Clothoid

明治大学　総合数理学部　現象数理学科
4年2組79番　学籍番号2610200064
桂田研究室所属　沼田龍斗

2025年2月28日

目 次

第I部
Introduction

1　本論文について

図 1:

Fresnel 積分は Augustin Jean Fresnel(図 1) の名を冠した光学分野で重要な役割を果たす超越関数である。そしてそれをもとに描かれる曲線を Clothoid という。Clothoid には美しい幾何学的な性質がある。曲率という概念がある。簡単に言えば曲線上の任意の点での曲がり具合を表す量だ。大きいほど曲がりが急で逆もまた然り。Clothoid には任意の点での曲率が原点からその点までの曲線の長さに比例するという性質がある。言い換えるならば曲率が弧長パラメータに関する 1 次関数で表されている。この性質の何が嬉しいのか。Clothoid の性質は高速道路のインターチェンジの設計に応用できる。インターチェンジが描く円弧を曲線と見立て曲率が 1 次関数に従って増えれば運転手はハンドルを一定角速度で操作するだけで良くスムーズな運転が可能になる事が想像できる。そして急激な曲率変化によって引き起こされる運転手の不快感や事故の危険性が軽減され道路利用者の安全性が向上する。実際に道路設計では頻繁に Clothoid が直線部分と円弧部分を滑らかに接続する緩和曲線として採用されている。また道路設計の他にも様々な分野で Clothoid の性質が応用されている。
本論文では Fresnel 積分を基礎として Clothoid を描画する手法に焦点を当てる。Maclaurin 展開、Python 数値計算ライブラリ、常微分方程式の初期値問題、誤差

関数、Frenet-Serret 枠を用いて描画することを考え、そして計算時間量の比較をする。
また本論文の留意事項は以下の通りである。

1. TEX の不調でコードの説明を行うコメント文が日本語を用いて適切に表示することができなかった。故に英語での表記となっている。
2. 図の比較がしやすいように描画様式の設定は統一している。
3. t に関する微分は「˙」で表すがそれ以外は「′」で表す。
4. 出力に対する考察は Conclusion にまとめている。
5. 定義は

test

で囲んでいる。

6. 定理は

test

Proof. test □

で囲み直後の証明とひとまとまりになっている。

2 Fresnel積分

定義 2.1.

$$C(x) = \int_0^x \cos t^2 dt, \quad S(x) = \int_0^x \sin t^2 dt \quad (x \in \mathbb{R})$$

をそれぞれ *Fresnel* 積分と呼ぶ。

定理 2.1. $x \to \infty$ のとき

$$C(x) \to \frac{\sqrt{2\pi}}{4}, \quad S(x) \to \frac{\sqrt{2\pi}}{4}$$

となる。

Proof. 複素積分を利用して示したい。

$$z = x + iy \quad (x, y \in \mathbb{R})$$

とし複素関数

$$f(z) = e^{iz^2} \tag{2.1}$$

を考える。(2.1) は正則である。積分経路を図 2 のように

$$C = C_1 + C_2 + C_3$$

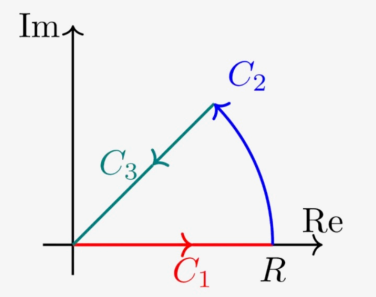

図 2:

$$C_1 = C_{(1,R)} : z = x \quad (0 < x < R)$$
$$C_2 = C_{(2,\theta)} : z = Re^{i\theta} \quad (0 < \theta < \frac{\pi}{4})$$
$$C_3 = C_{(3,R)} : z = xe^{i\frac{\pi}{4}} \quad (R > x > 0)$$

とし (2.1) の線積分を行う。

$$\oint_C f(z)dz = \int_{C_1} e^{iz^2}dz + \int_{C_2} e^{iz^2}dz + \int_{C_3} e^{iz^2}dz \tag{2.2}$$

Cauchy の積分定理より (2.1) が正則のとき (2.2) の左辺が 0 になるので右辺のみを考えれば良い。まずは C_1 上について考える。

$$\begin{aligned}\int_{C_1} e^{iz^2}dz &= \int_0^R \cos x^2 dx + i\int_0^R \sin x^2 dx \\ &= \int_0^\infty \cos x^2 dx + i\int_0^\infty \sin x^2 dx \quad (R\to\infty)\end{aligned} \tag{2.3}$$

次に C_2 上について考える。両辺を θ で微分して変形すると

$$dz = Rie^{i\theta}d\theta$$

よって

$$\begin{aligned}\left|\int_{C_2} e^{iz^2}dz\right| &= \left|\int_0^{\frac{\pi}{4}} e^{iR^2e^{i2\theta}}Rie^{i\theta}d\theta\right| \\ &\leqq \int_0^{\frac{\pi}{4}} |e^{iR^2(\cos 2\theta + i\sin 2\theta)}Rie^{i\theta}|d\theta \\ &= \int_0^{\frac{\pi}{4}} |e^{iR^2\cos 2\theta}||e^{-R^2\sin 2\theta}||Ri||e^{i\theta}|d\theta \\ &= \int_0^{\frac{\pi}{4}} Re^{-R^2\sin 2\theta}d\theta \\ &= 0 \quad (R\to\infty)\end{aligned} \tag{2.4}$$

次に C_3 上について考える。両辺を x で微分して変形すると

$$dz = \frac{1+i}{\sqrt{2}}dx$$

Gauss 積分の公式

$$\int_0^\infty e^{-x^2}dx = \frac{\sqrt{\pi}}{2}$$

を用いて

$$\begin{aligned}\int_{C_3} e^{iz^2}dz &= \int_R^0 e^{x^2 i\frac{\pi}{2}}\frac{1+i}{\sqrt{2}}dx \\ &= \frac{1+i}{\sqrt{2}}\int_R^0 e^{-x^2}dx \\ &= -\frac{\sqrt{2\pi}}{4} - i\frac{\sqrt{2\pi}}{4} \quad (R\to\infty)\end{aligned} \tag{2.5}$$

となる。(2.2)、(2.3)、(2.4)、(2.5) を用いて整理すると

$$\int_0^\infty \cos x^2 dx + i\int_0^\infty \sin x^2 dx = \frac{\sqrt{2\pi}}{4} + i\frac{\sqrt{2\pi}}{4}$$

となり。実部と虚部を比較し

$$C(x) \to \frac{\sqrt{2\pi}}{4}, \quad S(x) \to \frac{\sqrt{2\pi}}{4} \quad (x \to \infty)$$

となる。 □

定理 2.2.

$$\frac{\sqrt{2\pi}}{4} \approx 0.6267$$

Proof. 省略。 □

3 Clothoid

定義 3.1.

$$(C(t), S(t)) = (\int_0^t \cos t^2 dt, \int_0^t \sin t^2 dt) \quad (t \in \mathbb{R} > 0)$$

が描く曲線を $Clothoid$ と呼ぶ。

第II部
Methods

4 Maclaurin展開を用いたClothoidの描画

定理 4.1. すべての $z \in \mathbb{R}$ に対して次式が成り立つ

$$C(t) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(4k+1)(2k)!} t^{4k+1}, \quad S(t) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(4k+3)(2k+1)!} t^{4k+3}$$

Proof. $\sin t^2$、$\cos t^2$ が複素平面全体で正則であるので複素数平面全体で冪級数展開できて冪級数は収束円の中で項別積分できる。 □

定理 4.1 で示した Maclaurin 展開を用いて Clothoid を描画したい。以下に入力である Python のコード (Listing1) と出力 (図 3) を示す。なおこの手法はコンピュータにかける負担が大きい。故に実行環境においてパラメータの区間を $[0, 10]$ でしか取れず後に紹介する手法との計算時間量の比較が出来なかった。なので計算時間量の測定は省略している。

Listing 1: Clothoid by power series display

```
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
from math import factorial  # Using Python's built-in factorial function

# Number of terms for the Maclaurin series expansion
num_terms = 100

# Function definitions
def C_approx(t, terms=num_terms):
    """Maclaurin series approximation of C(t)"""
    result = 0
    for n in range(terms):
        # Compute each term of the series
        term = ((-1)**n / factorial(2 * n)) * (t**(4 * n + 1)) / (4 * n + 1)
        result += term
    return result

def S_approx(t, terms=num_terms):
    """Maclaurin series approximation of S(t)"""
    result = 0
    for n in range(terms):
        # Compute each term of the series
        term = ((-1)**n / factorial(2 * n + 1)) * (t**(4 * n + 3)) / (4 * n + 3)
        result += term
    return result

# Generate data for plotting
t_values = np.linspace(0, 10, 1000)  # Adjustable range of t
C_values = [C_approx(t) for t in t_values]  # Compute C(t) values
S_values = [S_approx(t) for t in t_values]  # Compute S(t) values

# Plot
plt.figure(figsize=(10, 10))
plt.plot(C_values, S_values, label="Clothoid curve by power series display", color
  ="blue")
plt.axhline(0, color="black", linewidth=1.0, linestyle="--")
plt.axvline(0, color="black", linewidth=1.0, linestyle="--")
plt.title("Parametric Plot of $(C(t), S(t))$", fontsize=10)
plt.xlabel("$C(t)$", fontsize=10)
plt.ylabel("$S(t)$", fontsize=10)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.axis("equal")
plt.show()
```

