1 核心算法

1.1 Romberg 积分

简单的梯形积分收敛速度太慢，Simpson 积分精度不高。Romberg 积分从自动控制精度算法出发，迭代实现更高的精度以及更快的收敛速度。

依据梯形公式增加分点的迭代公式，对于 n 等分的区间 $[a,b]$，有：

$$T_{2n}(f)=\frac{T_n}{2}(f)+h_{2n}\sum_{i=1}^{n}f(a+(2i-1)h_{2n})$$

其中 $h_{2n}=\dfrac{b-a}{2n}$。考虑截断误差：

$$I(f)-T_n(f)=-\frac{b-a}{12}h^2f''(\xi)$$

$$I(f)-T_{2n}(f)=-\frac{b-a}{12}\left(\frac{h}{2}\right)^2f''(\eta)$$

由 $f''(\xi)\approx f''(\eta)$，可以得到：

$$\begin{aligned}I(f)-T_{2n}(f)&\approx\frac{1}{3}(T_{2n}(f)-T_n(f))\\ \Longrightarrow I(f)&\approx T_{2n}(f)+\frac{1}{3}(T_{2n}(f)-T_n(f))=S_n(f)\end{aligned}$$

即利用外推法将梯形求积公式线性组合成 Simpson 求积公式，截断误差由 $O(h^2)$ 提高到 $O(h^4)$。对 Simpson 公式做同样的处理：

$$I(f)\approx S_{2n}(f)+\frac{1}{15}(S_{2n}(f)-S_n(f))=C_n(f)$$

得到 Cotes 公式，误差提高到 $O(h^6)$。继续对该公式做处理：

$$R_n(f)=C_{2n}(f)+\frac{1}{63}(C_{2n}(f)-C_n(f))$$

误差为 $O(h^8)$，该结果即为 Romberg 积分。令 $分点数=2^{k-1}n$，$h_k=\dfrac{h}{2^{k-1}}$，可以继续对 R_n 进行迭代：

$$R_{k,j}=R_{k,j-1}+\frac{R_{k,j-1}-R_{k-1,j-1}}{4^{j-1}-1},\quad k=2,3,\cdots$$

其中 $j=1,2,\cdots$ 分别表示梯形积分、Simpson 积分、Cotes 积分等。对每一个 k，从 $j=2$ 做到 $j=k$，直到 $|R_{k,k}-R_{k-1,k-1}|<\varepsilon$ 时停止计算，详见 Algorithm 1。

1.2 插值

根据实验要求质点运动轨迹，需对加速度求两次积分，由于加速度表达式为解析表达，积分后得到的速度函数为一个点列，为求得更精确的质点运动轨迹，需对该点列求插值函数，再对插值函数求积分得到质点运动轨迹方程。

在本次实验中，我选择线性插值对速度点列进行处理。若有插值点 $(x_0, f(x_0)), (x_1, f(x_1))$，可以由两点式得到插值函数：

$$L_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

由该插值函数作为速度的近似，详见 Algorithm 2。

2 实验结果

2.1 质点轨迹

两次积分得到质点轨迹如图：

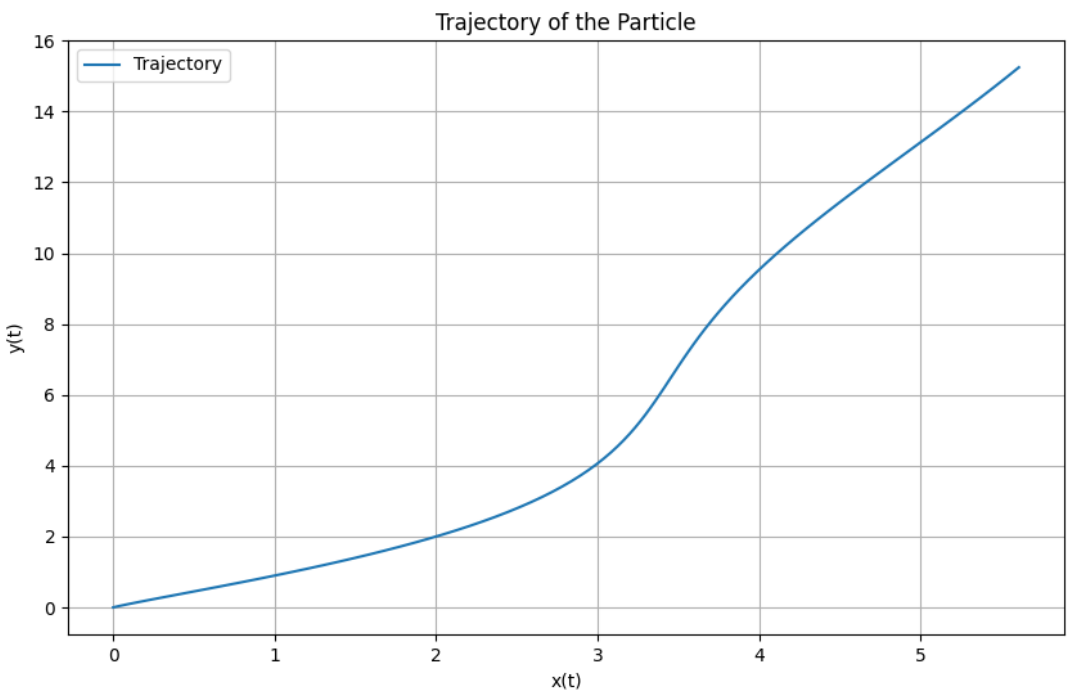

图 1: 质点轨迹

2.2 分析

从质点轨迹来看，质点在 x 方向先加速再减速再加速，符合表达式中 sin 函数的周期特性，y 方向加速度逐渐增大，符合 log 函数的特性。在 Romberg 算法中，参数 M 控制积分插值点的个数，M 越大，积分值越精确，图像越光滑，反之图像越粗糙。

A 附录

A.1 Romberg 积分算法

Algorithm 1: Romberg 积分算法

Input: 区间端点 a, b，控制精度 ε，循环次数 M，被积函数 $f(x)$，$n=1$，$h_1 = b-a$

Output: $\int_a^b f(x)\,\mathrm{d}x$

```
R_{1,1} ← h/2 (f(a) + f(b));
for k = 2 to M do
    h_k = h_{k-1} / 2;
    R_{k,1} ← ( R_{k-1,1} + h_{k-1} Σ_{i=1}^{2^{k-2}} f(a + (2i-1)h_k) ) / 2;
    for j = 2 to k do
        R_{k,j} ← R_{k,j-1} + (R_{k,j-1} - R_{k-1,j-1}) / (4^{j-1} - 1);
        if |R_{k,k} - R_{k-1,k-1}| < ε then
            return R_{k,k};
        end
    end
end
return R_{M,M};
```

A.2 线性插值算法

Algorithm 2: 线性插值算法

Input: x, x_1, x_2, y_1, y_2

Output: $f(x)$

```
if x ⩽ x_1 then
    return y_1
end
else
    if x ⩾ x_2 then
        return y_2
    end
    else
        return (x - x_2)/(x_1 - x_2) y_1 + (x - x_1)/(x_2 - x_1) y_2
    end
end
```