Section3

Elliott Ashby

November 10, 2023

1 q1

```
def f(x):
    return (x ** 4 * ((1 - x) ** 4) / (1 + (x ** 2)))


def trap0(f, a, b, n):
    '''Basic trapezium rule. Integrate f(x) over the
    interval from a to b using n strips.'''
    h: float = (b - a) / n
    s = 0.5 * (f(a) + f(b))
    for i in range(1, n):
```

The above code first defines the integrad, $f(x)$, and then the function trap0, which takes the integrand, and upper and lower bound and the number of strips to use.

2 q2

Calling the functions in the following way can calculate the value of the integral.

```
def q3_1_to_4():
    n = 2
    a = [trap0(f, 0, 1, 1)]
    while len(a) < 2 or not np.fabs((a[-1] - a[-2])) < 1.0e-6:
        if n == 0 or n == 1:
            break
        else:
            a.append(trap0(f, 0, 1, n))
            n += 1

    print('the integrals value is', a[-1])
    print(n)
```

Running this will give an output for the value of:

$$\int_0^1 \frac{x^4(1-x)^4}{1+x^2} dx = 0.0012649570769764054$$

3 q3

Which was achieved using 7 strips.

4 q4

```python
def fe(x):
    return np.e ** (-x ** 2)
```

```python
    print(f"Values for integral of e^'{'x^2'}' over a range of -1, 1 to -10, 10, increasing by 2 each time: {[trap1(fe, -x, x, 1.0e-3) for x in range(1, 11)]}")
```

Which uses the the new traps1 function to calculate the integral.
Continuing to increase this to get precision at 13 sig fig at an integral range of -10 to 10 will calculate:

$$\int_{-a}^{a} e^{-x^2} dx = 1.7724538509055157$$

Where a increases by one every iteration.
Simply running 1 to 10, hence increasing the range of the integral by 2 every time, reveals that results are precise to 12 sig fig at only a range of -5 to 5. This produces an output of:

```
            1.4931691935764426, 1.7639724905315541, 1.7723984788565614, 1.7724537930187396, 1.7724538509008183, 1.7724538509055159, 1.772453850905516, 1.772453850905516, 1.7724538509056167, 1.7724538509055157
```

5 q5

The integrand of:

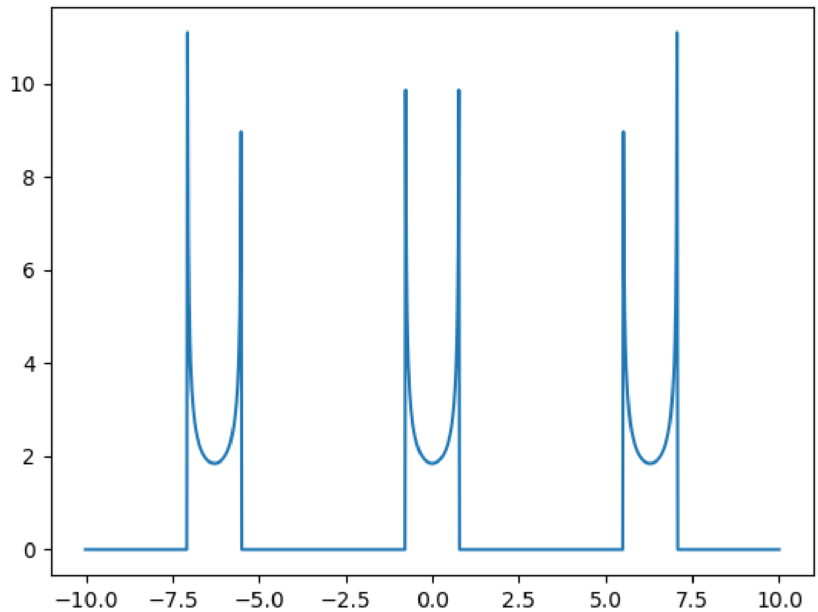
$$T = T_0 \frac{\sqrt{2}}{\pi} \int_0^{\alpha} \frac{d\theta}{(cos\theta - cos\alpha)^{1/2}}$$

is:

$$\frac{1}{(cos\theta - cos\alpha)^{1/2}}$$

```python
def period(a, phi):
    return 1 / ((math.cos(phi) - math.cos(a)) ** 0.5)
```

```python
    x = np.linspace(-10, 10, 1000)
    y = [period(math.pi/4, i) for i in x]

    plt.plot(x, y)
    plt.savefig('3_5.png')
    plt.clf()
```



