

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра АПУ

КУРСОВАЯ РАБОТА
по дисциплине «Теория автоматического управления»
Тема: СИНТЕЗ СИСТЕМ СТАБИЛИЗАЦИИ ПЕРЕВЕРНУТОГО
МАЯТНИКА НА КАРЕТКЕ

Студент гр. 2392

Жук Ф.П.

Преподаватель

Имаев Д.Х.

Санкт-Петербург

2025

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Жук Ф.П.

Группа 2392

Тема работы: **СИНТЕЗ СИСТЕМ СТАБИЛИЗАЦИИ ПЕРЕВЕРНУТОГО
МАЯТНИКА НА КАРЕТКЕ**

Исходные данные:

Динамические модели перевернутых маятников различной конфигурации часто используются в исследованиях как наглядные примеры неустойчивых объектов, в частности, при сравнении методов синтеза алгоритмов автоматической стабилизации.

Предполагаемый объем пояснительной записки:

Не менее 00 страниц.

Дата выдачи задания: 00.00.2000

Дата сдачи реферата: 06.01.2025

Дата защиты реферата: 05.01.2025

Студент

Жук Ф.П

Преподаватель

Имаев Д.Х.

АННОТАЦИЯ

В курсовой работе выполнен синтез системы стабилизации перевернутого маятника на каретке. На основе нелинейной математической модели объекта (уравнений Коши) проведена линеаризация и анализ устойчивости. Проверены два метода синтеза регулятора: структурный синтез и метод размещения собственных значений.

SUMMARY

This term paper presents a synthesis of a stabilization system for an inverted pendulum on a carriage. Based on a nonlinear mathematical model of the object (Cauchy equations), linearization and stability analysis were performed, confirming the instability of the equilibrium position. Two methods of controller synthesis were tested: structural synthesis and the eigenvalue placement method.

СОДЕРЖАНИЕ

Введение	6
1. Перевернутый маятник на каретке	7
1.1. Схема неустойчивого механического объекта	7
1.2. Математическая модель маятника на каретке как объекта управления	7
1.3. Дифференциальные уравнения в форме Коши	9
1.4. Линеаризация дифференциальных уравнений	10
1.5. Передаточная функция объекта	11
1.6. Анализ устойчивости положения равновесия	12
2. Компьютерное моделирование	13
2.1. ПО для моделирования	13
2.2. Уравнения объекта в форме Коши	13
2.3. Моделирование	14
2.4. Фазовый портрет	15
3. Линеаризация системы	17
3.1. Линеаризация уравнений	17
3.2. Наблюдаемость и управляемость системы	18
3.3. Анализ устойчивости положения равновесия	19
4. Синтез регулятора для перевернутого маятника на каретке операторным методом	20
4.1. Синтез	20
4.2. Анализ линейной системы с регулятором	22
4.3. Моделирование нелинейной системы с регулятором	23
5. Синтез системы стабилизации перевернутого маятника на каретке методами пространства состояний	26
5.1. Синтез матрицы регулятора	26
5.2. Анализ системы с динамическим регулятором	26

5.3. Синтеза наблюдателя	27
5.4. Синтез динамического регулятора	27
5.5. Анализ системы с динамическим регулятором	28
5.6. Моделирование нелинейной системы с динамическим регулятором	29
Заключение	32
Список использованных источников	33
Приложение Код Проекта	34

ВВЕДЕНИЕ

Синтез и анализ системы маятника на каретке. На основе нелинейной математической модели объекта (уравнений Коши) проведена линеаризация и анализ устойчивости.

В ходе работы было выполнено:

1. Построение нелинейной математической модели и преобразовать ее в форму Коши.
2. Линеаризация, получение передаточной функции и анализ её устойчивость.
3. Компьютерное моделирование нелинейной системы.
4. Синтезированы регуляторы операторным методом и методом пространства состояний.
5. Оценка эффективности регуляторов через моделирование.

1. ПЕРЕВЕРНУТЫЙ МАЯТНИК НА КАРЕТКЕ

1.1. Схема неустойчивого механического объекта

Принципиальная схема неустойчивого механического объекта — перевернутого маятника на каретке — изображена на рис. 1.1.1.

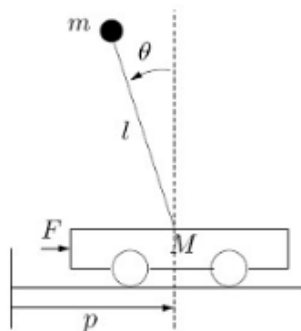


Рис. 1.1.1 - Принципиальная схема перевернутого маятника на каретке

На рисунке 1.1.1 приняты следующие обозначения параметров:

- m – масса маятника, кг;
- M – масса каретки, кг;
- l – длина маятника, м,

а также переменных:

- $\theta(t)$ – угол отклонения маятника, рад;
- $x(t)$ – положение каретки, м;
- $f(t)$ – сила, действующая на каретку, Н (кг*м/сек²).

Объект имеет две степени свободы – вращательное движение маятника и поступательное движение каретки. Управление таким объектом осложняется тем обстоятельством, что имеется только одно управляющее воздействие – сила $f(t)$, приложенная к каретке.

1.2. Математическая модель маятника на каретке как объекта управления

Классические и современные методы синтеза систем автоматического управления основаны на математических моделях в виде дифференциальных или разностных уравнений.

Построим аналитическую модель, со следующими допущениями:

- Массы сосредоточены (масса маятника сосредоточена в точке);
- Отсутствует сопротивление среды;
- Отсутствует трение.

В качестве обобщенных координат для рассматриваемой системы с двумя степенями свободы выберем:

- $\theta(t)$ — угол отклонения маятника;
- $x(t)$ — положение каретки.

Для записи уравнений динамики механической системы воспользуемся уравнениями Лагранжа второго рода

$$\begin{aligned} \frac{d}{dt} \frac{\partial T}{\partial \dot{x}} - \frac{\partial T}{\partial x} &= -\frac{\partial P}{\partial x} + f; \\ \frac{d}{dt} \frac{\partial T}{\partial \dot{\theta}} - \frac{\partial T}{\partial \theta} &= -\frac{\partial P}{\partial \theta}, \end{aligned} \quad (1)$$

, где T — кинетическая энергия, P — потенциальная энергия, f — приложенная к каретке неконсервативная обобщенная сила.

Выражение для кинетической энергии запишется так

$$T = \frac{1}{2} M \dot{x}^2 + \frac{1}{2} m v_A^2, \quad (2)$$

, где

$$\begin{aligned} v_A^2 &= \dot{x}_A^2 + \dot{y}_A^2; \\ x_A &= x + l \sin \theta; \quad \dot{x}_A = \dot{x} + l \cos \theta \dot{\theta}; \\ y_A &= l \cos \theta; \quad \dot{y}_A = -l \sin \theta \dot{\theta}; \\ v_A^2 &= \dot{x}^2 + 2\dot{x}\dot{\theta}l \cos \theta + l^2 \dot{\theta}^2. \end{aligned}$$

С учетом этих выражений вместо (2) получим

$$T = \frac{1}{2}(M + m)\dot{x}^2 + m\dot{x}\dot{\theta}l \cos \theta + \frac{1}{2}ml^2\dot{\theta}^2. \quad (3)$$

Потенциальная энергия для силы тяжести равна

$$P = -mgh = -mgl(1 - \cos \theta). \quad (4)$$

В результате подстановки (3) и (4) в (1) получим математическую модель рассматриваемого объекта в виде системы двух дифференциальных уравнений второго порядка

$$ml^2 \frac{d^2\theta}{dt^2} - mgl \sin \theta + ml^2 \cos \theta \frac{d^2x}{dt^2} = 0; \quad (5)$$

$$(M + m) \frac{d^2x}{dt^2} + ml \cos \theta \frac{d^2\theta}{dt^2} - ml \sin \theta \left(\frac{d\theta}{dt} \right)^2 = f. \quad (6)$$

Уравнения (5) и (6) представляют собой выражения баланса моментов, действующих на маятник, и баланса сил, действующих на каретку.

1.3. Дифференциальные уравнения в форме Коши

Анализ объекта и его компьютерное моделирование упрощаются, если дифференциальные уравнения разрешены относительно старших производных.

Заметим, что вторые производные в исходные уравнения (5), (6) входят линейно. С учетом этого приведем уравнения к матричной форме:

$$\begin{bmatrix} l & \cos \theta \\ ml \cos \theta & M + m \end{bmatrix} \times \begin{bmatrix} \ddot{\theta} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} g \sin \theta \\ ml \dot{\theta}^2 \sin \theta + f \end{bmatrix}.$$

Прежде всего, проверим существование и единственность решения — вычислим определитель матрицы:

$$D = l(M + m) - ml \cos^2 \theta = lM + lm \sin^2 \theta \neq 0$$

и убедимся в том, что он не равен нулю.

