2.3

Additional Project
Curves in the Complex Plane

1 Complex roots of $f(z)$

Programming Task: Plot $f(C_r)$ and find closest point to origin

A program for this task can be found on page 8. Below is sample output for $r = 2$, demonstrating that the program works as intended.

```
Enter radius r: 2
Closest point to 0+0i is :  (0.8109122548757242+3.0535245947615985j)
Modulus of closest point is: 3.15936562238713
```

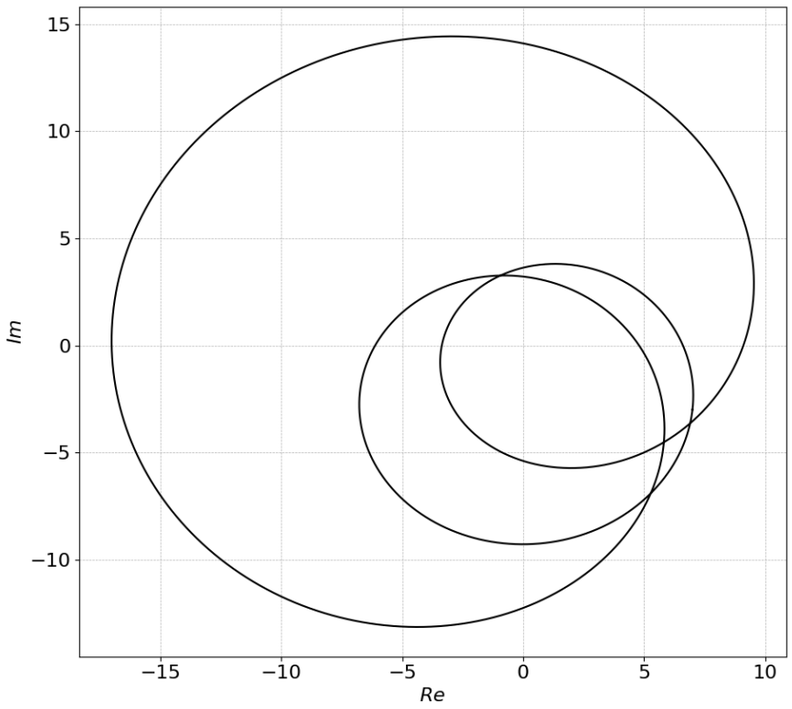


Figure 1: Sample output for $r = 2$

Question 1

We know there are 3 roots since f_1 is a cubic. Plotting a graph of the modulus of the closest point against r for r between 0 and 2, it is clear that the roots are near 0.6, 1.4 and 1.6.

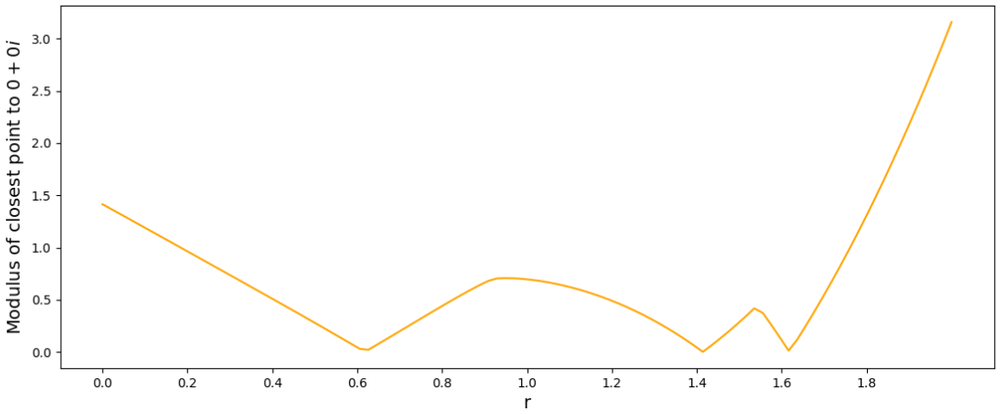


Figure 2: Modulus of closest point to $0 + 0i$ for $r \in [0, 2]$

The function *find_decimal* on page 9 is then used to work out the optimal value of r for an increasing number of decimal places. Using 0.6 as an example, we perform calculations as follows:

– Find minimum of the modulus of the closest point on $f(C_r)$ to $0 + 0i$ for $r = 0.50, 0.51, ...0.69, 0.70$.

– Find that 0.62 is minimal so we know that to 2d.p. the value of r at the root is $0.61, 0.62$ or 0.63 since the moduli of the closest points for different r decreases as r approaches the root value.

– Find minimum from $0.610, 0.611, ...0.629, 0.630$.

– After the 4th recursion we obtain the result $r = 0.61803$ so it must be that $r = 0.6180$ to 4d.p.

Then doing the same for 1.4 and 1.6, we obtain $r = 1.4142$ and $r = 1.6180$ also to 4d.p.

Let z be the value we approximate the root to be and write $z = re^{i\theta}$. We need to ensure that the real and imaginary parts of z are accurate up to 3s.f. From the $find\_decimal$ function, we also output the values of z at the optimal values of r (however we cannot be sure they are accurate to 3s.f.) and find that θ is approximately $1.57, 0.79, 4.71$ for $r = 0.6180, 1.4142, 1.6180$ respectively.

Let the exact values of r and θ at the root be r_0 and θ_0 respectively. For the three roots, we know r_0 to 4d.p. so we can say $r_0 \in [r - 5 \times 10^{-5}, r + 5 \times 10^{-5}]$.

