МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. И.А. БУНИНА»

О. Н. Масина, А. А. Петров, О. В. Дружинина

ИСПОЛЬЗОВАНИЕ ВЫСОКОУРОВНЕВЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ МОДЕЛИРОВАНИЯ

Учебное пособие

Елец – 2023

УДК 51
ББК 32.97
М 31

Печатается по решению редакционно-издательского совета
Елецкого государственного университета им. И.А. Бунина
от 22.02.2023, протокол № 1

Рецензенты:
Т.Ф. Климова, кандидат технических наук,
доцент кафедры «Высшая математика и естественные науки»
(Российский университет транспорта);
В.Е. Щербатых, кандидат физико-математических наук,
доцент кафедры математики и методики ее преподавания
(Елецкий государственный университет им. И.А. Бунина)

О.Н. Масина, А.А. Петров, О.В. Дружинина

М 31 Использование высокоуровневых языков программирования для решения задач моделирования: учебное пособие. – Елец: Елецкий государственный университет им. И.А. Бунина, 2023. – 98 с.
ISBN 978-5-00151-382-7

В учебном пособии рассмотрены аспекты использования высокоуровневых языков программирования для научных вычислений. Охарактеризованы программно-аппаратные средства для математического моделирования динамических систем. Рассмотрены некоторые задачи, связанные с разработкой программного обеспечения для научных вычислений, проведен сравнительный анализ соответствующих инструментальных средств, перечислены и охарактеризованы библиотеки для языков высокого уровня. Приведена общая характеристика языков Python, Julia и Octave. Подробно рассмотрены конкретные примеры построения и компьютерного исследования математических моделей динамических систем. Приведены задачи для самостоятельного решения.

Пособие предназначено для обучающихся в высших учебных заведениях студентов физико-математических и технических направлений подготовки. Пособие может быть использовано аспирантами соответствующих направлений обучения.

УДК 51
ББК 32.97

ISBN 978-5-00151-382-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящее пособие посвящено вопросам научных вычислений, связанных с математическим моделированием сложных управляемых систем. Существенная часть данного пособия посвящена аспектам построения компьютерных динамических моделей, которые задаются дифференциальными уравнениями. Рассмотрены конкретные примеры научных вычислений на трех языках программирования: Python, Julia и Octave. Отметим, что для освоения представленного материала требуются базовые знания в области математического анализа и теории дифференциальных уравнений, а также необходимы базовые навыки алгоритмизации и программирования. Данное пособие не следует рассматривать как руководство по программированию, и в то же время материал ориентирован на то, чтобы продемонстрировать ключевые различия в системе типов языков высокого уровня (ЯВУ), а также охарактеризовать возможные подходы к компьютерному моделированию. Назначением настоящего учебного пособия является помощь студентам и аспирантам в выборе инструментальных средств для проведения исследований в области математического моделирования и в получении навыков научных вычислений с применением различных математических библиотек. Приводится перечень программного обеспечения, функционал которого будет полезен не только специалистам в области математического моделирования, но и более широкому кругу исследователей. Для изучения программирования на языках Python, Julia и Octave, а также для облегчения понимания приведенных примеров рекомендуется ознакомиться с [3, 30, 47, 55, 61–63, 69, 79–81, 83, 84, 90, 91]. Ряд фундаментальных результатов по теории управления динамическими системами содержится в [1, 2, 5, 9–12, 15, 22, 40, 43, 48, 71].

Пособие состоит из 7 параграфов.

В § 1 рассмотрены некоторые задачи, связанные с разработкой программного обеспечения для научных вычислений, проведен сравнительный анализ соответствующих инструментальных средств, перечислены и охарактеризованы библиотеки для языков высокого уровня.

В § 2 приведена общая характеристика языка Python, рассмотрена система типов указанного языка и охарактеризованы некоторые аспекты применения объектно-ориентированного программирования при решении научных задач. Приведена математическая модель движения частицы в двумерном пространстве и предложена соответствующая компьютерная реализация на языке Python.

В § 3 рассмотрена научная задача по исследованию моделей поэтапного усвоения знаний, приведены результаты моделирования траекторной динамики, рассмотрены аспекты компьютерной реализации указанной модели.

В § 4 приведена общая характеристика языка Julia, рассмотрена система типов указанного языка и охарактеризованы некоторые аспекты применения подхода с множественной диспетчеризацией при решении научных задач. На основе модели движения частицы разработана компьютерная программа на языке Julia, приведены результаты вычислительного эксперимента.

В § 5 рассмотрена научно-исследовательская проблема по оптимальному нейросетевому управлению моделью ленточного конвейера с динамическим углом подъема ленты. Обсуждаются результаты вычислительных экспериментов. Приведены особенности компьютерной реализации модели на языке Julia.

В § 6 дана общая характеристика языка Octave, рассмотрена система типов, предложено решение научно-исследовательской проблемы по

анализу модели технической системы с переключениями. Рассмотрены особенности компьютерной реализации указанной модели на языке Octave.

В § 7 приведены задачи для самостоятельного решения.

Учебное пособие предназначено для обучающихся в высших учебных заведениях студентов физико-математических и технических направлений подготовки, а также для самостоятельной работы студентов-заочников различных специальностей. Пособие может быть использовано аспирантами таких направлений обучения, которые связаны с математическим и компьютерным моделированием, численными методами, информатикой и вычислительной техникой, с информационными технологиями.

§ 1. Особенности применения высокоуровневых языков в научных вычислениях

1.1. Некоторые задачи, связанные с разработкой программного обеспечения

В настоящем разделе рассматриваются задачи, которые связаны с разработкой программного обеспечения на языках высокого уровня.

Прежде всего, понятие научного программирования неразрывно связано с задачами, решение которых предполагает использование методов прикладной математики. Прикладная математика включает в себя множество разделов, каждый из которых направлен на решение определенного типа задач. Среди указанных разделов можно отметить следующие.

Линейная алгебра – раздел прикладной математики, который занимается изучением линейных уравнений и матриц. Линейная алгебра используется как математический аппарат для решения различных задач в физике, информатике, технических науках.

Численные методы дифференциального исчисления – это раздел прикладной математики, который посвящен численному решению уравнений, содержащих производные. Указанный тип уравнений широко используется для описания динамических систем. В частности, динамическими являются большинство физических процессов, таких как движение тел, жидкостей и газов, перенос тепла, распространение звуковых и электромагнитных волн.

Теория вероятностей – это раздел прикладной математики, изучающий вероятность и статистику. В рамках данного раздела изучаются случайные явления и процессы, что находит широкое применение в имитационном моделировании.

Методы оптимизации – это раздел прикладной математики, посвященный поиску оптимальных решений для широкого перечня задач, которые имеют количественные критерии оптимальности. Указанный тип задач повсеместно встречается в физике, информатике, экономике, экологии и других отраслях научного знания.

Машинное обучение и анализ данных – это раздел прикладной математики, посвященный, как правило, решению задач классификации, кластеризации или регрессии с применением интеллектуальных моделей, построенных на основе существующих данных. Следует отметить, что данных раздел прикладной математики является одним из наиболее динамично развивающихся.

Компьютерная графика – это раздел прикладной математики, который посвящен применению математических методов для создания изображений, анимации и трехмерных моделей.

Теория игр – раздел прикладной математики, изучающий стратегии в играх с несколькими участниками.

Несмотря на то, что многие численные методы, применяемые в прикладной математике, известны достаточно давно, на практике широкое применение указанных методов было ограничено высокой трудоемкостью вычислений.

Рассмотрим, например, численное решение задачи Коши для обыкновенных дифференциальных уравнений первого порядка методом Эйлера. Подобные уравнения часто встречаются при построении математических моделей динамических систем. Пусть дано уравнение

$$\frac{dx}{dt} = f\left(x\right), \qquad (1.1.1)$$

где x – фазовая переменная. Для простоты будем предполагать, что x – скаляр. Суть метода Эйлера можно понять из определения производной, т. е.

$$\frac{dx}{dt} = \lim \frac{x(t) - x(t_0)}{t - t_0} = \lim \frac{\Delta x}{\Delta t}, \qquad (1.1.2)$$

где Δt и Δx – приращения аргумента и приращения функции соответственно.

Нетрудно заменить, что при достаточно малых значениях приращения аргумента как правило справедливо соотношение

$$\frac{dx}{dt} \approx \frac{\Delta x}{\Delta t}.$$

Таким образом, разбив фазовое пространство на сетку с постоянным шагом приращения аргумента h можно записать уравнение (1) в разностной форме

$$\frac{x_i - x_{i-1}}{h} = f\left(x_{i-1}\right),$$

откуда следует

$$x_i = hf\left(x_{i-1}\right) + x_{i-1}. \qquad (1.1.3)$$

Указанное соотношение (3) является простейшей разностной схемой, которая позволяет итеративно получить траекторию уравнения (1), зная начальные условия.

