ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧИ КОШИ

15	7.1.15, 7.2.6, 7.6.6

7.1.15

Задача 7.1. Найти приближенное решение задачи Коши для обыкновенного дифференциального уравнения (ОДУ) 1 порядка

$$y'(t) = f(t, y(t)), \quad t \in [t_0, T], \qquad (1)$$
$$y(t_0) = y_0$$

и оценить погрешность решения задачи.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Задать исходные данные: функцию *f* правой части, начальное значение y_0.
2. Написать функцию **eyler**, реализующую метод Эйлера, и с помощью этой функции найти приближенное решение задачи Коши с шагом *h*=0.1 по явному методу Эйлера.
3. Написать функцию **rkfixed,** реализующую метод Рунге-Кутты, и с ее помощью найти приближенное решение задачи Коши с шагом *h*=0.1 по методу Рунге-Кутты 4 порядка точности.
4. Найти решение задачи Коши аналитически.
5. Построить таблицы значений приближенных и точного решений. На одном чертеже построить графики приближенных и точного решений.
6. Оценить погрешность приближенных решений двумя способами:

a) по формуле $\varepsilon = \max\limits_{0 \le i \le N} |y(t_i) - y_i|$; здесь $y(t_i)$ и y_i - значения точного и приближенного решений в узлах сетки t_i, *i=1,..N*;

b) по правилу Рунге (по правилу двойного пересчета) (см. *ПРИЛОЖЕНИЕ 7.С*).

7. Выяснить, при каком значении шага *h=h** решение, полученное по методу Эйлера, будет иметь такую же погрешность (см. п. 6а), как решение, полученное с помощью метода Рунге-Кутты с шагом *h*=0.1.

УКАЗАНИЕ. В п. 7 рекомендуется провести серию вычислений решения по методу Эйлера, дробя шаг *h* пополам.

N	*f(t,y)*	*t0*	*T*	*y0*
7.1.15	$-2y/t + t^3$	1	2	$-\frac{5}{6}$

```
import numpy as np
import matplotlib.pyplot as plt
```

$$\begin{cases} y'(t) = f(t, y(t)) & , t \in [t_0, T] \\ y(t_0) = y_0 \end{cases}$$

$$\begin{cases} y'(t) = -\frac{2y}{t} + t^3 & , t \in [1, 2] \\ y(1) = -\frac{5}{6} \end{cases}$$

$$y' = -\frac{2y}{t} + t^3$$

1 Решаем ЛОУ

$$y' = -\frac{2y}{t}$$

$$\frac{dy}{y} = -2\frac{dt}{t}$$

$$\ln y = -2\ln t + \ln c$$

$y = c \cdot t^{-2}$ - общее решение однородного

2 ЛНУ решаем методом вариации

$$y = c(t)t^{-2} \Rightarrow y' = c'(t)t^{-2} - 2c(t)t^{-3}$$

$$c't^{-2} - \cancel{2ct^{-3}} + \frac{2}{t}[\cancel{ct^{-2}}] = t^3$$

$$c'(t) = t^5$$

$$c(t) = \int t^5 dt = \frac{t^6}{6} + \tilde{c}$$

$$y(1) = 1^{-2} \cdot \left[\frac{1^6}{6} + \tilde{c}\right] = -\frac{5}{6} = \left[\frac{1}{6} + \tilde{c}\right] = -\frac{5}{6} \Rightarrow \tilde{c} = -1$$

Ответ: $y = \left[\frac{t^6}{6} - 1\right] \cdot t^{-2}$

In[17]:= `DSolveValue[{y'[x] == -2 y[x] / x + x^3 , y[1] == -5 / 6}, y[x], x]`

решение дифференциального уравнения

Out[17]= $\frac{-6 + x^6}{6 x^2}$

```
def f(t, y):
    return -(2*y/t) + t**3


def exact_solution(t):
    return (t**6 - 6) / (6*(t**2))

t0, T = 1, 2
y0 = -5/6
h = 0.1
```

Метод Эйлера

Это одношаговый (самостартующий) явный метод. Функция f(t,y(t)) не зависит от y_n+1

$$y_{n+1} = y_n + hf(t_n, y_n).$$

```
def eyler(f, t0, y0, T, h):
    n = int((T - t0) // h)
    t = np.linspace(t0, T, n + 1)
    y = np.zeros(n + 1)
    y[0] = y0

    for i in range(n):
        y[i + 1] = y[i] + h * f(t[i], y[i])
    return t, y
```

Метод Рунге-Кутта

а при $\alpha = 1/2$ — усовершенствованный метод Эйлера (см. § 14.5).

