Санкт-Петербургский государственный университет

Прикладная математика и информатика

Динамические системы, эволюционные уравнения, экстремальные задачи и математическая кибернетика

Барабанова Александра Сергеевна

Методы обучения с подкреплением в задачах управления механическими системами

Магистерская диссертация

Научный руководитель: к. ф.-м. н., доцент Ананьевский М.С.

Рецензент: д. т. н. Фуртат И. Б.

Санкт-Петербург

2019

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Computer Science

Dynamical systems, evolution equations, extremal problems and mathematical cybernetics

Alexandra Barabanova

Methods of learning with reinforcement in the control problems of mechanical systems

Master's Thesis

Scientific supervisor: Mikhail Ananyevskiy

Reviewer: Igor Furtat

Saint-Petersburg

2019

ОГЛАВЛЕНИЕ

Введение

Одним из популярных направлений науки в настоящее время является искусственный интеллект. Одной из важных составляющих этого направления – это машинное обучение. Оно применяется во многих областях, например в информационной безопасности, робототехники, медицине. Почему его применяют? Дело в том, что системы машинного обучения позволяют быстро применять знания, полученные при обучении на больших наборах данных, что позволяет им преуспевать в таких задачах, как распознавание лиц, распознавание речи, распознавание объектов, автоматический перевод текстов, оно помогает в управлении самолетами, ракетами, в диагностировании заболеваний и многое другое. В отличие от программ с закодированными вручную инструкциями для выполнения конкретных задач, машинное обучение позволяет системе научиться самостоятельно, распознавать шаблоны и делать прогнозы. В некоторых случаях машина по сравнению с человеком способна принимать более правильные, быстрые, решения.[1] Но эта область имеет множество нерешенных проблем. В машинном обучении выделяют такие направления, как: обучение по дереву принятия решений, обучение по ассоциативным правилам, обучение с подкреплением, искусственные нейронные сети, глубокое обучение, обучение по характерным особенностям, обучение по подобию, обучение по избыточному словарю и многое другое. [2] Данная работа относится к области обучения с подкреплением, а точнее к адаптивным алгоритмам. Обучение с подкреплением используют там, где задачей стоит не анализ данных, а выживание в реальной среде. Откликом среды (а не специальной системы управления подкреплением, как это происходит в обучении с учителем) на принятые решения являются сигналы подкрепления, поэтому такое обучение является частным случаем обучения с учителем, но учителем является среда или её модель. С точки зрения кибернетики, обучение с подкреплением является одним из

видов кибернетического эксперимента. В инженерных задачах управления механическими и техническими системами, самообучение и самонастраивание регулятора последние 50 лет изучается под флагом адаптивного управления.

Целью данной работы является изучение методов машинного обучения с подкреплением для задач управления механической системой на примере робота-велосипедиста.

Робот-велосипедист представляет собой велосипед с находящемся на нем регулятором. Последний должен поддерживать равновесие движущегося велосипеда. Возможно ли реализовать такой регулятор? Наверное, поскольку человек может ездить на велосипеде и регулировать его таким образом, что бы тот не упал, есть также уже и реализованные роботы данного класса. Любая задача автоматического управления превращается в задачу об адаптивном управлении. Содержание подобных задач весьма разнообразно, например, управление подвижными объектами, технологическими процессами, управление робототехническими системами. Адаптивные регуляторы обладают большой надежностью и устойчивостью в работе, ввиду способности компенсировать разного рода неточности (неизвестные параметры модели, внешние возмещения). Если адаптивный регулятор построен на основании неточно составленной математической модели объекта, но реальный объект принадлежит классу адаптации регулятора, то он будет работоспособен.[3]

Работа основана на статье В. А Якубовича «Адаптивные системы с многошаговыми целевыми условиями» в применении ее теории к прикладной задаче. Статья является довольно старой по написанию и структуре, поэтому при ее разборе возникает большое количество вопросов и затруднений. Поэтому было решено разобрать данную статью и написать прикладную часть таким образом, чтобы человеку, который хочет применить данный алгоритм, не пришлось сталкиваться свыше обозначенными

проблемами и тратить на них время. Так же в этой работе представлена программная реализация этого алгоритма.

Моделирование выполнено на языке Python[5], так как он сегодня является одним из самых распространенных языков и имеет большое количество библиотек помогающих в реализации, визуализации, вычислениях. При разработке использовались библиотеки: matplotlib – визуализация данные двумерной графикой; numpy[7] - расширение языка Python, добавляющее поддержку больших многомерных массивов и матриц (которые в нашем случае нужны для визуализации получаемых нами данных), вместе с большой библиотекой высокоуровневых математических функций для операции с этими массивами; math[6] – это модуль предоставляет обширный функционал для работы с числами.

Структура работы

Во введении обосновывается актуальность решаемой задачи, анализируются труды по данной тематике, ставится цель и задачи исследования, а также основные направления исследования.

В первой главе описывается задача, которая поставлена в статье[3]. Также обозначены проблемы, которые возникали во время ее изучения. Описывается концепция работы и дается математическая постановка задачи.

Во второй главе представлена программная реализация моделирования с алгоритмом и без него, графики для обоих случаев и описание программ.

В заключении делается вывод о полученных результатах исследования и приводится список литературы.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

1.1. Задача управления «роботом-велосипедистом»

Рассмотрим эксперимент, представленный в статье[4]. Пусть объектом управления является велосипед, который движется с постоянной скоростью (велосипед с мотором). Без управления он будет падать, особенно если есть внешние воздействия. Задача состоит в том, что бы построить регулятор, то есть «робот-велосипедист», который должен поддерживать равновесие движущегося велосипеда. В ряде случаев, если использовать традиционную теорию управления, решение оказывается неудовлетворительным, это показано в книге “Адаптивное управление динамическими объектами”[3].

Для определенности уточним, что «робот-велосипедист» представляет собой устройство, на вход которого поступает сигнал измерителя угла между плоскостью рамы велосипеда и вертикальной плоскостью, а выходом является сигнал, идущей на устройство поворота руля и указывающий нужный угол поворота руля.

Регулирование равновесия «робота-велосипеда» производится за счет изменения значения переменных в уравнении управления велосипедом. Эти переменные изменяются как раз по представленному алгоритму в статье[3]. Что бы его было возможно применить надо, что бы выполнялись все условия теоремы этой статьи. Если «робот-велосипедист» разумен, то число падений будет конечно и робот обучится за конечное время. Время, которое робот должен продержаться без падений задается человеком. Если робот упал в промежутке от начала игры до окончания данного нами времени, то изменяются переменные в уравнении управления велосипедом и игра снова возобновляется. Переменные, которые подбираются для равновесия робота, в статье называют «тактикой». Если они равны константе и робот не падает,

начиная с какого-то момента времени, то он разумен. Уравнения управления и тактики должны быть построены так, что бы робот стал разумным в указанном классе задач.

Целью является обучение робота кататься на велосипеде без падений. Для этого необходимо по данным о наклоне рамы относительно вертикальной плоскости, определять угол поворота руля велосипеда.

Замечание: задача заключается только в обучении робота кататься на велосипеде без падений, задача приехать в заданную точку не ставится.

1.2. Проблемы практического применения теоретических результатов

Как было отмечено во введении, статья является довольно старой по написанию и структуре, поэтому при ее разборе возникает большое количество вопросов и затруднений. В частности:

1. Нет начальных значений для некоторых переменных;
2. Не все параметры определены и описаны с точки зрения объекта управления;
3. В некоторых моментах можно столкнуться с такой проблемой, что не сказано, какому пространству принадлежат некоторые параметры и функции;
4. Поскольку за рамками статьи оставлены выводы математической модели, то непонятно как соотносятся параметры уравнения с велосипедом, в частности, не указаны единицы измерения;
5. Непонятно как определить значения некоторых параметров в задаче, так как не сказано какие из них известны, а какие нет;
6. Неясно так же как определяются новые параметры, где описывается движение для одной игры. (В данном случае под понятием «игра» понимаем

промежуток времени от начала движения велосипеда до его падения, либо пока не закончится время, которое мы давали ему на движение);

7. Описание теоремы очень общее, некоторые параметры не определены, а ведь они используются в алгоритме управления. В теореме говорится о существовании параметров, при которых алгоритм работает, но не говорится, как эти параметры найти.

8. В статье многие утверждения оформлены не формулами, а словесным описанием, что допускает обширное толкование. Если человек связан с математическими дисциплинами, то ему не составит труда разобраться. Но есть так же люди связанные с естественными науками, инженеры и многие другие, которые могут нуждаться в применении данного алгоритм для своей деятельности, и здесь у них может возникнуть сложность с обоснованностью данных утверждений;

9. В статье можно наблюдать присутствие перегруженных символов, к примеру, таковыми являются ξ_1, ξ_2, ξ_3 – это параметры системы, которые зависят от скорости, конструктивных параметров и частоты дискретизации. Но не ясно, например как они зависят от скорости, об этом написали, но не объяснили как.