Несмотря на то, что метод Эйлера был предложен еще в 1788 году, а позже был обобщен в класс методов Рунге-Кутты в 1900 году, широкое применение данного метода осложнялось тем, что для получения приемлемой погрешности требовалось совершить большое количество итераций алгоритма, что для практически значимых дифференциальных моделей является крайне трудоемкой задачей.

Широкое применение численные методы нашли с изобретением вычислительной техники, которая позволила полностью автоматизировать

процесс решения указанной задачи. На сегодняшний день инструментарий современных языков высокого уровня для научного программирования покрывает практически все потребности прикладной математики и вычислительных методов, а также позволяет решать часть задач в декларативном стиле.

1.2. Сравнительный анализ инструментальных средств для научного программирования

Изначально вычислительная техника первого поколения была ориентирована преимущественно на научное с использованием низкоуровневых инструментов для разработки программного обеспечения. К таким инструментам можно отнести разработку с помощью машинных кодов, а несколько позднее и использованием ассемблеров. Программы формировались в виде «заданий», т. е. данные и логика алгоритма были неотделимы друг от друга. Однако с уменьшением стоимости времени работы вычислительных машин было выявлено ряд недостатков подобного подхода, в частности:

1. Необходимость очень высокой квалификации для разработчика ПО.
2. Низкая скорость разработки.
3. Сложность поиска ошибок.
4. Отсутствие переносимости ПО.
5. Неэффективное использование времени вычислительной машины.

Очевидной стала необходимость создания инструментальных средств, нивелирующих указанные недостатки. В первую очередь ставились задачи увеличения скорости разработки и получения переносимого программного обеспечения. Единственной возможностью решить указанные задачи было создания инструментария разработки, логически

абстрагированного от аппаратного обеспечения. Это стало возможным с появлением языков программирования высокого уровня.

Характерным отличием языков высокого уровня (ЯВУ) от ассемблера является то, что любая синтаксически корректная конструкция на ЯВУ может быть преобразована в машинные команды, однако это преобразование в общем случае не является обратимым. В 50-е годы 20-го века появились несколько широко распространенных ЯВУ, ориентированных на научное программирование, одним из которых является Fortran.

Указанный язык был создан в 1957 году группой ученых из Национального управления по аэронавтике и исследованию космического пространства (NASA) в США. Они работали над созданием языка программирования для решения математических задач, связанных с космическими исследованиями.

Изначально Fortran назывался Fortran II и был разработан для работы на компьютерах IBM 704, которые использовались в NASA для обработки данных. Однако вскоре стало ясно, что Fortran II не подходит для решения ряда задач, связанных с моделированием сложных физических систем.

В 1960 году была выпущена новая версия Fortran, которая получила название Fortran III. Она была более мощной и гибкой, чем Fortran II, и позволяла программистам использовать более широкий спектр математических функций. Fortran III стал стандартом для научных вычислений в NASA и других организациях.

Язык Fortran продолжил свое развитие и стал одним из наиболее популярных языков программирования в мире. В 1977 году он был стандартизирован как Fortran IV, а в 1987 году был представлен Fortran 90, который значительно повлиял на современные языки программирования, включая C++, Java и Python.

На сегодняшний день язык Fortran обладает поддержкой ряда технологий, направленных на эффективную реализацию параллельных и гетерогенных вычислений. В частности, технология OpenMP представлена для двух языков: Fortran и C/C++. Кроме того, существует реализация технологии OpenACC для Fortran, которая позволяет воспользоваться возможностями неявного распараллеливания на графических процессорах.

Несмотря на то, что Fortran существует уже почти 70 лет, он по прежнему поддерживает ряд передовых возможностей аппаратного обеспечения и остается одним из самых популярных языков программирования для научных вычислений. Однако его использование требует сравнительно высокой меркам квалификации от разработчика. По этой причине Fortran часто используется при разработке библиотек, которые затем применяются для научного программирования на других языках. В качестве примера можно привести библиотеку OpenBLAS, используемую для реализации операций линейной алгебры во многих других библиотеках научных вычислений.

Существует ряд современных языков, порог вхождения в которые значительно ниже, чем у Fortran, но при этом они предоставляют схожие возможности для научного программирования. В рамках данного пособия рассматриваются языки Octave, Python и Julia.

Octave – это свободно распространяемый язык программирования, который используется для научных и инженерных вычислений. Проект разработки указанного языка был начат в 1988 году как часть инструментального обеспечения для моделирования химических процессов. В 1993 году вышла первая версия языка, лидером проекта выступил Джон Итан.

Язык Octave предоставляет широкий спектр функций для работы с матрицами, векторами, функциями и другими математическими объектами. Кроме того, Octave имеет интегрированную среду разработки

(IDE), которая позволяет пользователям создавать и редактировать код, а также просматривать результаты вычислений.

Одной из ключевых особенностей Octave является его совместимость с другими языками программирования. Это позволяет пользователям использовать Octave для разработки приложений на других языках программирования, таких как C, C++, Java и Python. Кроме того, существует множество библиотек, которые расширяют возможности Octave, включая библиотеки для работы с изображениями, звуком и графикой.

В целом, Octave является мощным и удобным инструментом для научных и инженерных расчетов, который может быть использован как самостоятельно, так и в сочетании с другими языками программирования и библиотеками [3].

Язык Python представляет собой высокоуровневый язык программирования общего назначения, который используется для создания различных программ, веб-приложений и скриптов. Он был создан в 1991 году Гвидо ван Россумом и с тех пор стал одним из самых популярных языков программирования.

Python имеет простой и понятный синтаксис, что делает его доступным для начинающих программистов. Он также имеет обширную стандартную библиотеку, которая содержит множество полезных функций и модулей, таких как NumPy, Pandas, Matplotlib и другие. Python также широко используется в науке о данных и машинном обучении благодаря своей гибкости и простоте использования.

В Python можно использовать различные парадигмы программирования, такие как императивное программирование, функциональное программирование и объектно-ориентированное программирование. Это позволяет создавать гибкие и масштабируемые программы и приложения.

Кроме того, Python имеет обширную экосистему инструментов и библиотек для разработки, тестирования и развертывания приложений. Это включает в себя такие инструменты, как PyCharm, Jupyter Notebook, Django и Flask, которые упрощают процесс разработки и позволяют быстро создавать веб-приложения.

Одним из главных преимуществ Python является его способность работать на различных платформах, включая Windows, macOS и Linux. Кроме того, он имеет открытый исходный код и доступен бесплатно для использования и изучения.

Таким образом, Python – это мощный и гибкий язык программирования, который подходит для широкого спектра задач и может быть использован как для создания простых скриптов, так и для разработки научного программного обеспечения [79, 84].

Язык Julia – это свободный высокоуровневый язык программирования, который был создан в 2012 году Джеффом Безансоном Аланом Эдельманом, Стефаном Карпински и Виралом Шахом [55]. Он является интерпретируемым языком программирования и имеет простой и понятный синтаксис. Одной из главных особенностей Julia является его высокая скорость работы и производительность. Он был разработан с учетом требований к научным вычислениям, поэтому он обладает высокой скоростью выполнения операций и возможностью работать с большими объемами данных.

Julia также имеет множество встроенных библиотек и инструментов, которые позволяют пользователям решать различные задачи, такие как обработка данных, научные вычисления, машинное обучение и многое другое.

Одним из основных преимуществ Julia является его открытый исходный код, что позволяет пользователям свободно использовать и изменять его для своих нужд. Кроме того, Julia имеет большое сообщество

разработчиков, которые постоянно работают над улучшением языка и созданием новых библиотек.

1.3. Библиотеки научных вычислений

Широкое распространение научного программирования создало необходимость в реализации инструментальных средств, доступных широкому кругу ученых, которые не являются специалистами в области программирования и компьютерных наук. Несмотря на то, что многие задачи научного программирования могут быть решены штатными средствами ЯВУ общего назначения, решения на основе этих средств не всегда являются оптимальными с точки зрения производительности и затрат времени на разработку. По этой причине активно развивается стороннее программное обеспечение для научных вычислений в виде специализированных библиотек, ориентированных на решение широкого спектра задач, связанных с прикладной математикой.

Следует отметить, что в рамках данного пособия рассматривается программное обеспечение с открытым исходным кодом с учетом высокой доступности для исследователей. С возможностями коммерческого программного обеспечения можно ознакомиться, например, в [36].

Ряд возможностей существующих библиотек научных вычислений, полезных при решении задач математического моделирования, приведены в таблице 1.3.1.