Plot of the output of the integrand against α

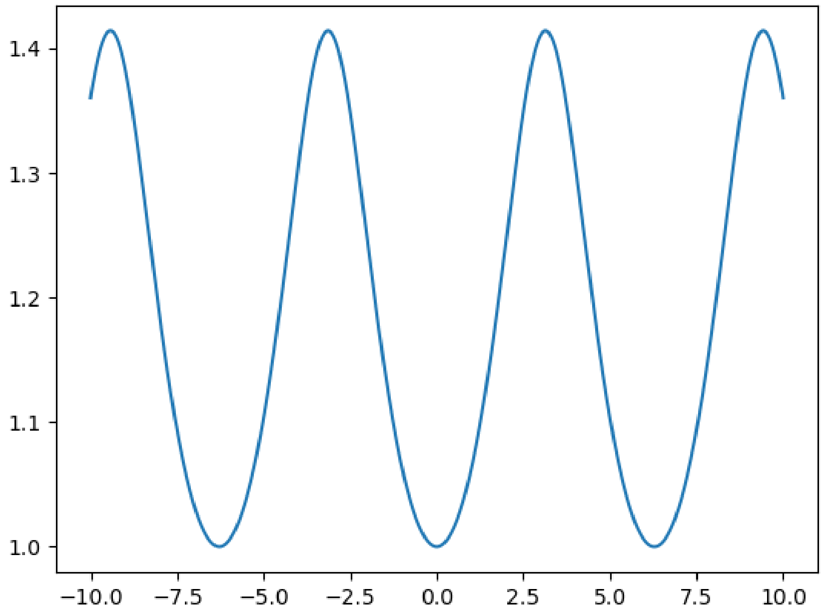
Here, we can see spikes to infinity at values where $\alpha = \theta$. The inconsistent height of the asymptotes are because only 1000 discrete value were calculated to graph, meaning that these asymptotes are only at $\alpha \approx \theta$.

We can rewrite the integral as:

$$\frac{1}{(1-sin^2(\alpha/2)+sin^2\phi)^{1/2}}$$

```
def period1(phi, a):
    return 1 / (1 - (math.sin(a / 2) ** 2) * (math.sin(phi)) ** 2) ** 0.5
```

```
    y1 = [period1(math.pi/4, i) for i in x]

    plt.plot(x, y1)
    plt.savefig('3_5_2.png')
```

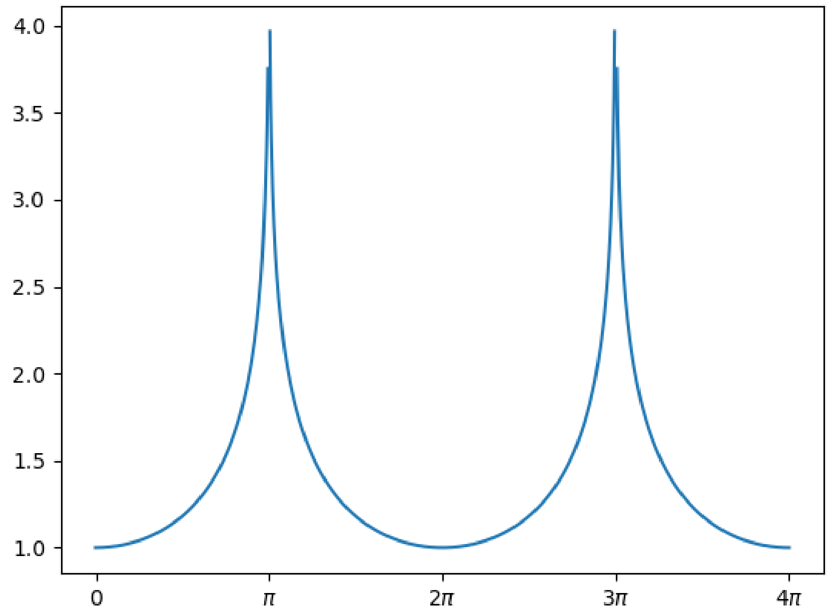


Plot of the output of the integrand against α

6 q6

```
def q3_6():
    global a # required since trap1() uses a single variable whereas period2() uses two, a and phi
    alphas = np.linspace(0, 4 * math.pi, 1000)
    q = []

    def period2(phi):
        return 1 / (1 - (math.sin(a / 2) ** 2) * (math.sin(phi)) ** 2) ** 0.5

    for a in alphas:
        q.append([a, trap1(period2, 0, math.pi / 2, 1.0e-3)])

    for i in q:
        try:
            i[1] = i[1] * 2 / math.pi
        except TypeError:
            pass

    print('alpha \t ratio')
```

```
    for i in q:
        print(f'{i[0]} \t {i[1]}')

    plt.plot(alphas, [i[1] for i in q])
    plt.xticks(ticks=[0, math.pi, 2 * math.pi, 3 * math.
        ↪ pi, 4 * math.pi], labels=['0',
                                    '$\pi$', '2$\pi$', '3$\pi$',
                                        ↪ '4$\pi$'])
    plt.savefig('3_6.png')
```



Plot of T/T_0 against α

7 q7

```
    q1 = q.copy()
    for x in q1:
        x[0] = round(x[0], 2)

    q1 = np.array(q1)
    condition1, condition2 = list(np.where(q1 == 1.57))
    condition1 = int(condition1)
    condition2 = int(condition2 + 1)
    print(f'when amplitude is at roughly pi/2, the value
        ↪ of \
T/T_0 is {q1[condition1][condition2]}')
```

In order to gain a value of T/T_0 at 90° we simply round both $\pi/2$ and our values of alpha to 2 decimal places, find the index of where $\alpha \approx \pi/2$ and print the associated ratio.