Что касается теоремы, в которой как раз описывается нужный нам алгоритм, то в ней очень много условий, вводится большое количество новых переменных и неравенств. Так же указывается много ссылок на литературу, в которой находятся нужные данные, и ее приходится искать, а она в более распространённых базах данных (Web of Science, Scopus, elibrary) отсутствует.

Сложно за разумное время разобраться с данными проблемами и написать алгоритм решения задачи, который, будет теоретически обоснован в данной статье.

1.3. Математическая постановка задачи

I. Уравнение велосипеда:

Уравнение движения велосипеда можно описать следующей дискретной моделью вида [3](так же оно описывается в [11,12]):

$$\chi_{t+1} = \xi_1\chi_t + \xi_2\chi_{t-1} + \xi_3\psi_t + \varphi_t, \quad (1)$$

χ – угол между плоскостью рамы велосипеда и вертикальной плоскостью,

ψ – угол поворота руля,

φ_t – неизвестное внешнее воздействие $|\varphi_t| \leq$ Ф,

$\{\xi_1, \xi_2, \xi_3\}$ – варьируемые параметры, зависящие от скорости, конструктивных параметров велосипеда и частоты дискретизации.

Предполагается, что $0 < \xi_i \leq \varkappa_i$, где $\{\varkappa_1, \varkappa_2, \varkappa_3\}$ – известны.

Требуется найти управление велосипедом в форме обратной связи[3]:

$$\psi_t = -\gamma_1\chi_t - \gamma_2\chi_{t-1}, \quad (2)$$

то есть, требуется найти коэффициенты γ_1, γ_2 так, чтобы велосипед не падал, т.е. чтобы $\chi_t <$ r для всех $t \leq \mathrm{T}_0$.

II. Опишем движение, которое происходит за одну игру. В данном случае под понятием «игра» понимаем промежуток времени от начала движения велосипеда t_0 до его падения, либо пока не закончится время, которое мы давали ему на движение T_0.

$\chi_0 = 0$ – в начальный момент движения t_0 велосипед устанавливается вертикально,

$\{\xi_1, \xi_2\}$ – задаются с учетом ограничений:

$$|\xi_0| \leq \delta,$$

$$|\xi_1| \leq \delta$$

Эволюция системы рассчитывается до тех пор, пока:

$$|\chi_t| < \mathrm{r},$$

$$t \leq \mathrm{T}_0$$

где r – угол, при котором велосипед падает,

T_0 – максимальное время игры.

Так же вводятся μ_t – сигнал или флаг активности игры(1 – игра идет, 0 – не идет). Δ_j – интервал игры, то есть:

Первая игра: $t \in \Delta_1, \mu_t = 1$

Вторая игра: $t \in \Delta_2, \mu_t = 1$

и т.д.

Замечания по применению статьи:

1. Непонятно, как определяются параметр δ;
2. Некоторые утверждения в статье представлены в виде слов, а не формул, например, что $\{\xi_1, \xi_2, \xi_3\}$ – зависят от скорости, конструктивных параметров и частоты дискретизации;
3. ξ_1, ξ_2, ξ_3 – не понятно как от скорости получить их оценку.

III. Описание алгоритма

Введем следующие параметры:

$\beta < 1,$

$\varepsilon > 0$

И функция Ляпунова V_t

$$V_t = \chi_t^2 + \beta\chi_{t-1}^2$$

Если Ф достаточно мало, то ∃ область E такая, что $(\gamma_1, \gamma_2) \in E \rightarrow (\forall t: V_{t+1} \geq \varepsilon^2\beta \rightarrow V_{t+1} < V_t)$. Область E зависит от $\{\xi_1, \xi_2, \xi_3\}$.

Область $V_{t+1} \geq \varepsilon^2\beta$ – инвариантная область притяжения для (χ_t, χ_{t+1}).

$V_{t+1} \geq \varepsilon^2\beta$ – целевое условие

Алгоритм “Полоска”: $\psi_t = \tau_t^{(1)}\chi_t + \tau_t^{(2)}\chi_{t-1}$, коэффициенты $\tau_t^{(1)}$, $\tau_t^{(2)}$ пересчитываются по правилу:

$\tau_{t+1} = \tau_t$, если выполнено одно из условий:

1. $V_{t+1} < \varepsilon^2\beta$
2. $V_{t+1} \geq \varepsilon^2\beta$ и $V_{t+1} < V_t$
3. $\chi_t = \chi_{t-1} = 0$(т.е $\psi_t = 0$)
4. $\mu_t = 0$

иначе:

$$\tau_{t+1}^{(1)} = \tau_t^{(1)} - \zeta_t\chi_t,$$

$$\tau_{t+1}^{(2)} = \tau_t^{(2)} - \zeta_t\chi_{t-1},$$

где $\zeta_t = \frac{\chi_{t+1}}{(\chi_t^2 + \chi_{t-1}^2)\varkappa_3}$, $V_t = \chi_t^2 + \beta\chi_{t-1}^2$, $\beta < 1$, $\varepsilon > 0$ (β, ε – параметры алгоритма)

Замечания:

1. Не понятно чему равны $\varepsilon^2\beta$, Ф. Их ограничения $\beta < 1$, $\varepsilon > 0$ не дают четкой картины, на сколько эти параметры могут быть большие и наоборот. Про Ф совсем ничего не сказано.

IV. Теорема

Теорема сформулированна следующим образом:

Пусть выполнены условия ($П_1$) с k = 1 ($П_2$’), ($П_3$), ($П_4$)[3]. Пусть число

N и «нейтральные функции» $\nu_j(\sigma), j = 1, \dots, N$, определяется по $V^{\text{И}}(\sigma, \xi)$ так, как указано ниже в доказательстве теоремы. Положим $\{\tau\} = R_N, \tau = \left\|\tau^{(j)}\right\|_{j=1}^{N}$. Определим уравнение управления соотношениями $u_t = u(\upsilon_t), \upsilon_t = \tau_t^{(1)}\nu_1(\sigma_t) + \cdots + \tau_t^{(N)}\nu_N(\sigma_t)$, а уравнение $\tau_{t+1} = A(\sigma_t, \sigma_{t+1}, \tau_t)$ соотношениями $\tau_{t+1} = \tau_t$, если $\left|\gamma_t^{(1)}\right| < \varepsilon_t^{(1)}$ или $\mu_t = 0$; $\tau_{t+1} = \tau_t - \zeta_t a_t$, $\zeta_t = \frac{\gamma_t^{(1)}}{\varkappa^{(1)}|a_t|^2}, a_t = \left\|\left(c_1, \nu_j(\sigma_t)\right)\right\|_{j=1}^{N}$, если $\left|\gamma_t^{(1)}\right| > \varepsilon$ и $\mu_t = 1$. Тогда робот будет разумен в классе задач M.

Условия (Π_1), (Π_2'), (Π_3), (Π_4) даны в статье [3].

Замечания:

1. В (Π_1) определяется значение $\gamma_t^{(h)} = (c_h, \upsilon_t)\alpha_t^{(h)} + \beta_t^{(h)}$, при этом сказано что α, β выражаются через $\xi, x_t, s_t, s_{t+1}, x_{t+1}$. Но сама формула не написана. Так же неизвестно как получить значение c_h.

Во второй главе эти параметры будут пробным путем смоделированы. Так же будет сделан вывод на основе полученных графиков и значений.

ГЛАВА 2. ИССЛЕДОВАНИЕ ЗАДАЧИ

2.1. Моделирование неуправляемой системы

Замечание: первая модель будет представлять график движения велосипеда без обучения.

Описание данных:

Входные данные (сенсоры): $\| \chi_t, \chi_{t-1} \|$

Известно: ξ_j, r, T_0, t, φ_t, δ

Неизвестно: X_t

Задание параметров:

Я задаю параметры, следующими значениями:

Пусть

t = 1, 2, 3, … T_0 – время игры;

T_0 = 50 – максимальное время игры;

r = π/3 – угол, при котором велосипед падает;

ψ_t = 0 – начальное значение управления;

ξ_0 = 2, ξ_1 = 3 (так как $|\xi_0| < \delta$, $|\xi_1| < \delta$ и задаются квазислучайно),

где δ = 4

χ = [0, π/6, …] (χ_0 = 0 и χ_1 = π/6) – угол между плоскостью рамы велосипеда и вертикальной плоскостью;

φ_t = 0 – неизвестное внешнее воздействие;

Это примерные значения. Параметры задаются таким образом, потому что с этими числами условия выполняются. Это можно будет увидеть в следующем разделе.