4. Метод Рунге—Кутты четвертого порядка точности. Наиболее известным из методов Рунге—Кутты является классический 4-этапный метод четвертого порядка точности:

$$y_{n+1} = y_n + hk_n,\ k_n = \frac{1}{6}\left(k_n^{(1)} + 2k_n^{(2)} + 2k_n^{(3)} + k_n^{(4)}\right),$$

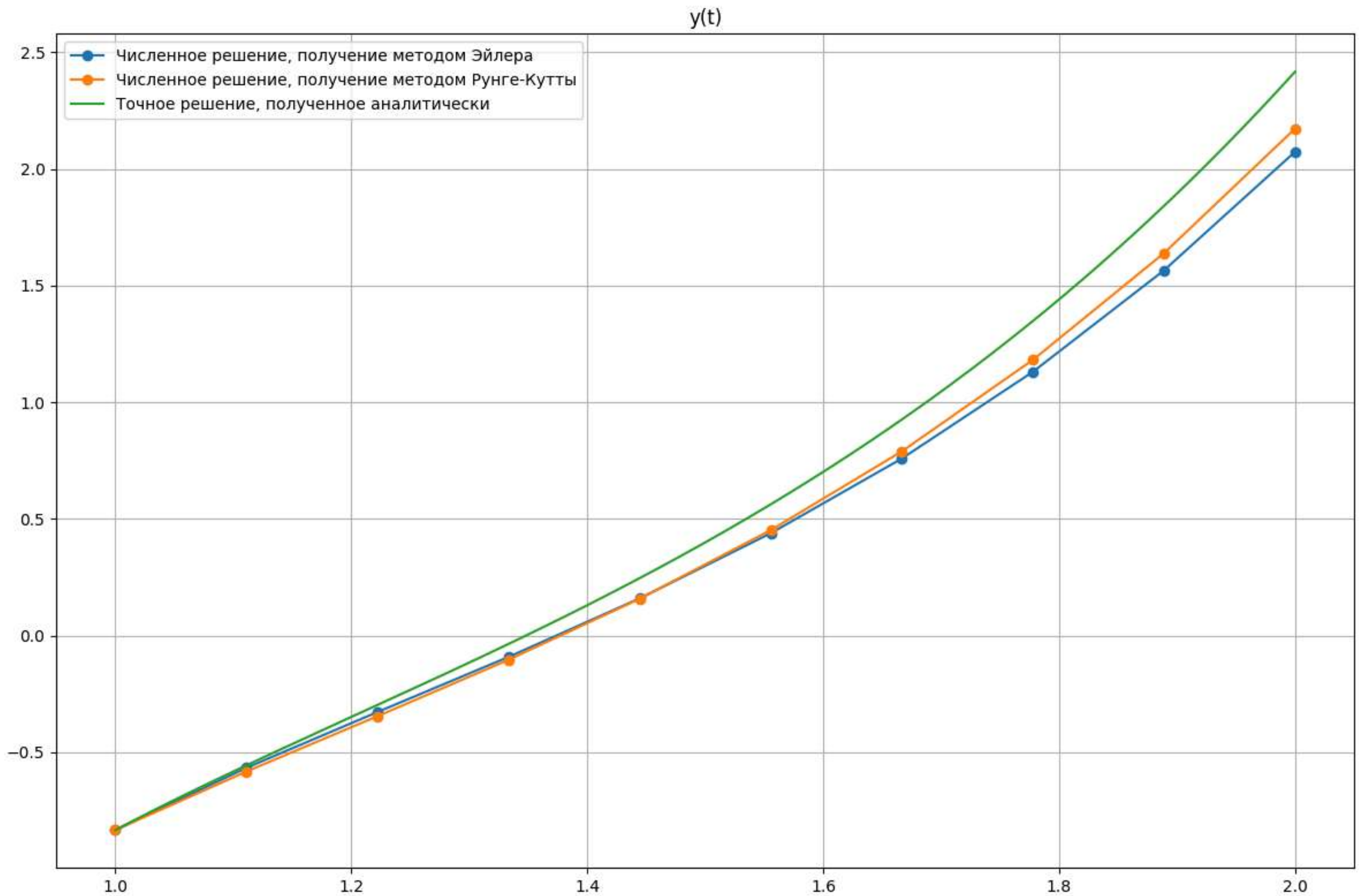
$$k_n^{(1)} = f(t_n, y_n),\ k_n^{(2)} = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_n^{(1)}\right), \quad (14.70)$$

$$k_n^{(3)} = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_n^{(2)}\right),\ k_n^{(4)} = f\left(t_n + h, y_n + hk_n^{(3)}\right).$$

