Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу “Информационные технологии оптимизации и управления”
по теме “Решение задачи Чаплыгина с использованием численных методов оптимизации”

Студент: А. В. Куликов
Преподаватель: Ю. Г. Евтушенко
Группа: М8О-110М-21
Дата:
Оценка:
Подпись:

Москва 2022

1. Постановка задачи

В горизонтальной плоскости Oxy движется самолет с постоянной скоростью v, составляющей угол u с осью Ox. На самолет действует ветер, скорость w которого постоянна и направлена по оси Ox (рис. 1). Задача Чаплыгина состоит в выборе такого управления u (угла курса самолета или же направления курса самолета), при котором самолет, начав движение из точки $A(x_0, y_0)$, облетит за заданное время T фигуру максимальной площади S.

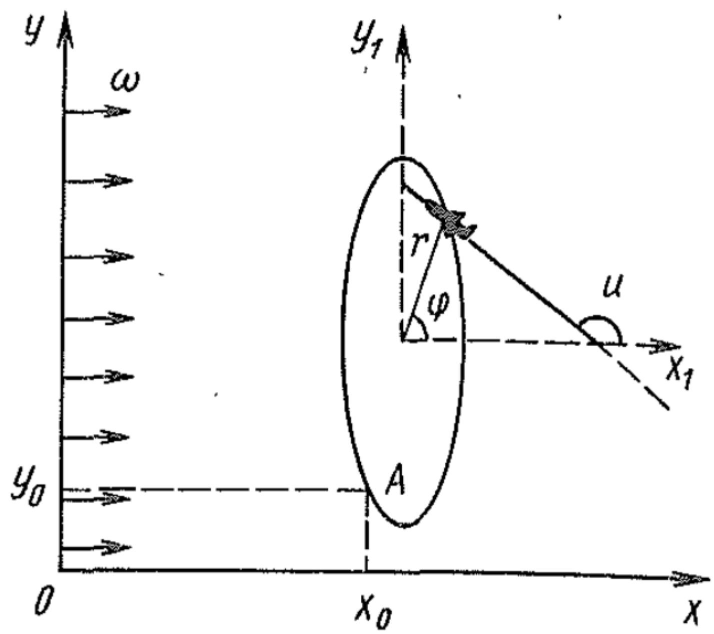

Рис. 1: Задача Чаплыгина

2. Метод решения

Обозначим текущие координаты самолета через $x(t)$ и $y(t)$. Тогда уравнения движения самолета, начальные и конечные условия имеют вид

$$\dot{x} = v \cos u - \omega,\ \dot{y} = v \sin u$$

$$x(0) = x(T) = x_0,\ y(0) = y(T) = y_0$$

Площадь S, облетаемая самолетом, определяется выражением

$$S = \frac{1}{2} \int\limits_0^T \left[x(t)\dot{y}(t) - y(t)\dot{x}(t) \right] \, dt \longrightarrow \max .$$

Значит, в качестве минимизируемого функционала $J(u)$ можно взять

$$J(u) = - \int\limits_0^T \left[xv \sin u - y(v \cos u - \omega) \right] \, dt.$$

Для вычисления функционала необходимо уметь находить функции x, y и вычислять интеграл.

2.1. Решение задачи Коши

Для получения x, y применеятся численный метод Эйлера для решения соответствующих задач Коши. Он состоит в следующем:
Пусть дана задача Коши для уравнения первого порядка:

$$\frac{dy}{dx} = f(x, y), \; y(x_0) = y_0,$$

где функция f определена на некоторой области $D \subset \mathbb{R}^2$. Решение ищется на интервале $(x_0, b]$. На этом интервале введем узлы:

$$x_0 < x_1 < \cdots < x_n \leq b.$$

Приближенное решение в узлах x_i, которое обозначим через y_i, определяется по формуле:

$$y_i = y_{i-1} + (x_i - x_{i-1}) f(x_{i-1}, y_{i-1}), \quad i = 1, 2, 3, \ldots, n.$$