Таблица 1.3.1 составлена с учетом трех ЯВУ – Octave, Python и Julia. Согласно таблице 1.3.1, библиотеки научных вычислений предоставляют схожие возможности для разработки научного программного обеспечения с применением каждого из перечисленных языков программирования. Тем не менее, следует отметить некоторые различия. В частности, для Octave существует лишь один пакет (ocl) для (ocl) выполнения вычислений на

специализированных процессорах. Реализация данного типа вычислений также возможна с применением вызовов на языке C, однако это является трудоемким для неспециалистов в области программирования. Кроме того, Octave не ориентирован на применение в области машинного обучения. В частности, в таблице 1.3.1 приводится название репозитория с примерами, однако прикладные библиотеки отсутствуют.

Таблица 1.3.1 Перечень программного обеспечения

Язык / возможности	**Octave**	**Python**	**Julia**
Линейная алгебра	Встроено	NumPy/SciPy	Встроено
Численное решение ОДУ	Встроено	SciPy	DifferentialEquations.jl
Оптимизация	optim	SciPy	BlackBoxOptim.jl
Датафреймы	dataframes	pandas	Dataframes.jl
JSON	pkg-json	Встроено	JSON.jl
Символьная алгебра	Symbolic (SymPy)	SymPy	Symbolics.jl
Статистика	statistics	SciPy	StatsBase.jl
Обработка сигналов	signal	SciPy	SignalAnalysys.jl
Графы	Не известно	NetworkX	Graphs.jl
Визуализация	Встроено	matplotlib, plotly	Plots.jl
GPGPU	ocl	PyOpenCL, PyCUDA, arrayfire и др.	JuliaGPU
Нечеткая логика	fuzzy-logic-toolkit	scikit-fuzzy	FuzzyLogic.jl
Машинное обучение	machine-learning-octave (репозиторий с примерами)	scikit-learn, pytorch и др.	MLJ.jl, Flux.jl
Работа с JupyterLab	Возможна	Возможна	Возможна

Язык Python представляет собой ЯВУ общего назначения, ориентированный на разработку прикладного программного обеспечения. В связи с этим в нем присутствуют лишь базовые возможности математики и работы с коллекциями. Однако возможности работы с сетью

и вводом-выводом являются достаточно развитыми. Широкий перечень инструментов научного программирования реализуют библиотеки экосистемы Numpy, среди которых можно отметить SymPy и SciPy. Согласно таблице 1.3.1, указанные библиотеки реализуют возможности по операциям линейной алгебры, решению дифференциальных уравнений, обработке сигналов, оптимизации, статистическим функциям и др.

Далее, Julia представляет собой ЯВУ общего назначения, который разработан с учетом потребностей научного программирования. Одним из наиболее важных преимуществ указанного языка является высокая производительность, которая достигается также при использовании научных библиотеках, разработанных для Julia. Кроме того, Julia имеет развитые встроенные средства по манипулированию различными типами массивов, которые обобщают понятия матриц и векторов с помощью системы абстрактных типов и матрично-векторных операций. Разработанное научное программное обеспечение на языке Julia зачастую объединено в различные инициативы по развитию наборов инструментальных средств. В частности, Symbolics.jl входит в инициативу JuliaSymbolics. Преимуществом подобной методического подхода к разработке программного обеспечения является более эффективное достижение цели разработки при сохранении гибкости и масштабируемости.

Более подробная информация по функционированию научных библиотек ЯВУ представлена в работах [79, 80, 81, 83–85, 91]. Конкретные примеры применения научных библиотек для решения научно-исследовательских задач по изучению управляемых динамических систем приведены в [16, 17, 19, 20, 38, 46, 57–59, 76–78, 86, 87].

§ 2. Общая характеристика языка Python

2.1. Система типов

Python представляет собой интерпретируемый язык со строгой динамической типизацией [62, 79, 84]. Важным аспектом реализации языка является то, что в нем используется концепция «все есть объект». Таким образом, в отличие от таких языков как C, любая переменная представляет собой объект. Однако целесообразно рассматривать несколько следующих типов данных, которые являются основными.

Число (int) – целочисленный тип данных, который может хранить целые числа, включая отрицательные.

Число с плавающей точкой (float) – тип данных, используемый для хранения чисел с плавающей запятой, таких как числа с десятичными знаками.

Строка (str) представляет собой такой тип данных для хранения текста, содержащего имена, адреса, пароли и другие строковые данные.

Булево значение (bool) – специальный тип данных, предназначенный для хранения логических значений, таких как «истина» или «ложь».

Кроме того, возможности Python предполагают наличие ряда встроенных перечисляемых типов данных. Среди них можно отметить списки (list), кортежи (tuple), словари (dict) и множества (set).

Список (list) представляет собой последовательность элементов произвольного типа. С помощью списков можно реализовывать аналоги ступенчатых массивов в других языках программирования, при этом ограничения на используемые внутри списка типы отсутствуют. Таким образом, списки являются довольно гибким инструментом создания коллекций.

Далее приведен пример работы со списком (см. листинг 2.1.1).

Листинг 2.1.1. Пример работы со списком

```
my_list = [1, 2, 3, 4, 5]
print(my_list[0]) # выводим первый элемент списка
print(len(my_list)) # выводим длину списка

# добавляем новый элемент в конец списка
my_list.append(6)
print(my_list)

# удаляем элемент по индексу 2
my_list.pop(2)
print(my_list)
```

Можно заметить, что операции для добавления и удаления элементов представляют собой методы, поскольку список является объектом.

Кортеж (tuple) также представляет собой перечисляемый тип данных, однако он не имеет возможностей по динамическому изменению содержимого.

Далее приведен пример работы с кортежем (см. листинг 2.1.2).

Листинг 2.1.2. Пример работы с кортежем

```
# Создание кортежа из двух элементов
my_tuple = (1, 2)

# Получение первого элемента кортежа
first_element = my_tuple[0]
print(first_element) # Выведет 1
```

Следует отметить, что при попытке присвоения элементу кортежа значения будет сгенерирована ошибка вида

```
my_tuple[0] = 3
```

```
----------------------------------
TypeError                                 Traceback (most
recent call last) Cell In[1], line 1 ----> 1 my_tuple[0]
= 1
   TypeError: 'tuple' object does not support item
assignment
```

Словари (dict) представляют собой перечисляемый тип данных, который хранит записи вида «ключ: значение». В отличие от списков и кортежей, словари не являются строго упорядоченными, а индексирование элементов производится по ключу.

Далее приведен пример работы с словарем (см. листинг 2.1.3).

Листинг 2.1.3. Пример работы с словарем

```
#Создаем пустой словарь
my_dict = {}

# Добавляем ключ-значение в словарь
my_dict['key'] = 'value'

# Выводим все ключи и значения из словаря
for key in my_dict:
    print(key, my_dict[key])

# Удаляем ключ из словаря
del my_dict['key']

# Проверяем, что ключ удален из словаря
print(my_dict)
```

Следует отметить, что характерным свойством словаря является то, что ключи в нем являются уникальными.

Множество (set) представляет собой неупорядоченную совокупность элементов произвольного типа. По аналогии со словарем, элементы множества являются уникальными и не могут быть извлечены по индексу.

Далее приведен пример работы с множеством (см. листинг 2.1.4).

Листинг 2.1.4. Пример работы с множеством

```
my_set = set(['apple', 'orange', 'banana'])

# Проверка, что множество содержит только уникальные элементы
print('Элементы множества:', my_set)
print('Уникальные элементы?', my_set.isdisjoint())

# Добавление нового элемента в множество
my_set.add('grape')
```

Важной концепцией языка Python является наличие изменяемых (mutable) и неизменяемых (immutable) типов данных. Указанное разделение является важным, поскольку Python не поддерживает работу с указателями.

Различие между указанными типами данных заключается в том, что при использовании операции присваивания неизменяемые типы копируются (аналогично передаче по значению в C/C++), а изменяемые типы передаются по ссылке на область памяти. Данное различие можно проиллюстрировать примерами, представленными на листингах 2.1.5 и 2.1.6. Пример работы с неизменяемым типом представлен на листинге 2.1.5.

Листинг 2.1.5. Работа с неизменяемыми типами

```
a = 1 #int — неизменяемый тип данных
b = a
b = 4
print(a,b) # выведется 1 4
```

В свою очередь, любой пользовательский класс или перечисляемый тип является изменяемым. Пример работы с изменяемым типом представлен на листинге 2.1.6.

Листинг 2.1.6. Работа с изменяемыми типами

```
a = [1,2,3] #list — изменяемый тип данных
b = a
b[0] = 5
print(a,b) # выведется [5, 2, 3] [5, 2, 3]
```

Согласно данным, представленным на листинге 2.1.6, эффект изменения списка `a` без непосредственного обращения к нему возникает по причине того, что `a` и `b` в данном случае ссылаются на одну и ту же область памяти. Для того, чтобы создать копию изменяемой переменной, необходимо явно вызвать функцию `copy()` или `deepcopy()` из штатного модуля `copy`. Кроме того, следует отметить, что вызов конструктора инициализирует переменную заново. Соответствующий пример представлен на листинге 2.1.7.