Для решения системы уравнений воспользуемся правилом Крамера

$$\ddot{\theta} = \frac{1}{D} \begin{vmatrix} g \sin \theta & \cos \theta \\ ml\dot{\theta}^2 \sin \theta + f & M + m \end{vmatrix} = \\ = ((M + m)g \sin \theta - ml\dot{\theta}^2 \sin \theta \cos \theta - f \cos \theta) / D; \\ \ddot{x} = \frac{1}{D} \begin{vmatrix} l & g \sin \theta \\ ml \cos \theta & ml\dot{\theta}^2 \sin \theta + f \end{vmatrix} = (ml^2\dot{\theta}^2 \sin \theta + lf - mgl \sin \theta \cos \theta) / D.$$

Заметим, что правые части уравнений не содержат переменных x , \dot{x} т. е. положение и скорость каретки не влияют на ускорения маятника и каретки. Объект может занимать любое положение или совершать равномерное поступательное движение.

Теперь легко записать уравнения объекта в форме Коши:

$$\frac{d\theta}{dt} = \dot{\theta}; \\ \frac{d\dot{\theta}}{dt} = \ddot{\theta} = ((M + m)g \sin \theta - ml\dot{\theta}^2 \sin \theta \cos \theta - f \cos \theta) / D; \quad (7) \\ \frac{dx}{dt} = \dot{x}; \\ \frac{d\dot{x}}{dt} = \ddot{x} = (ml^2\dot{\theta}^2 \sin \theta + lf - mgl \sin \theta \cos \theta) / D..$$

1.4. Линеаризация дифференциальных уравнений

Будем рассматривать малые отклонения переменных θ и $\dot{\theta}$, когда приближенно можно принять: $\sin \theta \approx \theta$; $\cos \theta \approx 1$; $\dot{\theta}^2 \approx 0$. Пренебрегая малыми величинами высших порядков, вместо нелинейных уравнений (7) получим линейные (линеаризованные) уравнения

$$\frac{d\theta}{dt} = \dot{\theta}; \\ \frac{d\dot{\theta}}{dt} = \frac{(M + m)g\theta - f}{lM}; \\ \frac{dx}{dt} = \dot{x}; \\ \frac{d\dot{x}}{dt} = \frac{f - mg\theta}{M}. \quad (8)$$

Запишем линейную систему (8) в матричной форме с использованием вектора состояний

$$\frac{d}{dt} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{m+M}{lM}g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{lM} \\ 0 \\ \frac{1}{M} \end{bmatrix} f; \quad (9)$$

$$x = [0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \end{bmatrix} + 0 \cdot f. \quad (10)$$

Получена линейная модель в так называемой форме пространства состояний (ФПС):

$$\begin{aligned} \frac{d\mathbf{v}}{dt} &= \mathbf{A}\mathbf{v} + \mathbf{B}f \\ x &= \mathbf{C}\mathbf{v} + Df \end{aligned}$$

Первое из этих уравнений называется уравнением состояний, второе — уравнением выхода.

В уравнении выхода (10) за выход объекта — измеряемую непосредственно переменную принято положение каретки, т. е. скаляр. Поэтому матрица выхода \mathbf{C} оказывается строкой. Если за выход принимать вектор $(\theta, x)'$, то матрица выхода будет иметь две строки

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

а матрица обхода получится столбцовой

$$\mathbf{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

1.5. Передаточная функция объекта

Передаточная функция линейного объекта представляет собой отношение изображений выхода и входа при нулевых начальных условиях.

Прежде всего, получим характеристический полином матрицы

$$A(s) = \det(s\mathbf{I} - \mathbf{A}) = s^2(s^2 - \frac{M+m}{lM}g), \quad (11)$$

который является знаменателем передаточных функций.

Далее получим полиномы числителей передаточных функций: от входа — силы F , приложенной к каретке, до переменной выхода — положения каретки x

$$B_{xF}(s) = \frac{1}{M}(s^2 - g/l), \quad (12)$$

и до переменной θ — углового положения маятника

$$B_{\theta F}(s) = \frac{1}{lM}s^2. \quad (13)$$

1.6. Анализ устойчивости положения равновесия

Устойчивость “в малом” положения равновесия $(0 \ 0 \ 0 \ 0)^T$ нелинейной модели (7) была выявлена по линеаризованной модели, т. е. на основе первого метода Ляпунова. Условием асимптотической устойчивости положения равновесия нелинейной системы является принадлежность корней характеристического полинома линеаризованной системы левой полуплоскости.

Корни характеристического полинома (11) равны:

$$s_1 = s_2 = 0; s_{3,4} = \pm \sqrt{\frac{M+m}{M}} \sqrt{\frac{g}{l}}.$$

Имеется действительный положительный корень (в правой полуплоскости); кроме того, полином имеет двукратный нулевой корень. Это свидетельствует о неустойчивости положения равновесия нелинейной системы.

2. КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ

2.1. ПО для моделирования

Линеаризованные уравнения (9) позволяют исследовать устойчивость и качественный характер движений при малых отклонениях состояния системы от положения равновесия. Для исследования поведения объекта управления при любых отклонениях необходимо решать нелинейные уравнения. Исходные нелинейные дифференциальные (7) уравнения нельзя решить аналитическим способом.

Поэтому используют численные решения при конкретных начальных условиях и внешних воздействиях. Для автоматизации численных решений разработаны программные средства; далее будем использовать Python и следующий набор библиотек:

- `numpy` — библиотека для быстрого и удобного вычисления с многомерными массивами и матрицами;
- `sympy` — библиотека для символьных математических вычислений и алгебры;
- `scipy` — библиотека для научных и инженерных вычислений, расширяющая возможности `numpy`;
- `matplotlib.pyplot` — библиотека для построения графиков и визуализации данных в Python;
- `control` — библиотека для анализа и синтеза систем управления и автоматизации;
- `tqdm` — библиотека для отображения прогресс-баров при выполнении циклов и длительных операций.

2.2. Уравнения объекта в форме Коши

Уравнения (7) были записаны в виде функции в Python (рис. 2.2.1)

```

def koshi(y, t):
    """
    y - вектор состояния
    theta - угол
    dtheta - скорость изменения угла
    x - положение в пространстве
    dx - скорость коретки
    t - время
    """
    theta, dtheta, x, dx = y
    sin_theta = np.sin(theta)
    cos_theta = np.cos(theta)

    denominator = (M + m)*l - m*l*cos_theta**2

    ddtheta = ((M + m)*g*sin_theta - m*l*dtheta**2*sin_theta*cos_theta - f*cos_theta) / denominator
    ddx = (m*l**2*dtheta**2*sin_theta + l*f - m*g*l*sin_theta*cos_theta) / denominator

    return [dtheta, ddtheta, dx, ddx]

```

Рис. 2.2.1 - Python функция уравнения объекта в форме Коши

, где y - вектор состояния в виде $[theta - \text{угол}, dtheta - \text{скорость изменения угла}, x - \text{положение в пространстве}, dx - \text{скорость коретки}]$, t – время (параметр для odeint).

2.3. Моделирование

Выберем следующие значения параметров: $l = 0.25$ м; $m = 0.2$ кг; $M = 0.4$ кг, $g = 10$ м/с².

Для моделирования во времени была написана функция model (рис. 2.3.1).

```

def model(
    func,
    theta0,
    dtheta0,
    ts=10, nt=0.1, x0 = 0.0, dx0 = 0.0
):
    """
    func - функция управления

    theta0 - начальный угол

    dtheta0 - начальная скорость изменения угла

    ts - время

    nt - шаг дискретизации времени

    x0 - начальная точка коретки

    dx0 - начальная скорость коретки
    """
    initial_conditions = [theta0, dtheta0, x0, dx0]
    t = np.arange(0, ts, nt)
    sol = odeint(func, initial_conditions, t)
    return sol

```

Рис. 2.3.1 - Функция model

Функция `model` численно решает задачу моделирования динамики системы с помощью метода `odeint`: она принимает функцию управления `func`, начальные условия (угол, скорость изменения угла, положение, скорость каретки), параметры времени (`ts` — длительность моделирования, `nt` — шаг по времени), строит массив времени, и возвращает решение системы дифференциальных уравнений в виде массива значений состояния системы на каждом временном шаге.

Модуляция проходила при следующих начальных значениях ($\theta_0 = 1$, $\dot{\theta}_0 = 0$, $x = 0$, $y = 0$). В её результате были получены рисунок 2.3.2.