2.2. Численное интегрирование

Интегрирование проводится методом трапеций (см. рис 2). Он состоит в следующем:
Если отрезок $[a, b]$ разбивается узлами интегрирования x_i, $i = 0, 1, \ldots, n$, так что $x_0 = a$ и $x_n = b$, и на каждом из элементарных отрезков $[x_i, x_{i+1}]$ применяется формула трапеций, то суммирование даст формулу для вычисления приближенного значения интеграла

$$\int_a^b f(x)\, dx \approx \sum_{i=0}^{n-1} S_i = \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} (x_{i+1} - x_i) =$$

$$= \frac{f(a)}{2}(x_1 - a) + \sum_{i=1}^{n-1} f(x_i)(x_{i+1} - x_i) + \frac{f(b)}{2}(b - x_{n-1})$$

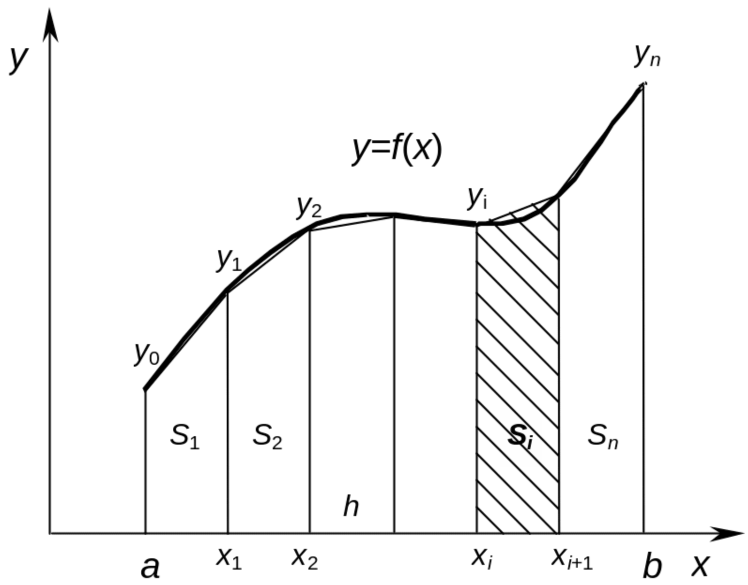

Рис. 2: Метод трапеций

2.3. Метод оптимизации

Оптимизация проводится методом Trust region с ограничениями. Его суть заключается в следующем:

1. Задаем стартовую точку x_0, максимальный радиус доверительной области $\overline{\Delta}$, начальный радиус доверительной области $\Delta_0 \in (0, \overline{\Delta})$ и константу $\eta \in \left[0, \frac{1}{4}\right)$, полагаем $k := 0$.

2. Решаем следующую оптимизационную задачу:

$$p_k = \underset{|p| \le \Delta_k}{\operatorname{argmin}}\, m_k(p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p,$$

   где $f_k = f(x_k)$, $g_k = \Delta f(x_k)$, $B_k = \Delta^2 f(x_k)$, $\Delta_k > 0$ – радиус доверительной области.

3. Вычисляем отношение:

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$$

4. Обновляем текущую точку:

$$x_{k+1} = \begin{cases} x_k + p_k, & \text{если } \rho_k > \eta, \\ x_k, & \text{в противном случае} \end{cases}$$

5. Проверяем критерии завершения алгоритма, например $|x_k - x_{k+1}| < \varepsilon$. Если удовлетворяет критерию, считаем алгоритм завершенным, а x_{k+1} – оптимальным.

6. обновляем радиус доверительной области:

$$\Delta_{k+1} = \begin{cases} \dfrac{1}{4}\Delta_k, & \text{если } \rho_k < \frac{1}{4} \\ \min(2\Delta_k, \overline{\Delta}), & \text{если } \rho_k > \frac{3}{4} \text{ и } |p_k| = \Delta_k, \\ \Delta_k, & \text{в противном случае} \end{cases}$$

7. $k := k + 1$, переход к шагу 2.

3. Результат работы программы

В результате работы программы было подобрано оптимальное управление изображенное на рис. 3.

