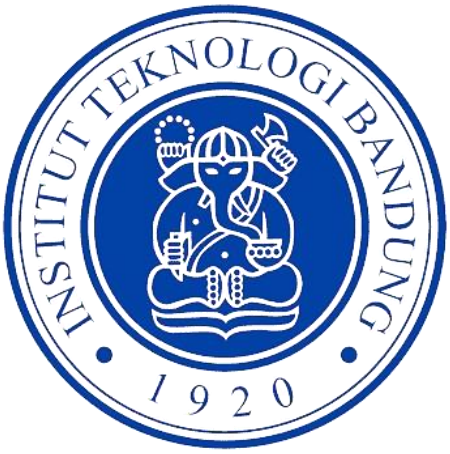
IF2211 STRATEGI ALGORITMA
LAPORAN TUGAS KECIL II

MEMBANGUN KURVA BEZIER DENGAN ALGORITMA TITIK TENGAH BERBASIS DIVIDE AND CONQUER



Disusun Oleh:

Karunia Syukur Baeha	**10023478**
Rafii Ahmad Fahreza	**10023570**

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

BAB I PENDAHULUAN

Kurva Bézier adalah salah satu konsep fundamental dalam matematika yang diterapkan secara luas dalam berbagai bidang, termasuk desain grafis, animasi, dan manufaktur. Karena sifatnya yang halus dan fleksibel, kurva Bézier memungkinkan para desainer untuk menciptakan bentuk-bentuk yang kompleks dan menarik dengan mudah. Proses pembuatan kurva Bézier dimulai dengan menentukan serangkaian titik kontrol. Titik kontrol ini berperan penting dalam menentukan bentuk dan arah dari kurva yang dihasilkan. Secara umum, kurva Bézier dibentuk oleh garis yang menghubungkan titik kontrol awal dan akhir, dengan titik kontrol tambahan yang mempengaruhi bentuk kurva di antara keduanya. Penting untuk dicatat bahwa titik-titik kontrol tambahan tidak selalu berada pada kurva itu sendiri, mereka hanya mempengaruhi bentuk kurva yang dihasilkan.

Sebagai contoh, dengan hanya dua titik kontrol, kita dapat membentuk kurva Bézier sederhana yang berupa garis lurus. Namun, dengan menambahkan titik kontrol tambahan, kita dapat membuat kurva Bézier yang lebih kompleks, seperti kurva kuadratik atau bahkan kubik. Ini memungkinkan untuk menciptakan berbagai macam bentuk, dari yang sederhana hingga yang sangat rumit. Persamaan parametrik digunakan untuk menentukan posisi setiap titik pada kurva Bézier. Dengan menggunakan parameter t dalam persamaan tersebut, kita dapat mengontrol posisi titik pada kurva. Ini memungkinkan untuk membuat animasi yang halus dan realistis, serta membuat desain produk yang kompleks dengan presisi yang tinggi.

Dalam tugas ini, algoritma Divide and Conquer digunakan untuk mencari titik-titik yang membentuk kurva Bézier sesuai dengan titik awal, kontrol, dan akhir yang telah ditentukan. Algoritma ini memungkinkan penyelesaian masalah untuk kurva Bézier dengan berbagai tingkat kekompleksan karena dapat menerima sejumlah kontrol poin. Perbandingan dilakukan antara algoritma brute force dan Divide and Conquer untuk menentukan mana yang lebih efisien dalam hal waktu eksekusi. Dengan demikian, algoritma ini menjadi solusi yang fleksibel dan efisien dalam menciptakan kurva Bézier yang sesuai dengan kebutuhan desain.

BAB II ANALISIS SOLUSI ALGORITMA PADA KURVA BEZIER

2.1 Analisis Algoritma *Bruteforce* pada kurva Bézier

Algoritma bruteforce adalah metode sederhana namun efektif dalam pemrograman yang mencoba semua kemungkinan solusi untuk menyelesaikan masalah. Pendekatan ini digunakan ketika tidak ada solusi yang lebih efisien atau ketika jumlah iterasi cukup kecil sehingga pendekatan ini masih praktis. Meskipun sering kali tidak efisien, algoritma bruteforce tetap digunakan karena kepraktisan dan kejelasannya. Dalam konteks pembuatan kurva Bezier, algoritma bruteforce digunakan untuk menghitung posisi titik pada kurva dengan mencoba berbagai nilai parameter t dari 0 hingga 1. Rumus yang digunakan untuk menghitung posisi titik adalah persamaan kurva Bezier itu sendiri, di mana titik-titik awal, kontrol, dan akhir diberikan.