Then we can use a very small step size for θ of $2\pi/350,000$ on a small arc of C_r (for fast calculation) around the values of θ above to find the optimal z with more accuracy. We obtain $\theta = 1.5708..., 0.7854..., 4.7123...$ and these results are accurate to within 0.00001 radians since $\pi/350,000 < 10^{-5}$.

$$\begin{aligned}
&\therefore \cos(\theta_0) \in [\cos(\theta - 10^{-5}), \cos(\theta + 10^{-5})] \\
&\qquad \subset [\cos(\theta) - 10^{-5}, \cos(\theta) + 10^{-5}] \\
\implies &(r - 5 \times 10^{-5})(\cos(\theta) - 10^{-5}) \leq r_0 \cos(\theta_0) \leq (r + 5 \times 10^{-5})(\cos(\theta) + 10^{-5}) \\
&\qquad r < 2 \text{ and } |\cos(\theta)| < 1, \\
&\qquad \implies r\cos(\theta) - 7 \times 10^{-5} < r_0 \cos(\theta_0) < r\cos(\theta) + 7 \times 10^{-5}
\end{aligned}$$

The same applies to $r\sin(\theta)$. For $r = 0.6180$, we have,

$$\begin{aligned}
&r\cos(\theta) + ir\sin(\theta) = 0.61803i \\
\implies &r_0\cos(\theta_0) \in [-7 \times 10^{-5}, 7 \times 10^{-5}] \\
&r_0\sin(\theta_0) \in [0.61803 - 7 \times 10^{-5}, 0.61803 + 7 \times 10^{-5}] = [0.61796, 0.6181] \\
\implies &z = 0.618i \text{ is accurate to 3s.f. for } r = 0.618.
\end{aligned}$$

Doing the same calculations for the other r, it follows that to 3s.f. the values of z for the corresponding r are,

$z = 0.618i$	$z = 1.00 + 1.00i$	$z = -1.62i$
$r = 0.618$	$r = 1.414,$	$r = 1.618$

Question 2

$$f_1'(z) = 3z^2 - 2z + 2 - i$$

We follow the same procedure as in Question 1, first plotting a graph to find that the values of r at the roots are approximately 0.8 and 0.95. Inputting these values into the recursive function we obtain,

$$\begin{aligned}
r = 0.785,\ z \approx 0.12 - 0.78i,\ \theta \approx 4.86 \\
r = 0.950,\ z \approx 0.55 + 0.78i,\ \theta \approx 0.96
\end{aligned}$$

Then, again using a step size for θ of $2\pi/350,000$ on a small arc of C_r, we increase the accuracy of the result for r to 5d.p. and find values of z. Using the same idea as in Question 1, this time the real and imaginary parts are accurate to 1.2×10^{-5} (not 7×10^{-5} like in Question 1 since r now accurate to 1 more d.p.). We find:

$$r = 0.78470,\ \text{Re}(z) \in [0.11847 - 1.2 \times 10^{-5}, 0.11847 + 1.2 \times 10^{-5}] = [0.118458, 0.118482]$$

$$\text{Im}(z) \in [-0.77571 - 1.2 \times 10^{-5}, -0.77571 + 1.2 \times 10^{-5}] = [-0.775722, -0.775698]$$

$$r = 0.94986, \quad \text{Re}(z) \in [0.54819 - 1.2 \times 10^{-5}, 0.54819 + 1.2 \times 10^{-5}] = [0.548178, 0.548202]$$
$$\text{Im}(z) \in [0.77570 - 1.2 \times 10^{-5}, 0.77570 + 1.2 \times 10^{-5}] = [0.775688, 0.775712]$$

It follows that to 3s.f. the values of z for the corresponding r must be,

$$z = 0.118 - 0.776i \qquad z = 0.548 + 0.776i$$
$$r = 0.785 \qquad r = 0.950$$

2 Images $f(C_r)$ of C_r

Question 3

The change made to the program is simply to add the function to be plotted as a parameter in the *generate_points* function found in the program for programming task 1 on page 8.

