

Задача 1

1. Линейная система. Устойчивость методов.

Задача Коши для системы уравнений

$$\begin{cases} \dot{v} = -\frac{1}{t}((1-t)v + 4w) = 0; \\ \dot{w} = v; \\ w(0) = 1; \\ v(0) = -4. \end{cases}$$

имеет точное решение — полином Лагерра $1 - 4t + 3t^2 + \frac{1}{24}t^4$. Задача решается на отрезке от 0 до 10. Для методов Эйлера (явного, неявного) и методов Рунге-Кутты порядков 2, 3 и 4 найти апостериорный порядок сходимости.

Использовать аналоги норм C и L2.

Систему и начальное условие можно переписать в виде:

$$\begin{pmatrix} \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} 1 - \frac{1}{t} & -\frac{4}{t} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

$$\begin{pmatrix} v \\ w \end{pmatrix}(0) = \begin{pmatrix} -4 \\ 1 \end{pmatrix}$$

Или

$$\begin{aligned} \dot{u} &= A(t) \cdot u \equiv f(t, u) \\ A(t) &= \begin{pmatrix} 1 - \frac{1}{t} & -\frac{4}{t} \\ 1 & 0 \end{pmatrix} \\ u(t) &= \begin{pmatrix} v(t) \\ w(t) \end{pmatrix} \\ u(0) &= \begin{pmatrix} -4 \\ 1 \end{pmatrix} \end{aligned}$$

Точное решение выражается в виде полинома Лагерра:

$$w^*(t) = 1 - 4t + 3t^2 - \frac{2t^3}{3} + \frac{t^4}{24}$$

Требуется решить задачу на отрезке $t \in [0, 10]$, используя методы:

- Явный метод Эйлера
- Неявный метод Эйлера
- Методы Рунге-Кутты 2, 3 и 4 порядков.

Также нужно найти апостериорный порядок сходимости для каждого из этих методов.

Использовать аналоги норм:

- C-норма:

$$\|x\|_C \sim \|x\|_\infty \equiv \|x\|_1 = \max_i |x_i|$$

- и L_2 -норма:

$$\|x\|_{L_2} \sim \|x\|_2 \equiv \|x\|_3 = \sqrt{\sum_i x_i^2}$$

Методы

```
In [1]: import numpy as np
from numpy.linalg import norm, inv
```

```
In [2]: def A_1(t):
    return np.array([[1 - 1 / t, -4 / t], [1, 0]])
```

```
In [3]: def exact_solution(t):
    return 1 - 4 * t + 3 * t**2 - 2/3 * t**3 + t**4 / 24
```

```
In [4]: u_0 = np.array([-4, 1])
```

Явный метод Эйлера

Метод имеет вид:

$$\frac{u^{n+1} - u^n}{\tau} = f(t_n, u^n) \equiv f^n,$$

где τ - величина шага на одной итерации, n - номер итерации, $t_n \equiv t_0 + n\tau$ (рассматриваем равномерную сетку).

В случае нашей задачи

$$\frac{u^{n+1} - u^n}{\tau} = A(t_n) u^n$$

То есть итерационный процесс задается уравнением:

$$\begin{aligned} u^{n+1} &= [E + \tau A(t_n)] u^n \\ u^0 &= u(0) = \begin{pmatrix} -4 \\ 1 \end{pmatrix}, \end{aligned}$$

где E - единичная матрица размера 2x2.

```
In [5]: def explicit_euler(A, t_0, t_max, tau, u_0, epsilon=1e-5):
    """
```

Явный метод Эйлера для многомерных дифференциальных уравнений.

Параметры:

A - матрица $A(t)$, на которую умножается вектор $u(t)$
 t_0 - левая граница отрезка, на котором решается задача Коши
 t_{\max} - правая граница отрезка, на котором решается задача Коши
 τ - величина шага на итерации
 u_0 - вектор начальных условий ($\text{dot}(x)(p), x(p)$) (в этом случае реализовано как значение $u(t_0)$)
 ϵ - малое значение для избежания ошибки деления на ноль при расчете матрицы $A(0)$

Возвращает значения t , $\text{dot}(x)(t)$ и $x(t)$ в узлах сетки.

```
"""
dim = len(u_0)
N = int((t_max - t_0) / tau) # число шагов
# В задаче t_0 == 0, а A(0) не определена
t = [t_0 + epsilon] + [t_0 + n * tau for n in range(1, N + 1)] # сетка
u = np.array([[0.] * dim for _ in range(N + 1)]) # вектор u(t) в узлах сетки
u[0] = u_0 # начальное приближение
for n in range(N):
    u[n + 1] = (np.eye(dim) + tau * A(t[n])) @ u[n]
return t, u[:, 0], u[:, 1]
```

Неявный метод Эйлера

Метод имеет вид:

$$\frac{u^{n+1} - u^n}{\tau} = f(t_{n+1}, u^{n+1}) \equiv f^{n+1},$$

где τ - величина шага на одной итерации, n - номер итерации, $t_n \equiv t_0 + n\tau$ (рассматриваем равномерную сетку).

В случае нашей задачи

$$\frac{u^{n+1} - u^n}{\tau} = A(t_{n+1}) u^{n+1}$$

То есть итерационный процесс задается уравнением:

$$\begin{aligned} u^{n+1} &= [E - \tau A(t_{n+1})]^{-1} u^n \\ u^0 &= u(0) = \begin{pmatrix} -4 \\ 1 \end{pmatrix}, \end{aligned}$$

где E - единичная матрица размера 2×2 , а степень -1 у выражения в квадратных скобках означает обратную матрицу.

```
In [6]: def implicit_euler(A, t_0, t_max, tau, u_0):
    """
    Неявный метод Эйлера для многомерных дифференциальных уравнений.

    Параметры:
    A - матрица A(t), на которую умножается вектор u(t)
    t_0 - левая граница отрезка, на котором решается задача Коши
    t_max - правая граница отрезка, на котором решается задача Коши
    tau - величина шага на итерации
    u_0 - вектор начальных условий (dot(x)(p), x(p)) (в этом случае реализовано как значение u(t_0))

    Возвращает значения t, dot(x)(t), x(t) на сетке.
    """

    dim = len(u_0)
    N = int((t_max - t_0) / tau) # число шагов
    t = [t_0 + n * tau for n in range(N + 1)] # сетка
    u = np.array([[0.] * dim for _ in range(N + 1)]) # вектор u(t) в узлах сетки
    u[0] = u_0 # начальное приближение
    for n in range(N):
        u[n + 1] = inv(np.eye(dim) - tau * A(t[n + 1])) @ u[n] # inv - обратная матрица
    return t, u[:, 0], u[:, 1]
```

Методы Рунге-Кутты

s -стадийным явным методом Рунге-Кутты (ЯМРК) с определяющими коэффициентами

$$\begin{aligned} a_{ij}, \quad 1 \leq i \leq s, \quad 1 \leq j \leq s-1, \quad a_{11} &\equiv 0 \\ c_i, \quad 1 \leq i \leq s, \quad c_1 &\equiv 0 \\ b_i, \quad 1 \leq i \leq s \end{aligned}$$

Называется метод вида

$$\begin{aligned} k_1 &= f(t_n, u^n) \\ k_i &= f(t_n + c_i \tau, u^n + \tau \sum_{j=1}^{i-1} a_{ij} k_j), \quad i > 1 \\ u^{n+1} &= u^n + \tau \sum_{i=1}^s b_i k_i \end{aligned}$$

