计算物理第二次作业

梁旭民*

Cuiying Hornors College, Lanzhou University
liangxm15@lzu.edu.cn

Abstract

本次计算物理作业主要利用*lorenz*因子混沌方程的粒子学习了*Euler*法和*Rungge-Kutta*法求解常微分方程的初值问题，并尝试讨论行星绕恒星运动轨道及三体（两个恒星一个行星）的运行情况。

I. Runge-Kutta法解常微分方程

A. 问题描述

推导3阶Runge-Kutta方程，并用推导的Runge-Kutta方程求解Lorenz吸引子方程组，画出蝴蝶

$$\begin{cases} \dfrac{dx}{dt} = 10(y-x) \\ \dfrac{dy}{dt} = x(28-z)-y \\ \dfrac{dz}{dt} = xy--\dfrac{8}{3}z \end{cases}$$

初始条件为

$$\begin{cases} x(0)=5 \\ y(0)=20 \\ z(0)=-10 \end{cases}$$

B. 推导3阶Runge-Kutta

考虑下面的常微分方程初值问题

$$\begin{cases} \dfrac{dy}{dx} = f(x,y) \\ y(x_0)=y_0 \end{cases}$$

Rugge-Kutta的思想是在区间$[x_n, x_{n+1}]$上取几个点的斜率，计算它们的平均斜率，从而求解出下一个相近邻位置处的函数值。对于3阶Rugge-Kutta来说，我们在区间$[x_n, x_{n+1}]$区间上取三个点x_n, x_{n+p}, x_{n+q}的斜率值分别为k_1, k_2, k_3的加权平均作为平均斜率$k^* = \lambda_1 k_1 + \lambda_2 k_2 + \lambda_3 k_3$，则有公式的形式为

$$y_{n+1} = y(x_n) + h(\lambda_1 k_1 + \lambda_2 k_2 + \lambda_3 k_3)$$

其中

$$k_1 = f(x_n, y_n)$$

取第二个点为x_{n+p}为

$$x_{n+p} = x_n + np \qquad 0 < p < 1$$

利用k_1来预报y_{n+p},根据Euler格式有x_{n+p}处的y_{n+p}处的预报值

$$\tilde{y}_{n+p} = y_n + phk_1$$

根据原方程可以计算出x_{n+p}处的斜率值为

$$k_2 = f(x_n + ph, y_n + phk_1)$$

取第三个点为x_{n+q}为

$$x_{n+q} = x_n + qh \qquad 0 < p < q < 1$$

利用k_1, k_2来预报y_{n+q}，根据Euler格式有x_{n+q}处的y_{n+q}处的预报值

$$\tilde{y}_{n+q} = y_n + qh(rk_1 + sk_2)$$

根据原方程可以计算出x_{n+q}处的斜率值为

$$k_3 = f(x_{n+q}, \tilde{y}_{n+q}) = f(x_{n+q}, y_n + qh(rk_1 + sk_2))$$

因此整理可得

$$\begin{cases} k_1 = f(x_n, y_n) \\ k_2 = f(x_n + ph, y_n + phk_1) \\ k_3 = f(x_n + qh, y_n + qh(rk_1 + sk_2)) \\ y_{n+1} = y(x_n) + h(\lambda_1 k_1 + \lambda_2 k_2 + \lambda_3 k_3) \end{cases}$$

*指导老师：齐新老师

注意到

$$
\begin{aligned}
&y'(x_n) = f(x_n, y_n)\\
&y''(x_n) = \frac{d}{dx} f(x_n, y_n(x_n))\\
&= f'_x(x_n, y_n) + f'_y(x_n, y_n) f(x_n, y_n)\\
&y^{(3)}(x) = \frac{d}{dx} y''(x_n)\\
&= f''_{xx}(x_n, y_n) + 2f(x_n, y_n) f''_{xy}(x_n, y_n) + f^2(x_n, y_n)\\
&f''_{yy}(x_n, y_n) + f'_y(x_n, y_n)[f'_x(x_n, y_n) + f(x_n, y_n) f'_y(x_n, y_n)]
\end{aligned}
$$