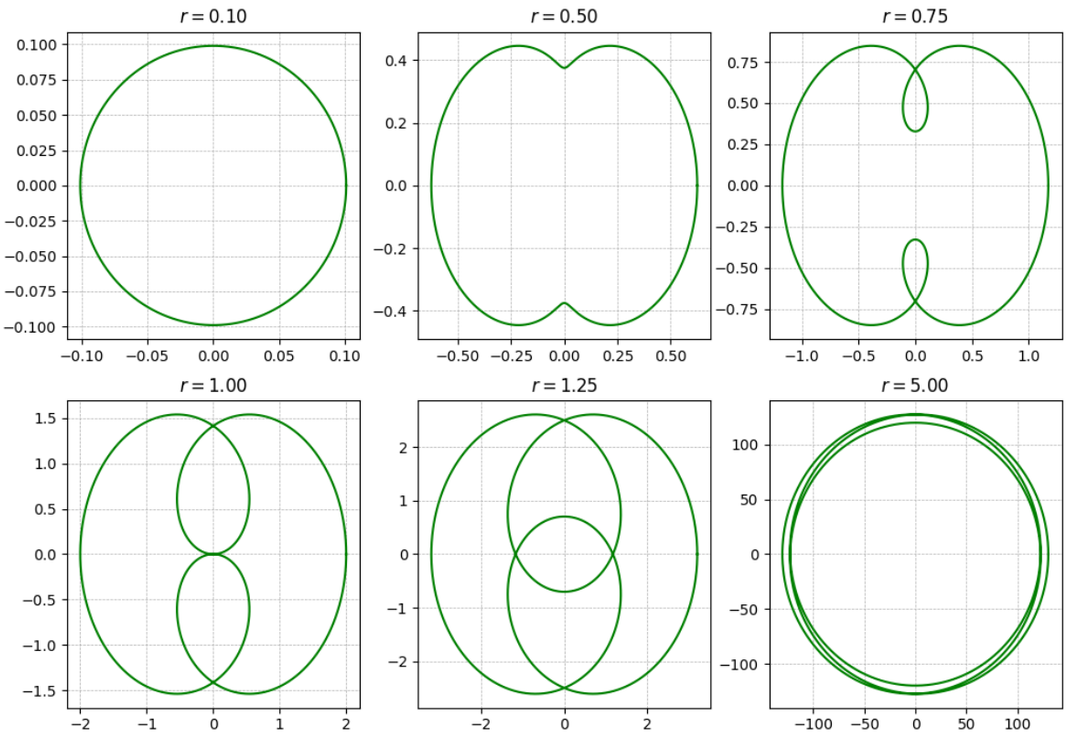


Figure 3: Plots of $g(C_r)$ for different values of r

- As r increases, the size of the shape formed by $g(C_r)$ also increases since the magnitude of the z and z^3 terms increases.
- For very small r, such as $r = 0.1$ in Figure 3, $g(C_r)$ approximates a circle. This is because z is much larger than z^3 here so the curve is almost the circle from the image of z.
- As r increases up to $1/\sqrt{3}$, the top and bottom of this circle begin curving inwards. Then two loops form simultaneously, one at the top and another at the bottom of the curve $g(C_r)$.
- The loops appear for $r > 1/\sqrt{3}$. This is derived by taking $z = re^{i\theta}$ and finding the derivative of the real part of g w.r.t. θ to be $-3r^3 sin(3\theta) + rsin(\theta)$ with $r = 1/\sqrt{3}$ as the positive root when $\theta = \pi/2$ ($g(z)$ is pure imaginary for this θ).
- An intuitive explanation for the loops is that we have a circle from z^3 which loops three times added to a circle from z which loops once. Then once the z^3 term is large enough to have an effect the loops start developing.

- We can examine the curve for $\theta = 0$ to $\theta = \pi/2$ and then use symmetry to form the rest of the curve.

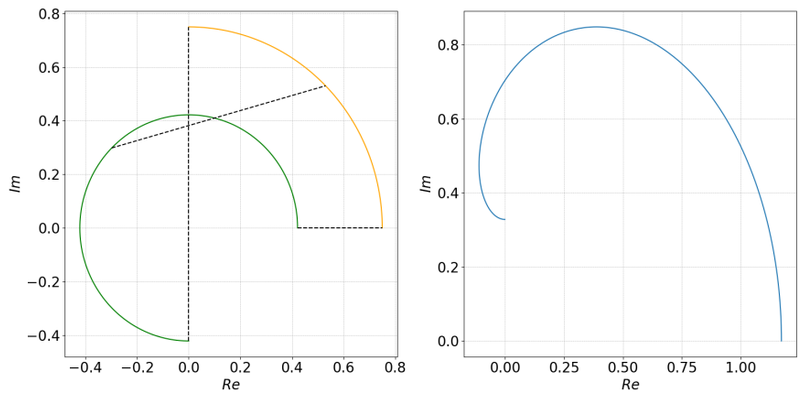


Figure 4: Formation of $g(C_{0.75})$ from a quarter of $C_{0.75}$

The black lines in Figure 4 show points which are added to create points in the subplot to the right of Figure 4. We can think of the orange curve distorting the green curve to create the blue curve. Using the reflection symmetry in the real and imaginary axes, we obtain the curve for $r = 0.75$ in Figure 3.

- The loops grow in size as r increases and overlap for $r > 1$. For large r the curve again resembles a circle as demonstrated by $r = 5$ in figure 3.

Question 4

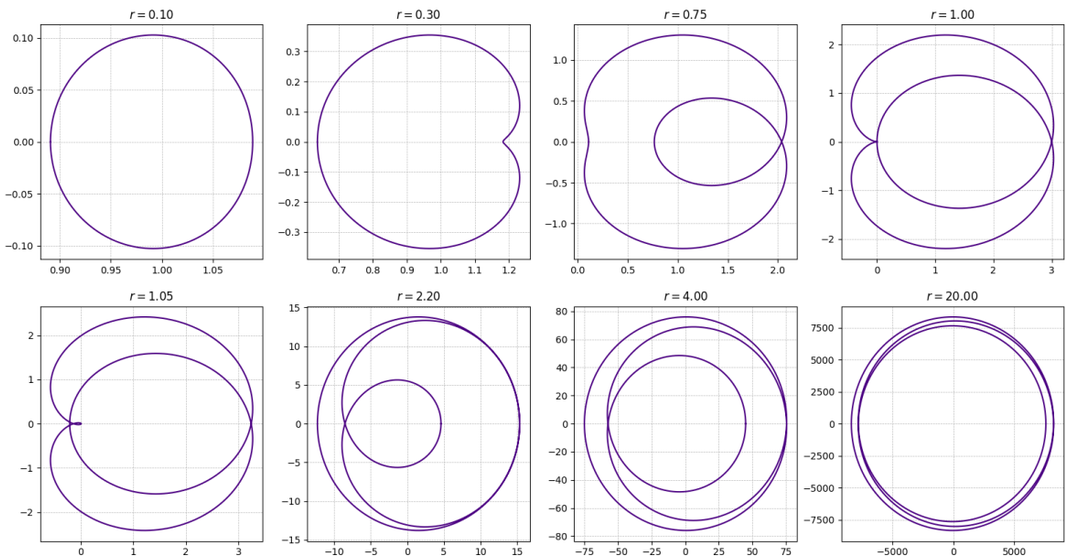
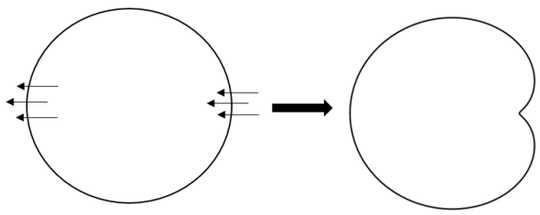


Figure 5: Plots of $h(C_r)$ for different values of r

- The change in $h(C_r)$ as r increases is similar to in Question 3 since two loops develop and because for very small and large r, the curve resembles a circle. This is clear to see in Figure 5.
- The curves are shifted to the right of the origin because of the $+1$ term in $h(z)$.
- Unlike in Question 3, the loops develop to the left and right of the curve as the terms of h pulls the curve inward on the real axis rather than on the imaginary axis.
- For $r < 1$ a right loop develops before the left loop. For these smaller values of r the z^3 is relatively small so can be ignored. Then the $-z^2$ term (whose image is a circle looping twice) stretches points near the real axis on the circle formed by $1 - z$ to the left. This results in right side of $h(C_r)$ being more warped than the left side. The image below represents this visually.