Этот метод весьма прост и, как показывает практика, довольно эффективен в обычных расчетах, когда отрезок $[t_0, T]$ не очень велик и нужна сравнительно невысокая точность.

```
def rkfixed(f, t0, y0, T, h=0.1): # метод Рунге-Кутты 4 порядка точности
  n = int((T - t0) // h)
  t = np.linspace(t0, T, n + 1) # будем делать шаги как раз длины h
  y = np.zeros(n + 1)
  y[0] = y0

  for i in range(n):
      k1 = f(t[i], y[i])
      k2 = f(t[i] + h/2, y[i] + (h/2) * k1)
      k3 = f(t[i] + h/2, y[i] + (h/2) * k2)
      k4 = f(t[i] + h, y[i] + h * k3)
      k = (k1 + 2 * k2 + 2 * k3 + k4) / 6
      y[i + 1] = y[i] + h * k

  return t, y
```

```
t_eyl, y_eyl = eyler(f, t0, y0, T, h)
t_rk, y_rk = rkfixed(f, t0, y0, T, h)
t_accurate = np.linspace(t0, T, 3000) # строю гладкое решение
y_accurate = exact_solution(t_accurate)


plt.figure(figsize=(12, 8))
plt.plot(t_eyl, y_eyl, '-o', label='Численное решение, получение методом Эйлера')
plt.plot(t_rk, y_rk,'-o', label='Численное решение, получение методом Рунге-Кутты')
plt.plot(t_accurate, y_accurate, label='Точное решение, полученное аналитически')
plt.title('y(t)')
plt.tight_layout()
plt.legend()
plt.grid(True)
```



Оценка погрешности приближенных решений 2 способами

а) eps = max|y(t_i) - y(i)| - значения точного и приближенного решений в узлах сетки

б) по правилу Рунге (по правилу двойного пересчета)

```
t_accurate = np.linspace(t0, T, int((T - t0) // h) + 1)
y_accurate = exact_solution(t_accurate) # точное решение
```

```
eyler_eps = max(abs(y_accurate - y_eyl))
rk_eps = max(abs(y_accurate - y_rk))
print("eyler_eps =", eyler_eps)
print("rk_eps =", rk_eps)
```

```
eyler_eps = 0.34316848848800996
rk_eps = 0.24399719899130412
```

Формула для апостериарной оценки локальной погрешности методом Рунге:

$$y_{n+1/2} = y_n + h_{n+1/2}\Phi\,(t_n,\ y_n,\ y_{n+1/2},\ h_{n+1/2}),$$
$$y_{n+1} = y_{n+1/2} + h_{n+1/2}\Phi\,(t_{n+1/2},\ y_{n+1/2},\ y_{n+1},\ h_{n+1/2}).$$

$$y\,(t_{n+1}) - y^{h_{n+1/2}} \approx \frac{y^{h_{n+1/2}} - y^{h_{n+1}}}{2^p - 1}$$

```
t_eyl2, y_eyl2 = eyler(f, t0, y0, T, h/2)
t_rk2, y_rk2 = rkfixed(f, t0, y0, T, h/2)

def b_runge(y_h, y_h2, p):
    return np.max(np.abs(y_h - y_h2[::2]) / (2**p - 1))


runge_eyler = b_runge(y_eyl, y_eyl2, 1)
runge_rk = b_runge(y_rk, y_rk2, 4)


print(f'eyler_eps = {runge_eyler}')
print(f'rk_eps = {runge_rk}')
```

```
eyler_eps = 0.09186994957781591
rk_eps = 0.009543162692673236
```