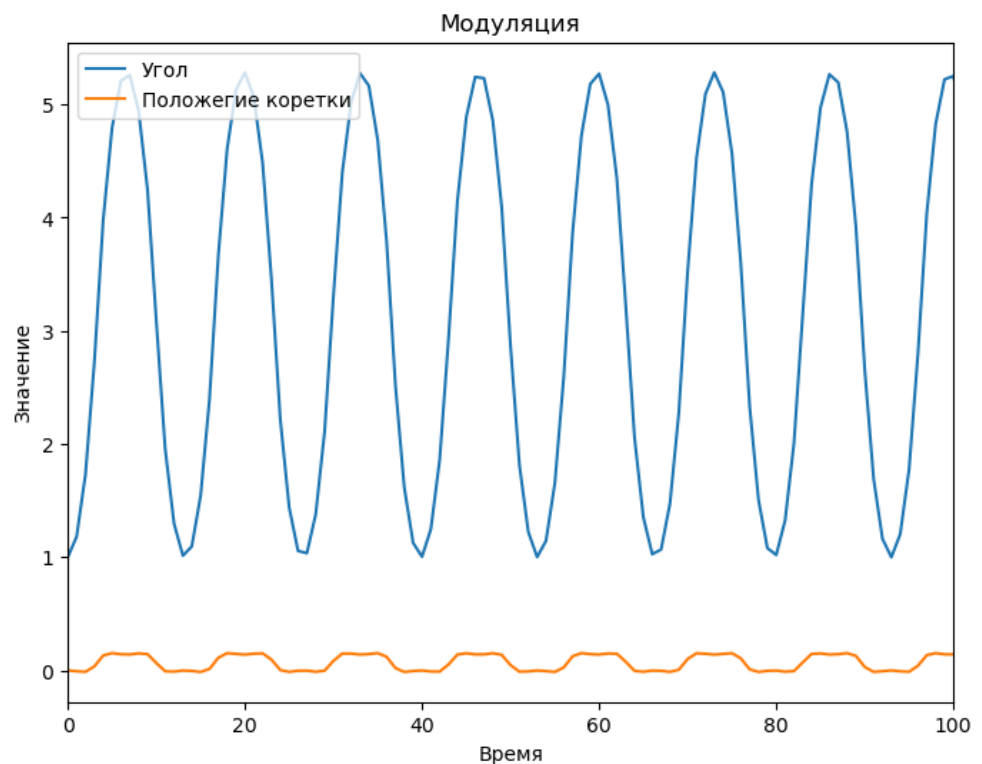


Рис 2.3.2 – Модуляция

Из результатов моделирования ясно, что верхнее положение маятника не устойчиво — при малейшем отклонении от него состояние системы не возвращается к нему, а начинаются колебания маятника относительно нижнего положения.

2.4. Фазовый портрет

После моделирования был построен фазовый портрет (рис. 2.4.1).

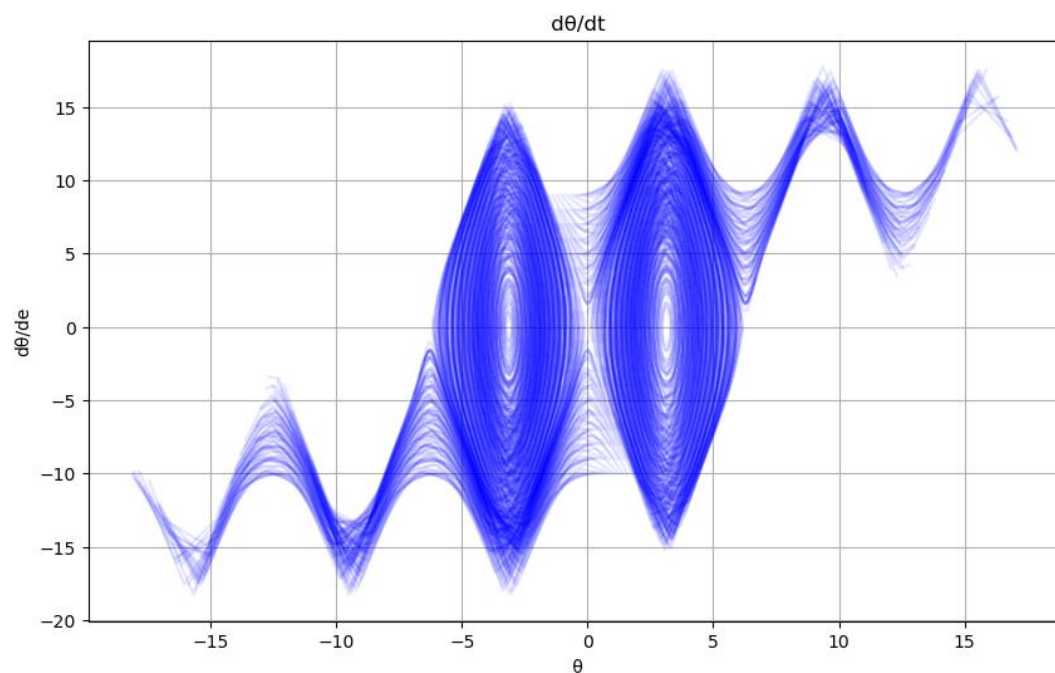


Рис. 2.4.1 – Фазовый портрет

На данном фазовом портрете показаны траектории движения динамической системы в пространстве переменных угол θ и его производная $d\theta/dt$: видны замкнутые и вытянутые траектории, что свидетельствует о наличии устойчивых колебательных режимов, что подтверждается моделированием.

3. ЛИНЕАРИЗАЦИЯ СИСТЕМЫ

3.1. Линеаризация уравнений

Изначально были сформированы матриц состояния (A, B, C, D), полученных из уравнений движения маятника на тележке (рис 3.1.1).

```
A = np.array([
    [0, 1, 0, 0],
    [(M+m)*g/(1*M), 0, 0, 0],
    [0, 0, 0, 1],
    [-m*g/M, 0, 0, 0]
])

B = np.array([
    [0], [-1/(1*M)], [0], [1/M]
])
C = np.array([[0, 0, 1, 0]])
D = np.array([[0]])

C_theta = np.array([[1, 0, 0, 0]])
```

Рис. 3.1.1 - Матрицы состояния

- Матрица A - матрица коэффициентов для вектора состояния системы $[\theta, d\theta, x, dx]$;
- Матрица B - входная матрица, показывает, как управляющее воздействие влияет на скорость и ускорение системы;
- Матрица C - матрица наблюдения, выбрана так, чтобы наблюдать только положение тележки;
- Матрица D - прямая передача;
- Матрица C_theta - альтернативная матрица наблюдения, если мы наблюдаем только угол маятника.

Далее была создана система пространства состояний с помощью библиотеки control (рис 3.1.2).

```
sys = ct.ss(A, B, C, D)
sys
```

$$\left(\begin{array}{cccc|c} 0 & 1 & 0 & 0 & 0 \\ 58.8 & 0 & 0 & 0 & -10 \\ 0 & 0 & 0 & 1 & 0 \\ -4.9 & 0 & 0 & 0 & 2.5 \\ \hline 0 & 0 & 1 & 0 & 0 \end{array} \right)$$

Рис. 3.1.2 - Система пространства состояний

3.2. Наблюдаемость и управляемость системы

Наблюдаемость системы по углу была определена следующим образом (рис. 3.2.1).

```
nabl = ct.observ(A, C_theta)
print(f"Размер матрицы наблюдаемости {nabl.shape}")
print(f"Ранг матрицы наблюдаемости: {np.linalg.matrix_rank(nabl)}")
✓ 0.0s Python
```

Размер матрицы наблюдаемости (4, 4)
Ранг матрицы наблюдаемости: 2

Рис. 3.2.1 - Наблюдаемость системы по углу

Так как ранга матрицы наблюдаемости не равен ее размеру, то следует что система не наблюдаема по углу.

Так как система не наблюдаема по углу, проверим наблюдаемость по положению каретки. Наблюдаемость системы по положению каретки была определена следующим образом (рис. 3.2.2).

```
nabl = ct.observ(A, C)
print(f"Размер матрицы наблюдаемости {nabl.shape}")
print(f"Ранг матрицы наблюдаемости: {np.linalg.matrix_rank(nabl)}")
✓ 0.0s Python
```

Размер матрицы наблюдаемости (4, 4)
Ранг матрицы наблюдаемости: 4

Рис. 3.2.2 - Наблюдаемость системы по положению каретки

Так как ранга матрицы наблюдаемости равен ее размеру, то система наблюдаема по положению каретки. И, следовательно в качестве параметра для управления был выбран положение каретки в пространстве.

Управляемость системы была определена следующим образом (рис. 3.2.3).

```
upr = ct.ctrb(A, B)
print(f"Размер матрицы управляемости {nabl.shape}")
print(f"Ранг матрицы управляемости: {np.linalg.matrix_rank(upr)}")
✓ 0.0s Python
```

Размер матрицы управляемости (4, 4)
Ранг матрицы управляемости: 4

Рис. 3.2.3 - Управляемость системы

Исходя из ранга матрицы управление, следует что система управляема.