- The left loop emerges at the same time as the the two sides of the curve intercept at $0 + 0i$ as shown in Figure 5 using $r = 1$ and $r = 1.05$. This is because $h(z) = (z-1)^2(z+1)$ so there is a triple root for $r = 1$ which is only possible if we have the configuration described.
- For larger r there is a transition where the curve has 2 points of self-intersection rather than 4. This transition happens at around $r = 2.2$ and is because for larger r the z^3 term dominates h causing the former right loop to grow such that it encircles the entire curve.

Question 5

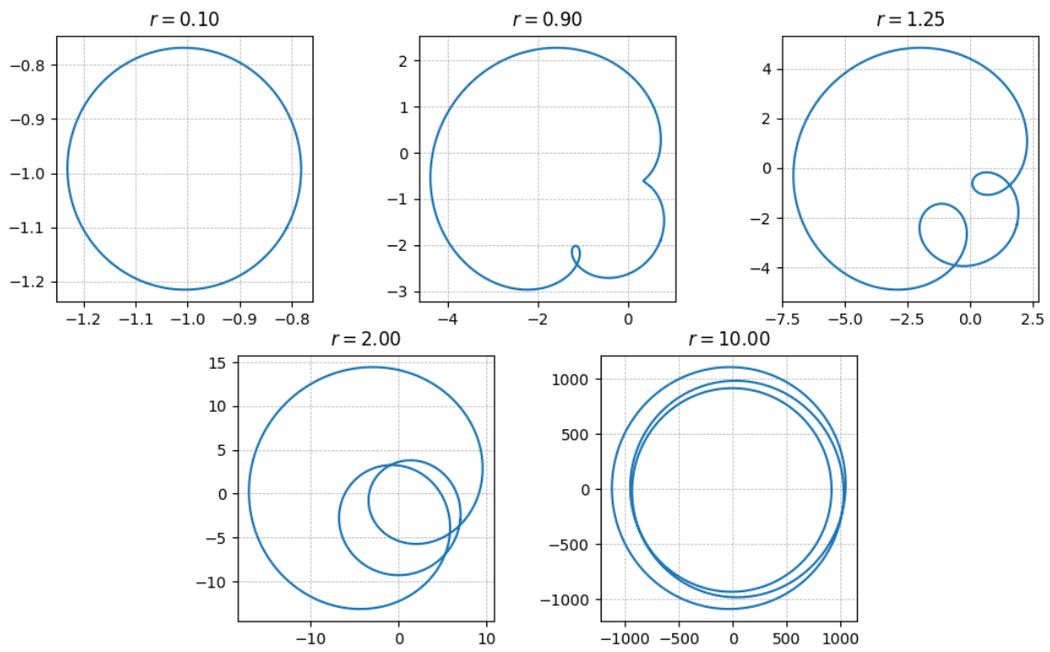


Figure 6: Plots of $f_1(C_r)$ for different values of r

- Similar to the curves $g(C_r)$ and $h(C_r)$, two loops develop as r increases. In this case, they're positioned down and to the right which can be explained as follows: without the $-z^2$ term, the curves $f_1(C_r)$ are like the plots in Question 3 Figure 3 but rotated by $arg(2 - i)$ clockwise. Then $-z^2$ shifts these the loops to the right and shifts the lower right section of the curve inwards.
- The lower loop develops before the loop to the right, as shown by the curve for $r = 0.9$ in Figure 6. The same ideas as Question 4 help explain this; for smaller r we get one loop developing faster as we the z^3 term has limited effect. Here this corresponds to the lower loop developing faster.
- As the loops grow they intersect and overlap in the same way as the curves in Questions 3 and 4. Then for very large r, the curve has a circular shape.

3 Curvature of images $f(C_r)$ of C_r

Question 6

$$|\kappa| = \left| \frac{d\boldsymbol{t}(s)}{d\phi} \frac{d\phi}{ds} \right|$$

$$\boldsymbol{t} = \frac{d\boldsymbol{x}}{d\phi}\Big/\left|\frac{d\boldsymbol{x}}{d\phi}\right| \quad \& \quad \left|\frac{ds}{d\phi}\right| = \left|\frac{d\boldsymbol{x}}{d\phi}\right| = |\boldsymbol{x'}|. \text{ Therefore,}$$

$$|\kappa| = \left|\frac{d}{d\phi}\left(\frac{\boldsymbol{x'}}{|\boldsymbol{x'}|}\right)\cdot|\boldsymbol{x'}|^{-1}\right|$$

$$= \left|\frac{|\boldsymbol{x'}|\boldsymbol{x''} - \boldsymbol{x'}\frac{d}{d\phi}|\boldsymbol{x'}|}{|\boldsymbol{x'}|^3}\right|$$