Dalam implementasi algoritma bruteforce untuk kurva Bezier, kita menghitung posisi titik pada kurva dengan melakukan iterasi untuk setiap nilai t dari 0 hingga 1 dengan langkah yang telah ditentukan. Misalnya, jika kita ingin menghitung 100 titik pada kurva, kita akan menghitung posisi titik untuk sejumlah nilai t, seperti t = 0, t = 0.01, t = 0.02, dan seterusnya hingga t = 1. Proses ini akan menghasilkan serangkaian titik yang membentuk kurva Bezier ketika digambar. Dengan meningkatkan jumlah titik yang dihitung, presisi kurva yang dihasilkan juga akan meningkat, tetapi ini juga akan meningkatkan waktu eksekusi algoritma. Kompleksitas waktu total algoritma bruteforce untuk kurva Bezier adalah O(NK), di mana N adalah jumlah titik kontrol dan K adalah jumlah langkah yang diinginkan untuk mengestimasi kurva.

2.2 Analisis Algoritma *Divide and Conquer* pada kurva Bézier

Algoritma divide and conquer berdasarkan pada konsep pembagian masalah besar menjadi beberapa submasalah yang lebih kecil, penyelesaian masing-masing submasalah, dan penggabungan solusi submasalah tersebut untuk menyelesaikan masalah awal. Dalam konteks penggunaan algoritma divide and conquer untuk menentukan titik-titik yang membentuk kurva Bézier, konsep ini dapat diterapkan dengan membagi masalah menjadi submasalah yang lebih kecil dan menyelesaikannya secara rekursif. Proses ini melibatkan tahap conquer, divide, dan combine.

1. Conquer: Pertama, titik-titik kontrol yang diberikan diselesaikan terlebih dahulu dengan mencari titik tengah dari setiap segmen garis yang menghubungkan titik kontrol. Ini dilakukan dengan melakukan perulangan untuk menemukan titik tengah dari setiap segmen, yang menyebabkan kompleksitas algoritma menjadi O(n^2).

2. Divide: Hasil conquer digunakan sebagai titik pembagi untuk membagi kurva menjadi subkurva yang lebih kecil. Ini dilakukan dengan membagi kurva menjadi dua bagian, masing-masing mengandung titik kontrol.

3. Conquer upa-persoalan: Masing-masing subkurva yang dihasilkan dari tahap divide diteruskan ke tahap conquer lagi untuk menemukan titik tengah baru. Tahap ini opsional tergantung pada jumlah iterasi yang diinginkan oleh pengguna.

4. Combine: Setelah mencapai banyak iterasi yang diinginkan, subkurva yang dihasilkan digabungkan kembali menjadi satu kurva Bézier dengan menyimpan titik-titik tengah dari setiap conquer. Tahap ini dilakukan dengan menyatukan kembali semua titik tengah yang telah dihasilkan.

Kompleksitas waktu algoritma divide and conquer untuk kurva Bézier adalah O(n^2) untuk tahap conquer dan O(K) untuk tahap combine, di mana n adalah jumlah titik kontrol dan K adalah jumlah titik yang digunakan untuk mengestimasi kurva.