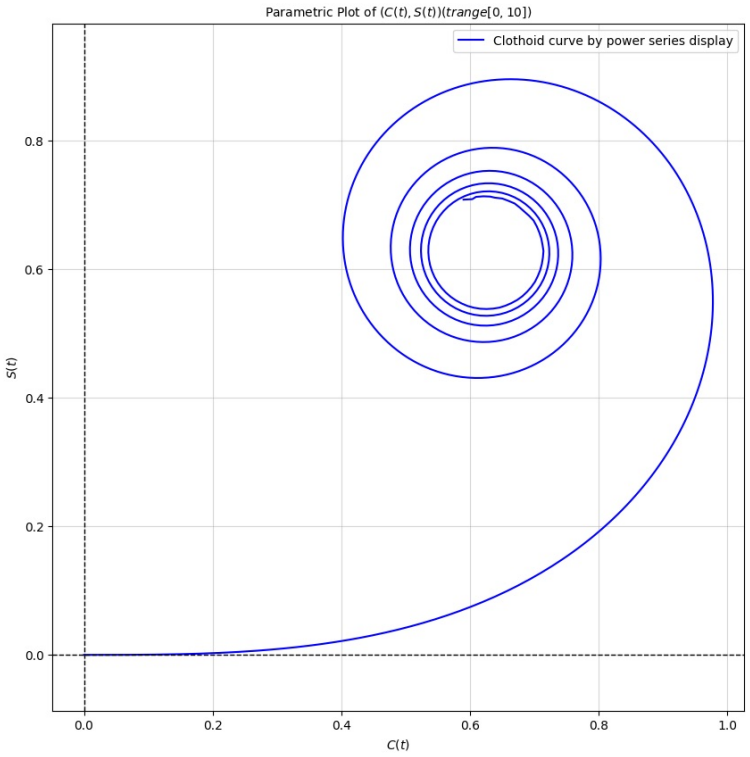
Parametric Plot of $(C(t), S(t))(trange[0, 10])$
Clothoid curve by power series display
0.8
0.6
0.4
0.2
0.0
$S(t)$
0.0
0.2
0.4
0.6
0.8
1.0
$C(t)$

図 3:

5 Python数値計算ライブラリを用いたClothoidの描画

定義 5.1.

$$\tilde{C}(x) = \int_0^x \cos\frac{\pi}{2}t^2 dt, \quad \tilde{S}(x) = \int_0^x \sin\frac{\pi}{2}t^2 dt \quad (x \in \mathbb{R})$$

をそれぞれ正規化された *Fresnel* 積分と呼ぶ。

$$C(x) = \sqrt{\frac{\pi}{2}}\tilde{C}\left(\sqrt{\frac{2}{\pi}}x\right), \quad S(x) = \sqrt{\frac{\pi}{2}}\tilde{S}\left(\sqrt{\frac{2}{\pi}}x\right)$$

という関係がある。

定理 5.1. $x \to \infty$ のとき

$$\tilde{C}(x) \to \frac{1}{2}, \quad \tilde{S}(x) \to \frac{1}{2}$$

となる。

Proof. 省略。 □

Python数値計算ライブラリ内の関数であるscipy.special.fresnelとは正規化されたFresnel積分を計算するためにあらかじめ用意されている関数である。scipy.special.fresnelを用いてClothoidを描画したい。しかし関数内のFresnel積分は設計上で正規化されているのでClothoidを描くよう調整するための関数myfresnelを定義した。以下に入力であるPythonのコード(Listing2)と出力(図4)を示す。なおパラメータの区間は$[0, 100]$、区間幅(曲線を描くための変数の刻み幅)は0.01である。

Listing 2: Clothoid by scipy.special.fresnel

```
1  import numpy as np  # Library for numerical computations
2  import matplotlib.pyplot as plt  # Library for plotting
3  import scipy.special as sc
4
5  def myfresnel(x):
6      factor = np.sqrt(np.pi / 2)
7      [s, c] = sc.fresnel(x / factor)
8      return [factor * s, factor * c]
9
10 def plot_clothoid(t_max, num_points):
11     """
```

```
    Function to plot a Clothoid (Cornu spiral)

    Parameters:
    t_max: float
        Maximum arc length (maximum value of t)
    num_points: int
        Number of divisions for the arc length (number of points for t)
    """
    # Generate values for the arc length t
    t = np.linspace(0, t_max, num_points)

    # Use myfresnel to get the Clothoid curve
    S, C = zip(*[myfresnel(ti) for ti in t])

    # Plotting
    plt.figure(figsize=(10, 10))
    plt.plot(C, S, label="Clothoid curve by myfresnel", color="blue")
    plt.axhline(0, color="black", linewidth=1.0, linestyle="--")
    plt.axvline(0, color="black", linewidth=1.0, linestyle="--")
    plt.title("Parametric Plot of $(C(t), S(t))$", fontsize=10)
    plt.xlabel("$C(t)$", fontsize=10)
    plt.ylabel("$S(t)$", fontsize=10)
    plt.legend(fontsize=10)
    plt.grid(alpha=0.5)
    plt.axis("equal")
    plt.show()

# Execute the function
plot_clothoid(t_max=100, num_points=10000)
```

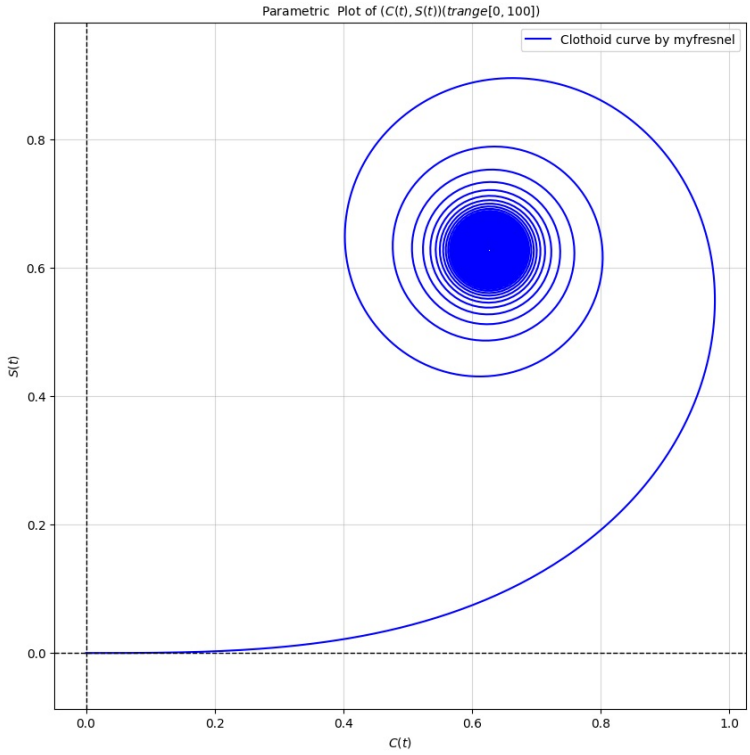

図 4:

Clothoid を描画する際に費やす計算時間量を求めたい。1000 回実行したときの 1 回あたりの算術平均時間を計算時間量とする。以下に入力である Python のコード (Listing3) と出力 (図 5) を示す。

Listing 3: Computation Time per Run

```
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
import scipy.special as sc
import time

# Define the Fresnel function
def myfresnel(x):
    factor = np.sqrt(np.pi / 2)
    s, c = sc.fresnel(x / factor)
    return factor * s, factor * c

# Define the function to plot the Clothoid
def plot_clothoid(t_max, num_points):

    # Generate values for the arc length t
    t = np.linspace(0, t_max, num_points)

    # Use myfresnel to get the Clothoid curve
    S, C = zip(*[myfresnel(ti) for ti in t])
```

```python

# Initialize list to store computation times
computation_times = []

# Run the function 100 times and record computation times
for _ in range(1000):
    start_time = time.time()
    plot_clothoid(t_max=100, num_points=10000)
    end_time = time.time()
    computation_times.append(end_time - start_time)

# Compute the average computation time
average_time = np.mean(computation_times)

# Plot the computation times as points
plt.figure(figsize=(10, 5))
plt.plot(computation_times, 'o', label="Computation Time per Run", color='blue')
  # Use 'o' for points
plt.axhline(y=average_time, color='red', linestyle='--', label=f"Average Time = {
  average_time:.4f}s")
plt.xlabel('Run Number')
plt.ylabel('Computation Time (seconds)')
plt.title('Computation Time for 1000 Runs')
plt.legend()
plt.grid(True)
plt.show()

# Print the average computation time
print(f"Average computation time: {average_time:.4f} seconds")
```

図 5:

6　常微分方程式の初期値問題を用いたClothoidの描画

定義 6.1. *Fresnel*積分 C、S は次の初期値問題の解である。

$$\begin{cases} \dot{C}(t) = \cos t^2 \\ C(0) = 0 \end{cases}$$

$$\begin{cases} \dot{S}(t) = \sin t^2 \\ S(0) = 0 \end{cases}$$

$C(t)$、$S(t)$ をそれぞれ両辺 t で微分すると

$$\dot{C}(t) = \cos t^2, \quad \dot{S}(t) = \sin t^2$$

微分方程式とは未知関数とその導関数の関係式として書かれている関数方程式である。分かりやすく書き換えると

$$C(t) = \int_0^t \dot{C}(t)dt, \quad S(t) = \int_0^t \dot{S}(t)dt$$

となり $C(t)$、$S(t)$ ともに t に関する常微分方程式であることが分かった。

$$C(0) = 0, \quad S(0) = 0$$

を与えれば常微分方程式の初期値問題となる。定義 6.1 で示した常微分方程式の初期値問題をそれぞれ Runge-Kutta 法で解き Clothoid を描画したい。以下に入力である Python のコード (Listing4) と出力 (図 6) を示す。なおパラメータの区間は $[0, 100]$、区間幅 (曲線を描くための変数の刻み幅) は 0.01 である。