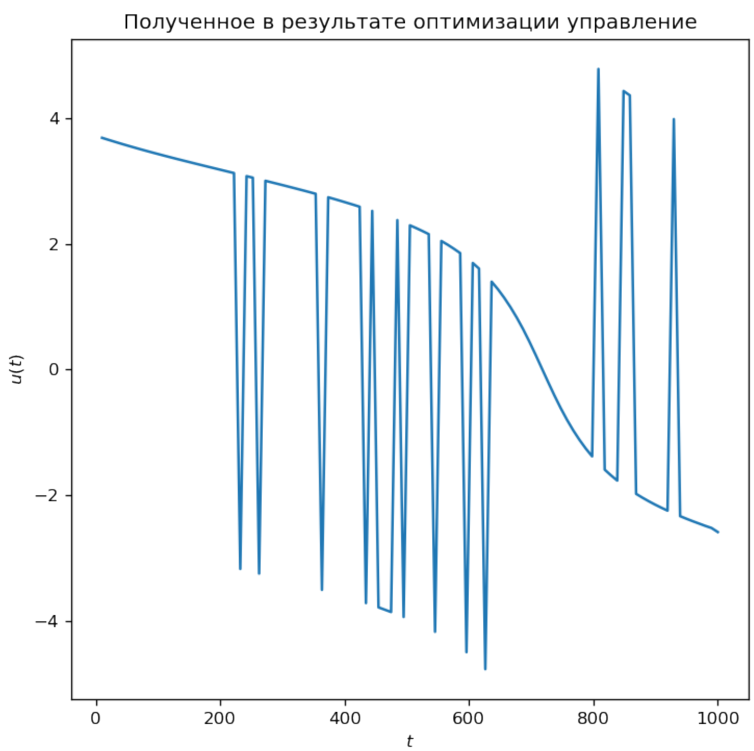

Рис. 3: Полученное в результате оптимизации управление

Видно, что оптимальное управление имеет вид ломанной, но при ближайшем рассмотрении можно заметить, что форма огибающей выбросов вверх и вниз повторяет форму линии, образованной большинством точек. Так происходит потому, что значения управления сошлись к сдвинутым на некоторое число периодов значениям угла, по факту же обозначающим один и тот же угол. Для того чтобы ”сгладить” кривую управления, заменим значения-выбросы, на ближайшие к предыдущему уже сглаженному значению. Т.е. применим следующую процедуру:

1. Положим $u'_1 = u_1$.
2. Для каждого $i = 2..n$, где n – число отсчетов по времени в котором вычисляется управление, положить

$$u'_i = u_i + 2\pi k,\ k = \operatorname*{argmin}_{k\in\mathbb{Z}} |u'_{i-1} - (u_i + 2\pi k)|$$

Т.о. получим управление изображенное на рис. 4.