Уравнения:

$$\chi_{t+1} = \xi_1 \chi_t - \xi_2 \chi_{t-1} + \varphi_t$$

$$|\chi_t| < r, \; t <= T_0$$

Замечание: мы не берем уравнение $\psi_t = -\gamma_1\chi_t - \gamma_2\chi_{t-1}$, так как управление нам в данной модели не понадобится.

Общее описание реализации(python):

```
for t in range(1, T0 - 1):
    if (abs(X[t]) <= r):
       X[t + 1] = xi[0] * X[t] - xi[1] * X[t - 1] + phi
    else:
       result = - math.pi / 2 if (X[t] < 0) else math.pi / 2
       for i in range(t, T0):
          X[i] = result
          break;
print(X)

plt.plot(np.arange(T0), X)
```

Листинг 1 – часть программной реализации моделирования движения «робота-велосипедиста» без обучения.

Вначале идет объявление переменных, углы задаются в радианах (параметры r, χ), не нужные переменные в данном алгоритме обнуляются(φ_t, ψ_t). Затем следует цикл по t до T_0(t = 1, 2, 3, … T_0), в котором идет проверка условия $|\chi_t| < r$(не меньше ли текущий угол наклона, чем r), если истина, то идем дальше, иначе велосипед упал и мы выходим из цикла обозначая все оставшиеся χ крайним значением ($\pi/2$). В цикле мы считаем формулу, по которой находим значение угла отклонения плоскости рамы велосипеда от вертикальной плоскости на текущей момент $\chi_{t+1} = \xi_1 \chi_t - \xi_2 \chi_{t-1} + \xi_3\psi_t + \varphi_t$. После выхода из цикла мы строим график по значениям t и χ (матрица).

Далее можно изменять, например, начальные значения угла χ и получать другие графики для сравнения, или добавить внешнее воздействие φ_t. Так же можно изменять значения вариационных переменных ξ_1, ξ_2.

Построение задачи:

χ_t, t – изменение угла по времени.

Реализация данного моделирования носит общий характер. Оно позволяет увидеть при каких заданных параметрах, возможно, удержать устойчивость велосипеда, и убедиться в том, что он вообще падает без управления. Так же задачей стоит оценить графики и сделать вывод по тому, какие значения, какой ситуации способствуют. Полный листинг программной реализации моделирования движения «робота-велосипедиста» без обучения будет представлен в приложении к данной работе.

В следующем разделе будет графики с описанием полученных результатов.

2.2. Графики моделирования неуправляемой системы

Ниже представлены графики изменения угла наклона велосипеда от вертикали по времени. Задаваемые разные значения параметров соответствуют различным графикам. По оси x идут значения переменной времени t, по оси y указывается переменная χ_t, которая обозначает угол между плоскостью рамы велосипеда и вертикальной плоскостью. В конце раздела дается описание к каждому графику по полученному результату, в соответствии с заданными параметрами и номеру рисунка. Далее делается вывод по общим результатам данного моделирования.

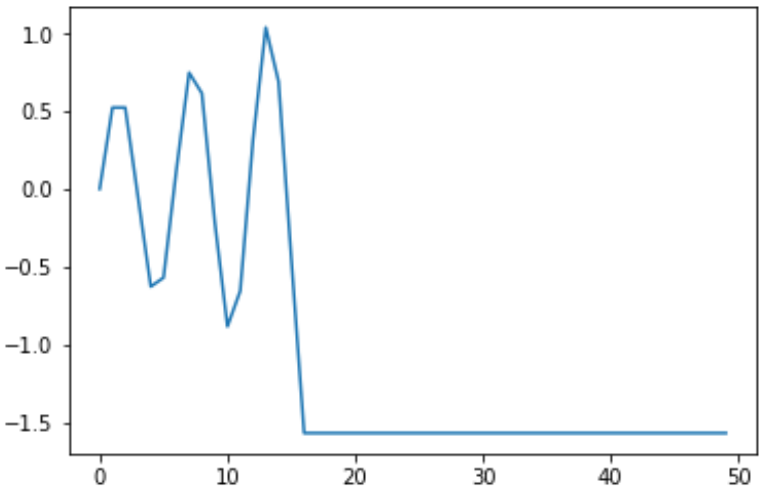

Рисунок 1 – График моделирования движения «робота-велосипедиста» без обучения со значениями параметров 1).

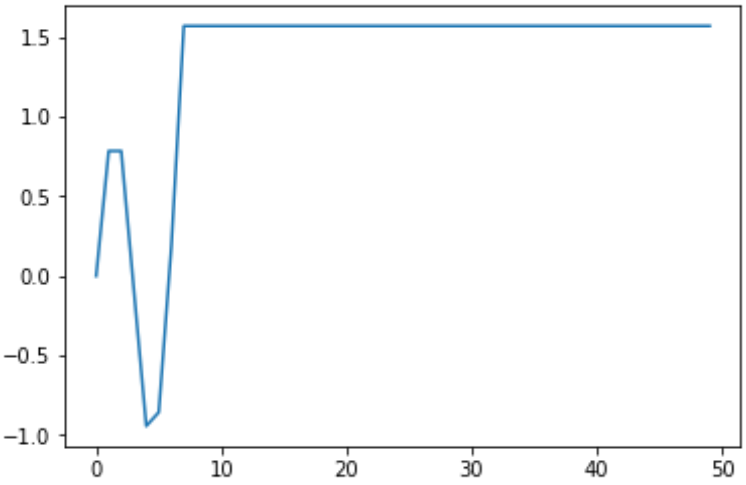

Рисунок 2 – График моделирования движения «робота-велосипедиста» без обучения со значениями параметров 2).

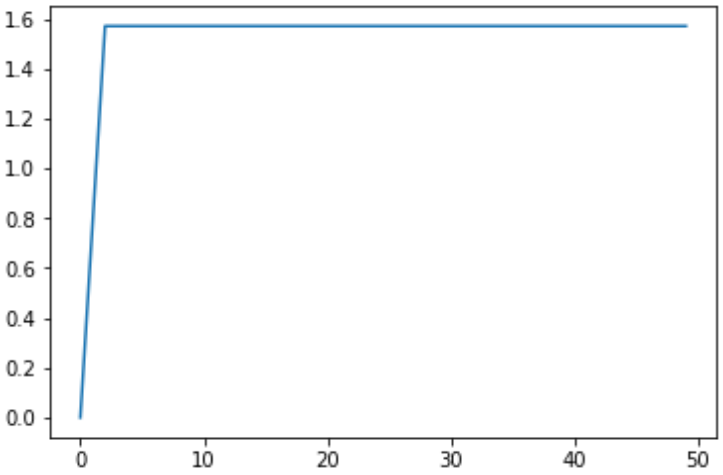

Рисунок 3 – График моделирования движения «робота-велосипедиста» без обучения со значениями параметров 3).

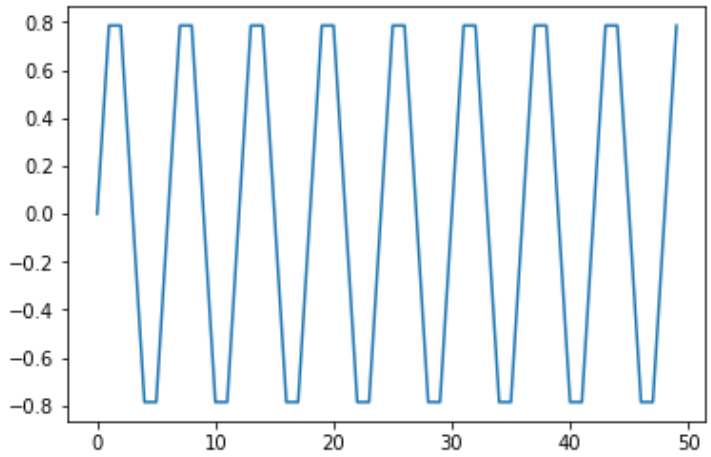

Рисунок 4 – График моделирования движения «робота-велосипедиста» без обучения со значениями параметров 4).

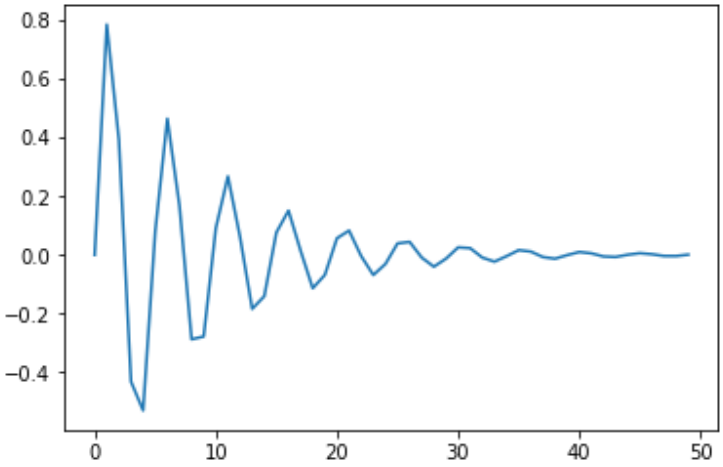

Рисунок 5 – График моделирования движения «робота-велосипедиста» без обучения со значениями параметров 5).

