实验七 数值计算方法应用(6 课时)

一、实验目的

初步具有使用数值计算方法解决实际问题的能力。

二、实验要求

在课后完成问题的分析，并编程上机求解。

三、实验内容

(一) 使用课程有关“非线性方程求解”的知识上机解决以下问题(在学习通上有作业,大家完成后，写成报告)：

1.弦位法求解高阶多项式:

给定 n 阶多项式 Pn(x)，先用弦位法求解一个根，两个初始值取为 0 和 1，求得的前后两个近似解相差的绝对值小于 1.e-8 退出迭代，或者迭代次数达到 50 就退出迭代。求得一个根之后，比如 5，用（x-5）除 Pn(x)，得到一个新的 n-1 阶多项式，然后重复用这个过程，求出所有根。

输入格式：
第一行，一个整数 n，多项式的阶（n<8）
第二行，（n+1）个实数，是多项式的系数，顺序为常数项、x 的系数、x 平方的系数等等，最后一个系数保证为 1。数与数之间有一个空格。数据确保根为实数而且互异。

输出格式：
n 个实数，依次求得的根，精确到小数点后 4 位，数与数之间有一个空格。
注意：用其它方法求得的根，可能顺序不同，而且有可能最后一两位有效数值不同。一定要用题目给出的方法求。这种方法也是通用的求高阶多项式的方法。

输入样例：
3
-5.7536 13.7022 -6.9327 1.0000
输出样例：
0.5716 2.9578 3.4033

（二）使用课程有关“常微分方程组的解法”的知识上机解决以下问题（在学习通上有作业，大家完成后，写成报告）

1. 三体模拟的四阶荣格-库塔法：

给定平面三体的质量、坐标、初始速度。用四阶荣格-库塔法进行模拟，万有引力常数 G=1，计算时间步长 0.1，输出第 1 步、第 101 步、第 201 步等等的坐标结果（初始状态算第一步)。

三体模拟的 Eular 法参考 https://www.guanjihuan.com/archives/858。如果在此基础上改为
四阶荣格-库塔法，非常麻烦。大家要建立良好的多变量四阶荣格-库塔法框架，才会又快又好。高阶可转化为多变量。

输入格式（数之间有一个空格)：
第一行，三个实数，三个天体的质量
第二行，六个实数，三个天体的坐标，第一个天体的 x 和 y 坐标，第二个、第三个。
第三行，六个实数，三个天体的速度，第一个天体的 x 和 y 速度，第二个、第三个。

输出格式（数之间有一个空格，精确到小数点后 4 位)：
共 20 行，一行是一个时间点的结果，共六个坐标，第一个天体的 x 和 y 坐标，第二个、第三个。
第一行，是第 1 步的结果，就是初始状态，第二行是第 101 步的计算结果，第三行是第 201 步的计算结果，依次递推。

输入样例：
```
15 12 8
300 50 -100 -200 -100 150
0 0 0 0 0 0
```

输出样例：
```
300.0000 50.0000 -100.0000 -200.0000 -100.0000 150.0000
299.9954 49.9991 -99.9971 -199.9949 -99.9957 149.9940
299.9817 49.9966 -99.9886 -199.9798 -99.9829 149.9761
299.9589 49.9923 -99.9743 -199.9545 -99.9615 149.9463
299.9269 49.9863 -99.9543 -199.9192 -99.9315 149.9045
299.8857 49.9785 -99.9285 -199.8737 -99.8930 149.8508
299.8355 49.9691 -99.8971 -199.8181 -99.8459 149.7851
299.7760 49.9579 -99.8599 -199.7524 -99.7902 149.7074
299.7074 49.9450 -99.8170 -199.6765 -99.7259 149.6178
299.6296 49.9304 -99.7683 -199.5905 -99.6531 149.5162
299.5427 49.9141 -99.7139 -199.4944 -99.5717 149.4027
299.4466 49.8960 -99.6538 -199.3881 -99.4816 149.2771
```

299.3413 49.8762 -99.5879 -199.2716 -99.3830 149.1395
299.2268 49.8547 -99.5163 -199.1450 -99.2758 148.9898
299.1031 49.8315 -99.4389 -199.0081 -99.1599 148.8282
298.9701 49.8065 -99.3557 -198.8610 -99.0354 148.6544
298.8280 49.7798 -99.2668 -198.7038 -98.9022 148.4686
298.6765 49.7513 -99.1721 -198.5362 -98.7604 148.2706
298.5159 49.7211 -99.0716 -198.3584 -98.6099 148.0606
298.3459 49.6892 -98.9652 -198.1704 -98.4507 147.8383