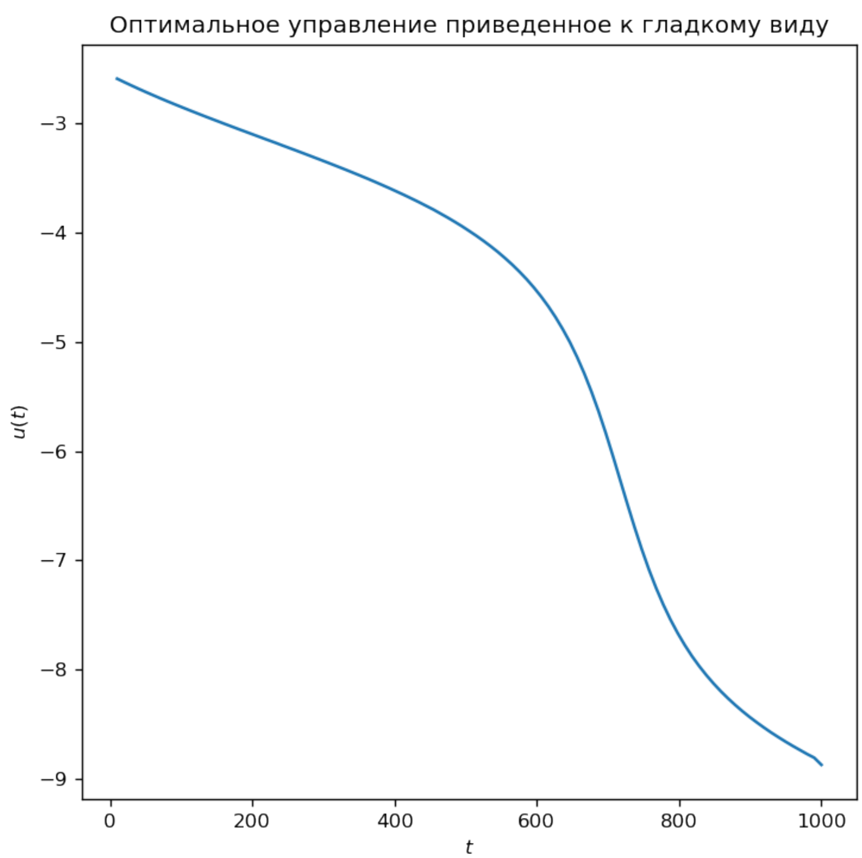

Рис. 4: Оптимальное управление

Полученному управлению соответствует траектория облета изображенная на рис 5.

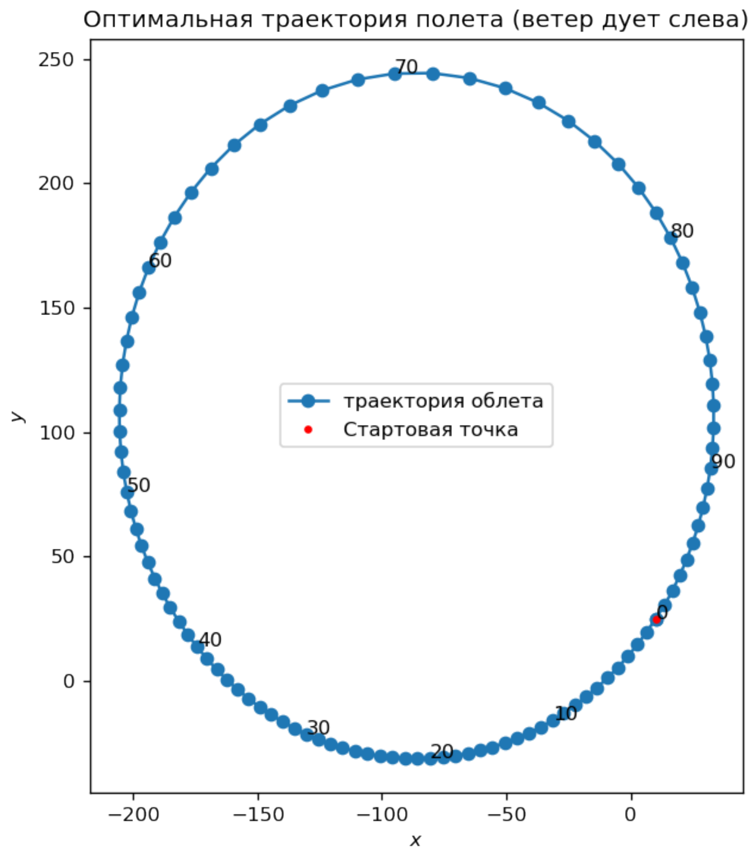

Рис. 5: Оптимальная траектория облета

Из графика видно, что полученная траектория по форме соответствует теоретической (эллипс). Кроме того, можно заметить, что точки снизу расположены плотнее, а сверху более разреженно. Это объясняется тем, что каждая точка соответствует отсчету по времени, а расстояние между соседними точками соответствует расстоянию, которое самолет преодолевает за один шаг по времени. Точки на графики пронумерованы в порядке облета с шагом в 10 отсчетов. Значит самолет стартует из начальной точки летя вниз-влево, а ветер как раз дует слева направо, значит самолет летит против ветра, и лететь ему "труднее", т.о. за отсчет по времени он преодолевает меньшее расстояние, чем, например, когда летит по ветру в верхней части троектории. Поэтому точки снизу расположены значительно гуще, чем точки сверху. На участках же пути, где скорость самолета направлена перпендикулярно ветру (в левой и правой части траектории) точки расположены с одинаковой плотностью вне зависимости от направления полета.