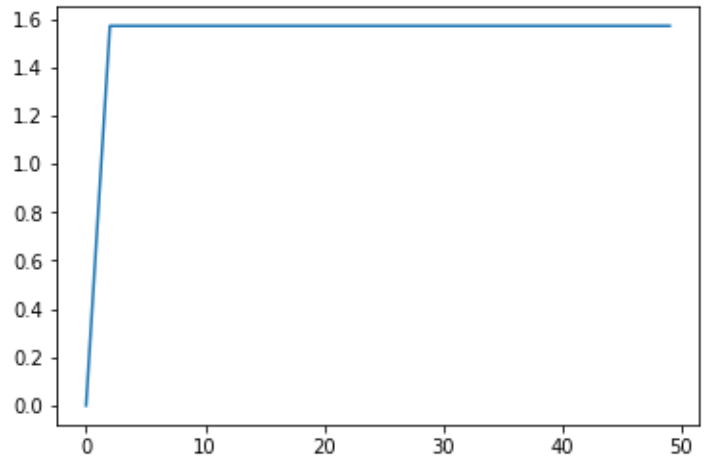

Рисунок 6 – График моделирования движения «робота-велосипедиста» без обучения со значениями параметров 6).

Описание полученных результатов:

1) При ξ = [1, 1.1], χ_0= 0, χ_1 = π/ 6, φ = 0 получаем график, показанный на рисунке 1. Мы видим, что велосипед падает не с самого начала, а какое-то время движется, увеличивая угол отклонения от вертикали (проходит три периода после чего угол становится больше r и велосипед падает).

2) При ξ = [1, 1.1], χ_0= 0, χ_1 = π / 4, φ = 0 получаем график на рисунке 3. Видим, что при увеличении угла χ_1 велосипед падает быстрее.

3) При ξ = [2, 2.5], χ_0= 0, χ_1 = π / 4, φ = 0 получаем график, представленный на рисунке 3. Здесь мы изменили параметры ξ увеличив их. Можно видеть, что при этом падение происходит практически сразу, то есть велосипед становится менее устойчивым. Так же было отмечено, что при ξ_1 = 1, и при увеличении ξ_0 до 2, равновесие сохраняется, но уменьшается частота колебания и увеличивается амплитуда. После ξ_0 = 2 происходит падение. При увеличении ξ_1 от 1 и при ξ_0 = 1, можно видеть увеличение амплитуды и сокращение времени до падения.

4) При ξ = [1, 1], χ_0= 0, χ_1 = π / 4, φ = 0 получаем график на рисунке 4. Видим, что ξ = [1, 1] возникает равновесия. Если ξ_1 = 1, то от минимального значения ξ_1 до 2, будет сохраняться это равновесие.

5) При ξ = [0.5, 0.8], χ_0= 0, χ_1 = π / 4, φ = 0 получаем график на рисунке 5. Видим как раз, что при значениях ξ < 1 угол наклона велосипеда с течением времени устремляется к устойчивому состоянию в 0. Так же установлено, то при ξ_1 < 1, и при изменении до какого-то значения ξ_0 > 1(это значение зависит от ξ_1, чем оно больше, тем больше можно указывать ξ_0 не боясь выйти из равновесия), можно удерживать состояние сходимости к равновесному состоянию.

6) Последний рисунок показывает нам случай при ξ = [1, 1.1], χ_0= 0, χ_1 = π / 4, φ = π / 10 (рисунок 6). Здесь мы задаем не нулевое внешнее воздействие, и видим, что это ускоряет падение велосипеда. В случае, если у

нас без внешнего воздействия получалось состояние равновесия, то до определенного увеличения φ, оно сохраняет это состояние, и в зависимости от величины φ может переходить в другую точку. Но так же есть некоторое граничное значение φ, после которого устойчивость теряется и велосипед падает (оно зависит значений ξ).

Можно сделать следующие выводы, что брать $\xi \leq 1$ не имеет смысла, так как будет сохраняться равновесие, а нам это не интересно. При значениях $\xi_0 < 1$ и до определенного значения ξ_1 сохраняется равновесие системы, поэтому такие значения нам тоже не очень подойдут. Поэтому мы рассмотрим случаи $\xi > 1$, и $\xi_1 > 1$ $\xi_0 < 1$, когда велосипед падает, что как раз таки предполагает использование регулятора. Так же чем больше значение χ_1 и φ, тем быстрее велосипед падает.

2.3. Моделирование управляемой системы и оценка коэффициента $\varepsilon^2\beta$

Замечание: эта модель будет представлять график движения велосипеда с обучения.

Описание данных

Входные данные (сенсоры): $\|\chi_t, \chi_{t-1}\|$

Выходные данные (управление): ψ_t

Известно модели: r, T_0, t, φ_t, δ

Известно регулятору: ξ_j, ε, β

Вычисляемые переменные: X_t, γ, ψ, V_t

Задание параметров

Задаю параметры, которые нужны в данном моделировании.

Пусть

$t = 1, 2, 3, \ldots T_0$ – время игры;

$T_0 = 30$ – максимальное время игры;

$r = \pi/3$ – угол, при котором велосипед падает;

$\psi_t = 0$ – начальное значение управления;

$\xi_0 = 2$, $\xi_1 = 3$, $\xi_2 = 1$ (так как $|\xi_0| < \delta$, $|\xi_1| < \delta$),

где $\delta = 4$

$\chi = [0, \pi/6, \ldots]$ ($\chi_0 = 0$ и $\chi_1 = \pi/6$) – угол между плоскостью рамы велосипеда и вертикальной плоскостью;

$\varphi_t = 0$ – неизвестное внешнее воздействие;

$\beta = 0.1$(т. к. по условию $\beta < 1$)

$\varepsilon = 1$.(т. к. по условию $\varepsilon > 0$ и $\varepsilon < r$)

$\varkappa = 3.$, известный параметр ($0 < \xi_i \leq \varkappa_i$)

Уравнения

$$\chi_{t+1} = \xi_1 \chi_t - \xi_2 \chi_{t-1} + \xi_3 \psi_t + \varphi_t.$$

$$|\chi_t| < r, \quad t <= T_0$$

$$\psi_t = - \gamma_1 \chi_t - \gamma_2 \chi_{t-1}$$

если $V_{t+1} \geq \varepsilon^2 \beta$, то $V_{t+1} < V_t$,

– свойство алгоритма (проверка правильно ли работает алгоритм)

где $V_t = \chi_t^2 + \beta \chi_{t-1}^2$

Условие (5) в статье: Идеальное управление для «робота-велосипедиста» определяется уравнением $\psi_t = - \gamma_1 X_t - \gamma_2 X_{t-}1$. В некоторых простых предположениях уравнение мозга велосипеда имеет вид $\psi_t = \tau^{(1)}_t$ Xt

$+ \tau^{(2)}_t X_{t-1}$; $\tau^{(j)}_{t+1} = \tau^{(j)}_t$, если или $V_{t+1} < \varepsilon^2\beta$, или $V_{t+1} \geq \varepsilon^2\beta$ и $V_{t+1} < V_t$, или $\mu_t=0$, или $X_t = X_{t-1} = 0$; В остальных случаях $\tau^{(1)}_{t+1} = \tau^{(1)}_t - \zeta_t X_t$, $\tau^{(2)}_{t+1} = \tau^{(2)}_t - \zeta_t X_{t-1}$, $\zeta_t = X_{t+1}/(Xt^2 + X^2_{t-1})\varkappa_3$. В теореме не совсем понятно как соотносятся γ заданные в уравнении управления и τ. Так же в самой теореме присутствует выражение $\upsilon_t = \tau_t^{(1)}\nu_1(\sigma_t) + \cdots + \tau_t^{(N)}\nu_N(\sigma_t),\ u_t = u(\nu_1)$. В нашем случае N=2, $\nu_1(\sigma_t) = \chi_t,\ \nu_2(\sigma_t) = \chi_{t-1}$.

Построение задачи:

χ_t, t – изменение угла по времени.

ψ_t, t – изменение угла поворота руля по времени.