$$\frac{d}{d\phi}|\boldsymbol{x'}|^2 = 2\boldsymbol{x'}\cdot\boldsymbol{x''} \quad \& \quad \frac{d}{d\phi}|\boldsymbol{x'}|^2 = 2|\boldsymbol{x'}|\frac{d}{d\phi}|\boldsymbol{x'}| \implies \frac{d}{d\phi}|\boldsymbol{x'}| = \left|\frac{\boldsymbol{x'}\cdot\boldsymbol{x''}}{\boldsymbol{x'}}\right|$$

$$\therefore\ |\kappa| = \left|\frac{|\boldsymbol{x'}|^2\boldsymbol{x''} - |\boldsymbol{x'}\cdot\boldsymbol{x''}|\,\boldsymbol{x'}}{|\boldsymbol{x'}|^4}\right|$$

$$= \left|\frac{|\boldsymbol{x'}\times(\boldsymbol{x''}\times\boldsymbol{x'})}{|\boldsymbol{x'}|^4}\right| \quad \text{using triple product identity}$$

$$\boldsymbol{x''} \text{ and } \boldsymbol{x''}\times\boldsymbol{x'} \text{ orthogonal} \implies |\kappa| = \frac{|\boldsymbol{x'}\times\boldsymbol{x''}|}{|\boldsymbol{x'}|^3} \qquad \square$$

Finding expressions for $\boldsymbol{x'}$ and $\boldsymbol{x''}$:

$$\boldsymbol{t}(s) = \frac{\boldsymbol{x'}}{|\boldsymbol{x'}|} \quad \& \quad |\boldsymbol{x'}| = \left|\frac{ds}{d\phi}\right| \implies \boldsymbol{x'} = \left|\frac{ds}{d\phi}\right|\boldsymbol{t}(s)$$

$$\boldsymbol{k}(s) = \frac{d\boldsymbol{t}(s)}{ds} = \frac{d}{d\phi}\left(\left|\frac{ds}{d\phi}\right|^{-1}\boldsymbol{x'}\right)\left(\frac{ds}{d\phi}\right)^{-1}$$

$$\frac{d}{d\phi}\left(\left|\frac{ds}{d\phi}\right|\right)^2 = 2\frac{ds}{d\phi}\frac{d^2s}{d\phi^2} \quad \& \quad \frac{d}{d\phi}\left(\left|\frac{ds}{d\phi}\right|\right)^2 = 2\left|\frac{ds}{d\phi}\right|\frac{d}{d\phi}\left|\frac{ds}{d\phi}\right|$$

$$\implies \frac{d}{d\phi}\left|\frac{ds}{d\phi}\right| = \frac{ds}{d\phi}\frac{d^2s}{d\phi^2}\Big/\left|\frac{ds}{d\phi}\right|$$

$$\therefore\ \boldsymbol{k}(s) = \frac{\left(\frac{ds}{d\phi}\right)^2\boldsymbol{x''} - \frac{ds}{d\phi}\frac{d^2s}{d\phi^2}\boldsymbol{x'}}{\left(\frac{ds}{d\phi}\right)^3\left|\frac{ds}{d\phi}\right|}$$

$$\therefore\quad \boldsymbol{x''} = \left|\frac{ds}{d\phi}\right|\left(\frac{ds}{d\phi}\boldsymbol{k}(s) + \left(\frac{ds}{d\phi}\right)^{-1}\frac{d^2s}{d\phi^2}\boldsymbol{t}(s)\right)$$

$\boldsymbol{k}(s)$ is orthogonal to $\boldsymbol{t}(s)$ since $\frac{d}{ds}(\boldsymbol{t}\cdot\boldsymbol{t}) = 0 \implies \boldsymbol{t}\cdot\boldsymbol{k} = 0$. Then $\boldsymbol{t}(s)\times\boldsymbol{k}(s)$ is orthogonal to the plane the curve is in. $\boldsymbol{k}(s)$ points towards the local centre of rotation. Therefore, using the right hand rule, the curve is turning anticlockwise at $\boldsymbol{x}(s)$ iff the component of $\boldsymbol{t}(s)\times\boldsymbol{k}(s)$ orthogonal to the plane is > 0. Hence $\operatorname{sgn}(\kappa) = \operatorname{sgn}(\boldsymbol{t}(s)\times\boldsymbol{k}(s))$, where sgn of the vector is taken to mean the sign of the component orthogonal to the plane.

$$\boldsymbol{x'}\times\boldsymbol{x''} = \left|\frac{ds}{d\phi}\right|^2\frac{ds}{d\phi}\left(\boldsymbol{t}(s)\times\boldsymbol{k}(s)\right)$$

$$\implies \operatorname{sgn}(\kappa) = \operatorname{sgn}\left(\frac{\boldsymbol{x'}\times\boldsymbol{x''}}{ds/d\phi}\right)$$

In our case, $\frac{ds}{d\phi} > 0$ since distance along the curve increases as ϕ increases, hence, $\operatorname{sgn}(\kappa) = \operatorname{sgn}(\boldsymbol{x'}\times\boldsymbol{x''})$.

Writing (8) and (9) in terms of derivatives of $f(z)$ with respect to z:

$$\frac{df(z(\phi))}{d\phi} = \frac{df(z)}{dz}\frac{dz}{d\phi}$$

$$\frac{d^2f(z(\phi))}{d\phi^2} = \frac{df(z)}{dz}\frac{d^2z}{d\phi^2} + \frac{d^2f(z)}{dz^2}\left(\frac{dz}{d\phi}\right)^2$$

$$z = re^{i\phi} \text{ along } C_r \implies \frac{dz}{d\phi} = ire^{i\phi} = iz, \quad \frac{d^2z}{d\phi^2} = -re^{i\phi} = -z$$

$$\therefore \quad \frac{df(z(\phi))}{d\phi} = i\frac{df(z)}{dz}re^{i\phi}$$

$$\frac{d^2f(z(\phi))}{d\phi^2} = -re^{i\phi}\left(\frac{df(z)}{dz} + \frac{d^2f(z)}{dz^2}re^{i\phi}\right)$$

$$\therefore \quad \boldsymbol{x'} = \left(\mathrm{Re}\left[i\frac{df(z)}{dz}z\right], \mathrm{Im}\left[i\frac{df(z)}{dz}z\right]\right) \tag{8}$$

$$\boldsymbol{x''} = \left(\mathrm{Re}\left[-z\frac{d}{dz}\left(\frac{df(z)}{dz}z\right)\right], \mathrm{Im}\left[-z\frac{d}{dz}\left(\frac{df(z)}{dz}z\right)\right]\right) \tag{9}$$

