*Any code not found directly in question answer can be found at the end of the document.

1. (a) Show that $(1+x)^n = 1 + nx + o(x)$ as $x \to 0$.

$$(1+x)^n = 1 + nx + o(x) \Rightarrow (1+x)^n - nx - 1 = o(x)$$
$$\lim_{x\to 0} \frac{(1+x)^n - nx - 1}{x} = \frac{0}{0}$$

We now apply L'Hopitals rule.

$$\lim_{x\to 0} \frac{(1+x)^n - nx - 1}{x} = \lim_{x\to 0} \frac{n(1+x)^{n-1} - n}{1} = 0$$

So $(1+x)^n - nx - 1 = o(x) \Rightarrow (1+x)^n = 1 + nx + o(x)$.

(b) Show that $x \sin\sqrt{x} = O(x^{3/2})$ as $x \to 0$.

$$\lim_{x\to 0} \frac{x\sin\sqrt{x}}{x^{3/2}} = \lim_{x\to 0} \frac{\sin\sqrt{x}}{\sqrt{x}} = 1$$

1 is a positive constant so $x \sin\sqrt{x} = O(x^{3/2})$ as $x \to 0$.

(c) Show that $e^{-t} = o(\frac{1}{t^2})$ as $t \to \infty$.

$$\lim_{t\to\infty} \frac{e^{-t}}{\frac{1}{t^2}} = \lim_{t\to\infty} \frac{t^2}{e^t} = \lim_{t\to\infty} \frac{2}{e^t} = 0$$

(I applied L'Hopitals twice above), so $e^{-t} = o(\frac{1}{t^2})$ as $t \to \infty$.

(d) Show that $\int_0^\epsilon e^{-x^2}\,dx = O(\epsilon)$ as $\epsilon \to 0$.

$$\lim_{\epsilon\to 0} \frac{\int_0^\epsilon e^{-x^2}\,dx}{\epsilon} = \frac{0}{0}$$

Again, I use L'Hopitals rule, paired with the 1st FTC in the numerator.

$$\lim_{\epsilon\to 0} \frac{e^{\epsilon^2}}{1} = 1$$

1 is a positive constant so $\int_0^\epsilon e^{-x^2}\,dx = O(\epsilon)$ as $\epsilon \to 0$.

2. (a) Find an exact formula for the change in the solution between the exact problem and the perturbed problem Δx.

$$\hat{b} = b + \Delta b = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} \Delta b_1 \\ \Delta b_2 \end{bmatrix}$$

$A\hat{x} = \hat{b} \Rightarrow \hat{x} = A^{-1}\hat{b}$, so we solve $A^{-1}\hat{b}$.

$$A^{-1}\hat{b} = \hat{x} = \begin{bmatrix} 1 - 10^{10} & 10^{10} \\ 1 + 10^{10} & -10^{10} \end{bmatrix} \begin{bmatrix} \Delta b_1 \\ \Delta b_2 \end{bmatrix} = \begin{bmatrix} 1 + \Delta b_1 + (\Delta b_2 - \Delta b_1)10^{10} \\ 1 + \Delta b_1 + (\Delta b_1 - \Delta b_2)10^{10} \end{bmatrix}$$

And the error in our solution is $\|x - \hat{x}\|$ for $x - \hat{x}$ equal to:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 + \Delta b_1 + (\Delta b_2 - \Delta b_1)10^{10} \\ 1 + \Delta b_1 + (\Delta b_1 - \Delta b_2)10^{10} \end{bmatrix} = \begin{bmatrix} (\Delta b_1 - \Delta b_2)10^{10} - \Delta b_1 \\ (\Delta b_2 - \Delta b_1)10^{10} - \Delta b_1 \end{bmatrix}$$

(b) What is the condition number of $\mathbf{A}$?

From NumPy linalg condition number function, $\kappa_{\mathbf{A}} \approx 19999973849$.

(c) Let Δb_1 and Δb_2 be of magnitude 10^{-5}; not necessarily the same value. What is the relative error in the solution? What is the relationship between the relative error, the condition number, and the perturbation. Is the behavior different if the perturbations are the same? Which is more realistic: same value of perturbation or different value of perturbation?

The relative error in the above solution $\hat{x} = \frac{\|x - \hat{x}\|}{\|x\|}$, the euclidean norm would be a good option for calculating these values. The condition number is an upper bound on the ratio of the relative errors in the outputs to the relative errors in the inputs. As the relative error in the inputs, i.e. the perturbations, grow, the relative error of the output by at most $\kappa\hat{b}$. If $\Delta b_1 = \Delta b_2$, then the relative error would be smaller, which can be seen simply by plugging into the above equation for $x - \hat{x}$, but the perturbations are unlikely to take the same value. This is because round-off errors will differ depending on the value we attempt to represent on the computer.

