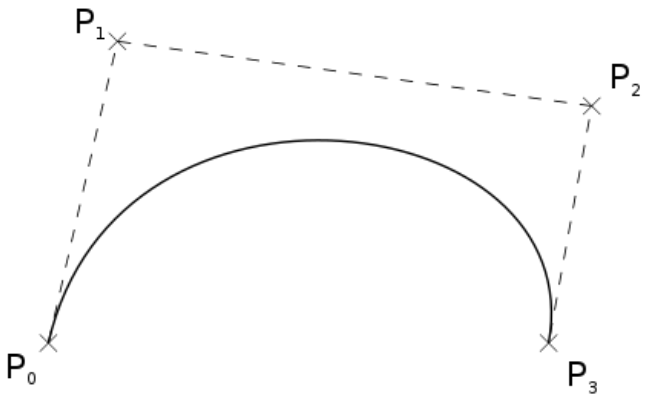
Laporan Tugas Kecil 2
IF2211 Strategi Algoritma
Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer
Semester II Tahun 2023/2024

Disusun Oleh:
Randy Verdian 13522067
Azmi Mahmud Bazeid 13522109

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

DAFTAR ISI

Bab 1
Deskripsi Masalah



Gambar 1. Kurva Bézier Kubik
(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti *pen tool*, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P_0 sampai P_n , dengan n disebut order (n = 1 untuk linier, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P_0, sedangkan titik kontrol terakhir adalah P_3. Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan **kurva Bézier linier**. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1. Misalnya ketika t = 0.25, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2, dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik P_0 dan P_2. Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

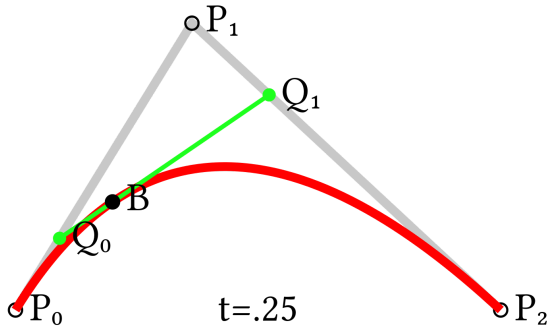
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

dengan melakukan substitusi nilai Q_0 dan Q_1, maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1-t)^2P_0 + (1-t)tP_1 + t^2P_2, \qquad t \in [0,1]$$

Berikut adalah ilustrasi dari kasus diatas.



Gambar 2. Pembentukan Kurva Bézier Kuadratik.
(Sumber: https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/)

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan **kurva Bézier kubik**, lima titik akan menghasilkan **kurva Bézier kuartik**, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

$$S_0 = B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t) t^3 P_3 + t^4 P_4, \qquad t \in [0,1]$$

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis ***divide and conquer*.**

Ilustrasi kasus

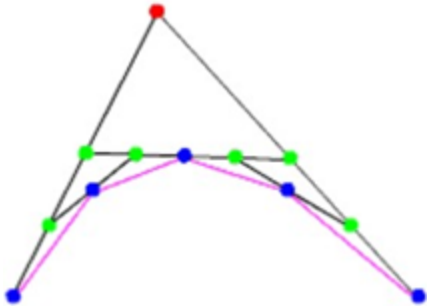
Idenya cukup sederhana, relatif mirip dengan pembahasan sebelumnya, dan dilakukan secara iteratif. Misalkan terdapat tiga buah titik, P_0, P_1, dan P_2, dengan titik P_1 menjadi titik kontrol antara, maka:

a. Buatlah sebuah titik baru Q_0 yang berada di tengah garis yang menghubungkan P_0 dan P_1, serta titik Q_1 yang berada di tengah garis yang menghubunhkan P_1 dan P_2.
b. Hubungkan Q_0 dan Q_1 sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru R_0 yang berada di tengah Q_0 dan Q_1.
d. Buatlah sebuah garis yang menghubungkan P_0 - R_0 - P_2.

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah “kurva” yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedurnya.

a. Buatlah beberapa titik baru, yaitu S_0 yang berada di tengah P_0 dan Q_0, S_1 yang berada di tengah Q_0 dan R_0, S_2 yang berada di tengah R_0 dan Q_1, dan S_3 yang berada di tengah Q_1 dan P_2.
b. Hubungkan S_0 dengan S_1 dan S_2 dengan S_3 sehingga terbentuk garis baru.
c. Buatlah dua buah titik baru, yaitu T_0 yang berada di tengah S_0 dan S_1, serta T_1 yang berada di tengah S_2 dan S_3.
d. Buatlah sebuah garis yang menghubungkan P_0 - T_0 - R_0 - T_1 - P_2.

Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik. Anda dapat membuat visualisasi atau gambaran secara mandiri terkait hal ini sehingga dapat diamati dan diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



Gambar 3. Hasil pembentukan Kurva Bézier Kuadratik dengan *divide and conquer* setelah iterasi ke-2.

Bab 2
Analisis dan Implementasi dalam Algoritma *Brute Force*

Prosedur pembentukan kurva Bezier kuadratik dengan algoritma *brute force* adalah sebagai berikut:

1. Inisialisasi: deklarasikan list kosong untuk menyimpan titik-titik dari kurva Bézier dan menetapkan variabel density ke 300, yang menentukan berapa banyak titik yang akan dihitung sepanjang kurva.

2. Iterasi: Fungsi melakukan loop untuk melakukan iterasi sebanyak density + 1 kali, dalam program ini dibuat 300 + 1 titik.

3. Persamaan kurva Bezier kuadratik:

$$B(t) = (1-t)^2 P_0 + (1-t)tP_1 \ + \ t^2 P_2, \qquad t \in [0,1]$$

4. Perhitungan Titik:

- First: titik kontrol pertama dihitung dengan mengalikan setiap koordinatnya dengan (1-t)^2.
- Second: titik kontrol kedua dihitung dengan mengalikan koordinatnya dengan 2(1-t)t.
- Third: titik kontrol ketiga dihitung dengan mengalikan koordinatnya dengan t^2.

5. Menambahkan Titik: ketiga variabel first, second, dan third dijumlahkan untuk menemukan koordinat titik pada kurva pada parameter t, dan titik ini ditambahkan ke list result.

6. Peningkatan Parameter: Setelah setiap iterasi, parameter t ditingkatkan sebesar 1/density, dan akan menuju 1, yang merupakan akhir dari kurva.

7. Hasil: Setelah semua titik dihitung, fungsi mengembalikan list result yang berisi semua titik yang membentuk kurva Bézier kuadratik.

Bab 3
Analisis dan Implementasi dalam Algoritma *Divide and Conquer*

Prosedur algoritma ini adalah implementasi dari algoritma divide and conquer untuk membagi kurva *Bezier* kuadratik berdasarkan titik kontrolnya. Berikut adalah prosedurnya:

1. Fungsi midpoint: Pertama, kita mendefinisikan fungsi midpoint yang mengambil dua titik kontrol dan mengembalikan titik tengah di antara keduanya.
2. Inisialisasi titik kontrol awal: Kita memulai dengan satu set titik kontrol awal yang diberikan dalam bentuk control_points.
3. Iterasi: Algoritma melakukan iterasi sebanyak iterasi yang diberikan.
4. Pembagian midpoints: Pada setiap iterasi, algoritma membagi setiap segmen garis antara dua titik kontrol ke dalam dua segmen yang lebih kecil dengan menambahkan titik tengah di antara mereka. Ini dilakukan dengan cara menghitung midpoint antara setiap pasang titik kontrol berturut-turut.
5. Pembentukan titik kontrol baru: Setelah menentukan midpoint untuk setiap pasang titik kontrol, kita membentuk set baru dari titik kontrol dengan menggunakan midpoint yang dihitung sebelumnya.
6. Membentuk kurva baru: Setelah mendapatkan set titik kontrol baru, kita menggunakan titik-titik tersebut untuk membentuk kurva baru.
7. Penambahan midpoints: Pada setiap iterasi, midpoint yang baru ditambahkan pada iterasi sebelumnya digunakan untuk menghitung midpoint baru dan menjadi bagian dari kurva baru.
8. Hasil: Setelah selesai iterasi, kurva terakhir yang dihasilkan akan dikembalikan sebagai output.

Algoritma ini berulang kali membagi segmen-segmen kurva hingga mencapai tingkat pemisahan yang diinginkan. Semakin banyak iterasi yang dilakukan, semakin halus kurva yang dihasilkan.

Bab 4
Source Code Program

4.1. Kelas *Line* (line.py)

```python
class Line:
    def __init__(self, start: tuple[float, float], end: tuple[float, float]):
        self.start = start
        self.end = end
    def midpoint(self) -> tuple[float, float]:
        return (self.start[0] + self.end[0]) / 2, (self.start[1] + self.end[1]) / 2

    def weight(self, weight: float) -> tuple[float, float]:
        return (1 - weight) * self.start[0] + weight * self.end[0], (1 - weight) * self.start[1] + weight * self.end[1]
    def __str__(self) -> str:
        return self.start.__str__() + "," + self.end.__str__()
```

4.2. Program Pembentukan Kurva *Bezier* (bezier_curve.py)

```python
from line import Line
from math import comb
import copy

def divide_and_conquer_from_lines(control_lines: list[Line], iteration: int, weight: float) -> list[tuple[float, float]]:
    original_control_lines = copy.deepcopy(control_lines)

    while len(control_lines) > 1:
        new_lines: list[Line] = []
        for i in range(len(control_lines) - 1):
            new_lines.append(Line(control_lines[i].weight(weight), control_lines[i + 1].weight(weight)))
        control_lines = new_lines

    current_result = [control_lines[0].weight(weight)]

    if iteration == 1:
        return current_result
    else:
        first_result: list[tuple[float, float]] = divide_and_conquer_from_lines(original_control_lines, iteration - 1, weight / 2)
        second_result: list[tuple[float, float]] = divide_and_conquer_from_lines(original_control_lines, iteration - 1, 1 - weight / 2)
        return first_result + current_result + second_result

def divide_and_conquer(order: int, control_points: list[tuple[float, float]], iteration: int) -> list[tuple[float, float]]:
    # if order == 2:
        # return quadratic_divide_and_conquer(control_points, iteration)

    control_lines: list[Line] = []
    for i in range(len(control_points) - 1):
        control_lines.append(Line(control_points[i], control_points[i + 1]))
    return divide_and_conquer_from_lines(control_lines, iteration, 0.5)


def quadratic_divide_and_conquer(control_points: list[tuple[float, float]], iteration) -> list[tuple[float, float]]:
    def midpoint(first: tuple[float, float], second: tuple[float, float]):
        return (first[0] + second[0]) / 2, (first[1] + second[1]) / 2

    current_points: list[tuple[float, float]] = control_points

    for _ in range(iteration):
        midpoints: list[tuple[float, float]] = []
        midpoints.append(current_points[0])

        for i in range(1, len(current_points)):
            midpoints.append(midpoint(current_points[i], current_points[i - 1]))
        midpoints.append(current_points[-1])

        current_points: list[tuple[float, float]] = []
        result: list[tuple[float, float]] = []
        current_points.append(midpoints[0])
        current_points.append(midpoints[1])
        result.append(midpoints[0])

        for i in range(2, len(midpoints) - 1):
            result.append(midpoint(midpoints[i], midpoints[i - 1]))
            current_points.append(midpoint(midpoints[i], midpoints[i - 1]))
            current_points.append(midpoints[i])

        current_points.append(midpoints[-1])
        result.append(midpoints[-1])

    return result
```

```python
def bruteforce_general(order: int, control_points: list[tuple[float, float]]) -> list[tuple[float, float]]:
    result: list[tuple[float, float]] = []
    density = 300
    n = len(control_points)

    for d in range(density + 1):
        t = d / density
        x, y = 0, 0

        # basis
        basis_pow = (1 - t) ** (n - 1)

        for i, (px, py) in enumerate(control_points):

            # bernstein poly
            if i == 0:
                b = basis_pow
            elif t == 1:
                if i < (n-1):
                    b = 0
                else:
                    b = 1
            else:
                b *= t / (1 - t) * (n - i) / i

            x += px * b
            y += py * b

            if t != 1:
                basis_pow /= (1 - t)

        result.append((x, y))

    return result

def bruteforce_quadratic(control_points: list[tuple[float, float]]) -> list[tuple[float, float]]:
    result: list[tuple[float, float]] = []
    density = 300
    t = 0
    for _ in range(density + 1):
        # B(t) = (1-t)^2 P_0 + 2(1-t)t P_1 + t^2 P_2
        first = control_points[0][0] * (1 - t) ** 2, control_points[0][1] * (1 - t) ** 2
        second = control_points[1][0] * 2 * (1-t) * t, control_points[1][1] * 2 * (1-t) * t
        third = control_points[2][0] * t * t, control_points[2][1] * t * t
        result.append((first[0] + second[0] + third[0], first[1] + second[1] + third[1]))
        t += 1 / density

    return result
```