Листинг 2.1.7. Копирование и реинициализация изменяемых типов

```
a = [1,2,3] # list — изменяемый тип данных
b = a
b = [4,5,6] # переменная инициализируется заново
print(a,b) # выведется [1, 2, 3] [4, 5, 6]
```

```
from copy import copy
c = copy(a) # поверхностное копирование a
c[0] = 5
print(c,a) # выведется [5, 2, 3] [1, 2, 3]
```

Наличие изменяемых и неизменяемых типов в Python имеет принципиальное значение при решении задач моделирования систем. Это связано с тем, что часто возникает необходимость оперировать множеством копий модели, которая представлена в объектно-ориентированной форме. В данном случае любой пользовательский класс является изменяемым, и существует проблема копирования. Пример моделирования системы с множеством однотипных объектов представлен в разделе 2.2 настоящего пособия.

2.2. Подход к построению и анализу математических моделей с использованием возможностей языка Python

Рассмотрим подход к построению и анализу математических моделей на примере модели движения частицы в двумерном пространстве. Данная модель в целом аналогична задаче бросания тела под углом к горизонту. Для указанной задачи известны аналитические решения. Исходя из второго закона Ньютона, запишем уравнения

$$\begin{aligned} m\ddot{x} &= 0, \\ m\ddot{y} &= -mg, \end{aligned} \tag{2.2.1}$$

где m – масса частицы, g – величина ускорения свободного падения.

В рамках анализа указанной модели будем рассматривать численный расчет траекторий движения. Для указанного численного расчета целесообразно воспользоваться функцией solve_ivp из состава scipy.integrate. При использовании численных решателей обыкновенных

дифференциальных уравнений (ОДУ) систему (2.2.1) необходимо привести к виду ОДУ первого порядка. Сократив переменную массы и выполнив подстановку, получим уравнения вида

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= 0, \\ \dot{y}_1 &= y_2, \\ \dot{y}_2 &= -g. \end{aligned} \tag{2.2.2}$$

Следует отметить, что указанные уравнения согласуются с теоретическими положениями классической механики. В частности, уравнения (2.2.2) показывают, что величина скорости движения частицы под воздействием силы тяготения не зависит от массы.

Далее перейдем к компьютерному представлению модели. При использовании языка Python целесообразно представить указанную модель в объектно-ориентированной форме, поскольку в данном случае мы имеем дело с шаблоном повторяемого объекта. Например, в системе может присутствовать множество динамически генерируемых частиц. Стоит отметить, что коллизиями частиц модель (2.2.2) пренебрегает.

Существенными характеристиками частицы в статичном срезе состояния системы являются вектор скорости и вектор положения в пространстве, определяющие фазовое состояние частицы. Указанные характеристики частицы представим в виде атрибутов. В листинге 2.2.1 представлен программный код, реализующий класс модели.

Листинг 2.2.1. Модель частицы в двумерном пространстве

```
from scipy.integrate import solve_ivp
from matplotlib import pyplot as plt

class particle2d:
    def __init__(self, x0, y0, dx0, dy0):
```

```
        self.x = x0
        self.y = y0
        self.dx = dx0
        self.dy = dy0

    def system(self, t, y):
        g = 9.8
        return y[1], 0, y[3], -g

    def solve(self, time, maxdt = 0.01):
        y0 = [self.x, self.dx, self.y, self.dy]
        sol = solve_ivp(self.system,
                        [0, time], y0,
                        max_step = maxdt)
        self.x = sol.y[0][-1]
        self.y = sol.y[2][-1]
        self.dx = sol.y[1][-1]
        self.dy = sol.y[3][-1]
        return sol
```

Начальная скорость и начальные координаты частицы задаются в виде атрибутов объекта с помощью конструктора `__init__`. Сама система дифференциальных уравнений имеет достаточно простой вид и определяется методом `system`. Поскольку система (2.2.2) является стационарной и инвариантна по отношению ко времени, необходимость задавать начальное время в модели отсутствует (при каждом расчете автоматически задается $t = 0$).

Поскольку логика модели привязывает движение частицы к ее атрибутам, целесообразно реализовать решение уравнений как метод модели. Для этого разработана функция `solve`, назначение которой в основном заключается в построении траектории и обновлении атрибутов частицы. Указанный метод является относительно простой прослойкой для `solve_ivp` и требует в качестве обязательного аргумента только время расчета `time`. Дополнительно используется библиотека matplotlib для вывода траекторий в графическом виде.

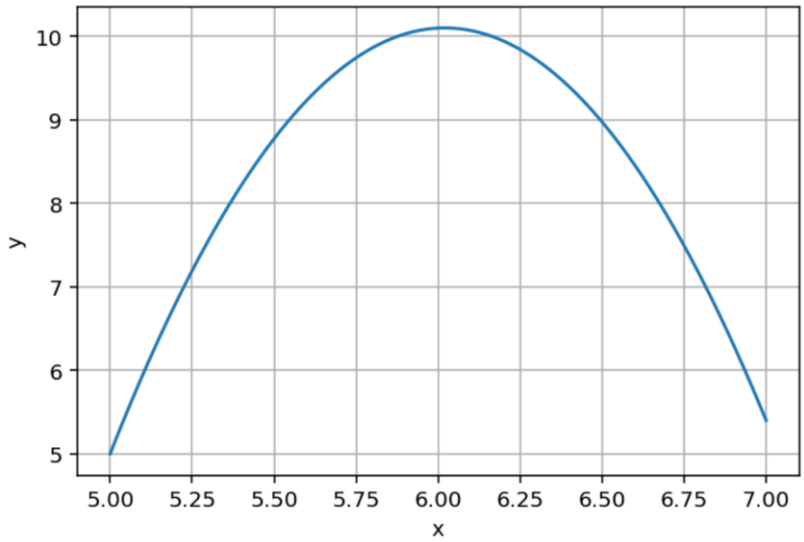
В листинге 2.2.2 приведен пример использования класса `particle2d` для построения траектории движения частицы.

Листинг 2.2.2. Построение траектории частицы

```
p1 = particle2d(5,5,1,10) #создание частицы с н.с.
sol = p1.solve(2.) # решение для t = 2
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
# вывод и отрисовка траектории в координатах x1, y1
plt.plot(sol.y[0], sol.y[2])
plt.show()

print("p1.x: {x}, \np1.y: {y}".format(x = p1.x,
                                      y = p1.y))
```

Результаты работы программного кода, приведенного в листинге 2.2.2, представлены на рис. 2.2.1.




```
p1.x: 6.999999999999956,
p1.y: 5.399999999999978
```

Рис. 2.2.1. Траектория частицы p1 и ее граничное положение

Согласно рис. 2.2.1, частица движется по параболической траектории, что согласуется с физическим смыслом. Через `p1.x` и `p1.y` обозначены координаты частицы в конечный момент расчета.

Указанный пример может быть легко адаптирован к изучению большого количества повторяемых объектов (множество частиц). Модель может быть модифицирована с учетом воздействия внешних сил, а также с учетом коллизий и взаимодействий между частицами. Кроме того, существенный интерес представляет переключаемый вариант модели (2.2.2), который может быть легко реализован через систему событий (events) функции `solve_ivp`. Указанный вариант модели может учитывать, например, отскакивание частиц от какой-либо поверхности. Более подробно переключаемые системы такого типа описаны, например, в работах [19, 20, 57–59].

§ 3. Примеры решения задач моделирования с использованием языка Python

3.1. Исследование моделей поэтапного усвоения знаний

В настоящем разделе приведен пример исследования математической модели образовательного процесса с применением языка Python3, системы JupyterLab, математических библиотек scipy, numpy, matplotlib, sklearn. Вопросы синтеза и анализа математических моделей образовательного процесса рассматривались в [24, 31–33, 41, 46] и в других работах. Сущность основных направлений кибернетической педагогики и положения теории управления образовательными системами изложены в [32, 41]. Следует отметить, что при построении и изучении моделей процесса обучения в рамках гибридной интеллектуальной обучающей среды (ГИОС) представляет интерес следующее обобщение часто используемой постановки задачи: с учетом параметров учащихся, характеристик используемых методов учителя и используемой структуры ГИОС, учебной программы, оценить уровень знаний учеников в конце обучения. Для решения задач такого типа можно использовать методы имитационного моделирования [32, 33]. Указанные методы в сочетании с методами теории оптимизации можно применить и к решению оптимизационных задач процесса обучения: найти распределение учебной информации, уровень требований учителя и оптимальную структуру соответствующего модуля ГИОС, при которых знания учеников в конце периода обучения достигнут заданного уровня с учетом того, что ограничения, накладываемые на процесс обучения, будут выполнены.