3. (a) What is the relative condition number $\kappa_f(x)$? Are there any values of x for which this is ill-conditioned (for which $\kappa_f(x)$ is very large)?
For $f(x) = e^x - 1$,

$$\kappa_{f(x)} = \left| \frac{xe^x}{e^x - 1} \right| \approx \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

So $f(x)$ is ill-conditioned as $x \to \infty$.

(b) Is this algorithm stable? Justify your answer.
Here we calculate the relative error between similar input values x to determine the stability. Relative error is calculated by treating the NumPy $expm1$ function as the "true" value, and the above function, $exp(x) - 1$, as the measured value. As seen in the table below, as x gets closer to zero (with each increment within factor of 10 of each other) the relative error grows at a much faster rate, meaning the algorithm is not stable.

x	Relative Error
$5 * 10^{-10}$	$9.85696 * 10^{-10}$
$6 * 10^{-10}$	$6.27002 * 10^{-9}$
$7 * 10^{-10}$	$2.97540 * 10^{-8}$
$8 * 10^{-10}$	$4.65661 * 10^{-10}$
$9 * 10^{-10}$	$1.86518 * 10^{-7}$
$10 * 10^{-10}$	$8.26903 * 10^{-8}$

(c) How many correct digits does the algorithm listed above give you? Is this expected?

```
x = 9.999999995000000**-10
print(np.exp(x) - 1)
print(np.expm1(x))
```

outputs values of $1.000000082740371 * 10^{-10}$ and $1.0000000050500004 * 10^{-10}$, so the above algorithm is accurate to eight digits which is inline with what we found above.

(d) Find a polynomial approximation of $f(x)$ that is accurate to 16 digits for $9.999999995000000 * 10^{-10}$.

$$\hat{f}(x) = -1 + \sum_{k=0}^{3} \frac{x^k}{k!} = \sum_{k=1}^{3} \frac{x^k}{k!}$$

(e) Verify that previous answer is correct.
The previous formula was a summation from $k = 1 \ldots n$, but looping through until the relative error for the starting value of $1.0000000050500004 * 10^{-10}$ was less than 10^{-16} yielded $n = 3$, thus verifying our previous algorithm.

4. (a) Write code to evaluate the following sum:

$$S = \sum_{k=1}^{N} \mathbf{t}(k)\mathbf{y}(k)$$

Print the statement "the sum is: S", with the numerical value of S.

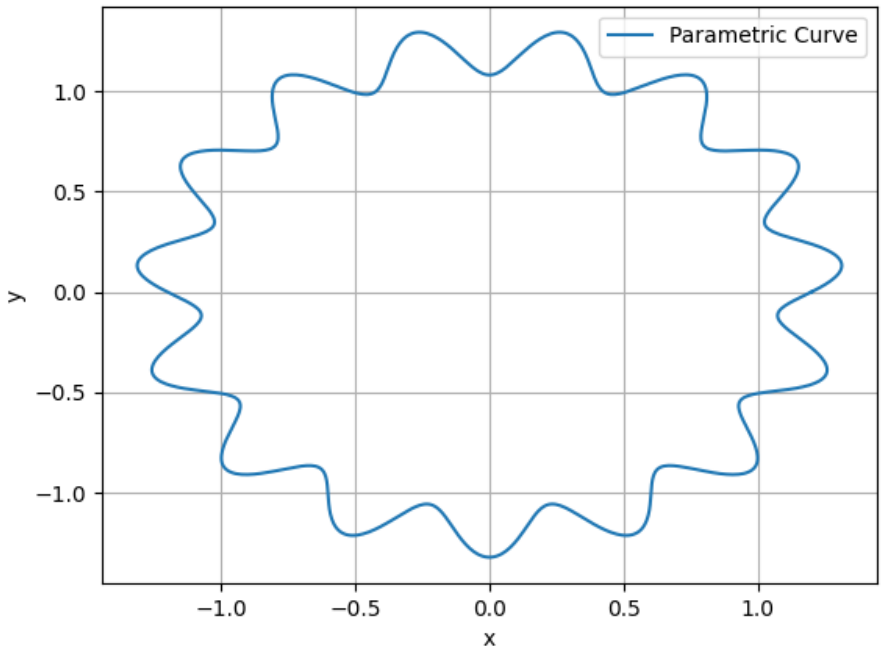
```
def question4A():
    t = np.linspace(0, np.pi, 31)
    y = np.cos(t)
    S = np.dot(t, y)

    print('the sum is:', S)
    return
```

The function prints "the sum is: -20.68685236434684".

