Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
"Национальный исследовательский университет
"Высшая школа экономики"

Московский институт электроники и математики Национального исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

# ОТЧЕТ
По лабораторной работе №6
на тему
**«ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧИ КОШИ»**

| ФИО студента | Номер группы | Дата | Вариант |
|---|---|---|---|
| Ткаченко Никита Андреевич | БПМ211 | 06.06.2024 | 7.1.21, 7.2.8, 7.6.2 |

Москва – 2024г.

# 7.1 Приближенное решение задачи Коши для ОДУ 1 порядка

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:

а) 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.C*).

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.

7.1.21 $\left| \quad y/t - 2/t^2 \quad \right|^1_1$

Для того чтобы решить данную задачу Коши, мы воспользуемся ее дискретной версией (с которой работают численные методы). Разобьем интервал интегрирования на «сетку» с частотой h=0.1 (получим (t0 — t1) / h точек). Тогда дискретным аналогом уравнения y’ = f(x,t) будет:

$\frac{1}{h}\sum_{j=0}^{k} \alpha_j y_{n+1-j} = \Phi\,(t_n,\ y_{n+1-k},\ \ldots,\ y_n,\ y_{n+1},\ h),$ , где левая часть — разностная аппроксимация производной y’, а правая — некоторая аппроксимация функции f.

Значение в точке yn+1 находится из этого уравнения. При этом могут использоваться значения в k предыдущих точках (k-шаговые методы). Метод Эйлера (явный) является одношаговым, поэтому его запись сводится к ура $\frac{y_{n+1} - y_n}{h} = \Phi\,(t_n,\ y_n,\ y_{n+1},\ h)$. Иначе, *yn+1 = yn + h*f(tn, yn)*

Реализуем метод Эйлера:

```
def euler(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
```

Теперь рассмотрим метод Рунге-Кутты (явный). Если мы еще раз рассмотрим изначальное уравнение yn+1 = yn + f(x,t), то если его проинтегрировать, $y\,(t_{n+1}) = y\,(t_n) + \int\limits_{t_n}^{t_{n+1}} f\,(t,\ y\,(t))\mathrm{d}t.$ Кажется, что если как-то суметь найти интеграл от функци f(x,t), то это и будет является решением шага n+1.

Аппроксимируем этот интеграл квадратурной суммой. Введем k (4 для метода 4-го порядка) доп. точек на интервале [tn, tn+1] (точки 1 и 4 совпадают с границами), и в этих точках найдем

значение производной y (т. е. f(x,t) по определению), т. е. значение коэффициента наклона прямой в этих точках. Серединные точки возьем с большим коэффициентом (2).
Тогда получим формулы:

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

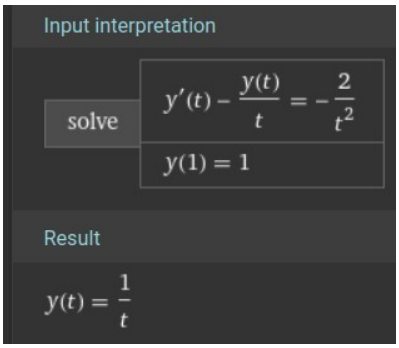
$$k_n^{(1)} = f(t_n, y_n),\ k_n^{(2)} = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_n^{(1)}),$$

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

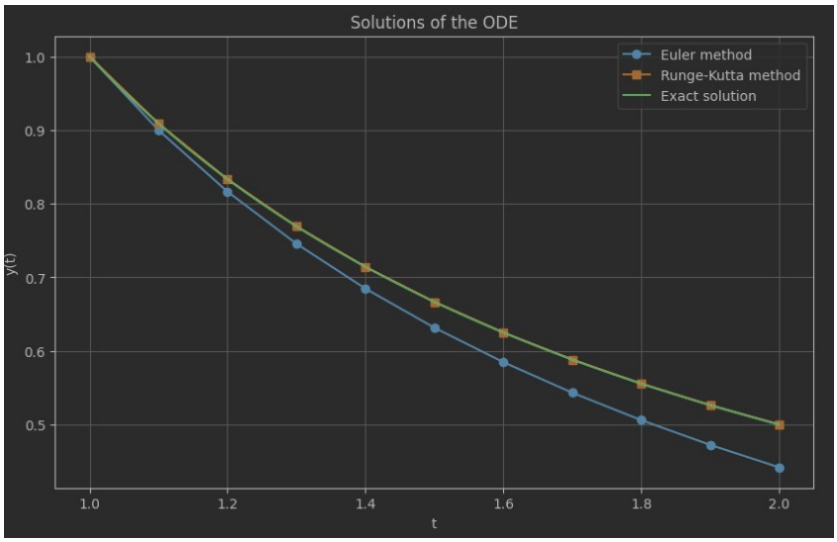
Имплементируем этот метод:

```
def rkfixed(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):
        k1 = f(t[i], y[i])
        k2 = f(t[i] + 0.5*h, y[i] + h*0.5*k1)
        k3 = f(t[i] + 0.5*h, y[i] + h*0.5*k2)
        k4 = f(t[i] + h, y[i] + h*k3)
        y[i+1] = y[i] + h*(k1 + 2*k2 + 2*k3 + k4) / 6
    return t, y
```