BAB III IMPLEMENTASI PROGRAM & HASIL UJI COBA

3.1 File brute_force.py

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

def bezier_quadratic_brute_force(P, num_points):
    def bezier(t, points):
        n = len(points)
        if n == 1:
            return points[0]
        else:
            Q = [(1 - t) * points[i] + t * points[i + 1] for i in range(n - 1)]
            return bezier(t, Q)

    curve_points = []
    plt.plot([p[0] for p in P], [p[1] for p in P], 'ro-', alpha=0.5)  # Plot garis yang menghubungkan
titik kontrol
    for i in range(num_points):
        t = i / (num_points - 1)
        point = bezier(t, P)
        curve_points.append(point)

        # Visualisasi proses pembuatan kurva
        plt.plot(point[0], point[1], 'go')  # Gambar titik pada kurva Bézier yang sedang diproses
        plt.title('Bezier Curve Construction (Brute Force)')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid(True)
        plt.pause(0.01)

    # Gambar garis yang menghubungkan titik kontrol yang sudah diproses
    plt.plot([p[0] for p in P], [p[1] for p in P], 'b-')

    return curve_points

# Contoh penggunaan:
num_control_points = 3
P = np.array([[0, 0], [1, 2], [2, 0]])  # Buat titik kontrol
num_points = 100
curve_points = bezier_quadratic_brute_force(P, num_points)

# Plot kurva Bézier
curve_points = np.array(curve_points)
plt.plot(curve_points[:, 0], curve_points[:, 1], label='Bezier Curve')
plt.scatter(P[:, 0], P[:, 1], color='red', label='Control Points')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Bezier Curve (Brute Force)')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