Programming Task: Write a program to compute the integral κ_{tot}

$$\kappa_{tot} = \int_0^{2\pi} \kappa|\boldsymbol{x'}|d\phi$$

The integral is calculated using this equation, and the trapezium rule. The code for this program can be found on page 15.

Question 7

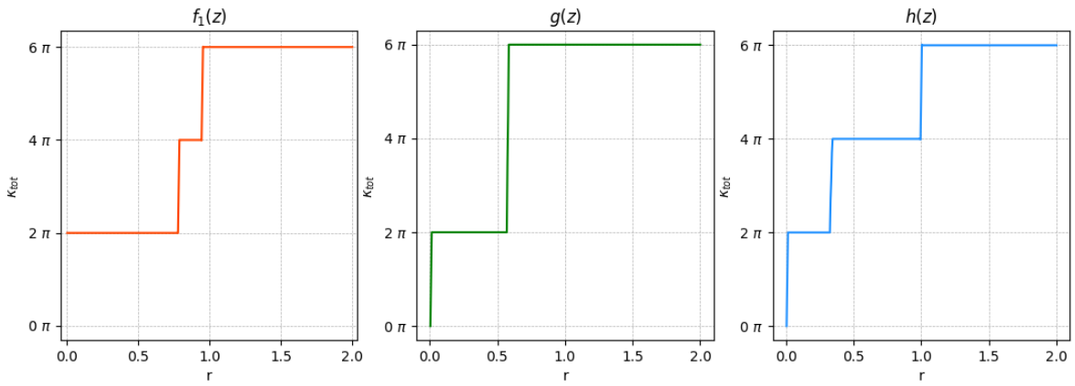


Figure 7: Plots of κ_{tot} against r for $f_1(z)$, $g(z)$ and $h(z)$

From the graphs, it is clear to see that the value of k_{tot} jumps and plateaus at increasing multiples of 2π. The jumps are at the values of r where there is a root in the derivative of the function for this r. For instance, the two values of r at the roots found in Question 2 correspond exactly with the jumps in Figure 7 for $f_1(z)$. Additionally, there is a jump at $r = 1/\sqrt{3}$ in $g(z)$ where we said the loop starts to form for $g(C_r)$. This demonstrates that the curvature jumps when another (anticlockwise) loop forms and this is precisely where we will find roots in the derivative. Numerically, when a new small loop forms, the value of $|\boldsymbol{x'}|$ on this loop will also be small, so we get a spike in the value of κ using the formula in Question 6.

For a general polynomial $f(z)$, κ_{tot} tells us the number of 2π rotations anticlockwise someone would experience as they move along the curve. This is supported by all the evidence of the previous paragraph.

programming_task_1.py for programming task

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


def generate_points(r, func, begin = 0, end = 2*cmath.pi,
                                    step = 2*cmath.pi/10000):
    # change begin and end for different sections of C_r
    # 0, 2 for full circle
    vf = np.vectorize(func)
    z_in = []
    no_points = round((end - begin)/step)
    for counter in range(no_points + 1):
        # range is number of points in plot
        z_in.append(r*cmath.exp((begin + step*counter)*1j))
    z_in = np.array(z_in)
    w_out = vf(z_in)
    return z_in, w_out


def find_closest(z_in, w_out):
    smallest_modulus = float('inf')
    closest_point = 0
    index = 0  # for finding inputted value of z
    for point in w_out:
        modulus = abs(point)
        if modulus < smallest_modulus:
            smallest_modulus = modulus
            closest_point = point
            closest_point_input = z_in[index]
        index += 1
    return smallest_modulus, closest_point, closest_point_input


if __name__ == '__main__':
    # ^ will not run when imported into other files
    r = input('Enter radius r: ')
    r = float(r)

    z_in, w_out = generate_points(r, lambda z:
                    z**3 - z**2 + (2-1j)*z - 1 - 1j)
    smallest_modulus, closest_point, \
            closest_point_input = find_closest(z_in, w_out)
    print('Closest point to 0+0i is: ', closest_point)
    print('Modulus of closest point is:', smallest_modulus)