В нашем случае выражение можно переписать в виде

$$\begin{aligned} k_1 &= A(t_n) \cdot u^n \\ k_i &= A(t_n + c_i \tau) \cdot (u^n + \tau \sum_{j=1}^{i-1} a_{ij} k_j), \quad i > 1 \\ u^{n+1} &= u^n + \tau \sum_{i=1}^s b_i k_i \end{aligned}$$

Удобное представление коэффициентов метода - таблица Бутчера (пустые места \Leftrightarrow нули):

	0				
c_2	a_{21}				
c_3	a_{31}	a_{32}			
\vdots	\vdots	\vdots	\ddots		
c_s	a_{s1}	a_{s2}	\dots	a_{ss-1}	
	b_1	b_2	\dots	b_{s-1}	b_s

Таблицы Бутчера для ЯМРК разных порядков:

- Метод Эйлера с пересчетом (второй порядок точности)

0	0	
1	1	0
	1/2	1/2

- Метод Хойна (третий порядок точности)

$$\begin{array}{c|cc} 0 & 0 \\ \hline 1/3 & 1/3 & 0 \\ 2/3 & 0 & 2/3 \\ \hline & 1/4 & 0 & 3/4 \end{array}$$

- Классический 4-стадийный метод (четвертый порядок точности)

$$\begin{array}{c|cccc} 0 & 0 & & & \\ \hline 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 2/6 & 2/6 & 1/6 \end{array}$$

```
In [7]: a2 = np.array([
    [0, 0],
    [1, 0]
])

a3 = np.array([
    [0, 0, 0],
    [1/3, 0, 0],
    [0, 2/3, 0]
])

a4 = np.array([
    [0, 0, 0, 0],
    [1/2, 0, 0, 0],
    [0, 1/2, 0, 0],
    [0, 0, 1, 0]
])
```

```
In [8]: b2, b3, b4 = np.array([1/2 * 2), np.array([1/4, 0, 3/4]), np.array([1/6, 2/6, 2/6, 1/6])
```

```
In [9]: c2, c3, c4 = np.array([0, 1]), np.array([0, 1/3, 2/3]), np.array([0, 1/2, 1/2, 1])
```

```
In [10]: _2_order = [a2, b2, c2]
_3_order = [a3, b3, c3]
_4_order = [a4, b4, c4]
```

```
In [11]: def runge_kutta(a, b, c, A, t_0, t_max, tau, u_0, epsilon=1e-4):
    """
    Явные методы Рунге-Кутты для многомерных дифференциальных уравнений.

    Параметры:
    a - матрица размера sxs
    b, c - векторы размера sx1
    A - матрица A(t), на которую умножается вектор u(t)
    t_0 - левая граница отрезка, на котором решается задача Коши
    t_max - правая граница отрезка, на котором решается задача Коши
    tau - величина шага на итерации
    u_0 - вектор начальных условий (dot(x)(p), x(p)) (в этом случае реализовано как значение u(t_0))
    epsilon - малое значение для избежания ошибки деления на ноль при расчете матрицы A(0)

    Возвращает значения t, dot(x)(t), x(t) на сетке.
    """
    dim = len(u_0) # размерность системы
    N = int((t_max - t_0) / tau) # число шагов
    # @ задаче t_0 = 0, а A(0) не определена
    t = [t_0 + epsilon] + [t_0 + n * tau for n in range(1, N + 1)] # сетка
    u = np.array([[0.] * dim for _ in range(N + 1)]) # вектор u(t) в узлах сетки
    u[0] = u_0 # начальное приближение
    s = len(b) # число стадий
    for n in range(N):
        k = np.array([[0.] * dim for _ in range(s)])
        for i in range(s):
            k[i] = A(t[n] + c[i] * tau) @ (u[n] + tau * a[i] @ k)
        u[n + 1] = u[n] + tau * b @ k
    return t, u[:, 0], u[:, 1]
```

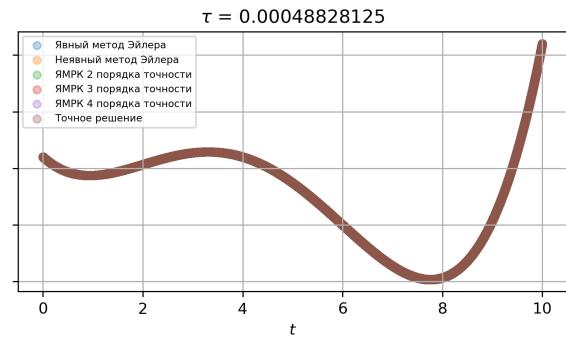
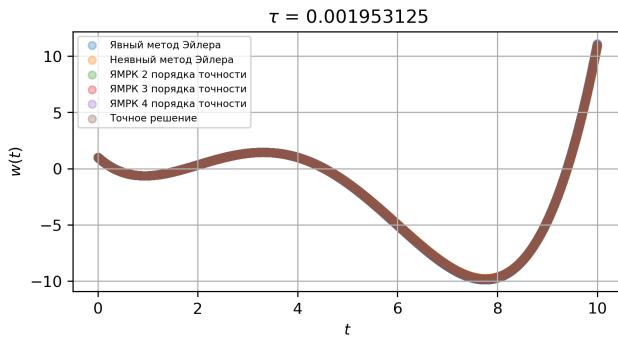
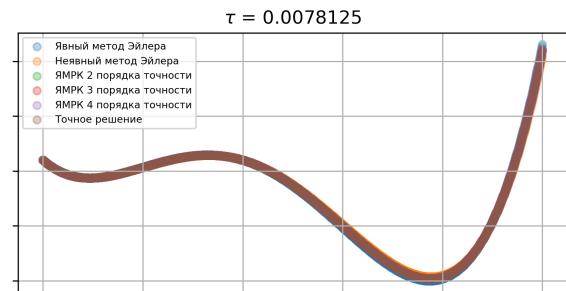
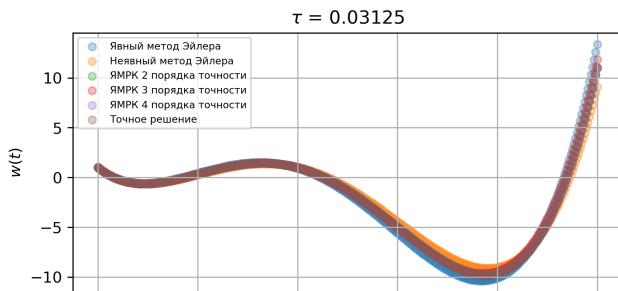
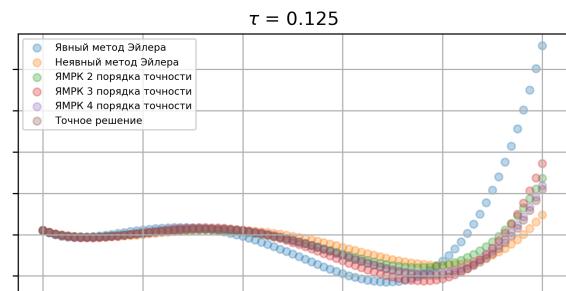
Вычисление решения при помощи описанных методов

```
In [12]: import pandas as pd
import matplotlib.pyplot as plt
```

```
In [13]: step = [2**(-i) for i in range(1, 12, 2)]
```