Вычисление h

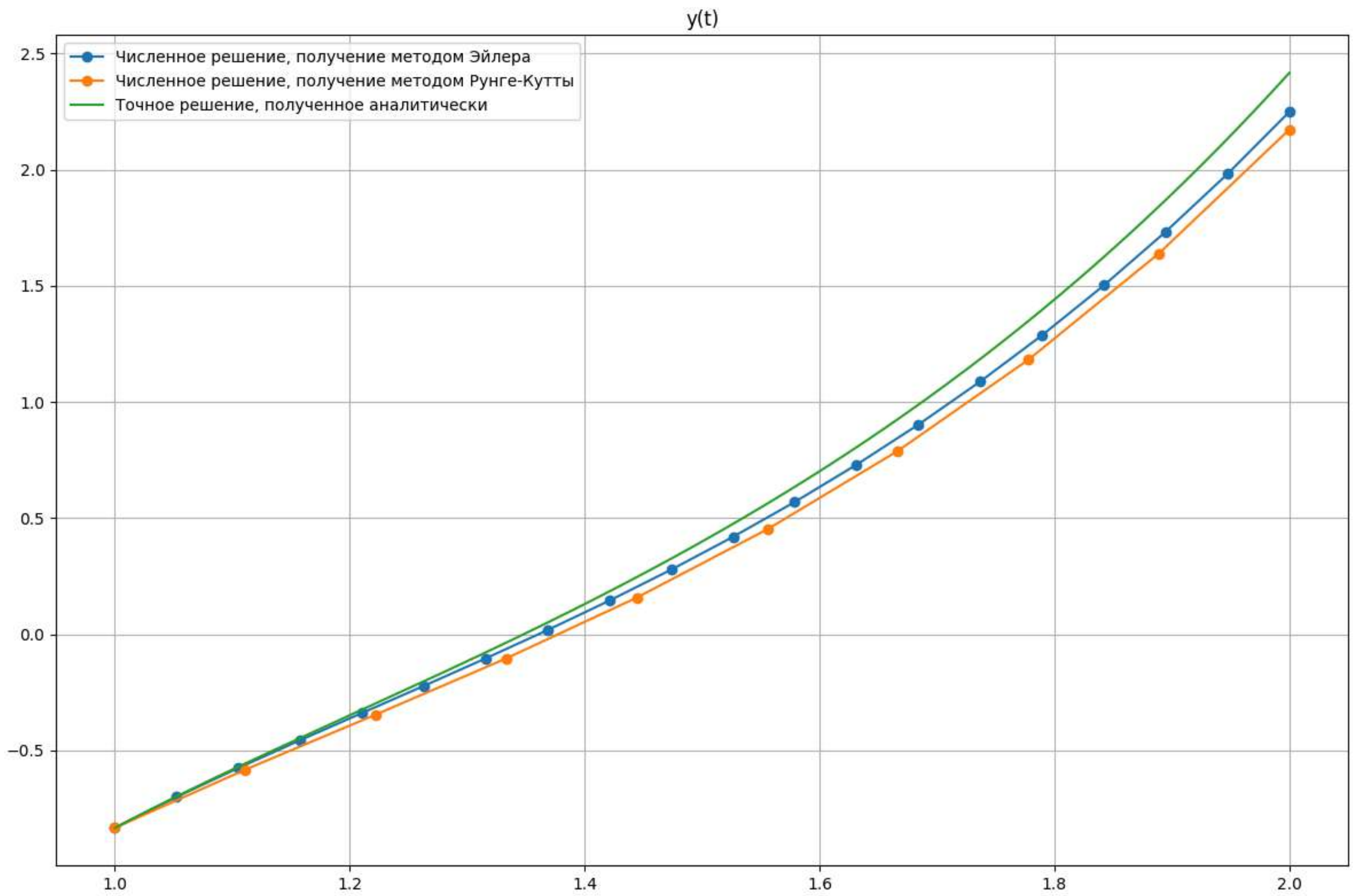
На каком значении шага h решение, полученное по методу Эйлера будет иметь такую же погрешность, как решение, полученное по методу Рунге-Кутты с шагом h = 0.1

Провести серию вычислений по методу Эйлера, дробя h пополам

```
h_star = h
t_eyl, y_eyl = eyler(f, t0, y0, T, h)

def error(t, y):
    return np.max(np.abs(exact_solution(t) - y))

while error(t_eyl, y_eyl) > error(t_rk, y_rk): # abs(error(t_eyl, y_eyl) - error(t_rk, y_rk)) > 0.0001 and h_
    h_star = h_star / 2
    t_eyl, y_eyl = eyler(f, t0, y0, T, h_star)
print(f"h* = {h_star}")
```

```
h* = 0.05
```

```
t_eyl3, y_eyl3 = eyler(f, t0, y0, T, 0.05)
t_rk3, y_rk3 = rkfixed(f, t0, y0, T, h)
t_accurate = np.linspace(t0, T, 3000) # строю гладкое решение
y_accurate = exact_solution(t_accurate) # точное решение

plt.figure(figsize=(12, 8))
plt.plot(t_eyl3, y_eyl3, '-o', label='Численное решение, получение методом Эйлера')
plt.plot(t_rk3, y_rk3,'-o', label='Численное решение, получение методом Рунге-Кутты')
plt.plot(t_accurate, y_accurate, label='Точное решение, полученное аналитически')
plt.title('y(t)')
plt.tight_layout()
plt.legend()
plt.grid(True)
```



Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

7.2.6

Задача 7.2. Задача Коши для ОДУ 2 порядка

$$mx'' + Hx' + kx = f(t), \quad t \in [0, T],$$
$$x(0) = x_0$$
$$x'(0) = v_0$$

описывает движение груза массы *m*, подвешенного к концу пружины. Здесь *x(t)* – смещение груза от положения равновесия, *H* – константа, характеризующая силу сопротивления среды, *k* –коэффициент упругости пружины, *f(t)* – внешняя сила. Начальные условия: x_0 – смещение груза в начальный момент времени t=0, v_0 – скорость груза в начальный момент времени. Промоделировать движение груза на временном отрезке $[0,T]$ при заданных в индивидуальном варианте трех наборах (I, II, III) значений параметров задачи. Для каждого набора по найденной таблице (или графику) решения задачи определить максимальное и минимальное значения
функции *x(t)* и моменты времени, в которые эти значения достигаются. Предложить свой вариант задания параметров, при которых характер колебаний груза существенно отличается от рассмотренного ранее.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Заменить исходную задачу эквивалентной задачей Коши для системы ОДУ 1 порядка:

$$x_1' = x_2$$
$$x_2' = \frac{f(t) - Hx_2 - kx_1}{m} \qquad (2)$$
$$x_1(0) = x_0$$
$$x_2(0) = v_0$$

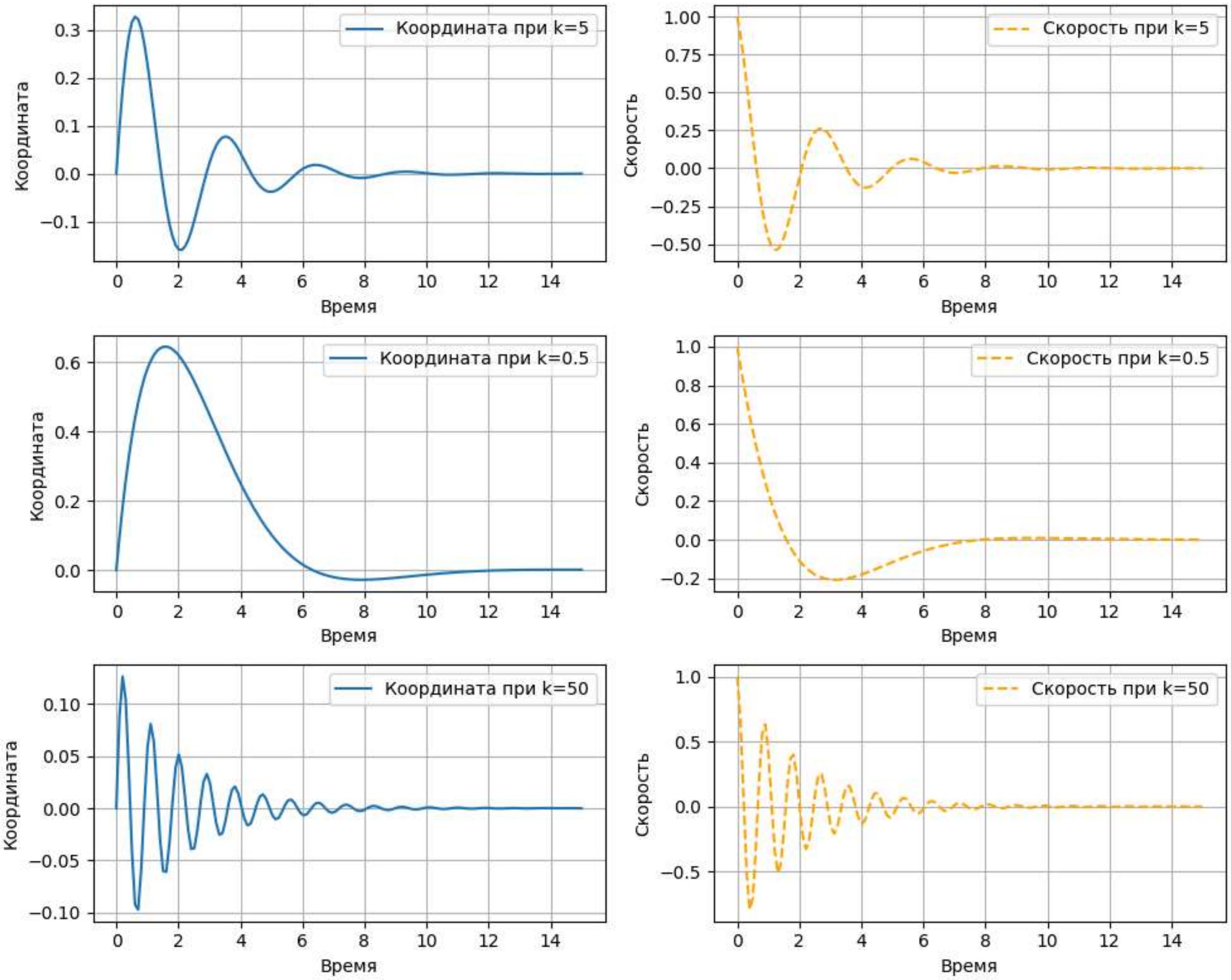
2. Для каждого варианта выбора параметров решить задачу (2) с помощью метода Рунге-Кутты 4 порядка точности с шагом h=0.1.
3. Для каждого варианта выбора параметров построить график найденного решения. Сравнить характер движения груза и дать интерпретацию полученного движения.
4. Для каждого варианта выбора параметров определить требуемые в задаче характеристики.