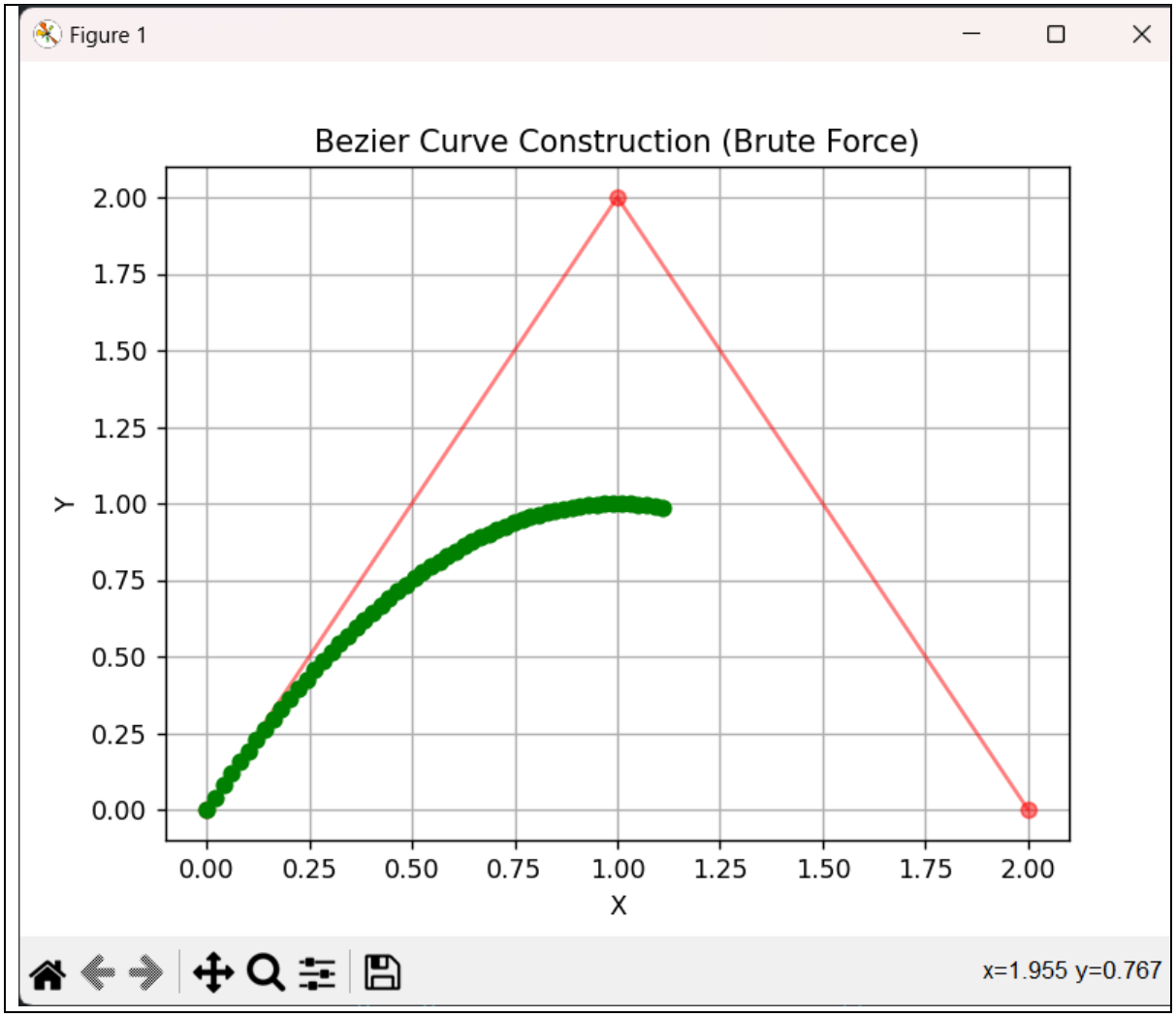
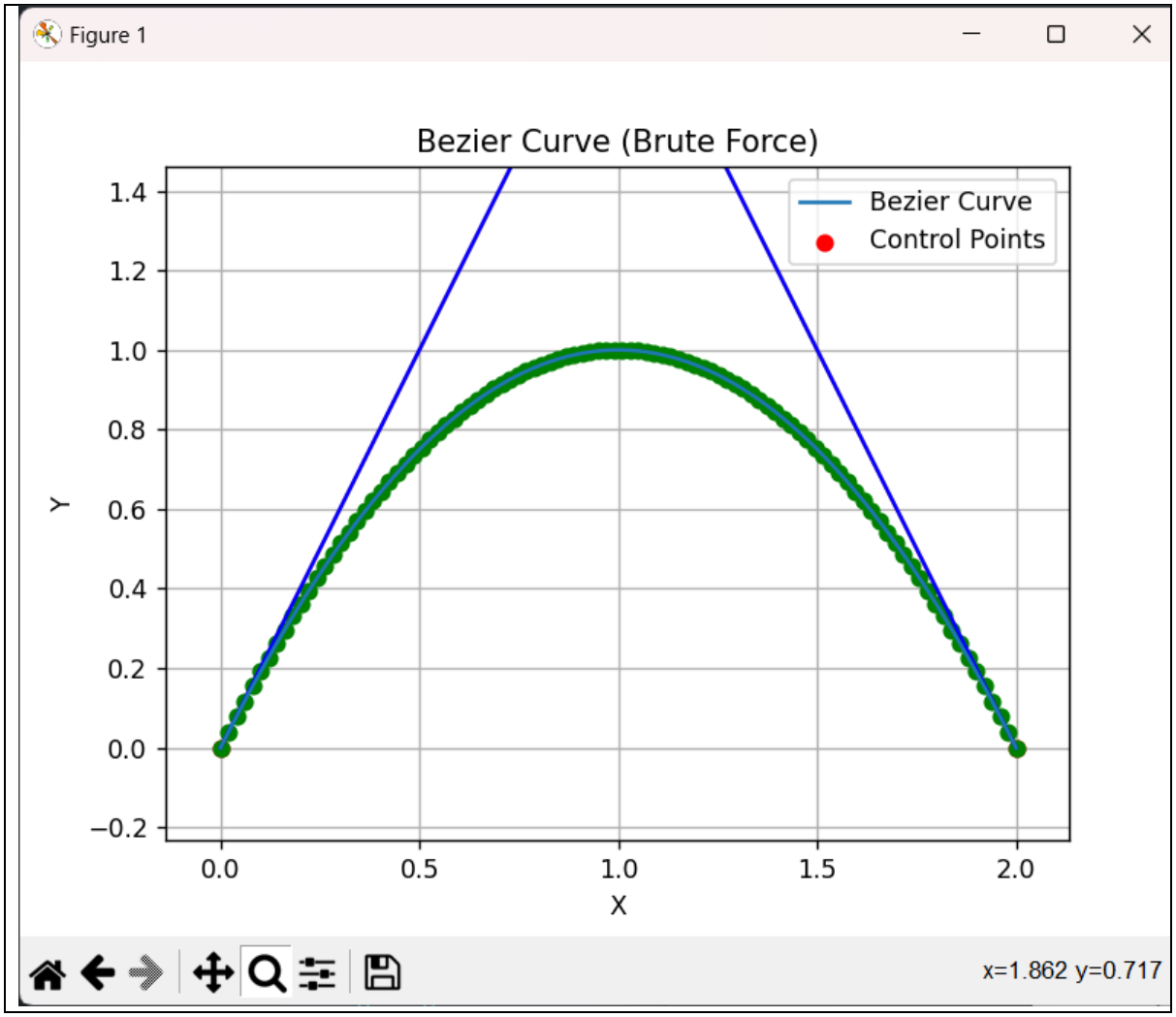