В [32, 33] предложены многокомпонентные модели обучения и рассмотрены вопросы их использования для исследования дидактических систем. Автором развивается информационно-кибернетический подход к анализу учебного процесса. Модели такого типа допускают эффективное использование методов компьютерного моделирования, позволяющих проводить серии компьютерных экспериментов с учетом различных начальных условий, параметров, внешних воздействий, а также позволяющих оценить состояние дидактической системы в конце обучения.

Далее мы рассмотрим обобщенную модель педагогического процесса, основанную на концепции «прочных» и «непрочных» знаний. Теоретический и прикладной интерес для дальнейшего обобщения и адаптации в рамках цифровизации образования представляют модели кибернетической педагогики, предложенные в [32, 33]. В частности, модель «учитель – ученик» основывается на концепции разделения знаний на категории прочности Z_1, Z_2, …, Z_n, где Z_1 – наименее прочные знания, процесс забывания которых протекает наиболее быстро, Z_n – наиболее прочные знания, сохраняющиеся продолжительное время. В двумерном случае модель задается обыкновенными дифференциальными уравнениями вида

$$\begin{aligned} \dot{Z}_1 &= k\alpha_1\left(U - Z\right) - k\alpha_2 Z_1 - \gamma_1 Z_1, \\ \dot{Z}_2 &= k\alpha_2 Z_1 - \gamma_2 Z_2, \end{aligned} \qquad (3.1.1)$$

где $Z = Z_1 + Z_2$, k – индикатор наличия процесса обучения (0 или 1), α_1, α_2 – коэффициенты скорости усвоения знаний, γ_1, γ_2 – коэффициенты забывания для Z_1 и Z_2 соответственно (определяются как $\frac{1}{\tau}$, где τ – время

снижения знаний в e раз, U – уровень требований учителя (сложность заданий).

Модель (3.1.1) является переключаемой моделью, поскольку процесс обучения подразделяется на несколько уроков с разным уровнем требований учителя $U(t)$. Следует отметить, что уровень требований может определяться разным образом, в частности, постоянные требования на протяжении временного интервала, возрастающие согласно определенному закону, либо заданные ступенчатой функций (выполнение заданий по очереди) [33].

Рассмотрим следующие условия. Пусть обучение разделено на два урока по 45 минут с переменами по 5 минут. Для иллюстрирования траекторной динамики системы (3.1.1) выбраны параметры $\alpha_1 = 0.5$, $\alpha_2 = 0.2$, $\gamma_1 = 1/3$, $\gamma_2 = 1/6$. Траекторная динамика системы (3.1.1) с учетом различных законов изменения уровня требований представлена на рис. 3.1.1, 3.1.2.

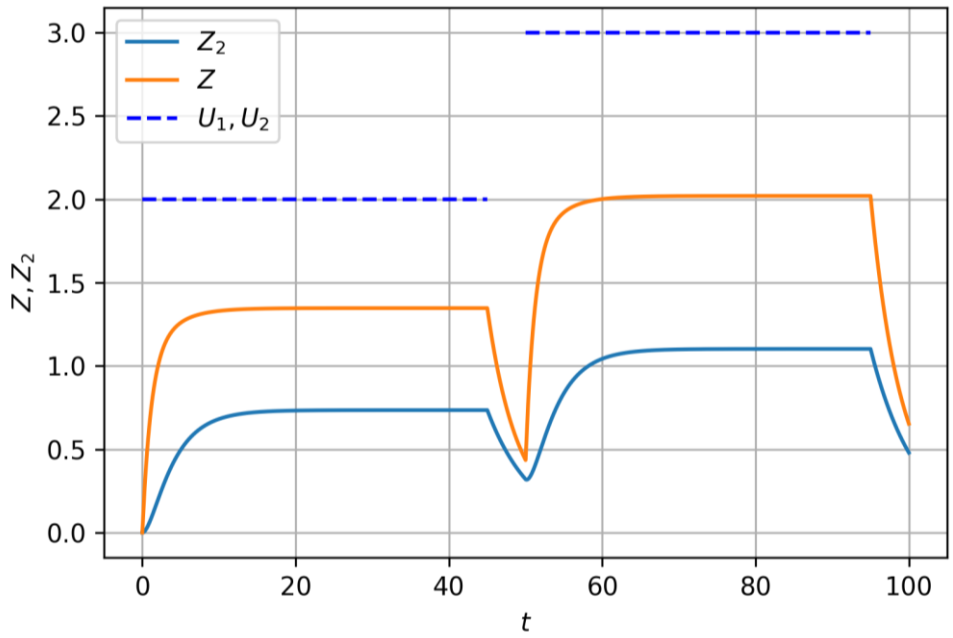


Рис. 3.1.1. Траекторная динамика системы (3.1.1) для постоянного уровня требований $U_1 = 2$, $U_2 = 3$

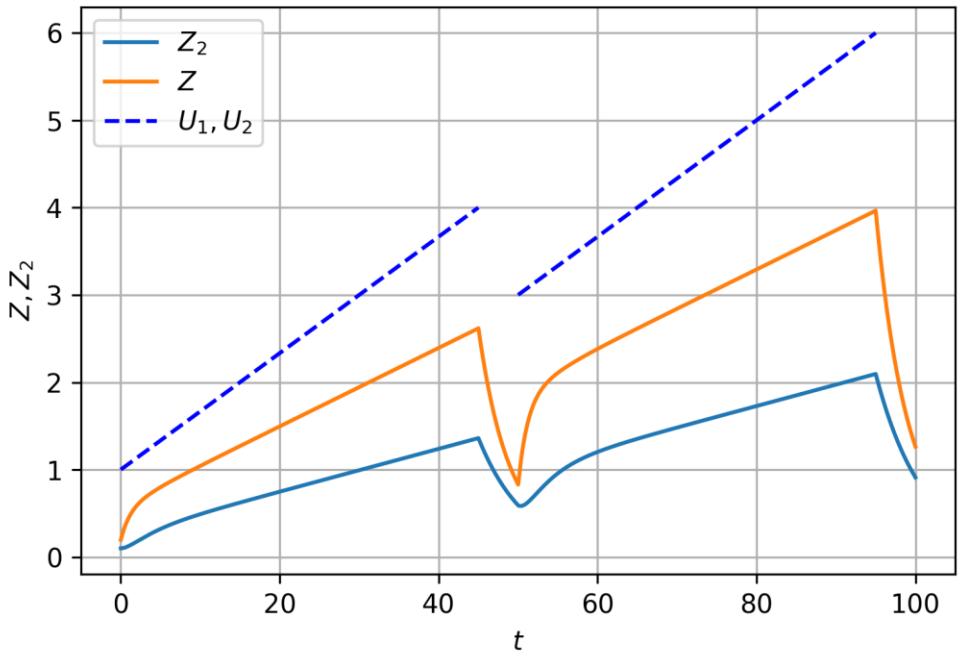


Рис. 3.1.2. Траекторная динамика системы (3.1.1) для линейно возрастающего уровня требований

Согласно представленным на рис. 3.1.1, 3.1.2 результатам следует отметить, что общий уровень знаний стремится к уровню требований U и снижается в момент наступления перемены, в основном за счет уменьшения наименее прочных знаний.

Мы предлагаем модификацию модели (3.1.1) с учетом того, что менее прочные знания преобразуются в более прочные не мгновенно, а через некоторое время после осмысливания учеником. В таком случае модель (3.1.1) принимает вид

$$\begin{aligned} \dot{Z}_1 &= k\alpha_1\left(U - Z\right) - k\alpha_2 Z_1 - \gamma_1 Z_1, \\ \dot{Z}_2 &= k\alpha_2 Z_1\left(t - \Delta\right) - \gamma_2 Z_2, \\ Z &= Z_1 + Z_2. \end{aligned} \tag{3.1.2}$$

где приняты обозначения, аналогичные обозначениям модели (3.1.1), Δ – время осмысливания знаний учеником. Траектории системы (3.1.2) для

линейно возрастающего уровня требований и задержкой в 25 минут представлены на рис. 3.1.3.