УКАЗАНИЕ. В п. 2 использовать функцию **rkfixed**, написанную для задачи 7.1.

N		*H*	*k*	*m*	*f(t)*	*x0*	*v0*	*T*
7.2.6	I	1	5	1	0	0	1	15
	II	-“-	0.5	-“-	-“-	-“-	-“-	-“-
	III	-“-	50	-“-	-“-	-“-	-“-	-“-

```
H = 1
m = 1
k = [5, 0.5, 50]
T = 15.0
x0 = 0
v0 = 1
h = 0.1
t0 = 0
y = [x0, v0]
f = 0
```

```
def system(t, y, koef):
    x1, x2 = y
    x1_prime = x2
    x2_prime = (f - H*x2 - koef*x1) / m
    return np.array([x1_prime, x2_prime])

# def f(t):
#     return np.sin(t)

def rkfixed(system, t0, y0, T, h, koef): # метод Рунге-Кутты 4 порядка точности
  n = int((T - t0) // h)
  t = np.linspace(t0, T, n + 1) # будем делать шаги как раз длины h
  y = np.zeros((n+1, len(y0)))
  y[0] = y0

  for i in range(n):
      k1 = system(t[i], y[i], koef)
      k2 = system(t[i] + h/2, y[i] + (h/2) * k1, koef)
      k3 = system(t[i] + h/2, y[i] + (h/2) * k2, koef)
      k4 = system(t[i] + h, y[i] + h * k3, koef)
      k = (k1 + 2 * k2 + 2 * k3 + k4) / 6
      y[i + 1] = y[i] + h * k

  return t, y

plt.figure(figsize=(10, 8))

for i, koef in enumerate(k):
    t, x = rkfixed(system, t0, y, T, h, koef)
    x1, x2 = np.split(x,2,axis=1)
    plt.subplot(len(k), 2, 2*i+1)
    plt.plot(t, x1, label=f'Координата при k={koef}')
    plt.xlabel('Время')
    plt.ylabel('Координата')
    plt.legend()
    plt.grid(True)

    plt.subplot(len(k), 2, 2*i+2)
    plt.plot(t, x2, label=f'Скорость при k={koef}', ls='--', color='orange')
    plt.xlabel('Время')
    plt.ylabel('Скорость')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```



7.6.6 (5)

Задача 7.6. Даны две задачи Коши для систем ОДУ 1 порядка с постоянными коэффициентами на отрезке [0, 1]

$$Y'(t) = AY(t),\ Y(0) = Y_0,$$
$$Z'(t) = BZ(t),\ Z(0) = Z_0,$$

где A и B – заданные матрицы, Y_0, Z_0 - заданные векторы. Выяснить, какая из задач является жесткой.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Составить программу-функцию нахождения решения системы ОДУ 1 порядка с постоянными коэффициентами по явному методу Эйлера. Используя составленную программу, решить обе задачи с шагом h=0.01. Определить, для какой из задач явный метод неустойчив при данном шаге h.
2. Используя встроенную функцию для нахождения собственных чисел матриц A и B, найти коэффициенты жесткости обеих систем. Какая из задач является жесткой?
3. Для жесткой задачи теоретически оценить шаг h^*, при котором явный метод Эйлера будет устойчив (см. *ПРИЛОЖЕНИЕ 7.С*).
4. Составить программу-функцию нахождения решения системы ОДУ 1 порядка с постоянными коэффициентами по неявному методу Эйлера. Используя составленную программу, найти решение жесткой задачи с шагом h=0.01. Построить графики компонент полученного решения.
5. Для жесткой задачи экспериментально подобрать шаг h, при котором графики компонент решения, полученного по явному методу Эйлера, визуально совпадают с графиками компонент решения, полученного по неявному методу с шагом h=0.01. Сравнить найденное значение шага

с шагом h^*. Объяснить различие поведения явного и неявного методов Эйлера при решении жесткой задачи.

N	A		Y_0	B		Z_0
7.6.5	-229.934	301.266	1	-2.018	-0.818	1
	227.624	-303.576	1	-0.082	-1.282	1

```
A_arr = np.array([[-229.934, 301.266], [227.624, -303.576]])
B_arr = np.array([[-2.018, -0.818], [-0.082, -1.282]])
Y0 = np.array([1, 1])
Z0 = np.array([1, 1])
h = 0.01
T = 1
t0 = 0
```