因此则有

$$
\begin{cases}
k_1 = f(x_n, y_n) = y'(x_n)\\
k_2 = f(x_n + ph, y_n + phk_1)\\
\quad = f(x_n, y_n) + phf'_x(x_n, y_n) + phk_1 f'_y(x_n, y_n)\\
\quad + \dfrac{(ph)^2}{2}\left[f''_{xx}(x_n, y_n) + k_1^2 f''_{yy}(x_n, y_n)\right] + O(h^3)\\
k_3 = f(x_n + qh, y_n + qh(rk_1 + sk_2))\\
\quad = f(x_n, y_n) + qhf'_x(x_n, y_n) + qh(rk_1 + sk_2) f'_y(x_n\\
\quad + \dfrac{(qh)^2}{2}[f''_{xx}(x_n, y_n) + (rk_1 + sk_2)^2 f''_{yy}(x_n, y_n)] +\\
y_{n+1} = y(x_n) + h(\lambda_1 k_1 + \lambda_2 k_2 + \lambda_3 k_3)
\end{cases}
$$

与y_{n+1}在x_n处的Taylor展开式

$$
y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2} y''(x_n) + \frac{h^3}{6} y^{(3)}(x_n) + O(h^4)
$$

相比较，可以得到以下不定方程组

$$
\begin{cases}
r + s = 1\\
\lambda_1 + \lambda_2 + \lambda_3 = 1\\
\lambda_2 p + \lambda_3 q = \dfrac{1}{2}\\
\lambda_2 p^2 + \lambda_3 q^2 = \dfrac{1}{3}\\
\lambda_3 pqs = \dfrac{1}{6}
\end{cases}
$$

取$p = \frac{1}{2}, q = 1$，我们可以解得

$$
\begin{cases}
\lambda_1 = \dfrac{1}{6}\\
\lambda_2 = \dfrac{2}{3}\\
\lambda_3 = \dfrac{1}{6}\\
r = -1\\
s = 2
\end{cases}
$$

因此便可以得到我们常用的3阶Runge-Kutta方法

$$
\begin{cases}
k_1 = f(x_n, y_n)\\
k_2 = f(x_x + \dfrac{h}{2}, y_n + \dfrac{h}{2})\\
k_3 = f(x_n + h, y_n - hk_1 + 2hk_2)\\
y_{n+1} = y_n + \dfrac{h}{6}(k_1 + 4k_2 + k_3)
\end{cases}
$$

C. 求解Lorenz吸引子方程

我选择了0.001作为步长，并且求解了100000次，利用Python绘制成Figure 1的Lorenz蝴蝶图像。

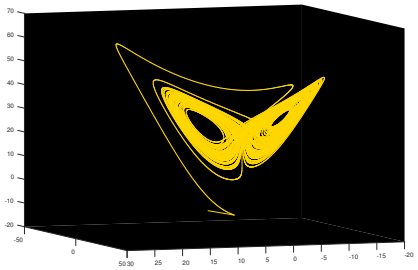

Figure 1: *Lorenz*蝴蝶图像

II. 地球公转与机械能

A. 问题描述

用你所知道的最高精度和最低精度算法计算地球公转轨道。并统计系统机械能。讨论高精度和低精度算法在保持机械能守恒的物理前提下，所消耗的计算资源。

B. 问题分析

由于该问题是两体问题，且受到的是有心力，因此只需要考虑一个平面内的问题即可。设太阳的质量为M，地球的质量为m，约化质量为m_a，选择极坐标系研究该问题，设地球相对于太阳的距离为r，转过的角度为θ，角动量为$L = m_a h$,根据角动量守恒及能量守恒

$$
\begin{cases}
r^2 m_a \dfrac{\partial \theta}{\partial t} = L\\
\dfrac{1}{m_a}\left(\left(\dfrac{\partial r}{\partial t}\right)^2 + r^2\left(\dfrac{\partial \theta}{\partial t}\right)^2\right) - \dfrac{GMm}{r} = E
\end{cases}
$$

为了求解轨道方程我们消去时间t，则可以得到轨道的常微分方程

$$\frac{L}{m_a r^2\sqrt{\frac{2}{m_a}(E-\frac{1}{2}\frac{L^2}{m_a r^2}+\frac{GMm}{r})}}=d\theta$$

