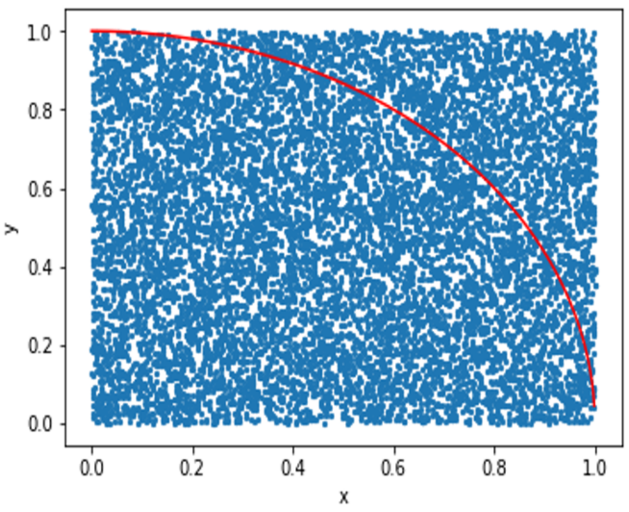

Figure 1.3: Monte Carlo estimation of π

For the sample sizes N, the approximations are as follows

N	100	200	400	800	1600	3200	6400	12800
estimate	3.0400	2.8800	3.2500	3.1250	3.0750	3.1050	3.1250	3.1475
error	0.1016	0.2616	0.1084	0.0166	0.0667	0.0366	0.0166	0.0059

Table 1.1: Table of estimates π for different sample sizes N

Clearly as the sample size increases the approximation gets closer the actual value of

π, however even with 12800 random points selected for just one quarter of the circle the approximation is only accurate to two decimal places. With this method if the random points happened to be distributed more on the outside of the quarter circle or more on the inside, it can result in an unreliable estimate, so large errors can occur.

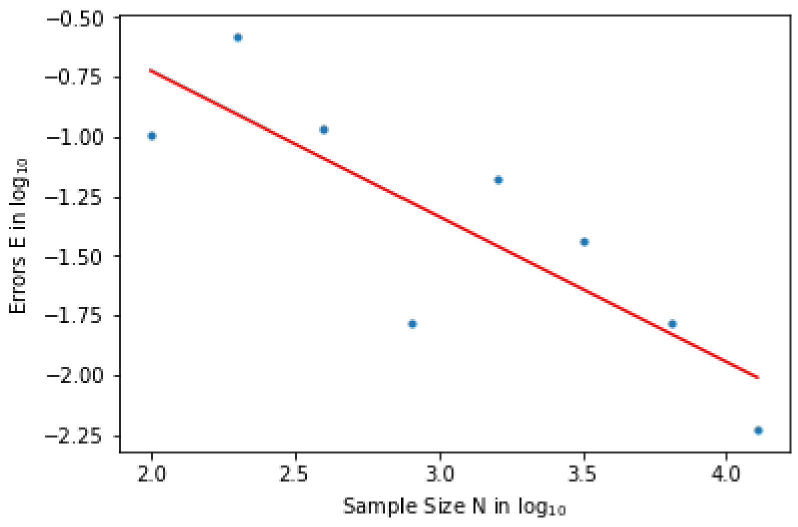

Figure 1.4: Monte Carlo errors in $\log_{10}$

While the errors are decreasing is it very gradual, this can be improved by taking an average of multiple large sample estimates as this reduces the effect of drastic outliers. The convergence for this error sequence is 0.6109, therefore Errors scaled close to $N^{-0.5}$ as for 1 dimension. Running this code, with $N = 10000$, 101 times then removing outliers greater than 2 standard deviations away from the mean, and taking the mean of the resulting list gives the estimate as 3.1425 with error of 0.0009448. This is more accurate that the estimate for $N = 12800$ in Table (1.1) however only by one decimal place.

1.4 Multidimensional Integration approximations

Unlike in 1D the Monte Carlo method is favoured when considering multidimensional integration. This is because the method of random selecting numbers within the domain of integration over a N-Dimensional space with $N = 2, 3, 4, ...,$ is easily possible using python, whereas using other numerical methods such as trapezium rule would require the space to be split into multiple nested integration's within 1 dimension which is very tedious. For example lets consider hyper-spheres. The function for this is as follows

$$f(x) = \begin{cases} 1, & \text{if x lies within the hyper-sphere} \\ 0, & \text{otherwise} \end{cases} \tag{1.3}$$

within the domain of a hyper-cube containing the hyper-sphere. In terms of vector calculus, for integrating $f(x)$ over a D-dimensional domain $x \in V$, in terms of one-dimensional integrals

$$\int_V f(x)dV = \int_{a_1}^{b_1} ... \int_{a_D}^{b_D} f(x)dx_D...dx_1$$

where V is contained in the hyper-cuboid (a, b) and $f = 0$, for $x \notin V$. For this, the errors scale as $N^{\frac{1}{D}}$. The Monte Carlo method for approximating this integral is

$$\int_V f(x)dV \approx \frac{V}{N}\sum_{i=1}^{N} f(x_i) \pm V\frac{\sigma}{\sqrt{N}} \tag{1.4}$$

And therefore no matter the dimensions the error scales as $N^{0.5}$.

1.5 Volume of a Sphere

Monte Carlo for the volume of a Sphere.

By taking the method described for estimating the area of a unit circle and applying it to the area of a sphere we can generate random numbers within a unit-cube and count the proportion that fall within an $\frac{1}{8}$ of the sphere. Running this in python we sample size $N = [100, 200, 400, 800, 1600, 3200, 6400]$ gives the estimates, compared to the actual value of $\frac{4}{3}\pi r^3 = \frac{4}{3}\pi(1)^3 = 4.188790205$ [2].

N	100	200	400	800	1600	3200	6400	12800
estimate	4.9600	4.3200	4.3800	4.2300	4.3550	4.0825	4.1950	4.2588
error	0.7712	0.1312	0.1912	0.0412	0.1462	0.1062	0.0062	0.0700

Table 1.2: Estimate error for the volume of a sphere for sample sizes N

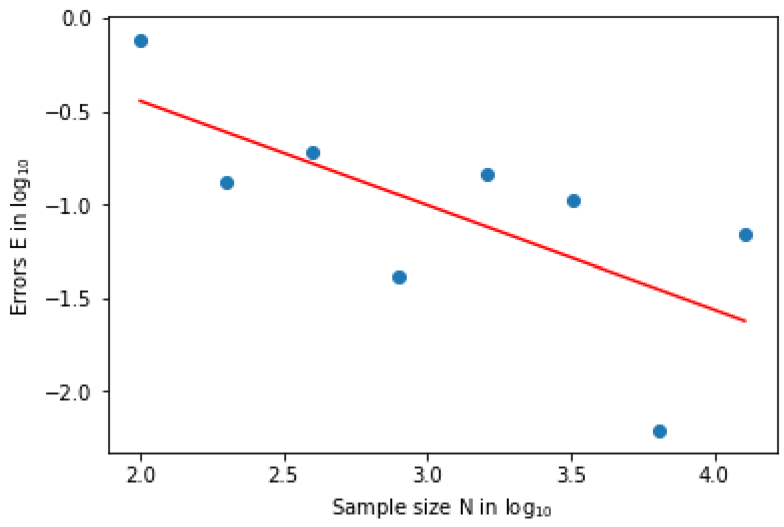

Figure 1.5: 3 Dimensions errors

Looking at these results in Figure (1.5) the estimate errors decrease generally as the sample sizes increase as expected and the estimates lie close to the actual value. The convergence for this sequence of errors is 0.5590 which is very similar to the expected error scaled as close to $N^{-0.5}$.

1.6 Volume of a 3-Sphere.

Then applying this method again with 4-dimensions results in the estimates in Table (1.3) below, compared to actual value of $\frac{1}{2}\pi^2 r^4 = \frac{1}{2}\pi^2(1)^4 = 4.934802201$ [2]

N	100	200	400	800	1600	3200	6400	12800
estimate	4.0000	5.6800	4.9600	5.0600	5.000	4.8200	5.0825	4.9475
error	0.9348	0.7452	0.0252	0.1252	0.0652	0.1148	0.1477	0.0127

Table 1.3: Estimate error for the volume of a 3-sphere for sample sizes N

Again these errors generally decrease as the sample size increases, which can be seen in Figure (1.6).

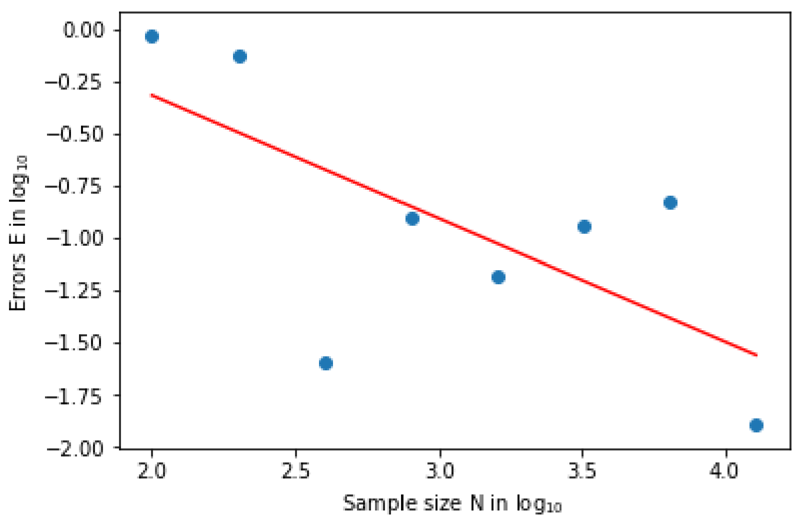

Figure 1.6: Estimation errors in $\log_{10}$ for the volume of 3-sphere for different sample sizes N

The convergence for this sequence is 0.5889 which again close to the expected error scaling, $E = N^{-0.5}$.