```
def eyler_explicit(A, Y0, T, h):
    n = int(T / h)
    t = np.linspace(0, T, n+1)
    Y = np.zeros((n+1, len(Y0)))
    Y[0] = Y0
    for i in range(n):
        Y[i+1] = Y[i] + h * A.dot(Y[i])
    return t, Y
```

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}). \tag{14.98}$$

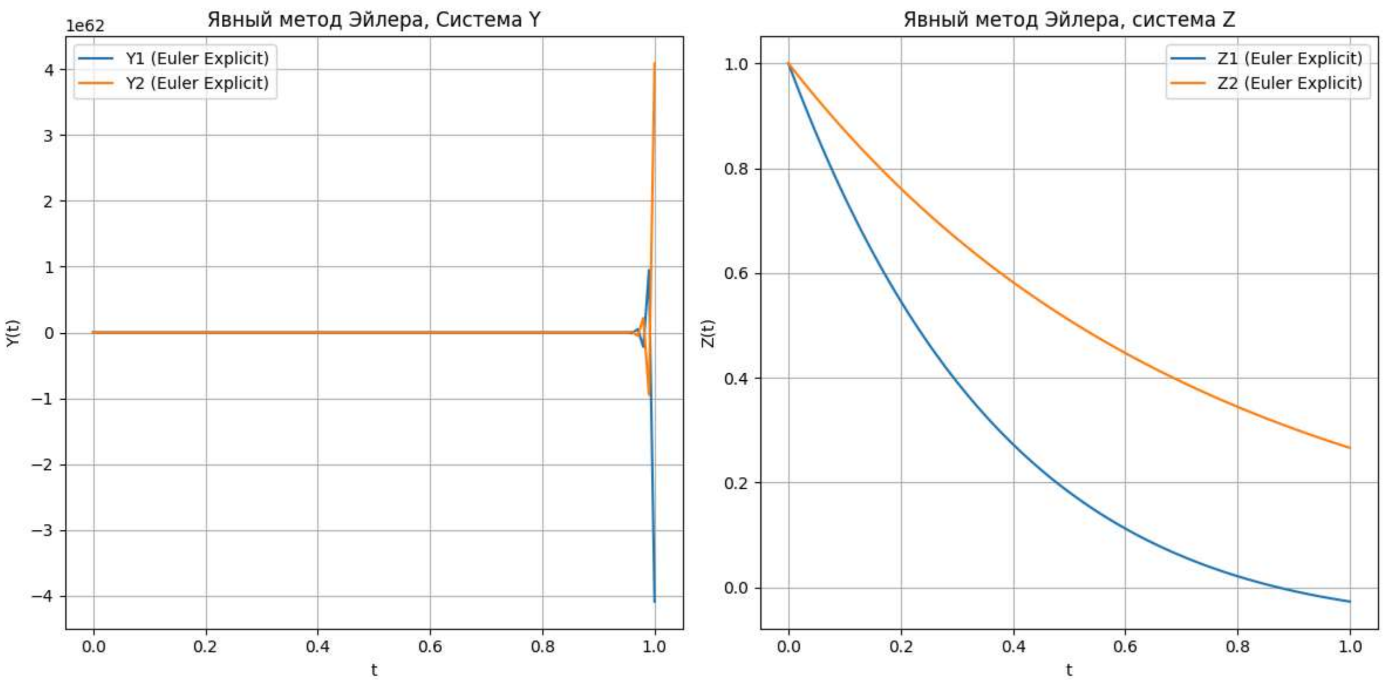
Как нетрудно понять, геометрическая интерпретация одного шага метода (14.98) заключается в том, что решение на отрезке $[t_n, t_{n+1}]$ аппроксимируется касательной $y = y_{n+1} + y'(t_{n+1})(t - t_{n+1})$, проведенной в точке (t_{n+1}, y_{n+1}) к интегральной кривой, проходящей через эту точку (рис. 14.15).