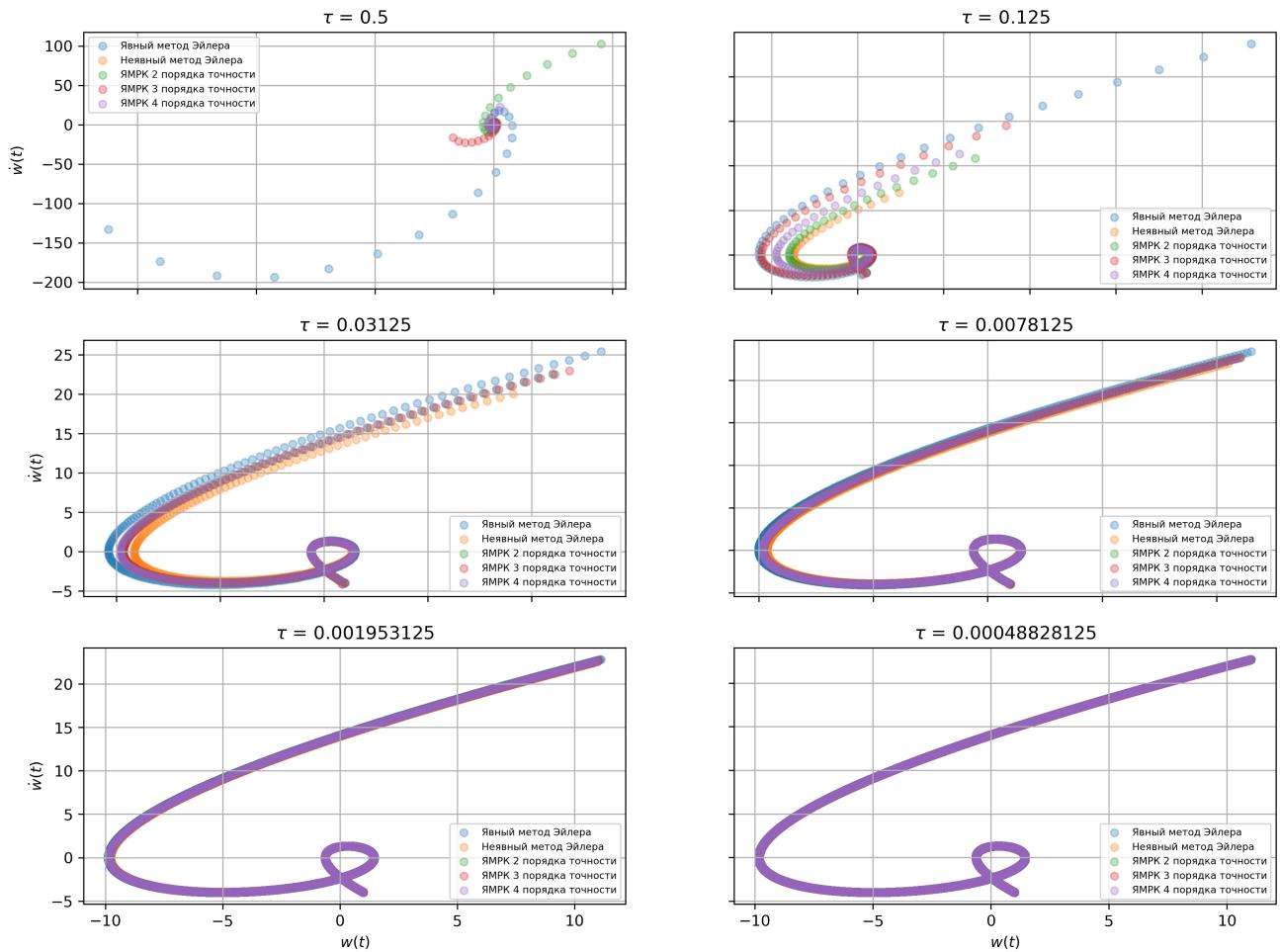
```
In [14]: fig, axes = plt.subplots(3, 2)
fig.set_size_inches(6.4 * 2.2, 4.8 * 2.2)
fig.set_dpi(300)
fig.suptitle(r'Сравнение методов для разных значений величины шага  $\tau$ ')
for i, tau in enumerate(step):
    t, v, w = explicit_euler(A_1, 0, 10, tau, u_0)
    axes[i // 2, i % 2].scatter(t, w, s=25, alpha=0.3, label='Явный метод Эйлера')
    t, v, w = implicit_euler(A_1, 0, 10, tau, u_0)
    axes[i // 2, i % 2].scatter(t, w, s=25, alpha=0.3, label='Неявный метод Эйлера')
    for j, order in enumerate([_2_order, _3_order, _4_order]):
        t, v, w = runge_kutta(*order, A_1, 0, 10, tau, u_0)
        axes[i // 2, i % 2].scatter(t, w, s=25, alpha=0.3, label='ЯМРК {} порядка точности'.format(j + 2))
    axes[i // 2, i % 2].scatter(t, exact_solution(np.array(t)), s=25, alpha=0.3, label='Точное решение')
    axes[i // 2, i % 2].set_title(r'$\tau$ = {}'.format(tau))
    axes[i // 2, i % 2].grid()
    if i == 0:
        axes[i // 2, i % 2].legend(loc='lower left', prop={'size': 6.5})
    else:
        axes[i // 2, i % 2].legend(loc='upper left', prop={'size': 6.5})

for ax in axes.flat:
    ax.set(xlabel=r'$t$', ylabel=r'$w(t)$')
for ax in axes.flat:
    ax.label_outer()
```

Сравнение методов для разных значений величины шага τ 

```
In [15]: fig, axes = plt.subplots(3, 2)
fig.set_size_inches(6.4 * 2.2, 4.8 * 2.2)
fig.set_dpi(300)
fig.suptitle('Сравнение фазовых траекторий для разных значений величины шага $\tau$')
for i, tau in enumerate(step):
    t, v, w = explicit_euler(A_1, 0, 10, tau, u_0)
    axes[i // 2, i % 2].scatter(w, v, s=25, alpha=0.3, label='Явный метод Эйлера')
    t, v, w = implicit_euler(A_1, 0, 10, tau, u_0)
    axes[i // 2, i % 2].scatter(w, v, s=25, alpha=0.3, label='Неявный метод Эйлера')
    for j, order in enumerate([_2_order, _3_order, _4_order]):
        t, v, w = runge_kutta(*order, A_1, 0, 10, tau, u_0)
        axes[i // 2, i % 2].scatter(w, v, s=25, alpha=0.3, label='ЯМРК {} порядка точности'.format(j + 2))
    axes[i // 2, i % 2].set_title(r'$\tau$ = {}'.format(tau))
    axes[i // 2, i % 2].grid()
    if i == 0:
        axes[i // 2, i % 2].legend(loc='upper left', prop={'size': 6.5})
    else:
        axes[i // 2, i % 2].legend(loc='lower right', prop={'size': 6.5})

for ax in axes.flat:
    ax.set(xlabel=r'$w(t)$', ylabel=r'$\dot{w}(t)$')
for ax in axes.flat:
    ax.label_outer()
```

Сравнение фазовых траекторий для разных значений величины шага τ 

Видно, что при большом τ все методы расходятся. При уменьшении величины шага лучше показывают себя методы Рунге-Кутты, методы Эйлера сходятся к точному решению хуже.

Вычисление апостериорных порядков сходимости

Апостериорный порядок сходимости определяется из соотношения

$$\|w_\tau - [w]_\tau\| \leq C\tau^{p+1},$$

где w_τ - значения приближенного решения, $[w]_\tau$ - проекция точного решения на сетку, p - порядок сходимости. Показатель степени больше на 1 ($p + 1$), т.к. мы умножали уравнение на τ при поиске решения.