3.3. Анализ устойчивости положения равновесия

В ходе исследования была получена карта полюсов и нулей системы (рис. 3.3.1).

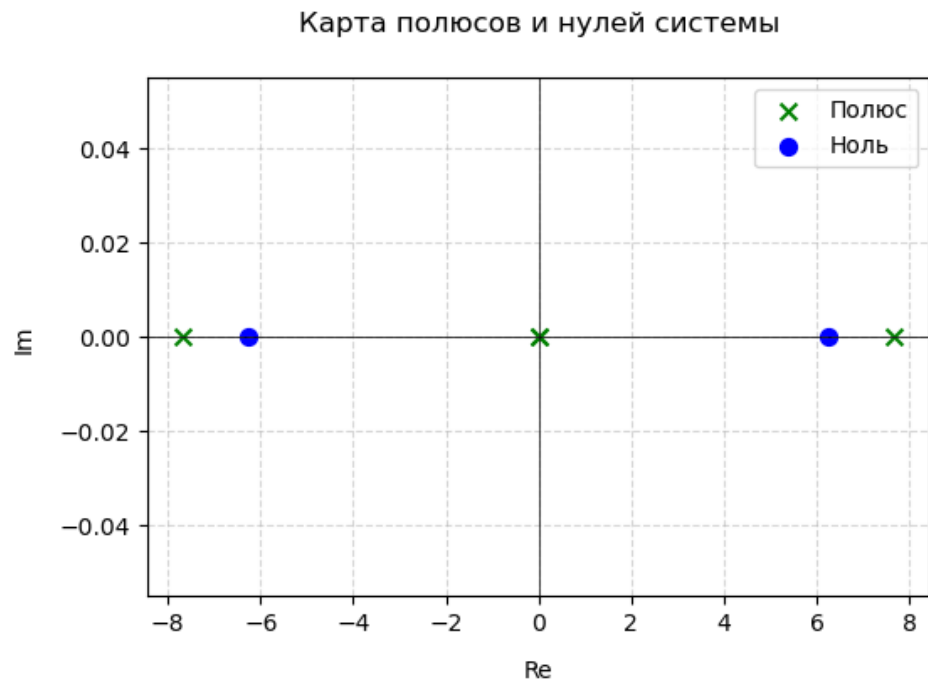


Рис. 3.3.1 - Карта полюсов и нулей системы

Все полюса и нули находятся на вещественной оси — это отражает, что система не имеет колебательных частей в отклике, а все динамические процессы апериодические. Система в исходном виде (без регулятора) является неустойчивой, так как имеет полюса в правой полуплоскости. Для стабилизации системы необходим синтез регулятора, который перенесёт все полюса в левую полуплоскость.

4. СИНТЕЗ РЕГУЛЯТОРА ДЛЯ ПЕРЕВЕРНУТОГО МАЯТНИКА НА КАРЕТКЕ ОПЕРАТОРНЫМ МЕТОДОМ

4.1. Синтез

Синтез проведем по линеаризованной модели объекта, полученной в форме пространства состояний. Характеристический полином матрицы (11) является знаменателем передаточных функций. Корни полинома, т. е. полюсы передаточных функций равны:

$$s_1 = s_2 = 0; s_{3,4} = \pm \sqrt{(M+m)/M} \sqrt{g/l}.$$

В нашей системе они равны: $[0+0j, 0+0j, 7.67+0j, -7.67+0j]$. Поскольку полюсы не равны нулям, передаточная функция по этому каналу является полной — объект полностью управляем и наблюдаем.

Далее была получена передаточная функция линеаризованной системы (рис. 4.1.1).

```
A_o = 1/M*(s**2-g/l)
B_o = (s**2*(s**2-(M+m)/(1*M)*g))
W_o = A_o/B_o
W_o
```

✓ 0.0s Python

$$\frac{2.5s^2 - 98.0}{s^2 (s^2 - 58.8)}$$

Рис. 4.1.1 - Передаточная функция линеаризованной системы

Так же была составлена передаточная функция регулятора (рис. 4.1.2).

```
A_p = (a_3*s**3+a_2*s**2+a_1*s+a_0)
B_p = (s**3+b_2*s**2+b_1*s+b_0)
W_p = A_p/B_p
W_p
```

✓ 0.0s Python

$$\frac{a_0 + a_1s + a_2s^2 + a_3s^3}{b_0 + b_1s + b_2s^2 + s^3}$$

Рис. 4.1.2 - Передаточная функция регулятора

С помощью обратной связи система была связана с регулятором и была получена соответствующая передаточная функция линеаризованной системы с регулятором (рис. 4.1.3).

```

A_s = A_o*B_p
B_s = A_o*A_p + B_o*B_p
W_s = sympy.together(A_s/B_s)
W_s

```

$$\frac{(2.5s^2 - 98.0)(b_0 + b_1s + b_2s^2 + s^3)}{s^2(s^2 - 58.8)(b_0 + b_1s + b_2s^2 + s^3) + (2.5s^2 - 98.0)(a_0 + a_1s + a_2s^2 + a_3s^3)}$$

Рис. 4.1.3 - Передаточная функция линейризованной системы с регулятором

Поскольку порядок системы равен семи, назначим семь желаемых корней в левой полуплоскости: [-0.2, -0.4, -1, -1.2247, -2, -4, -8]. Уравнение изображено на рисунке 4.1.4.

```

a = [-0.2, -0.4, -1, -1.2247, -2, -4, -8]
polynom = sympy.prod(sympy.Array([s]*7)-sympy.Array(a))
polynom = sympy.cancel(polynom)
#polynom = (s + 1)**7
polynom

```

$$1.0s^7 + 16.8247s^6 + 98.18532s^5 + 260.049276s^4 + 341.47104s^3 + 221.41752s^2 + 63.9056s + 6.270464$$

Рис. 4.1.4 – Уравнение желаемого характеристического полинома

Далее с помощью библиотеки были рассчитаны коэффициенты для синтеза регулятора (рис. 4.1.5).

```

eq = sympy.Eq(B_s, polynom)
eq

```

$$s^2(s^2 - 58.8)(b_0 + b_1s + b_2s^2 + s^3) + (2.5s^2 - 98.0)(a_0 + a_1s + a_2s^2 + a_3s^3) = 1.0s^7 + 16.8247s^6 + 98.18532s^5 + 260.049276s^4 + 341.47104s^3 + 221.41752s^2 + 63.9056s + 6.270464$$

```

solv = sympy.solve(eq, [a_1, a_2, a_3, a_0, b_1, b_2, b_0])
solv

```

```

{a_0: -0.0639843265306122,
 a_1: -0.652097959183673,
 a_2: 1503.73195260441,
 a_3: 195.384451038734,
 b_0: -2509.98824551104,
 b_1: -331.475807596835,
 b_2: 16.8247000000000}

```

Рис. 4.1.4 – Решение системы урвний

Для данного случая были получены следующие коэффициенты регулятора системы:

- a_0: -0.0639843265306122,
- a_1: -0.652097959183673,
- a_2: 1503.73195260441,
- a_3: 195.384451038734,
- b_0: -2509.98824551104,
- b_1: -331.475807596835,
- b_2: 16.8247000000000

Подставим данные коэффициенты и получим следующий регулятор:

$$\frac{195.384451038734s^3 + 1503.73195260441s^2 - 0.652097959183673s - 0.0639843265306122}{s^3 + 16.8247s^2 - 331.475807596835s - 2509.98824551104}$$

4.2. Анализ линейной системы с регулятором

Передаточная функция замкнутой системы имеет вид:

$$\frac{2.5s^5 + 42.06175s^4 - 926.689518992087s^3 - 7923.79121377759s^2 + 32484.6291444898s + 245978.848060082}{s^7 + 16.8247s^6 + 98.18532s^5 + 260.049276s^4 + 341.471040000004s^3 + 221.417520000017s^2 + 63.9056s + 6.270464}$$

Найдем собственные значения замкнутой системы (рис. 4.2.1).

```
W_s_den = W_s.den
W_s_den
✓ 0.0s Python

s7 + 16.8247s6 + 98.18532s5 + 260.049276s4 + 341.471040000004s3 + 221.417520000017s2 + 63.9056s + 6.270464

eq = sympy.Eq(W_s_den, 0)
eq
✓ 0.0s Python

s7 + 16.8247s6 + 98.18532s5 + 260.049276s4 + 341.471040000004s3 + 221.417520000017s2 + 63.9056s + 6.270464 = 0

solv = sympy.solve(eq, s)
solv
✓ 0.1s Python

[-7.99999999999999,
-3.99999999999998,
-2.000000000000134,
-1.224699999999344,
-1.000000000000574,
-0.399999999999431,
-0.200000000000074]
```

Рис. 4.2.1 - Собственные значения замкнутой системы

Как видно они получились в точности заданными (с учетом погрешности при расчетах).