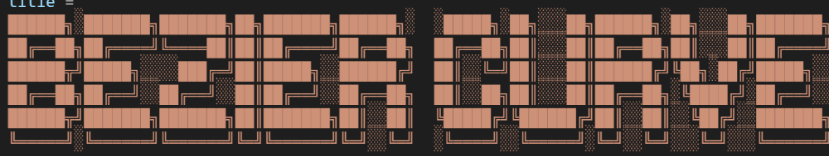
4.3. Program Visualisasi Kurva *Bezier* (bezier_visualize.py)

```python
import matplotlib.pyplot as plt
import bezier_curve
import time


def visualize(order: int, control_points: list[tuple[float, float]], iteration: int):

    # Static visualization

    for point in control_points:
        plt.scatter(point[0], point[1], s=100, color="orange")

    start_time = time.time()

    result = bezier_curve.divide_and_conquer(order, control_points, iteration)

    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Waktu eksekusi Divide and Conquer: {execution_time} detik")

    for point in result:
        plt.scatter(point[0], point[1], s=30, color="red")

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Hasil titik-titik Divide and Conquer')
    plt.grid(True)
    plt.show()


def visualize_bruteforce(order: int, control_points: list[tuple[float, float]]):
    for point in control_points:
        plt.scatter(point[0], point[1], s=100, color="orange")

    start_time = time.time()

    result = bezier_curve.bruteforce_general(order, control_points)

    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Waktu eksekusi Bruteforce: {execution_time} detik")

    for point in result:
        plt.scatter(point[0], point[1], s=5, color="red")

    plt.grid(True)
    plt.show()
```

4.4. Program Animasi Kurva *Bezier* (bezier_animation.py)

```python
# Program ini hanya untuk animasi saja

from line import Line
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

class Draw:

    # t= 0
    def __init__(self, control_points: list[tuple[float, float]]):
        self.order = len(control_points) - 1
        self.line_tree: list[list[Line]] = []
        self.cur_t = 0

        lines: list[Line] = []
        for i in range(1, len(control_points)):
            lines.append(Line(control_points[i - 1], control_points[i]))
        self.line_tree.append(lines)

        for i in range(self.order - 1):
            lines: list[Line] = []
            for j in range(self.order - i - 1):
                start_point = self.line_tree[i][j].weight(0)
                end_point = self.line_tree[i][j + 1].weight(0)
                lines.append(Line(start_point, end_point))
            self.line_tree.append(lines)


    def update(self, t: float) -> tuple[float, float]:
        if t >= 1:
            return None
        for i in range(1, self.order):
            for j in range(self.order - i):
                start_point = self.line_tree[i - 1][j].weight(t)
                end_point = self.line_tree[i - 1][j + 1].weight(t)

                self.line_tree[i][j].start = start_point
                self.line_tree[i][j].end = end_point
        return self.line_tree[self.order - 1][0].weight(t)
```

```python
print("---------------------------------------------------------------------------------------")
title = """
```



```python
"""
print(title)
print("---------------------------------------------------------------------------------------")


order = int(input("Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): "))

control_points: list[tuple[float, float]] = []
print(f"Masukkan {order + 1} titik kontrol (P_0, ..., P_{order}):")
for i in range(order + 1):
    point = map(float, input().split())
    control_points.append(tuple(point))

control_points_str = ', '.join([f'({x}, {y})' for x, y in control_points])
print("Titik kontrol:", control_points_str)


d = Draw(control_points)


def update_lines(frame):
    try:
        lines: list[Line] = []
        for i in d.line_tree:
            for j in i:
                lines.append(j)
        points.append( d.update(d.cur_t))
        d.cur_t += 0.01

        ax.clear()
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid(True)


        for point in points:
            ax.scatter(*point, color="black")

        for line in lines:
            x_values = [line.start[0], line.end[0]]
            y_values = [line.start[1], line.end[1]]
            ax.plot(x_values, y_values)
    except:
        pass

points: list[tuple[float, float]] = []

fig, ax = plt.subplots()
ax.set_aspect('equal')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Plot of Lines')
plt.grid(True)

ani = FuncAnimation(fig, update_lines)

plt.show()
```

4.5. Program Utama (main.py)

```python
import bezier_visualize

def print_title():
    print("--------------------------------------------------------------------------------------------")
    title = """
 ▄▄▄▄  ▄▄▄▄  ▄▄▄▄ ▄▄▄▄▄ ▄▄▄▄  ▄▄▄▄▄ ▄▄▄▄  ▄▄▄▄  ▄▄▄▄▄ ▄▄▄▄  ▄▄▄▄▄ ▄▄▄▄  ▄▄▄▄▄ ▄▄▄▄
 ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄ ▀▀▀▄
 ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀ ▄▄▄▀
 ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀ ▀▀▀▀
"""
    print(title)
    print("--------------------------------------------------------------------------------------------")

def run_program():
    order = int(input("Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): "))

    control_points: list[tuple[float, float]] = []
    print(f"Masukkan {order + 1} titik kontrol (P_0, ..., P_{order}):")
    for i in range(order + 1):
        point = map(float, input().split())
        control_points.append(tuple(point))

    control_points_str = ', '.join([f'({x}, {y})' for x, y in control_points])
    print("Titik kontrol:", control_points_str)

    iteration = int(input("Masukkan jumlah iterasi: "))

    while True:
        print("""Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce""")

        option = int(input("Masukkan pilihan: "))
        if option in [1, 2]:
            break
        print("Opsi tidak valid. Harap masukkan angka 1 atau 2.")

    if option == 1:
        bezier_visualize.visualize(order, control_points, iteration)
    else:
        bezier_visualize.visualize_bruteforce(order, control_points)

print_title()
while True:
    # print_title()
    run_program()
    print("--------------------------------------------------------------------------------------------")
    run_again = input("Apakah Anda ingin menjalankan program lagi? (Y/N): ").strip().upper()
    print("--------------------------------------------------------------------------------------------")
    if run_again != 'Y':
        break
```

Bab 5
Uji Coba Program

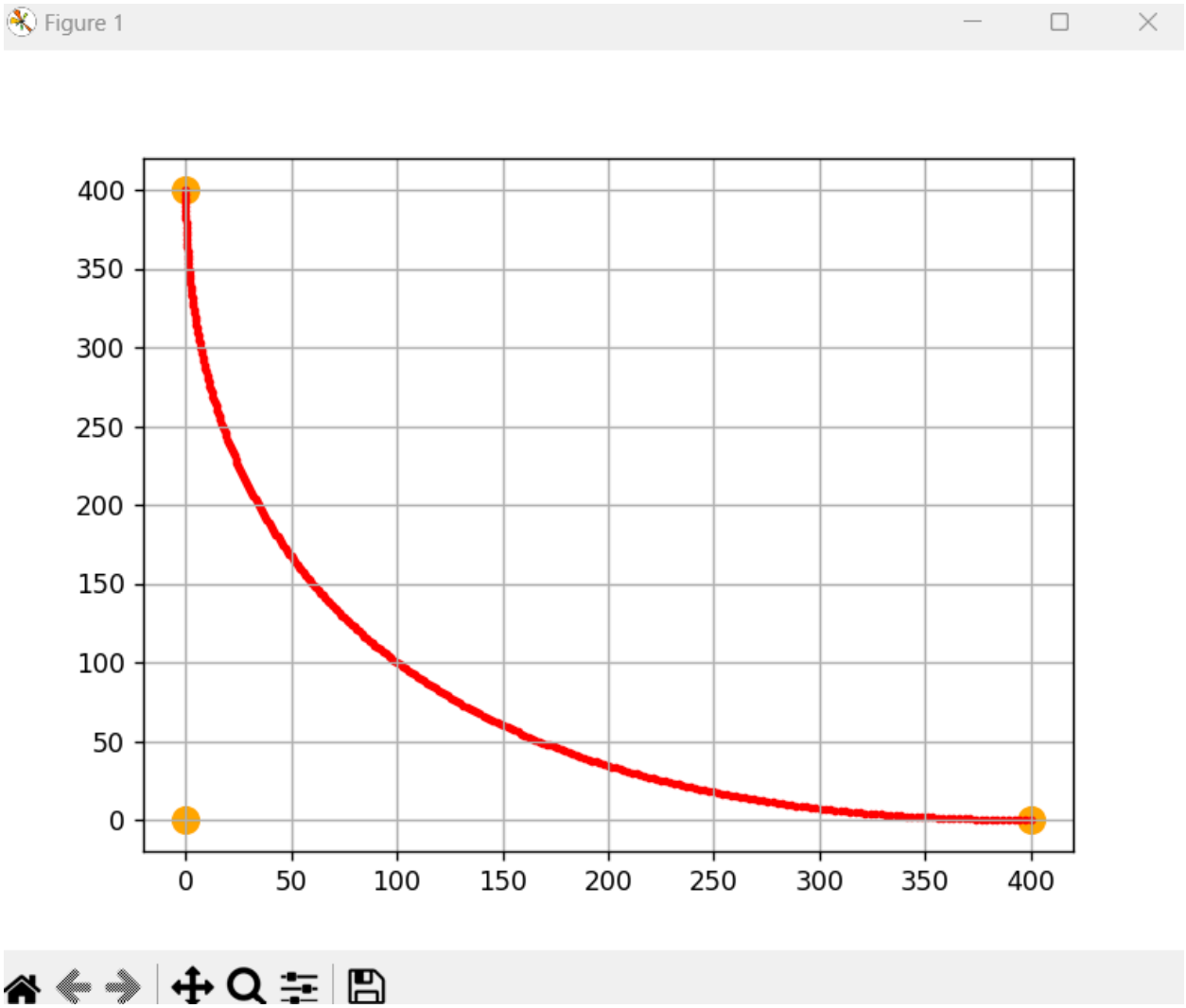
5.1. Algoritma *Brute Force*

1. Kasus Uji 1

   Input:

```
----------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 2
Masukkan 3 titik kontrol (P_0, ..., P_2):
0 400
0 0
400 0
Titik kontrol: (0.0, 400.0), (0.0, 0.0), (400.0, 0.0)
Masukkan jumlah iterasi: 5
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0 detik
----------------------------------------------------------------------------------------
```

   Output:

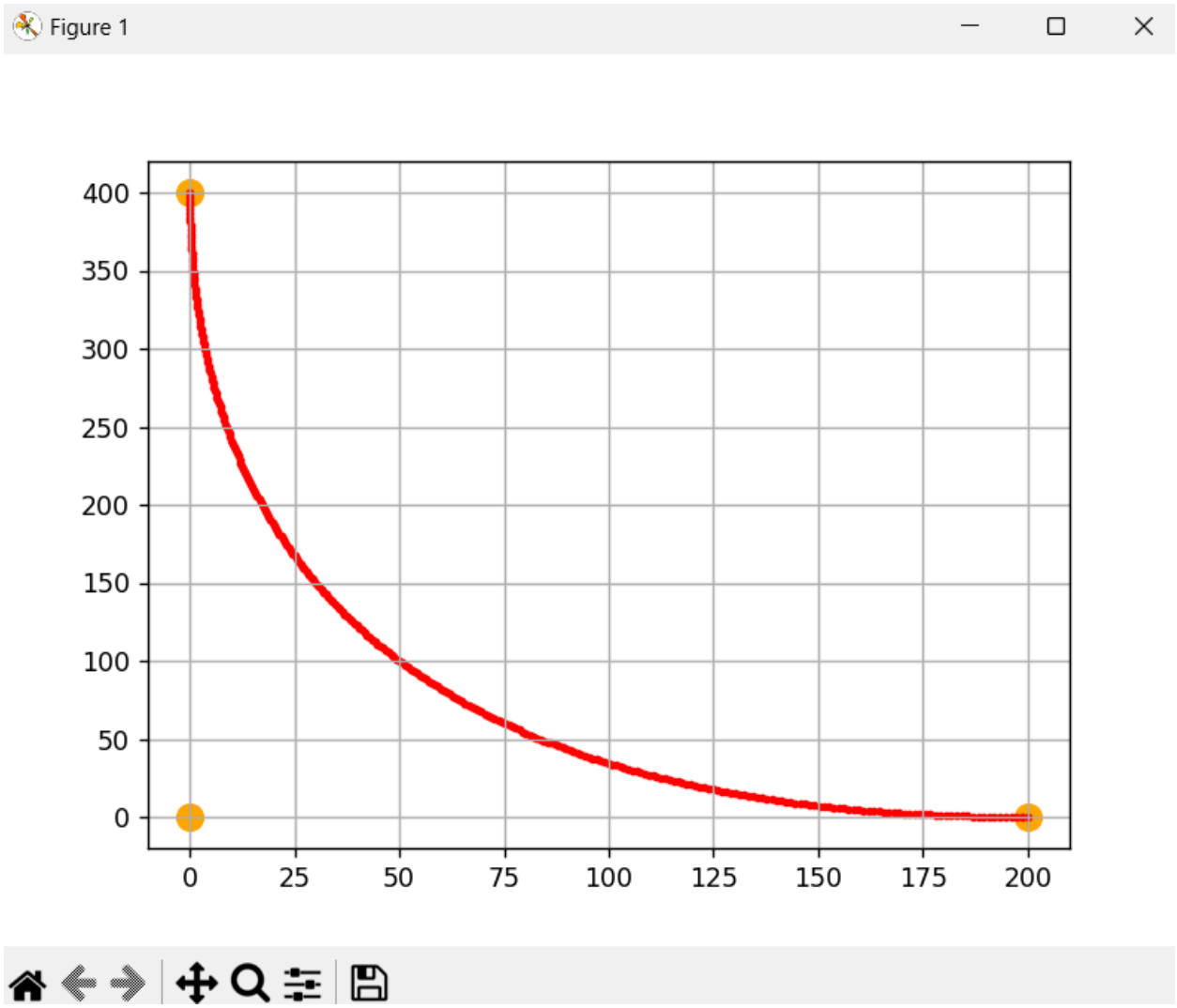


2. Kasus Uji 2

Input:

```
----------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 2
Masukkan 3 titik kontrol (P_0, ..., P_2):
0 400
0 0
200 0
Titik kontrol: (0.0, 400.0), (0.0, 0.0), (200.0, 0.0)
Masukkan jumlah iterasi: 5
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0010221004486083984 detik
----------------------------------------------------------------------------------------
```