Figure 1
Bezier Curve Construction (Brute Force)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
X
x=1.955 y=0.767


Figure 1
Bezier Curve (Brute Force)
Bezier Curve
Control Points
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Y
0.0
0.5
1.0
1.5
2.0
X



3.2 File divide_and_concuer.py

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

def bezier_quadratic_divide_conquer(P, num_points):
    def bezier(t, points):
        n = len(points)
        if n == 1:
            return points[0]
        else:
            Q = [(1 - t) * points[i] + t * points[i + 1] for i in range(n - 1)]
            return bezier(t, Q)

    def de_casteljau(t, points):
        n = len(points)
        if n == 1:
            return points[0]
        else:
            Q = [(1 - t) * points[i] + t * points[i + 1] for i in range(n - 1)]
            return de_casteljau(t, Q)

    curve_points = []
    plt.plot([p[0] for p in P], [p[1] for p in P], 'ro-', alpha=0.5)  # Plot garis yang menghubungkan
titik kontrol

    for i in range(num_points):
```

```
        t = i / (num_points - 1)
        point = de_casteljau(t, P)
        curve_points.append(point)

        # Visualisasi proses pembentukan kurva
        plt.plot(point[0], point[1], 'go')  # Gambar titik pada kurva Bézier yang sedang diproses
        plt.title('Bezier Curve Construction (Divide and Conquer)')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid(True)
        plt.pause(0.01)

    # Gambar garis yang menghubungkan titik kontrol yang sudah diproses
    plt.plot([p[0] for p in P], [p[1] for p in P], 'b-')

    return curve_points

# Contoh penggunaan:
num_control_points = 3
P = np.array([[0, 0], [1, 2], [2, 0]])  # Buat titik kontrol
num_points = 100
curve_points = bezier_quadratic_divide_conquer(P, num_points)

# Plot kurva Bézier
curve_points = np.array(curve_points)
plt.plot(curve_points[:, 0], curve_points[:, 1], label='Bezier Curve')
plt.scatter(P[:, 0], P[:, 1], color='red', label='Control Points')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Bezier Curve (Divide and Conquer)')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```