参考文献

[1] 易大义等，《计算方法（第二版）》，浙江：浙江大学出版社，2002 年。
[2] 封建湖等，《数值分析原理（第四版）》，北京：科学出版社，2002 年。
[3] 吴筑筑等，《计算方法（第二版）》，北京：清华大学出版社，2004 年。
[4] 钟尔杰等，《数值分析》，北京：高等教育出版社，2004 年。
[5] 刘师少等，《计算方法》，北京：科学出版社，2005 年。

四、代码实现

1.弦位法求解高阶多项式:

根据题意，需要寻找 n 个根
找根：选择初始值 x0=0，x1=1，根据迭代公式 x0 = x1，x1 = x0 - f(x0) * (x1 - x0) / (f(x1) - f(x0))进行迭代，最大迭代次数为 50 次，在精度到达 1e-8 时提前退出迭代
在找到一个根 root 后，需要将原多项式除以(x-root)，得到新的多项式用于求解下一个根，由于多项式除法不好处理，可以在求特定的 f(x)时进行除运算

```
def chord_method(coefficients, roots, x0=0, x1=1, epsilon=1e-8, max_iterations=50):
    def f(x):
        s = sum([c * (x ** i) for i, c in enumerate(coefficients)])
        r = 1
        for i in roots:
            r *= (x - i)
        return s / r

    for iteration in range(max_iterations):
        if abs(x1 - x0) < epsilon:
            break
        newX0 = x1
        newX1 = x0 - f(x0) * (x1 - x0) / (f(x1) - f(x0))
        x0, x1 = newX0, newX1
    return x1



def find_roots(n, coefficients):
```

```
    roots = []
    for i in range(n, 0, -1):
        root = chord_method(coefficients, roots)
        roots.append(root)
    return roots


"""
3
-5.7536 13.7022 -6.9327 1.0000
"""
n = int(input())
coefficients = list(map(float, input().split()))
roots = find_roots(n, coefficients)
for r in roots:
    print(f"{r:.4f}", end=" ")
# print(" ".join(map(str, roots)))
```

运行示例:

2. 三体模拟的四阶荣格-库塔法:

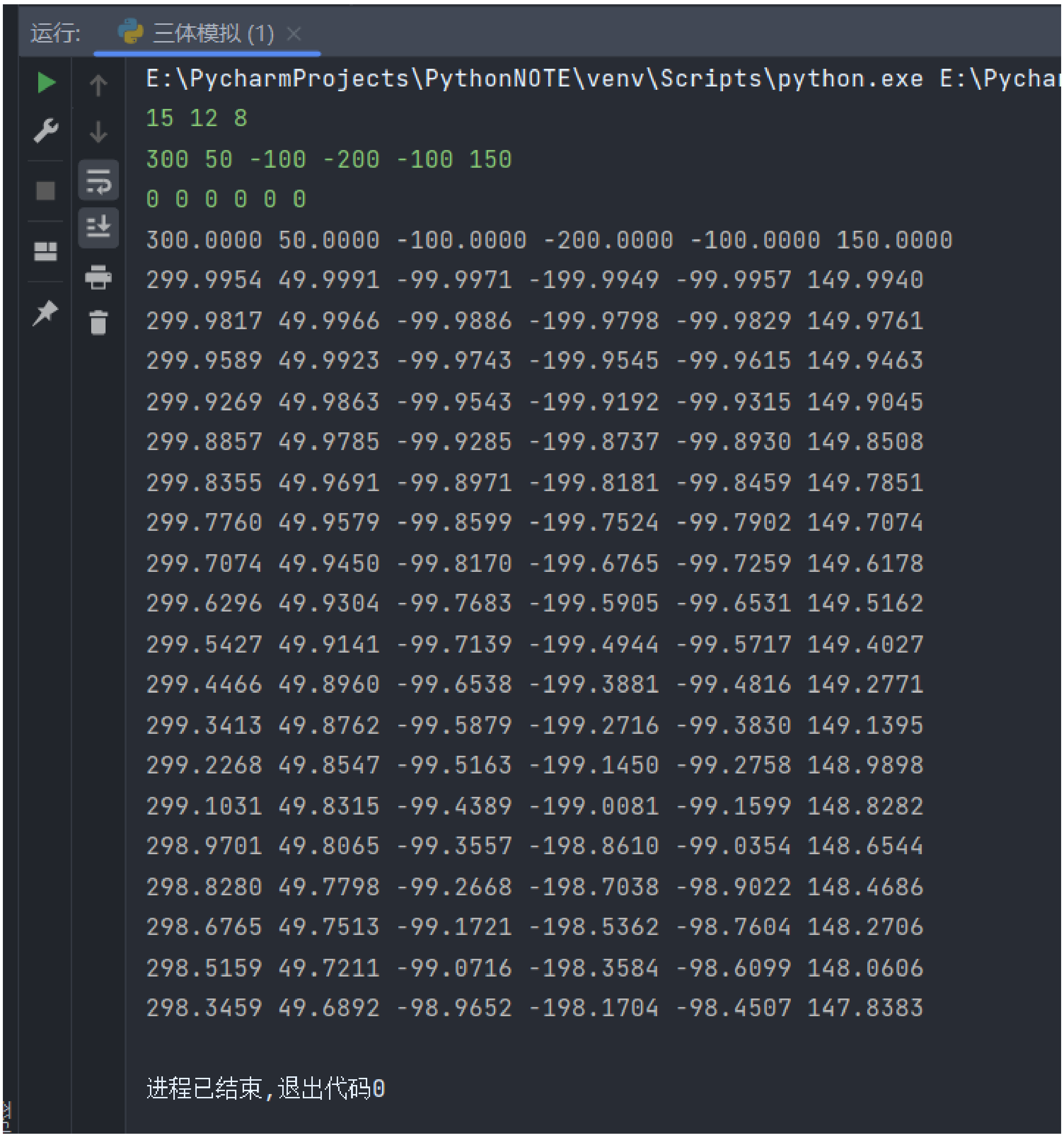
三体模拟，用 Globe 类表示三个球体，每次迭代 100 步输出
迭代时先求出加速度，再求出四阶，最后更新速度以及位置

```
class Globe:
    def __init__(self, m=0, x=0, y=0, vx=0, vy=0):
        self.m = m
        self.x = x
        self.y = y
        self.vx = vx
        self.vy = vy
```

```python
dt = 0.1
N = 3
"""
15 12 8
300 50 -100 -200 -100 150
0 0 0 0 0 0
"""
globes = [Globe() for _ in range(N)]
mass = list(map(float, input().split()))
for i in range(N):
    globes[i].m = mass[i]
pos = list(map(float, input().split()))
for i in range(0, N * 2, 2):
    globes[i // 2].x, globes[i // 2].y = pos[i], pos[i + 1]
v = list(map(float, input().split()))
for i in range(0, N * 2, 2):
    globes[i // 2].vx, globes[i // 2].vy = v[i], v[i + 1]


def cal(globes):
    a = List()
    for i in range(N):
        for j in range(N):
            if i == j:
                continue
            dx = globes[j].x - globes[i].x
            dy = globes[j].y - globes[i].y
            distance = (dx * dx + dy * dy) ** 0.5
            force = globes[j].m / (distance * distance * distance)
            a[i][0] += force * dx
            a[i][1] += force * dy
    return a


def List():
    return [[0] * 2 for _ in range(N)]


def copy(k_a, a, k_v, globes):
    for i in range(N):
        k_a[i] = a[i].copy()
        k_v[i] = [globes[i].vx, globes[i].vy]
```

```
def globeClone(globes, k_a, k_v, e):
    globes_clone = [Globe() for _ in range(N)]
    for i in range(N):
        globes_clone[i] = Globe(globes[i].m, globes[i].x, globes[i].y, globes[i].vx,
globes[i].vy)
        globes_clone[i].x += dt * k_v[i][0] * e
        globes_clone[i].y += dt * k_v[i][1] * e
        globes_clone[i].vx += dt * k_a[i][0] * e
        globes_clone[i].vy += dt * k_a[i][1] * e
    return globes_clone


def runge(globes):
    a = cal(globes)

    a1, a2, a3, a4 = List(), List(), List(), List()
    v1, v2, v3, v4 = List(), List(), List(), List()
    copy(a1, a, v1, globes)
    globes_clone = globeClone(globes, a1, v1, 0.5)

    a = cal(globes_clone)
    copy(a2, a, v2, globes_clone)
    globes_clone = globeClone(globes, a2, v2, 0.5)

    a = cal(globes_clone)
    copy(a3, a, v3, globes_clone)
    globes_clone = globeClone(globes, a3, v3, 1)

    a = cal(globes_clone)
    copy(a4, a, v4, globes_clone)
    # 改变量
    for i in range(N):
        globes[i].x += dt * (v1[i][0] + 2 * v2[i][0] + 2 * v3[i][0] + v4[i][0]) /
6.0
        globes[i].y += dt * (v1[i][1] + 2 * v2[i][1] + 2 * v3[i][1] + v4[i][1]) /
6.0
        globes[i].vx += dt * (a1[i][0] + 2 * a2[i][0] + 2 * a3[i][0] + a4[i][0]) /
6.0