1.7 Discussion

Multi-Dimension Integration Errors

The relative error for the Monte Carlo method scales as $N^{-0.5}$, which is true independent of the number of dimensions the integration occurs over. As realised from equation (1.4). For the trapezium method, with dimensions D, number of function evaluations N. For 1 Dimension with domain width W and M number of points sampled, $N = M$ and error $E = h^2 \propto M^{-2} = N^{-2}$ where $h = \frac{W}{M}$. Transforming this in to a 2 dimensional problem with equal h, there is now $N = M^2$ function evaluations and error $E = h^2 \propto M^{-2} = N^{-1}$. As stated in [3][pg 2], the errors for trapezium scale as $N^{-\frac{2}{D}}$,

D	1	2	3	4	5	6
Error	N^{-2}	N^{-1}	$N^{\frac{-2}{3}}$	$N^{\frac{-1}{2}}$	$N^{\frac{-2}{5}}$	$N^{\frac{-1}{3}}$

Table 1.4: Scale of errors by Trapezium Rule as the dimension increases

Therefore from Table (1.4), it can be concluded that for $D \approx 3 - 4$ the error is approximately equal to that for Monte Carlo, as explained in [4][pg 109]. I would next test this hypothesis with a non separable D-dimensional integration problem with $D = [2, 3, 4]$ and solve with both Trapezium and Mont Carlo method to see the point at which Monte Carlo is a preferable integration method.

Chapter 2

The Rayleigh-Ritz method

2.1 Introduction

The Rayleigh-Ritz method is a numerical method used to approximate eigenvalue equations that are not easily solved analytically by the use of a trial function. As explained in [5][pg 18-20], this method starts with a set of trial functions, of which linear combinations of these are taken aiming to make the lambda function stationary. The accuracy of this methods relies mainly on the trial function chosen. The trial functions are defined over the entire region and also must satisfy the boundary conditions across the whole boundary. This means that for problems that require a high level of accuracy or have a complex boundary need more trial functions with greater accuracy. So it is not a suitable method for complex problems found in engineering [6][pg 491].

2.2 Bessel's equation for a circular drum

The oscillations from vibrations of a circular drum with frequency ω and wave speed c satisfy the Bessel equation of order zero.

$$\frac{d^2y}{dx^2} + \frac{1}{x}\frac{dy}{dx} + \lambda y = 0 \quad \text{for} \quad 0 \leq x \leq 1 \tag{2.1}$$

with boundary conditions $y(1) = 0$, y non-singular at $x = 0$ and $\lambda = \frac{\omega^2}{c^2}$. This problem can be rearranged into a Sturm-Liouville eigenvalue problem.

$$-\frac{d}{dx}\left(x\frac{dy}{dx}\right) = \lambda x y, \quad 0 \leq x \leq 1 \tag{2.2}$$

With the exact solution given by

$$y(x) = J_0\big(\lambda^{\frac{1}{2}} x\big) \tag{2.3}$$

where $\lambda^{\frac{1}{2}}$ is a zero of the Bessel function J_0. The eigenvalues of (2.1) are determined from the zeros of the Bessel function J_0 (2.3), as the squares of the zeros of the Bessel's function [7][pg 50], due to general Sturm-Liouville theory, and can be computed using python scipy functions [8][pg 221]. The first four zeros of J_0 are

$$J = [2.4048, 5.5201, 8.6537, 11.7915]$$

with corresponding eigenvalues

$$\lambda = [5.7832, 30.4713, 74.8870, 139.0403]$$

calculated using the python library scipy.special. For each of these zeros exact solutions $y(x)$ for Bessel's equation can be calculated using equation (2.3), and plotted in figure (2.1).

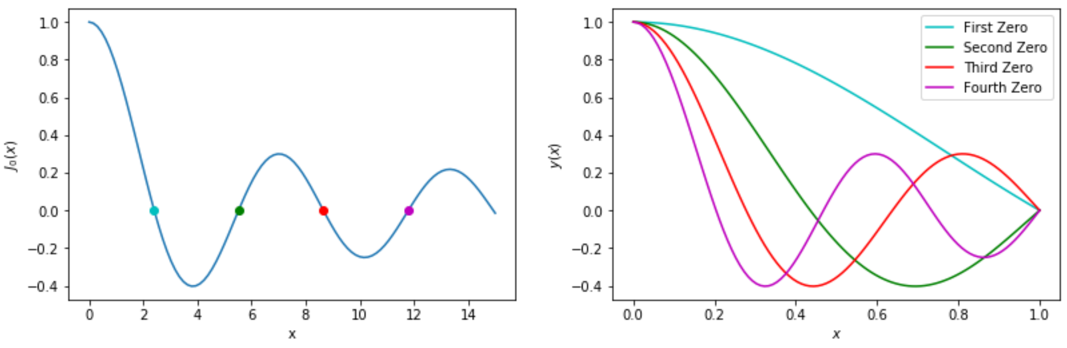

Figure 2.1: Graph of $J_0(x)$, Graph of corresponding solutions $y(x)$ of Bessel's equation for the first four zero's of $J_0(x)$

2.3 Approximating the exact solutions of Bessel's Equation

The Rayleigh-Ritz method can approximate the eigenvalue solutions to Bessel's equation using the equivalent Sturm-Liouville problem (2.2) with the same boundary conditions from equation (2.1). By multiplying by y and integrating between the boundary conditions on x, this becomes

$$\int_0^1 y(xy')'dx = \lambda \int_0^1 xy^2 dx$$

Next applying integration by parts for the left hand side, then using the boundary conditions on $y(x)$

$$xyy'|_0^1 - \int_0^1 xy'^2 dx = -\lambda \int_0^1 xy^2 dx$$

$$\implies \lambda = \frac{\int_0^1 xy'^2 dx}{\int_0^1 xy^2 dx} = \frac{I}{J}$$

$$\text{Letting} \quad I = \int_0^1 xy'^2 dx, \quad J = \int_0^1 xy^2 dx \tag{2.4}$$

Where the extrema of $\frac{I}{J}$ are exactly the eigenvalues λ of the Sturm-Liouville, provides upper bound on λ_0. For a trial assume $y_t(x)$ an even function, of the form

$$y_t(x) = ax^6 + bx^4 + cx^2 + d \tag{2.5}$$

for arbitrary parameters a, b, c, d with $a+b+c+d=0$ and allowing $d=1$. Subbing $y_t(x)$ into equation (2.4) for I_t and J_t.

$$I_t = \int_0^1 xy'2dx = 3a^2 + \frac{24ab}{5} + 3ac + 2b^2 + \frac{8bc}{3} + c^2$$

$$J_t = \int_0^1 xy^2dx = \frac{a^2}{14} + \frac{ab}{6} + \frac{ac}{5} + \frac{a}{4} + \frac{b^2}{10} + \frac{bc}{4} + \frac{b}{3} + \frac{c^2}{6} + \frac{c}{2} + \frac{1}{2}$$

By letting $a=0$, so $c=-1-b$, I_t and J_t become

$$I_t = \frac{b^2}{3} - \frac{2b}{3} + 1, \quad J_t = \frac{b^2}{60} - \frac{b}{12} + \frac{1}{6} \implies \lambda = \frac{\frac{b^2}{3} - \frac{2b}{3} + 1}{\frac{b^2}{60} - \frac{b}{12} + \frac{1}{6}}$$

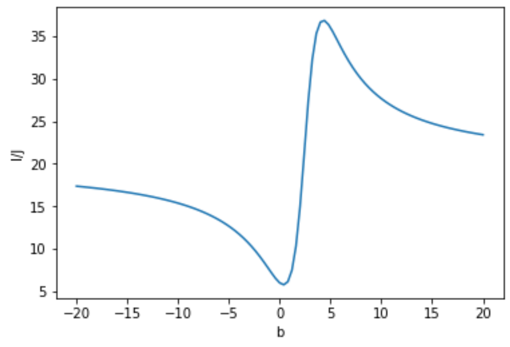

Figure 2.2: Graph of $\frac{I_t}{J_t}$ for varying b

By plotting $\frac{I_t}{J_t}$ against varying values of b shows the behaviour of λ and the approximate value of b at which λ is at a minimum. This minimum of $\lambda = \frac{I_t}{J_t}$ can be calculated as 5.7843 with $b \approx 0.4$. The graph of $y_t(x)$ with $b = 0.4$ is shown in figure (2.3)

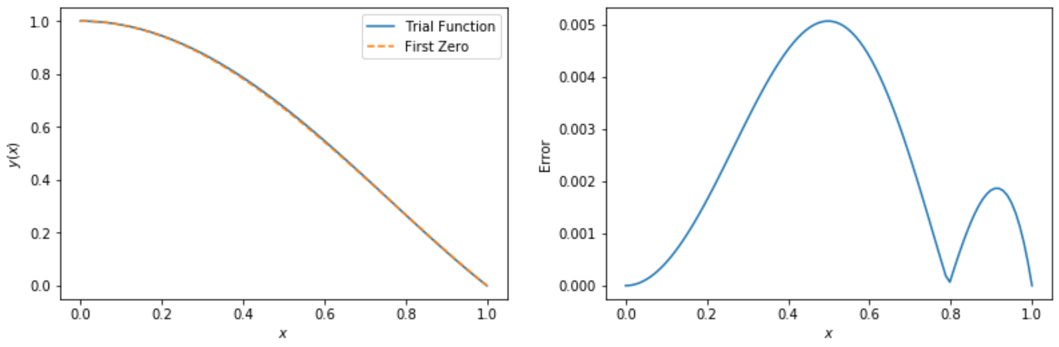

Figure 2.3: Graph of solution for first zero of $J_0(x)$ with the trial function $y_t(x)$, graph of the errors of this estimate

From Figure (2.3) the trial function $y_t(x) = bx^4 + cx^2 + 1$ is an accurate approximation to

the exact solution for the first zero to 2 decimal places, with the largest error at $x = 0.4949...$ and most accurate around $x = [0, 0.8, 1]$.
Then re-introducing a again as a variable and applying $a + b + c + 1 = 0$, new formulas for I_t and J_t are as follows.

$$I_t = a^2 + \frac{17ab}{15} - a + \frac{b^2}{3} - \frac{2b}{3} + 1, \qquad J_t = \frac{4a^2}{105} + \frac{ab}{20} - \frac{7a}{60} + \frac{b^2}{60} - \frac{b}{12} + \frac{1}{6}$$