Отсюда

$$\ln ||w_\tau - [w]_\tau|| \leq \ln C + (p + 1) \ln \tau$$

Построим прямые (случай равенства) и по значению коэффициентов наклона определим порядки сходимости.

```
In [16]: from math import inf
from scipy.optimize import curve_fit
```

```
In [17]: def linear_func(x, k, b):
    return k * x + b
```

```
In [18]: C_norm = pd.DataFrame(
    columns=['Явный метод Эйлера', 'Неявный метод Эйлера'] + ['ЯМРК {} порядка точности'.format(j + 2) for j in range(3)],
    index=[2**(-i) for i in range(1, 12, 2)])
L2_norm = pd.DataFrame(
    columns=['Явный метод Эйлера', 'Неявный метод Эйлера'] + ['ЯМРК {} порядка точности'.format(j + 2) for j in range(3)],
    index=[2**(-i) for i in range(1, 12, 2)])
```

```
In [19]: C_norm.index.name = 'Величина шага'
L2_norm.index.name = 'Величина шага'
```

```
In [20]: for tau in [2**(-i) for i in range(1, 12, 2)]:
    t, v, w = explicit_euler(A_1, 0, 10, tau, u_0)
    C_norm['Явный метод Эйлера'][tau] = np.log(norm(w - exact_solution(np.array(t)), inf))
    L2_norm['Явный метод Эйлера'][tau] = np.log(norm(w - exact_solution(np.array(t)), 2))
    t, v, w = implicit_euler(A_1, 0, 10, tau, u_0)
    C_norm['Неявный метод Эйлера'][tau] = np.log(norm(w - exact_solution(np.array(t)), inf))
    L2_norm['Неявный метод Эйлера'][tau] = np.log(norm(w - exact_solution(np.array(t)), 2))
    for j, order in enumerate([_2_order, _3_order, _4_order]):
        t, v, w = runge_kutta(*order, A_1, 0, 10, tau, u_0)
        C_norm['ЯМРК {} порядка точности'.format(j + 2)][tau] = np.log(norm(w - exact_solution(np.array(t)), inf))
        L2_norm['ЯМРК {} порядка точности'.format(j + 2)][tau] = np.log(norm(w - exact_solution(np.array(t)), 2))
```

```
In [21]: C_norm
```

Out[21]:

	Явный метод Эйлера	Неявный метод Эйлера	ЯМРК 2 порядка точности	ЯМРК 3 порядка точности	ЯМРК 4 порядка точности
Величина шага					
0.500000	6.49202	2.52573	5.1342	4.38148	-0.577071
0.125000	3.54783	1.82066	1.0966	1.83342	-0.198754
0.031250	0.850022	0.645868	-2.24149	-0.193574	-3.57669
0.007812	-0.654957	-0.684042	-5.22777	-3.01842	-6.41399
0.001953	-2.0495	-2.056	-7.8211	-5.99612	-7.82412
0.000488	-3.4371	-3.43869	-7.82412	-7.82412	-7.82412

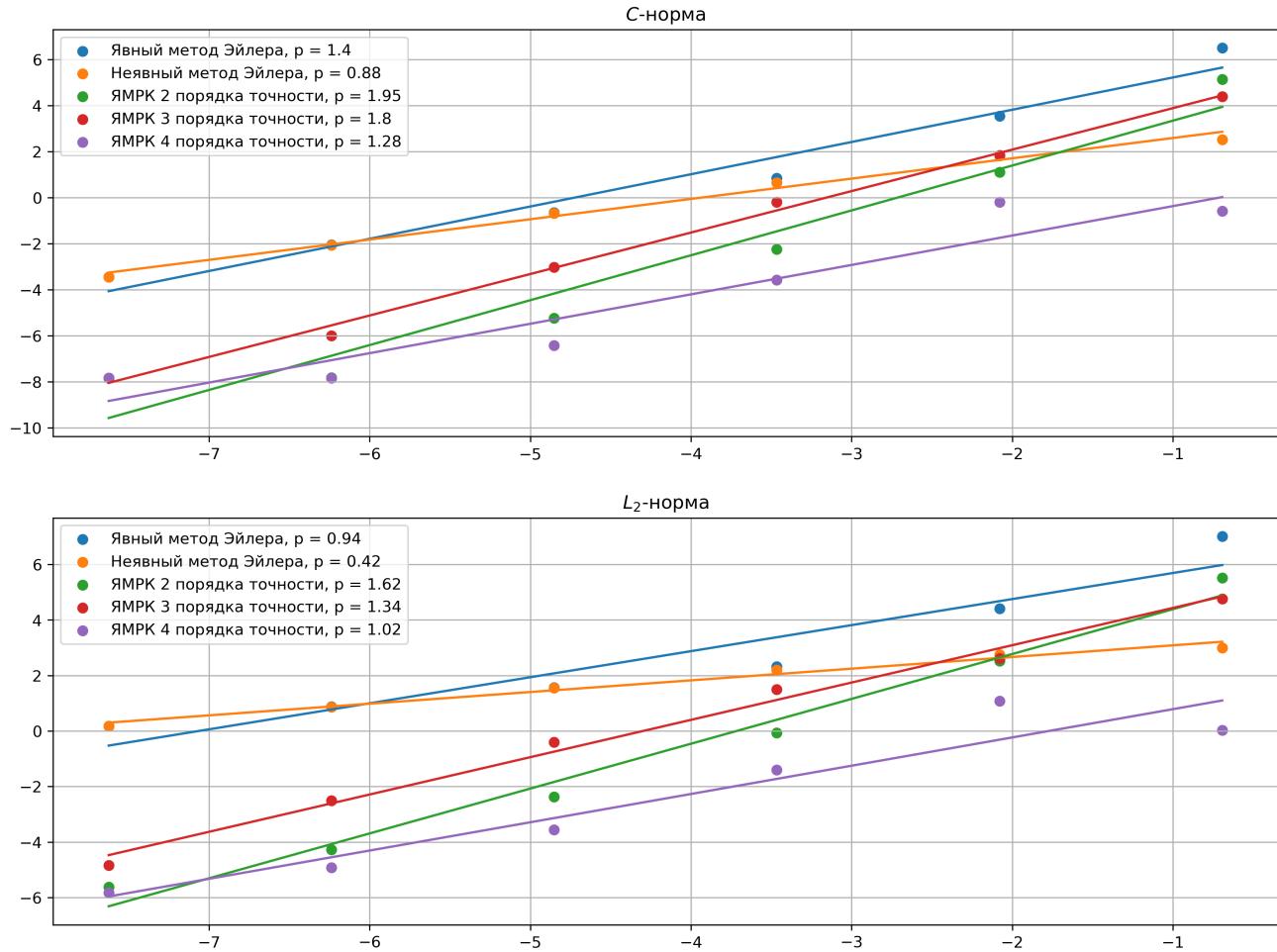
```
In [22]: L2_norm
```

Out[22]:

	Явный метод Эйлера	Неявный метод Эйлера	ЯМРК 2 порядка точности	ЯМРК 3 порядка точности	ЯМРК 4 порядка точности
Величина шага					
0.500000	7.00909	2.99462	5.51317	4.7516	0.0270658
0.125000	4.40484	2.74668	2.52613	2.61908	1.08285
0.031250	2.32195	2.20102	-0.0745467	1.48893	-1.40827
0.007812	1.56313	1.548	-2.37428	-0.402673	-3.55903
0.001953	0.868816	0.865116	-4.27365	-2.51024	-4.92185
0.000488	0.175487	0.174548	-5.62599	-4.83804	-5.82471

```
In [23]: fig, (axC, axL2) = plt.subplots(2)
fig.set_size_inches(6.4 * 2.2, 4.8 * 2.2)
fig.set_dpi(300)
fig.suptitle(r'Сравнение порядков сходимости для норм $C$ и $L_2$')
for column in C_norm.columns:
    params, _ = curve_fit(
        linear_func,
        np.log(C_norm.index),
        C_norm[column]
    )
    axC.scatter(np.log(C_norm.index), C_norm[column], label=column + ', p = {}'.format(round(params[0], 2)))
    axC.plot(np.log(C_norm.index), linear_func(np.log(C_norm.index), *params))
for column in L2_norm.columns:
    params, _ = curve_fit(
        linear_func,
        np.log(L2_norm.index),
        L2_norm[column]
    )
    axL2.scatter(np.log(L2_norm.index), L2_norm[column], label=column + ', p = {}'.format(round(params[0], 2)))
    axL2.plot(np.log(L2_norm.index), linear_func(np.log(L2_norm.index), *params))
axC.set_title(r'$C$-норма')
axL2.set_title(r'$L_2$-норма')
axC.grid()
axL2.grid()
axC.legend(loc='upper left')
axL2.legend(loc='upper left')
```

Out[23]: <matplotlib.legend.Legend at 0x26437c78a30>

Сравнение порядков сходимости для норм C и L_2 

Видим, что порядки апостериорной сходимости слабо коррелируют с порядками аппроксимации методов. Это может быть связано с неудачным начальным условием (для всех методов p колеблется от 1 до 2 при использовании C -нормы).

Задача 2

2. Нелинейная система уравнений. Маятник П.Л. Капицы — маятник с колеблющимся подвесом — описывается следующей нелинейной неавтономной системой ОДУ

$$\dot{x} = -v,$$

$$\dot{v} = \frac{A\omega^2 \cos(\omega t) + g}{l} \sin(x)$$

Здесь l — длина подвеса маятника, A , ω — амплитуда и частота колебаний подвеса.

При переходе к безразмерным переменным запись системы можно упростить:

$$\dot{x} = -v,$$

$$\dot{v} = (1 + a\omega^2 \cos(\omega t)) \sin(x).$$

Все расчеты выполняются для безразмерной постановки задачи.

Численно решить задачу Коши о колебаниях маятника Капицы в случае разных начальных скоростей маятника в зависимости от частоты колебаний подвеса и амплитуды этих колебаний. Рассмотреть значения амплитуды $a = 0.01, 0.05, 0.1, 0.25$, а частота колебаний точки подвеса меняется от 0 до 100. Использовать метод трапеций и методы Рунге-Кутты порядка аппроксимации 2, 3, 4.

Сравнить численные результаты с результатами расчетов по Варианту 19а. Объяснить результаты сравнения.

Условие задачи можно переписать в следующем виде:

$$\begin{aligned} \begin{pmatrix} \dot{v}(t) \\ \dot{x}(t) \end{pmatrix} &= A(t, x) \cdot \begin{pmatrix} v(t) \\ x(t) \end{pmatrix}, \\ A(t, x) &= \begin{pmatrix} 0 & \alpha(t, x) \\ -1 & 0 \end{pmatrix}, \\ \alpha(t, x) &= \frac{\sin(x)}{x} \cdot (1 + a\omega^2 \cos \omega t) \equiv \\ &\equiv \text{sinc}(x) \cdot (1 + a\omega^2 \cos \omega t), \\ \begin{pmatrix} v(0) \\ x(0) \end{pmatrix} &= \begin{pmatrix} v(0) \\ 0 \end{pmatrix} \end{aligned}$$

Или

$$\begin{aligned} \dot{u} &= A(t, x) \cdot u, \\ A(t, x) &= \begin{pmatrix} 0 & \alpha(t, x) \\ -1 & 0 \end{pmatrix}, \\ \alpha(t, x) &= \text{sinc}(x) \cdot (1 + a\omega^2 \cos \omega t), \\ u(t) &= \begin{pmatrix} v(t) \\ x(t) \end{pmatrix}, \\ u(0) &= \begin{pmatrix} v(0) \\ 0 \end{pmatrix} \end{aligned}$$

Нужно решить эту систему следующими методами:

- Метод трапеций
- ЯМРК 2, 3 и 4 порядков точности,

а также исследовать зависимость решения от начальной скорости $v(0)$, параметра частоты $w \in [0, 100]$ и безразмерной амплитуды $a \in \{0.01, 0.05, 0.1, 0.25\}$.

Методы

```
In [24]: def alfa(t, x, w, a):
    return np.sinc(x) * (1 + a * w**2 * np.cos(w * t))
```

```
In [25]: def A_2(t, x, w, a):
    return np.array([[0, alfa(t, x, w, a)], [-1, 0]])
```

Метод трапеций

Общий вид метода:

$$\frac{u^{n+1} - u^n}{\tau} = \frac{f^n + f^{n+1}}{2},$$

где $f^n \equiv f(t_n, u^n)$, $f^{n+1} \equiv f(t_{n+1}, u^{n+1})$, $t_n \equiv t_0 + n\tau$, $u^n \equiv u(t_n)$, $u^0 = u(0)$. Таким образом этот метод получается, если взять среднее арифметическое правых частей явного и неявного методов Эйлера.

Перепишем это уравнение относительно u^{n+1} :

$$u^{n+1} = u^n + \frac{\tau}{2} [f(t_{n+1}, u^{n+1}) + f(t_n, u^n)]$$

Рассмотрим отображение

$$R(w) \equiv w - u^n - \frac{\tau}{2} [f(t^{n+1}, w) + f(t^n, u^n)]$$

Тогда u^{n+1} можно вычислить, решив систему нелинейных уравнений

$$R(u^{n+1}) = 0$$

В нашем случае $R(w)$ можно переписать как

$$R(w) \equiv w - u^n - \frac{\tau}{2} [A(t_{n+1}, w_2) \cdot w + A(t_n, x^n) \cdot u^n],$$

где $w = (w_1, w_2)^T$.

Таким образом на каждой итерации метода придется решать нелинейное уравнение $R(u^{n+1}) = 0$ для поиска u^{n+1} . Это происходит потому, что система уравнений нелинейная, а метод включает в себя неявную часть (нельзя явно выразить u^{n+1} через t_n , t_{n+1} и u^n).

Для решения нелинейной системы воспользуемся функцией `scipy.optimize.fsolve` библиотеки `scipy`.

```
In [26]: def R(A, w, a, t_old, t_new, tau, u_old, u_new):
    return (
        u_new - u_old - tau / 2 *
        (
            A(t_new, u_new[1], w, a) @ u_new +
            A(t_old, u_old[1], w, a) @ u_old
        )
    )
```

```
In [27]: class NonlinearSystem:
    def __init__(self, A, w, a, t_old, t_new, tau, u_old):
        self.A = A
        self.w = w
        self.a = a
        self.t_old = t_old
        self.t_new = t_new
        self.tau = tau
        self.u_old = u_old

    def system_with_params(self, u_new):
        return R(self.A, self.w, self.a, self.t_old, self.t_new, self.tau, self.u_old, u_new)
```

```
In [28]: from scipy.optimize import fsolve
```

```
In [29]: def trapeze_nonlinear(A, t_0, t_max, tau, u_0, w, a):
    N = int((t_max - t_0) / tau) # число шагов
    t = [t_0 + n * tau for n in range(N + 1)] # сетка
    u = np.array([[0.] * len(u_0) for _ in range(N + 1)]) # вектор u(t) в узлах сетки
    u[0] = u_0 # начальное приближение
    for n in range(N):
        system = NonlinearSystem(A, w, a, t[n], t[n + 1], tau, u[n])
        u[n + 1] = fsolve(system.system_with_params, u[n])
    return t, u[:, 0], u[:, 1]
```