Теперь сначала найдем решение данного уравнения аналитически:




А теперь при помощи метода Эйлера и Рунге-Кутта:



Построим таблицы решений:

```
def print_table(t, exact, euler, rk):
    print(f"{'t':>8} {'Exact':>13} {'Euler':>15} {'Runge-Kutta':>19}")
    print("="*60)
    for ti, yi_exact, yi_euler, yi_rk in zip(t, exact, euler, rk):
        print(f"{ti:>10.2f} {yi_exact:>15.10f} {yi_euler:>15.10f} {yi_rk:>15.10f}")

# Display the table
print_table(t_euler, exact_solution(t_euler), y_euler, y_rk)
```

Executed at 2024.06.16 19:05:27 in 4ms

```
 ============================================================
      1.00    1.0000000000    1.0000000000    1.0000000000
      1.10    0.9090909091    0.9000000000    0.9090896300
      1.20    0.8333333333    0.8165289256    0.8333311990
      1.30    0.7692307692    0.7456841139    0.7692280099
      1.40    0.7142857143    0.6847012351    0.7142824615
      1.50    0.6666666667    0.6315676498    0.6666629987
      1.60    0.6250000000    0.5847832709    0.6249959651
      1.70    0.5882352941    0.5432072253    0.5882309231
      1.80    0.5555555556    0.5059564393    0.5555508684
      1.90    0.5263157895    0.4723367353    0.5263107998
      2.00    0.5000000000    0.4417949014    0.4999947169
```

Оценим погрешность решения двумя способами:
1) по формуле e = max|y(ti) — yi|
2) по правилу Рунге np.max(np.abs(y_h - y_h2_interpolated) / (2**p — 1)) (p = 1 для метода Эйлела и 4 для метода Р-К)

```
Max absolute error (Euler): 0.058205098593050564
Max absolute error (Runge-Kutta): 5.283077273432468e-06
```

```
t_euler_h2, y_euler_h2 = euler(f, t0, y0, T, h/2)
t_rk_h2, y_rk_h2 = rkfixed(f, t0, y0, T, h/2)

def runge_error(y_h, y_h2, p):
    # Calculate the error based on the Runge rule
    y_h2_interpolated = y_h2[::2]
    return np.max(np.abs(y_h - y_h2_interpolated) / (2**p - 1))

# Calculate errors using Runge's rule
error_runge_euler = runge_error(y_euler, y_euler_h2, 1)
error_runge_rk = runge_error(y_rk, y_rk_h2, 4)

print(f"Runge error (Euler): {error_runge_euler}")
print(f"Runge error (Runge-Kutta): {error_runge_rk}")
```

Executed at 2024.06.16 19:05:28 in 4ms

```
Runge error (Euler): 0.02905454833370391
Runge error (Runge-Kutta): 3.297372018614316e-07
```

Наконец найдем такой размер шага h*, при котором погрешности метода Эйлера и Рунге-Кутты будут совпадать (деля шаг на 2 каждую итерацию):

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

h_star = h
t_euler, y_euler = euler(f, t0, y0, T, h_star)

while error(t_euler, y_euler) >= error(t_rk, y_rk):
    h_star = h_star / 2
    t_euler, y_euler = euler(f, t0, y0, T, h_star)
print(f"h* = {h_star}")
```

Executed at 2024.06.16 19:05:28 in 237ms

```
h* = 6.103515625e-06
```

# 7.2 Задача Коши для ОДУ 2-го 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.

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

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.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7.2.8 | I<br>II<br>III | 1<br>-“-<br>-“- | 1<br>-“-<br>-“- | 0.5<br>5<br>50 | sin(t)<br>-“-<br>-“- | 0<br>-“-<br>-“- | 0<br>-“-<br>-“- | 20<br>-“-<br>-“- |

Зададим эквивалентную систему ОДУ 1-го порядка, и начальные условия:

```
def system(t, y, m):
    x1, x2 = y
    dx1dt = x2
    dx2dt = (f(t) - H * x2 - k * x1) / m
    return np.array([dx1dt, dx2dt])
```

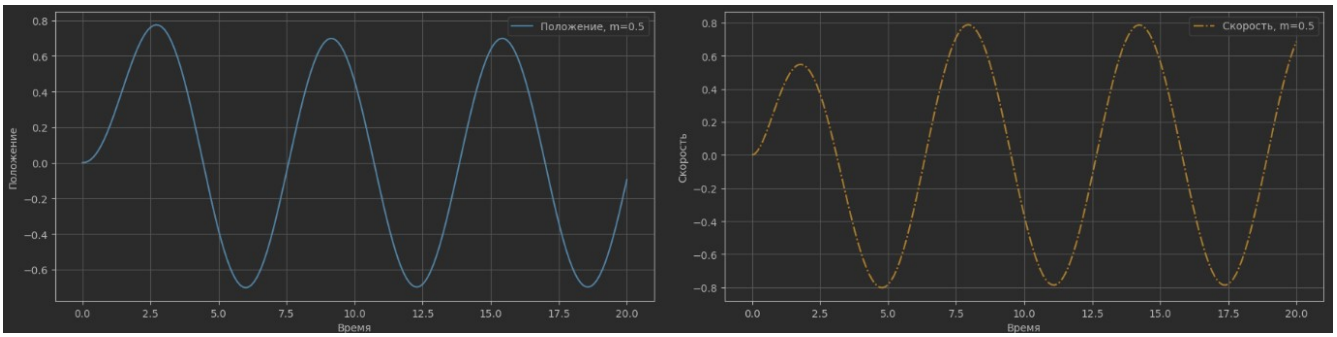
```
# Параметры
H = 1
m = [0.5, 5, 50, 200]
k = 1
T = 20
x0 = 0
v0 = 0
h = 0.1

y0 = [x0, v0]

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

Немного перепишем метод Рунге-Кутты для работы с системой и вариативным параметром:

```
def rkfixed(system, t0, y0, T, h, mass):
    n = int((T - t0) / h)
    t = np.linspace(t0, T, n+1)
    y = np.zeros((n+1, len(y0)))
    y[0] = y0
    for i in range(n):
        k1 = system(t[i], y[i], mass)
        k2 = system(t[i] + 0.5*h, y[i] + h*0.5*k1, mass)
        k3 = system(t[i] + 0.5*h, y[i] + h*0.5*k2, mass)
        k4 = system(t[i] + h, h*y[i] + k3, mass)
        y[i+1] = y[i] + h*(k1 + 2*k2 + 2*k3 + k4) / 6
    return t, y
```

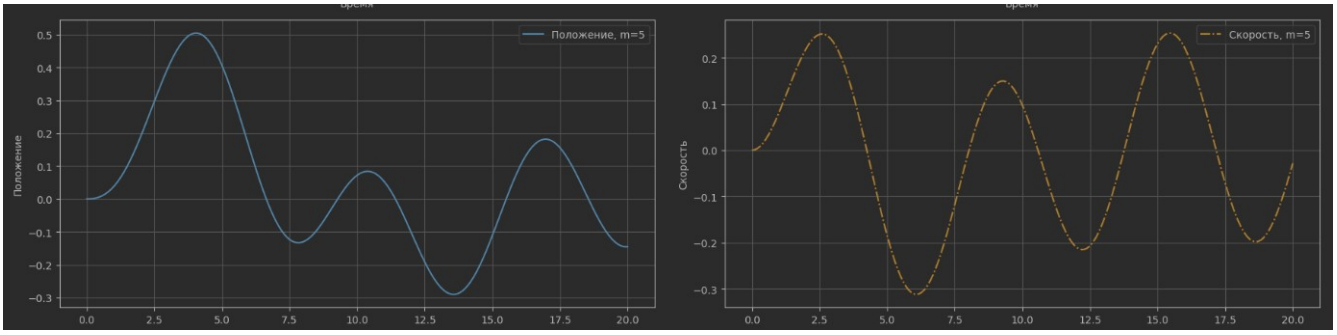
Построим графики решений:

1) mass = 0.5



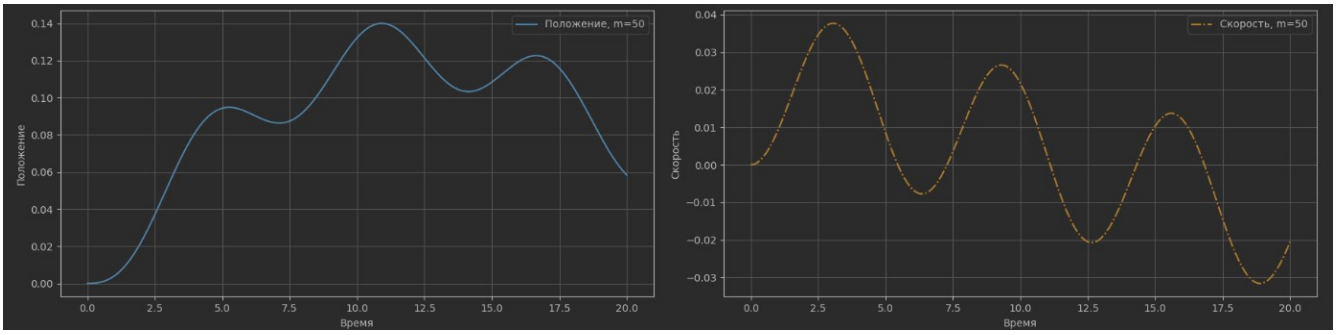

Груз движется колебательно, видно что масса слишком маленькая чтобы противостоять внешней силе sin(t). Минимум: x(6) =-0.7, Максимум: x(2.5) = 0.8

2) mass = 5




Груз все еще движется колебательно, большая масса дала большую общий период, однако видно что влияение внешней силы уменьшилось (теперь она работает как бы замешена внутрь основной амплитуды колебаний).
Минимум: x(13.5) =-0.3, Максимум: x(4) = 0.5

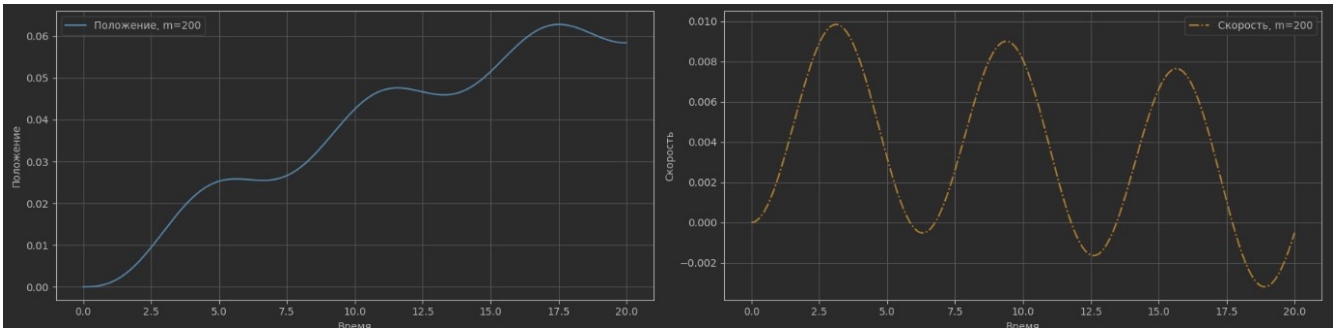
3) mass = 50




Еще большая масса дала еще большую общий период, влияние внешней силы еще меньше
Минимум: x(0) = 0, Максимум: x(10) = 0.14

4) mass = 200, мой вариант

Чтобы еще больше уменьшить эффекти внешней силы, увеличим массу в 4 раза:




Период стал еще больше, и внешняя сила теперь успевает пройти 3 своих периода в черверть основного периода.