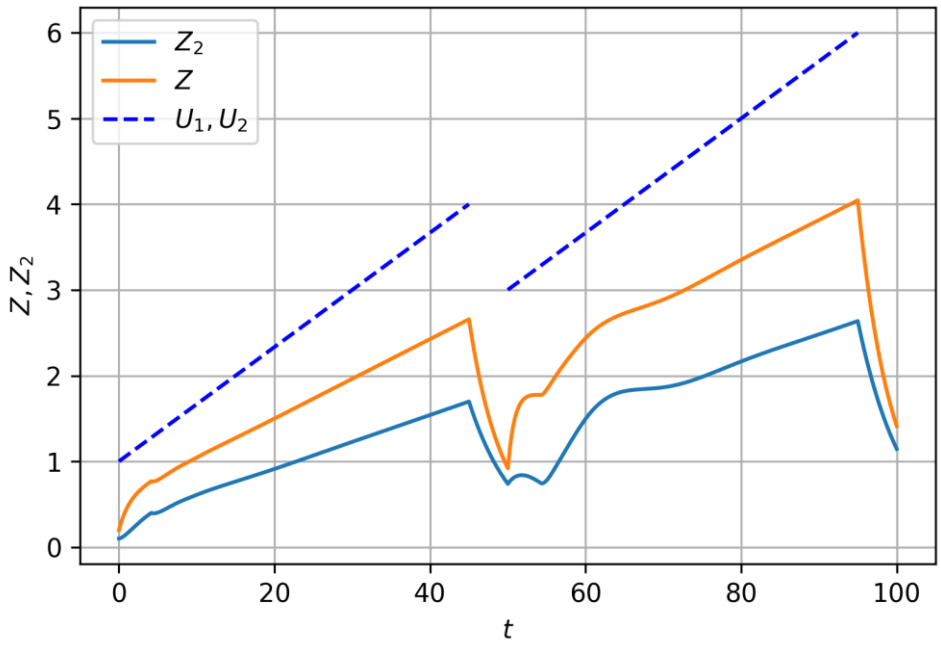


Рис. 3.1.3. Траекторная динамика системы (3.1.2) для линейно возрастающего уровня требований с учетом запаздывания

В соответствии с представленными на рис. 3.1.3 результатами можно отметить, что введение запаздывания существенно не влияет на динамику системы. Тем не менее, выявлена возможность возникновения нелинейных переходных процессов.

Использование моделей вида (3.1.2) предполагается для проведения дальнейших вычислительных экспериментов и для анализа особенностей образовательного процесса при варьировании наборов параметров и с учетом запаздывания. Выявлению новых качественных эффектов образовательного процесса, регулируемого с помощью модулей ГИОС, способствует методика моделирования с применением нейросетевого агентного подхода.

3.2. Особенности компьютерной реализации

Далее рассмотрим аспекты реализации разработанного программного обеспечения, которое было использовано для численного изучения динамики системы (3.1.2). По аналогии с разделом 2.2, используется язык Python в сочетании с объектно-ориентированным подходом к построению компьютерной модели. Класс модели представлен в листинге 3.2.1.

Листинг 3.2.1. Компьютерная реализация модели 3.1.2

```
class mayer_model_delay:
    def __init__(self, a1, a2, g1, g2, U, k = 0):
        self.k = k
        self.a1 = a1
        self.a2 = a2
        self.g1 = g1
        self.g2 = g2
        self.U = U

        self.Z_t = []

    def system(self, t, y):
        self.Z_t.append(y[0])

        Z1 = y[0]
        Z11 = Z1
        Z2 = y[1]
        Z = Z1 + Z2
```

```
        try:
            Z11 = self.Z_t[-2500]
        except:
            pass

        k = self.k
        alpha_1 = self.a1
        alpha_2 = self.a2
        gamma_1 = self.g1
        gamma_2 = self.g2
        U = self.U

        dZ1=k*alpha_1*(U - Z)-k*alpha_2*Z1-gamma_1*(Z1)
        dZ2 = k*alpha_2*Z11 - gamma_2*(Z2)

        return (dZ1, dZ2)

    def __call__(self, t, y):
        return self.system(t, y)

model1 = mayer_model_delay(.5, .35, 1/3, 1/6, 0)

solution = ode(model1, 0, [.1,.1],45+5+45+5,max_step=.01)
```

Представленная в листинге 3.1.2 программа имеет ряд отличий от программы, представленной в листинге 2.2.1. В частности, траектория системы в данном случае является атрибутом объекта и обновляется при расчете автоматически, в модели присутствует управление, причем на программном уровне реализовано запаздывание. Кроме того, важным

отличием является использование «магического» метода (magic method) `__call__`, который позволяет обращаться к объекту модели как к функции для передачи в решатель дифференциальных уравнений.

Для вывода графиков траекторий системы (3.1.2) и построения графиков управления предложен программный код, представленный в листинге 3.2.2.

Листинг 3.2.2. Программный код для вывода графиков траекторий системы (3.1.2)

```
S = []
T = []
U1 = []
U2 = []

def line(x1, y1, x2, y2):
    a = (y1 - y2)/(x1 - x2)
    b = ((x1 * y2) - (x2 * y1))/(x1 - x2)
    return a,b

for y in solution:
    T.append(solution.t)
    if 0 < solution.t < 45:
        a,b = line(0,1,45,4)
        model1.U = a*solution.t + b
        U1.append(model1.U)
        model1.k = 1
    elif 50 < solution.t < 95:
        a,b = line(50,3,95,6)
        model1.U = a*solution.t + b
```

```
            U2.append(model1.U)
            model1.k = 1
        else:
            model1.k = 0

        if 95 < solution.t:
            model1.k = 0

        S.append(y)

S = np.array(S)

plt.plot(T, S[:,1], label="$Z_2$")
plt.plot(T, S[:,0] + S[:,1], label = "$Z$")

plt.plot(np.linspace(0,45,len(U1)),
            U1, "b--", label="$U_1, U_2$")
plt.plot(np.linspace(50,95,len(U2)), U2, "b--")

plt.legend()
plt.xlabel("$t$")
plt.ylabel("$Z, Z_2$")
plt.grid()
plt.savefig("./output/mayer1_delay.png", dpi=300)
plt.show()
```

Согласно содержимому листинга 3.2.2, функции управления носят линейный характер, и используется вспомогательная функция для построения управлений. Для вывода графической информации

используется библиотека matplotlib, поддерживается экспорт в файлы формата *.png.

Таким образом, разработанная компьютерная реализация на языке Python позволяет проводить серии вычислительных экспериментов по моделированию образовательных процессов с учетом поэтапного усвоения знаний в условиях запаздывания усвоения и с учетом управляющих воздействий. Подробное описание моделей образовательных процессов и результатов компьютерных экспериментов можно найти в [46] и в других работах.

§ 4. Общая характеристика языка Julia

4.1. Система типов

Язык Julia, представленный в 2012 году, представляет собой высокопроизводительный ЯВУ со строгой динамической типизацией. Несмотря на то, что спецификации Julia разрабатывались под влиянием Python, существует ряд важных отличий. Первым отличием является использование компилятора на базе Low Level Virtual Machine (LLVM) для достижения высокой производительности программного обеспечения на Julia. Вторым отличием является то, что синтаксис языка является более строгим по отношению к типам данных и предполагает преимущественно явную типизацию. Следует отметить, что явная типизация накладывает ряд ограничений на процесс разработки. Данное решение обусловлено следующими причинами.

Первая причина заключается в том, что Julia не ориентирована на объектно-ориентированное программирование (ООП) в классическом понимании. Авторы данного языка считают ООП методологически неоправданным и придерживаются концепции множественной диспетчеризации (multiple dispatch) как основного средства для достижения высокой выразительности языка [55]. Ключевым понятием, на котором основана множественная диспетчеризация, является понятие метода.

Метод в Julia представляет собой особый вид функции, содержание которой зависит от того, какие аргументы были переданы при вызове. Таким образом, метод допускает множественную реализацию одной и той же функции, что делает интерфейс программирования достаточно универсальным. Методы с точки зрения множественной диспетчеризации существуют в различных языках программирования, таких как Common

Lisp, R, Haskell, Raku, Clojure, Nim. В Julia допускается диспетчеризация по типу данных, в результате чего явная типизация при разработке, как правило, является необходимой. Пример определения метода в Julia представлен в листинге 4.1.1.

Листинг 4.1.1. Метод в Julia

```
function norm(a::Float64)
      abs(a)
end

function norm(a::Float64, b::Float64)
       sqrt(a^2 + b^2)
end
```

Продемонстрированные в листинге две реализации метода `norm` будут вызываться в зависимости от того, сколько аргументов передано при вызове. Необходимо отметить, что указанный метод работает только с Float64, автоматического преобразования типов не предусмотрено.

Вторая причина преимущественно явной типизации в Julia заключается в том, что такой способ типизации необходим для достижения высокой производительности разработанного программного обеспечения. Поэтому в документации языка рекомендуется использовать конкретные типы даже в случае, если в этом нет прямой необходимости [55].

Следует отметить, что ограничением явной типизации является то, что она зачастую усложняет процесс разработки программного обеспечения и делает его менее гибким. Для устранения данного ограничения в Julia существует определенная форма наследования, которая заключается в существовании абстрактных типов данных.

Иерархия абстрактных типов для числовых данных представлена в рис. 4.1.1.

