```python
#! python3.7
# -*- coding: utf-8 -*-
from numpy import zeros, linspace, log, sqrt, sum
from matplotlib.pyplot import style, figure, axes

# Функция f подготавливает массив, содержащий элементы вектор-функции,
# определяющей правую часть решаемой системы ОДУ
def f(u,m_sun,G) :
    f = zeros(4)
    f[0] = u[2]
    f[1] = u[3]
    f[2] = - G*m_sun*u[0]/(u[0]**2 + u[1]**2)**(3/2)
    f[3] = - G*m_sun*u[1]/(u[0]**2 + u[1]**2)**(3/2)
    return f

# Функция реализует решение системы ОДУ
# на сетке с M_0*r**s интервалами по схеме ERK2
def ODESolving(t_0,T,x_0,y_0,v_x_0,v_y_0,M_0,s,r,m_sun,G) :
    # Входные параметры:
    # t_0, T - начальный и конечный моменты счёта
    # x_0, y_0, v_x_0, v_y_0 - начальные условия
    # M_0 - число интервалов базовой сетки по времени
    # s - номер сетки, на которой вычисляется решение
    # (если s = 0, то решение ищется на базовой сетке)
    # r - коэффициент сгущения сетки
    # m_sun, G - параметры задачи

    # Выходной параметр:
    # u_basic - массив, содержащий сеточные значения
    # решения системы ОДУ только в узлах,
    # совпадающих с узлами базовой сетки

    # Формирование сгущённой в r^s раз сетки с номером s:

    # Вычисление числа интервалов на сетке с номером s
    M = M_0*r**s
    # Определение шага сгущённой сетки
    tau = (T - t_0)/M
    # Определение сгущённой сетки
    t = linspace(t_0,T,M + 1)

    # Выделение памяти под массив сеточных значений решения системы ОДУ,
    # в котором будут храниться сеточные значения из узлов,
    # совпадающих с узлами базовой сетки
    u_basic = zeros((M_0 + 1,4))

    # Выделение памяти под массив сеточных значений
    # решения на сгущённой сетке
    # В строке с номером m этого массива хранятся сеточные значения решения,
    # соответствующие моменту времени t_m
    u = zeros((M + 1,4))

    # Задание начального условия
    u[0] = [x_0, y_0, v_x_0, v_y_0]

    # Реализация схемы ERK2
    for m in range(M) :
        w_1 = f(u[m],m_sun,G)
        w_2 = f(u[m] + tau*2/3*w_1,m_sun,G)
        u[m + 1] = u[m] + tau*(1/4*w_1 + 3/4*w_2)

    # Из массива u выбираются сеточные значения из узлов,
    # совпадающих с узлами базовой сетки
    for m in range(M_0 + 1) :
        u_basic[m] = u[m*r**s]

    return u_basic

# Определение входных данных задачи
```

```
t_0 = 0.; T = 365.25*24*60*60
x_0 = 147098291*10**3; y_0 = 0.
v_x_0 = 0.; v_y_0 = 30.4*10**3
G = 6.674301515151515*10**(-11)
m_sun = 1.98847*10**30

# Определение числа интервалов БАЗОВОЙ сетки,
# на которой будет искаться приближённое решение
M = 36

# Число сеток, на которых ищется приближённое решение
S = 7
# Коэффициент сгущения сеток
r = 2
# Теоретические параметры схемы
p = 2; q = 1

# Выделение памяти под массивы сеточных значений
# решений ОДУ на разных сетках с номерами s = 0,...,S-1,
# в которых хранятся сеточные значения решения из узлов,
# совпадающих с узлами базовой сетки
U = zeros((S,S,M + 1,4))

# "Большой цикл", который пересчитывает решение S раз
# на последовательности сгущающихся сеток
# Массив сеточных значений решения содержит только
# сеточные значения из узлов, совпадающих с узлами базовой сетки
for s in range(S) :
    U[s,0,:,:]  = ODESolving(t_0,T,x_0,y_0,v_x_0,v_y_0,M,s,r,m_sun,G)

# Выделение памяти под массивы ошибок R,
# относительных ошибок R_rel и эффективных порядков точности p_eff
R = zeros((S,S,M + 1,4))
R_rel = zeros((S,S))
p_eff = zeros((S,S))

for s in range(1,S) :
    for l in range(s) :
        R[s,l,:,:] = (U[s,l,:,:] - U[s-1,l,:,:])/(r**(p + l*q) - 1)
        U[s,l+1,:,:] = U[s,l,:,:] + R[s,l,:,:]
        R_rel[s,l] = sqrt(sum(R[s,l,:,:]**2))/sqrt(sum(U[s,l+1,:,:]**2))*100

for s in range(2,S) :
    for l in range(s-1) :
        p_eff[s,l] = log(sqrt(sum(R[s-1,l,:,:]**2))/sqrt(sum(R[s,l,:,:]**2)))/log(r)

# Функция выводит форматированную таблицу
def PrintTriangular(A,i) :
    print('     ',end=' ')
    for l in range(len(A)) :
        print(' p={0:<4d}'.format(p + l*q),end=' ')
    print()
    for m in range(len(A)) :
        print('s={0:<2d}'.format(m),end=' ')
        for l in range(m + 1 - i) :
            print('{0:5.2f}'.format(A[m,l]),end=' ')
        print()
    print()

print('Таблица оценок относительных ошибок (в процентах):')
PrintTriangular(R_rel,1)
print('Таблица эффективных порядков точности:')
PrintTriangular(p_eff,2)

# Отрисовка решения, полученного на сетке с номером S-1
# (отмечаются только узлы, совпадающие с узлами базовой сетки)
style.use('dark_background')

fig1 = figure()
```

```
ax1 = axes(xlim=(-2*10**11,2*10**11), ylim=(-2*10**11,2*10**11))
ax1.set_aspect('equal'); ax1.set_xlabel('x'); ax1.set_ylabel('y');
ax1.plot(0,0,'yo',markersize=15)
ax1.plot(U[S-1,1,:,0],U[S-1,1,:,1],'-w',markersize=5)
ax1.plot(U[S-1,1,M,0],U[S-1,1,M,1], color='w', marker='o', markersize=7)
ax1.set_title('Траектория движения Земли')

# Отрисовка зависимости ошибки от числа интервалов сетки
fig2 = figure()
ax2 = axes()
ax2.plot([r**s*M for s in range(1,S)],[sqrt(sum(R[s,0,:,:]**2)) for s in
range(1,S)],'-wo')
ax2.set_xscale('log'); ax2.set_yscale('log')

# Листинг программы, реализущей приближённое решение системы ОДУ
# (на примереме моделирования движения Земли вокург Солнца)
# с помощью рекурретного сгущения сеток и многократного повышения
# точности по Ричардсону (с вычислением эффективных порядков точности)
```