Output:

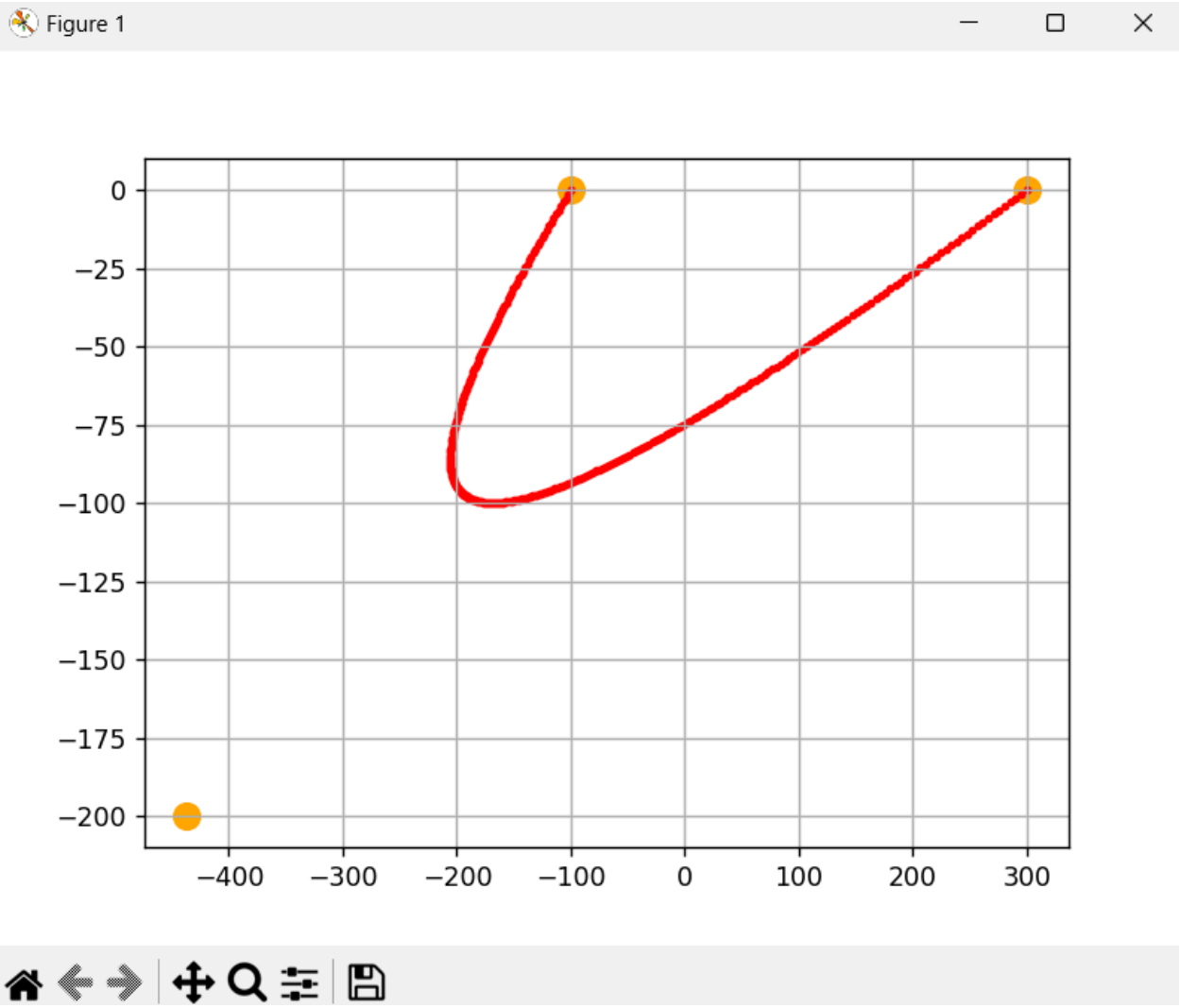


3. Kasus Uji 3

Input:

```
---------------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 2
Masukkan 3 titik kontrol (P_0, ..., P_2):
-100 0
-437 -200
300 0
Titik kontrol: (-100.0, 0.0), (-437.0, -200.0), (300.0, 0.0)
Masukkan jumlah iterasi: 5
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0 detik
---------------------------------------------------------------------------------------------
```

Output:

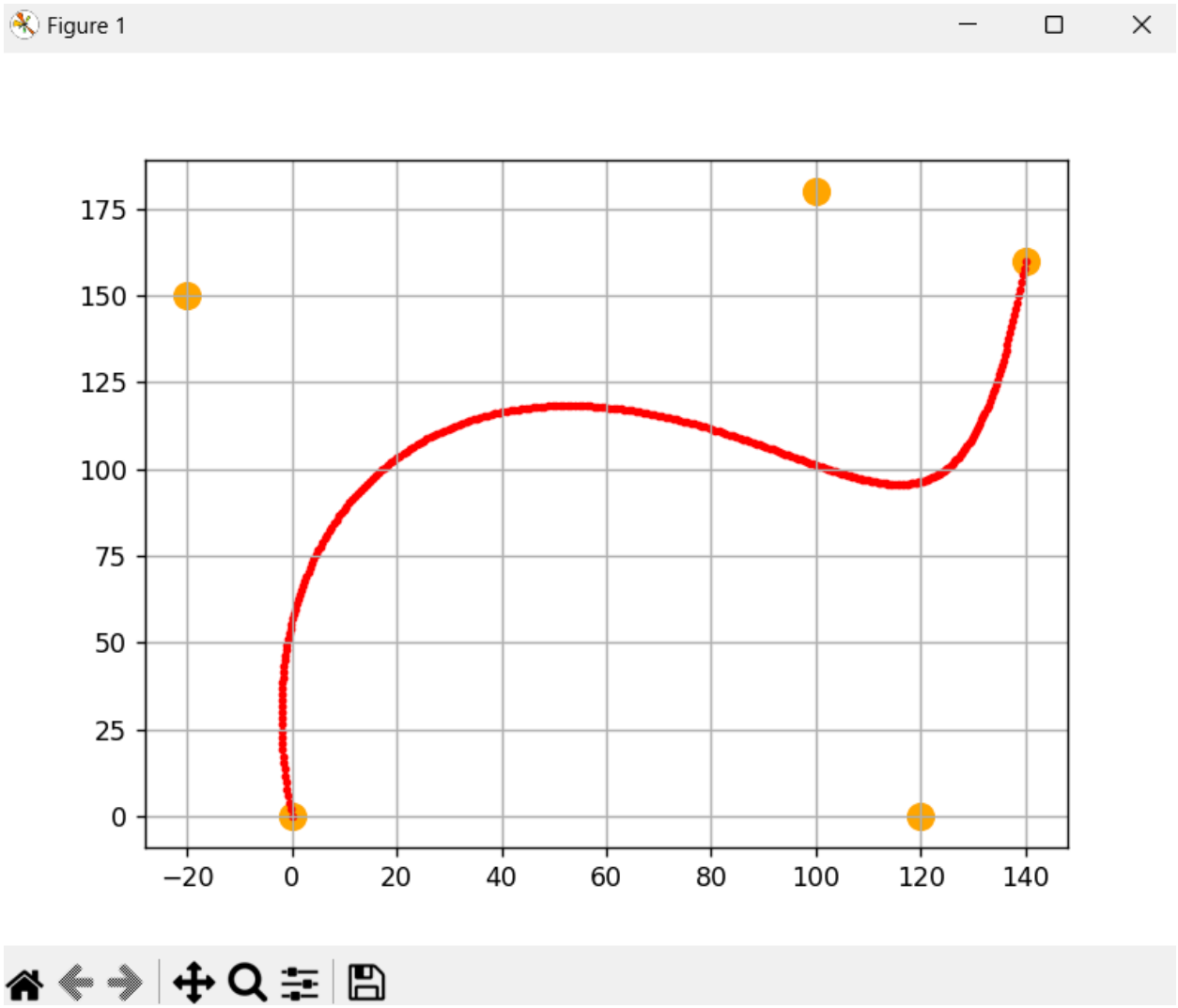


4. Kasus Uji 4

Input:

```
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 4
Masukkan 5 titik kontrol (P_0, ..., P_4):
0 0
-20 150
100 180
120 0
140 160
Titik kontrol: (0.0, 0.0), (-20.0, 150.0), (100.0, 180.0), (120.0, 0.0), (140.0, 160.0)
Masukkan jumlah iterasi: 6
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0 detik
--------------------------------------------------------------------------------
```

Output:

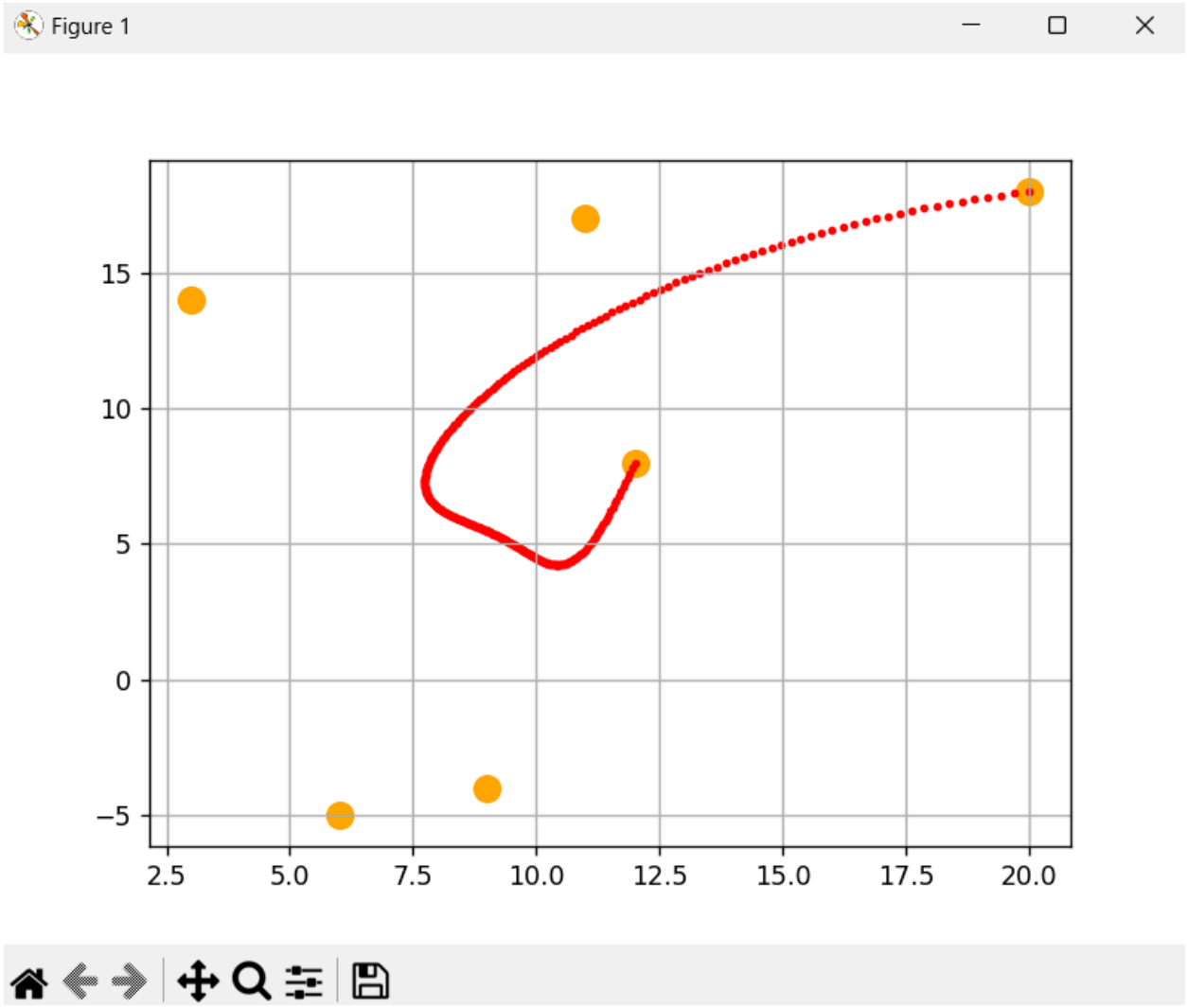


5. Kasus Uji 5

Input:

```
----------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 5
Masukkan 6 titik kontrol (P_0, ..., P_5):
12 8
9 -4
11 17
6 -5
3 14
20 18
Titik kontrol: (12.0, 8.0), (9.0, -4.0), (11.0, 17.0), (6.0, -5.0), (3.0, 14.0), (20.0, 18.0)
Masukkan jumlah iterasi: 8
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0009481906890869141 detik
----------------------------------------------------------------------------------------
```