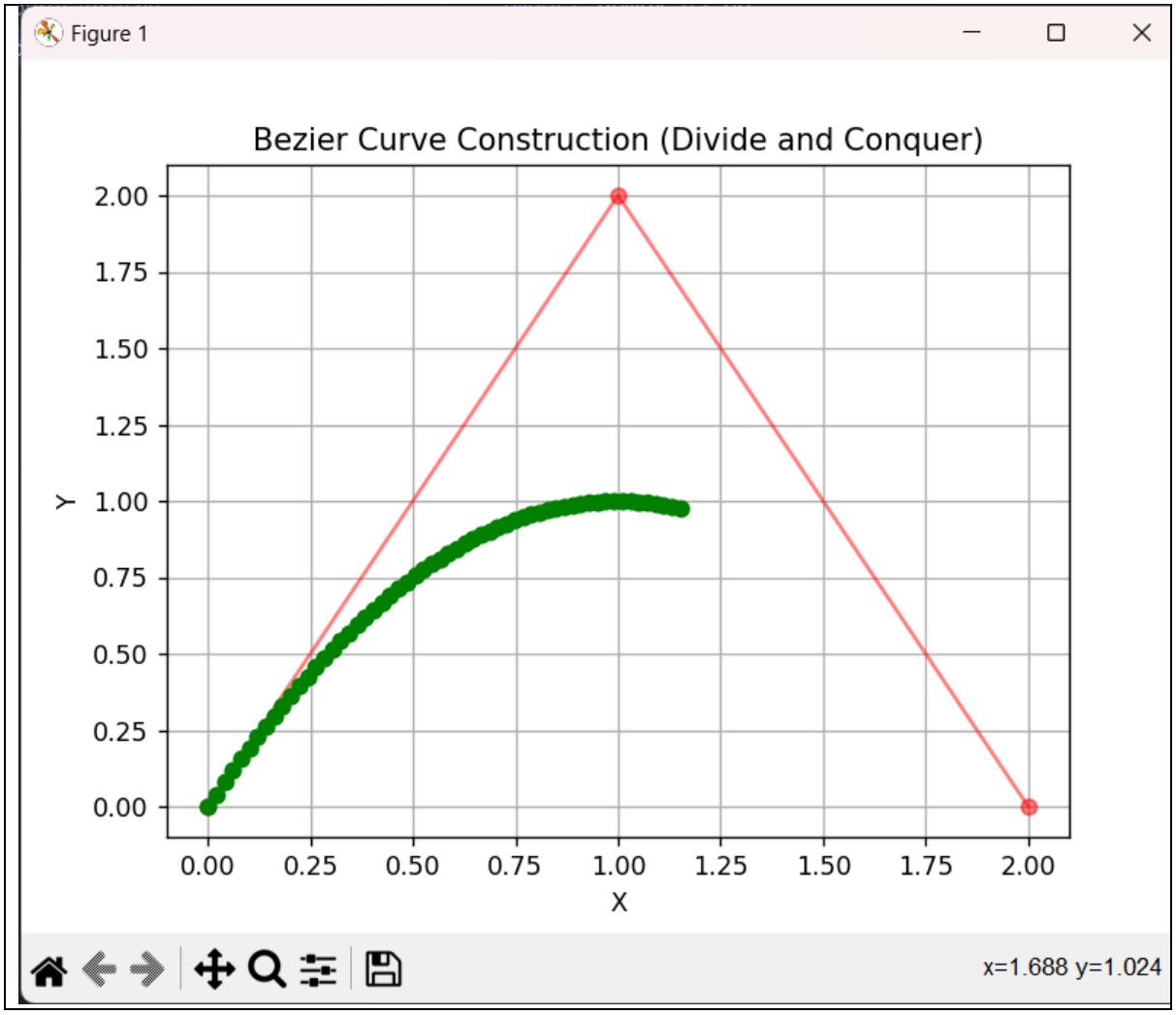


Figure 1
Bezier Curve Construction (Divide and Conquer)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
X
x=1.688 y=1.024