Описание алгоритма:

```
while (t < T0 - 2):
  t = t + 1;
  if (abs(X[t]) <= r):
    psi = gamma[0] * X[t] + gamma[1] * X[t - 1]
    X[t + 1] = xi[0] * X[t] - xi[1] * X[t - 1] + xi[2] * psi +
phi
    Vt1 = X[t + 1] ** 2 + beta * X[t]** 2
    Vt = X[t] ** 2 + beta * X[t - 1]** 2
    if ((Vt1 < (eps ** 2 * beta)) or
       (X[t] == 0 and X[t - 1] == 0) or
       ((Vt1 >= (eps ** 2 * beta)) and (Vt1 < Vt)) or (mu == 0)):
        print (color.Green + "ЦУ выполнено" + color.END)
    else:
        print (color.Yellow + "Изменяем gamma" + color.END)
        dz = cappa * X[t + 1]/(X[t - 1] ** 2 + X[t]** 2)
        gamma[0] = gamma[0] - dz * X[t]
        gamma[1] = gamma[1] - dz * X[t - 1]
        mu = 0
  else:
    mu = 1
    t = 0
plt.plot(np.arange(T0), X)
```

Листинг 2 – часть программной реализации моделирования движения «робота-велосипедиста» с обучением.

Вначале идет объявление переменных, углы задаются в радианах

(параметры r, χ), не нужные переменные в данном алгоритме обнуляются(φ_t, ψ_t). Затем следует цикл по t до T_0(t = 1, 2, 3, … T_0), в котором идет проверка условия $|\chi_t|$ < r(не меньше ли текущий угол наклона, чем r), если истина, то идем дальше, иначе велосипед упал, то начинаем цикл с начала, заменяем значение mu и обнуляя цикл. Иначе, находим ψ_t, χ_{t+1}, V_t, V_{t+1}. Далее идет условие в котором проверяется выполнение ЦУ(это $V_{t+1} < \varepsilon^2\beta$, или $V_{t+1} \geq \varepsilon^2\beta$ и $V_{t+1} < V_t$, или $\mu_t=0$), если оно не выполнено, то пересчитываем γ_1 и γ_2, обнуляем наш флаг μ. Доходим цикл до конца и строим 2 графика: 1) изменение угла по времени; 2) изменение угла поворота руля по времени.

Реализация данного моделирования носит практический характер. Мы хотим увидеть работу алгоритма представленного в статье[3] в действии, также хотим убедиться в правильности выбранных нами значений, в правильном понимании структуры работы и сделать выводы исходя из оценки полученных нами графиков.

Полный листинг программной реализации моделирования движения «робота-велосипедиста» с обучением будет представлен в приложении к данной работе.

Так же реализован код, который дает оценку барьерному значению $\varepsilon^2\beta$.

```
…
start_bord = 0
for epsbeta in range(0, 50, 1):
    if (start_bord == border) and (epsbeta != 0):
          print("Граничное значение " + str((epsbeta - 1)/10) +"
"+ str(beta) +" "+ str(eps))
          break
    start_bord = 0
…
    epsbeta_div = epsbeta / 10
    beta = epsbeta_div / 1.03
    eps = m.sqrt(1.03)

    while (t < T0 - 2):
      if (start_bord == border):
          for i in range(t + 1, T0):
            X[i] = m.pi/3.
```

```
            break
        start_bord = start_bord + 1
        t = t + 1;
        if (abs(X[t]) <= r):
          psi[t] = gamma[0] * X[t] + gamma[1] * X[t - 1]
          X[t + 1] = xi[0] * X[t] - xi[1] * X[t - 1] + xi[2] *
psi[t] + phi
          Vt1 = X[t + 1] ** 2 + beta * X[t]** 2
          Vt = X[t] ** 2 + beta * X[t - 1]** 2
          if ((Vt1 < (epsbeta_div)) or
             (X[t] == 0 and X[t - 1] == 0) or
             ((Vt1 >= (epsbeta_div)) and (Vt1 < Vt)) or (mu == 0)):
              print (color.Green + "ЦУ выполнено" + color.END)
          else:
              dz = cappa * X[t + 1]/(X[t - 1] ** 2 + X[t]** 2)
              gamma[0] = gamma[0] - dz * X[t]
              gamma[1] = gamma[1] - dz * X[t - 1]
              mu = 0
        else:
          mu = 1
          t = 0
```

Листинг 3 – часть программной реализации оценки граничного значения $\varepsilon^2\beta$.

Значение $\varepsilon^2\beta$ в программе получаем перебором. Мы ограничиваем количество игр значением border. Если за разумное количество игр робот-велосипедист так и не научился ездить, то значит, мы нашли граничное значение, то есть область после попадания, в которую он падает.

Полный листинг программной реализации оценки значения $\varepsilon^2\beta$ будет представлен в приложении к данной работе.

В следующем разделе будет графики с описанием полученных результатов.

2.4. Графики моделирования управляемой системы и оценка коэффициента $\varepsilon^2\beta$

Ниже представлены графики изменения угла наклона велосипеда от вертикали (график 1 на рисунке) и изменение угла поворота руля (график 2 на рисунке) по времени. Задаваемые разные значения параметров соответствуют различным графикам. По оси x идут значения переменной

времени t, по оси y указывается переменная χ_t, которая обозначает угол между плоскостью рамы велосипеда и вертикальной плоскостью. Так же присутствует график, изображаемый справа, где ось абсцисс та же что и на графике слева, а ось ординат это ψ, которая обозначает угол поворота руля. В конце раздела дается описание к каждому графику по полученному результату, в соответствии с заданными параметрами и номеру рисунка. Далее делается вывод по общим результатам данного моделирования.

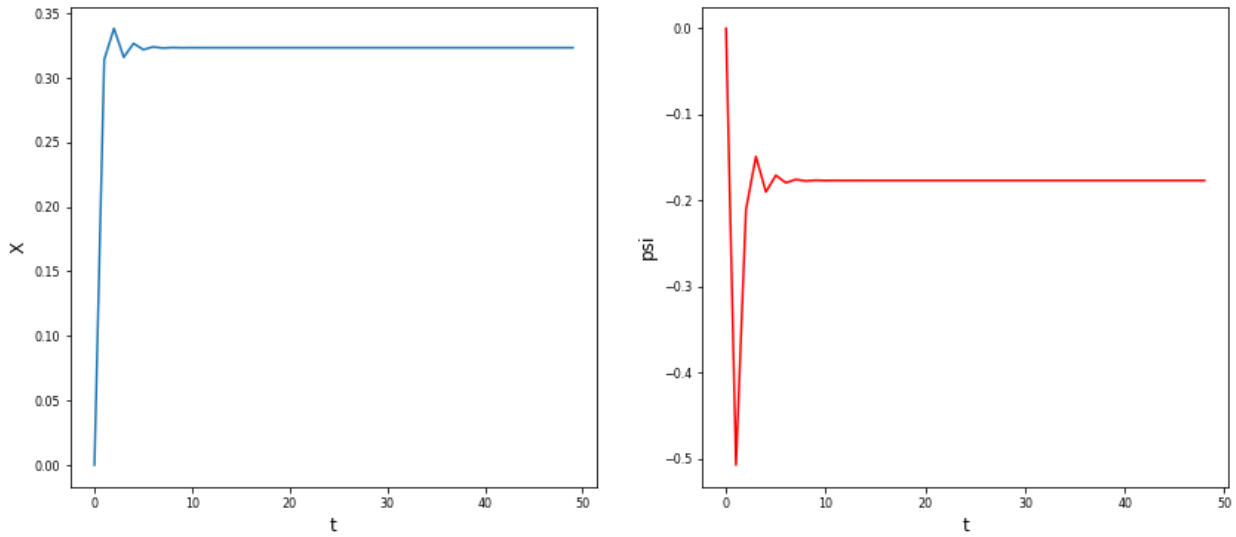

Рисунок 7 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 1).

1) При ξ = [1.1, 1.1, 1], χ_0= 0, χ_1 = π/ 10, φ = 0.5, γ = [0.9, 1], β = 0.6, ε = 1.03, $\varkappa$ = 1.4 получаем график, показанный на рисунке 7. Можно видеть, что при φ ≠ 0 угол наклона при управлении никогда не будет стремиться к 0, так как это не будет его точкой сходимости. Видно, что при небольших значениях ξ, равновесие достигается быстро. Так же для данного примера получено граничное значение $\varepsilon^2\beta$, которое равно 1.4.