Т.к. начальное управление задается случайным, то результат оптимизации тоже в значительной степени случаен. Хотя оптимальное управление и удовлетворяет всем условиям задачи, условий не достаточно для единственного решения, поэтому правильным решением можно считать целое семейство кривых. Несколько из возможных траекторий представлено на рис 6.

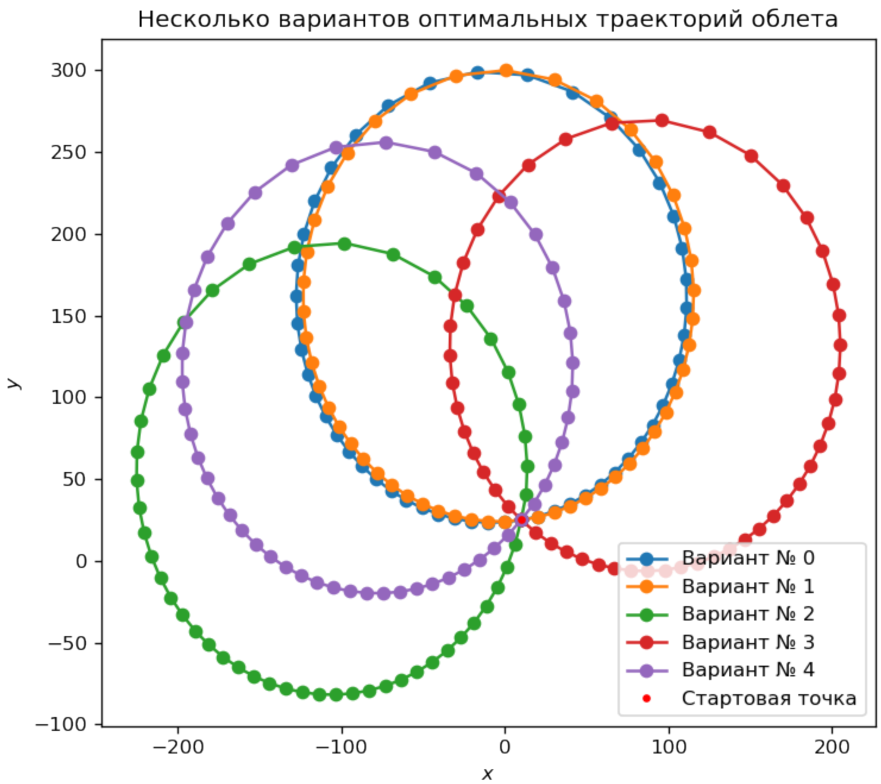

Рис. 6: Оптимальная траектория облета

Видно, что все траектории имеют одну общую точку – стартовую, при этом самолет из нее может вылетать в совершенно любом направлении, форма траектории от этого не изменится.

Вывод

В ходе работы над данной курсовой работой была реализована программа, позволяющая численно решать задачу Чаплагина. С помощью разработанной программы произведены расчеты. Полученные результаты удовлетворяют физическому смыслу задачи.

Список источников

1. Trust region methods. Conn, A. R., Gould, N. I., Toint, P. L.
2. Статья «SciPy, оптимизация с условиями»: `https://habr.com/ru/company/ods/blog/448054/`
3. Статья «Метод Эйлера»: `https://ru.wikipedia.org/wiki/Метод_Эйлера`
4. Статья «Метод трапеций»: `https://ru.wikipedia.org/wiki/Метод_Трапеций`