        globes[i].vy += dt * (a1[i][1] + 2 * a2[i][1] + 2 * a3[i][1] + a4[i][1]) /
6.0


for step in range(2000):
    if step % 100 == 0:
```

```python
        for i in range(N):
            print(f"{globes[i].x:.4f} {globes[i].y:.4f}", end="")
            print("\n" if i == N - 1 else " ", end="")
    runge(globes)
"""
300.0000 50.0000 -100.0000 -200.0000 -100.0000 150.0000
299.9954 49.9991 -99.9971 -199.9949 -99.9957 149.9940
299.9817 49.9966 -99.9886 -199.9798 -99.9829 149.9761
299.9589 49.9923 -99.9743 -199.9545 -99.9615 149.9463
299.9269 49.9863 -99.9543 -199.9192 -99.9315 149.9045
299.8857 49.9785 -99.9285 -199.8737 -99.8930 149.8508
299.8355 49.9691 -99.8971 -199.8181 -99.8459 149.7851
299.7760 49.9579 -99.8599 -199.7524 -99.7902 149.7074
299.7074 49.9450 -99.8170 -199.6765 -99.7259 149.6178
299.6296 49.9304 -99.7683 -199.5905 -99.6531 149.5162
299.5427 49.9141 -99.7139 -199.4944 -99.5717 149.4027
299.4466 49.8960 -99.6538 -199.3881 -99.4816 149.2771
299.3413 49.8762 -99.5879 -199.2716 -99.3830 149.1395
299.2268 49.8547 -99.5163 -199.1450 -99.2758 148.9898
299.1031 49.8315 -99.4389 -199.0081 -99.1599 148.8282
298.9701 49.8065 -99.3557 -198.8610 -99.0354 148.6544
298.8280 49.7798 -99.2668 -198.7038 -98.9022 148.4686
298.6765 49.7513 -99.1721 -198.5362 -98.7604 148.2706
298.5159 49.7211 -99.0716 -198.3584 -98.6099 148.0606
298.3459 49.6892 -98.9652 -198.1704 -98.4507 147.8383
"""
```
运行示例:




```
E:\PycharmProjects\PythonNOTE\venv\Scripts\python.exe E:\Pychar
15 12 8
300 50 -100 -200 -100 150
0 0 0 0 0 0
300.0000 50.0000 -100.0000 -200.0000 -100.0000 150.0000
299.9954 49.9991 -99.9971 -199.9949 -99.9957 149.9940
299.9817 49.9966 -99.9886 -199.9798 -99.9829 149.9761
299.9589 49.9923 -99.9743 -199.9545 -99.9615 149.9463
299.9269 49.9863 -99.9543 -199.9192 -99.9315 149.9045
299.8857 49.9785 -99.9285 -199.8737 -99.8930 149.8508
299.8355 49.9691 -99.8971 -199.8181 -99.8459 149.7851
299.7760 49.9579 -99.8599 -199.7524 -99.7902 149.7074
299.7074 49.9450 -99.8170 -199.6765 -99.7259 149.6178
299.6296 49.9304 -99.7683 -199.5905 -99.6531 149.5162
299.5427 49.9141 -99.7139 -199.4944 -99.5717 149.4027
299.4466 49.8960 -99.6538 -199.3881 -99.4816 149.2771
299.3413 49.8762 -99.5879 -199.2716 -99.3830 149.1395
299.2268 49.8547 -99.5163 -199.1450 -99.2758 148.9898
299.1031 49.8315 -99.4389 -199.0081 -99.1599 148.8282
298.9701 49.8065 -99.3557 -198.8610 -99.0354 148.6544
298.8280 49.7798 -99.2668 -198.7038 -98.9022 148.4686
298.6765 49.7513 -99.1721 -198.5362 -98.7604 148.2706
298.5159 49.7211 -99.0716 -198.3584 -98.6099 148.0606
298.3459 49.6892 -98.9652 -198.1704 -98.4507 147.8383

进程已结束,退出代码0
```