即得到r关于θ的一阶常微分方程

$$\frac{dr}{d\theta}=\sqrt{ar^4+br^3-r^2}$$

其中

$$a=\frac{2E(m+M)}{mMh^2}\qquad b=\frac{2G(m+M)}{h^2}$$

C. 数据分析

我在求解地球轨道的时候，分别选择了2、3、4阶Runge-kutta法，其中2阶Runge-kutta法相当于Eular法作为低精度算法，4阶Runge-kutta认为是高精度算法，分别求解出平均轨道$\bar{r}$,然后可以求解出地球公转的平均机械能

$$E=-\frac{2GMm}{r}$$

见Table 1。其中，轨道半径中位数为$r_{\frac{1}{2}}$，公转轨道机械能表征值为$\bar{E}$。

Table 1: 地球公转轨道及机械能

Runge-Kutta阶数	2	3	4
$r_{\frac{1}{2}}(\times 10^{11})$	1.510363	1.510363	1.510363
$\bar{E}(\times 10^{34})$	-1.048877	-1.048877	-1.048877

III. 成为三体星人

A. 问题描述

如上题中，如果太阳系中加入第二颗恒星，导致地球在两颗恒星引力下运动，我们将成为典型的三体星人，讨论此情形下我们的公转轨道。

由于在求解18个方程的常微分方程组的时候对于两颗恒星参数调节以尝试讨论所有情况还没有完全完成，因此这里姑且现将该问题搁置，后续将会对于该问题有专门的讨论。

Appendices

A. Lorenz Equation

Here is the program to solve the Lorenz ODE.

Input C source:

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define N 100000
//Ordinary Differential Equation
double f1(double x,double y,double z)
{
    return(10.0*(y-x));
}
double f2(double x,double y,double z)
{
    return(x*(28.0-z)-y);
}
double f3(double x,double y,double z)
{
    return(x*y-8.0/3*z);
}
int main()
{
    double f1(double x,double y,double z);
    double f2(double x,double y,double z);
    double f3(double x,double y,double z);
    int RungeKutta(double x0,double y0,double z0,
int n,double *x,double *y,double *z,double (*f1)
(double,double,double),double (*f2)(double,double
,double),double (*f3)(double,double,double));
    int i;
    FILE *fp;
    fp = fopen("Lorenz.txt", "w");
    double x0=5.0,y0=20.0,z0=-10.0,x[N],y[N],z[N];
    RungeKutta(x0,y0,z0,N,x,y,z,f1,f2,f3);
    for(int i=0;i<N;i++)
    {
        fprintf(fp,"%lf,%lf,%lf\n", x[i],y[i],z[i]);
    }
    fclose(fp);
    return 0;
}
//3 Order Runge Kutta
int RungeKutta(double x0,double y0,double z0,
int n,double *x,double *y,double *z,double (*f1)
(double,double,double),double (*f2)(double,double
,double),double (*f3)(double,double,double))
{
    double h=0.001,k1,k2,k3,l1,l2,l3,m1,m2,m3;
    int i;
    x[0]=x0;
    y[0]=y0;
    z[0]=z0;
        for(i=0;i<n;i++)
        {
            k1=f1(x[i],y[i],z[i]);
            l1=f2(x[i],y[i],z[i]);
            m1=f3(x[i],y[i],z[i]);
            k2=f1(x[i]+h/2.0,y[i]+h*k1/2.0,z[i]
```