Приложение

Листинг кода

chaplygin_problem.py

```python
import numpy as np
np.random.seed(234)

from scipy.optimize import minimize, NonlinearConstraint
import matplotlib.pyplot as plt

# численное интергрирование методом трапеций
def integrate(function_values, points):
    h = points[1] - points[0]
    return np.sum(function_values) * h  - 0.5 * h *
     ↪  (function_values[0] + function_values[-1])

# Класс, реализующий решение задачи Чаплыгина
class ChaplyginProblemSolver:
    def __init__(self, v, w, T, x0, y0, use_cache=True):
        self.v = v
        self.w = w
        self.T = T
        self.x0 = x0
        self.y0 = y0
        self.use_cache = use_cache

        if use_cache:
            self.__de_solution_cache = dict()

            self.cached_calls = 0
            self.euler_method_calls = 0

    # правая часть дифференциального уравнения производной x по
     ↪  времени
    def x_right_part(self, u, i):
        return self.v * np.cos(u[i]) - self.w

    # правая часть дифференциального уравнения производной y по
     ↪  времени
    def y_right_part(self, u, i):
        return self.v * np.sin(u[i])

```

```python
    # решение задачи Коши методом Эйлера
    def euler_method(self, u, t):
        x = np.zeros(t.shape)
        y = np.zeros(t.shape)

        n_nodes = t.shape[0]

        x[0] = self.x0
        y[0] = self.y0

        for i in range(1, n_nodes):
            x[i] = x[i-1] + (t[i] - t[i-1]) * self.x_right_part(u, i)
            y[i] = y[i-1] + (t[i] - t[i-1]) * self.y_right_part(u, i)

        return x, y

    # решение задачи Коши с кешированием результата, чтобы не решать уравнение несколько раз при проверке на ограничения
    def solve_ivp(self, u, t):
        if self.use_cache:
            key = hash(u.data.tobytes())
            if key in self.__de_solution_cache:
                self.cached_calls += 1
                return self.__de_solution_cache[key]

            result = self.euler_method(u, t)
            self.__de_solution_cache[key] = result
            self.euler_method_calls += 1
            return result
        else:
            return self.euler_method(u, t)

    # минимизируемый функционал
    def minimization_func(self, u):
        t = np.linspace(0, self.T, n_nodes)
        x, y = self.solve_ivp(u, t)

        function_to_integrate = x * self.v * np.sin(u) - y * (self.v * np.cos(u) - self.w)