4.3. Моделирование нелинейной системы с регулятором

Моделирование было проведено аналогично как в случае нелинейной системы. За исключением мелких правок. Функция изображена на рисунке 4.3.1.

```
def regul_sys(y, t):  
    """  
    y - вектор состояния  
    theta - угол  
    dtheta - скорость изменения угла  
    x - положение в пространстве  
    dx - скорость коретки  
    t - время  
    """  
    theta, dtheta, x, dx = y[0:4]  
    reg_vect = y[4:]  
  
    error_x = 0 - x  
    f = (reg.C @ reg_vect + reg.D * error_x)[0][0]  
  
    sin_theta = np.sin(theta)  
    cos_theta = np.cos(theta)  
  
    denominator = (M + m)*1 - m*1*cos_theta**2  
  
    ddtheta = ((M + m)*g*sin_theta - m*1*dtheta**2*sin_theta*cos_theta - f*cos_theta) / denominator  
    ddx = (m*1**2*dtheta**2*sin_theta + 1*f - m*g*1*sin_theta*cos_theta) / denominator  
  
    dreg_vec = reg.A@reg_vect+reg.B.reshape(-1)*error_x  
  
    return np.concatenate([[dtheta, ddtheta, dx, ddx], dreg_vec])
```

Рис. 4.3.1 - Функция нелинейной системы с регулятором

Она имеет следующие различия:

- Принимает на вход и обрабатывает дополнительный вектор состояния регулятора, что позволяет моделировать совместную динамику системы и регулятора;
- В возвращаемом векторе производных теперь присутствуют не только производные основных переменных, но и производные состояния регулятора, что позволяет моделировать и анализировать совместную эволюцию всей системы и регулятора.

Так же были внесены поправки в функцию моделирования, в частности на вход податься расширенный вектор состояния регулятора, включающий производные состояния регулятора (рис. 4.3.2).

```
def model_reg(
    func,
    theta0,
    dtheta0,
    ts=10, nt=0.1, x0 = 0.0, dx0 = 0.0
):
    """
    func - функция управления

    theta0 - начальный угол

    dtheta0 - начальная скорость изменения угла

    ts - время

    nt - шаг дискретизации времени

    x0 - начальная точка коретки

    dx0 - начальная скорость коретки
    """
    initial_conditions = np.array([theta0, dtheta0, x0, dx0, 0, 0, 0])
    t = np.arange(0, ts, nt)
    sol = odeint(func, initial_conditions, t)
    return sol
```

Рис. 4.3.2 - Функция моделирования нелинейной системы с регулятором

В ходе моделирования был получен следующий график (рис. 4.3.3). Начальные условия ($\theta_0 = 0.07$, $d\theta_0 = 0$, $x = 0$, $y = 0$).

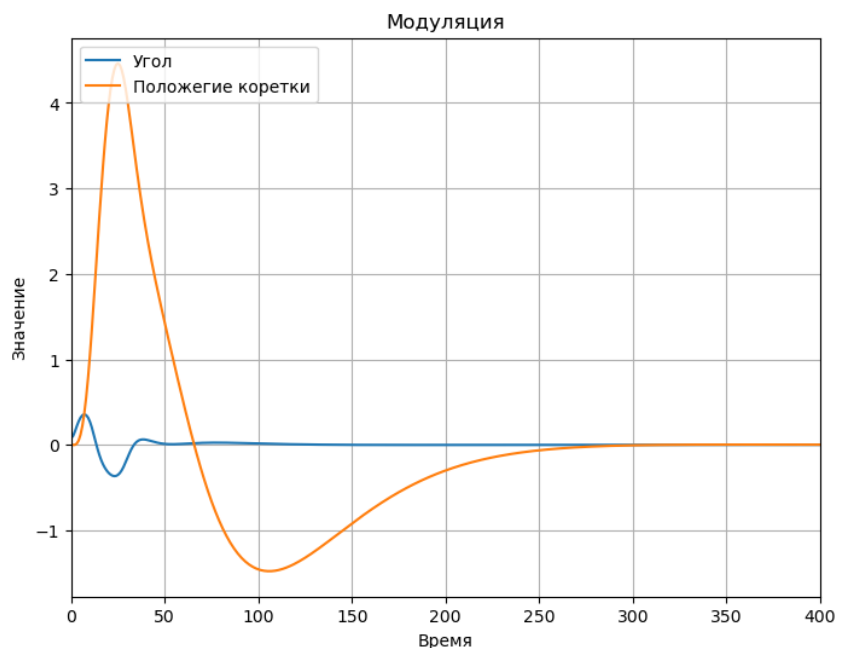


Рис. 4.3.3 – Модуляция нелинейной системы с регулятором

При увеличении начального угла (> 0.08) регулятор уже не справляется с задачей стабилизации. Регулятор справляется с задачей при угле отклонения < 0.08 и стабилизирует системы в точке равновесия, сводя колебания 0.

5. СИНТЕЗ СИСТЕМЫ СТАБИЛИЗАЦИИ ПЕРЕВЕРНУТОГО МАЯТНИКА НА КАРЕТКЕ МЕТОДАМИ ПРОСТРАНСТВА СОСТОЯНИЙ

5.1. Синтез матрицы регулятора

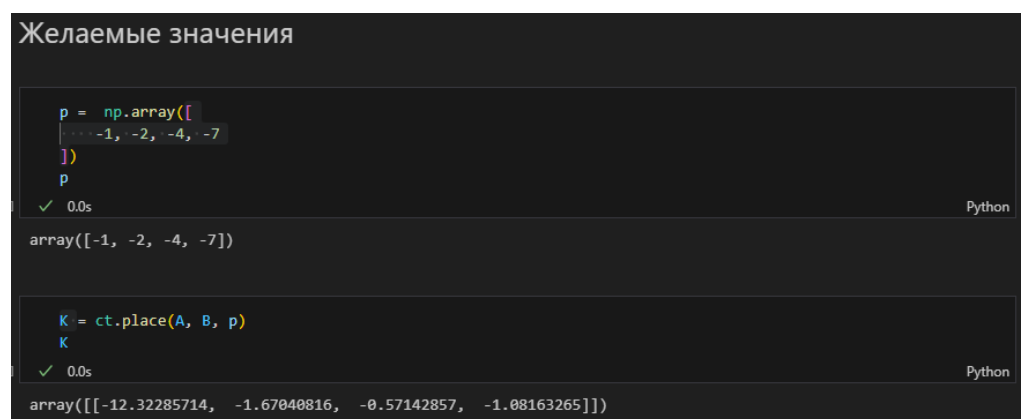
Задача синтеза заключается в определении матрицы регулятора состояния K из условия желаемого расположения собственных значений матрицы системы.

Существует произвол в выборе желаемых собственных значений системы. Чем дальше находится от мнимой оси собственное значение, тем быстрее затухают процессы. Однако, требование большего быстродействия означает необходимость приложения чрезмерных усилий на каретку.

Синтез будем проводить с использованием линеаризованной модели, данная модель имеет следующие собственные значения системы: $[0+0j, 0+0j, 7.67+0j, -7.67+0j]$.

В качестве желаемых выберем: $[-1, -2, -4, -7]$.

Матрицу регулятора, обеспечивающего желаемое расположение собственных значений, найдем с помощью библиотеки control (рис. 5.1.1).



```
Желаемые значения

p = np.array([
    -1, -2, -4, -7
])
p
array([-1, -2, -4, -7])

K = ct.place(A, B, p)
K
array([[ -12.32285714,  -1.67040816,  -0.57142857,  -1.08163265]])
```

Рис. 5.1.1 - Матрица регулятора состояния K

5.2. Анализ системы с динамическим регулятором

Проведем анализ устойчивости системы (рис. 5.2.1).

```
np.linalg.eig(A-B*K)[0]
✓ 0.0s
array([-7., -4., -2., -1.])
```

Замкнутая система имеет желаемое расположение собственных значений.

Рис. 5.2.1 - анализ устойчивости системы

Замкнутая система имеет желаемое расположение собственных значений.

5.3. Синтеза наблюдателя

Регулятор состояния формирует управляющие воздействия на основе текущей информации обо всех переменных состояния. Реально измеряется только положение каретки x . Для вычисления остальных переменных состояния используют так называемый наблюдатель состояния.

Задача синтеза наблюдателя сводится к поиску матрицы L , и имеет решение, если объект наблюдаем полностью. А объект является полностью наблюдаемым, если мы в качестве параметра для наблюдения выбираем положение каретки в пространстве.

Для синтеза наблюдателя воспользуемся методом размещения собственных значений – назначим собственные значения наблюдателя несколько дальше от мнимой оси в левой полуплоскости: $p_o = [-5, -10, -20, -35]$.