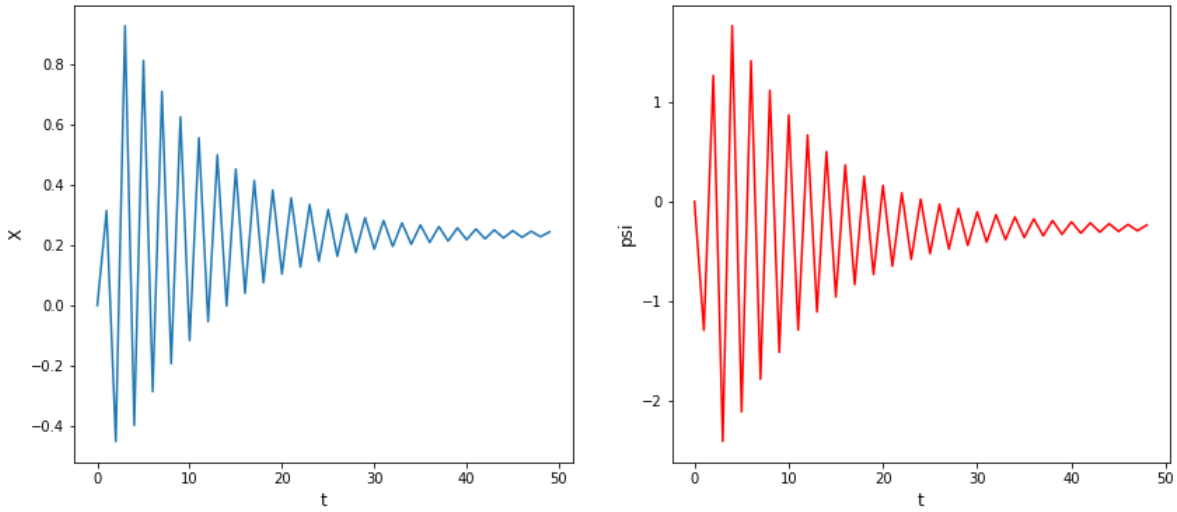

Рисунок 8 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 2).

2) При ξ = [1.1, 1.1, 1], χ_0= 0, χ_1 = π/ 10, φ = 0.5, γ = [0.9, 1], β = 0.1, ε = 1.03, $\varkappa$ = 1.4 получаем график, показанный на рисунке 8. При отдельном тестировании, где изменялось только значение β(β < 1) при значениях параметров указанных выше, было установлено, что самое лучшее управление происходит при β > 0.4(то есть надо брать либо значение ближе к середине, либо ближе к 1), при отклонении от этого значения в меньшую сторону, уравновешивание велосипеда достигается за более долгое время.

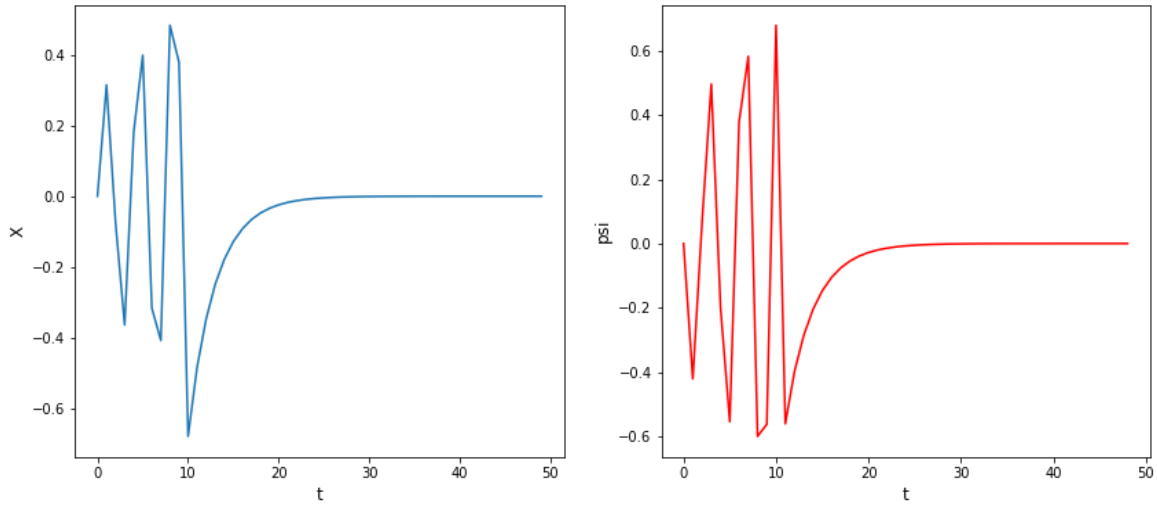

Рисунок 9 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 3).

3) При ξ = [1.1, 1.1, 1], χ_0= 0, χ_1 = π/ 10, φ = 0, γ = [1, 1], β = 0.5, ε = 1, $\varkappa$ = 1.4 получаем график, показанный на рисунке 9. При изменении φ = 0,

видим стремление функции к 0. Изменение γ в дальнейшем не будем рассматривать, так как оно влияет только вначале на управление, а в основном его начальное значение нам не интересно. Так же для данного примера получено граничное значение $\varepsilon^2\beta$, которое равно 1.3.

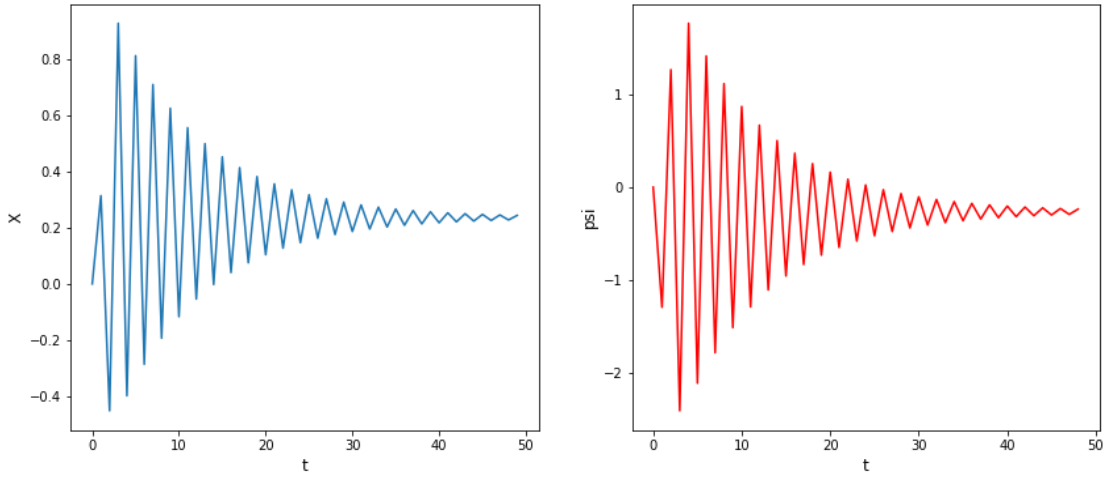

Рисунок 10 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 4).

4) При ξ = [1.1, 1.1, 1], χ_0= 0, χ_1 = π/ 10, φ = 0.5, γ = [1, 1], β = 0.6, ε = 0.1, $\varkappa$ = 1.4 получаем график, показанный на рисунке 10. Видим, что при уменьшении ε результат становится хуже, по сравнению с рисунком 7, у которого все значения параметров те же, кроме ε. Такие результаты происходят и в ряде тестирований, где все значения, кроме ε, остаются не изменяемыми.

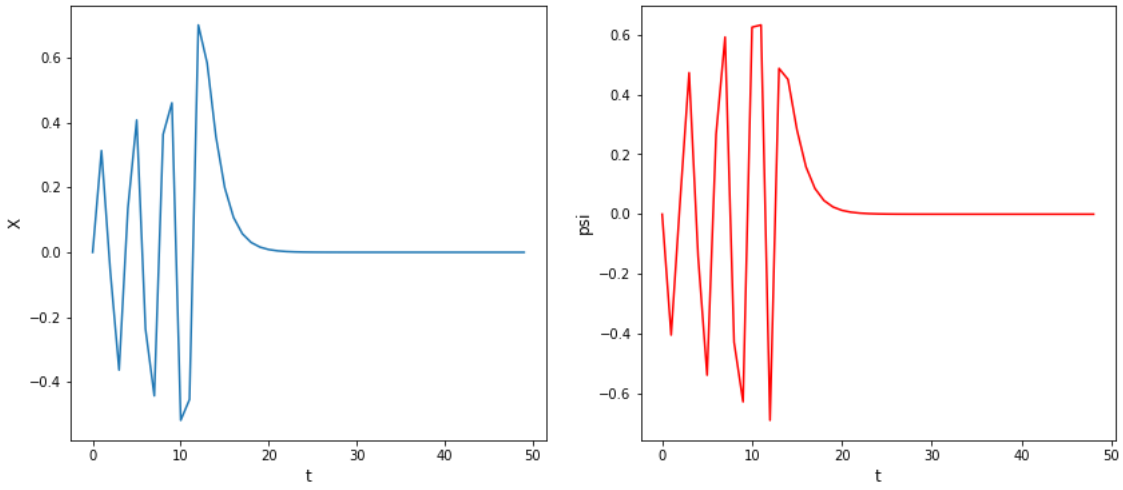

Рисунок 11 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 5).