Listing 4: Clothoid by separate ODE solutions

```
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
from scipy.integrate import solve_ivp  # Function to solve initial value problems
  for ODEs
import time

# Define the right-hand side of the ODE for c(t)
def dc_dt(t, c):
    return np.cos(t**2)

# Define the right-hand side of the ODE for s(t)
def ds_dt(t, s):
    return np.sin(t**2)

# Function to solve ODEs and generate the clothoid plot
def solve_and_plot():
```

```
    # Initial conditions and time range
    c0 = 0  # Initial condition for c(0)
    s0 = 0  # Initial condition for s(0)
    t_span = (0, 100)  # Time range: t ∈ [0, 100]
    t_eval = np.linspace(0, 100, 10000)  # Time points for evaluation (for
      plotting)

    # Solve the ODE for c(t) using solve_ivp
    c_solution = solve_ivp(dc_dt, t_span, [c0], method='RK45', t_eval=t_eval, rtol
      =1e-6, atol=1e-8)
    C_values = c_solution.y[0]  # Extract the values of c(t)

    # Solve the ODE for s(t) using solve_ivp
    s_solution = solve_ivp(ds_dt, t_span, [s0], method='RK45', t_eval=t_eval, rtol
      =1e-6, atol=1e-8)
    S_values = s_solution.y[0]  # Extract the values of s(t)

    # Plot
    plt.figure(figsize=(10, 10))
    plt.plot(C_values, S_values, label="Clothoid curve by separate ODE solutions",
       color="blue")
    plt.axhline(0, color="black", linewidth=1.0, linestyle="--")
    plt.axvline(0, color="black", linewidth=1.0, linestyle="--")
    plt.title("Parametric Plot of $(C(t), S(t))$", fontsize=10)
    plt.xlabel("$C(t)$", fontsize=10)
    plt.ylabel("$S(t)$", fontsize=10)
    plt.legend(fontsize=10)
    plt.grid(alpha=0.5)
    plt.axis("equal")
    plt.show()

# Call the function to execute the computation and plot
solve_and_plot()
```

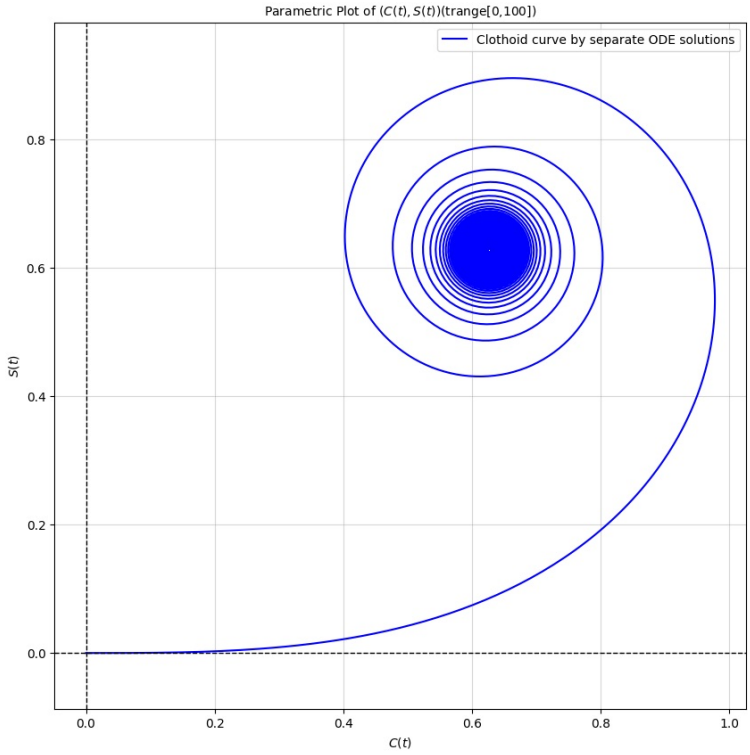

図 6:

Clothoid を描画する際に費やす計算時間量を求めたい。1000 回実行したときの 1 回あたりの算術平均時間を計算時間量とする。以下に入力である Python のコード (Listing5) と出力 (図 7) を示す。

Listing 5: Computation Time per Run

```
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
from scipy.integrate import solve_ivp  # Function to solve initial value problems for ODEs
import time

# Define the right-hand side of the ODE for c(t)
def dc_dt(t, c):
    return np.cos(t**2)

# Define the right-hand side of the ODE for s(t)
def ds_dt(t, s):
    return np.sin(t**2)

# Function to solve ODEs and generate the clothoid plot
def solve_and_plot():
    # Initial conditions and time range
    c0 = 0  # Initial condition for c(0)
    s0 = 0  # Initial condition for s(0)
```

```
    t_span = (0, 100)  # Time range: t ∈ [0, 100]
    t_eval = np.linspace(0, 100, 10000)  # Time points for evaluation (for plotting)

    # Solve the ODE for c(t) using solve_ivp
    c_solution = solve_ivp(dc_dt, t_span, [c0], method='RK45', t_eval=t_eval, rtol=1e-6, atol=1e-8)
    C_values = c_solution.y[0]  # Extract the values of c(t)

    # Solve the ODE for s(t) using solve_ivp
    s_solution = solve_ivp(ds_dt, t_span, [s0], method='RK45', t_eval=t_eval, rtol=1e-6, atol=1e-8)
    S_values = s_solution.y[0]  # Extract the values of s(t)

# Initialize list to store computation times
computation_times = []

# Run the function 1000 times and record computation times
for _ in range(1000):
    start_time = time.time()
    solve_and_plot()
    end_time = time.time()

    computation_times.append(end_time - start_time)

# Compute the average computation time
average_time = np.mean(computation_times)

# Plot the computation times as points
plt.figure(figsize=(10, 5))
plt.plot(computation_times, 'o', label="Computation Time per Run", color='blue')  # Use 'o' for points
plt.axhline(y=average_time, color='red', linestyle='--', label=f"Average Time = {average_time:.4f}s")
plt.xlabel('Run Number')
plt.ylabel('Computation Time (seconds)')
plt.title('Computation Time for 1000 Runs')
plt.legend()
plt.grid(True)
plt.show()

# Print the average computation time
print(f"Average computation time: {average_time:.4f} seconds")
```

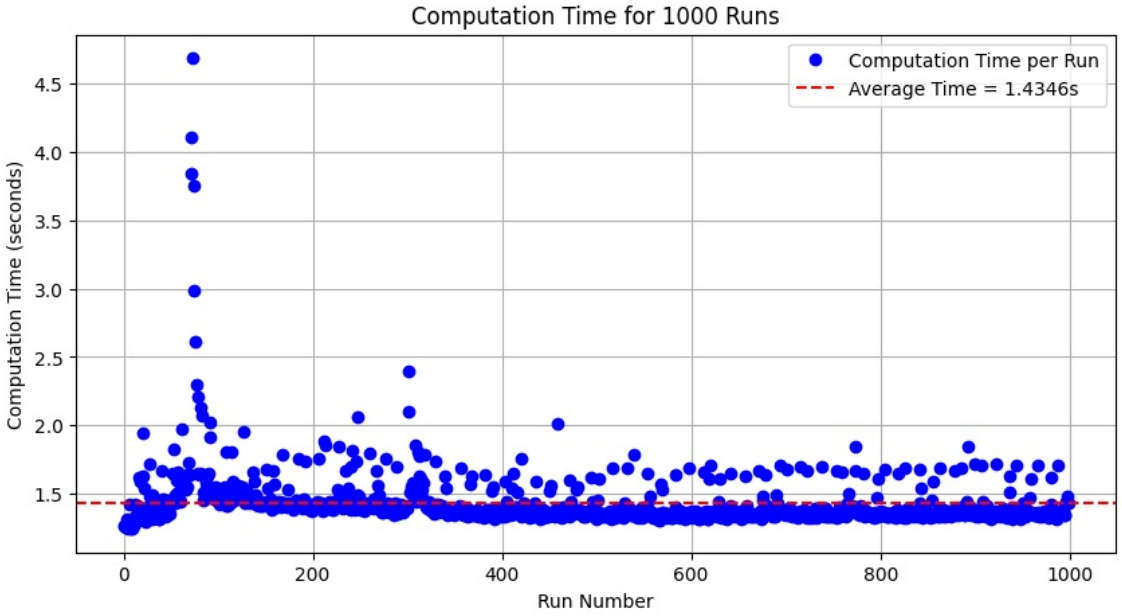

図 7:

7　誤差関数を用いた Clothoid の描画

定義 7.1.