Минимум: x(0) = 0, Максимум: x(17.5) = 0.06

# 7.6 Две задачи Коши, жесткость.

**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:

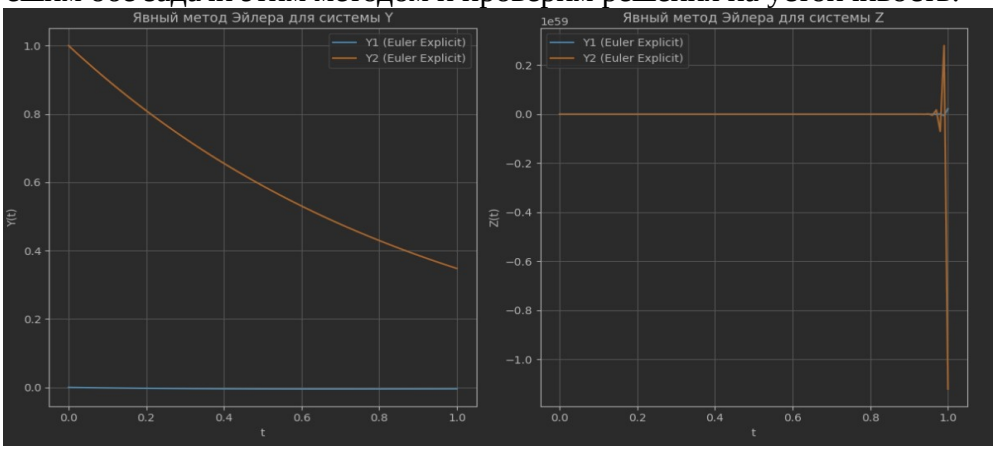
1. Составить программу-функцию нахождения решения системы ОДУ 1 порядка с постоянными коэффициентами по явному методу Эйлера. Используя составленную программу, решить обе задачи с шагом $h$=0.01. Определить, для какой из задач явный метод неустойчив при данном шаге $h$.
2. Используя встроенную функцию 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

| | | | | | | |
|---|---|---|---|---|---|---|
| 7.6.1 | -1.999 | -0.019 | 0 | -10.850 | 9.787 | 1 |
| | -0.063 | -1.051 | 1 | 32.515 | -499.55 | 0 |

Реализуем метод эйлера для системы ОДУ:

```
def euler_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
```

Решим обе задачи этим методом и проверим решения на устойчивость:




Явный метод Эйлера устойчив, если |1 — h*lambda| <= 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
```

*Система Y с матрицей Y_A устойчива при шаге h=0.01*
*Система Z с матрицей Z_A неустойчива при шаге h=0.01*

Определим жесткость систем. Система жесткая, если она содержит собственные числа с существенно различными по величине реальными частями. Жесткость проявляется в том, что в таких системах существуют быстрые и медленные процессы, которые требуют применения очень малых шагов при использовании явных методов, чтобы сохранять численную устойчивость. Аналитическая формула для оценки жесткости системы выглядит так:

$$s = \frac{\max\limits_{1 \leqslant k \leqslant n} |\mathrm{Re}\lambda_k|}{\min\limits_{1 \leqslant k \leqslant n} |\mathrm{Re}\lambda_k|}$$ , тогда система жесткая, если s » 1 (сильно больше).

Проведем расчеты на практике:

```
def analyze_stiffness(A):
    eigenvalues = np.linalg.eig(A)[0]
    print(eigenvalues)
    real_parts = np.abs(np.real(eigenvalues))
    min_real_part = np.min(real_parts)  # Наименьшая реальная часть
    max_real_part = np.max(real_parts)  # Наибольшая реальная часть
    stiffness_ratio = abs(max_real_part) / abs(min_real_part)  # Расчет жесткости
    return stiffness_ratio

print("Жесткость системы Y (матрица A):", analyze_stiffness(Y_A))
print("Жесткость системы Z (матрица B):", analyze_stiffness(Z_A))
```

Executed at 2024.06.17 01:45:53 in 8ms

```
[-2.00026098 -1.04973902]
Жесткость системы Y (матрица A): 1.9054840723352096
[ -10.1997004 -500.2002996]
Жесткость системы Z (матрица B): 49.04068551787617
```

Выходит, что первая система — жесткая (36 » 1). Оценим для нее шаг метода Эйлера, для которого он будет устойчивым:

Предположим, например, что все собственные значения матрицы $A$ отрицательны. Тогда условие (14.96) абсолютной устойчивости метода Эйлера приводит к следующему ограничению на длину шага интегрирования:

$$h \leqslant h_0 = \frac{2}{\max\limits_{1 \leqslant i \leqslant m} |\lambda_i|}. \qquad (14.123)$$