Using a sympy.solve function for the solving the partial differentials equal to zero to find the critical points of F gives the 5 resulting points

$$\begin{aligned}&[(-19.3659 + 0.e^{-22}i, 31.0412 - 0.e^{-20}i), (-5.4476 - 1.5130i, 10.6122 + 0.8525i),\\ &(-5.4476 + 1.5130i, 10.6122 - 0.8525i), (-3.9815 - 0.e^{-22}i, 9.7861 + 0.e^{-19}i),\\ &(-0.0680 - 0.e^{-22}i, 0.5110 + 0.e^{-19}i)]\end{aligned} \tag{2.6}$$

These points have negligible imaginary parts in the values of (a, b). This is likely due to a computational error with the limitations on working with large decimals in python. Therefore disregarding the imaginary parts the location of the critical points with the corresponding trial functions are

$$(-0.0680, 0.5110) \qquad y_1(x) = -0.0680x^6 + 0.5110x^4 - 1.443x^2 + 1 \tag{2.7}$$

$$(-3.9815, 9.7861) \qquad y_2(x) = -3.9815x^6 + 9.7861x^4 - 6.8046x^2 + 1 \tag{2.8}$$

$$(-19.3659, 31.0412) \qquad y_3(x) = -19.3659x^6 + 31.0412x^4 - 12.6753x^2 + 1 \tag{2.9}$$

And the corresponding eigenvalues for these points are $[5.7832, 30.7121, 113.5047]$. Comparing these to the first three exact eigenvalues, the errors are $[9.6490 \cdot 10^{-7}, 0.2408, 38.6177]$ therefore the corresponding trial functions for (2.7) and (2.8) will be close approximations to the first and second zero solution of Bessel's equation. However (2.9) isn't close to either the third or fourth eigenvalue therefore the corresponding trial function will not be an approximation to any exact solution. These conclusions are shown in Figures (2.4), (2.5) and (2.6).

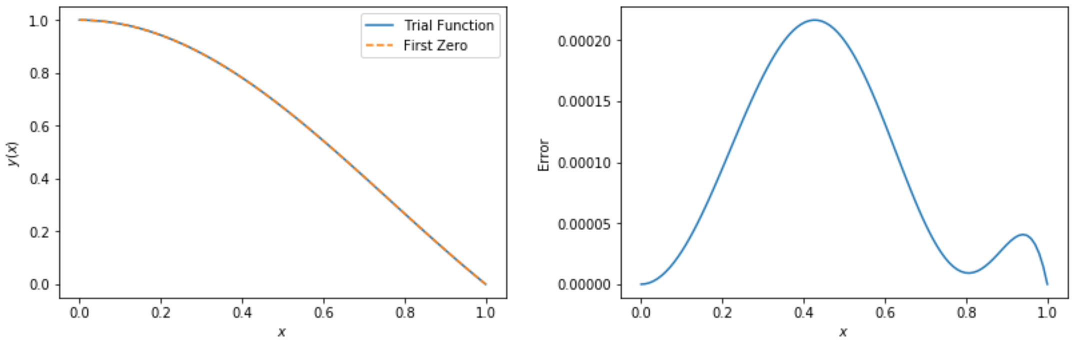

Figure 2.4: Graph for the solution $y(x)$ for the second zero of $J_0(x)$ with the new trial function $y_1(x)$, graph of the errors of this estimate

From Figure (2.4) the Trial function $y_1(x)$ is an accurate estimate for the First Zero solution of $J_0(x)$ to 3 decimal places with the largest error occuring at $x = 0.4242$.

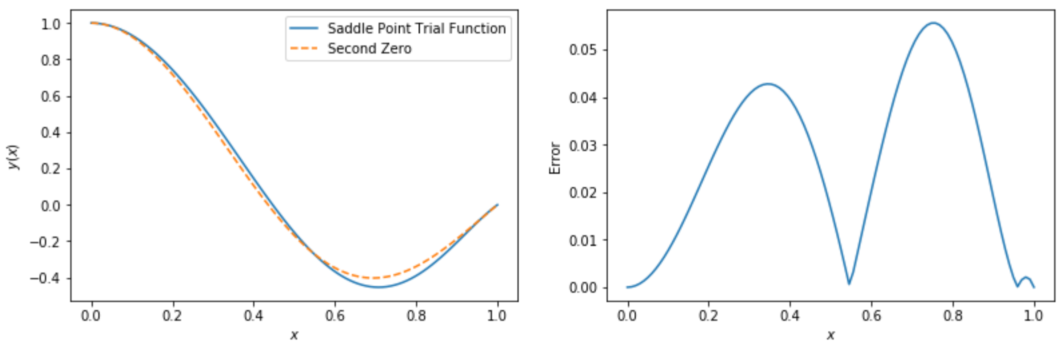

Figure 2.5: Graph for the solution $y(x)$ for the second zero of $J_0(x)$ with the trial function $y_2(x)$, graph of the errors of this estimate

From figure (2.5), $y_2(x)$ provides a close estimate to the second zero solution $y(x)$ to 1 decimal place with the greatest error at $x = 0.7575$. Finally, the trial function $y_3(x)$ is not an approximation to any of the first four zeros shown in figure (2.1).

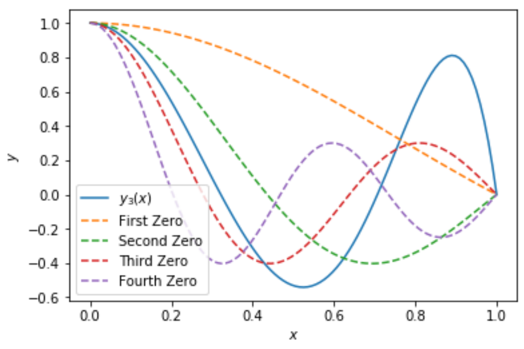

Figure 2.6: $y_3(x)$ compared to the exact solutions of Bessel's equation

2.4 Discussion

Since the accuracy of the Rayleigh-Ritz method relies on the trial function chosen, other trial functions could provide a better approximation to the exact solutions of the Bessel's equation. Since for example trial function used was only a good approximation to two of the exact solutions it can be improved upon. For example other trial function which could produce a more accurate approximation are

$$y_t(x) = ax^8 + bx^6 + cx^4 + dx^2 + 1 \tag{2.10}$$

$$y_t(x) = a\cos\left(\frac{5\pi x}{2}\right) + b\cos\left(\frac{3\pi x}{2}\right) + \cos\left(\frac{\pi x}{2}\right) \tag{2.11}$$

Where (2.10) is another polynomial of a higher order, and (2.11) is a trigonometric trial function which satisfies the condition $y(1) = 0$. However following the same coding method used with the trial function (2.5) from this chapter is more complex than my computer is capable to run. For trial function (2.11) by using the optimize.minimize python function, I found one set of $(a, b) = (0.0319, -0.0072)$ such that λ is minimised. However, the Errors shown in Figure (2.7) show that the first exact solution of Bessel's function from the trial function (2.5) provides a better approximation.

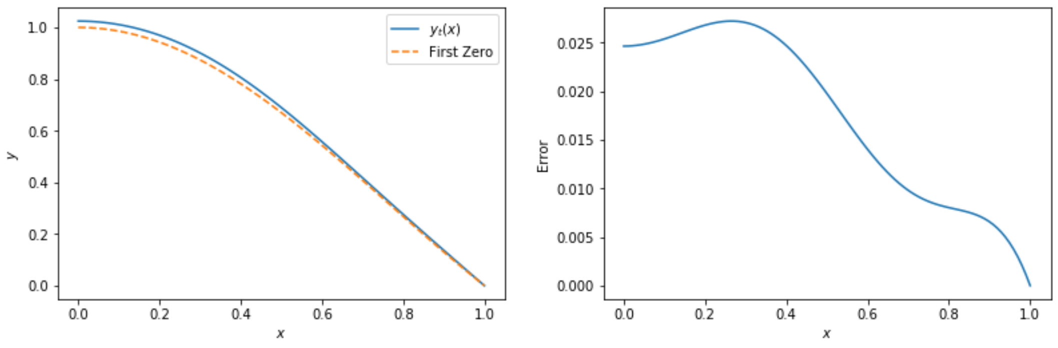

Figure 2.7: $y_t(x)$ compared to the exact solution of Bessel's equation

Chapter 3

Determining the period of a pendulum

3.1 Theory of the pendulum

A simple pendulum is made of a heavy mass attached to the end of a string or light rigid rod with negligible mass. The period T of pendulum is the time it takes for the mass to complete one cycle, and if there is no friction theoretically the mass will swing infinitely with no change in T unless interrupted by an outside force. If the mass is held perfectly at π radians from equilibrium the mass will not move. The study into the motion of the pendulum can be traced back to Galileo Galolei around 1602, where suggested it could be used to measure time [9]. He observed that the length of a period depended on the length of the rod and was very nearly independent of the angle from equilibrium at which it was released [10]. A pendulum was first used by Dutch Physicist Christiaan Huygens as a way to regulate clocks in 1673 [11].

3.2 Mathematics in the motion of a pendulum.

Let l be the length of the rod holding the mass, g be the downward gravitational acceleration influencing the mass, θ be the angle away from the downward equilibrium and τ be the time corresponding to the position of the mass. Shown in figure (3.1)

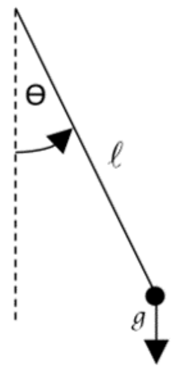

Figure 3.1: Diagram of the motion of a Pendulum

Starting from Newton's second law $F = ma$, two forces are acting on the mass, firstly there is gravity which acts downwards with a magnitude of mg and the other is tension in the string acting from the mass to the pivot. Combing these forces results in

$$F = mg\sin\theta \tag{3.1}$$

acting along the line of motion. Then using Newtons second law gives this force as equal to the mass·acceleration. $m\ddot{x} = -mg\sin\theta$, where x is the distance travelled (length of arc) and $x = l\theta$. Therefore, we can write (3.1) as $ml\ddot{\theta} = -mg\sin\theta$, rearranging to

$$\frac{d^2\theta}{d\tau^2} = -\frac{g}{l}\sin\theta \quad \text{with } \theta(0) = \theta_0, \quad \frac{d\theta}{d\tau}(0) = 0 \tag{3.2}$$

Then by introducing a re-scaled non-dimensional time $t = \tau/k$ with $k = \sqrt{\frac{g}{l}}$ then it can be written in the form

$$\ddot{\theta} = -\sin\theta(t), \quad \theta(0) = \theta_0, \quad \dot{\theta}(0) = 0, \tag{3.3}$$

with $0 < \theta_0 < \pi$ and $\ddot{\theta} = \frac{d^2\theta}{dt^2}$.