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \quad (z \in \mathbb{C})$$

を誤差関数と呼ぶ。

定理 7.1. すべての $z \in \mathbb{C}$ に対して次式が成り立つ

$$C(z) = \frac{\pi}{4} \left[\sqrt{-i} \operatorname{erf}(\sqrt{i}\, z) + \sqrt{i} \operatorname{erf}(\sqrt{-i}\, z) \right] \quad (z \in \mathbb{C})$$

$$S(z) = \frac{\pi}{4} \left[\sqrt{i} \operatorname{erf}(\sqrt{i}\, z) + \sqrt{-i} \operatorname{erf}(\sqrt{-i}\, z) \right] \quad (z \in \mathbb{C})$$

ただし

$$\sqrt{i} = \frac{1+i}{\sqrt{2}}, \quad \sqrt{-i} = \frac{1-i}{\sqrt{2}}$$

とする。

Proof. e^{-t^2}、$\sin t^2$、$\cos t^2$ は整関数（$\mathbb{C}$ 全体で正則な関数という意味）であり $\mathbb{C}$ は単連結（あるいは星型と言っても良い）であるから積分は始点と端点のみで定まることに注意すると erf は次のように冪級数展開できる。

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{n!(2n+1)} \quad (z \in \mathbb{C}).$$

項別積分により

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \sum_{n=0}^{\infty} \frac{(-t^2)^n}{n!}\, dt = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)n!} \int_0^z t^{2n+1}\, dt = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{(2n+1)n!}$$

C、S についても同様にして次の冪級数展開を得る。

$$C(z) = \sum_{n=0}^{\infty} \frac{(-1)^n z^{4n+1}}{(4n+1)(2n)!}, \quad S(z) = \sum_{n=0}^{\infty} \frac{(-1)^n z^{4n+3}}{(4n+3)(2n+1)!}$$

を得る。

erf、C、S は整関数であるからこれらの冪級数の収束半径は $+\infty$ である。また

$$\sqrt{i}\,\mathrm{erf}(\sqrt{i}z) = \sqrt{i}\frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n (\sqrt{i})^{2n+1} z^{2n+1}}{n!(2n+1)} = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n i^{n+1} z^{2n+1}}{n!(2n+1)}$$

$$\sqrt{-i}\,\mathrm{erf}(\sqrt{-i}z) = \sqrt{-i}\frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n (\sqrt{-i})^{2n+1} z^{2n+1}}{n!(2n+1)} = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n i^{n+1} z^{2n+1}}{n!(2n+1)}$$

であり

$$i^{n+1} + (-i)^{n+1} = (1 + (-1)^{n+1}) i^{n+1} = \begin{cases} 0 & (n \text{ は偶数}) \\ 2 \cdot i^{(2k+1)+1} = 2(-1)^{k+1} & (n = 2k+1,\ k \in \mathbb{Z}) \end{cases}$$

であるから

$$\begin{aligned} \frac{\sqrt{\pi}}{4} \left[\sqrt{i}\,\mathrm{erf}(\sqrt{i}z) + \sqrt{-i}\,\mathrm{erf}(\sqrt{-i}z) \right] &= \frac{1}{2} \sum_{k=0}^{\infty} \frac{[-1 \cdot 2(-1)^{k+1}] z^{2(2k+1)+1}}{(2k+1)!(2(2k+1)+1)} \\ &= \sum_{k=0}^{\infty} \frac{(-1)^k z^{4k+3}}{(4k+3)(2k+1)!} = S(z) \end{aligned}$$

C についても同様に成り立つ。 □

定理 7.2. すべての $z \in \mathbb{R}$ に対して $C(z)$、$S(z)$ の右辺は実数である。

Proof.

$$\sqrt{i} = \pm\frac{1}{\sqrt{2}}(1+i), \quad \sqrt{-i} = \pm\frac{1}{\sqrt{2}}(1-i)$$

よって

$$\overline{\sqrt{i}} = \sqrt{-i}$$

である (複合同順)。また $z \in \mathbb{C}$ のとき

$$\overline{\operatorname{erf}(z)} = \operatorname{erf}(\overline{z})$$

が成り立つ。なので

$$\overline{\operatorname{erf}\left(\sqrt{i}\,t\right)} = \operatorname{erf}\left(\overline{\sqrt{i}\,t}\right) = \operatorname{erf}\left(\sqrt{-i}\,t\right)$$

となる。よって C は

$$C(t) = \frac{\pi}{4}\left(\sqrt{-i}\operatorname{erf}\left(\sqrt{i}\,t\right) + \sqrt{i}\,\overline{\operatorname{erf}\left(\sqrt{i}\,t\right)}\right)$$

と書き換えられる。

$$\operatorname{erf}(\sqrt{i}\,t) = a(t) + ib(t) \quad (a(t), b(t) \in \mathbb{R})$$

とすると

$$C(t) = \frac{\pi}{4}[\pm\frac{1}{\sqrt{2}}(1-i)(a(t)+ib(t)) \pm \frac{1}{\sqrt{2}}(1+i)(a(t)-ib(t))] = \pm\frac{\pi\sqrt{2}}{4}(a(t)+b(t))$$

となり C の右辺の虚部が 0 であることが示された。同様に S に関しても成り立つ。 □

定理7.1の$C(z)$、$S(z)$を用いてClothoidを描画したい。以下に入力であるPythonのコード (Listing6) と出力 (図 8) を示す。なおパラメータの区間は $[0, 100]$、区間幅 (曲線を描くための変数の刻み幅) は 0.01 である。

Listing 6: Clothoid by complex number display

```
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
from scipy.special import erf  # Error function for Fresnel integrals
```

```
4
5  # Define constants
6  sqrt_pi = np.sqrt(np.pi)  # Square root of pi
7  sqrt_i = np.sqrt(1j)  # Square root of imaginary unit i
8  sqrt_neg_i = np.sqrt(-1j)  # Square root of -i
9
10 # Compute Fresnel integral C(z)
11 def fresnel_C(z):
12     """
13     Compute the Fresnel cosine integral C(z) using complex representations.
14
15     Parameters:
16     z: ndarray
17         Input values (can be complex or real)
18
19     Returns:
20     C(z): ndarray
21         Fresnel cosine integral values
22     """
23     return (sqrt_pi / 4) * (sqrt_neg_i * erf(sqrt_i * z) + sqrt_i * erf(sqrt_neg_i
       * z))
24
25 # Compute Fresnel integral S(z)
26 def fresnel_S(z):
27     """
28     Compute the Fresnel sine integral S(z) using complex representations.
29
30     Parameters:
31     z: ndarray
32         Input values (can be complex or real)
33
34     Returns:
35     S(z): ndarray
36         Fresnel sine integral values
37     """
38     return (sqrt_pi / 4) * (sqrt_i * erf(sqrt_i * z) + sqrt_neg_i * erf(sqrt_neg_i
       * z))
39
40 # Generate the clothoid curve
41 def generate_clothoid(t_vals):
42     """
43     Generate x and y values for the clothoid curve.
44
45     Parameters:
46     t_vals: ndarray
47         Parameter values for the curve
48
49     Returns:
50     x_vals, y_vals: tuple of ndarrays
51         Real parts of C(t) and S(t), representing the clothoid curve
52     """
53     C_vals = fresnel_C(t_vals)
54     S_vals = fresnel_S(t_vals)
55     return C_vals.real, S_vals.real
56
57 # Define the range of parameter t
58 t_vals = np.linspace(0, 100, 10000)  # Parameter t (larger values expand the curve
     )
59
60 # Generate the clothoid curve
61 x_vals, y_vals = generate_clothoid(t_vals)
62
```

```
# Plot
plt.figure(figsize=(10, 10))
plt.plot(x_vals, y_vals, label="Clothoid curve by complex number display", color="
  blue")
plt.axhline(0, color="black", linewidth=1.0, linestyle="--")
plt.axvline(0, color="black", linewidth=1.0, linestyle="--")
plt.title("Parametric Plot of $(C(t), S(t))$", fontsize=10)
plt.xlabel("$C(t)$", fontsize=10)
plt.ylabel("$S(t)$", fontsize=10)
plt.legend(fontsize=10)
plt.grid(alpha=0.5)
plt.axis("equal")
plt.show()
```

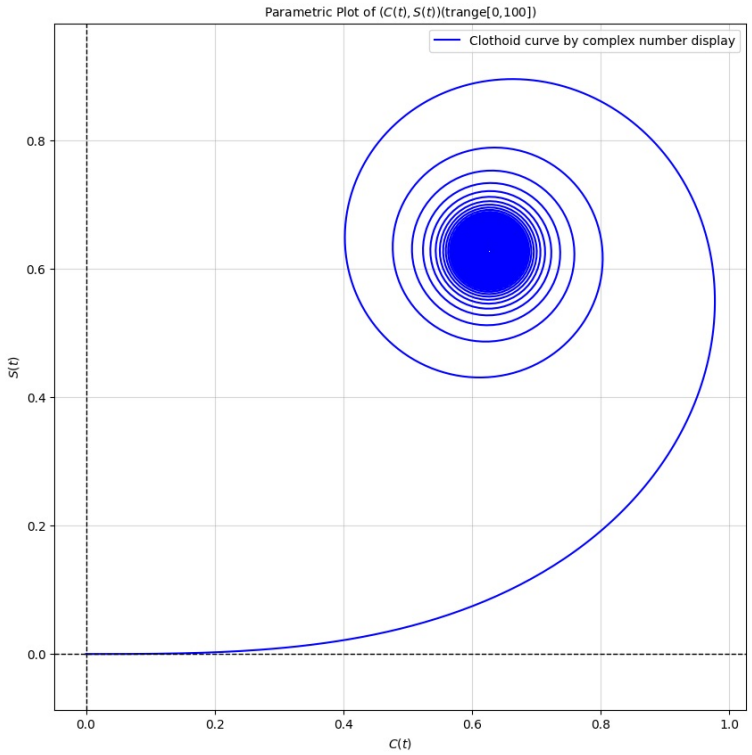

図 8:

Clothoid を描画する際に費やす計算時間量を求めたい。1000 回実行したときの 1 回あたりの算術平均時間を計算時間量とする。以下に入力である Python のコード (Listing7) と出力 (図 9) を示す。