    plt.rc('font', size = 16)
    plt.plot(w_out.real, w_out.imag, color = 'black')
    plt.xlabel('$Re$')
    plt.ylabel('$Im$')
    plt.grid(linestyle = '--', linewidth = 0.5)
    plt.show()
```

q1and2_functions.py used in questions 1 and 2

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

from programming_task_1 import generate_points, find_closest


def modulus_plot(r_start, r_end, func):
    # plots graph showing shortest modulus for different r
    r_list = np.linspace(r_start, r_end, 100)
    # 100 points used between start and end
    r_modulus_list = []
    for r in r_list:
        z_in, w_out = generate_points(r, func)  # change for Q2
        smallest_modulus = find_closest(z_in, w_out)[0]
        r_modulus_list.append(smallest_modulus)

    r_modulus_list = np.array(r_modulus_list)
    plt.plot(r_list, r_modulus_list, color = 'orange')
    plt.xticks(np.arange(r_start, r_end, 0.2))
    plt.xlabel('r', fontsize = 14)
    plt.ylabel('Modulus of closest point to $0+0i$',
                                          fontsize = 14)
    # change step size for plots in q1 and q2
    plt.show()


def find_decimal(start, decimal_place, func, begin = 0,
                 end = 2*math.pi, step = 2*math.pi/10000):
    # finds next decimal of r which minimising smallest modulus
    # begin, end, no_points used in generate_points function
    r_list = np.linspace(start - 0.1**(decimal_place),
                         start + 0.1**(decimal_place), 21)
    # 20 next decimal points between start and end tested
    smallest_modulus = math.inf
    optimal_r = start
    optimal_z = 0  # this will be the actual root
    for r in r_list:
        z_in, w_out = generate_points(r, func, begin, end, step)
        modulus, _, input_point = find_closest(z_in, w_out)
        if modulus < smallest_modulus:
            smallest_modulus = modulus
            optimal_r = r
            optimal_z = input_point
        # finds smallest modulus for all values of r in list
    return optimal_r, optimal_z

def find_r(start, func = lambda z:
        z**3 - z**2 + (2-1j)*z - 1 - 1j):
    # know start to 0 d.p
    for decimal in range(4):
        start, z = find_decimal(start, decimal + 1, func)
    return start, z
```

question_1.py for question 1

```
from q1and2_functions import modulus_plot, find_r, find_decimal
import math
import cmath

modulus_plot(0, 2, lambda z: z**3 - z**2 + (2-1j)*z - 1 - 1j)

# Use different start, end pairs in programming task 1
r1, z = find_r(0.6)
print(r1, z, cmath.phase(z))  # phase is arg
r2, z = find_r(1.4)
print(r2, z, cmath.phase(z))
r3, z = find_r(1.6)
print(r3, z, cmath.phase(z), '\n')

r, z = find_decimal(0.61803, 5, lambda z: z**3 - z**2
            + (2-1j)*z - 1 - 1j, (0.5 - 0.1)*math.pi,
             (0.5 + 0.1)*math.pi, 2*math.pi/350000)
arg = cmath.phase(z)
print(r, z, arg)
print(r1*math.cos(arg), r1*math.sin(arg))
r, z = find_decimal(1.41421, 5, lambda z: z**3 - z**2
                + (2-1j)*z - 1 - 1j, (0.25 - 0.1)*math.pi,
                 (0.25 + 0.1)*math.pi, 2*math.pi/350000)
arg = cmath.phase(z)
print(r, z, arg)
print(r2*math.cos(arg), r2*math.sin(arg))
r, z = find_decimal(1.61803, 5, lambda z: z**3 - z**2
                + (2-1j)*z - 1 - 1j, (1.5 - 0.1)*math.pi,
                 (1.5 + 0.1)*math.pi, 2*math.pi/350000)
arg = cmath.phase(z) + 2*math.pi
print(r, z, arg)
print(r3*math.cos(arg), r3*math.sin(arg))
# Smaller step size giving more accuracy
# Smaller arc used for fast calculation
```

question_2.py for question 2

```
from q1and2_functions import modulus_plot, find_decimal, find_r
import math
import cmath

modulus_plot(0.6, 1.2, lambda z: 3*z**2 - 2*z + 2 - 1j)
# From this is is obvious the roots are roughly 0.8, 0.95

r, z = find_r(0.7, lambda z: 3*z**2 - 2*z + 2 - 1j)
print(r, z, cmath.phase(z))
r, z = find_r(0.9, lambda z: 3*z**2 - 2*z + 2 - 1j)
print(r, z, cmath.phase(z), '\n')

r1, z = find_decimal(0.78470, 5, lambda z: 3*z**2 - 2*z + 2 - 1j,
                     4.86 - 0.1*math.pi, 4.86 + 0.1*math.pi,
                     2*math.pi/350000)
arg = cmath.phase(z)+2*math.pi
print(r1, z, arg)
print(r1*math.cos(arg), r1*math.sin(arg))
r2, z = find_decimal(0.94986, 5, lambda z: 3*z**2 - 2*z + 2 - 1j,
                     0.96 - 0.1*math.pi, 0.96 + 0.1*math.pi,
                     2*math.pi/350000)
arg = cmath.phase(z)+2*math.pi
print(r2, z, arg)
print(r2*math.cos(arg), r2*math.sin(arg))
```

question_3.py for question 3

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

from programming_task_1 import generate_points

r_list = [0.1, 0.5, 0.75, 1, 1.25, 5]
subplot_no = 1
for i in range(6):
    z_in, w_out = generate_points(r_list[i],
                                  lambda z: z**3 + z)
    plt.subplot(2, 3, i+1)
    plt.plot(w_out.real, w_out.imag, antialiased=True,
                                        color = 'green')
    # plt.xlabel('$Re$')
    # plt.ylabel('$Im$')
    plt.grid(linestyle = '--', linewidth = 0.5)
    plt.title('$r = %.2f$' %r_list[i])
plt.show()


# first plot the circle arcs
plt.rc('font', size = 20)
r = 0.75
vf = np.vectorize(lambda z: z**3)
z_in = []
for counter in range(2001):
    # range is number of points in plot
    z_in.append(r*cmath.exp((0.5*math.pi*(1/2000)*counter)*1j))
z_in = np.array(z_in)
w_out = vf(z_in)
plt.subplot(1, 2, 1)
plt.plot(w_out.real, w_out.imag, color = 'green')
vf = np.vectorize(lambda z: z)
z_in = []
for counter in range(2001):
    # range is number of points in plot
    z_in.append(r*cmath.exp((0.5*math.pi*(1/2000)*counter)*1j))
z_in = np.array(z_in)
w_out = vf(z_in)
plt.plot(w_out.real, w_out.imag, color = 'orange')
# now plot some dashed lines
p1, q1, s1 = r**3, (r**3)*cmath.exp((3*math.pi/4)*1j), \
                    (r**3)*cmath.exp((3*math.pi/2)*1j)
p2, q2, s2 = r, r*cmath.exp((math.pi/4)*1j), \
                    r*cmath.exp((math.pi/2)*1j)
plt.plot([p1.real, p2.real], [p1.imag, p2.imag],
            linestyle = 'dashed', color = 'black')
plt.plot([q1.real, q2.real], [q1.imag, q2.imag],
            linestyle = 'dashed', color = 'black')
plt.plot([s1.real, s2.real], [s1.imag, s2.imag],
            linestyle = 'dashed', color = 'black')
plt.xlabel('$Re$')
plt.ylabel('$Im$')
plt.grid(linestyle = '--', linewidth = 0.5)
```