3.3 The small-amplitude approximation.

Since the system (3.3) cannot be solved analytically, the problem can be approached from a few different angles, the first being the small angle approximation. This can approximate well for small angles but is not very accurate for $\theta \gg 1$. The small angle approximation gives approximate formulas for $\sin(\theta) \approx \theta$, $\cos\theta \approx 1 - \frac{\theta^2}{2}$ and $tan(\theta) \approx \theta$, using the $\sin\theta$ approximation in 3.3 gives

$$\ddot{\theta} = -\sin\theta \to \ddot{\theta} \approx -\theta$$

This differential equation can be solved with complex solutions, so roots are of the form $\theta(t) = e^{\alpha t}[A\cos(\beta t) + B\sin(\beta t)]$ and then solving for A and B using initial conditions $\theta(0) = \theta_0$ and $\dot{\theta}(0) = 0$ gives $A = \theta_0$ and $B = 0$. Therefore the solution is

$$\theta(t) \approx \theta_0\cos(t)$$

When $\theta(t) = 0$ we get that $\cos(t) = 0 \implies t \approx \frac{\pi}{2}$, this is a quarter of the total period so then multiplying by 4 gives

$$T \approx 2\pi \tag{3.4}$$

Calculating T in the form of an integral.
Starting with the system (3.3) with the small angle approximation to give the system

$$\ddot{\theta} = -\theta, \quad \theta(0) = \theta_0, \quad \dot{\theta}(0) = 0 \tag{3.5}$$

Then multiplying this (3.5) by $\dot{\theta}$ to give

$$\frac{d^2\theta}{dt^2}\frac{d\theta}{dt} = -\theta\frac{d\theta}{dt} \to \frac{d}{dt}\left[\frac{1}{2}\left(\frac{d\theta}{dt}\right)^2 - \left(1 - \frac{\theta^2}{2}\right)\right] = 0$$

Then by integration we get

$$\left(\frac{d\theta}{dt}\right)^2 - 2\left(1 - \frac{\theta^2}{2}\right) = c \quad \text{with initial conditions} \quad \dot{\theta}(0) = 0, \quad \theta(0) = \theta_0 \tag{3.6}$$

Therefore we can solve for $c = -2(1 - \frac{\theta^2}{2})$ then by subbing back into (3.5) we get

$$\left(\frac{d\theta}{dt}\right)^2 = \theta_0^2 - \theta^2$$

$$\implies \frac{dt}{d\theta} = \frac{1}{\sqrt{\theta_0^2 - \theta^2}} \implies T = 4\int_0^{\theta_0} \frac{d\theta}{\sqrt{\theta_0^2 - \theta^2}} \tag{3.7}$$

By rearranging (3.7) as follows and then using the substitution $\frac{\theta}{\theta_0} = \sin u$ the integral can be evaluated to

$$T = \frac{4}{\theta_0}\int_0^{\theta_0} \frac{d\theta}{\sqrt{1 - (\frac{\theta}{\theta_0})^2}} = 4\int_0^{\frac{\pi}{2}} 1du \qquad \text{,by substitution}$$

$$= 4\big[u\big]_0^{\frac{\pi}{2}} = 4 \cdot \frac{\pi}{2} = 2\pi$$

This proves the same value as before for (3.4) with an alternative integral method.

3.4 The extension to arbitrary amplitude

Then multiplying this (3.3) by $\dot{\theta}$ to give

$$\frac{d^2\theta}{dt^2}\frac{d\theta}{dt} = -\sin\theta\frac{d\theta}{dt} \to \frac{d}{dt}\left[\frac{1}{2}\left(\frac{d\theta}{dt}\right)^2 - \cos\theta\right] = 0$$

Then by integration we get

$$\left(\frac{d\theta}{dt}\right)^2 - 2\cos\theta = c \quad \text{with initial conditions} \quad \dot{\theta}(0) = 0, \quad \theta(0) = \theta_0 \tag{3.8}$$

Therefore we can solve for $c = -2\cos\theta$ then by subbing back into (3.8) we get

$$\left(\frac{d\theta}{dt}\right)^2 = 2(\cos\theta - \cos\theta_0)$$

$$\implies \frac{dt}{d\theta} = \frac{1}{\sqrt{2(\cos\theta - \cos\theta_0)}} \implies T = 4\int_0^{\theta_0} \frac{1}{\sqrt{2(\cos\theta - \cos\theta_0)}}d\theta$$

$$\implies T = 2\sqrt{2}\int_0^{\theta_0} \frac{1}{\sqrt{(\cos\theta - \cos\theta_0)}} d\theta \tag{3.9}$$

This cannot be calculated by a simple substitution as in (3.7) as this integrand in unbounded as $\theta \to \theta_0^-$. There are a few ways this integrand can be approached. One of which is using systematic method approximation of the integral, such as, the Trapezium Rule briefly mentioned in Chapter 2 also.

3.5 Solving for T when $\theta_0 = \frac{\pi}{2}$

When $\theta_0 = \frac{\pi}{2}$, $\cos\theta_0 = 0$ so equation (3.9) becomes

$$T = 2\sqrt{2}\int_0^{\frac{\pi}{2}} \frac{d\theta}{\sqrt{\cos\theta}} \tag{3.10}$$

This still cannot be evaluated, however it can be expressed in terms of the Beta function

$$B(\alpha,\beta) = 2\int_0^{\frac{\pi}{2}} \cos^{2\alpha-1}\theta \sin^{2\beta-1}\theta \quad d\theta = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{3.11}$$

as defined in [12][pg 13-15]. With $(\alpha,\beta) = (\frac{1}{4},\frac{1}{2})$, this can be used to solve (3.10) in terms of the gamma function, which can be solved with scipy.special.gamma in python, therefore

$$T = \frac{\Gamma(\frac{1}{4})\Gamma(\frac{1}{2})}{2\Gamma(\frac{3}{4})} = \frac{\sqrt{\pi}}{2}\frac{\Gamma(\frac{1}{4})}{\Gamma(\frac{3}{4})}, \qquad \text{since} \quad \Gamma\left(\frac{1}{2}\right) = \sqrt{\pi} \tag{3.12}$$

Then using Euler's reflection formula for gamma functions $\Gamma(x)\Gamma(1-x) = \frac{\pi}{\sin(x\pi)}$.

$$T = \frac{\Gamma^2(\frac{1}{4})}{\sqrt{\pi}} \approx 7.4163 \tag{3.13}$$

3.6 Using Numerical analysis.

Since some values of T can be calculated, such as in equation (3.13). Taking the Trapezium method with $\theta_0 = \frac{\pi}{2}$, $f(x) = \frac{1}{\sqrt{\cos x - \cos\theta_0}}$, and

$$N = [100, 200, 400, 800, 1600, 3200, 6400, 12800]$$

points we can approximate the integrand and compare it to the true value. However because the integral is unbounded the trapezium rule cannot be calculated including the singular point when $\theta \to \theta_0$, so the end trapezium has to be discarded. Since the trapezium rule is estimating the area under the curve from points x_0 to x_{N-1} the estimate will always be smaller but as N increases it will tend to the true value of T at $\theta_0 = \frac{\pi}{2}$.

N	100	200	400	800	1600	3200	6400
estimate	6.7214	6.9249	7.0688	7.1706	7.2426	7.2934	7.3294
error	0.6949	0.4914	0.3475	0.2457	0.1737	0.1298	0.0869

Table 3.1: Sequence of approximations for estimates and errors, Trapezium

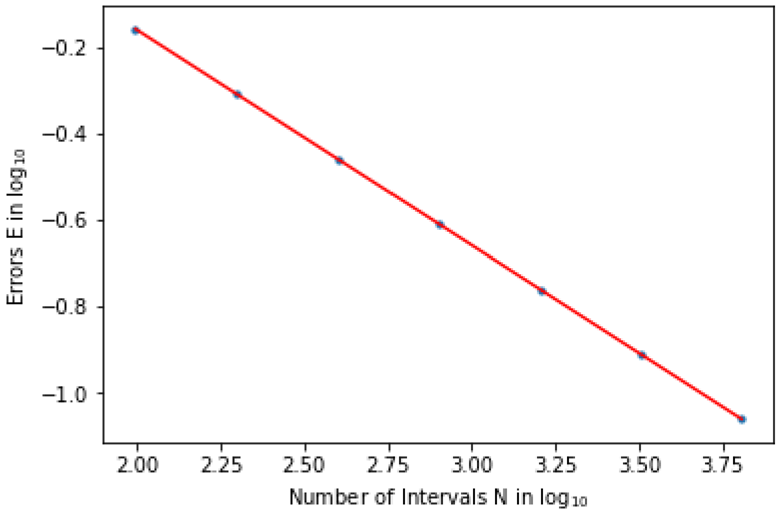

Figure 3.2: Errors for trapezium estimate in $\log_{10}$

The rate of convergence for this graph is computed as 0.49894, so the errors scale as N^{-p} with $p = 0.5$, is a very slow rate of convergence, which means it will take a very large number of intervals N to get an accurate approximation by Trapezoidal rule.