Listing 7: Computation Time per Run

```
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
```

```
from scipy.special import erf  # Error function for Fresnel integrals
import time

# Define constants
sqrt_pi = np.sqrt(np.pi)  # Square root of pi
sqrt_i = np.sqrt(1j)  # Square root of imaginary unit i
sqrt_neg_i = np.sqrt(-1j)  # Square root of -i

# Compute Fresnel integral C(z)
def fresnel_C(z):

    return (sqrt_pi / 4) * (sqrt_neg_i * erf(sqrt_i * z) + sqrt_i * erf(sqrt_neg_i
       * z))

# Compute Fresnel integral S(z)
def fresnel_S(z):

    return (sqrt_pi / 4) * (sqrt_i * erf(sqrt_i * z) + sqrt_neg_i * erf(sqrt_neg_i
       * z))

# Generate the clothoid curve
def generate_clothoid(t_vals):

    C_vals = fresnel_C(t_vals)
    S_vals = fresnel_S(t_vals)
    return C_vals.real, S_vals.real

# Parameter t (larger values expand the curve)
np.linspace(0, 100, 10000)

# Initialize list to store computation times
computation_times = []

# Run the function 1000 times and record computation times
for _ in range(1000):
    start_time = time.time()
    generate_clothoid(t_vals)
    end_time = time.time()

    computation_times.append(end_time - start_time)

# Compute the average computation time
average_time = np.mean(computation_times)

# Plot the computation times as points
plt.figure(figsize=(10, 5))
plt.plot(computation_times, 'o', label="Computation Time per Run", color='blue')
  # Use 'o' for points
plt.axhline(y=average_time, color='red', linestyle='--', label=f"Average Time = {
  average_time:.4f}s")
plt.xlabel('Run Number')
plt.ylabel('Computation Time (seconds)')
plt.title('Computation Time for 1000 Runs')
plt.legend()
plt.grid(True)
plt.show()

# Print the average computation time
print(f"Average computation time: {average_time:.4f} seconds")
```

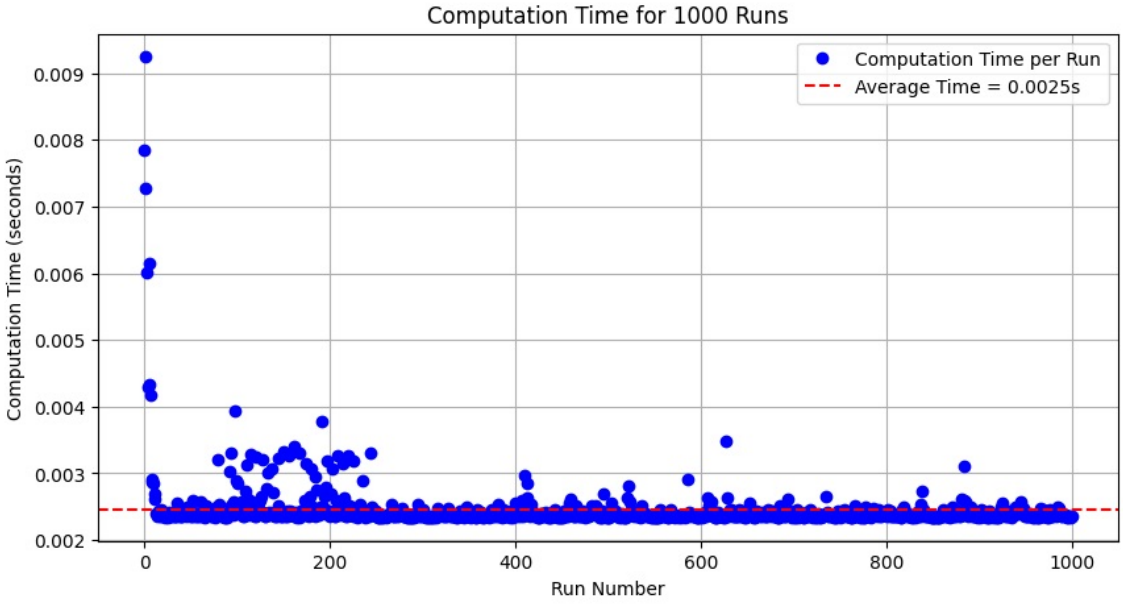

図 9:

8 Frenet-Serret 枠を用いた Clothoid の描画

定義 8.1. *Euclid* 空間 $\mathbb{E}^2$ *(*xy 平面上*)* の質点の軌跡

$$p(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} : [a, b] \to \mathbb{E}^2 \quad (t \in \mathbb{R})$$

がパラメータ t で描く平面曲線を列ベクトルで表す。このとき $p(t)$ を平面曲線と呼ぶ。$p(t)$ は C^2 級で

$$\dot{p}(t) = \frac{dp(t)}{dt} \neq 0$$

を満たすとする。

$\dot{p}(t)$ を曲線の接ベクトルといい t が増える方向を曲線の向きという。また $\dot{p}(t) \neq 0$ は尖点が現れないことを保証するものである。

定義 8.2.

$$\tau : [a, b] \ni t \mapsto \tau(t) \in [\alpha, \beta] \quad (\tau'(t) \neq 0)$$

が単調な C^2 級関数のとき

$$p(\tau) = p(\tau(t)) = p(t)$$

を平面曲線のパラメータ変換と呼ぶ。

定理 8.1. $\tau(t)$ が狭義単調増加のとき曲線の向きは変わらない。また $p(t)$ が曲線のとき $p(\tau)$ は曲線である。

Proof. $p(t)$ は曲線であるから

$$\frac{dp(t)}{dt} \neq 0$$

また $\tau(t)$ は仮定より狭義単調増加であるから

$$\frac{d\tau(t)}{dt} > 0$$

よって

$$\frac{dp(\tau)}{d\tau} = \frac{dp(\tau)}{dt}\frac{dt}{d\tau} = \frac{dp(t)}{dt}\frac{dt}{d\tau} \neq 0$$

となり尖点が現れないという意味で曲線である。 □

定義 8.3.

$$s(t) = \int_a^t |\dot{p}(t)|dt$$

を $p(t)$ の弧長と呼ぶ。

$$\frac{ds}{dt} = |\dot{p}(t)| > 0$$

であるから s は t について狭義単調増加なので

$$p(s) = p(s(t)) = p(t)$$

は曲線のパラメータ変換である。

定理 8.2. 曲線の長さはパラメータの取り方によらない。

Proof.

$$|\dot{p}(t)| = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}$$

から $p(t)$ の長さは

$$L(p) = \int_a^b |\dot{p}(t)|dt$$

で与えられる。よって積分範囲を変更すると

$$\tau = \beta \to t = b$$
$$\tau = \alpha \to t = a$$

$$\int_\alpha^\beta |p'(\tau)|d\tau = \int_\alpha^\beta \left|\frac{dp(\tau)}{d\tau}\right| d\tau = \int_\alpha^\beta \left|\frac{dp(t)}{dt}\right| \left|\frac{dt}{d\tau}\right| d\tau = \int_a^b |\dot{p}(t)|dt$$

□

定義 8.4. s を弧長パラメータと呼ぶ。

定義 8.5. $p(s) : [0, l] \to \mathbb{E}^2$ を弧長パラメータ表示された曲線とするとき

1. $e_1(s) = p'(s) = \begin{pmatrix} x'(s) \\ y'(s) \end{pmatrix}$ *:* 単位接ベクトル
2. $e_2(s) = \begin{pmatrix} -y'(s) \\ x'(s) \end{pmatrix}$ *:* 単位法ベクトル

$e_1(s)$、$e_2(s)$ を平面曲線の *Frenet-Serret* 枠と呼ぶ。

1. について

$$|p'(s)| = |\dot{p}(t)| \left|\frac{dt}{ds}\right| = |\dot{p}(t)| \frac{1}{|\dot{p}(t)|} = 1$$

となり $e_1(s)$ は単位ベクトルである。

定義 8.6. 関数 $\kappa(s)$ を用いて

$$p''(s) = \kappa(s)e_2(s)$$

と表せる。そして $\kappa(s)$ を平面曲線の曲率と呼ぶ。

$$\langle p'(s), p'(s) \rangle = 1$$

の両辺を s で微分すると

$$2\langle p''(s), p'(s)\rangle = 0$$

よって $p''(s)$ は $e_1(s)$ と直交するから $e_2(s)$ のスカラー倍になる。

定義 8.7. $e_1(s)$、$e_2(s)$ についての連立常微分方程式

$$\begin{cases} e_1'(s) = & \kappa(s)e_2(s) \\ e_2'(s) = -\kappa(s)e_1(s) & \end{cases}$$

を平面曲線の *Frenet-Serret* 公式と呼ぶ。

第 2 式は

$$\langle e_2(s), e_2(s)\rangle = 1$$

の両辺を微分すると $e_2'(s)$ が $e_1(s)$ 方向を向くことが分かり

$$\langle e_2(s), e_1(s)\rangle = 0$$

の両辺を微分して

$$\langle e_2'(s), e_1(s)\rangle = -\langle e_2(s), e_1'(s)\rangle = -\kappa(s)$$

が分かる。

定理 8.3. $p(s)$ が平面曲線のときその $\kappa(s)$ により運動 *(*回転、平行移動*)* を除いて形が一意に定まる。

Proof. 弧長パラメータ表示された平面曲線 $p(s) : [0, l] \to \mathbb{E}^2$ の Frenet-Serret 枠を用いて 2×2 行列

$$X(s) = (e_1(s) \quad e_2(s))$$

を考える。平面曲線の Frenet-Serret 公式は

$$X'(s) = X(s)A(s), \quad A(s) = \begin{pmatrix} 0 & -\kappa(s) \\ \kappa(s) & 0 \end{pmatrix}$$

と書ける常微分方程式である。常微分方程式の初期値問題の解の存在と一意性の定理より初期値 $X(0)$ を与えると解 $X(s)$ が一意に定まる。ここで

$$X(0) = (e_1(0) \quad e_2(0))$$

の選び方には平面の回転の自由度がある。また

$$p(s) - p(0) = \int_0^s e_1(s)ds$$

となるので $p(0)$ が決まれば $p(s)$ が得られる。つまり $p(s)$ は平行移動の自由度を持つ。よって $p(s)$ が平面曲線のときその $\kappa(s)$ により運動を除いて形が一意に定まることが分かる。 □

定理 8.4. $p(s)$ が平面曲線のときその $\kappa(s)$、*Frenet-Serret* 枠 $e_1(0)$、$e_2(0)$、初期位置 $p(0)$ により運動 *(*回転、平行移動*)* も含めて形が一意に定まる。