```
def eyler_implicit(A, Y0, T, h):
    n = int(T / h)
    t = np.linspace(0, T, n+1)
    Y = np.zeros((n+1, len(Y0)))
    Y[0] = Y0
    I = np.eye(len(Y0))
    for i in range(n):
        Y[i+1] = np.linalg.solve(I - h * A, Y[i])
    return t, Y
```



```
t_Y, Y_explicit = eyler_explicit(A_arr, Y0, T, h)
t_Z, Z_explicit = eyler_explicit(B_arr, Z0, T, h)
```

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(t_Y, Y_explicit[:, 0], label='Y1 (Euler Explicit)')
plt.plot(t_Y, Y_explicit[:, 1], label='Y2 (Euler Explicit)')
plt.xlabel('t')
plt.ylabel('Y(t)')
plt.legend()
plt.grid()
plt.title('Явный метод Эйлера, Система Y')

plt.subplot(1, 2, 2)
plt.plot(t_Z, Z_explicit[:, 0], label='Z1 (Euler Explicit)')
plt.plot(t_Z, Z_explicit[:, 1], label='Z2 (Euler Explicit)')
plt.xlabel('t')
plt.ylabel('Z(t)')
plt.legend()
plt.grid()
plt.title('Явный метод Эйлера, система Z')

plt.tight_layout()
plt.show()
```



Найти коэффициенты жесткости систем. Какая из систем является жесткой?

Приведем применительно к системе (14.129) одно из определений жесткости. Пусть $\mathrm{Re}\lambda_k < 0$ для всех $k = 1, \ldots, m$. Определим *число жесткости* системы (14.129) с помощью формулы

$$s = \frac{\max\limits_{1 \leqslant k \leqslant n} |\mathrm{Re}\lambda_k|}{\min\limits_{1 \leqslant k \leqslant n} |\mathrm{Re}\lambda_k|}. \tag{14.130}$$

Систему уравнений (14.129) назовем *жесткой*, если для нее $s \gg 1$.

```
def stiffness(arr):
    lyambda = np.linalg.eigvals(arr) # встроенная функция
    re_lyambda = np.real(lyambda)
    s = np.abs(re_lyambda).max() / np.abs(re_lyambda).min()
    return s


print(f'Коэффициент жесткости системы Y = {stiffness(A_arr)}')
print(f'Коэффициент жесткости системы Z = {stiffness(B_arr)}')
```

```
Коэффициент жесткости системы Y = 229.95670995670974
Коэффициент жесткости системы Z = 1.7499999999999998
```

Из разностного уравнения для погрешности, eps_n+1 - (1 + h*l*)eps_n = 0

Следует, что если

|1 + h*l| > 1 - условие не устойчивости явного метода Эйлера

Иначе, неустойчивый

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

$$\left|\frac{1}{1+ah}\right| \leq 1$$

```
def check_stability(A, h):
    eigenvalues = np.linalg.eigvals(A)
    unstable_eigenvalues = [ev for ev in eigenvalues if np.abs(1 + h * ev) > 1]
    is_stable = len(unstable_eigenvalues) == 0
    return is_stable, unstable_eigenvalues

is_stable_A, unstable_eigenvalues_A = check_stability(A_arr, h)
print(f"Система Y с матрицей A {'устойчива' if is_stable_A else 'неустойчива'} при шаге h={h}")

is_stable_B, unstable_eigenvalues_B = check_stability(B_arr, h)
print(f"Система Z с матрицей B {'устойчива' if is_stable_B else 'неустойчива'} при шаге h={h}")
```

```
Система Y с матрицей A неустойчива при шаге h=0.1
Система Z с матрицей B устойчива при шаге h=0.1
```

Будем говорить, что выполнено *корневое условие*, если все корни q_1, q_2, ..., q_r характеристического уравнения лежат внутри или на границе единичного круга комплексной плоскости (т.е удовлетворяют условию $|q_i| \leqslant 1$, $i = 1, 2, ..., r$), причем на границе единичного круга нет кратных корней. Заметим, что в силу равенства (14.30) число $q = 1$ всегда является корнем характеристического уравнения.

Теорема 14.9. *Для того чтобы метод* (14.83) *обладал нуль-*

Система Y является жесткой

Для жесткой задачи теоретически оценить h*, при котором явный метод Эйлера будет устойчив

```
def compute_stable_step(A):
    eigenvalues = np.linalg.eigvals(A)
    max_real_part = np.max(np.abs(np.real(eigenvalues)))
    h_star = 2 / max_real_part
    return h_star

h_star = compute_stable_step(A_arr)
print(f'Явный метод Эйлера будет устойчив при h* = {h_star}')
```

```
Явный метод Эйлера будет устойчив при h* = 0.003765060240963855
```

```
# t_Y, Y_explicit_new = eyler_explicit(A_arr, Z0, T, h = 0.0038)
plt.figure(figsize=(16, 6))

plt.subplot(1,2,1)
t_Y, Y_explicit = eyler_explicit(A_arr, Y0, T, h = 0.002)
plt.plot(t_Y, Y_explicit[:, 0], label='Y1 (Euler Explicit)')
plt.plot(t_Y, Y_explicit[:, 1], label='Y2 (Euler Explicit)')
plt.xlabel('t')
plt.ylabel('Y(t)')
plt.legend()
plt.grid()
```