Матрица наблюдателя L вычисляется с помощью `control` (рис 5.3.1).

```
L = ct.place(A.T, C.T, po).T
L
✓ 0.0s
array([[ -3544.08163265],
       [-26748.45714286],
        [   70.         ],
        [ 1633.8         ]])
```

Рис. 5.3.1. - Матрица наблюдателя

5.4. Синтез динамического регулятора

Уравнения динамического регулятора имеют вид:

$$\frac{dv_r}{dt} = \mathbf{A}_r \mathbf{v}_r + \mathbf{B}_r x;$$

$$-f = \mathbf{C}_r \mathbf{v}_r + \mathbf{D}_r x,$$

Динамический регулятор получим, если объединить регулятор состояния и наблюдатель (рис. 5.4.1). В результате динамический регулятор имеет четвертый порядок.

```
def dynamic_reg(A, B, C, D, K, L):
    Ar = A - B @ K - L @ C
    Br = L
    Cr = -K
    Dr = np.zeros((K.shape[0], C.shape[0]))
    return Ar, Br, Cr, Dr

Ar, Br, Cr, Dr = dynamic_reg(A, B, C, D, K, L)
```

Рис. 5.4.1 – расчет коэффициентов динамического регулятора

В итоге регулятор изображён на рисунке 5.4.2.

```
regl = ct.ss(Ar, Br, Cr, Dr)
regl
```

0	1	$3.54 \cdot 10^3$	0	$-3.54 \cdot 10^3$
-64.4	-16.7	$2.67 \cdot 10^4$	-10.8	$-2.67 \cdot 10^4$
0	0	-70	1	70
25.9	4.18	$-1.63 \cdot 10^3$	2.7	$1.63 \cdot 10^3$
12.3	1.67	0.571	1.08	0

Рис. 5.4.2 - Динамический регулятор

5.5. Анализ системы с динамическим регулятором

Для анализа по линейным моделям получим матрицы системы уравнений замкнутой системы (рис. 5.5.1).

```
sys_reg = ct.feedback(sys, regl, sign=1)
sys_reg
```

0	1	0	0	0	0	0	0	0
58.8	0	0	0	-123	-16.7	-5.71	-10.8	-10
0	0	0	1	0	0	0	0	0
-4.9	0	0	0	30.8	4.18	1.43	2.7	2.5
0	0	$-3.54 \cdot 10^3$	0	0	1	$3.54 \cdot 10^3$	0	0
0	0	$-2.67 \cdot 10^4$	0	-64.4	-16.7	$2.67 \cdot 10^4$	-10.8	0
0	0	70	0	0	0	-70	1	0
0	0	$1.63 \cdot 10^3$	0	25.9	4.18	$-1.63 \cdot 10^3$	2.7	0
0	0	1	0	0	0	0	0	0

Рис. 5.5.1 - Замкнутая система

Вычислим собственные значения системы (рис. 5.5.2).

```
eig = np.linalg.eig(sys_reg_.A)
print(f"Собственные числа системы: {eig[0]}")
✓ 0.0s
Собственные числа системы: [-35. -20. -10. -7. -5. -4. -2. -1.]
```

Рис. 5.5.2 - Собственные значения системы

Замкнутая система с динамическим регулятором устойчива и имеет желаемые собственные значения.

Недостатком процедуры синтеза является избыточность наблюдателя, который заключается в превышении необходимого порядка. Поскольку одна из переменных состояния, а именно, положение каретки измеряется непосредственно, то нет необходимости в ее восстановлении.

5.6. Моделирование нелинейной системы с динамическим регулятором

Моделирование было проведено аналогично как в случае нелинейной системы. За исключением мелких правок. Функция изображена на рисунке 5.6.1.

```
def regul_sys_(y, t):
    """
    y - вектор состояния
    x - положение в пространстве
    theta - угол
    dtheta - скорость изменения угла
    dx - скорость каретки
    t - время
    """
    theta, dtheta, x, dx = y[0:4]
    reg_vect = y[4:]

    y_meas = x

    f = -K @ reg_vect

    sin_theta = np.sin(theta)
    cos_theta = np.cos(theta)
    denominator = (M + m)*1 - m*1*cos_theta**2
    ddtheta = ((M + m)*g*sin_theta - m*1*dtheta**2*sin_theta*cos_theta - f*cos_theta) / denominator
    ddx = (m*1**2*dtheta**2*sin_theta + 1*f - m*g*1*sin_theta*cos_theta) / denominator

    dreg_vec = A @ reg_vect + B.flatten()*f + L.flatten()*(y_meas - reg_vect[2])

    return np.concatenate([[dtheta, ddtheta[0], dx, ddx[0]], dreg_vec])
✓ 0.0s
```

Рис. 5.6.1 - Функция нелинейной системы с регулятором

Она имеет следующие различия:

- Принимает на вход и обрабатывает дополнительный вектор состояния регулятора, что позволяет моделировать совместную динамику системы и регулятора;
- В возвращаемом векторе производных теперь присутствуют не только производные основных переменных, но и производные состояния регулятора, что позволяет моделировать и анализировать совместную эволюцию всей системы и регулятора.

Так же были внесены поправки в функцию моделирования, в частности на вход податься расширенный вектор состояния регулятора, включающий производные состояния регулятора (рис. 5.6.2).

```
def model_reg_(  
    func,  
    theta0,  
    dtheta0,  
    ts=10, nt=0.1, x0 = 0.0, dx0 = 0.0  
):  
    """  
    func - функция управления  
  
    theta0 - начальный угол  
  
    dtheta0 - начальная скорость изменения угла  
  
    ts - время  
  
    nt - шаг дискретизации времени  
  
    x0 - начальная точка коретки  
  
    dx0 - начальная скорость коретки  
    """  
    initial_conditions = np.array([theta0, dtheta0, x0, dx0, 0, 0, 0, 0])  
    t = np.arange(0, ts, nt)  
    sol = odeint(func, initial_conditions, t)  
    return sol
```

Рис. 5.6.2 - Функция моделирования нелинейной системы с регулятором

В ходе моделирования был получен следующий график (рис. 5.6.3). Начальные условия ($\theta_0 = 0.07$, $d\theta_0 = 0$, $x = 0$, $y = 0$).

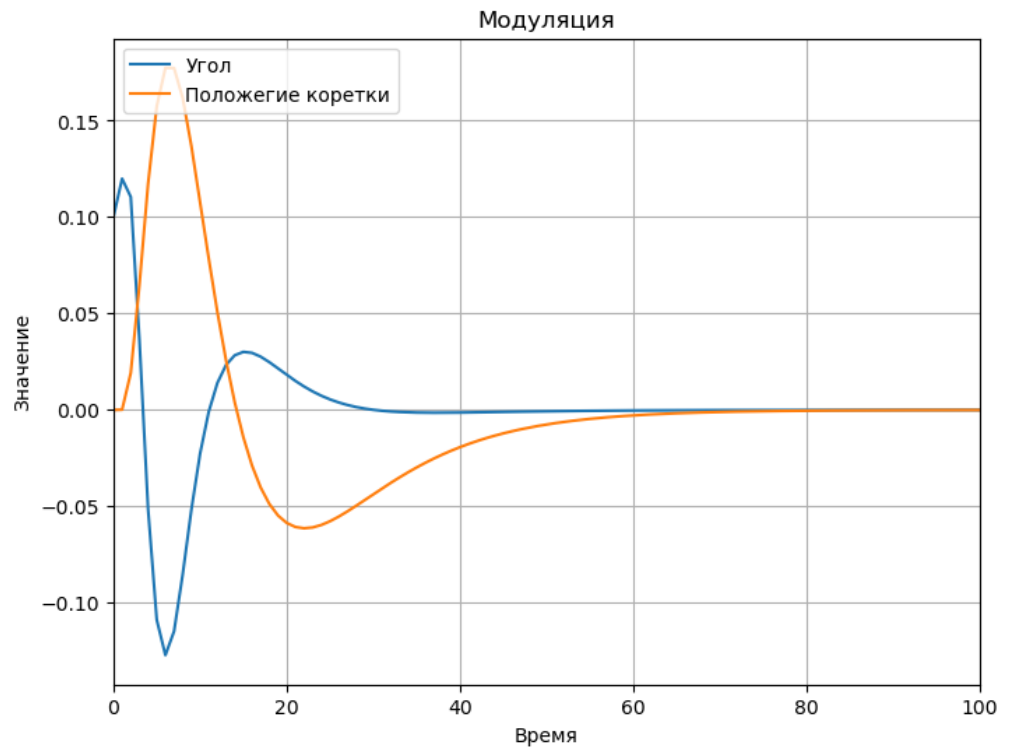


Рис. 5.6.3 – Модуляция нелинейной системы с регулятором

Оценим размеры области притяжения положения равновесия путём многократных компьютерных экспериментов. Приведены результаты для максимального начального отклонения маятника, когда линейный динамический регулятор способен стабилизировать нелинейный объект.