Removing the singular point.
The above method can be improved upon by evaluating the improper integral at the singular point and approximate the remainder with the Trapezoidal Rule. By using the Taylor expansion of $\cos\theta$ at $\theta = \theta_0$,

$$\frac{1}{\sqrt{\cos\theta - \cos\theta_0}} = \frac{1}{\sqrt{(\sin\theta_0)(\theta_0 - \theta)}} + b(\theta_0 - \theta)^{\frac{1}{2}} + c(\theta_0 - \theta)^{\frac{3}{2}} + ... \quad \text{as} \quad \theta \to \theta_0^-$$

This can be used to rearrange (3.9) into

$$T = 4\sqrt{\frac{2\theta_0}{\sin\theta_0}} + 2\sqrt{2}\int_0^{\theta_0}\left(\frac{1}{\sqrt{\cos\theta - \cos\theta_0}} - \frac{1}{\sqrt{(\sin\theta_0)(\theta_0 - \theta)}}\right)d\theta \tag{3.14}$$

Where the integral does not have this singular point so can be evaluated more accurately when is the trapezium rule. Using $N = [100, 200, 400, 800, 1600, 3200, 6400]$ with the trapezium rule same as before with the new integral equation (3.14) gives the data

estimate	7.4163	7.4163	7.4163	7.4163	7.4163
error 10^{-5}	1.0940	$3.0155 \cdot 10^{-1}$	$8.0346 \cdot 10^{-2}$	$2.0963 \cdot 10^{-2}$	$5.3957 \cdot 10^{-3}$
estimate	7.4163	7.4163			
error 10^{-5}	$1.3763 \cdot 10^{-3}$	$3.4892 \cdot 10^{-4}$			

Table 3.2: Sequence of approximations for estimates and errors, Trapezium

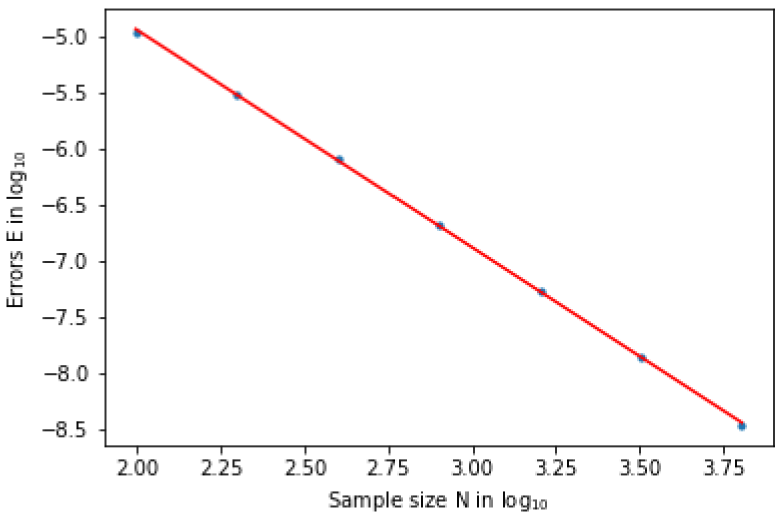

Figure 3.3: Errors for improved trapezium estimate in $\log_{10}$

Looking at the gradient of this graph as with figure (3.2) the rate of convergence is 1.9348. which is an improvement the previously calculated Trapezoidal Rule at 0.4991 and is close to the theoretical maximum of 2 for Trapezium Rule.

3.7 Integral Evaluation.

Equation (3.9) can be re-arranged

$$T = 2\sqrt{2}\int_0^{\theta_0} \frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} = 2\int_0^{\theta_0} \frac{d\theta}{\sqrt{\sin^2\frac{\theta_0}{2} - \sin^2\frac{\theta}{2}}} = 4\int_0^{\frac{\pi}{2}} \frac{d\phi}{\sqrt{1 - k^2\sin^2\phi}} \tag{3.15}$$

with $k = \sin\frac{\theta_0}{2}$, $\sin\phi = \frac{\sin\frac{\theta}{2}}{\sin\frac{\theta_0}{2}}$. This can be expanded by using the Binomial series formula as described in [13][pg 12-13]

$$(1-x)^\alpha = \sum_{m=0}^{\infty}\binom{\alpha}{m}(-1)^m x^m$$

so equation (3.15) becomes

$$T = 4\int_0^{\frac{\pi}{2}} \sum_{m=0}^{\infty} \binom{-\frac{1}{2}}{m} (-1)^m k^{2m} \sin^{2m}\phi \quad d\phi \tag{3.16}$$

$$= 4\sum_{m=0}^{\infty} \binom{-\frac{1}{2}}{m} (-1)^m k^{2m} \int_0^{\frac{\pi}{2}} \sin^{2m}\phi \quad d\phi \tag{3.17}$$

Using the Beta function (3.11) this becomes

$$= 4\sum_{m=0}^{\infty} \frac{(-1)^m \Gamma(\frac{1}{2})}{m!\Gamma(\frac{1}{2}-m)} \cdot \frac{\Gamma(\frac{1}{2})\Gamma(\frac{1}{2}+m)}{2m!} k^{2m}$$

Then applying Euler's reflection formula on $\Gamma(\frac{1}{2}+m)$ and $\Gamma(\frac{1}{2}-m)$. Also using that $\Gamma^2(\frac{1}{2}) = \pi$.

$$T = 4\frac{\pi}{2}\sum_{m=0}^{\infty} \frac{(-1)^m \Gamma^2(\frac{1}{2}+m)}{\pi(m!)^2} \sin\left(\pi\left(m+\frac{1}{2}\right)\right) k^{2m} \tag{3.18}$$

$$= 2\pi \sum_{m=0}^{\infty} \left[\frac{(2m-1)!!}{2^m m!}\right]^2 k^{2m}, \quad \text{since} \quad (-1)^m \sin\left(\pi\left(m+\frac{1}{2}\right)\right) = 1 \quad \forall m \in \mathrm{I}. \tag{3.19}$$

$$= 2\pi\left[1 + \frac{1}{2^2}k^2 + \left(\frac{3\cdot 1}{2\cdot 2}\right)^2 \frac{1}{2!^2}k^4 + \left(\frac{5\cdot 3\cdot 1}{2\cdot 2\cdot 2}\right)^2 \frac{1}{3!^2}k^6 + ...\right] \tag{3.20}$$

Running this in python with N terms in the summation gives

N	5	10	20	30	40
Estimate	7.3955	7.4160	7.4163	7.4163	7.4163
Error	0.0208	$3.5{\cdot}10^{-4}$	$1.8{\cdot}10^{-7}$	$8.9 \cdot 10^{-14}$	$8.9 \cdot 10^{-16}$

N	50	60	70	80
Estimate	7.4163	7.4163	7.4163	7.4163
Error	$8.9 \cdot 10^{-16}$	$8.9 \cdot 10^{-16}$	$8.9 \cdot 10^{-16}$	$8.9 \cdot 10^{-16}$

Table 3.3: Sequence of approximations for T($\frac{\pi}{2}$), using a series expansion

After $N = 40$ there is no improvement in the estimation, thus is likely due to the limitations on working with large decimals in python. And is shown by the plateau in figure (3.4).

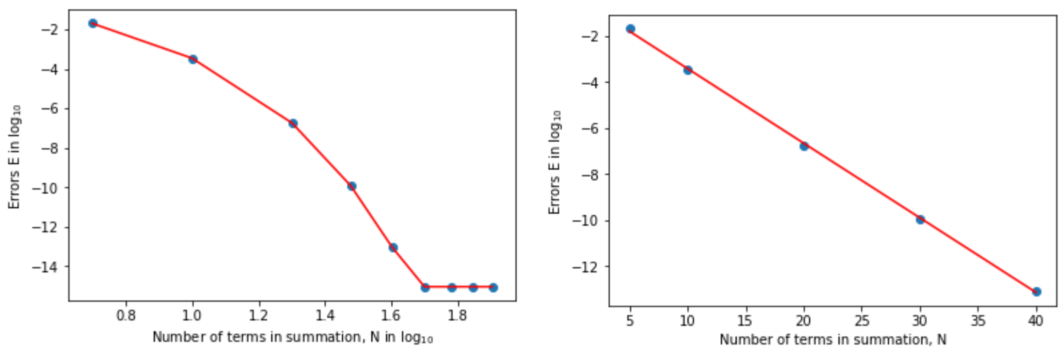

Figure 3.4: Graph of errors for Integrals Evaluation Estimates in $\log_{10}$, Graph of errors with $\log_{10}(E)$ against N

In log this graph does not form a line therefore the errors are scaled as N^{-p} as in previous example, however when plotting $\log_{10}(E)$ against N, it result in a linear graph of the form $E = Be^{aN}$ with a equal to the gradient of $a = 0.3241$ when discarding $N > 40$.
Since this method is the most accurate out of those investigated in this chapter, using it approximate T for values where the exact solution may not be known $a = [\frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{5\pi}{6}]$ and plotting T as a function of a, showing how it increases from 2π.

a	$\frac{\pi}{6}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	$\frac{2\pi}{3}$	$\frac{5\pi}{6}$
T(a)	6.3926	6.7430	7.4163	8.6261	11.0360

Table 3.4: Approximations of T(a) for varying a, using 3.14

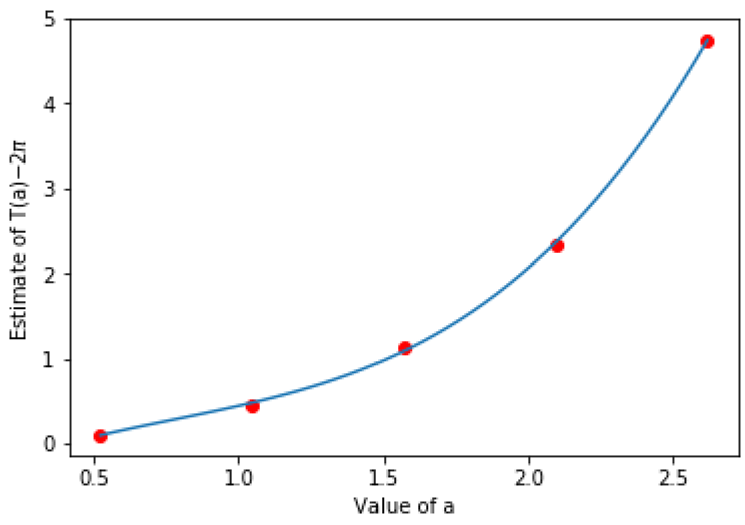

Figure 3.5: Estimates of T(a)-2π

3.8 Discussion

For the data in Table (3.4) it considers two angles smaller than $\frac{\pi}{2}$ and two larger. Since we already know that for the value for $a = \frac{\pi}{2}$ is accurate as calculated in Section (3.5), the other values calculated for the other values of a are estimates. From Section (3.3) we know that for small values of a, $T \approx 2\pi = 6.2832$, looking at the smaller angles $a = [\frac{\pi}{6}, \frac{\pi}{3}]$, the corresponding $T(a)$ values are close to 2π by $[0.1094, 0.4598]$ therefore I am inclined to believe these are good approximate estimates for $T(a)$.
For studying the Period of a pendulum at large angles there are other formula which produce a reasonably low error. For example from [14][489-490] and [15][pg 5], by an interpolation-like linearisation on Equation (3.1) resulting in an formula

$$T_M = T_0 \left(\frac{\sin \theta_0}{\theta_0} \right)^{\frac{3}{8}}$$

Which has an error of $\approx 0.4\%$ for $\theta_0 = \frac{\pi}{2}$. I would next investigate whether there are particular approximations or methods which focus on solving the for the period of a pendulum when θ_0 is large. Also find some results from practical experiments for these pendulum cases to see how to how theoretical side matches the outcomes.