Output:

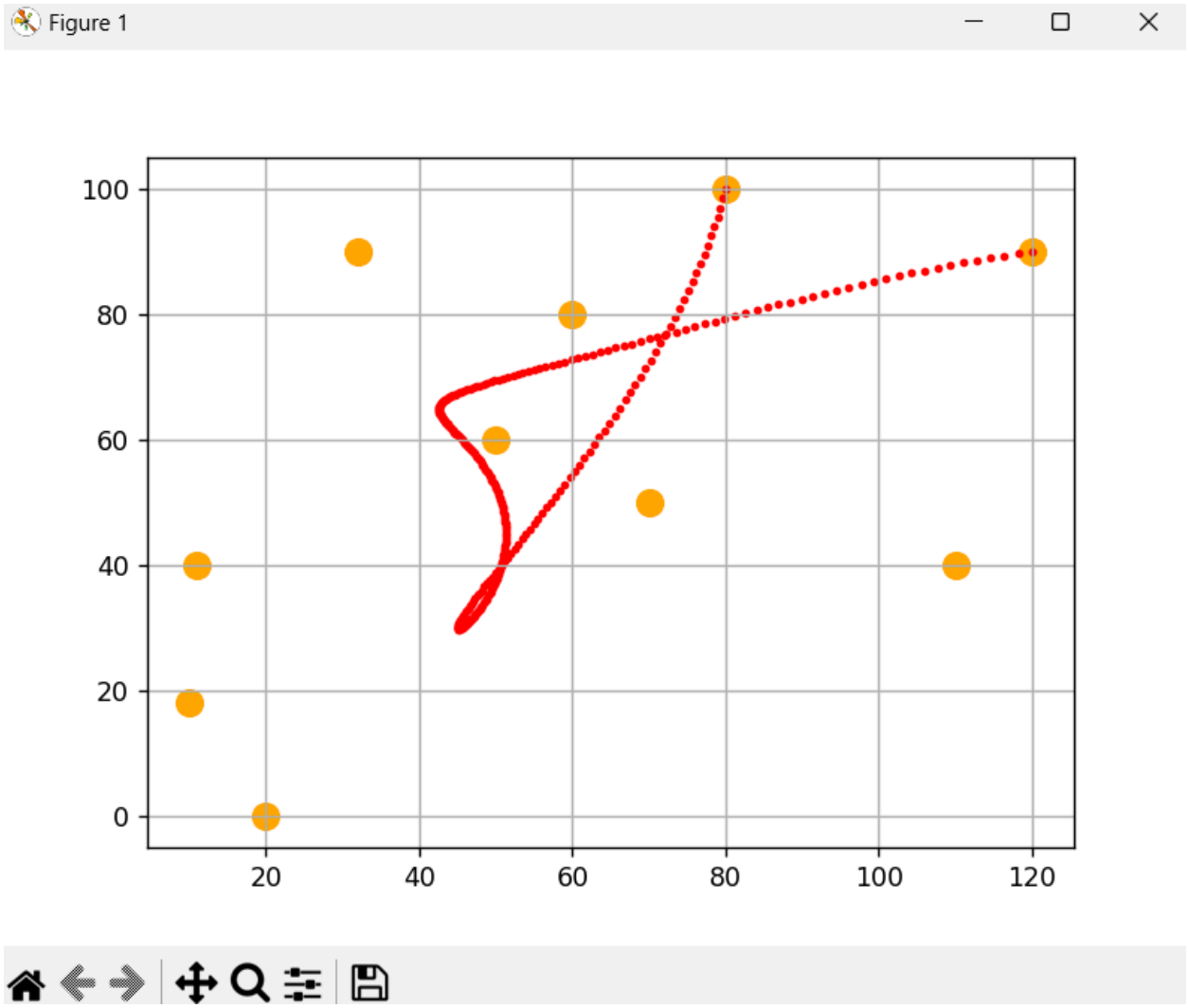


6. Kasus Uji 6

Input:

```
--------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 9
Masukkan 10 titik kontrol (P_0, ..., P_9):
80 100
70 50
20 0
10 18
110 40
50 60
32 90
11 40
60 80
120 90
Titik kontrol: (80.0, 100.0), (70.0, 50.0), (20.0, 0.0), (10.0, 18.0), (110.0, 40.0), (50.0, 60.0), (32.0, 90.0), (11.0, 40.0), (60.0, 80.0), (120.0, 90.0)
Masukkan jumlah iterasi: 10
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0009965896606445312 detik
--------------------------------------------------------------------------------
```

Output:

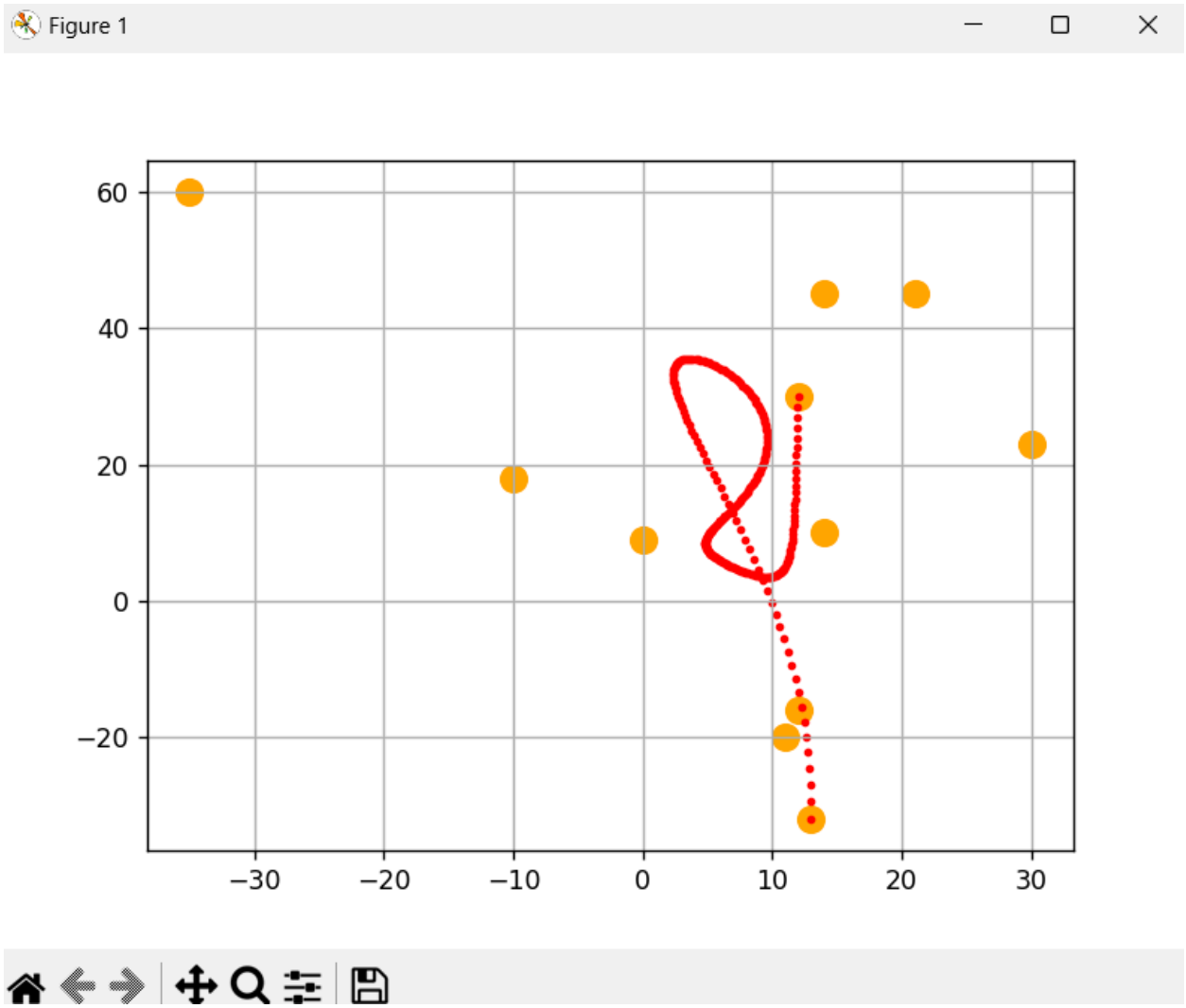


7. Kasus Uji 7

Input:

```
--------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 10
Masukkan 11 titik kontrol (P_0, ..., P_10):
12 30
11 -20
14 10
0 9
-10 18
12 -16
21 45
30 23
-35 60
14 45
13 -32
Titik kontrol: (12.0, 30.0), (11.0, -20.0), (14.0, 10.0), (0.0, 9.0), (-10.0, 18.0), (12.0, -16.0), (21.0, 45.0), (30.0, 23.0), (-35.0, 60.0), (14.0, 45.0), (13.0, -32.0)
Masukkan jumlah iterasi: 10
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.001623392105102539 detik
--------------------------------------------------------------------------------
```

Output:

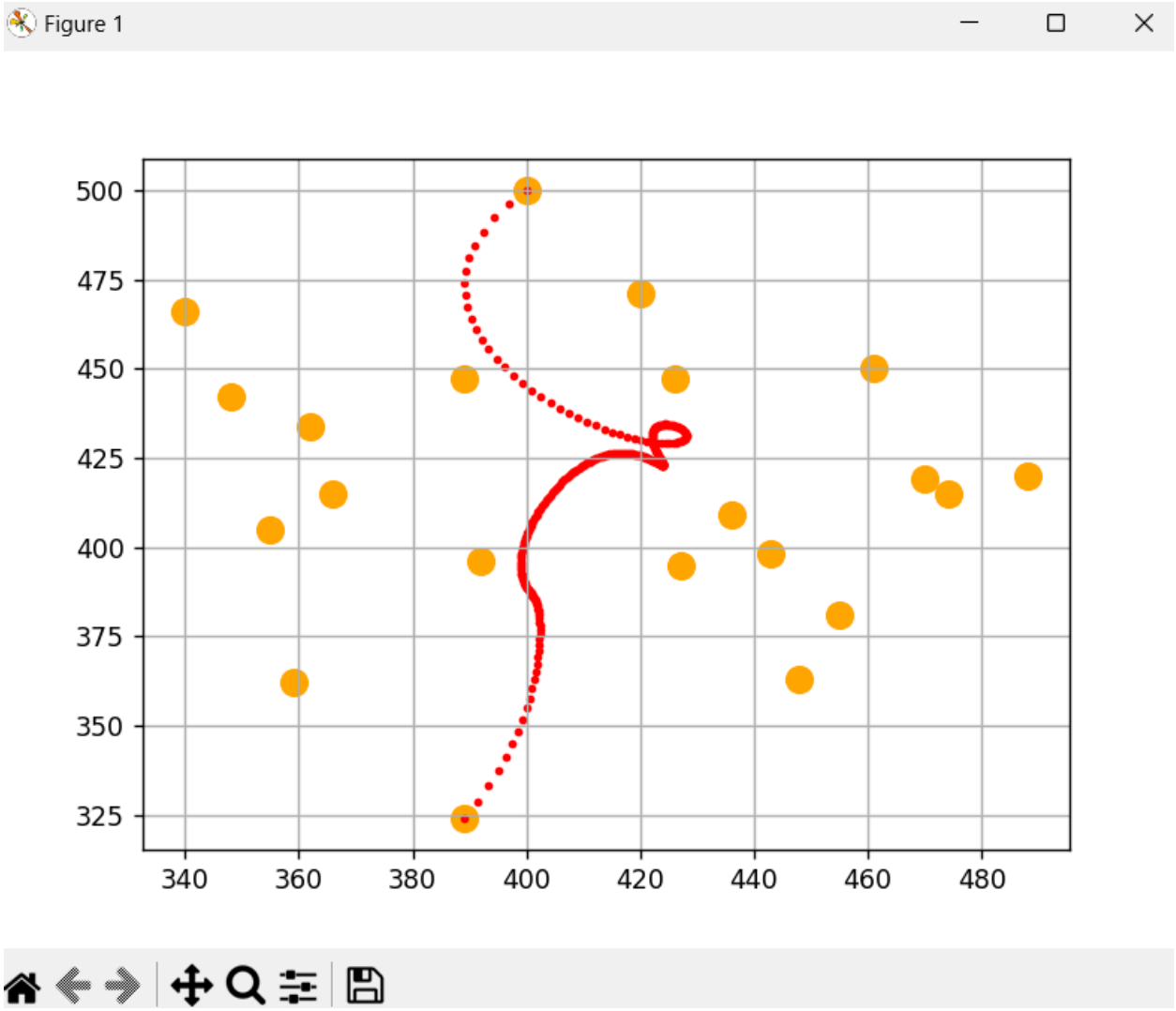


8. Kasus Uji 8

Input:

```
----------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 20
Masukkan 21 titik kontrol (P_0, ..., P_20):
389 324
427 395
366 415
448 363
359 362
389 447
443 398
355 405
426 447
461 450
362 434
436 409
470 419
392 396
474 415
340 466
420 471
488 420
455 381
348 442
400 500
Titik kontrol: (389.0, 324.0), (427.0, 395.0), (366.0, 415.0), (448.0, 363.0), (359.0, 362.0), (389.0, 447.0), (443.0, 398.0), (355.0, 405.0), (426.0, 447.0), (461.0, 450.0
), (362.0, 434.0), (436.0, 409.0), (470.0, 419.0), (392.0, 396.0), (474.0, 415.0), (340.0, 466.0), (420.0, 471.0), (488.0, 420.0), (455.0, 381.0), (348.0, 442.0), (400.0, 5
00.0)
Masukkan jumlah iterasi: 10
Pilihan pembuatan kurva Bezier
       1. Divide and Conquer
       2. Bruteforce
Masukkan pilihan: 2
Waktu eksekusi Bruteforce: 0.0009970664978027344 detik
----------------------------------------------------------------------------------------
```

Output:

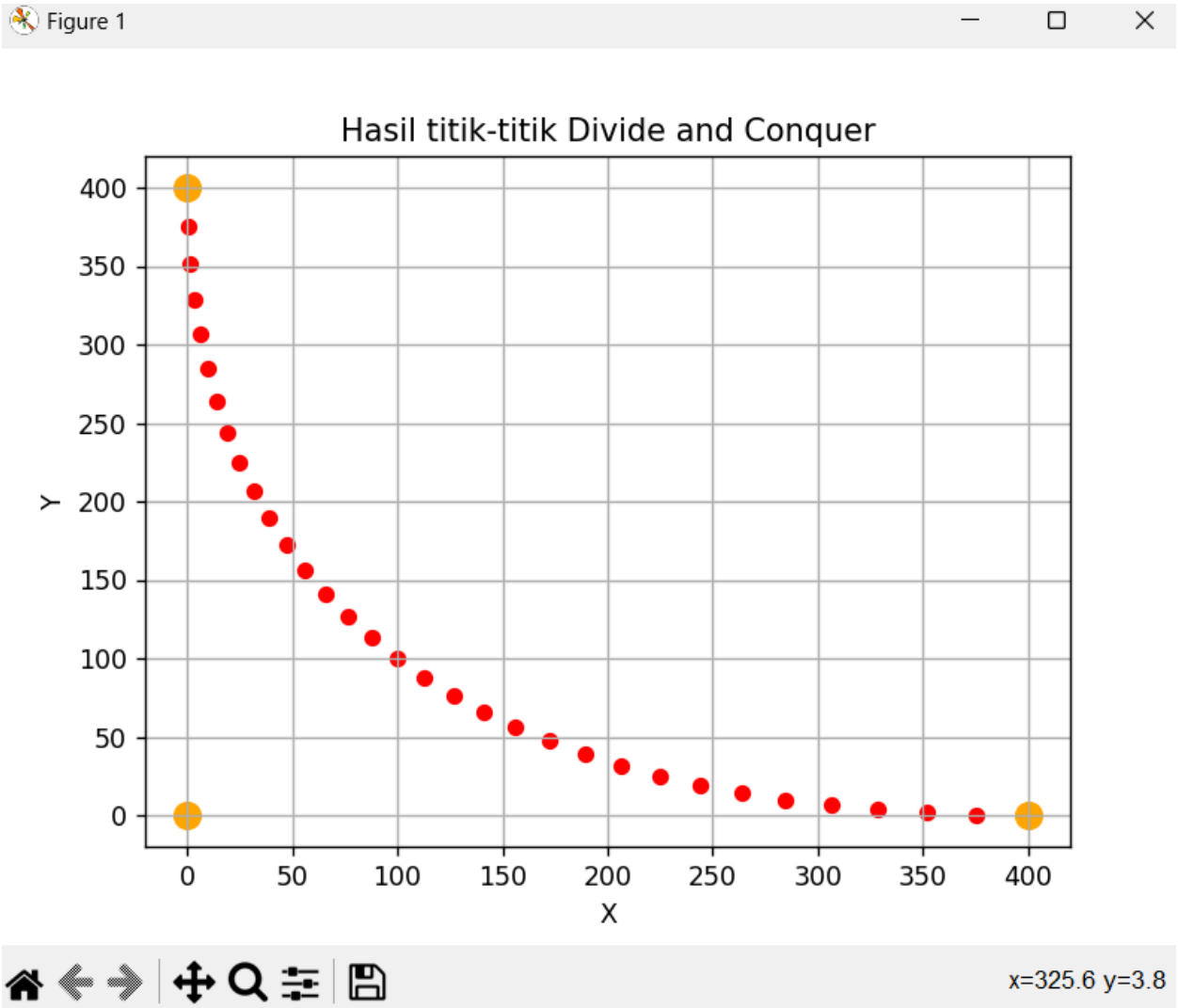


5.2. Algoritma *Divide and Conquer*

1. Kasus Uji 1
   Input:

```
--------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 2
Masukkan 3 titik kontrol (P_0, ..., P_2):
0 400
0 0
400 0
Titik kontrol: (0.0, 400.0), (0.0, 0.0), (400.0, 0.0)
Masukkan jumlah iterasi: 5
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.0010056495666503906 detik
--------------------------------------------------------------------------------
```

Output:

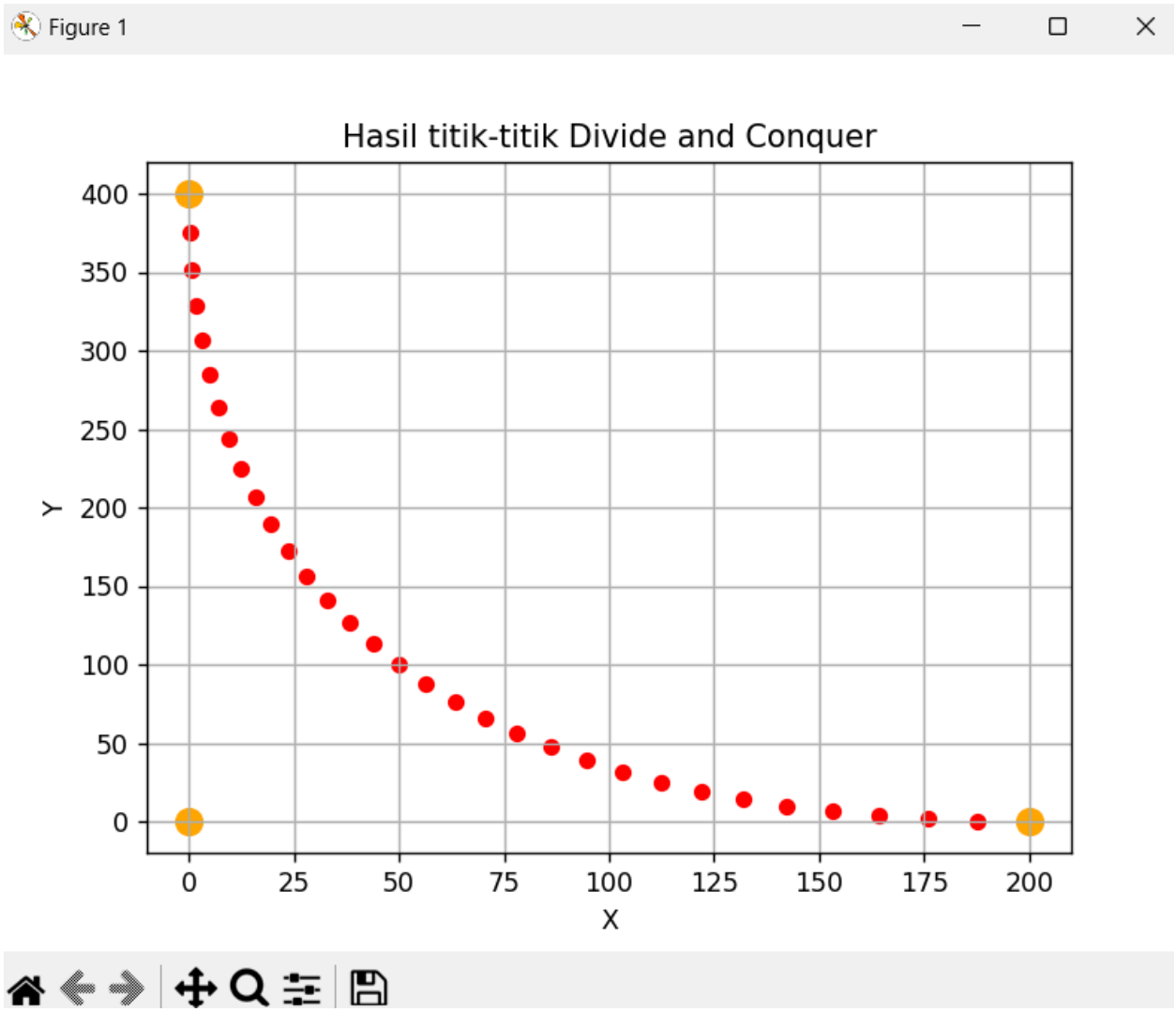


2. Kasus Uji 2

   Input:

```
--------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 2
Masukkan 3 titik kontrol (P_0, ..., P_2):
0 400
0 0
200 0
Titik kontrol: (0.0, 400.0), (0.0, 0.0), (200.0, 0.0)
Masukkan jumlah iterasi: 5
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.0 detik
--------------------------------------------------------------------------------
```

Output:

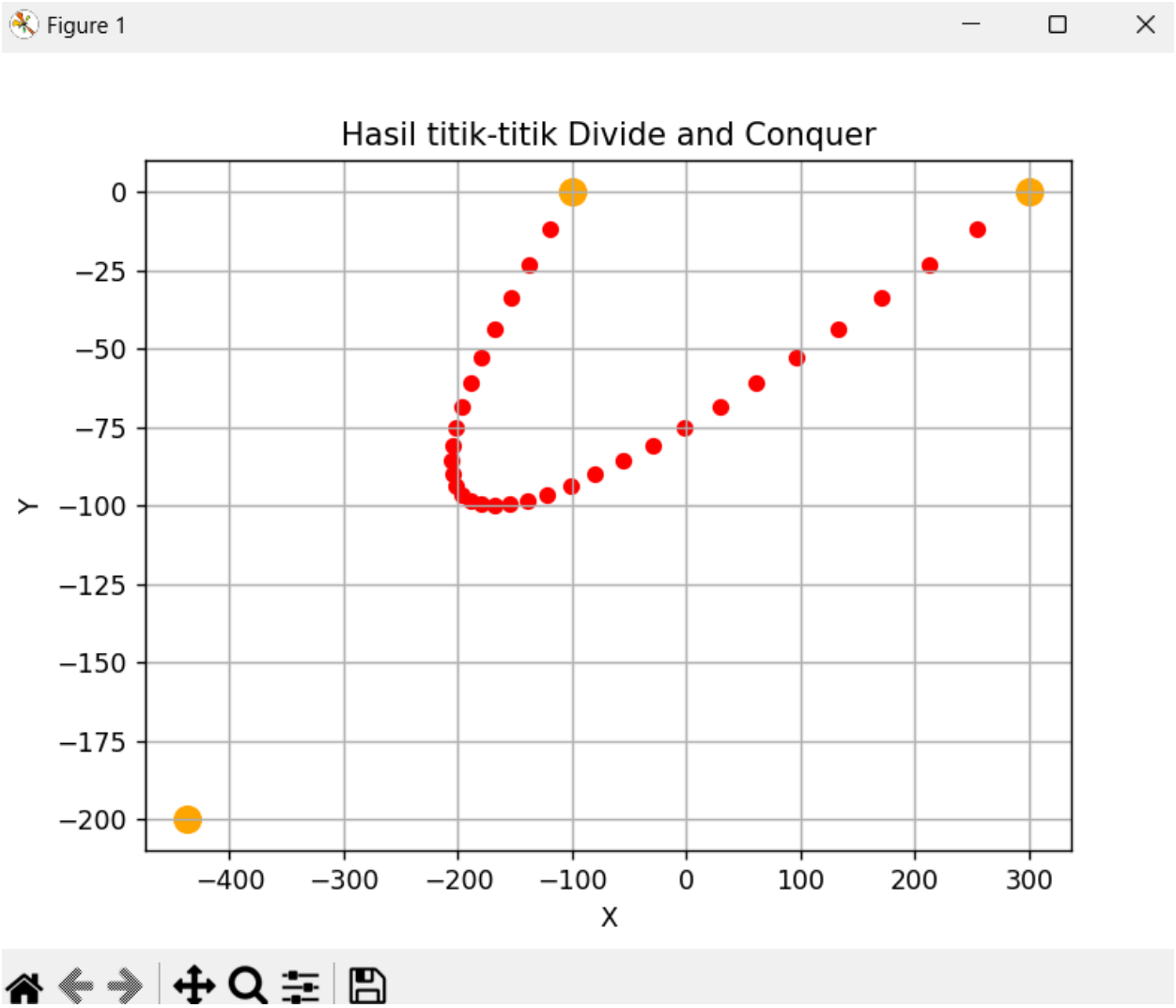


3. Kasus Uji 3

   Input:

```
----------------------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 2
Masukkan 3 titik kontrol (P_0, ..., P_2):
-100 0
-437 -200
300 0
Titik kontrol: (-100.0, 0.0), (-437.0, -200.0), (300.0, 0.0)
Masukkan jumlah iterasi: 5
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.0 detik
----------------------------------------------------------------------------------------------------
```

Output:

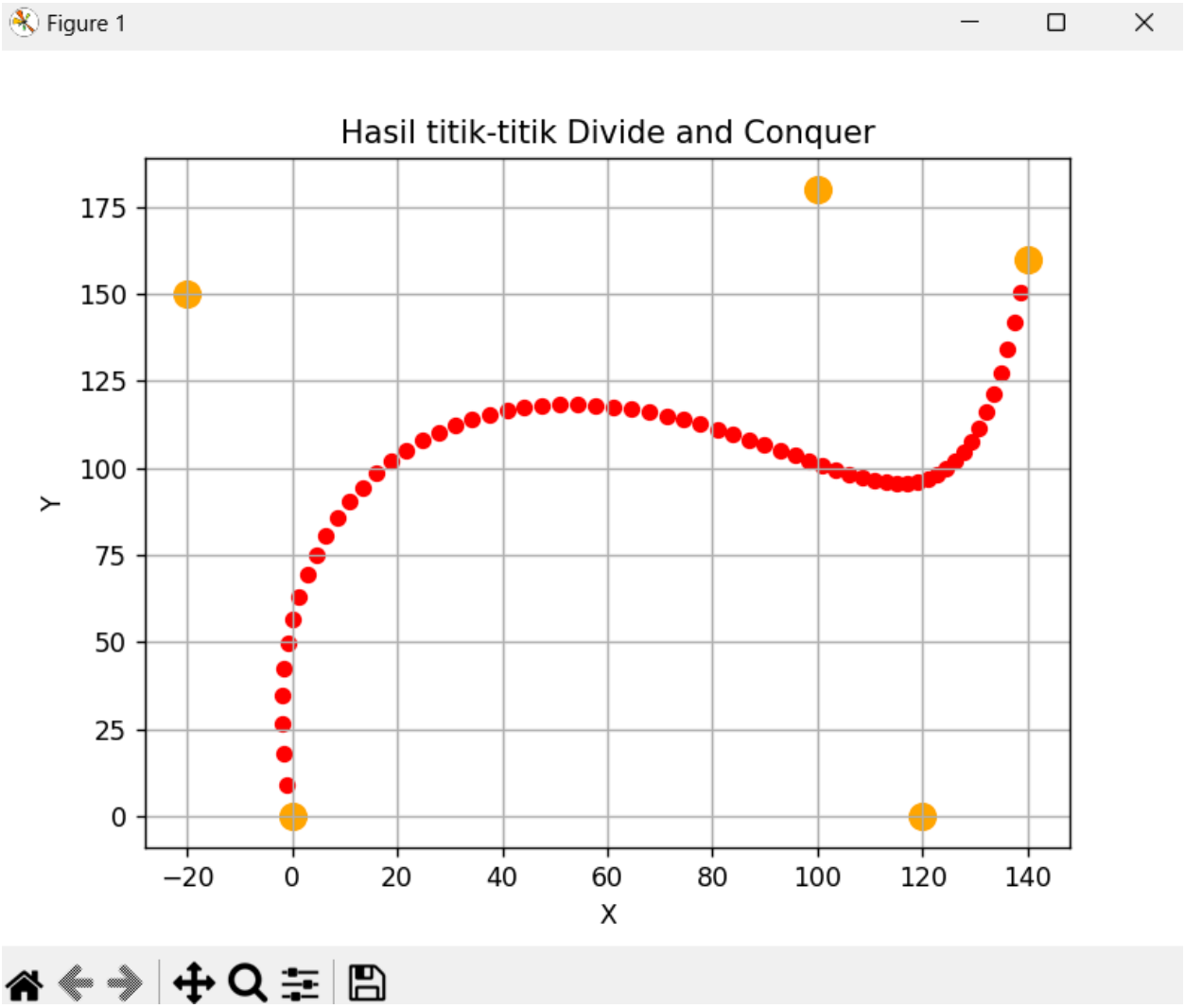


4. Kasus Uji 4

   Input:

```
----------------------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 4
Masukkan 5 titik kontrol (P_0, ..., P_4):
0 0
-20 150
100 180
120 0
140 160
Titik kontrol: (0.0, 0.0), (-20.0, 150.0), (100.0, 180.0), (120.0, 0.0), (140.0, 160.0)
Masukkan jumlah iterasi: 6
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.008203268051147461 detik
----------------------------------------------------------------------------------------------------
```

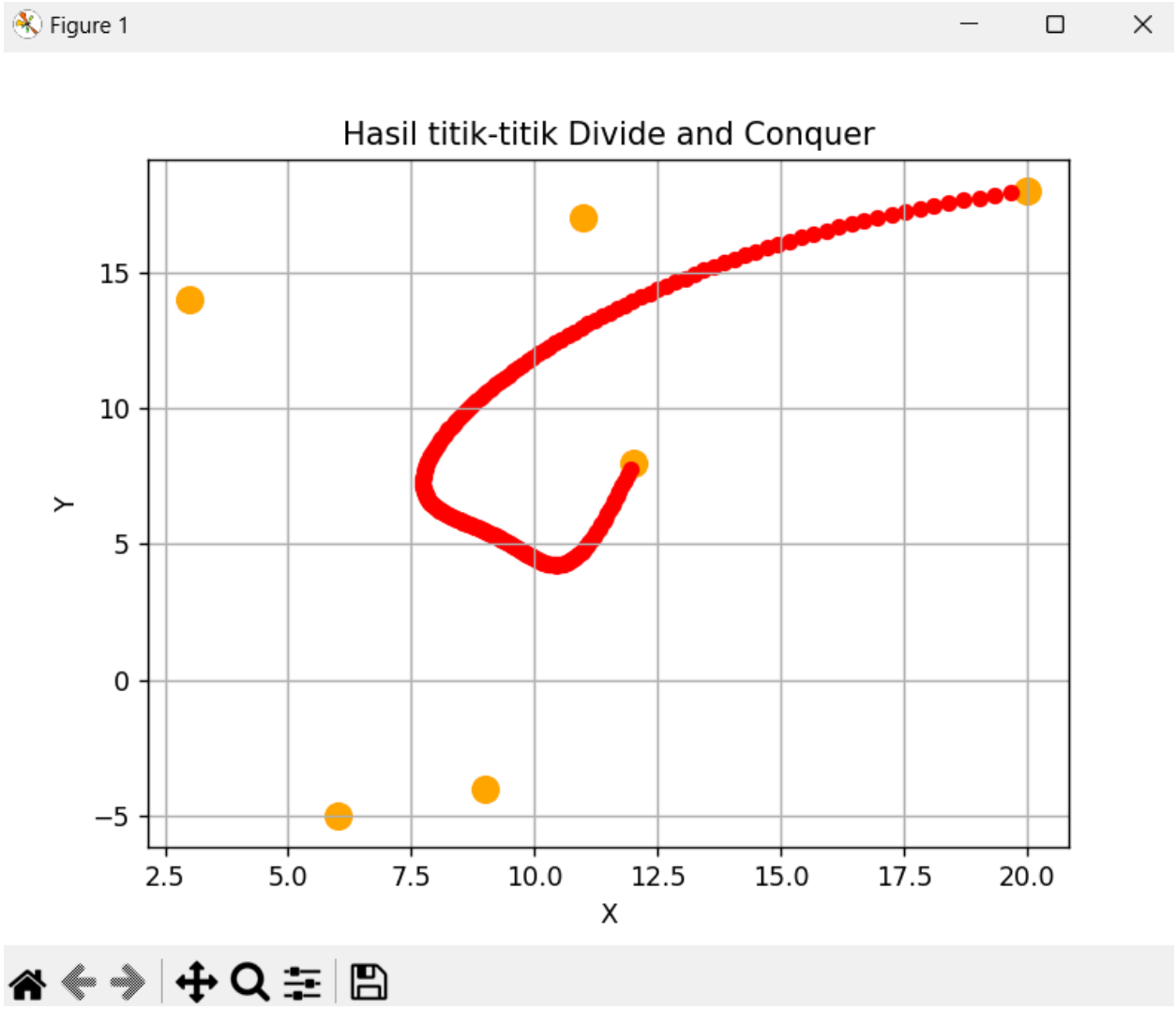
Output:



5. Kasus Uji 5
   Input:

```
--------------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 5
Masukkan 6 titik kontrol (P_0, ..., P_5):
12 8
9 -4
11 17
6 -5
3 14
20 18
Titik kontrol: (12.0, 8.0), (9.0, -4.0), (11.0, 17.0), (6.0, -5.0), (3.0, 14.0), (20.0, 18.0)
Masukkan jumlah iterasi: 8
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.01669788360595703 detik
--------------------------------------------------------------------------------------------
```

Output:

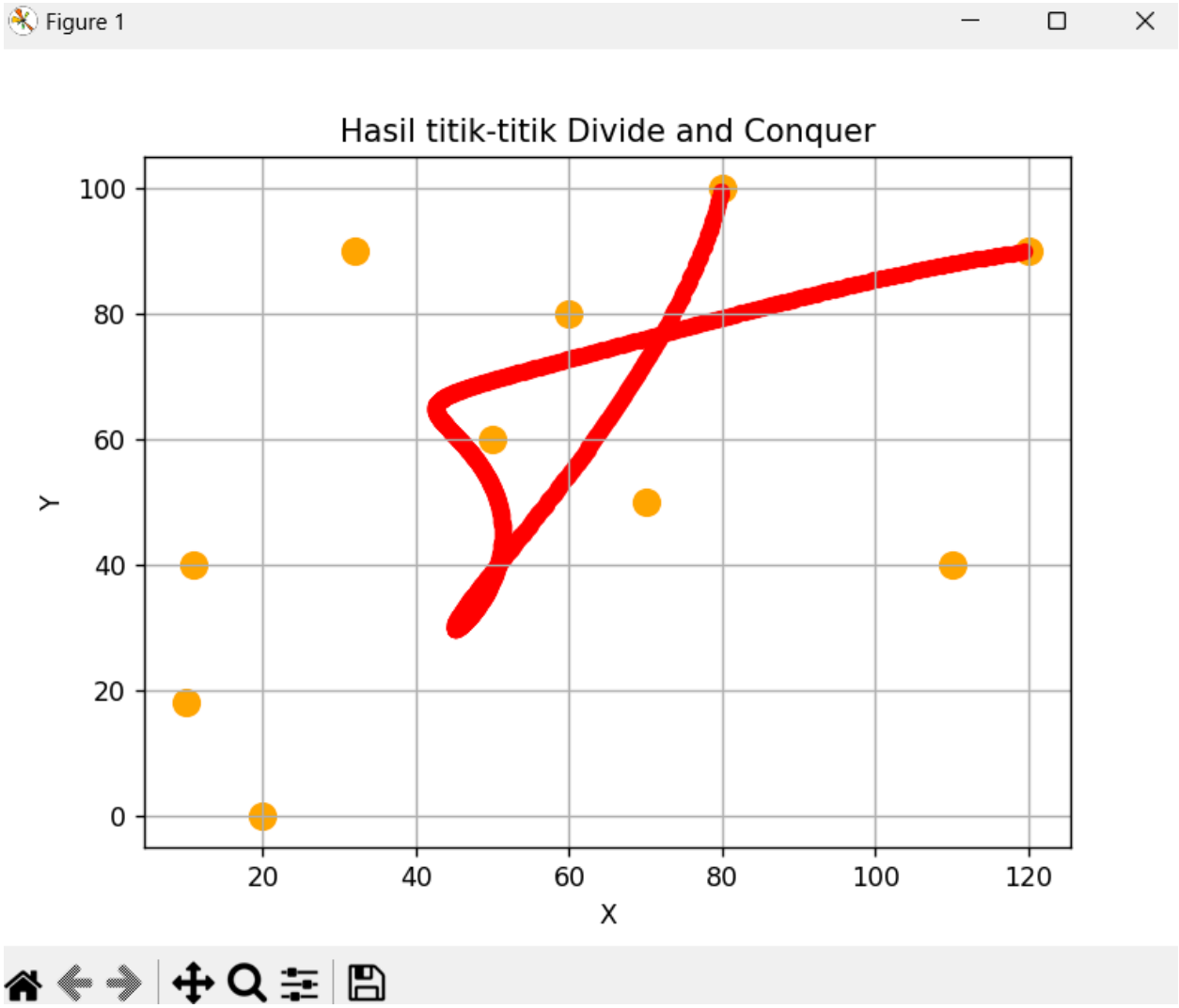


6. Kasus Uji 6

Input:

```
--------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 9
Masukkan 10 titik kontrol (P_0, ..., P_9):
80 100
70 50
20 0
10 18
110 40
50 60
32 90
11 40
60 80
120 90
Titik kontrol: (80.0, 100.0), (70.0, 50.0), (20.0, 0.0), (10.0, 18.0), (110.0, 40.0), (50.0, 60.0), (32.0, 90.0), (11.0, 40.0), (60.0, 80.0), (120.0, 90.0)
Masukkan jumlah iterasi: 10
Pilihan pembuatan kurva Bezier
       1. Divide and Conquer
       2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.15851974487304688 detik
--------------------------------------------------------------------------------
```

Output:

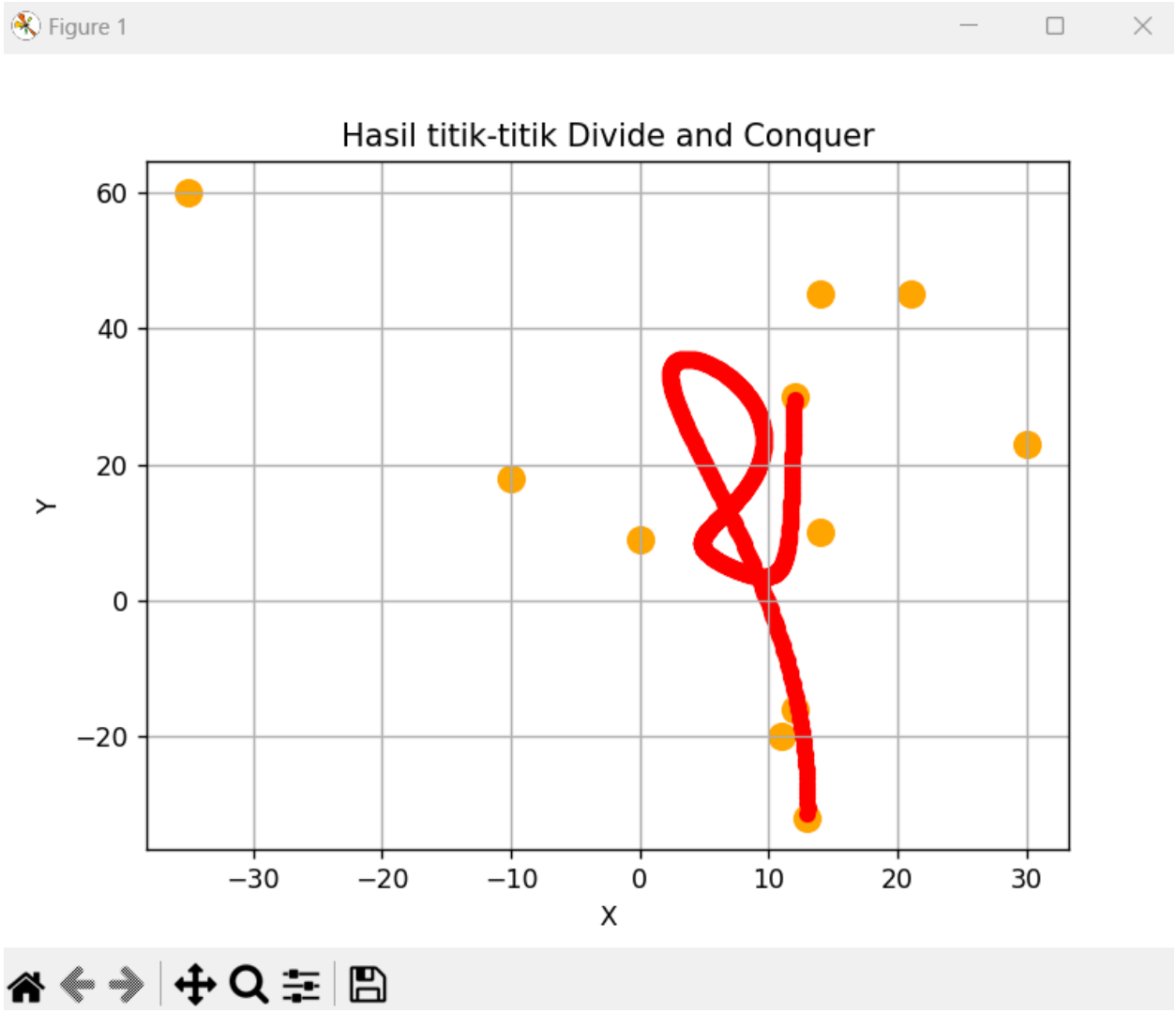


7. Kasus Uji 7

   Input:

```
----------------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 10
Masukkan 11 titik kontrol (P_0, ..., P_10):
12 30
11 -20
14 10
0 9
-10 18
12 -16
21 45
30 23
-35 60
14 45
13 -32
Titik kontrol: (12.0, 30.0), (11.0, -20.0), (14.0, 10.0), (0.0, 9.0), (-10.0, 18.0), (12.0, -16.0), (21.0, 45.0), (30.0, 23.0), (-35.0, 60.0), (14.0, 45.0), (13.0, -32.0)
Masukkan jumlah iterasi: 10
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.18997907638549805 detik
----------------------------------------------------------------------------------------
```

Output:

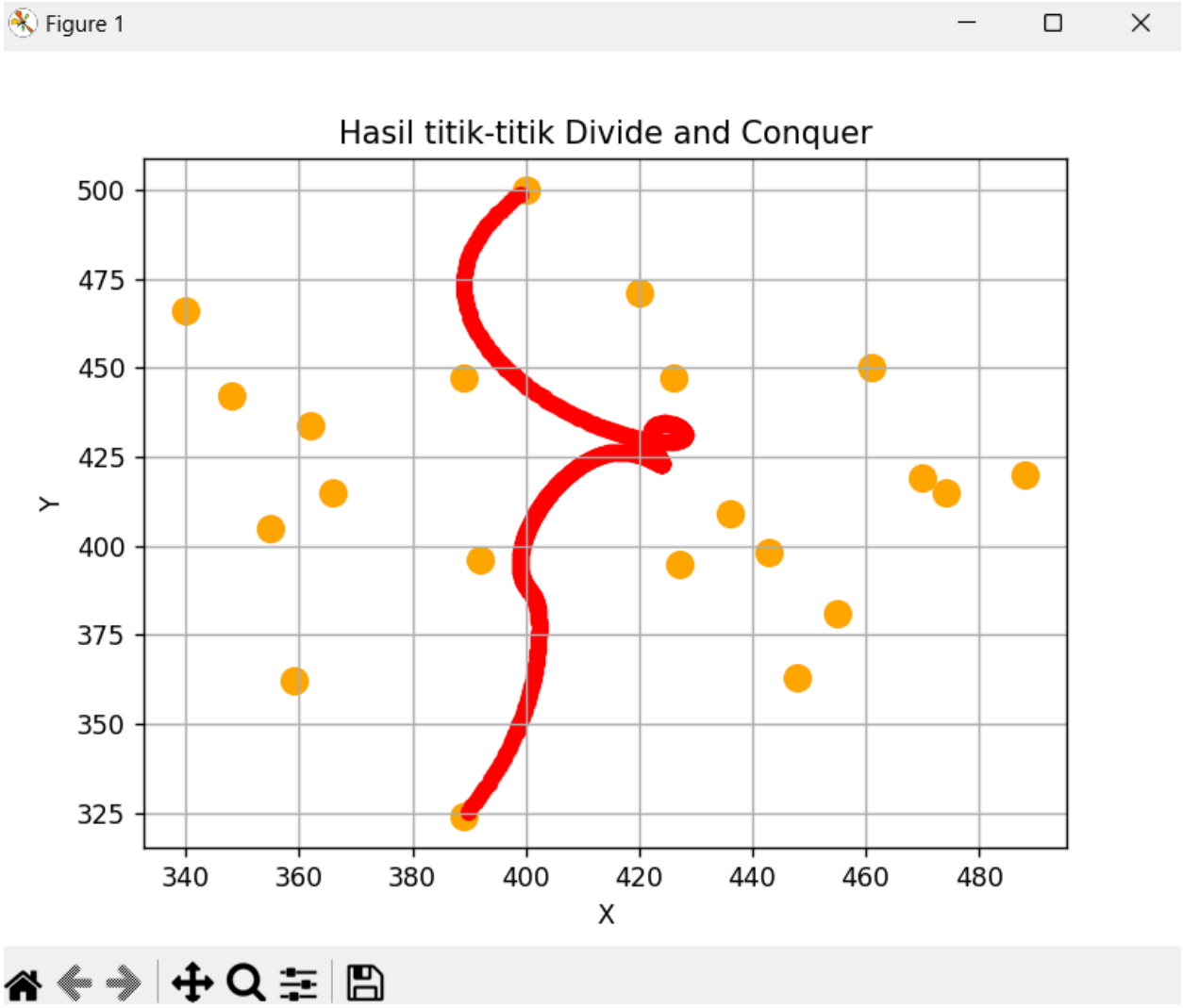


8. Kasus Uji 8

   Input:

```
--------------------------------------------------------------------------------
Masukkan order kurva n (1 untuk linier, 2 untuk quadratic, dst): 20
Masukkan 21 titik kontrol (P_0, ..., P_20):
389 324
427 395
366 415
448 363
359 362
389 447
443 398
355 405
426 447
461 450
362 434
436 409
470 419
392 396
474 415
340 466
420 471
488 420
455 381
348 442
400 500
Titik kontrol: (389.0, 324.0), (427.0, 395.0), (366.0, 415.0), (448.0, 363.0), (359.0, 362.0), (389.0, 447.0), (443.0, 398.0), (355.0, 405.0), (426.0, 447.0), (461.0, 450.0
), (362.0, 434.0), (436.0, 409.0), (470.0, 419.0), (392.0, 396.0), (474.0, 415.0), (340.0, 466.0), (420.0, 471.0), (488.0, 420.0), (455.0, 381.0), (348.0, 442.0), (400.0, 5
00.0)
Masukkan jumlah iterasi: 10
Pilihan pembuatan kurva Bezier
        1. Divide and Conquer
        2. Bruteforce
Masukkan pilihan: 1
Waktu eksekusi Divide and Conquer: 0.4123847484588623 detik
--------------------------------------------------------------------------------
```

Output:



Bab 6
Analisis solusi *Brute Force* dengan *Divide and Conquer*

Dalam analisis kompleksitas, misalkan n adalah orde dan i jumlah iterasi.

a. DnC untuk kurva Bezier orde 2 ($n = 2$)

```python
def quadratic_divide_and_conquer(control_points: list[tuple[float, float]], iteration) -> list[tuple[float, float]]:
    def midpoint(first: tuple[float, float], second: tuple[float, float]):
        return (first[0] + second[0]) / 2, (first[1] + second[1]) / 2

    current_points: list[tuple[float, float]] = control_points

    for _ in range(iteration):
        midpoints: list[tuple[float, float]] = []
        midpoints.append(current_points[0])

        for i in range(1, len(current_points)):
            midpoints.append(midpoint(current_points[i], current_points[i - 1]))
        midpoints.append(current_points[-1])

        current_points: list[tuple[float, float]] = []
        result: list[tuple[float, float]] = []
        current_points.append(midpoints[0])
        current_points.append(midpoints[1])
        result.append(midpoints[0])

        for i in range(2, len(midpoints) - 1):
            result.append(midpoint(midpoints[i], midpoints[i - 1]))
            current_points.append(midpoint(midpoints[i], midpoints[i - 1]))
            current_points.append(midpoints[i])

        current_points.append(midpoints[-1])
        result.append(midpoints[-1])

    return result
```

Perhatikan bahwa setiap iterasi, jumlah titik bertambah menjadi $\mathcal{O}(2^i)$ sehingga *space complexity*-nya adalah $\mathcal{O}(2^i)$ yang digunakan untuk menyimpan titik-titik yang dibuat.