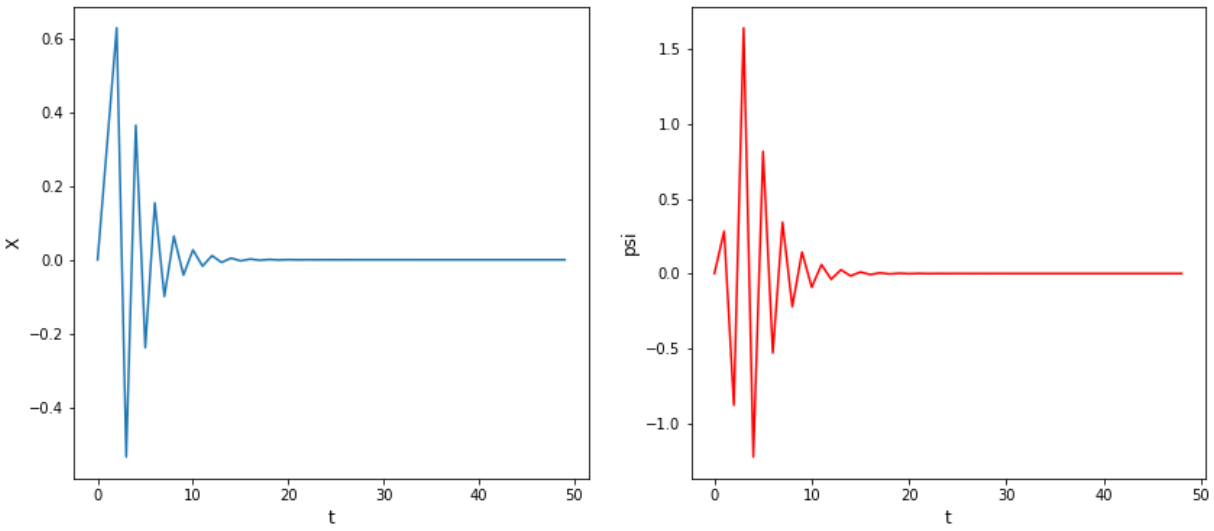

Рисунок 12 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 5).

5) При ξ = [1.1, 1.1, 1], χ_0= 0, χ_1 = π/ 10, φ = 0, γ = [1, 1], β = 0.6, ϰ = 1.4, ε = 1.0 (рис.11), ε = 0.1 (рис.12) получаем графики, показанный на рисунке 11, 12. Видим, что при уменьшении ε результат становится лучше, по сравнению с рисунком 11, у которого все значения параметров те же, кроме ε. Такие результаты происходят и в ряде тестирований, где все значения, кроме ε, остаются не изменяемыми. Можно заметить, что при φ близком к 0, действуют выводы полученные в пункте 5), чем больше значение φ, тем ситуация меняется и становится справедлив пункт 4).

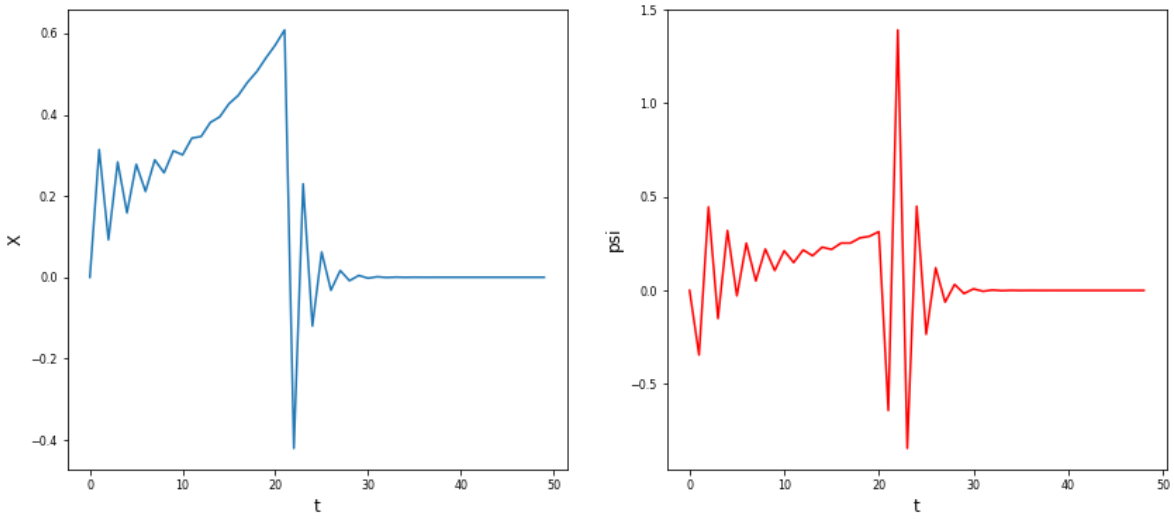

Рисунок 13 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 6).

6) При ξ = [1.5, 1.1, 1.1], χ_0= 0, χ_1 = π/ 10, φ = 0, γ = [1, 1], β = 0.5, ε = 1, $\varkappa$ = 1.5 получаем график, показанный на рисунке 11. При увеличении значения ξ_3 можно видеть, период до уравновешивания очень увеличивается. Так же для данного примера получено граничное значение $\varepsilon^2\beta$, которое равно 4.7.

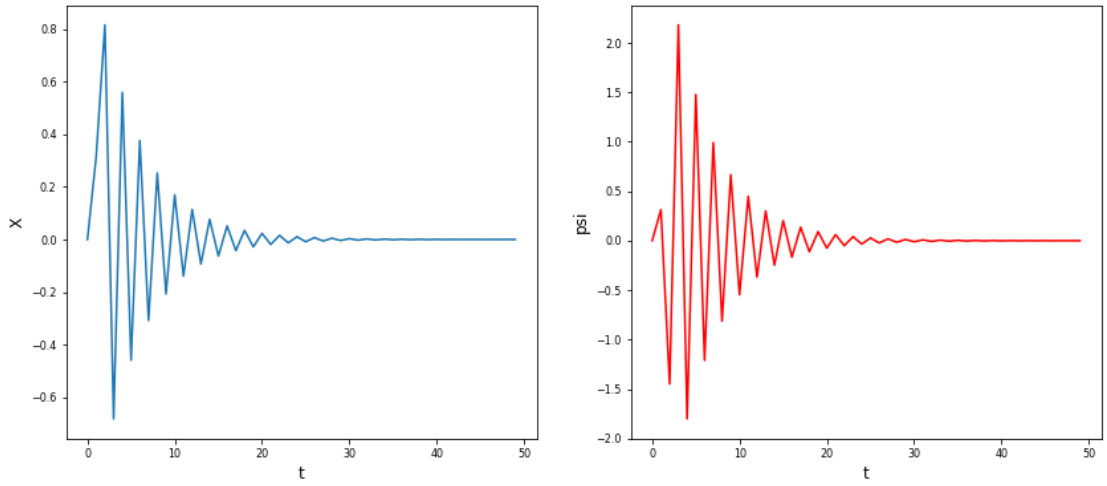

Рисунок 14 – Графики моделирования движения и управления «робота-велосипедиста» с обучением со значениями параметров 7).

7) При ξ = [1.5, 1, 1.1], χ_0= 0, χ_1 = π/ 10, φ = 0, γ = [1, 1], β = 0.5, ε = 1.1, $\varkappa$ = 1.2 получаем график, показанный на рисунке 12. Варьирование $\varkappa$ при неизменяемость других параметров дает неопределенный результат, так как управление то ухудшается, то увеличивается. Так же для данного примера получено граничное значение $\varepsilon^2\beta$, которое равно 3.5.

По графикам представленных выше видно, что алгоритм работает при соблюдении условий, которые присутствуют в статье. Следовательно, единицы измерения, начальные условия, известные и не известные компоненты, пространства параметров и функций определены верно. Так же был сделан вывод, какие лучше брать параметры ε и β. Они являлись одной из основных проблем в 1.2. ε стоит брать в зависимости от φ. β должно быть ближе к 1, чем к 0. Что касается ξ, по ним вывод был сделан в 2.2. Все

графики после регулирования уравновешиваются и стремятся к своей точке равновесия.

Заключение

В данной работе проводилась попытка практического применения алгоритма адаптации из статьи В. А Якубовича «Адаптивные системы с многошаговыми целевыми условиями». Изучение литературы использовавшейся для разбора данной статьи и реализация прикладной части с описаниями параметров.

Целью статьи В. А Якубовича являлось обучение робота кататься на велосипеде без падений. То есть по данным о наклоне рамы относительно вертикальной плоскости, определять угол поворота руля велосипеда. Вначале мной были разобраны параметры, которые были необходимы для моделирования движения «робота-велосипедиста» без обучения. Это моделирование было важно для того, что бы понять какие параметры, на что влияют и какие ситуации стоит рассматривать, а какие нет, так как при некоторых условиях равновесие достигается и без управления, в данной работе нам такие случаи нам не интересны.