Bibliography

[1] W.L. Dunn and J.K. Shultis. *Exploring Monte Carlo Methods.* Elsevier Science, 2011.

[2] Keith Enevoldsen. N-dimensions. `https://thinkzone.wlonk.com/MathFun/Dimens.htm`, 2003.

[3] G. Leobacher and F. Pillichshammer. *Introduction to Quasi-Monte Carlo Integration and Applications.* Compact Textbooks in Mathematics. Springer International Publishing, 2014.

[4] R.H. Landau, M.J. Páez, and C.C. Bordeianu. *Computational Physics: Problem Solving with Python.* EBL-Schweitzer. Wiley, 2015.

[5] Y.M. Desai. *Finite Element Method with Applications in Engineering.* Dorling Kindersley, 2011.

[6] J. Noye. *Computational Techniques for Differential Equations.* ISSN. Elsevier Science, 2000.

[7] E. Zauderer. *Partial Differential Equations of Applied Mathematics.* Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley, 2011.

[8] J.H. Davis. *Methods of Applied Mathematics with a MATLAB Overview.* Applied and Numerical Harmonic Analysis. Birkhäuser Boston, 2004.

[9] Galileo. `https://www.oxfordreference.com/view/10.1093/acref/9780199546091.001.0001/acref-9780199546091-e-4448`, 2004.

[10] Denys Vaughan. pendulum. `https://www.oxfordreference.com/view/10.1093/acref/9780195112290.001.0001/acref-9780195112290-e-0559`, 2003.

[11] World encyclopedia. `https://www.oxfordreference.com/view/10.1093/acref/9780199546091.001.0001/acref-9780199546091-e-8840`, 2004.

[12] W.E. Deming and USDA. Graduate School. *The Gamma and Beta Functions: Notes and Problems Designed for Use in Mathematical Statistics and Mathematical Physics.* Graduate School, Department of Agriculture, 1944.

[13] Dr. George Feinstein. Impulsing the pendulum: Escapement error. `www.nawcc-index.net/Articles/Feinstein-ImpulsingThePendulum.pdf`, 2005.

[14] M. I. Molina. *Simple linearization of the simple pendulum for any amplitude.* Phys. Teach. 35, 1997.

[15] P. Arun F. M. S. Lima. An accurate formula for the period of a simple pendulum oscillating beyond the small angle regime. `https://arxiv.org/pdf/physics/0510206.pdf`, 05/07/2006.