Perhatikan bahwa kode hanya melakukan pengiterasian titik dan pembuatan titik baru dan program hanya mengiterasi setiap titik tidak lebih dari 3 kali sehingga *time complexity*-nya adalah linier terhadap banyaknya titik-titik $\mathcal{O}(2^i)$.

b. Bruteforce untuk kasus umum

```python
def bruteforce_general(order: int, control_points: list[tuple[float, float]]) -> list[tuple[float, float]]:
    result: list[tuple[float, float]] = []
    density = 300
    n = len(control_points)

    for d in range(density + 1):
        t = d / density
        x, y = 0, 0

        # basis
        basis_pow = (1 - t) ** (n - 1)

        for i, (px, py) in enumerate(control_points):

            # bernstein poly
            if i == 0:
                b = basis_pow
            elif t == 1:
                if i < (n-1):
                    b = 0
                else:
                    b = 1
            else:
                b *= t / (1 - t) * (n - i) / i

            x += px * b
            y += py * b

            if t != 1:
                basis_pow /= (1 - t)

        result.append((x, y))

    return result
```

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1} t \mathbf{P}_1 + \cdots + \binom{n}{n-1}(1-t) t^{n-1} \mathbf{P}_{n-1} + t^n \mathbf{P}_n, \qquad 0 \leqslant t \leqslant 1\end{aligned}$$

Perhatikan rumus eksplisit di atas. Perhatikan bahwa t^n dapat dihitung dalam $\mathcal{O}(n)$ (atau lebih optimal lagi dalam $\mathcal{O}(\log n)$ namun tidak dilakukan di sini). Maka secara keseluruhan $\binom{n}{i}(1-t)^{n-i}t^i$ dapat dihitung di setiap iterasi di loop dengan $\mathcal{O}(1)$ dengan mengalikan dengan $\frac{t}{1-t} \cdot \frac{n-i}{i}$ di setiap iterasi.

Maka $B(t)$ untuk suatu t dapat dihitung dengan time complexity $\mathcal{O}(n)$ dan dengan space complexity $\mathcal{O}(1)$ untuk menyimpan variabel koefisien terhadap $\mathbf{P_i}$ di setiap iterasi.

Bonus

c. DnC untuk kasus umum

```python
def divide_and_conquer_from_lines(control_lines: list[Line], iteration: int, weight: float) -> list[tuple[float, float]]:
    original_control_lines = copy.deepcopy(control_lines)

    while len(control_lines) > 1:
        new_lines: list[Line] = []
        for i in range(len(control_lines) - 1):
            new_lines.append(Line(control_lines[i].weight(weight), control_lines[i + 1].weight(weight)))
        control_lines = new_lines

    current_result = [control_lines[0].weight(weight)]

    if iteration == 1:
        return current_result
    else:
        first_result: list[tuple[float, float]] = divide_and_conquer_from_lines(original_control_lines, iteration - 1, weight / 2)
        second_result: list[tuple[float, float]] = divide_and_conquer_from_lines(original_control_lines, iteration - 1, 1 - weight / 2)
        return first_result + current_result + second_result
```

Dengan cara yang sama seperti sebelumnya, dapat diperoleh bahwa *space complexity* dari kode adalah $\mathcal{O}(2^i)$.

Perhatikan bahwa perhitungan untuk perhitungan setiap titik membutuhkan waktu $\mathcal{O}(n^2)$ dengan algoritma de Casteljau (dapat dipercepat menggunakan teknik lain sehingga membutuhkan waktu $\mathcal{O}(n)$, namun tidak dilakukan di sini). Perhatikan juga bahwa, melakukan konkatenasi list membutuhkan waktu linier time sehingga membutuhkan waktu $\mathcal{O}(2^i)$. Maka, *time complexity* dirumuskan oleh relasi rekurens berikut:

$$T(i) = 2T(i-1) + c_1 n^2 + c_2 2^i$$

dan dapat diselesaikan seperti berikut

$$\begin{aligned} & T(i) = 2T(i-1) + c_1 n^2 + c_2 2^i \\ &= 2(2T(i-2)+c_1 n^2+c_2 2^{i-1})+c_1 n^2+c_2 2^i = 2^2 T(i-2)+(1+2)c_1 n^2+c_2(2^i+2^i) \\ &= \dots \\ &= 2^i T(0) + (1+2+\cdots+2^{i-1})c_1 n^2 + c_2(2^i+2^i+\cdots+2^i) \\ &= 2^i T(0) + (2^i-1)c_1 n^2 + c_2(2^i i) = \mathcal{O}(2^i(n^2+i)) \end{aligned}$$

di mana $T(0)$ membutuhkan waktu $\mathcal{O}(n^2)$.

Secara intuitif, makna dari relasi rekurens terhadap kode di atas adalah

1. $2T(i-1)$ memanggil dirinya sendiri
2. $\mathcal{O}(n^2)$ untuk algoritma de Casteljau
3. $\mathcal{O}(2^i)$ untuk melakukan konkatenasi list first_result + current_result + second_result

Bab 7
Implementasi Bonus

7.2. Implementasi Algoritma *Brute Force* dan *Divide and Conquer* secara general

7.2.1. Algoritma Brute Force

1. Input:
    - Input berupa banyaknya *control points* dan daftar tupel yang berisi koordinat titik kontrol. Setiap tupel berisi dua nilai: koordinat x dan koordinat y.
2. Precompute Binomial Coefficients:
    - Melakukan pre komputasi koefisien binomial untuk digunakan dalam perhitungan polinomial Bernstein. Ini dilakukan untuk meningkatkan efisiensi perhitungan.
3. Fungsi Bernstein Polynomial:
    - Fungsi untuk menghitung nilai polinomial Bernstein ke-i dari orde-n pada titik evaluasi t. Polinomial Bernstein digunakan dalam interpolasi Bezier untuk memberikan bobot pada setiap *control point*.

$$\begin{aligned}\mathbf{B}(t) &= \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \\ &= (1-t)^n \mathbf{P}_0 + \binom{n}{1}(1-t)^{n-1} t \mathbf{P}_1 + \cdots + \binom{n}{n-1}(1-t) t^{n-1} \mathbf{P}_{n-1} + t^n \mathbf{P}_n, \qquad 0 \leqslant t \leqslant 1\end{aligned}$$

4. Iterasi Melalui Parameter t:
    - Program melakukan iterasi melalui nilai parameter t dari 0 hingga 1 dengan interval yang ditentukan oleh variabel *density*. Semakin tinggi density, semakin halus kurva yang dihasilkan.
    - Untuk setiap nilai t, program menghitung titik pada kurva Bezier menggunakan polinomial Bernstein dan bobotnya.
5. Perhitungan Titik pada Kurva Bezier:
    - Untuk setiap nilai t, program menghitung koordinat x dan y pada kurva Bezier dengan menggunakan polinomial Bernstein.
    - Koordinat x dan y dihitung dengan menjumlahkan hasil perkalian bobot polinomial Bernstein dengan koordinat x dan y dari setiap *control point*.
6. Penyimpanan Hasil:
    - Setiap titik yang dihasilkan dari perhitungan ditambahkan ke dalam list result.
7. Output:

- Program mengembalikan list result yang berisi semua titik yang membentuk kurva Bezier.

Dengan cara ini, prosedur program ini secara *brute force* menghitung titik-titik pada kurva Bezier dengan mengiterasi melalui semua nilai parameter t dan menghitung koordinat untuk setiap titik.

7.2.2. Algoritma Divide and Conquer

Prosedur umum dalam algoritma Divide and Conquer, di mana kurva dibagi menjadi dua bagian, kemudian solusi untuk setiap bagian dibangun secara rekursif, dan akhirnya digabungkan kembali. Dalam konteks ini, kurva Bezier direpresentasikan sebagai serangkaian garis kontrol, yang kemudian dibagi dan diaproksimasi dengan bobot tertentu untuk setiap segmen garis. Prosedur pembentukan kurva Bezier dengan algoritma divide and conquer adalah sebagai berikut:

1. Fungsi divide_and_conquer_from_lines
   Fungsi ini bertugas untuk mengaproksimasi kurva Bezier dari serangkaian garis kontrol. Input berupa control_lines (daftar garis kontrol), iteration (jumlah iterasi yang ingin dilakukan), dan weight (bobot yang digunakan dalam pembentukan kurva Bezier).

   Prosedur yang dilakukan pada fungsi ini adalah:

   - Fungsi ini pertama-tama menduplikasi control_lines asli untuk digunakan dalam iterasi.
   - Selama jumlah garis kontrol masih lebih dari satu, fungsi akan membuat garis-garis baru dengan bobot yang diberikan menggunakan nilai weight, dan mengganti control_lines dengan garis-garis baru tersebut.
   - Setelah iterasi selesai, hasil akhir dari aproksimasi kurva Bezier disimpan dalam current_result.
   - Jika iterasi adalah 1, maka fungsi langsung mengembalikan current_result.
   - Jika tidak, fungsi melakukan pemanggilan rekursif terhadap dirinya sendiri untuk membagi kurva menjadi dua bagian, yaitu bagian kiri dan bagian kanan, dan menggabungkan hasilnya.

2. Fungsi divide_and_conquer
   Fungsi ini merupakan fungsi yang akan dipanggil untuk memulai proses pembentukan kurva Bezier menggunakan algoritma Divide and Conquer. Input berupa order (orde dari kurva Bezier), control_points (daftar titik kontrol), dan iteration (jumlah iterasi). Prosedur yang dilakukan adalah:

- Fungsi ini pertama-tama menginisialisasi daftar garis kontrol berdasarkan titik-titik kontrol yang diberikan.
- Kemudian, ia memanggil fungsi divide_and_conquer_from_lines dengan daftar garis kontrol, jumlah iterasi, dan bobot awal yang ditetapkan ke 0.5.
- Hasil dari fungsi divide_and_conquer_from_lines kemudian dikembalikan sebagai hasil akhir dari pembentukan kurva Bezier.

7.3. Animasi Pembentukan Kurva *Bezier*

Program ini adalah animasi untuk menggambar kurva *Bezier*, berikut adalah implementasinya:

1. Input Titik Kontrol: Pengguna diminta untuk memasukkan titik kontrol kurva. Jumlah titik kontrol ini ditentukan oleh variabel order, yang menentukan orde dari kurva yang akan digambar.
2. Interpolasi Kurva: Kurva digambar dengan menggunakan interpolasi dari titik kontrol yang diberikan. Metode interpolasi yang digunakan tidak secara eksplisit disebutkan dalam kode, tetapi mungkin menggunakan metode seperti interpolasi linier atau interpolasi polinomial.
3. Animasi: Animasi dilakukan dengan cara mengupdate posisi titik pada kurva dalam setiap frame animasi. Setiap kali frame diupdate, kurva akan berevolusi lebih dekat ke bentuk akhirnya.
4. Libraries: Program menggunakan beberapa library, seperti numpy untuk operasi numerik, matplotlib untuk plotting, dan FuncAnimation untuk animasi.
5. Struktur Program: Program terdiri dari kelas draw yang bertanggung jawab untuk menggambar kurva berdasarkan titik kontrol yang diberikan, serta fungsi update_lines yang mengatur animasi dan update kurva pada setiap frame.
6. Output: Output dari program ini adalah animasi yang menunjukkan proses pembentukan kurva berdasarkan titik kontrol yang diberikan.

Bab 8
Lampiran

Poin	Ya	Tidak
Program berhasil dijalankan.	✓	
Program dapat melakukan visualisasi kurva Bézier.	✓	
Solusi yang diberikan program optimal.	✓	
[Bonus] Program dapat membuat kurva untuk *n* titik kontrol.	✓	
[Bonus] Program dapat melakukan visualisasi proses pembuatan kurva.	✓	

Link *repository* : https://github.com/randyver/Tucil2_13522067_13522109