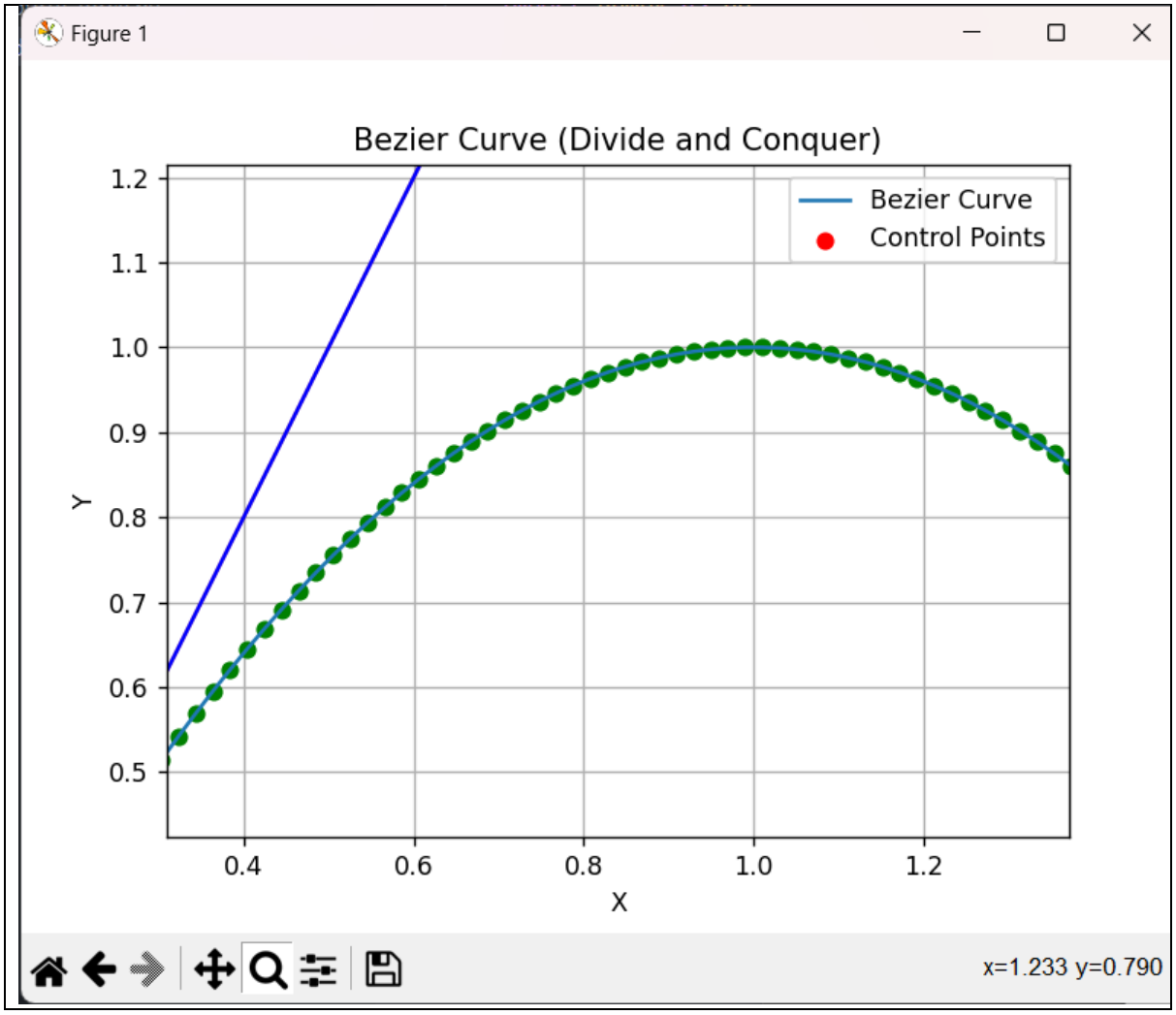
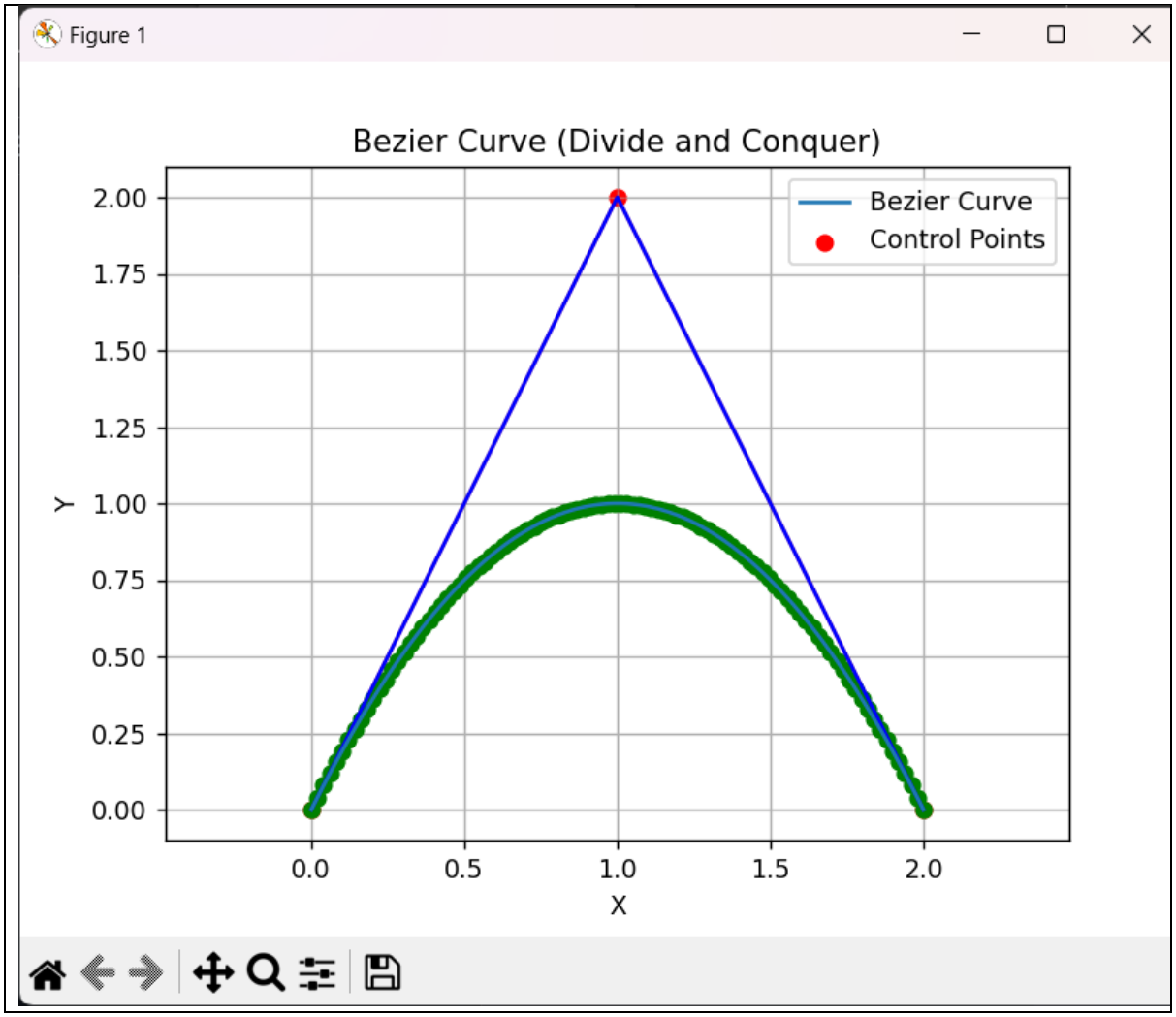