Методы Рунге-Кутты

Для ЯМРК такой сложности возникать не будет, так как эти методы явные, и u^{n+1} можно выразить через значения t_n и u^n .

```
In [30]: def runge_kutta_nonlinear(a_matrix, b, c, A, t_0, t_max, tau, u_0, w, a):
    """
    Явные методы Рунге-Кутты для многомерных дифференциальных уравнений.
```

Параметры:

```
a_matrix - матрица размера sxs
b, c - векторы размера sx1
A - матрица A(t), на которую умножается вектор u(t)
t_0 - левая граница отрезка, на котором решается задача Коши
t_max - правая граница отрезка, на котором решается задача Коши
tau - величина шага на итерации
u_0 - вектор начальных условий (dot(x)(p), x(p)) (в этом случае реализовано как значение u(t_0))
```

Возвращает значения t, dot(x)(t), x(t) на сетке.

```
"""
dim = len(u_0) # размерность системы
N = int((t_max - t_0) / tau) # число шагов
t = [t_0 + n * tau for n in range(N + 1)] # сетка
u = np.array([[0.] * dim for _ in range(N + 1)]) # вектор u(t) в узлах сетки
u[0] = u_0 # начальное приближение
s = len(b) # число стадий
for n in range(N):
    k = np.array([[0.] * dim for _ in range(s)])
    for i in range(s):
        k[i] = A(t[n] + c[i] * tau, (u[n] + tau * a_matrix[i] @ k)[1], w, a) @ (u[n] + tau * a_matrix[i] @ k)
    u[n + 1] = u[n] + tau * b @ k
return t, u[:, 0], u[:, 1]
```

Вычисление решения при помощи описанных методов

Рассмотрим системы для ситуаций, когда маятник изначально находился в положении равновесия, т.е.

$$x(0) = 0$$

Для скорости и частоты выберем промежутки [1, 100] с шагом в 25 единиц.

В безразмерном варианте мы положили собственную частоту колебаний маятника $\omega_0 = \sqrt{\frac{g}{l}} = 1$. Поэтому при $\omega = 1$ мы будем наблюдать резонанс.

Одно полное колебание маятник совершает за

$$T = \frac{2\pi}{\omega}$$

Это значение будем использовать в качестве правой границы отрезка времени для моделирования.

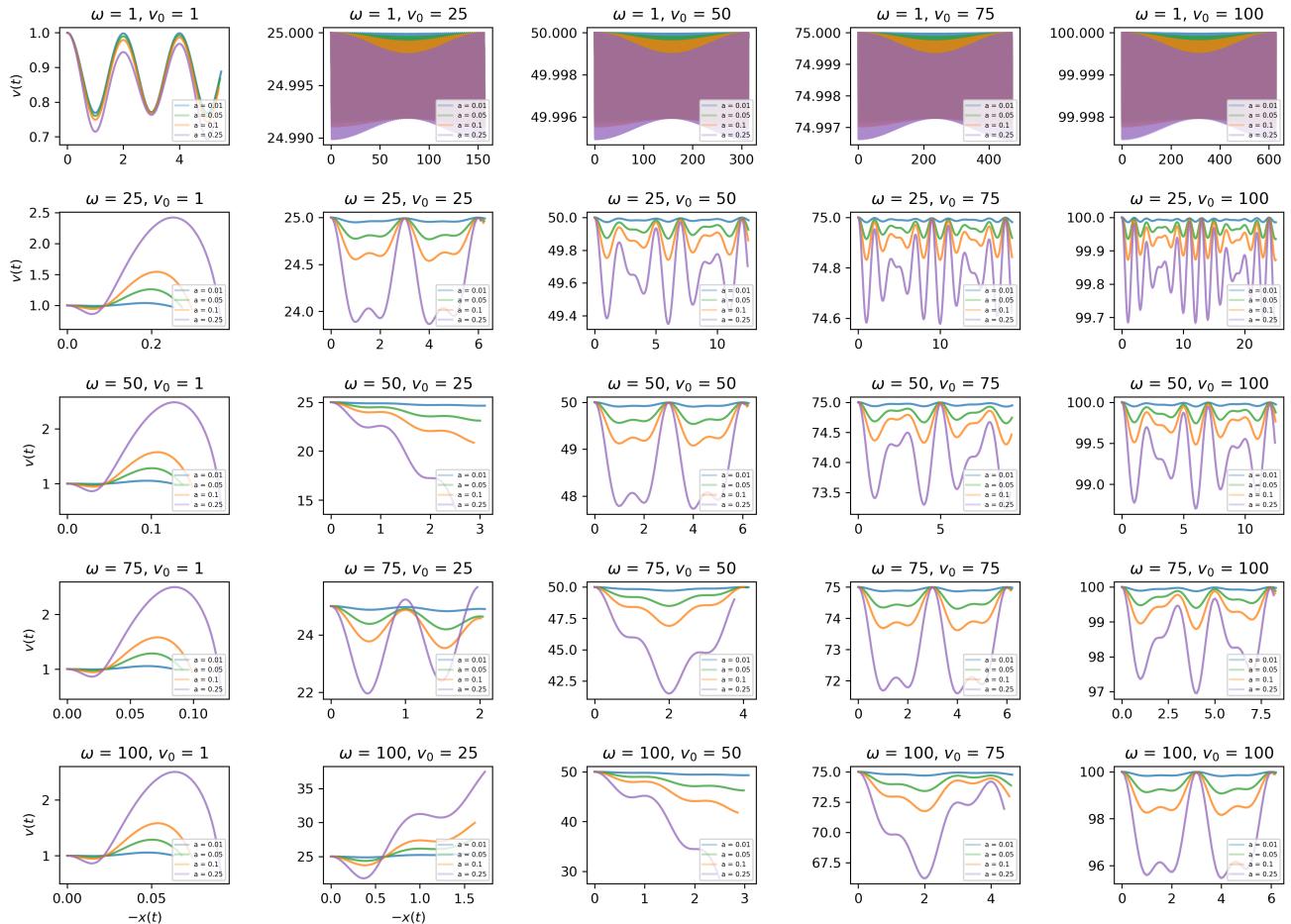
```
In [31]: amplitude = [0.01, 0.05, 0.1, 0.25]
color = ['tab:blue', 'tab:green', 'tab:orange', 'tab:purple']
frequency = [1] + list(range(25, 101, 25))
speed = [1] + list(range(25, 101, 25))
tau = 1e-3 # фиксируем величину шага, при которой сходимость будет хорошей
```

```
In [32]: def u0(v0):
    return np.array([v0, 0])
```

```
In [33]: from itertools import product
```

```
In [34]: def plot_phase_diagrams(amps, clrs, freqs, speeds):
    s, f = len(speeds), len(freqs)
    fig, axes = plt.subplots(s, f)
    fig.set_size_inches(6.4 * 2.2, 4.8 * 2.2)
    fig.set_dpi(300)
    fig.suptitle(r'Фазовые траектории в зависимости от  $\omega$ ,  $a$  и  $v_0$ ')
    fig.tight_layout(pad=3.0)
    i = 0
    for w, v0 in product(freqs, speeds):
        for a, c in zip(amps, clrs):
            t, v, x = trapeze_nonlinear(A_2, 0, 2*np.pi/w, tau, u0(v0), w, a)
            axes[i // f, i % f].plot(-x, v, c, alpha=0.3, label='a = {}'.format(a))
            for order in [_2_order, _3_order, _4_order]:
                t, v, x = runge_kutta_nonlinear(*order, A_2, 0, 2*np.pi/w, tau, u0(v0), w, a)
                axes[i // f, i % f].plot(-x, v, c, alpha=0.3)
            axes[i // f, i % f].set_title(r'$\omega$ = {}, $v_0$ = {}".format(w, v0))
            axes[i // f, i % f].grid()
            axes[i // f, i % f].legend(loc='lower right', prop={'size': 5})
            axes[i // f, i % f].ticklabel_format(useOffset=False)
        i += 1
    for i, ax in enumerate(axes):
        if i == len(axes) - 1:
            ax[0].set(xlabel=r'$-x(t)$', ylabel=r'$v(t)$')
            ax[1].set(xlabel=r'$-x(t)$')
        else:
            ax[0].set(ylabel=r'$v(t)$')
    # for ax in axes.flat:
    #     ax.Label_outer()
```

```
In [35]: plot_phase_diagrams(amplitude, color, frequency, speed)
```