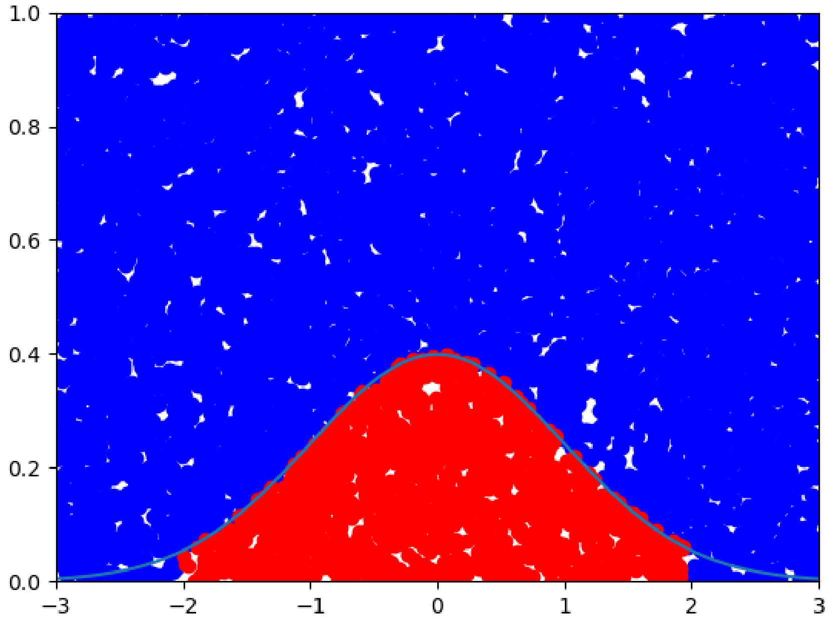
at $\alpha = 90°$:

$$\frac{T}{T_0} = \frac{2}{\pi}\int_0^{\pi/2} \frac{d\phi}{(1 - \sin^2(\alpha/2)\sin^2\phi)^{1/2}} = 1.1807649945835401 \tag{1}$$

8 q8

```python
def q3_8():
    from scipy import stats

    def rand(xmin, ymin, xmax, ymax, n=1000):
        x = np.random.uniform(xmin, xmax, n)
        y = np.random.uniform(ymin, ymax, n)
        return (x, y)

    def norm():
        mu = 0
        variance = 1
        sigma = math.sqrt(variance)
        x = np.linspace(mu - 3*sigma, mu + 3*sigma, 100)
        return (x, stats.norm.pdf(x, mu, sigma))

    samplesize = 10000
    xmin, ymin, xmax, ymax = -3, 0, 3, 1
    x, y = rand(xmin, ymin, xmax, ymax, samplesize)
    count_under = 0
    count_over = 0
    tempa = []
    tempb = []
    for i, x in zip(range(samplesize), x):
        if not y[i] > stats.norm.pdf(x, 0, 1) and -2 < x < 2:
            tempb.append((x, y[i]))
            # plt.scatter(x, y[i], color='red')
            count_under += 1
        else:
            tempa.append((x, y[i]))
            # plt.scatter(x, y[i], color='blue')
            count_over += 1

    plt.scatter(*zip(*tempa), color='blue')
    plt.scatter(*zip(*tempb), color='red')

    print(f'Ratio of points under the curve and within 2 sigma of the mean to all points is {count_under / (count_under + count_over)}')
```

```
    print(f'Area of the curve is {count_under / (count_under + count_over) * 6}')
    plt.plot(*norm())
    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)
```



Random points plotted against a normal distribution, if a points lands within the normal distribution and within 2σ of the mean it is plotted in red, otherwise blue.

Calculating the number of red points and dividing by the total number of points gives a value of:

$$\frac{N_{red}}{N_{total}} = 0.1595 \tag{2}$$

Which, when proportionaly multiplied by the total area of the plot (6), is roughly the same as the area under the normal distribution curve within 2σ of the mean.

$$\frac{N_{red}}{N_{total}} \times 6 = 0.957 \tag{3}$$

Given that the actual value of the area under the normal distribution curve within 2σ of the mean is 0.9545, this is a very good approximation.