# Mathematics 24/25 Group Project – Group 40

# Task 1

## Task 1.1 - Code:

|  |
| --- |
| import math  import cmath  import numpy as np  import matplotlib as mpl  import matplotlib.pyplot as plt  mpl.rcParams['font.size'] = 20  plt.xlim(-1, 1)  plt.ylim(-1, 1)  plt.axhline(c='k', ls=":")  plt.axvline(c='k', ls=":")  plt.xlabel("Real")  plt.ylabel("Imaginary")  # --- Task 1 ---  # -- 1.1 --  # Setting up the starting complex number  z\_num = (1 + 1j) # Defining the numerator  z\_den = math.sqrt(2) \* (1j\*\*4) \* ((1 + 1j) / abs((1 + 1j)))\*\*40 # Defining the denominator  z\_initial = z\_num / z\_den  # This is the complex number to calculate movement  u = math.sqrt(2) / (1 + 1j)  # Plot the initial point on the graph  plt.plot(z\_initial.real, z\_initial.imag, 'o') # Mark it with a dot  total = 0  z\_position = z\_initial  # We loop 64 times, updating the position each time  for i in range(64):  z\_position /= u # Move to the next position by dividing by u  total += z\_position # Add the new position to the total  plt.plot(z\_position.real, z\_position.imag, 'o', c='r') # Plot it as a red dot  # After all the moves, calculate the average (mean) position  mean = total / 64  # Plot the mean position on the graph  plt.plot(mean.real, mean.imag, 'o', c='g') # Mark it with a green dot  plt.text(0.05, 0.05, "Key Card") # Label it as "Key Card"  # Answer: The key card is at (0, 0) |

**Graph Plot:**

A diagram of a key card

Description automatically generated

## Task 1.2 – Maths:

The first part of solving Task 1.2 is to simplify the . To do this, we take α (our group number: 40) along with *j* (the imaginary number ) and apply it to the below equation:

We can simplify this equation. Take . We know that , so can be shown as:

This then makes our equation:

Next, we take . The complex number can be expressed as . In this case, a and b are both 1. We can then calculate the magnitude of the complex number as shown below:

Our equation is then:

The can be canceled out to give us:

From this point we can perform fraction division (flip and multiply):

Which in turn can be simplified to:

The resulting equation from the above simplification is a power of , where , the complex number to calculate the next position from the previous. The formula to calculate the next positions is as follows:

We see that with each movement, the power decreases by 1. The division by the complex number , is what causes the circular pattern we see when we graph the results. We can take this even further by calculating the angle it is being rotated by.

Using,

Every time we divide by the complex number , we rotate 45° around the origin.

A rotation of 45° means it takes to get back to the initial position, and since she moves 64 times, to get back to the initial position.

**Using 8 movements as an example (64 would yield the same results):**

A graph with red dots and green points

Description automatically generated

Since dividing a position by the complex number gives us the next position, dividing by it 8 times, multiple times will give us an equivalent position, but one where we can do feasible calculations to get the (Re, Ima) coordinates for each movement:

When it comes to adding up all her movements, since she is moving in a circular motion, the total will give us a coordinate at the origin (0,0) and dividing it by the total amount of moves gives us the same answer.