Proof. 定理 8.3 から。 □

定義 8.8. 未知関数 $e_1(s)$、$e_2(s)$、$p(s)$ に関する常微分方程式の初期値問題を

$$e_1(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad e_2(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad p(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

として平面曲線の *Frenet-Serret* 公式を応用して

$$\begin{cases} e_1'(s) = & 2se_2(s) \\ e_2'(s) = -2se_1(s) & \\ p'(s) = e_1(s) & \end{cases}$$

とする。

$p(t) = (C(t), S(t))^\top$ のパラメータ t にパラメータ変換を行い弧長パラメータ s に変換することを考える。まず $p(t)$ が描く平面曲線の弧長を求める。

$$\int_0^t \sqrt{(\cos t^2)^2 + (\sin t^2)^2} dt$$

$$\int_0^t dt = t$$

都合が良いことに t はそもそも弧長パラメータであることが分かった。よってパラメータ変換をする必要がない。しかし話の整合性を保つために t を s に変換する。

$$p(s) = \begin{pmatrix} C(s) \\ S(s) \end{pmatrix} = \begin{pmatrix} \int_0^s \cos s^2 ds \\ \int_0^s \sin s^2 ds \end{pmatrix}$$

から

$$\begin{cases} e_1(s) = \begin{pmatrix} \cos s^2 \\ \sin s^2 \end{pmatrix} \\ e_2(s) = \begin{pmatrix} -\sin s^2 \\ \cos s^2 \end{pmatrix} \end{cases} \quad \begin{cases} e_1'(s) = \begin{pmatrix} -2s \sin s^2 \\ 2s \cos s^2 \end{pmatrix} \\ e_2'(s) = \begin{pmatrix} -2s \cos s^2 \\ -2s \sin s^2 \end{pmatrix} \end{cases}$$

が分かり $\kappa(s) = 2s$ を得た。定理 8.4 を応用した定義 8.8 の常微分方程式の初期値問題を用いて Clothoid を描画したい。しかし既に $p(s)$ が $\kappa(s)$ 導出の過程で求まってしまっていることは留意するべきである。特別に Clothoid を描画するために $p(t) = (C(t), S(t))^\top$ の具体的な式が分かっている前提で $\kappa(s)$ を求めているので $p(s)$ も $\kappa(s)$ の導出過程で分かってしまっている。よってこのアルゴリズムは $p(s)$ が求まっているときあまり意味をなさず $\kappa(s)$、$e_1(0)$、$e_2(0)$、$p(0)$ のみ分かっているときに意味を持つ。以下に入力である Python のコード (Listing8) と出力 (図 10) を示す。このコードは $p(s)$ が求まっていることは仮定していない。なおパラメータの区間は $[0, 100]$、区間幅 (曲線を描くための変数の刻み幅) は 0.01 である。

Listing 8: Clothoid by frenetSerret formulas

```python
import numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
from scipy.integrate import solve_ivp  # Function to solve initial value problems for ODEs

# Define the Frenet-Serret equations for planar curves
def frenet_serret(s, y, kappa_func):
    """
    Compute derivatives based on Frenet-Serret equations.

    Parameters:
    s: float
        Curve parameter
    y: ndarray
        State vector [T, N, r], where:
        T: Tangent vector (2D)
        N: Normal vector (2D)
        r: Position vector (2D)
    kappa_func: function
        Curvature as a function of s

    Returns:
    dyds: ndarray
        Derivative of the state vector
    """
    T = y[0:2]  # Tangent vector (2D)
    N = y[2:4]  # Normal vector (2D)
    r = y[4:6]  # Position vector (2D)

    kappa = kappa_func(s)  # Curvature at parameter s

    # Compute derivatives
    dT_ds = kappa * N  # Tangent vector derivative
    dN_ds = -kappa * T  # Normal vector derivative
```

```
    dr_ds = T  # Position vector derivative

    return np.concatenate([dT_ds, dN_ds, dr_ds])

# Function to compute and plot the planar curve
def reconstruct_and_plot(kappa_func, s_span=(0, 100), initial_conditions=None,
  resolution=10000):
    """
    Solve Frenet-Serret equations and plot the planar curve.

    Parameters:
    kappa_func: function
        Curvature as a function of s
    s_span: tuple
        Range of parameter s (start, end)
    initial_conditions: list or None
        Initial values for T (tangent), N (normal), and r (position)
    resolution: int
        Number of points for parameter s
    """
    if initial_conditions is None:
        T0 = [1, 0]  # Initial tangent vector
        N0 = [0, 1]  # Initial normal vector
        r0 = [0, 0]  # Initial position
        initial_conditions = T0 + N0 + r0

    s_eval = np.linspace(s_span[0], s_span[1], resolution)  # Parameter range

    # Solve Frenet-Serret equations
    solution = solve_ivp(
        frenet_serret, s_span, initial_conditions, args=(kappa_func,), t_eval=
          s_eval, method='RK45',rtol=1e-6, atol=1e-8
    )

    r = solution.y[4:6]  # Extract position vectors
    x, y = r[0], r[1]  # Separate x and y components

    # Plot
    plt.figure(figsize=(10, 10))
    plt.plot(x, y, label='Clothoid   by frenetSerret formulas', color='blue')
    plt.title('Parametric   Plot of $(C(t), S(t))$')
    plt.xlabel('C(t)')
    plt.ylabel('S(t)')
    plt.grid(True)
    plt.axis('equal')
    plt.legend()
    plt.show()

# Define curvature function (linear increase)
kappa_func = lambda s: 2.0 * s

# Generate and plot the curve
reconstruct_and_plot(kappa_func, s_span=(0, 100))
```

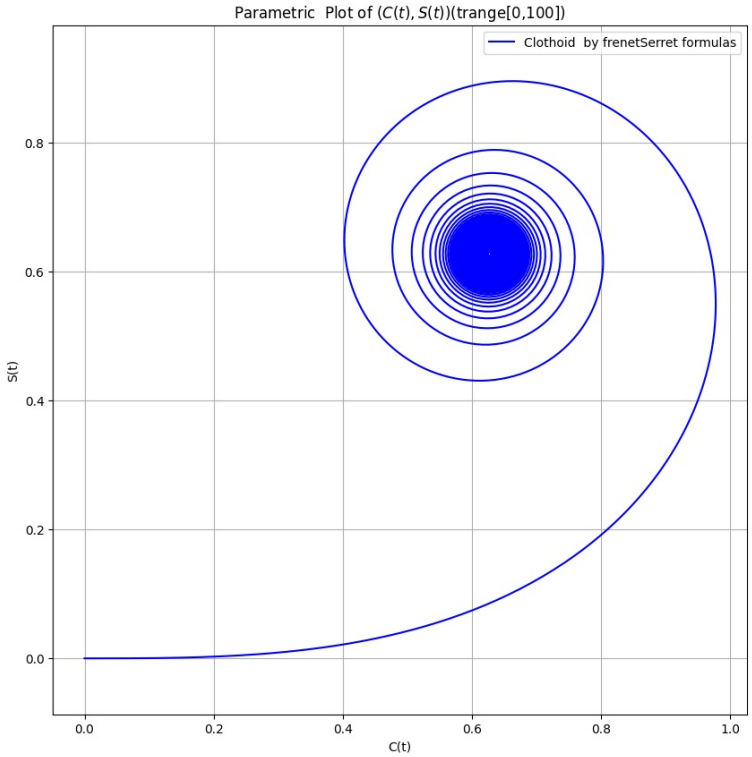

図 10:

Clothoid を描画する際に費やす計算時間量を求めたい。1000 回実行したときの 1 回あたりの算術平均時間を計算時間量とする。以下に入力である Python のコード (Listing9) と出力 (図 11) を示す。

Listing 9: Computation Time per Run

```
iimport numpy as np  # Library for numerical computations
import matplotlib.pyplot as plt  # Library for plotting
from scipy.integrate import solve_ivp  # Function to solve initial value problems
  for ODEs
import time

# Define the Frenet-Serret equations for planar curves
def frenet_serret(s, y, kappa_func):

    T = y[0:2]  # Tangent vector (2D)
    N = y[2:4]  # Normal vector (2D)
    r = y[4:6]  # Position vector (2D)

    kappa = kappa_func(s)  # Curvature at parameter s

    # Compute derivatives
    dT_ds = kappa * N  # Tangent vector derivative
    dN_ds = -kappa * T  # Normal vector derivative
```

```
    dr_ds = T  # Position vector derivative

    return np.concatenate([dT_ds, dN_ds, dr_ds])

# Function to compute and plot the planar curve
def reconstruct_and_plot(kappa_func, s_span=(0, 100), initial_conditions=None,
  resolution=10000):

    if initial_conditions is None:
        T0 = [1, 0]  # Initial tangent vector
        N0 = [0, 1]  # Initial normal vector
        r0 = [0, 0]  # Initial position
        initial_conditions = T0 + N0 + r0

    s_eval = np.linspace(s_span[0], s_span[1], resolution)  # Parameter range

    # Solve Frenet-Serret equations
    solution = solve_ivp(
        frenet_serret, s_span, initial_conditions, args=(kappa_func,), t_eval=
          s_eval, method='RK45',rtol=1e-6, atol=1e-8
    )

    r = solution.y[4:6]  # Extract position vectors
    x, y = r[0], r[1]  # Separate x and y components

kappa_func = lambda s: 2.0 * s

# Initialize list to store computation times
computation_times = []

# Run the function 100 times and record computation times
for _ in range(1000):
    start_time = time.time()
    # Define curvature function (linear increase)
    reconstruct_and_plot(kappa_func, s_span=(0, 100))
    end_time = time.time()
    computation_times.append(end_time - start_time)

# Compute the average computation time
average_time = np.mean(computation_times)

# Plot the computation times as points
plt.figure(figsize=(10, 5))
plt.plot(computation_times, 'o', label="Computation Time per Run", color='blue')
  # Use 'o' for points
plt.axhline(y=average_time, color='red', linestyle='--', label=f"Average Time = {
  average_time:.4f}s")
plt.xlabel('Run Number')
plt.ylabel('Computation Time (seconds)')
plt.title('Computation Time for 1000 Runs')
plt.legend()
plt.grid(True)
plt.show()

# Print the average computation time
print(f"Average computation time: {average_time:.4f} seconds")
```