ЗАКЛЮЧЕНИЕ

В ходе работы достигнута основная цель — синтезированы системы стабилизации перевернутого маятника.

Была написана программа для расчета регуляторов.

Получены нелинейные уравнения динамики (5)–(6) и линеаризованная модель в форме пространства состояний (9). Анализ показал неустойчивость положения равновесия (полюсы: $[0, 0, \pm 7.67]$).

Операторный метод: Регулятор стабилизирует систему при $\theta < 0.08$ рад, но требует высокой точности расчетов. Метод пространства состояний: Динамический регулятор с наблюдателем обеспечил большую область устойчивости и робастность.

Моделирование: Подтверждено, что: без регулятора система неустойчива (рис. 2.3.2), оба регулятора стабилизируют маятник, но динамический регулятор эффективнее (рис. 4.3.3, 5.6.3).

Метод пространства состояний предпочтительнее благодаря управлению всеми переменными состояния и учету ограничений на измеряемые параметры, в нашем случае только положение каретки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Д. Х. Имаев, УЧЕБНОЕ ПОСОБИЕ по курсовому проектированию по дисциплине «Теория автоматического управления» (Электронный вариант), СПбГЭТУ «ЛЭТИ», 2011

ПРИЛОЖЕНИЕ

КОД ПРОЕКТА

```
# %%
import numpy as np
import sympy
import scipy

import matplotlib.pyplot as plt
from scipy.integrate import odeint
from scipy.signal import tf2ss

import control as ct

from tqdm import tqdm

# %%
M = 0.4 # Масса тележки
m = 0.2 # Масса маятника
l = 0.25 # Длина маятника
g = 9.8 # Ускорение свободного падения
f = 0.0 # Начальная сила

# %% [markdown]
# # Уравнения объекта в форме Коши

# %% [markdown]
# ![image.png](attachment:image.png)
# ![image-2.png](attachment:image-2.png)

# %%
def koshi(y, t):
    """
    y - вектор состояния
    theta - угол
    dtheta - скорость изменения угла
    x - положение в пространстве
    dx - скорость коретки
    t - время
    """
    theta, dtheta, x, dx = y
    sin_theta = np.sin(theta)
    cos_theta = np.cos(theta)

    denominator = (M + m)*l - m*l*cos_theta**2

    ddtheta = ((M + m)*g*sin_theta - m*l*dtheta**2*sin_theta*cos_theta -
f*cos_theta) / denominator
    ddx = (m*l**2*dtheta**2*sin_theta + l*f - m*g*l*sin_theta*cos_theta) /
denominator
```

```

        return [dtheta, ddtheta, dx, ddx]

# %% [markdown]
# ### Моделируем

# %%
def model(
    func,
    theta0,
    dtheta0,
    ts=10, nt=0.1, x0 = 0.0, dx0 = 0.0
):
    """
    func - функция управления

    theta0 - начальный угол

    dtheta0 - начальная скорость изменения угла

    ts - время

    nt - шаг дискретизации времени

    x0 - начальная точка коретки

    dx0 - начальная скорость коретки
    """
    initial_conditions = [theta0, dtheta0, x0, dx0]
    t = np.arange(0, ts, nt)
    sol = odeint(func, initial_conditions, t)
    return sol

# %%
plt.figure(figsize=(10, 6))

radian_line = np.arange(
    np.radians(-180),
    np.radians(180),
    np.radians(10)
)

for theta0 in tqdm(radian_line):
    for dtheta0 in np.arange(-10, 10, 1):
        sol = model(koshi, theta0, dtheta0, 1.5, 0.1)
        plt.plot(sol[:, 0], sol[:, 1], 'b', alpha=0.1)

plt.title('Фазовый портрет')
plt.xlabel('θ')
plt.ylabel('dθ/de')

```

```

plt.title(' dθ/dt')
plt.grid()

plt.show()

# %% [markdown]
# # Моделирование физ модели

# %%
stats = model(koshi, 1, 0, 100, 0.1)

# %%
plt.figure(figsize=(8, 6))
plt.plot(stats[:, 0], label = 'Угол')
plt.plot(stats[:, 2], label = 'Положение коретки')

plt.xlim(0, 100)

plt.legend(loc = 'upper left')

plt.xlabel('Время')
plt.ylabel('Значение')
plt.title('Модуляция')
plt.show()

# %% [markdown]
# # Линеризация уравнений

# %% [markdown]
# ![image.png](attachment:image.png)
# ![image-2.png](attachment:image-2.png)

# %%
A = np.array([
    [0, 1, 0, 0],
    [(M+m)*g/(l*M), 0, 0, 0],
    [0, 0, 0, 1],
    [-m*g/M, 0, 0, 0]
])

B = np.array([
    [0], [-1/(l*M)], [0], [1/M]
])

C = np.array([[0, 0, 1, 0]])
D = np.array([[0]])

C_theta = np.array([[1, 0, 0, 0]])

# %% [markdown]

```

```

# ## Передаточная функция

# %% [markdown]
# ### Создаем систему

# %%
sys = ct.ss(A, B, C, D)
sys

# %%
G = ct.ss2tf(sys)
G

# %%
A_s = G.num[0][0]
A_s

# %%
B_s = G.den[0][0]
B_s

# %% [markdown]
# ## Наблюдаемость

# %% [markdown]
# ### Наблюдаемость по углу
#

# %%
nabl = ct.obsv(A, C_theta)
print(f"Размер матрицы наблюдаемости {nabl.shape}")
print(f"Ранг матрицы наблюдаемости: {np.linalg.matrix_rank(nabl)}")

# %% [markdown]
# Система не наблюдаема по углу

# %% [markdown]
# ### Наблюдаемость по положению коретки

# %%
nabl = ct.obsv(A, C)
print(f"Размер матрицы наблюдаемости {nabl.shape}")
print(f"Ранг матрицы наблюдаемости: {np.linalg.matrix_rank(nabl)}")

# %% [markdown]
# Система наблюдаема по положению коретки

# %% [markdown]
# ## Управляемость

```

```

# %%
upr = ct.ctrb(A, B)

print(f"Размер матрицы управляемости {nabl.shape}")
print(f"Ранг матрицы управляемости: {np.linalg.matrix_rank(upr)}")

# %% [markdown]
# Система управляема

# %% [markdown]
# # Анализ устойчивости положения равновесия

# %% [markdown]
# ### Полюса и нули

# %%
poles = ct.poles(sys)
poles

# %%
zeros = ct.zeros(sys)
zeros

# %%
plt.figure(figsize=(6, 4))
plt.scatter(np.real(poles), np.imag(poles), marker='x', color='g', s=50,
label='Полюс')
plt.scatter(np.real(zeros), np.imag(zeros), marker='o', color='b', s=50,
label='Ноль')

plt.legend()

plt.axhline(0, color='k', linestyle='--', linewidth=0.5)
plt.axvline(0, color='k', linestyle='--', linewidth=0.5)
plt.grid(True, linestyle='--', alpha=0.5)

plt.title('Карта полюсов и нулей системы', pad=20)
plt.xlabel('Re', labelpad=10)
plt.ylabel('Im', labelpad=10)
plt.legend()

# %% [markdown]
# Система не устойчива

# %% [markdown]
# # Синтез регулятора

# %% [markdown]
# ### Задаем переменные

```

```

# %%
from sympy.abc import s
from sympy.physics.control.lti import TransferFunction as tf
from sympy.physics import control as sct

# %%
a_1, a_2, a_3, a_0 = sympy.symbols('a_1, a_2, a_3, a_0')
b_1, b_2, b_0 = sympy.symbols('b_1, b_2, b_0')

# %% [markdown]
# ### Исходная функция системы

# %%
A_o = 1/M*(s**2-g/l)
B_o = (s**2*(s**2-(M+m)/(l*M)*g))
W_o = A_o/B_o
W_o

# %% [markdown]
# ### Функция регулятора

# %%
A_p = (a_3*s**3+a_2*s**2+a_1*s+a_0)
B_p = (s**3+b_2*s**2+b_1*s+b_0)
W_p = A_p/B_p
W_p

# %%
A_s = A_o*B_p
B_s = A_o*A_p + B_o*B_p
W_s = sympy.together(A_s/B_s)
W_s

# %%
a = [-0.2, -0.4, -1, -1.2247, -2, -4, -8]
polynom = sympy.prod(sympy.Array([s]**7)-sympy.Array(a))
polynom = sympy.cancel(polynom)
#polynom = (s + 1)**7
polynom

# %%
eq = sympy.Eq(B_s, polynom)
eq

# %%
solv = sympy.solve(eq, [a_1, a_2, a_3, a_0, b_1, b_2, b_0])
solv

# %%
W_s = sympy.together(A_s/B_s).subs(solv)

