```
from numpy import *
from matplotlib.pyplot import *

# Функция f возвращает значение правой части решаемого ОДУ
def f(u):
    f = u**2
    return f

# Функция f_u возвращает значение частной производной f_u
def f_u(u):
    f_u = 2*u
    return f_u

def ODESolving(t_0,T,u_0,M_0,alpha,s,r):
    # Функция реализует решение ОДУ на сетке с M интервалами
    # по схеме, определяемой параметром alpha

    # Входные параметры:
    # t_0, T - начальный и конечный моменты счёта
    # u_0 - начальное условие
    # M_0 - число интервалов базовой сетки по времени
    # alpha - параметр определяющий схему (ERK1,KN,DIRK1,CROS1)
    # s - номер сетки, на которой вычисляется решение
    # (если s = 0, то решение ищется на базовой сетке)
    # r - коэффициент сгущения сетки

    # Выходной параметр:
    # u_basic - массив, содержащий сеточные значения
    # решения 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 ОДУ,
    # в котором будут храниться сеточные значения из узлов,
    # совпадающих с узлами базовой сетки
    u_basic = zeros(M_0 + 1)

    # Выделение памяти под массив сеточных значений
    # решения на сгущённой сетке
    u = zeros(M + 1)

    # Задание начального условия
    u[0] = u_0

    # Реализация схемы из семейства ROS1
    # Конкретная схема определяется коэффициентом alpha
    for m in range(M):
        w_1 = f(u[m])/(1 - alpha*tau*f_u(u[m]))
        u[m + 1] = u[m] + tau*w_1.real

    # 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 = 2.; u_0 = 1.

# Определение числа интервалов БАЗОВОЙ сетки,
# на которой будет искаться приближённое решение
M = 50

# Число сеток, на которых ищется приближённое решение
S = 10
# Коэффициент сгущения сеток
r = 2

# Определение параметра схемы (нужный раскомментировать)
alpha = (1 + 1j)/2; p_theor = 2 # CROS1
# alpha = 1; p_theor = 1        # DIRK1
# alpha = 1/2; p_theor = 2      # KN
# alpha = 0; p_theor = 1        # ERK1

# Выделение памяти под массивы сеточных значений
# решений ОДУ на разных сетках с номерами s = 0,...,S-1
# Первый индекс - номер сетки s из последовательности
# сгущающихся сеток, на которых ищется решение
# Второй индекс определяет массив (для фиксированного s),
# в котором хранятся сеточные значения решения из узлов,
# совпадающих с узлами базовой сетки
array_of_u = zeros((S,M + 1))

# "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 = ODESolving(t_0,T,u_0,M,alpha,s,r)
    array_of_u[s,:] = u

# Отрисовка решения, полученного на сетке с номером S-1
# (отмечаются только узлы, совпадающие с узлами базовой сетки)
figure()
t = linspace(t_0,T,M + 1) # Определение базовой сетки
plot(t,array_of_u[S-1,:],'-or',markersize=5,lw=1)
title('График u(t)')
xlabel('t'); ylabel('u')
xlim((t_0,T)); ylim((-30,30))

# Выделение памяти под массив значений эффективных
# порядков точности расчёта приближённого решения
# в каждом узле t_m, 1 <= m <= M (второй индекс массива),
# кроме t_0, так как в нём решение задано точно,
# и на разных сетках (первый индекс массива)
p_eff_ForEveryTime = zeros((S,M + 1));

# Вычисление эффективных порядков точности
for m  in range(1,M + 1):
    # Вычисление p^{eff}_{(0)}(t_m) и p^{eff}_{(1)}(t_m) невозможно
    p_eff_ForEveryTime[0,m] = NaN
    p_eff_ForEveryTime[1,m] = NaN
    for s in range(2,S):
        p_eff_ForEveryTime[s,0] = inf
        p_eff_ForEveryTime[s,m] = log(abs(array_of_u[s-1,m]-array_of_u[s-2,m])\
            /abs(array_of_u[s,m]-array_of_u[s-1,m]))/log(r)

# Отрисовка резульатов расчётов для сетки с номером S
figure()
# Рисуется зависимость теоретического порядка точности p_theor
```

```
# от узла базовой сетки t_m
plot(t,t*0 + p_theor,'-*g',markersize=3)
# Рисуется зависимость эффективного порядка точности
# от узла базовой сетки
plot(t[1:M+1],p_eff_ForEveryTime[S-1,1:M + 1],'-sr',markersize=5,lw=1);
xlim((t_0,T)); ylim((-2.0,3.0))
xlabel('t'); ylabel('p^{eff}')


```

Комментарий к файлу:

Листинг программы, реализующей вычисление эффективных порядков точности для полученного численного решения с целью определения области существования решения задачи Коши.