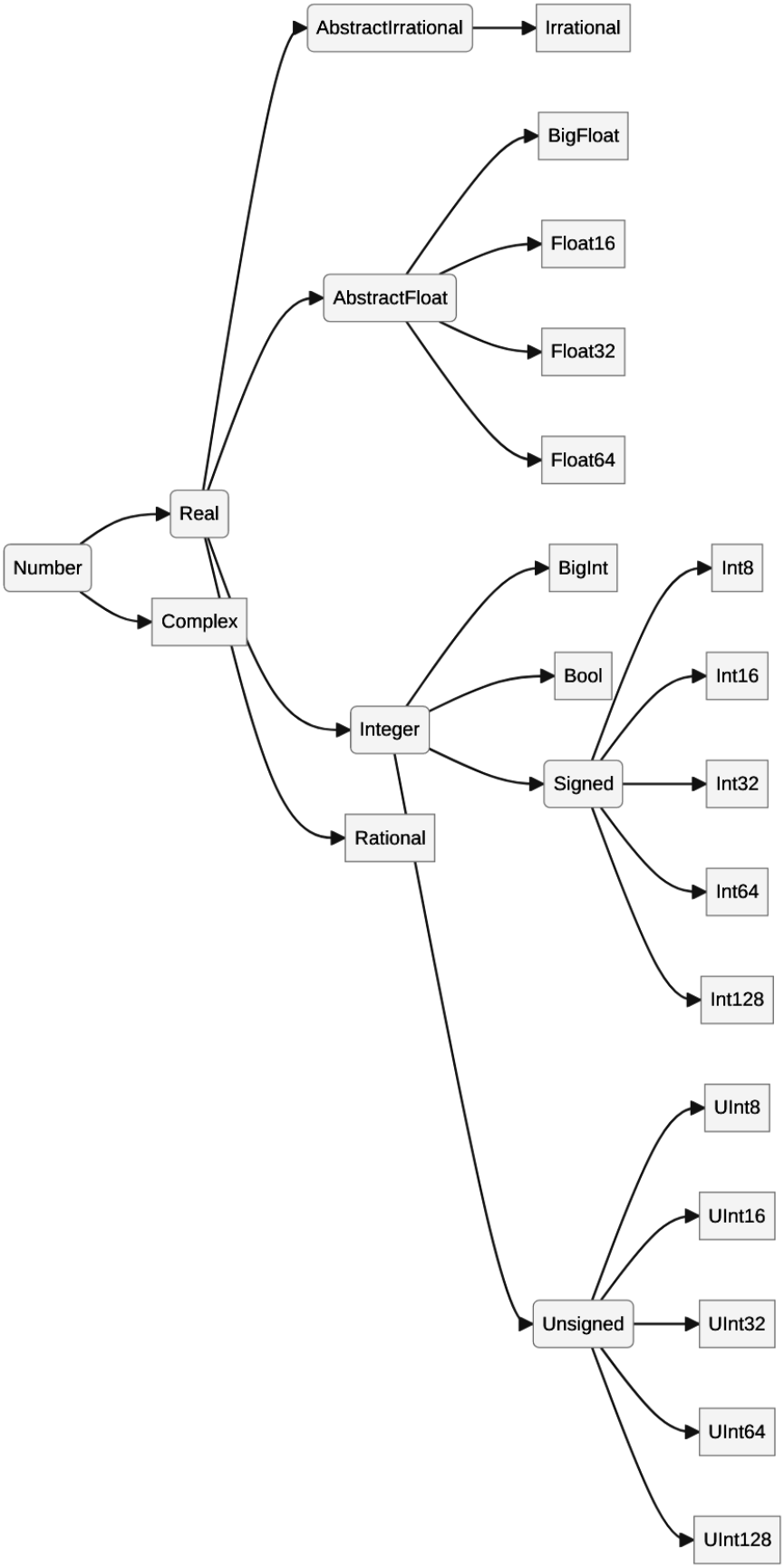


Рис. 4.1.1. Иерархия числовых типов в Julia

На рис. 4.1.1 прямоугольниками обозначены конкретные типы данных, «скругленными» прямоугольниками обозначены абстрактные типы данных. Важным отличием абстрактных типов в Julia от абстрактных классов в Python заключается в том, что абстрактные типы не могут содержать в себе реализации, однако могут быть использованы для диспетчеризации в мультиметодах.

Необходимо отметить, что диспетчеризация посредством использования абстрактных типов не рекомендуется для достижения высокой производительности (в некоторых случаях производительность может быть на уровне неявной типизации), но позволяет сохранить логическую целостность программы.

Все типы языка Julia можно разделить на примитивные (primitive types) и композитные (composite types). Примитивные типы представляют собой ссылку на участок памяти, по аналогии с типами в языке C. В отличие от большинства других ЯВУ, в Julia существует возможность определения примитивных типов пользователем с помощью директивы primitive, которая имеет следующий синтаксис:

```
primitive type «name» <: «supertype» «bits» end
```

Следует отметить, что стандартные примитивные типы языка определены аналогичным образом. Например, типы Float определяется следующим образом:

```
primitive type Float16 <: AbstractFloat 16 end
primitive type Float32 <: AbstractFloat 32 end
primitive type Float64 <: AbstractFloat 64 end
```

Пользователь может определять собственные примитивные типы данных, которые наследуются от штатных абстрактных типов. Тем не

менее, в Julia предопределены все примитивные типы, которые поддерживаются в LLVM, поэтому необходимость в дополнительных примитивных типах, как правило, отсутствует.

Композитные типы в Julia представляют собой любые отличные от примитивных типы данных. Такие типы, как правило, содержат в себе определенную структуру. Синтаксис языка позволяет определять композитные типы с помощью ключевого слова `struct`. Пример определения композитного типа представлен в листинге 4.1.2.

Листинг 4.1.2. Композитный тип, задающий координаты точки

```
struct Point
    x::Float64
    y::Float64
end
```

В отличие от Python, определяемые пользователем композитные типы не являются изменяемыми. Для определения изменяемого композитного типа используется специальное ключевое слово `mutable`. Пример изменяемого композитного типа приведен в листинге 4.1.3.

Листинг 4.1.3. Изменяемый композитный тип, задающий координаты точки

```
mutable struct mPoint
    x::Float64
    y::Float64
end
```

Также отметим, что для остальных типов концепция изменяемых и неизменяемых переменных в Julia содержательно близка к аналогичной концепции в Python. В частности, строки являются неизменяемыми объектами, а массивы являются изменяемыми объектами.

4.2. Подход к построению и анализу математических моделей с использованием возможностей языка Julia

В данном разделе рассмотрим пример, аналогичный примеру, приведенному в разделе 2.2. В качестве дифференциальной модели будем использовать уравнения (2.2.2). Отметим, что пакет DifferentialEquations.jl наряду с пакетом SciPy может применяться для решения ОДУ первого порядка в процессе изучения динамики моделей.

Компьютерная реализация модели (2.2.2) представлена в листинге 4.2.1.

Листинг 4.2.1. Реализация модели движения частицы с использованием языка Julia

```
using DifferentialEquations
using Plots

mutable struct particle
    xy::Vector{Float64}
    dxdy::Vector{Float64}
end

function solve!(p1::particle,
                t::Float64, dt_max::Float64 = 0.01)
    g = 9.8
```

```
        system!(dx,x,p,t) = begin
            dx[1] = x[2]
            dx[2] = 0
            dx[3] = x[4]
            dx[4] = -g
        end
        x₀ = [p1.xy[1] p1.dxdy[1] p1.xy[2] p1.dxdy[2]]
        t□□ₐ□ = (0.0, t)
        prob₁ = ODEProblem(system!, x₀, t□□ₐ□)
        sol = solve(prob₁, dtmax=dt_max)
        p1.xy = [sol[end][1], sol[end][3]]
        p1.dxdy = [sol[end][2], sol[end][4]]
        sol
    end
```

По сравнению с Python, компьютерная реализация модели выполнена не в объектно-ориентированном стиле. Для хранения атрибутов используется пользовательский композитный тип `particle`. Уравнения системы определяют внутри метода `solve!` с помощью функции `system!`. Как и в разделе 2.2, метод `solve!` возвращает траекторию движения частицы и меняет ее атрибуты.

Несмотря на некоторую схожесть реализаций, основная концепция приведенной на листинге 4.2.1 реализации модели (2.2.2) кардинально отличается от концепции, предложенной в разделе 4.2. В случае объектно-ориентированной реализации объект содержит свою версию метода, однако при множественной диспетчеризации методы работают с «объектами», т. е. с композитными типами. Несмотря на то, что такой подход к построению модели не позволяет использовать инкапсуляцию, его достоинством является высокая степень масштабируемости. В

частности, композитные типы, определяющие различные виды физических объектов в пространстве, можно связать абстрактным типом данных. Предложенная абстракция позволяет реализовать единый метод `solve!` для расчета траекторий каждого из видов физических объектов. Благодаря этому в полученном прикладном интерфейсе программирования легче достигнуть ортогональности по отношению к используемым типам.