```
+h*k1/2.0);
            l2=f2(x[i]+h/2.0,y[i]+h*l1/2.0,z[i]
+h*l1/2.0);
            m2=f3(x[i]+h/2.0,y[i]+h*l1/2.0,z[i]
+h*l1/2.0);
            k3=f1(x[i]+h,y[i]-h*k1+2.0*h*k2,z[i]
-h*k1+2.0*h*k2);
            l3=f2(x[i]+h,y[i]-h*l1+2.0*h*l2,z[i]
-h*l1+2.0*h*l2);
            m3=f3(x[i]+h,y[i]-h*l1+2.0*h*l2,z[i]
-h*l1+2.0*h*l2);
            x[i+1]=x[i]+h*(k1+4.0*k2+k3)/6.0;
            y[i+1]=y[i]+h*(l1+4.0*l2+l3)/6.0;
            z[i+1]=z[i]+h*(m1+4.0*m2+m3)/6.0;
        }
}
```

B. Two body question

Here is the program to solve the Earth Revolution Orbit.

Input C source:

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
//Step
#define N 10000
//Constant
#define a 1.733691835*pow(10,22)
#define b 1.295934646*pow(10,-11)
//Boundary
#define xx0 0.0
#define xxf 2*3.141592653589793238 4626
#define r0 149500000000
int RungeKutta(double x0,double xf,double y0
,int n,double *x,double *y,int style,double
(*f)(double,double))
{
    double h=(xf-x0)/n,k1,k2,k3,k4;
    int i;
    x[0]=x0;
    y[0]=y0;
    switch(style)
    {
    case 2:
    //2 Order Runge-Kutta
    for(i=0;i<n;i++)
    {
    x[i+1]=x[i]+h;
    k1=f(x[i],y[i]);
    k2=f(x[i]+h/2,y[i]+h*k1/2);
    y[i+1]=y[i]+h*k2;
    }
    break;
    case 3:
    //3 Order Runge-Kutta
    for(i=0;i<n;i++)
    {
    x[i+1]=x[i]+h;
    k1=f(x[i],y[i]);
    k2=f(x[i]+h/2,y[i]+h*k1/2);
    k3=f(x[i]+h,y[i]-h*k1+2*h*k2);
```

```
    y[i+1]=y[i]+h*(k1+4*k2+k3)/6;
    }
    break;
    case 4:
    //4 Order Runge—Kutta
    for(i=0;i<n;i++)
    {
    x[i+1]=x[i]+h;
    k1=f(x[i],y[i]);
    k2=f(x[i]+h/2,y[i]+h*k1/2);
    k3=f(x[i]+h/2,y[i]+h*k2/2);
    k4=f(x[i]+h,y[i]+h*k3);
    y[i+1]=y[i]+h*(k1+2*k2+2*k3+k4)/6;
    }
    break;
    default:
    return 0;
    }
    return 1;
}
//Ordinary Differerial Equation
double f(double x,double y)
{
    return(sqrt(a*pow(x,4)+b*pow(x,3)—x*x));
}
//Main Function
int main()
{
    int i;
    double x[N],y[N];
    double f(double x,double y);
    int RungeKutta(double x0,double xf,double
y0,int n,double *x,double *y,int style,double
(*f)(double,double));
    RungeKutta(xx0,xxf,r0,N,x,y,2,f);
    for(int i=0;i<N;i++)
    printf("x[%d]=%f,y[%d]=%f\n",i,x[i],i,y[i]);
    RungeKutta(xx0,xxf,r0,N,x,y,3,f);
    for(i=0;i<N;i++)
    printf("x[%d]=%f,y[%d]=%f\n",i,x[i],i,y[i]);
    RungeKutta(xx0,xxf,r0,N,x,y,4,f);
    for(i=0;i<N;i++)
    printf("x[%d]=%f,y[%d]=%f\n",i,x[i],i,y[i]);
    return 1;
}
```

参考文献

[1] Runge, Carl David Tolmé (1895), "Über die numerische Auflösung von Differentialgleichungen", Mathematische Annalen, Springer, 46 (2): 167–178, doi:10.1007/BF01446807.

[2] Kutta, Martin Wilhelm (1901), Beitrag zur näherungweisen Integration totaler Differentialgleichungen.

[3] Ascher, Uri M.; Petzold, Linda R. (1998), Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, Philadelphia: Society for Industrial and Applied Mathematics, ISBN 978-0-89871-412-8.

[4] Atkinson, Kendall A. (1989), An Introduction to Numerical Analysis (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-50023-0.

[5] Butcher, John C. (May 1963), Coefficients for the study of Runge-Kutta integration processes, 3 (2), pp. 185–201, doi:10.1017/S1446788700027932.

[6] Butcher, John C. (1975), "A stability property of implicit Runge-Kutta methods", BIT, 15: 358–361, doi:10.1007/bf01931672.

[7] Butcher, John C. (2008), Numerical Methods for Ordinary Differential Equations, New York: John Wiley & Sons, ISBN 978-0-470-72335-7.

[8] Williams, David R. (2004-09-01). "Earth Fact Sheet". NASA. Retrieved 2007-03-17.