Appendix

My Functions

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 11 14:50:59 2019

@author: ashle
"""
import random as ra
import numpy as nm
import matplotlib.pyplot as plt
import MyFunctions as mf
import math as ma
import scipy.special as ss
import sympy as s
import sympy.utilities as uti
import scipy.optimize as opt
import cmath as cm
def mean(a):
    c=0
    for i in a:
        c=c+i
    mean=c/(len(a))
    return mean
def samvar(a):
    d=0
    e=0
    for i in list(range(0,len(a))):
        e=(a[i]-mean(a))**2
        d=d+e
    j=len(a)- 1
    variance=d/j
    return variance
def samstan(a):
    standard=nm.sqrt(samvar(a))
    return standard
def normalsample(a,b,N):
    i, sample = 0, []
    while i in list(range(0,N)):
        si=ra.uniform(a,b)
        i=i+1
        sample.append(si)
    return sample
def simps(f,a,b,N):
    if N % 2 == 1:
        raise ValueError
    dx = (b-a)/N
    x = nm.linspace(a,b,N+1)
    y = f(x)
    S = dx/3 * nm.sum(y[0:-1:2] + 4*y[1::2] + y[2::2])
    return S
def best_fit(X,Y):
    xbar=sum(X)/len(X)
    ybar=sum(Y)/len(Y)
    n= len(X)
    numer = sum([xi*yi for xi, yi in zip(X,Y)]) - n*xbar*ybar
    denum = sum([xi**2 for xi in X]) - n*xbar**2
    b=numer/denum
    a= ybar - b*xbar
    return((a,b))
def dim2(radius,N,Ans=None):
    ER,ES=[],[]
    Ans=nm.pi
    a,b=0,radius
    for n in N:
        inside=0
        sample1=mf.normalsample(a,b,n)
        sample2=mf.normalsample(a,b,n)
        i=0
        for i in list(range(0,n-1)):
            if sample1[i]**2 +sample2[i]**2 <=1:
                inside=inside+1
        ae=(inside/n)*4
        ES.append(ae)
        ER.append(nm.abs(ae-Ans))
    return ER
def dim3(radius,N,Ans=None):
    ES,ER=[],[]
    Ans=(4/3)*nm.pi
    a,b=0,radius
    for n in N:
        inside=0
        sample1=mf.normalsample(a,b,n)
        sample2=mf.normalsample(a,b,n)
        sample3=mf.normalsample(a,b,n)
        i=0
        for i in list(range(0,n-1)):
            if sample1[i]**2 +sample2[i]**2 +sample3[i]**2 <=1:
                inside=inside+1
        ae=(inside/n)*8
        ES.append(ae)
        ER.append(nm.abs(ae-Ans))
    return [ER,ES]
def dim4(radius,N,Ans=None):
    ES,ER=[],[]
    Ans=(1/2)*nm.pi*nm.pi
    a,b=0,radius
    for n in N:
        inside=0
        sample1=mf.normalsample(a,b,n)
        sample2=mf.normalsample(a,b,n)
        sample3=mf.normalsample(a,b,n)
        sample4=mf.normalsample(a,b,n)
        i=0
        for i in list(range(0,n-1)):
            if sample1[i]**2 +sample2[i]**2 +sample3[i]**2 +sample4[i]**2 <=1:
                inside=inside+1
        ae=(inside/n)*16
        ES.append(ae)
        ER.append(nm.abs(ae-Ans))
    return [ER, ES]
def doubfact(n):
    if n <=0:
        return 1
    else:
        return n*doubfact(n-2)
```

Coding

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 16 11:22:41 2020

@author: ashle
"""
#Figure 1.1
y=[]
x=mf.normalsample(0,1.75,5)
for i in list(range(0,len(x))):
    fx=-2*(x[i]**2) +1
    y.append(fx)
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
v=nm.linspace(0,1.75,1000)
p=-2*(1.75**2)+1
plt.plot(v, -2*(v**2)+1, 'k')
plt.scatter(x,y)
for i in list(range(0,len(y))):
    plt.plot([0,x[i]], [y[i],y[i]], linestyle='dotted')
    plt.plot([x[i],x[i]],[p,y[i]], linestyle='dotted')
yu=sum(y)/len(y)
plt.plot([0,1.75],[yu,yu],'c')
plt.plot([0,0],[yu,-5.1],'c')
plt.plot([1.75,1.75],[yu,-5.1],'c')
plt.show()

#Monte Carlo for Figure 1.2
a, b, N = 0, 2, 10000
def Monte_carlo_cos(a,b,N,Ans):
    sample=mf.normalsample(a,b,N)
    f=0
    for i in list(range(0,N)):
        y=0.5*(nm.cos(sample[i]))**2
        f=f+y
    Monte_estimate=((b-a)/N)*f
    Error_estimate1=nm.abs(Monte_estimate - Ans)
    return(Monte_estimate, Error_estimate1)
E=[]
G=[]
CCC=[]
```

```
Ans=0.5*0.5*(0.5*nm.sin(4)+2)
n=[100,200,400,800,1600,3200,6400]
C=list(range(0,len(n)))
for i in C:
    e=Monte_carlo_cos(0,2,n[i],Ans)
    D=nm.log10(e[1])
    E.append(D)
    F=nm.log10(e[0])
    G.append(F)
for i in list(range(0,len(n))):
    CC=nm.log10(n[i])
    CCC.append(CC)
BF=mf.best_fit(CCC,E)
plt.scatter(CCC,E,s=10)
yfit = [BF[0] +BF[1]*xi for xi in CCC]
plt.plot(CCC, yfit , c='r')
plt.xlabel(r'Sample size N in log${}_{10}$')
plt.ylabel(r'Errors E in log${}_{10}$')
print(-BF[1])

#Trapezium for Figure 1.2
I,ares=0,[]
y,N,Y,a=[],[100,200,400,800,1600,3200,6400],[],(2)
ans=(0.5*0.5*(0.5*nm.sin(4)+2))
for j in list(range(0,len(N))):
    x=list(nm.linspace(0,a,N[j]))
    for i in list(range(0,len(x))):
        Yy=0.5*(nm.cos(x[i])**2)
        Y.append(Yy)
    area=0.5*(nm.abs(x[1]-x[0]))*((Y[0]+Y[len(Y)-1])+2*(sum(Y)-Y[0]-Y[len(Y)-1
    ares.append(area)
    x,Y=[],[]
PP,pp,nn=[],[],[]
for i in list(range(0,len(ares))):
    P=nm.abs(ans-ares[i])
    PP.append(P)
    p=nm.log10(P)
    n=nm.log10(N[i])
    nn.append(n)
    pp.append(p)
plt.scatter(nn,pp,s=10)
```

```
BF=mf.best_fit(nn,pp)
 yfit = [BF[0] +BF[1]*xi for xi in nn]
 plt.plot(nn, yfit , c='r')
 plt.xlabel(r'Sample size N in log${}_{10}$')
 plt.ylabel(r'Errors E in log${}_{10}$')
 print(-BF[1])

#Monte Carlo for Circle Figure 1.3
a, b, N = 0, 1, [12800]
 for q in list(range(0,len(N))):
     sample1=mf.normalsample(a,b,N[q])
     sample2=mf.normalsample(a,b,N[q])
     plt.scatter(sample1,sample2,s=5)
X=list(nm.linspace(0,1,100))
Y=[]
 for x in list(range(0,len(X))):
     y=nm.sqrt(1-X[x]**2)
     Y.append(y)
 plt.plot(X,Y,'r')
 plt.xlabel('x')
 plt.ylabel('y')

#Table 1.1 and Figure 1.4
P=list(range(0,50))
Pe,Nn,Ee,EE=[],[],[],[]
a, b, N = 0, 1, [100,200,400,800,1600,3200,6400,12800]
 for q in list(range(0,len(N))):
     sample1=mf.normalsample(a,b,N[q])
     sample2=mf.normalsample(a,b,N[q])
     i=0
     M=N[q]
     while i <= M-1:
          if sample1[i]**2 +sample2[i]**2 >=1:
               sample1.remove(sample1[i])
               sample2.remove(sample2[i])
               i=i
               M=M-1
          else:
               i=i+1
     Ans=nm.pi
     pi_estimate=(len(sample1)/N[q])*4
```

```
    E2=nm.abs(pi_estimate-Ans)
    Pe.append(pi_estimate)
    Ee.append(E2)
    EE.append(nm.log10(E2))
    Nn.append(nm.log10(N[q]))
BF=mf.best_fit(Nn,EE)
plt.scatter(Nn,EE,s=10)
yfit = [BF[0] +BF[1]*xi for xi in Nn]
plt.plot(Nn, yfit, c='r')
plt.xlabel('Sample Size N in log${}_{10}$')
plt.ylabel('Errors E in log${}_{10}$')

#trying to remove outliers from a Monte Carlo average approximation
a, b, N, Pe, Ee = 0, 1, 10000,[],[]
q=0
while q<100:
    sample1=mf.normalsample(a,b,N)
    sample2=mf.normalsample(a,b,N)
    i=0
    M=N
    while i <= M-1:
        if sample1[i]**2 +sample2[i]**2 >=1:
            sample1.remove(sample1[i])
            sample2.remove(sample2[i])
            i=i
            M=M-1
        else:
            i=i+1
    Ans=nm.pi
    pi_estimate=(len(sample1)/N)*4
    E2=nm.abs(pi_estimate-Ans)
    Pe.append(pi_estimate)
    Ee.append(E2)
    q=q+1
s=mf.samstan(Pe)
mean=sum(Pe)/len(Pe)
um=mean+2*s
lm=mean-2*s
ans=[]
for i in list(range(0,len(Pe))):
    if Pe[i]<um:
        if Pe[i]>lm:
            ans.append(Pe[i])
mans=sum(ans)/len(ans)

#Table 1.2 and Figure 1.5
a, b= 0, 1
N=[100,200,400,800,1600,3200,6400,12800]
ER,ES=[],[]
DIM=mf.dim3(1,N)
Nn,erer,=[],[]
for i in list(range(0,len(DIM[0]))):
    nn=nm.log10(N[i])
    er=nm.log10(DIM[0][i])
    Nn.append(nn)
    erer.append(er)
plt.scatter(Nn,erer)
BF=mf.best_fit(Nn,erer)
yfit = [BF[0] +BF[1]*xi for xi in Nn]
plt.plot(Nn,yfit,'r')
plt.xlabel(r'Sample size N in log${}_{10}$')
plt.ylabel(r'Errors E in log${}_{10}$')

#Table 1.3 and Figure 1.6
a, b= 0, 1
N=[100,200,400,800,1600,3200,6400,12800]
ER,ES=[],[]
DIM=mf.dim4(1,N)
Nn,erer,=[],[]
for i in list(range(0,len(DIM[0]))):
    nn=nm.log10(N[i])
    er=nm.log10(DIM[0][i])
    Nn.append(nn)
    erer.append(er)
plt.scatter(Nn,erer)
BF=mf.best_fit(Nn,erer)
yfit = [BF[0] +BF[1]*xi for xi in Nn]
plt.plot(Nn,yfit,'r')
plt.xlabel(r'Sample size N in log${}_{10}$')
plt.ylabel(r'Errors E in log${}_{10}$')

#First four zeros
```

```
A=list(ss.jn_zeros(0,4))

#Figure 2.1
C=[0,0,0,0]
B=ss.j0(A)
X=list(nm.linspace(0,15,100))
Y=[]
for i in list(range(0,len(X))):
    y=ss.j0(X[i])
    Y.append(y)
plt.plot(X,Y)
plt.plot(A[0],[0],'co')
plt.plot(A[1],[0],'go')
plt.plot(A[2],[0],'ro')
plt.plot(A[3],[0],'mo')
plt.xlabel(r'x')
plt.ylabel(r'$J_0(x)$')

#Figure 2.1
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[0]*X[i]))
plt.plot(X,Y,'c', label='First Zero')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[1]*X[i]))
plt.plot(X,Y,'g',label='Second_Zero')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[2]*X[i]))
plt.plot(X,Y,'r',label='Third_Zero')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[3]*X[i]))
plt.plot(X,Y,'m', label='Fourth_Zero')
plt.legend(['First Zero','Second Zero','Third Zero','Fourth Zero'])
plt.xlabel('$x$')
plt.ylabel('$y(x)$')

#calculating I_t and J_t
x,a,b,c=s.symbols('x a b c')
Y=a*x**6+b*x**4+c*x**2+1
I=x*s.diff(Y,x)**2
I=s.integrate(Yy,(x,0,1))
J=x*Y**2
J=s.integrate(Yy,(x,0,1))

#for a=0
a=0
F=I/J
print(F)

#Figure 2.2