```
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(Y_A)
print(h_star)
```

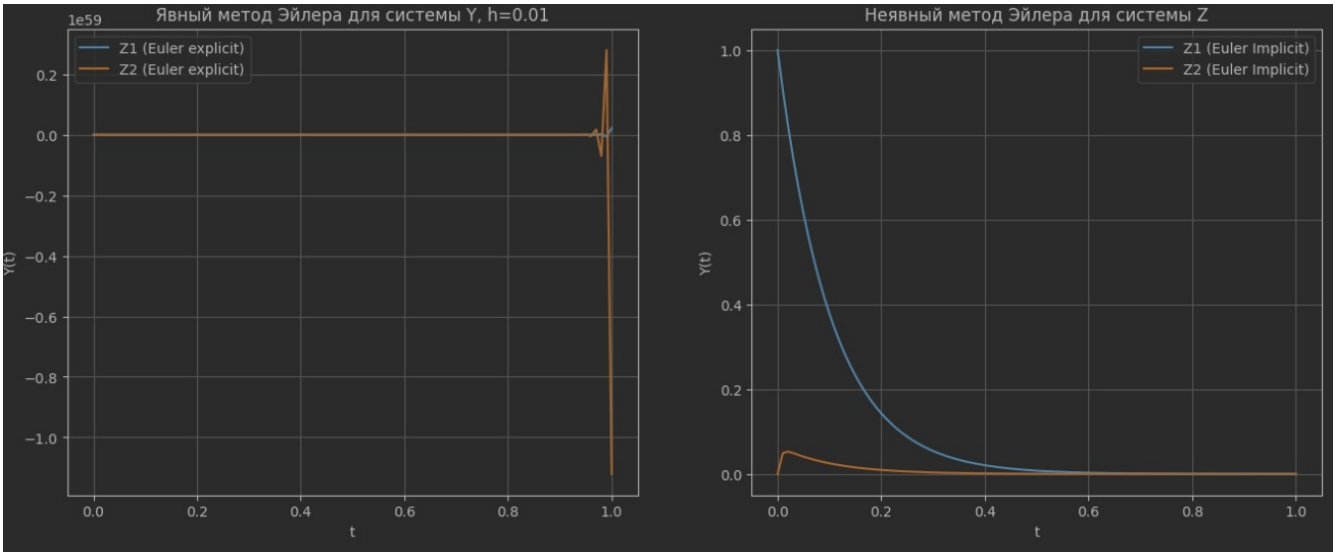
Executed at 2024.06.17 00:40:59 in 4ms

```
0.003822999119397214
```

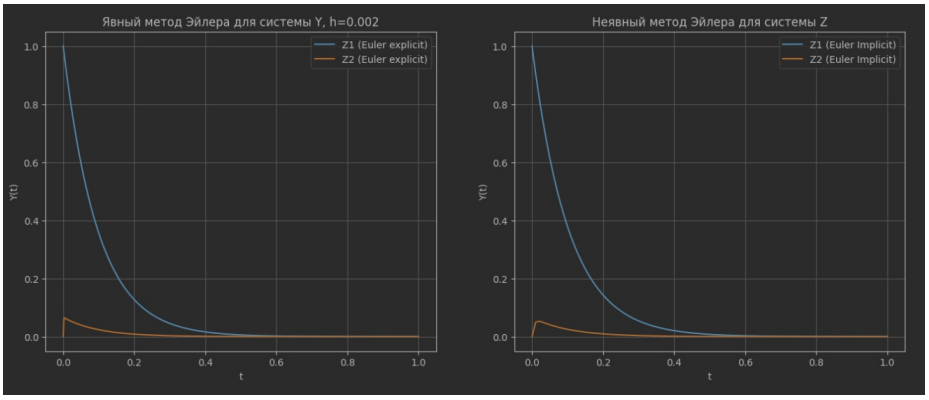
Займемся неявным методом Эйлера. Он отличается от явного тем, что правая часть зависит теперь не от yn, а от yn+1, т. е. у нас есть yn+1 и в левой, и в правой части. Тогда yn+1 находится методами решения СЛАУ (np.solve в моем случае будет достаточно):

```
def euler_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
```

Решим жесткую задачу этим методом:




Подберем коэффициент h* такой, чтобы график решения явным методом Эйлера совпадал с неявным:




h* = 0.002. Этот шаг на порядок меньше чем допустимый шаг для неявного метода Эйлера.

Неявный метод Эйлера лушче справляется с жесткими задачами, потому что оценка устойчивости для него
выглядит вот так: $\left|\frac{1}{1+ah}\right| \leq 1$ . В отличие от оценки явного метода Эйлера: $|1 + h\lambda| \leqslant 1,$

первая оценка лучше справляется с большими собственными значениями системы, потому что 1 + ah ~ ah, и 1 / ah при большом ah « 1.

Неявный метод Эйлера лучше справляется с решением жестких задач, потому что в отличие от критерия устойчивости явного метода: |1 — λh| <= 1