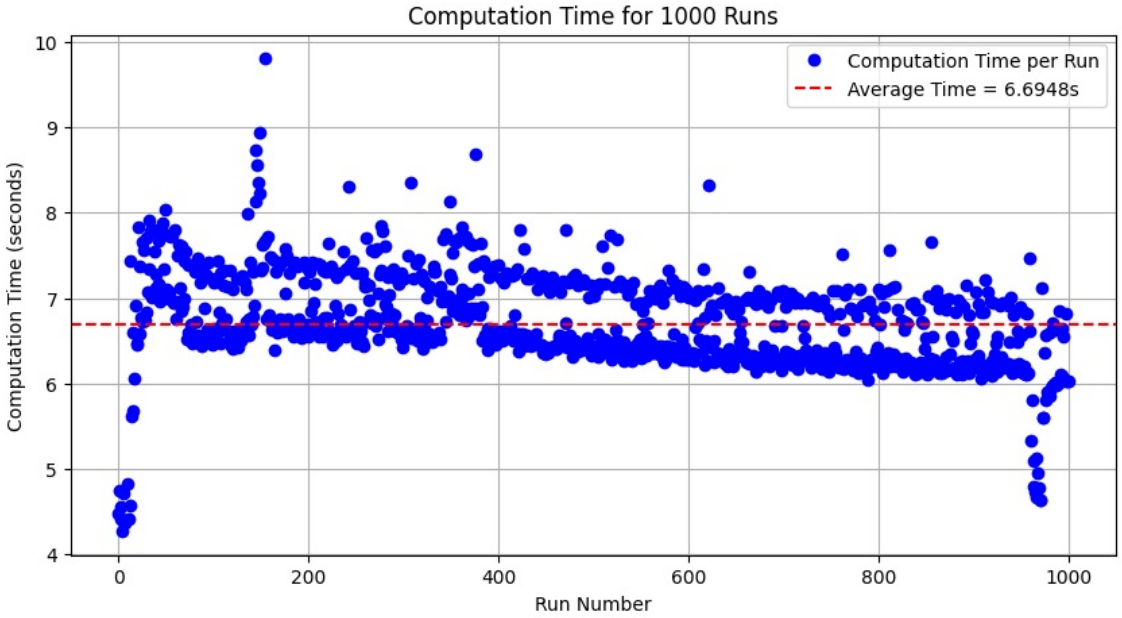

図 11:

第 III 部
Additional notes

9　曲率円と曲率

定義 9.1. 中心が $p(s)+\frac{1}{\kappa}e_2(s)$ で半径が $\frac{1}{|\kappa(s)|}$ の円 $\gamma(t)$ を曲線 p の $p(s)$ における曲率円と呼ぶ。*(s、$p(s)$、$\kappa(s)$、$e_2(s)$ に関する仮定は Section.8 を引き継ぐ)*

定理 9.1. 曲線上の任意の点における曲率円はその曲線と *Maclaurin* 展開の *2* 次の項まで一致する。

Proof. $p(s)$ は無限回微分可能とすると Maclaurin 展開は

$$p(s)=p(0)+\frac{e_1(0)}{1!}s+\frac{\kappa(0)e_2(0)}{2!}s^2+\sum_{k=3}^{\infty}\frac{p^{(k)}(0)}{k!}s^k$$

と書き換えられる。パラメータをずらして $s=0$ で考える。$p(0)=p$、$\kappa(0)=\kappa$、$e_1(0)=e_1$、$e_2(0)=e_2$ と書く。$p+\frac{1}{\kappa}e_2$ を中心とする半径 $\frac{1}{\pm\kappa}$ の円 γ を考える。ただし符号は $\kappa>0$ のとき $+$、$\kappa<0$ のとき $-$ をとる。点 p と γ 上の点のなす曲率円の中心角をパラメータ $t\in\mathbb{R}$ とすると γ は

$$\gamma(t)=p+\frac{1}{\kappa}e_2+\frac{1}{\pm\kappa}\{(\sin t)(e_1)\mp(\cos t)(e_2)\}$$

と表せる。また弧長パラメータを σ として

$$\sigma=|\gamma'(t)|=\frac{t}{\pm\kappa}$$

パラメータ変換を行うと

$$\gamma(\sigma)=p+\frac{1}{\kappa}e_2+\frac{1}{\pm\kappa}\{(\sin(\pm\kappa\sigma))(e_1)\mp(\cos(\pm\kappa\sigma))(e_2)\}$$

となる。両辺を σ で微分すると

$$\frac{d\gamma(\sigma)}{d\sigma}=\cos(\pm\kappa\sigma))(e_1)\pm\sin(\pm\kappa\sigma)(e_2)$$

$\sigma=0$ のとき 1 階微分は e_1。2 階微分を考えると

$$\frac{d^2\gamma(\sigma)}{d\sigma^2}=\pm\kappa(-\sin(\pm\kappa\sigma))(e_1)\pm\cos(\pm\kappa\sigma)(e_2))$$

$\sigma=0$ のとき κe_2 となる。無限回微分可能なので Maclaurin 展開でき

$$\gamma(\sigma)=\gamma(0)+\frac{e_1(0)}{1!}\sigma+\frac{\kappa(0)e_2(0)}{2!}\sigma^2+\sum_{k=3}^{\infty}\frac{\gamma^{(k)}(0)}{k!}\sigma^k$$

となり $p(s)$ と $\gamma(t)$ が $p(0)$ において 2 次の項まで一致していることが示された。他の点でも同様に示せば良い。 □

曲線の各点で曲率円がその曲線の良い近似になっていることが分かる。そして曲率円は $\kappa(s)$ によって定義される。曲率が大きい (曲率円の半径が小さい) ほど曲線のその点での曲がりが急になることは想像しやすい。逆もまた然り。

10 Frenet-Serret 枠を用いた空間曲線の描画

$(C(t),S(t))^\top$ の描画に用いることはないが空間曲線における Frenet-Serret 枠と Frenet-Serret 公式について考えるために Euclid 空間 $\mathbb{E}^3$ の曲線を定義する。平面

曲線との違いは単位法ベクトルの方向が定まらないこと、$p(t)$ の微分可能性を 3 階まで仮定することである。

定義 10.1. $\mathbb{E}^3$*(xyz* 空間内*)* の質点の軌跡

$$p(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} : [a, b] \to \mathbb{E}^3 \quad (t \in \mathbb{R})$$

がパラメータ t で描く空間曲線を列ベクトルで表す。このとき $p(t)$ を空間曲線と呼ぶ。$p(t)$ は C^3 級かつ

$$\dot{p}(t) = \frac{dp(t)}{dt} \neq 0$$

を満たすとする。

空間曲線の弧長パラメータは平面曲線の弧長パラメータと同様の過程で与えられる。

定義 10.2. $p(s) : [0, l] \to \mathbb{E}^3 \quad (p''(s) \neq 0)$ を弧長パラメータ表示された曲線、$\kappa(s)$ を曲率とするとき

1. $e_1(s) = p'(s)$*:* 単位接ベクトル
2. $e_2(s) = \frac{e_1'(s)}{\kappa(s)} \quad (\kappa(s) \neq 0)$*:* 単位法ベクトル
3. $e_3(s) = e_1(s) \times e_2(s)$*:* 単位従法ベクトル

$e_1(s)$、$e_2(s)$、$e_3(s)$ を空間曲線の *Frenet-Serret* 枠と呼ぶ。

平面曲線との違いは $e_1(s)$ に直交する単位ベクトルの選び方に 2 次元の自由度があることである。そこで空間曲線の曲率を

$$\kappa(s) = |p''(s)|$$

で定義し

$$e_2(s) = \frac{p''(s)}{|p''(s)|} = \frac{e_1'(s)}{\kappa(s)}$$

で定める。また $\mathbb{E}^3$ の外積ベクトルを用いて

$$e_3(s) = e_1(s) \times e_2(s)$$

と定めると $e_2(s)$、$e_3(s)$ ともに $e_1(s)$ に直交する単位ベクトルとなる。

定義 10.3. 連立常微分方程式

$$\begin{cases} e_1'(s) = & \kappa(s)e_2(s) & \\ e_2'(s) = -\kappa(s)e_1(s) & & +\tau(s)e_3(s) \\ e_3'(s) = & -\tau(s)e_2(s) & \end{cases}$$

を空間曲線の *Frenet-Serret* 公式、$\tau(s) = e_2'(s)e_3(s)$ を捩率と呼ぶ。

導出は省略する。式から分かるように

$$\tau(s) = 0$$

のとき平面曲線の Frenet-Serret 公式と一致する。

定理 10.1. $p(s)(p''(s) \neq 0)$ が空間曲線のときその $\kappa(s)$ と捩率 $\tau(s)$ により運動を除いて形が一意に定まる。

Proof. 弧長パラメータ表示された空間曲線 $p(s) : [0, l] \to \mathbb{E}^3$ の Frenet-Serret 枠を用いて 3×3 行列

$$X(s) = (e_1(s) \quad e_2(s) \quad e_3(s))$$

を考える。空間曲線の Frenet-Serret 公式は

$$X'(s) = X(s)A(s), \quad A(s) = \begin{pmatrix} 0 & -\kappa(s) & 0 \\ \kappa(s) & 0 & -\tau(s) \\ 0 & \tau(s) & 0 \end{pmatrix}$$