```

```

# %%
a_0, a_1, a_2, a_3, b_0, b_1, b_2 = solv.values()

# %% [markdown]
# ### Передаточная функция регулятора

# %%
W_p = tf((a_3*s**3+a_2*s**2+a_1*s+a_0) ,(s**3+b_2*s**2+b_1*s+b_0), var = s)
W_p

# %%
W_o = tf(1/M*(s**2-g/l), (s**2*(s**2-(M+m)/(l*M)*g)), var = s)
W_o

# %% [markdown]
# ## Анализ линейной системы

# %% [markdown]
# ### Передаточная функция замкнутой системы

# %%
W_s_ = sct.Feedback(W_o, W_p).doit(cancel=True, expand=True)
W_s_

# %% [markdown]
# ### Собственные значения замкнутой системы

# %%
W_s_den = W_s_.den
W_s_den

# %%
eq = sympy.Eq(W_s_den, 0)
eq

# %%
solv = sympy.solve(eq, s)
solv

# %% [markdown]
# С учетом погрешности корни совпадают

# %% [markdown]
# ## Компьютерное моделирование нелинейной системы
#
#

# %% [markdown]
# ### Матрица параметров

```



```

# %%
W_p_num = np.array(sympy.Poly(W_p.num, s).all_coeffs()).astype(np.float32)
W_p_den = np.array(sympy.Poly(W_p.den, s).all_coeffs()).astype(np.float32)

# %%
A_, B_, C_, D_ = tf2ss(W_p_num, W_p_den)

# %% [markdown]
# ##### Регулятор

# %%
reg = ct.ss(A_, B_, C_, D_)
reg

# %% [markdown]
# ##### Система

# %%
sys

# %% [markdown]
# ##### Система с регулятором

# %%
sys_reg = ct.feedback(sys, reg)
sys_reg

# %% [markdown]
# ### Функции моделирования

# %%
def regul_sys(y, t):
    """
    y - вектор состояния
    theta - угол
    dtheta - скорость изменения угла
    x - положение в пространстве
    dx - скорость коретки
    t - время
    """
    theta, dtheta, x, dx = y[0:4]
    reg_vect = y[4:]

    error_x = 0 - x
    f = (reg.C @ reg_vect + reg.D * error_x)[0][0]

    sin_theta = np.sin(theta)
    cos_theta = np.cos(theta)

```

```

denominator = (M + m)*l - m*l*cos_theta**2

ddtheta = ((M + m)*g*sin_theta - m*l*dtheta**2*sin_theta*cos_theta -
f*cos_theta) / denominator
ddx = (m*l**2*dtheta**2*sin_theta + l*f - m*g*l*sin_theta*cos_theta) /
denominator

dreg_vec = reg.A@reg_vect+reg.B.reshape(-1)*error_x

return np.concatenate([[dtheta, ddtheta, dx, ddx], dreg_vec])

# %%
def model_reg(
    func,
    theta0,
    dtheta0,
    ts=10, nt=0.1, x0 = 0.0, dx0 = 0.0
):
    """
    func - функция управления

    theta0 - начальный угол

    dtheta0 - начальная скорость изменения угла

    ts - время

    nt - шаг дискретизации времени

    x0 - начальная точка коретки

    dx0 - начальная скорость коретки
    """
    initial_conditions = np.array([theta0, dtheta0, x0, dx0, 0, 0, 0])
    t = np.arange(0, ts, nt)
    sol = odeint(func, initial_conditions, t)
    return sol

# %% [markdown]
# ### Модуляция

# %%
stats = model_reg(regul_sys, 0.08, 0, 100, 0.1)

# %%
plt.figure(figsize=(8, 6))
plt.plot(stats[:, 0], label = 'Угол')
plt.plot(stats[:, 2], label = 'Положение коретки')

```

```

plt.legend(loc = 'upper left')
plt.xlabel('Время')
plt.ylabel('Значение')
plt.xlim(0, 400)
plt.grid()
plt.title('Модуляция')
plt.show()

# %%
stats = model_reg(regul_sys, 0.09, 0, 100, 0.1)

# %%
plt.figure(figsize=(8, 6))
plt.plot(stats[:, 0], label = 'Угол')
plt.plot(stats[:, 2], label = 'Положение коретки')

plt.legend(loc = 'upper left')
plt.xlabel('Время')
plt.ylabel('Значение')
plt.xlim(0, 400)
plt.grid()
plt.title('Модуляция')
plt.show()

# %% [markdown]
# # Синтез системы стабилизации перевернутого маятника

# %% [markdown]
# ## Метод размещения собственных значений

# %% [markdown]
# ### Линеаризация модели

# %%
A = np.array([
    [0, 1, 0, 0],
    [(M+m)*g/(l*M), 0, 0, 0],
    [0, 0, 0, 1],
    [-m*g/M, 0, 0, 0]
])

B = np.array([
    [0], [-1/(l*M)], [0], [1/M]
])

C = np.array([[0, 0, 1, 0]])
D = np.array([[0]])

# %% [markdown]
# ### Собственные значения системы

```

```

# %%
eig_A = np.linalg.eig(A)[0]
eig_A

# %% [markdown]
# ### Желаемые значения

# %%
p = np.array([
    -1, -2, -4, -7
])
p

# %%
K = ct.place(A, B, p)
K

# %% [markdown]
# ## Проведем анализ устойчивости системы

# %%
np.linalg.eig(A-B@K)[0]

# %% [markdown]
# Замкнутая система имеет желаемое расположение собственных значений.

# %% [markdown]
# ### Наблюдатель состояния

# %%
np.linalg.matrix_rank(ct.observ(A, C))

# %% [markdown]
# система наблюдаема

# %%
po = 5*p
po

# %%
L = ct.place(A.T, C.T, po).T
L

# %% [markdown]
# ## Динамический регулятор

# %%
def dynamic_reg(A, B, C, D, K, L):
    Ar = A - B @ K - L @ C
    Br = L

```

```

    Cr = -K
    Dr = np.zeros((K.shape[0], C.shape[0]))

    return Ar, Br, Cr, Dr

# %%
Ar, Br, Cr, Dr = dynamic_reg(A, B, C, D, K, L)

# %% [markdown]
# ### Регулятор

# %%
regl = ct.ss(Ar, Br, Cr, Dr)
regl

# %%
np.linalg.eig(Ar)[0]

# %% [markdown]
# Регулятор неустойчив – имеется положительное собственное значение

# %% [markdown]
# ### Передаточная функция регулятора

# %%
W_reg = ct.ss2tf(regl)

# %% [markdown]
# ### Анализ системы с динамическим регулятором

# %%
sys_reg_ = ct.feedback(sys, regl, sign=1)
sys_reg_

# %%
eig = np.linalg.eig(sys_reg_.A)
print(f"Собственные числа системы: {eig[0]}")

# %%
def regul_sys_(y, t):
    """
    y - вектор состояния
    x - положение в пространстве
    theta - угол
    dtheta - скорость изменения угла
    dx - скорость коретки
    t - время
    """
    theta, dtheta, x, dx = y[0:4]
    reg_vect = y[4:]

```

```

y_meas = x

f= -K @ reg_vect

sin_theta = np.sin(theta)
cos_theta = np.cos(theta)
denominator = (M + m)*l - m*l*cos_theta**2
ddtheta = ((M + m)*g*sin_theta - m*l*dtheta**2*sin_theta*cos_theta -
f*cos_theta) / denominator
ddx = (m*l**2*dtheta**2*sin_theta + l*f - m*g*l*sin_theta*cos_theta) /
denominator

dreg_vec = A @ reg_vect + B.flatten()*f + L.flatten()*(y_meas - reg_vect[2])

return np.concatenate([[dtheta, ddtheta[0], dx, ddx[0]], dreg_vec])

# %%
def model_reg_(
    func,
    theta0,
    dtheta0,
    ts=10, nt=0.1, x0 = 0.0, dx0 = 0.0
):
    """
    func - функция управления

    theta0 - начальный угол

    dtheta0 - начальная скорость изменения угла

    ts - время

    nt - шаг дискретизации времени

    x0 - начальная точка коретки

    dx0 - начальная скорость коретки
    """
    initial_conditions = np.array([theta0, dtheta0, x0, dx0, 0, 0, 0, 0])
    t = np.arange(0, ts, nt)
    sol = odeint(func, initial_conditions, t)
    return sol

# %%
stats = model_reg_(regul_sys_, 0.1, 0, 100, 0.1)

# %%
plt.figure(figsize=(8, 6))

```

```
plt.plot(stats[:, 0], label = 'Угол')
plt.plot(stats[:, 2], label = 'Положение коретки')

plt.legend(loc = 'upper left')
plt.xlim(0, 100)
plt.xlabel('Время')
plt.ylabel('Значение')
plt.grid()
plt.title('Модуляция')
plt.show()

# %%
```