```
#next plot image
vf = np.vectorize(lambda z: z**3+z)
z_in = []
for counter in range(2001):
    # range is number of points in plot
    z_in.append(r*cmath.exp((0.5*math.pi*(1/2000)*counter)*1j))
z_in = np.array(z_in)
w_out = vf(z_in)
plt.subplot(1, 2, 2)
plt.plot(w_out.real, w_out.imag)
plt.xlabel('$Re$')
plt.ylabel('$Im$')
plt.grid(linestyle = '--', linewidth = 0.5)
plt.show()
```

question_4.py for question 4

```
import matplotlib.pyplot as plt

from programming_task_1 import generate_points

r_list = [0.1, 0.3, 0.75, 1, 1.05, 2.2, 4, 20]
subplot_no = 1
for i in range(8):
    z_in, w_out = generate_points(r_list[i],
                  lambda z: z**3 - z**2 - z + 1)
    plt.subplot(2, 4, i+1)
    plt.plot(w_out.real, w_out.imag, color = 'indigo')
    plt.grid(linestyle = '--', linewidth = 0.5)
    plt.title('$r = %.2f$' %r_list[i])
    # plt.annotate('origin', xy=(0, 0), xytext=(1, 0.5),
    #           arrowprops=dict(facecolor='black', shrink=0.01))
plt.show()
```

question_5.py for question 5

```
import matplotlib.pyplot as plt

from programming_task_1 import generate_points

r_list = [0.7, 0.9, 1.25, 2, 10]
loc_list = [(0, 0), (0, 4), (0, 8), (1,2), (1, 6)]
subplot_no = 1
for i in range(5):
    z_in, w_out = generate_points(r_list[i],
        lambda z: z**3 - z**2 + (2-1j)*z - 1 - 1j)
    plt.subplot2grid(shape = (2, 12), loc = loc_list[i],
                                                colspan = 3)
    plt.plot(w_out.real, w_out.imag, color = 'black')
    plt.grid(linestyle = '--', linewidth = 0.5)
    plt.title('$r = %.2f$' %r_list[i])
plt.show()
```

programming_task_2.py for programming task

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

from programming_task_1 import generate_points


def k_tot(r, f_prime, f_dprime):
    no_points = 1000 # increase for more accuracy
    z_array, f_prime_array = generate_points(r,
            f_prime, step = 2*math.pi / no_points)
    f_dprime_array = generate_points(r, f_dprime,
                step = 2*math.pi / no_points)[1]

    x_prime_array = 1j*f_prime_array*z_array
    x_dprime_array = -z_array*(f_prime_array +
                        f_dprime_array*z_array)

    A = x_prime_array.real * x_dprime_array.imag
    B =  x_prime_array.imag * x_dprime_array.real
    C = np.square(np.absolute(x_prime_array))
    integrand_array = np.divide(A - B, C,
        out = np.zeros_like(A), where = C > 0.0001)
    # where condition eliminates (most) problems dividing by 0
    # where a small loops forst forms

    # trapezium rule:
    k_tot = ((2*math.pi / no_points)*(sum(integrand_array) - \
            (integrand_array[0] + integrand_array[-1]) / 2))
    return k_tot, integrand_array

if __name__ == '__main__':
    k_t, integrand_array = k_tot(0.9, lambda z :
        3*z**2 - 2*z + 2 - 1j, lambda z : 6*z - 2)
    plt.plot(integrand_array)
    plt.show()
    print(k_t)
```

question_7.py for question 7

```
import math
import matplotlib.pyplot as plt
import matplotlib.ticker as tck
import numpy as np
from programming_task_2 import k_tot

function_names = ['f_1(z)', 'g(z)', 'h(z)']
function_derivates = [lambda z : 3*z**2 - 2*z + 2 - 1j,
    lambda z : 3*z**2 + 1, lambda z : 3*z**2 - 2*z - 1]
function_2nd_derivates = [lambda z : 6*z - 2, lambda z :
                                6*z, lambda z : 6*z - 2]
colors = ['orangered', 'green', 'dodgerblue']


for i in range(3):
    r_max = 2
    r_array = []
    k_tot_array = []
    for c in range(400):
        r = r_max * (c+1) / 400  # c+1 so r = 0 not included
        r_array.append(r)
        k_tot_array.append(k_tot(r, function_derivates[i],
                            function_2nd_derivates[i])[0])
    plt.subplot(1, 3, i+1)
    plt.plot(r_array, (1 / math.pi) * np.array(k_tot_array),
                                        color = colors[i])
    ax = plt.gca()
    ax.yaxis.set_major_formatter \
        (tck.FormatStrFormatter('%g $\pi$'))
    ax.yaxis.set_major_locator \
        (tck.MultipleLocator(base=2))
    plt.grid(linestyle = '--', linewidth = 0.5)
    plt.xlabel('r')
    plt.ylabel('$\kappa_{tot}$')
    plt.title('$%s$' %function_names[i])
plt.show()
```