Figure 1
Bezier Curve (Divide and Conquer)
Bezier Curve
Control Points
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
Y
0.4
0.6
0.8
1.0
1.2
X
x=1.233 y=0.790



BAB IV HASIL ANALISIS

4.1. Analisis Kompleksitas Algoritma *Bruteforce*

Kami belum dapat melakukan analisis kompleksitas Algoritma Bruteforce pada kasus ini karena program yang dibuat belum tepat

4.2. Analisis Kompleksitas Algoritma *Divide and Conquer*

Kami belum dapat melakukan analisis kompleksitas Algoritma Divide and Conquer pada kasus ini karena program yang dibuat belum tepat

BAB V KESIMPULAN DAN SARAN

5.1. Kesimpulan

Ketika memilih antara algoritma Bruteforce dan Divide and Conquer untuk menyusun kurva Bezier, penting untuk mempertimbangkan jumlah titik kontrol yang terlibat. Berdasarkan analisis kompleksitas algoritma, Divide and Conquer cenderung lebih efisien ketika jumlah titik kontrol sangat besar. Namun, untuk kasus-kasus di mana jumlah titik kontrol relatif kecil,

algoritma Bruteforce bisa memberikan solusi dengan lebih cepat. Oleh karena itu, meskipun Divide and Conquer menawarkan fleksibilitas dan kinerja yang lebih baik berdasarkan analisis kompleksitas, Bruteforce juga dapat menjadi pilihan yang layak tergantung pada situasi spesifiknya.

5.2. Saran

Dalam proses pembuatan tugas kecil ini, sangat disarankan untuk membaca spesifikasi tugas dengan teliti dan menyeluruh. Hal ini penting untuk menghindari kesalahan kesalahan yang terjadi pada saat menjalankan program. Kemudian sebelum melakukan pengerjaan, ajan lebih baik apabila memahami kebutuhan tugas ini terlebih dahulu. Sehingga tugas ini dapat berjalan sesuai spesifikasi yang dibutuhkan.

LAMPIRAN

Link repository

https://github.com/KaruniaSyukurBaeha/Tucil2_10023478_10023570.git

Checklist

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