x,b=s.symbols('x b')
c=-1-b
Y=b*x**4+c*x**2 +1
Yy=x*s.diff(Y,x)**2
I=s.integrate(Yy,(x,0,1))
c=-1-b
Y=b*x**4+c*x**2 +1
Yy=x*Y**2
J=s.integrate(Yy,(x,0,1))
F=I/J
F=(b**2/3 - 2*b/3 + 1)/(b**2/60 - b/12 + 1/6)
B=list(nm.linspace(-20,20,100))
f=[]
for bb in list(range(0,len(B))):
    Ff=(B[bb]**2/3 - 2*B[bb]/3 +1)/(B[bb]**2/60 - B[bb]/12 +1/6)
    f.append(Ff)
plt.plot(B,f)
plt.xlabel('$b$')
plt.ylabel('$I/J$')

#Figure 2.3
```

```
x=list(nm.linspace(0,1,100))
y=[]
for X in list(range(0,len(x))):
    Y=0.4*x[X]**4 + (-1-0.4)*x[X]**2 +1
    y.append(Y)
plt.plot(x,y, label='Trial Function')
plt.xlabel('$x$')
plt.ylabel('$y$')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[0]*X[i]))
plt.plot(X,Y,'--', label='First Zero')
plt.legend(['Trial Function','First Zero'])
plt.xlabel('$x$')
plt.ylabel('$y(x)$')
A=list(ss.jn_zeros(0,4))
x=list(nm.linspace(0,1,100))
y=[]
for X in list(range(0,len(x))):
    Yy=(0.4*x[X]**4 + (-1-0.4)*x[X]**2 +1)
    yY=ss.j0(A[0]*x[X])
    Y=abs(Yy-yY)
    y.append(Y)
plt.plot(x,y)
plt.xlabel('$x$')
plt.ylabel('Error')

#re-introducing a
x,a,b=s.symbols('x a b')
c=-1-a-b
Y=a*x**6+b*x**4+c*x**2 +1
Yy=x*s.diff(Y,x)**2
I=s.integrate(Yy,(x,0,1))
yY=x*Y**2
J=s.integrate(yY,(x,0,1))
F=I/J

#Critical points of F
a,b=s.symbols('a b')
```

```
I=(a**2 + 17*a*b/15 - a + b**2/3 - 2*b/3 + 1)
J=(4*a**2/105 + a*b/20 - 7*a/60 + b**2/60 - b/12 + 1/6)
F=I/J
Fa=s.simplify(s.diff(F,a))
Fb=s.simplify(s.diff(F,b))
eq1=s.Eq(Fa)
eq2=s.Eq(Fb)
cp1=s.solve((eq1,eq2),a,b)

#minimising F to a and b
A=list(nm.linspace(-10,10,200))
B=list(nm.linspace(-10,10,200))
AB=[]
ab=[]
a,b=s.symbols('a b')
I=(a**2 + 17*a*b/15 - a + b**2/3 - 2*b/3 + 1)
J=(4*a**2/105 + a*b/20 - 7*a/60 + b**2/60 - b/12 + 1/6)
F=I/J
myfunc=uti.lambdify((a,b),F)
def myfuncv(x):
    return myfunc(*tuple(x))
q=opt.minimize(myfuncv,[1,1])
minF=q['fun']
minab=q['x']

#Figure 2.4
x=list(nm.linspace(0,1,100))
y=[]
minab=list(q['x'])
a=minab[0]
b=minab[1]
for X in list(range(0,len(x))):
    Y=a*x[X]**6+b*x[X]**4 + (-1-a-b)*x[X]**2 +1
    y.append(Y)
plt.plot(x,y,label='Trial Function')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[0]*X[i]))
plt.plot(X,Y,'--',label='First Zero')
plt.legend(['Trial Function','First Zero'])
plt.xlabel('$x$')
plt.ylabel('$y(x)$')
A=list(ss.jn_zeros(0,4))
x=list(nm.linspace(0,1,100))
y=[]
for X in list(range(0,len(x))):
    Yy=a*x[X]**6+b*x[X]**4 + (-1-a-b)*x[X]**2 +1
    yY=ss.j0(A[0]*x[X])
    Y=abs(Yy-yY)
    y.append(Y)
plt.plot(x,y)
plt.xlabel('$x$')
plt.ylabel('Error')

#contour plot Figure 2.5
x=list(nm.linspace(0,1,100))
y=[]
a=-3.98149934231607
b=9.78609192864497
for X in list(range(0,len(x))):
    Y=a*x[X]**6+b*x[X]**4 + (-1-a-b)*x[X]**2 +1
    y.append(Y)
plt.plot(x,y,label='Saddle Point Trial Function')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[1]*X[i]))
plt.plot(X,Y,'--',label='First Zero')
plt.legend(['Saddle Point Trial Function','First Zero'])
plt.xlabel('$x$')
plt.ylabel('$y(x)$')
A=list(ss.jn_zeros(0,4))
x=list(nm.linspace(0,1,100))
y=[]
for X in list(range(0,len(x))):
    Yy=a*x[X]**6+b*x[X]**4 + (-1-a-b)*x[X]**2 +1
    yY=ss.j0(A[1]*x[X])
    Y=abs(Yy-yY)
    y.append(Y)
```

```
plt.plot(x,y)
plt.xlabel('$x$')
plt.ylabel('Error')

#Figure 2.6
x=list(nm.linspace(0,1,100))
y=[]
a,b=-19.3659,31.0412

for X in list(range(0,len(x))):
    Y=a*x[X]**6+b*x[X]**4 + (-1-a-b)*x[X]**2 +1
    y.append(Y)
plt.plot(x,y,label='$y_3(x)$')
A=list(ss.jn_zeros(0,4))
x=list(nm.linspace(0,1,100))
y=[]
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[0]*X[i]))
plt.plot(X,Y,'--', label='First Zero')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[1]*X[i]))
plt.plot(X,Y,'--',label='Second_Zero')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[2]*X[i]))
plt.plot(X,Y,'--',label='Third_Zero')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
    Y.append(ss.j0(A[3]*X[i]))
plt.plot(X,Y,'--', label='Fourth_Zero')
plt.legend(['$y_3(x)$','First Zero','Second Zero','Third Zero','Fourth Zero'])
plt.xlabel('$x$')
plt.ylabel('$y$')

#Figure 2.8
x,a,b,c,d=s.symbols('x a b c d')
Y=s.cos(nm.pi*x/2)+a*(s.cos(3*nm.pi*x/2))+b*(s.cos(nm.pi*5*x/2))
I=x*s.diff(Y,x)**2
I=s.integrate(I,(x,0,1))
J=x*Y**2
J=s.integrate(J,(x,0,1))
F=I/J
F=s.simplify(F)
Fa=s.diff(F,a)
Fb=s.diff(F,b)
eq1=s.Eq(Fa)
eq2=s.Eq(Fb)
cp1=s.solve((eq1,eq2),a,b)
A=list(nm.linspace(-10,10,200))
B=list(nm.linspace(-10,10,200))
AB=[]
ab=[]
a,b=s.symbols('a b')
F=(5.80165247561276*a**2 - 7.5*a*b - 1.5*a + 15.6712568767021*b**2 + 0.27777777
myfunc=uti.lambdify((a,b),F)
def myfuncv(x):
     return myfunc(*tuple(x))
q=opt.minimize(myfuncv,[1,1])
x=list(nm.linspace(0,1,100))
y=[]
a,b=0.0318793 , -0.00724002
for X in list(range(0,len(x))):
     Y=s.cos(nm.pi*x[X]/2)+a*(s.cos(3*nm.pi*x[X]/2))+b*(s.cos(nm.pi*5*x[X]/2))
     y.append(Y)
plt.plot(x,y,label='$y_(x)$')
A=list(ss.jn_zeros(0,4))
X=list(nm.linspace(0,1,100))
Y=[]
for i in list(range(0,len(X))):
     Y.append(ss.j0(A[0]*X[i]))
plt.plot(X,Y,'--', label='First Zero')
plt.legend(['$y_t(x)$','First Zero'])
plt.xlabel('$x$')
plt.ylabel('$y$')

y=[]
for X in list(range(0,len(x))):
    Yy=s.cos(nm.pi*x[X]/2)+a*(s.cos(3*nm.pi*x[X]/2))+b*(s.cos(nm.pi*5*x[X]/2))
    yY=ss.j0(A[0]*x[X])
    Y=abs(Yy-yY)
    y.append(Y)
plt.plot(x,y)
plt.xlabel('$x$')
plt.ylabel('Error')

#Gamma for T=pi/2
ss.gamma(0.25)**2/(nm.sqrt(nm.pi))

#Table 3.1 and Figure 3.2
ares=[]
y,N,Y,a=[],[100,200,400,800,1600,3200,6400],[],(nm.pi/2)
ans=((ss.gamma(0.25)**2)/nm.sqrt(nm.pi))
for j in list(range(0,len(N))):
    x=list(nm.linspace(0,a,N[j]+1))
    for i in list(range(0,len(x)-1)):
        Yy=(1/nm.sqrt(nm.cos(x[i])-nm.cos(x[len(x)-1])))*2*nm.sqrt(2)
        Y.append(Yy)
    area=0.5*(nm.abs(x[1]-x[0]))*((Y[0]+Y[len(Y)-1])+2*(sum(Y)-Y[0]-Y[len(Y)-1
    ares.append(area)
    x,Y=[],[]
PP,pp,nn=[],[],[]
for i in list(range(0,len(ares))):
    P=nm.abs(ans-ares[i])
    PP.append(P)
    p=nm.log10(P)
    n=nm.log10(N[i]-1)
    nn.append(n)
    pp.append(p)
plt.scatter(nn,pp,s=10)
BF=mf.best_fit(nn,pp)
yfit = [BF[0] +BF[1]*xi for xi in nn]
plt.plot(nn, yfit, c='r')
plt.xlabel(r'Number of Intervals N in log${}_{10}$')
plt.ylabel(r'Errors E in log${}_{10}$')
print(-BF[1])

#Table 3.2 and Figure 3.3
ares=[]
y,N,Y,a=[],[100,200,400,800,1600,3200,6400],[],(nm.pi/2)
ans=((ss.gamma(0.25)**2)/nm.sqrt(nm.pi))
for j in list(range(0,len(N))):
    x=list(nm.linspace(0,a,N[j]+1))
    for i in list(range(0,len(x)-1)):
        Yy=((1/nm.sqrt(nm.cos(x[i])-nm.cos(a)))-
            (1/(nm.sqrt(nm.sin(a)*(a-x[i])))))*2*nm.sqrt(2)
        Y.append(Yy)
    area=4*nm.sqrt((2*a)/(nm.sin(a)))+0.5*(nm.abs(x[1]-x[0]))*
            ((Y[0]+Y[len(Y)-1])+2*(sum(Y)-Y[0]-Y[len(Y)-1]))
    ares.append(area)
    x,Y=[],[]
PP,pp,nn=[],[],[]
for i in list(range(0,len(ares))):
    P=nm.abs(ans-ares[i])
    PP.append(P)
    p=nm.log10(P)
    n=nm.log10(N[i]-1)
    nn.append(n)
    pp.append(p)
plt.scatter(nn,pp,s=10)
BF=mf.best_fit(nn,pp)
yfit = [BF[0] +BF[1]*xi for xi in nn]
plt.plot(nn, yfit, c='r')
plt.xlabel(r'Number of Intervals N in log${}_{10}$')
plt.ylabel(r'Errors E in log${}_{10}$')
print(-BF[1])

#Table 3.3 and Figure 3.4
T=(ss.gamma(1/4)**2)/nm.sqrt(nm.pi)
k=nm.sin(nm.pi/4)
N=[5,10,20,30,40,50,60,70,80]
a,e=[],[]
for i in N:
    KK,Kk=0,0
    for m in list(range(0,i)):
            K=(nm.pi/2)*(((doubfact(2*m-1)
                 /(ma.factorial(m)*(2**m))))**2)*(k**(2*m))
            KK=KK+K
        Kk=4*KK
        a.append(Kk)
        e.append(nm.abs(Kk-T))
n,L=[],[]
for l in list(range(0,len(a))):
        n.append(nm.log10(N[l]))
        L.append(nm.log10(e[l]))
plt.scatter(n,L)
BF=mf.best_fit(n,L)
yfit = [BF[0] +BF[1]*xi for xi in n]
plt.plot(n,L,'r')
plt.xlabel(r'Number of terms in summation, N in log${}_{10}$')
plt.ylabel(r'Errors E in log${}_{10}$')

T=(ss.gamma(1/4)**2)/nm.sqrt(nm.pi)
k=nm.sin(nm.pi/4)
N=[5,10,20,30,40]
a,e=[],[]
for i in N:
        KK,Kk=0,0
        for m in list(range(0,i)):
            K=(nm.pi/2)*(((doubfact(2*m-1)
                 /(ma.factorial(m)*(2**m))))**2)*(k**(2*m))
            KK=KK+K
        Kk=4*KK
        a.append(Kk)
        e.append(nm.abs(Kk-T))
n,L=[],[]
for l in list(range(0,len(a))):
        L.append(nm.log10(e[l]))
plt.scatter(N,L)
BF=mf.best_fit(N,L)
yfit = [BF[0] +BF[1]*xi for xi in N]
plt.plot(N,yfit,'r')
plt.xlabel(r'Number of terms in summation, N')
plt.ylabel(r'Errors E in log${}_{10}$')

#Table 3.4 and Figure 3.5
```

```
a,e,A=[nm.pi/6,nm.pi/3,nm.pi/2,2*nm.pi/3,5*nm.pi/6],[],[]
for b in list(range(0,len(a))):
    k=nm.sin(a[b]/2)
    N=40
    KK,Kk=0,0
    for m in list(range(0,N)):
        K=(nm.pi/2)*(((doubfact(2*m-1)/(ma.factorial(m)*(2**m))))**2)*(k**(2*m)
        KK=KK+K
    Kk=4*KK
    A.append(Kk)
    e.append(Kk-2*nm.pi)
plt.scatter(a,e,s=10)
points=nm.array([(a[0],e[0]),(a[1],e[1]),(a[2],e[2])
                ,(a[3],e[3]),(a[4],e[4])])
X=points[:,0]
Y=points[:,1]
Z=nm.polyfit(X,Y,3)
F=nm.poly1d(Z)
Xx=nm.linspace(X[0],X[-1],50)
Yy=F(Xx)
plt.plot(X,Y,'or',Xx,Yy)
plt.xlabel(r'Value of a')
plt.ylabel(r'Estimate of T(a)$-2\pi$')
```