**0 + 1j + 0.7071 + 0.7071j + 1 + 0j + 0.7071 - 0.7071j + 0 – 1j - 0.7071 - 0.7071j - 1 + 0j - 0.7071 + 0.7071j**

**= 0/8 (Mean) = 0**

# Task 2:

## Task 2.1 – Code:

|  |
| --- |
| import math  import cmath  import numpy as np  import matplotlib as mpl  import matplotlib.pyplot as plt  mpl.rcParams['font.size'] = 15  plt.xlim(-2, 2)  plt.ylim(-2, 2)  plt.axhline(c='k', ls=":")  plt.axvline(c='k', ls=":")  plt.xlabel("Real")  plt.ylabel("Imaginary")  # ---Task 2---  # --2.1--  #Setting up the starting complex number  z\_num = math.sqrt(2) \* (1 + 1j) # Defining the numerator  z\_den = ((1 + 1j) / (math.sqrt(2))) \*\* 40 # Defining the denominator  z\_initial = z\_num / z\_den  u = math.sqrt(2) / (1 + 1j) # The complex number to calculate movement  plt.plot(z\_initial.real, z\_initial.imag, 'o', c='b') # Plotting initial point on graph as a blue dot  z\_position = z\_initial  moves = 0  total = 0  # Indefinite loop to determine the amount of moves  while True:  z\_position /= u # Move the point  plt.plot(z\_position.real, z\_position.imag, 'o', c='r') # Plot each move as a red dot  moves += 1 # Increment move counter  total += z\_position # Add current position to total  mean = total / moves # Calculate mean position  # Check if mean position is at origin  if abs(mean.real \* 10).\_\_trunc\_\_() == 0 and abs(mean.imag \* 10).\_\_trunc\_\_() == 0:  plt.plot(z\_initial.real, z\_initial.imag, 'o', c='b') # Clarify initial point in blue  plt.text(z\_initial.real - 1, z\_initial.imag, "Initial Point") # Label initial point  plt.plot(mean.real, mean.imag, 'o', c='g') # Plot keycard in green  plt.text(0.1, 0.1, "Key Card") # Label keycard  break # Exit loop  print(moves) #Outputs 8  #Answer = The minimum number of movements is 8 |

**Graph Plot:**

A graph with red dots and green dots

Description automatically generated

## Task 2.2 – Maths:

First, we simplify this equation:

Again, we can get the next position by dividing by the complex number ,

Here we see that every time we divide by , the exponents of and go down by 1.

Since she moves in the exact same way as **Task 1**, we know that she rotates 45° around the origin every move.

Which means to get to the same coordinates as in **Task 1**, she needs to move until the mean of all her movements gives us the coordinate at the origin.

# Task 3:

## Task 3.1 – Code:

|  |
| --- |
| import math  import cmath  import numpy as np  import matplotlib as mpl  import matplotlib.pyplot as plt  # Setup initial graph plot  mpl.rcParams['font.size'] = 15  plt.xlabel("x")  plt.ylabel("y")  plt.ylim(-10, 2)  # ---Task 3---  # --3.1--  # Define the range for x values from 2.98 to 3.01, with step size of 1e-4  x = np.arange(3 - 2e-2, 3 + 1e-2, 0.0001)  #Setting up the limit  y\_num = (x - 3) \* np.exp(1 / (x - 3)) # Defining the numerator  y\_den = x \* np.exp(-40 \* x) # Defining the Denominator  y = y\_num / y\_den  # Plot the function in blue  plt.plot(x, y, 'b')  # Draw a vertical line at x = 3 in dotted red  plt.axvline(3, c='r', ls=':')  plt.text(3 - 5e-3, 0.5, "(3⁻, 0)") # Adding left limit text  plt.text(3 + 4e-4, 0.5, "(3⁺, +∞)") # Adding right limit text  plt.legend([r'$\frac{(x-3)e^{\frac{1}{x-3}}}{xe^{-40x}}$']) # Function legend for the plot  #Answer = Limit as x approaches 3 does not exist (NaN, as the left-sided limit is zero and the right-sided limit is +∞, which differ from each other), therefore push a Big Red Button |

**Limit Plot:**

A graph with numbers and a line

Description automatically generated

## Task 3.2 – Maths:

Looking at the limit from both sides we get:

, dominates, so we can ignore everything besides it

,dominates, so we can ignore everything besides it

Since the limits vary from both sides, that means the limit as x approaches 3 does not exist (NaN), therefore you should press the Big Red Button.