と書ける常微分方程式である。常微分方程式の初期値問題の解の存在と一意性の定理より初期値 $X(0)$ を与えると解 $X(s)$ が一意に定まる。これは各点で正規直交基底を並べたものとなる。ここで

$$X(0) = (e_1(0) \quad e_2(0) \quad e_3(0))$$

の選び方には $\mathbb{E}^3$ の回転の自由度がある。また

$$p(s) - p(0) = \int_0^s e_1(s)ds$$

となるので $p(0)$ が決まれば $p(s)$ が得られる。つまり $p(s)$ は平行移動の自由度を持つ。よって $p(s)$ が空間曲線のときその $\kappa(s)$ と捩率 $\tau(s) > 0$ により運動を除いて形が一意に定まることが分かる。 □

定理 10.2. $p(s)$ が空間曲線のときその $\kappa(s)$、$\tau(s)$、*Frenet-Serret* 枠 $e_1(0)$、$e_2(0)$、$e_3(0)$、初期位置 $p(0)$ により運動 *(*回転、平行移動*)* も含めて形が一意に定まる。

Proof. 定理 10.1 から。 □

定義 10.4. 未知関数 $e_1(s)$、$e_2(s)$、$e_3(s)$、$p(s)$ に関する常微分方程式の初期値問題を

$$e_1(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2(0) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3(0) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad p(0) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

として空間曲線の *Frenet-Serret* 公式を応用して

$$\begin{cases} e_1'(s) = & 2se_2(s) \\ e_2'(s) = -2se_1(s) & +0.5e_3(s) \\ e_3'(s) = & -0.5e_2(s) \\ p'(s) = e_1(s) \end{cases}$$

とする。

定理 10.2 を応用した定義 10.4 の常微分方程式の初期値問題を用いて空間曲線を描画したい。曲率は $2s$、捩率は 0.5 とする。以下に入力である Python のコード (Listing10) と出力 (図 12) を示す。

Listing 10: Clothoid by frenetSerret formulas

```python
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# Definition of the Frenet-Serret differential equations
def frenet_serret(s, y, kappa_func, tau_func):
    T = y[0:3]  # Tangent vector
    N = y[3:6]  # Principal normal vector
    B = y[6:9]  # Binormal vector
    r = y[9:12]  # Position vector of the curve

    kappa = kappa_func(s)
    tau = tau_func(s)

    dT_ds = kappa * N
    dN_ds = -kappa * T + tau * B
    dB_ds = -tau * N
```

```
    dr_ds = T  # Position of the curve

    return np.concatenate([dT_ds, dN_ds, dB_ds, dr_ds])

# Reconstruction and plotting
def reconstruct_and_plot(kappa_func, tau_func, s_span=(0, 100), initial_conditions
  =None, resolution=10000):
    if initial_conditions is None:
        T0 = [1, 0, 0]  # Tangent vector
        N0 = [0, 1, 0]  # Principal normal vector
        B0 = [0, 0, 1]  # Binormal vector
        r0 = [0, 0, 0]  # Initial position of the curve
        initial_conditions = T0 + N0 + B0 + r0  # List instead of array

    s_eval = np.linspace(s_span[0], s_span[1], resolution)

    # Solve using Runge-Kutta method
    solution = solve_ivp(
        frenet_serret,
        s_span,
        initial_conditions,
        args=(kappa_func, tau_func),
        t_eval=s_eval,
        method='RK45'
    )

    r = solution.y[9:12]  # Position of the curve
    x, y, z = r[0], r[1], r[2]

    # Check if the torsion is zero
    is_planar = np.allclose([tau_func(s) for s in s_eval], 0, atol=1e-10)

    # Plot for planar curve
    if is_planar:
        plt.figure(figsize=(10, 10))
        plt.plot(x, y, label='Example␣of␣curve', color='blue')
        plt.title('Parametric␣Plot␣of␣$(X(s),␣Y(s))$')
        plt.xlabel('X(t)')
        plt.ylabel('Y(t)')
        plt.grid(True)
        plt.axis('equal')
        plt.legend()
        plt.show()
    else:
        # 3D Plot
        fig = plt.figure(figsize=(10, 10))
        ax = fig.add_subplot(111, projection='3d')
        ax.plot(x, y, z, label='Example␣of␣curve', color='blue')
        ax.set_title('Parametric␣Plot␣of␣$(X(s),Y(s),Z(s))$')

        # Set axis limits
        ax.set_xlim(0, 1)
        ax.set_ylim(0, 1)
        ax.set_zlim(0, 1)

        ax.set_xlabel('X(s)')
        ax.set_ylabel('Y(s)')
        ax.set_zlabel('Z(s)')
        ax.legend()
        plt.show()


```

```
kappa_func = lambda s: 2*s  #Curvature
tau_func = lambda s: 0.5 #Torsion

reconstruct_and_plot(kappa_func, tau_func, s_span=(0, 100))
```

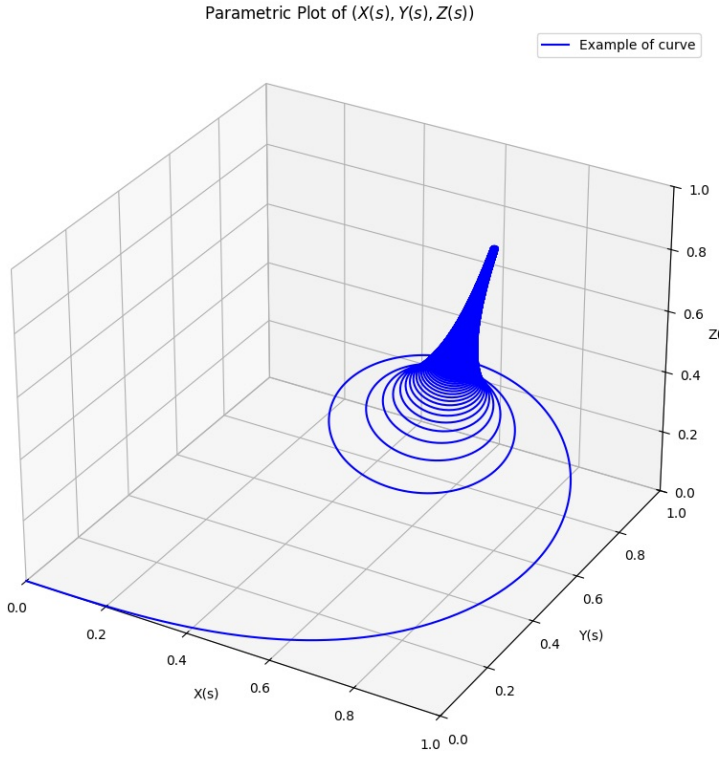

図 12:

第IV部

Conclusion

5つの手法でClothoidを描画することができた。定理2.2からClothoidが$(0.6267, 0.6267)$あたりに収束する様子が分かる。

Method	Time
Python 数値計算ライブラリを用いた描画	0.0378s
常微分方程式の初期値問題を用いた描画	1.4346s
誤差関数を用いた描画	0.0025s
Frenet-Serret 枠を用いた描画	6.6948s

Clothoid の描画にかかる計算時間量を表にした。もちろん計算時間量はコードや実行環境に依存する故に手法間で速さの順序入れ替えが起こる可能性はある。しかし表から手法間で計算時間量に有意な差があることが分かる。

1. 最も計算時間量が大きい「Frenet-Serret 枠を用いた描画」に比べて、最も計算時間量が小さい「誤差関数を用いた描画」は約 2678 倍速い。

2. Fresnel 積分を計算する故に最適な設計がされていると考えられる「Python 数値計算ライブラリ内の関数である scipy.special.fresnel を用いた描画」に比べて、複素数の計算を含む故に数値計算量が比較的大きくなることが予想される「誤差関数を用いた描画」の方が速いだけでなく、約 15 倍速い。

ことが挙げられる。反省点として

1. 使用したコンピュータのスペックが低く「Maclaurin 展開を用いた描画」においてパラメータを長く取ること が出来なかった。

2. 使用した Python 数値計算ライブラリ内の種々の関数の設計を深く調べることが出来なかった。

3. 空間曲線において曲率、捩率が 0 になるときの一意性の検証が出来なかった。

ことが挙げられる。Fresnel 積分を基礎として Clothoid を描画する手法に焦点を当てた本論文を執筆する中で常微分方程式の数値解法、複素関数論、微分幾何学、Python などに関する知見が深められたことは有意義であった。
本論文を執筆するにあたり多大なご支援とご助言を頂いた皆様に深く感謝いたします。

第 V 部
References

[1] 宮岡礼子, **曲線と曲面の現代幾何学** , 岩波書店,2019 年

[2] **特殊関数探訪 三角関数からはじめる不思議な世界**, 日本評論社,2024 年

[3] Fresnel 積分, *Wikipedia*,https://ja.wikipedia.org/wiki/フレネル積分,(2025/2/4)

[4] Clothoid, *Wikipedia*,https://ja.wikipedia.org/wiki/クロソイド曲線,(2025/2/4)

[5] 誤差関数, *Wikipedia*,https://ja.wikipedia.org/wiki/誤差関数,(2025/2/4)

第 VI 部
Computational Environment

Item	Description
PC	MacBook Air Retina 13-inch 2020
OS	macOS Sequoia
CPU	1.1 GHz デュアルコア Intel Core i3
RAM	8 GB 3733 MHz LPDDR4X
GPU	Intel Iris Plus Graphics 1536 MB
TeX 環境	TeX Live 2024
エディタ	TeXShop