Фазовые траектории в зависимости от ω , a и v_0 

При $\omega = 1$ и $v > 1$ мы действительно наблюдаем резонанс. При увеличении частоты видим, что период картины уменьшается. При малой скорости колебания возникают слабо. При увеличении начальной скорости v_0 начинают возникать колебания (и где-то даже биения). С

ростом амплитуды минимумы на фазовом потрете становятся глубже.

Также видно, что рисунки на диагоналях схожи по структуре.

Задача 3

3. Особые точки и особые траектории. Рассматриваются две близкие системы – модель «хищник-жертва» Лотки – Вольтерры и та же модель, но со слабым самоограничением численности жертв. Обе системы записаны в безразмерном виде с конкретными значениями коэффициентов.

$$\begin{aligned}\dot{x} &= 2x - xy, \\ \dot{y} &= 0.5xy - 0.25y\end{aligned}$$

и

$$\begin{aligned}\dot{x} &= 2x - xy - 10^{-3}x^2. \\ \dot{y} &= 0.5xy - 0.25y\end{aligned}$$

Построить численно характерные траектории вблизи особых точек системы, используя явный метод Эйлера и метод Рунге-Кутты четвертого порядка аппроксимации. Объяснить полученные в численном счете эффекты.

Уравнения из условия можно переписать в виде

$$\begin{aligned}\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} &= \begin{pmatrix} 2 & -x \\ 0.5y & -0.25 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} &= \begin{pmatrix} 2 - 10^{-3}x & -x \\ 0.5y & -0.25 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}\end{aligned}$$

Или эквивалентно

$$\begin{aligned}\dot{u} &= A(u) \cdot u, \\ A(u) &= \begin{pmatrix} 2 & -x \\ 0.5y & -0.25 \end{pmatrix} \\ \dot{u} &= A^*(u) \cdot u, \\ A^*(u) &= \begin{pmatrix} 2 - 10^{-3}x & -x \\ 0.5y & -0.25 \end{pmatrix} \\ u(t) &= \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}\end{aligned}$$

Найдем особые точки системы из условия $\det A = 0$

Решения для системы без квадратичной по x поправки:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \vee \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 \\ 2 \end{pmatrix}$$

Решения для системы с поправкой:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \vee \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1.9995 \end{pmatrix} \vee \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2000 \\ 0 \end{pmatrix}$$

Якобианы моделей будут равны

$$\begin{aligned}J(x, y) &= \begin{pmatrix} 2 - y & -x \\ 0.5y & 0.5x - 0.25 \end{pmatrix} \\ J^*(x, y) &= \begin{pmatrix} 2 - y - 2 \cdot 10^{-3}x & -x \\ 0.5y & 0.5x - 0.25 \end{pmatrix}\end{aligned}$$

В точке $(0, 0)$

$$J(0, 0) = J^*(0, 0) = \begin{pmatrix} 2 & 0 \\ 0 & -0.25 \end{pmatrix}$$

Собственные числа равны соответственно $\lambda_1 = 2$ и $\lambda_2 = -0.25$, $\lambda_1 \cdot \lambda_2 < 0$, значит это особая точка типа "седло".

В точках $(0.5, 2)$ и $(0.5, 1.9995)$

$$J(0.5, 2) = \begin{pmatrix} 0 & -0.5 \\ 1 & 0 \end{pmatrix}$$

$$J^*(0.5, 1.9995) = \begin{pmatrix} 0 & -0.5 \\ 0.99975 & 0 \end{pmatrix}$$

Соответственно собственные числа равны

$$\lambda_{1,2} = \pm \frac{i}{\sqrt{2}}, \quad \lambda_{1,2}^* \approx \pm \frac{i}{\sqrt{2}}$$

Последнее примерное равенство написано с точностью до четвертого знака после запятой. Получается, что эти точки являются особыми точками типа "центр" для соответствующих систем.

Наконец в точке $(2000, 0)$ матрица и якобиан второй системы равны

$$A^*(2000, 0) = \begin{pmatrix} 0 & -2000 \\ 0 & -0.25 \end{pmatrix}$$

$$J^*(2000, 0) = \begin{pmatrix} -2 & -2000 \\ 0 & 999.75 \end{pmatrix}$$

Видно, что $\det A = 0$ и $\exists \lambda_{1,2} \neq 0$. Значит, фазовая картина возле этой точки будет выглядеть как множество параллельных прямых.

Нужно построить фазовые траектории вблизи особых точек, используя методы:

- Явный метод Эйлера
- ЯМРК 4 порядка аппроксимации

```
In [36]: def A_3(u):
    return np.array([[2, -u[0]], [0.5 * u[1], -0.25]])
```

```
In [37]: def A_3_star(u):
    return np.array([[2 - 1e-3 * u[0], -u[0]], [0.5 * u[1], -0.25]])
```

Методы

Явный метод Эйлера

```
In [38]: def explicit_euler_nonlinear(A, N, tau, u_0):
    """
    Явный метод Эйлера для многомерных дифференциальных уравнений.

    Параметры:
    A - матрица A(u), на которую умножается вектор u(t)
    N - число шагов
    tau - величина шага на итерации
    u_0 - вектор начальных условий (x(p), y(p))

    Возвращает значения x(t) и y(t) в узлах сетки.
    """
    u = np.array([[0.] * len(u_0) for _ in range(N + 1)]) # вектор u(t) в узлах сетки
    u[0] = u_0 # начальное приближение
    for n in range(N):
        u[n + 1] = (np.eye(len(u_0)) + tau * A(u[n])) @ u[n]
    return u[:, 0], u[:, 1]
```

Метод Рунге-Кутты 4 порядка аппроксимации

```
In [39]: def runge_kutta_nonlinear(a_matrix, b, c, A, N, tau, u_0):
    """
    Явные методы Рунге-Кутты для многомерных дифференциальных уравнений.

    Параметры:
    a_matrix - матрица размера sxs
    b, c - векторы размера sx1
    A - матрица A(u), на которую умножается вектор u(t)
    N - число итераций
    tau - величина шага на итерации
    u_0 - вектор начальных условий (x(p), y(p))

    Возвращает значения x(t), y(t) на сетке.
    """
    dim = len(u_0) # размерность системы
    u = np.array([[0.] * dim for _ in range(N + 1)]) # вектор u(t) в узлах сетки
    u[0] = u_0 # начальное приближение
    s = len(b) # число стадий
    for n in range(N):
        k = np.array([[0.] * dim for _ in range(s)])
        for i in range(s):
            k[i] = A(u[n] + tau * a_matrix[i] @ k) @ (u[n] + tau * a_matrix[i] @ k)
        u[n + 1] = u[n] + tau * b @ k
    return u[:, 0], u[:, 1]
```

```
In [40]: def initial_point_with_deviation(x0, y0):
    return np.array([x0 + np.random.normal(scale=0.25), y0 + np.random.normal(scale=0.25)])
```