```

```
        return integrate(function_to_integrate, t)

    # решение задачи Коши с кешированием результата, чтобы не решать уравнение несколько раз при проверке на ограничения
    def solve(self, n_points):
        functional = lambda u: self.minimization_func(u)

        t = np.linspace(0, self.T, n_points)

        def constrain_xb(u):
            x, _ = self.solve_ivp(u, t)
            return x[-1]

        def constrain_yb(u):
            _, y = self.solve_ivp(u, t)
            return y[-1]

        constrains = [NonlinearConstraint(constrain_xb, self.x0, self.x0),
                      NonlinearConstraint(constrain_yb, self.y0, self.y0)]

        u0 = np.random.random(n_nodes)

        sol = minimize(functional, u0, method='trust-constr', constraints=constrains, options={'verbose': 3, 'maxiter': 200})

        x, y = self.solve_ivp(sol.x, t)

        if self.use_cache:
            print(f'cached calls: {self.cached_calls}, euler method calls: {self.euler_method_calls}, total solve calls: {self.cached_calls + self.euler_method_calls}')

        return sol.x, x, y


v = 1           # скорость движения самолета (в безветренную погоду)
w = -0.5        # скорость ветра (минус нужен чтобы направить ветер слева направо)
```

```

x0 = 10          # x-координата начальной точки полета
y0 = 25          # y-координата начальной точки полета

T = 1000         # время полета, за которое самолет должен вернуться в начальную точку

n_nodes = 100    # число отсчетов по времени (а значит и по всем остальным величинам) для которых будут расчитаны искомые величины

# конфигурация решателя задачи и, собственно, решение задачи
solver = ChaplyginProblemSolver(v, w, T, x0, y0, use_cache=True)
u_sol, x_sol, y_sol = solver.solve(n_points=n_nodes)


t = np.linspace(0, T, n_nodes)

plt.clf()
plt.figure(figsize=(12, 12), dpi=120)
plt.plot(t[1:], u_sol[1:])
plt.title('Полученное в результате оптимизации управление')
plt.ylabel('$u(t)$')
plt.xlabel('$t$')

plt.savefig('optimization_result.png')
plt.show()


# функция для приведения управления к гладкому виду путем вычитания/добавления 2 * pi * k (т.к. управление -- это угол)
def reduce_period(u):
    l = [u[0]]
    for i in range(1, u.shape[0]):
        best_diff = float('inf')
        best_x = -1
        for j in range(-2, 3):
            new_x = u[i] + j * 2 * np.pi
            diff = abs(l[i-1] - new_x)
            if diff < best_diff:
                best_x = new_x
                best_diff = diff
```

```

        l.append(best_x)
    return np.array(l)

u_sol = reduce_period(u_sol)

plt.clf()
plt.figure(figsize=(12, 12), dpi=120)
plt.plot(t[1:], u_sol[1:])
plt.title('Оптимальное управление приведенное к гладкому виду')
plt.ylabel('$u(t)$')
plt.xlabel('$t$')

plt.savefig('reduced_optimization_result.png')
plt.show()


plt.clf()
plt.figure(figsize=(12, 12), dpi=120)

plt.gca().set_aspect('equal', adjustable='box')

plt.plot(x_sol, y_sol, '-o', label='траектория облета')
for j, (x, y) in enumerate(zip(x_sol, y_sol)):
    if j % 10 == 0:
        plt.text(x, y, str(j))

plt.plot(x0, y0, 'r.', label='Стартовая точка')
plt.title('Оптимальная траектория полета (ветер дует слева)')
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.legend()
plt.savefig('optimal_trajectory.png')
plt.show()

vx = v * np.cos(u_sol) - w
vy = v * np.sin(u_sol)
v_abs = np.sqrt(vx ** 2 + vy ** 2)

plot_start_index = 1

plt.clf()
```

```
plt.figure(figsize=(12, 12), dpi=120)

plt.plot(t[plot_start_index:], vx[plot_start_index:], 'r-o', label='$V_{x}$')
plt.plot(t[plot_start_index:], vy[plot_start_index:], 'g-o', label='$V_{y}$')
plt.plot(t[plot_start_index:], v_abs[plot_start_index:], 'b-o', label='$|V|$')

plt.title('Зависимость проекций скорости и абсолютного значения скорочти от времени')

for j in range(plot_start_index, vx.shape[0]):
    if j % 10 == 0:
        plt.text(t[j], vx[j], str(j))
        plt.text(t[j], vy[j], str(j))

plt.legend()
plt.savefig('velocities.png')

plt.show()

n_nodes = 50

sols = []

different_solutions = 5
for i in range(different_solutions):
    solver = ChaplyginProblemSolver(v, w, T, x0, y0, use_cache=True)
    sols.append(solver.solve(n_points=n_nodes))

plt.clf()
plt.figure(figsize=(12, 12), dpi=120)

plt.gca().set_aspect('equal', adjustable='box')

for i, sol in enumerate(sols):
    _, x_sol, y_sol = sol
    plt.plot(x_sol, y_sol, '-o', label=f'Вариант № {i}')

plt.plot(x0, y0, 'r.', label='Стартовая точка')
```

```
plt.title('Несколько вариантов оптимальных траекторий облета')
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.legend()
plt.savefig('multiple_trajectories.png')

plt.show()


plt.clf()
plt.figure(figsize=(12, 12), dpi=120)

t = np.linspace(0, T, n_nodes)

for i, sol in enumerate(sols):
    u_sol = reduce_period(sol[0])
    plt.plot(t[1:], u_sol[1:], label=f'Вариант управление {i}')

plt.title('Несколько вариантов оптимальных управлений')
plt.ylabel('$u(t)$')
plt.xlabel('$t$')
plt.legend()

plt.savefig('multiple_controls.png')
plt.show()
```