Моделирование состояло из нескольких этапов: это определение переменных, условий, задание начальных значений и удовлетворение условиям теоремы 2 статьи[3], так же написание программной реализации и отображение данных полученных в программе на графиках. Далее, после варьирования значений переменных и параметров, получилось сделать вывод о влиянии их на движение и на скорость уравновешивания. По графикам видно, что алгоритм работает верно, при соблюдении условий, которые присутствуют в статье. Следовательно, единицы измерения, начальные условия, известные и не известные компоненты, пространства параметров и функций определены верно. Все графики после регулирования уравновешиваются и стремятся к своей точке равновесия.

Достигнута следующая цель: разобран алгоритм, написана прикладная часть статьи[3] и реализована программа описывающая движение «робота-велосипедиста» с обучением (алгоритм статьи [3]).

Были выполнены следующие задачи:

1. Изучена литература по рассматриваемому вопросу и сделано заключение о необходимости развития работы на прикладной уровень, для облегчения построения алгоритма управления по параметрам объекта без глубокого анализа теоремы;
2. Реализована программа моделирования движения робота-велосипедиста без обучения.
3. Реализована программа моделирования движения робота-велосипедиста с обучением, написана программа позволяющая вычислять оценку $\varepsilon^2\beta$.
4. Сделаны выводы о сложности применения этой теории и получены замечания для специалистов в смежных областях.

Список литературы

1. Стюарт Рассел, Питер Норвиг(2017) Искусственный интеллект, Вильямс, 2

2. Машинное обучение /http://tcyber.ru/tk/research/machine_learning.html(Дата обращения: 12.02.2019)

3. Фомин В. Н, Фрадков А.Л, Якубович В.А(1981) “Адаптивное управление динамическими объектами”, Наука

4. В. А. Якубович, “Адаптивные системы с многошаговыми целевыми условиями”, Докл. АН СССР, 183:2 (1968), 303–306

5. Марк Лутц, «Программирование на Python»

6. Python 2.7.3 /https://www.python.org/ (Дата обращения: 12.03.2019)

7. NumPy 1.8.1 /http://www.scipy.org/ (Дата обращения: 15.03.2019)

8. Якубович В. А., ДАН, 166, №6 (1966)

9. Якубович В.А., “Об одной задаче самообучения целесообразному поведению”, Автомат. и телемех., 1969, № 8, 119–139

10. Якубович В. А. , “К теории адаптивных систем”, Докл. АН СССР, 182:3 (1968), 518–521

11. Лойцянский Ф. Г., Лурье Е. И., Курс теоретической механики, 3, 1934.

12. Неймарк Ю. И., Фуфаев М. А., Механика твердых тел, № 2, 12 (1967)

13. Гелиг А. Х., Матвеев А. С., Введение в математическую теорию обучаемых распознающих систем и нейронных сетей, Спб (2014)

Приложение

Листинг 4

Программная реализация моделирования движения «робота-велосипедиста» без обучения.

```
        import matplotlib.pyplot as plt
        import numpy as np
        import math
        T0 = 50
        r = math.pi / 3.
        psi = 0
        xi = [1, 1.1]
        X = [0.] * T0
        X[0] = 0
        X[1] = math.pi / 6
        phi = 0
        time = 0

        for t in range(1, T0 - 1):
        if (abs(X[t]) <= r):
               X[t + 1] = xi[0] * X[t] - xi[1] * X[t - 1] + phi
        else:
               result = - math.pi / 2 if (X[t] < 0) else math.pi / 2
               for i in range(t, T0):
               X[i] = result
               break;
        print(X)

  plt.plot(np.arange(T0), X)
```

Листинг 5

Программная реализация моделирования движения «робота-велосипедиста» с обучением.

```
     import matplotlib.pyplot as plt
     import numpy as np
     import math as m
     class color:

         Red = '\033[91m'
         Green = '\033[92m'
         Yellow = '\033[93m'
         Blue = '\033[94m'
         Magenta = '\033[95m'
         Cyan = '\033[96m'
```

```
        White = '\033[97m'
        Grey = '\033[90m'
        BOLD = '\033[1m'
        ITALIC = '\033[3m'
        UNDERLINE = '\033[4m'
        END = '\033[0m'
    T0 = 50
    r = m.pi/3
    xi = [1.5, 1.1, 1.1]
    X = [0.] * T0
    X[0] = 0
    X[1] = m.pi/10
    phi = 0
    psi = [0.] * T0
    psi[0] = 0
    psi[0] = 0
    gamma = [1, 1]
    beta = 0.5
    eps = 1
    #delta = 4
    mu = 0
    t = 0
    cappa = 1.5
    #Vt = X[t] ** 2 + beta * X[t - 1]** 2
    while (t < T0 - 2):
      t = t + 1;
      print("t=", t)
      print("X[t]=", X[t])
      if (abs(X[t]) <= r):
        psi[t] = gamma[0] * X[t] + gamma[1] * X[t - 1]
        print("psi=", psi)
        X[t + 1] = xi[0] * X[t] - xi[1] * X[t - 1] + xi[2] * psi[t] +
    phi
        print("X[t + 1]=", X[t + 1])
        Vt1 = X[t + 1] ** 2 + beta * X[t]** 2
        Vt = X[t] ** 2 + beta * X[t - 1]** 2
        print (color.BOLD + "Условие" + color.END, (Vt1 < (eps ** 2 *
    beta),
                                                    (X[t] == 0 and
    X[t - 1] == 0),
                                                    ((Vt1 >= (eps **
    2 * beta)) and Vt1 < Vt)))
        if ((Vt1 < (eps ** 2 * beta)) or
           (X[t] == 0 and X[t - 1] == 0) or
           ((Vt1 >= (eps ** 2 * beta)) and (Vt1 < Vt)) or (mu == 0)):
            print (color.Green + "ЦУ выполнено" + color.END)
        else:
            print (color.Yellow + "Изменяем каким-то образом gamma" +
    color.END)
            dz = cappa * X[t + 1]/(X[t - 1] ** 2 + X[t]** 2)
            gamma[0] = gamma[0] - dz * X[t]
```

```
            gamma[1] = gamma[1] - dz * X[t - 1]
            mu = 0
            print(color.Blue + "gamma = " + color.END, gamma)
      else:
        print (color.Red + "Упал" + color.END)
        mu = 1
        t = 0
    print(X)
    print (gamma[0], gamma[1])
    fig, (ax1, ax2) = plt.subplots(
        nrows=1, ncols=2,
        figsize=(14, 6)
    )
    ax1.plot(np.arange(T0), X)
    ax1.set_xlabel("t")
    ax1.set_ylabel("X")
    plt.rc('axes', labelsize=12)
    ax2.plot(np.arange(T0 - 1), psi[0:T0 - 1], 'r')
    ax2.set_xlabel("t")
    ax2.set_ylabel("psi")
```

Листинг 6

Программная реализация оценки граничного значения $\varepsilon^2\beta$.

```
    import matplotlib.pyplot as plt
    import numpy as np
    import math as m
    T0 = 50
    r = m.pi/3.
    border = 200
    start_bord = 0
    for epsbeta in range(0, 50, 1):
        if (start_bord == border) and (epsbeta != 0):
              print("Граничное значение " + str((epsbeta - 1)/10) +"
    "+ str(beta) +" "+ str(eps))
              break
        start_bord = 0
        xi = [1.5, 1, 1.1]
        X = [0.] * T0
        X[0] = 0
        X[1] = m.pi/10
        phi = 0
        psi = [0.] * T0
        psi[0] = 0
        psi[0] = 0
        gamma = [1, 1]
        mu = 0
        t = 0
        cappa = 1.2
        epsbeta_div = epsbeta / 10
        beta = epsbeta_div / 1.03
```

```
          eps = m.sqrt(1.03)

          while (t < T0 - 2):
            if (start_bord == border):
                for i in range(t + 1, T0):
                  X[i] = m.pi/3.
                break
            start_bord = start_bord + 1
            t = t + 1;
            if (abs(X[t]) <= r):
              psi[t] = gamma[0] * X[t] + gamma[1] * X[t - 1]
              X[t + 1] = xi[0] * X[t] - xi[1] * X[t - 1] + xi[2] *
      psi[t] + phi
              Vt1 = X[t + 1] ** 2 + beta * X[t]** 2
              Vt = X[t] ** 2 + beta * X[t - 1]** 2
              if ((Vt1 < (epsbeta_div)) or
                 (X[t] == 0 and X[t - 1] == 0) or
                 ((Vt1 >= (epsbeta_div)) and (Vt1 < Vt)) or (mu == 0)):
                  gamma[0] = gamma[0]
              else:
                  dz = cappa * X[t + 1]/(X[t - 1] ** 2 + X[t]** 2)
                  gamma[0] = gamma[0] - dz * X[t]
                  gamma[1] = gamma[1] - dz * X[t - 1]
                  mu = 0
            else:
              mu = 1
              t = 0
```