(b) In one figure plot the parametric curve

$$x(\theta) = R(1 + \sin(f\theta + p))\cos(\theta)$$
$$y(\theta) = R(1 + \sin(f\theta + p))\sin(\theta)$$

for $0 \leq \theta \leq 2\pi$ and for $R = 1.2$, $\delta r = 0.1$, $f = 15$ and $p = 0$.



In a second figure, use a for loop to plot 10 curves and let with $R = i$, $\delta r = 0.05$, $f = 2 + i$ for the ith curve. Let the value of p be a uniformly distributed random number between 0 and 2.

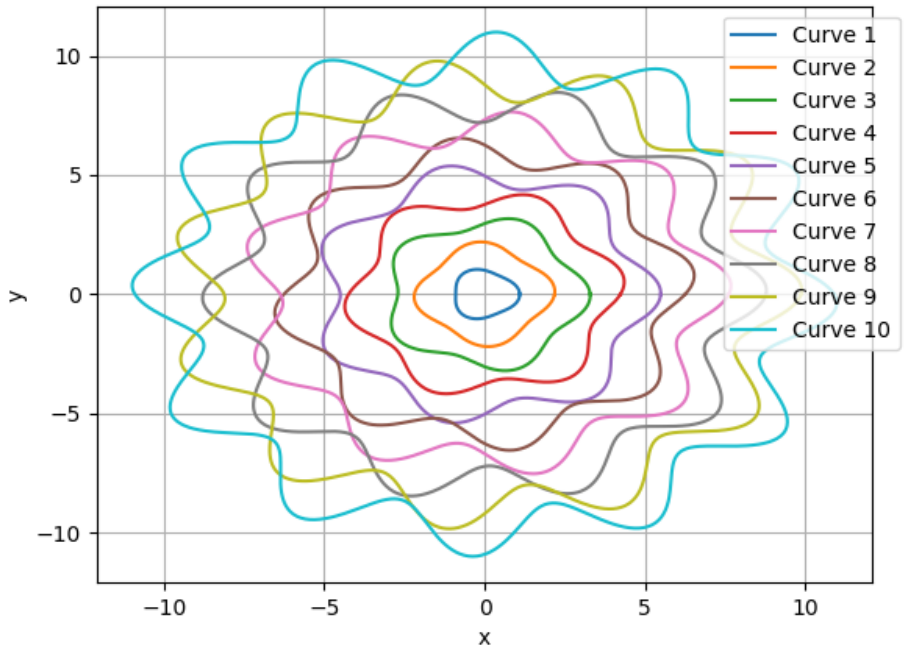


Figure 1: Code for both graphs below.

5. Code Appendix

```
def question2B():
    A = 1 / 2 * np.array([[1, 1], [1 + 10 ** -10, 1 - 10 ** -10]])
    print(LA.cond(A))
    return

def question3B():
    x = 5**-10
    actual = np.expm1(x)
    estimate = np.exp(x) - 1

    for i in range(5, 11, 1):
        x = i ** -10
        actual = np.expm1(x)
        estimate = np.exp(x) - 1
        relError = np.abs(actual - estimate) / np.abs(actual)
        print('Relative Error:', relError)

    return

def question3C():
    S = 0
    n = 1
    x = 9.999999995000000 ** -10
    actual = np.expm1(x)

    while np.abs(actual - S) / np.abs(actual) >= 10 ** -16:
        S += x ** n / math.factorial(n)
        n += 1

    print(n)
    return

def question4B(R=1.2, delta_r=0.1, f=15, p=float(0), theta=0):
    x = R * (1 + delta_r * np.sin(f * theta + p)) * np.cos(theta)
    y = R * (1 + delta_r * np.sin(f * theta + p)) * np.sin(theta)
    return [x, y]
```

```
def Driver4B():
    theta_values = np.linspace(0, 2 * np.pi, 1000)

    for i in range(1, 11):
        p = np.random.uniform(0, 2)

        values = np.array([question4B(i, 0.1, 2 + i, p, theta)
        for theta in theta_values])

        x_values = values[:, 0]
        y_values = values[:, 1]

        plt.plot(x_values, y_values, label=f'Curve {i}')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper right')
    plt.grid(True)
    plt.show()

    return
```