# Task 4:

## Task 4.1 - Code:

|  |
| --- |
| import math  import cmath  import numpy as np  import matplotlib as mpl  import matplotlib.pyplot as plt  #Setup graph  mpl.rcParams['font.size'] = 12  plt.xlabel("x")  plt.ylabel("y")  plt.ylim(0,100)  #---Task 4---  #--4.1--  x\_initial = -3 \* 40 # Initial x value = -120  x\_final = 3 \* 40 # Final x value = 120  x = np.arange(x\_initial, x\_final, 0.001) # Values of x from initial x to final x with a step of 0.001  #Setting up the function  y\_num = (x\*\*3) - (2 \* 40 \* (x\*\*2)) + ((40\*\*2) \* x) # Defining the numerator  y\_den = (x - 2 \* 40)\*\*2 # Defining the Denominator  y = np.sqrt(y\_num / y\_den)  # Plot the function in red on the graph  plt.plot(x, y, 'r')  # Adding a legend for the function  plt.legend([r'$\sqrt{\frac{x^3 - 80x^2 + 1600x}{(x-80)^2}}$'])  # This calculation below is to find the x-value that makes the function head towards +inf  x = np.arange(0, x\_final, 0.001) # Values of x from 0 to final with a step of 0.001  y\_num = (x\*\*3) - (2 \* 40 \* (x\*\*2)) + ((40\*\*2) \* x)  y\_den = (x - 2 \* 40)\*\*2  y = np.sqrt(y\_num / y\_den) # Calculate the y values  # Iterate through the coordinates to find where y is +inf  for coord in zip(x, y):  if coord[1] == np.inf:  print(f"y({coord[0]}) = +inf") # Print the x value where y is +inf  # Prints y(80.0) = +inf  #Answer = Safe positions: x ∈ [0, 80) ∪ (80, +∞) |

**Function Plot:**

A graph of a function

Description automatically generated

## Task 4.2 - Maths:

First, we turn this implicit equation into a function of :

To determine the safe positions on the x-axis, we set the expression inside the square root to be non-negative:

Meaning the safe positions are values > 0, otherwise they will go into complex numbers.

Moreover, we can solve the numerator,

The numerator indicates that the graph intersects the x-axis at 0 and 40.

We then solve the denominator,

The denominator indicates that when x equals 80, a divide by 0 occurs; the graph heads towards +inf, therefore not being a safe position.

From this we can say the domain of the function is .

# Task 5:

## Task 5.1 - Code:

|  |
| --- |
| import math  import cmath  import numpy as np  import matplotlib as mpl  import matplotlib.pyplot as plt  from scipy.special import comb, perm  #Setup graph  mpl.rcParams['font.size'] = 12  plt.xlabel("x")  plt.ylabel("y")  #---Task 5---  #--5.1--  #Combination  c = comb(40,1) # Equals 40  #Permutation  p = perm(40,1) # Equals 40  #Limit  x = np.linspace(-1,1,1000)  y\_num = math.factorial(40)  y\_den = x  y = -(y\_num/y\_den)  plt.plot(x,y,'g')  plt.legend([r'$\frac{40!}{-x}$'])  #+inf and -inf, therefore not a number  S = {40, 40, np.nan, np.nan} #Despite the set removing duplicates, the set operations still provide the correct results  #Evaluating the states of p, q and r  p = S.intersection({40,0}) == {40}  q = {-np.inf}.issubset(S)  r = S.union({0}).issubset(S)  #Outputting their logical states  print(f"p: {p}, q: {q}, r: {r}")  #Evaluating the statement  outcome = (q^p)^r == (not(not(q)) or not(r)) # p: True, q: False, r: False  print("Outcome:",outcome) # Outputs True  #Answer = True, therefore choose the left lift |

**Limit Graph:**

A graph with green lines

Description automatically generated

## Task 5.2 - Maths:

To start, we must fill in the spots in the multiset (allows duplicate values):

First, the combinations:

Second, the permutations:

Third, the limit:

Different limits on both sides, therefore limit does not exist (NaN).

Task 3’s answer is NaN, so the final multiset is now:

Now, we find the states of , and :

Now we can evaluate ,

True, therefore choose the left lift.