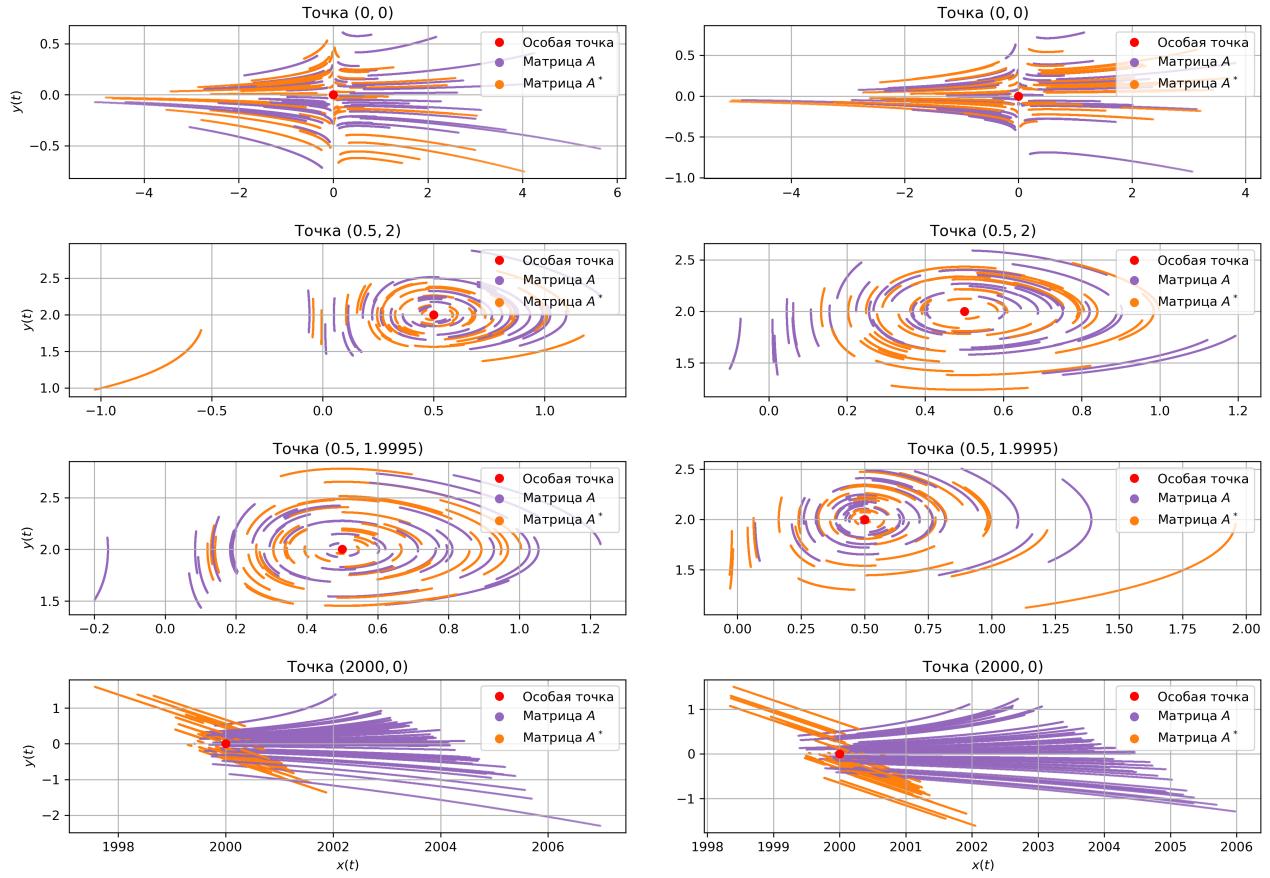
```
In [41]: fig, axes = plt.subplots(4, 2)
fig.set_size_inches(6.4 * 2.2, 4.8 * 2.2)
fig.set_dpi(300)
fig.suptitle('Фазовые траектории системы в окрестности особых точек\пяный метод Эйлера - слева, ЯМРК 4 порядка точности -')
fig.tight_layout(pad=3.0)
for i, (x0, y0) in enumerate([(0, 0), (0.5, 2), (0.5, 1.9995), (2000, 0)]):
    for _ in range(50):
        if (x0, y0) == (2000, 0):
            x, y = explicit_euler_nonlinear(A_3, 1000, 0.000001, initial_point_with_deviation(x0, y0))
            axes[i, 0].scatter(x, y, c='tab:purple', s=0.5, alpha=0.3)
            x, y = explicit_euler_nonlinear(A_3_star, 1000, 0.000001, initial_point_with_deviation(x0, y0))
            axes[i, 0].scatter(x, y, c='tab:orange', s=0.5, alpha=0.3)
        else:
            x, y = explicit_euler_nonlinear(A_3, 1000, 0.001, initial_point_with_deviation(x0, y0))
            axes[i, 0].scatter(x, y, c='tab:purple', s=0.5, alpha=0.3)
            x, y = explicit_euler_nonlinear(A_3_star, 1000, 0.001, initial_point_with_deviation(x0, y0))
            axes[i, 0].scatter(x, y, c='tab:orange', s=0.5, alpha=0.3)

        if (x0, y0) == (2000, 0):
            x, y = runge_kutta_nonlinear(*_4_order, A_3, 1000, 0.000001, initial_point_with_deviation(x0, y0))
            axes[i, 1].scatter(x, y, c='tab:purple', s=0.5, alpha=0.3)
            x, y = runge_kutta_nonlinear(*_4_order, A_3_star, 1000, 0.000001, initial_point_with_deviation(x0, y0))
            axes[i, 1].scatter(x, y, c='tab:orange', s=0.5, alpha=0.3)
        else:
            x, y = runge_kutta_nonlinear(*_4_order, A_3, 1000, 0.001, initial_point_with_deviation(x0, y0))
            axes[i, 1].scatter(x, y, c='tab:purple', s=0.5, alpha=0.3)
            x, y = runge_kutta_nonlinear(*_4_order, A_3_star, 1000, 0.001, initial_point_with_deviation(x0, y0))
            axes[i, 1].scatter(x, y, c='tab:orange', s=0.5, alpha=0.3)

        for j in range(2):
            axes[i, j].plot(x0, y0, 'ro', label='Особая точка')
            axes[i, j].scatter([], [], c='tab:purple', label='Матрица $A$')
            axes[i, j].scatter([], [], c='tab:orange', label='Матрица $A^{**}$')
            axes[i, j].set_title(r'Точка ${\{x_0, y_0\}}$')
            axes[i, j].grid()
            axes[i, j].legend(loc='upper right')

for i, ax in enumerate(axes):
    if i == len(axes) - 1:
        ax[0].set(xlabel=r'$x(t)$', ylabel=r'$y(t)$')
        ax[1].set(xlabel=r'$x(t)$')
    else:
        ax[0].set(ylabel=r'$y(t)$')
# for ax in axes.flat:
#     ax.label_outer()
```

Фазовые траектории системы в окрестности особых точек
Явный метод Эйлера - слева, ЯМРК 4 порядка точности - справа



Как видно, оба метода успешно нашли решения в окрестностях особых точек. Виды фазовых портретов согласуются с теорией.