Кроме того, следует отметить, что в конце названий методов используется восклицательный знак. Это согласуется с принятым соглашением о том, что изменяющие (in-place) функции обозначаются таким образом. Также обычно считается, что изменяющая функция должна изменять первый аргумент, но не возвращать результат. Несмотря на это, в данном случае `solve!` не является строго изменяющей функцией, но все же уместно помечать ее как изменяющую, поскольку происходит изменение первого аргумента.

В листинге 4.2.2 приведен пример использования типа `particle` для построения траектории движения частицы.

Листинг 4.2.2. Построение траектории частицы

```
p1 = particle([5., 5.],
              [1., 10.])
sol = solve!(p1, 2.)

println(p1.xy)
plt = plot()
xlabel!(plt, "x")
ylabel!(plt, "y")
plot!(plt, sol[1,:], sol[3,:], label = "y")
```

В данном листинге также можно отметить, что используются изменяющие функции `xlabel!, ylabel!, plot!,` которые работают с композитным типом, возвращаемым неизменяющей функцией `plot`. Можно отметить, что интерфейс `Plots.jl` напоминает `matplotlib`, однако построен с применением множественной диспетчеризации.

Результаты работы программного кода, приведенного в листинге 4.2.2, представлены на рис. 4.2.1.

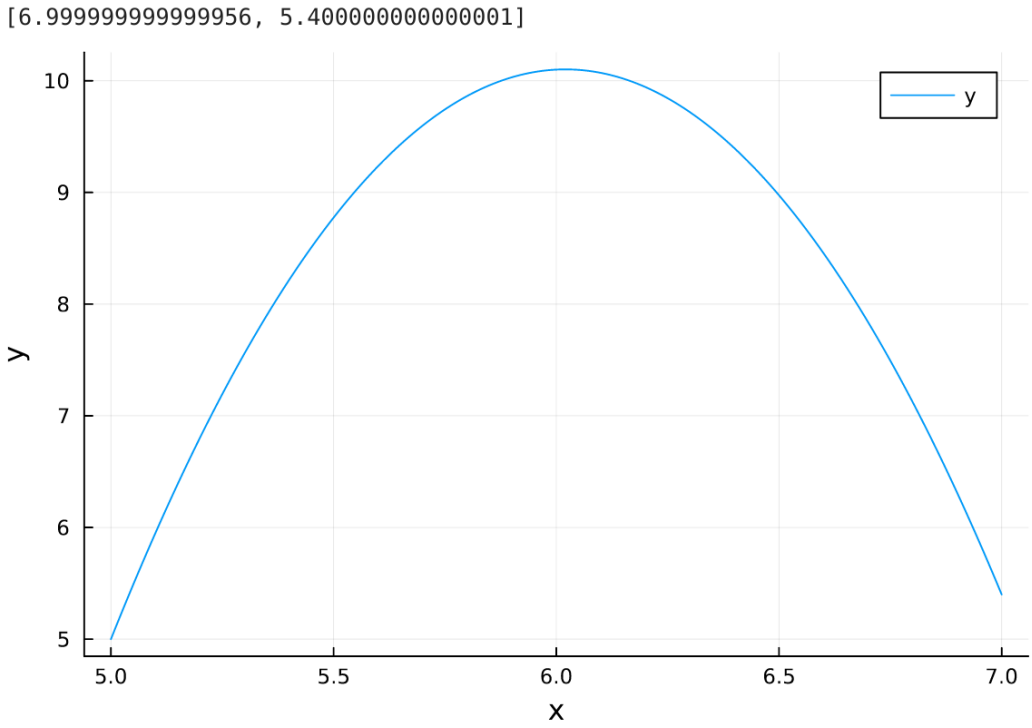


Рис. 4.2.1. Траектория движения частицы

Согласно рис. 4.2.1, параболическая траектория движения частицы совпадает с траекторией, полученной на рис. 2.2.1 при аналогичных условиях, граничное положение частицы соответствует положению, показанному на рис. 2.2.1.

Как и в случае с реализацией на языке Python, на Julia возможна реализация с учетом повторяемых объектов (случай множества частиц).

Кроме того, пакет DifferentialEquations.jl предоставляет широкие возможности по изучению различных типов динамических систем, которые описываются дифференциальными уравнениями в частных производных, дифференциальными уравнениями с запаздыванием, стохастическими дифференциальными уравнениями и другими видами дифференциальных уравнений. Также важное значение имеет тот факт, что численный расчет дифференциальных уравнений на языке Julia производится существенно быстрее, чем аналогичный расчет на языке Python [64].

§ 5. Примеры решения задач моделирования с использованием языка Julia

5.1. Нейросетевое управление моделью ленточного конвейера с динамическим углом подъема

Далее рассматривается переключаемая модель ленточного конвейера с динамическим изменением угла между горизонтальной плоскостью и плоскостью ленты конвейера. Модель задается четырехмерным нелинейным дифференциальным уравнением с управляющими функциями. Поиск управляющих функций согласно заданному критерию качества осуществляется с помощью нейросетевого регулятора. Переключения в предложенной модели учитываются в виде факторов плавной погрузки и мгновенной разгрузки грузов. Для исследования построенной модели использованы методы численного решения обыкновенных дифференциальных уравнений, методы численной оптимизации и методы интеллектуального анализа. Разработано программное обеспечение на языке Julia с привлечением библиотек DifferentialEquations, Plots, BlackBoxOptim, а также оригинальной библиотеки нейросетевых вычислений.

Проектирование, автоматизация и мониторинг систем конвейерного транспорта являются актуальными направлениями исследований [53, 54, 72, 94, 98]. В круг важных задач входят, например, задачи стабилизации тягового фактора конвейера, мониторинг динамической нагрузки конвейерных лент, оптимизация параметров управления конвейером, проектирование smart conveyer belt, создание многофункциональных систем непрерывного транспорта. Решение указанных задач связано с

необходимостью использования методов теории управления с элементами искусственного интеллекта [66, 67, 70, 96, 97].

При математическом моделировании систем конвейерного транспорта используются такие инструменты искусственного интеллекта, как нечеткое управление, искусственные нейронные сети и машинное обучение [29, 52, 60, 65, 73 – 75, 99]. В работе [52] рассматриваются вопросы построения нечеткой трекинговой системы для отслеживания состояния конвейера. В [65] изучены аспекты оптимизации управления конвейера с применением искусственных нейронных сетей. В [99] рассмотрены вопросы применения моделей машинного обучения для высокоточной классификации типов нагрузок на резиновую конвейерную ленту.

Различные аспекты моделирования динамики ленточного конвейера с переменным углом подъема рассмотрены в [57, 76, 78, 86]. В [76] представлена модель ленточного конвейера с динамическим изменением угла между горизонтальной плоскостью и плоскостью ленты конвейера. Изучены вопросы оптимального управления указанной моделью с использованием комбинированного регулятора на основе скользящего режима и системы нечеткого вывода в форме Мамдани. В [57] предложены методы исследования модели ленточного конвейера на основе построения ПИД-регуляторов и нейросетевых регуляторов. В [78] разработана такая модифицированная математическая модель ленточного конвейера, в которой учитываются дополнительные диссипативные эффекты. Кроме того, в указанной статье проведен сравнительный анализ различных типов интеллектуального управления ленточным конвейером.

Уточнение моделей, рассмотренных в [76, 57, 78] может учитывать различные факторы, среди которых следует отметить воздействие

переходных процессов на динамику системы при изменении режимов нагрузки конвейера. В настоящей статье мы решаем задачу синтеза нейросетевого регулятора для новой модели ленточного конвейера с динамическим изменением угла между горизонтальной плоскостью и плоскостью ленты конвейера. Предложенная модель учитывает факторы плавной погрузки и мгновенной разгрузки грузов. Мы проводим сравнительный анализ результатов компьютерного исследования этой модели с результатами для модели, рассмотренной в [78].

Рассмотрим базовую модель одноприводного ленточного конвейера с динамически изменяемым углом подъема [76]. Мы предполагаем провести исследование модели при следующих условиях: лента конвейера является нерастяжимой, грузы на ленте оказывают существенное влияние на импульс и момент импульса конвейера, сила трения грузов о конвейерную ленту не учитывается. С учетом этих условий дифференциальные уравнения модели можно записать в виде

$$\begin{gathered}
\dot{x}_0 = x_1, \\
\dot{x}_1 = \frac{u_1(t) - kx_1 - m_1 g \sin(\alpha_0)}{m_1 + m_0}, \\
\dot{\alpha}_0 = \alpha_1, \\
\dot{\alpha}_1 = \frac{u_2(t)}{s\varepsilon^2 (m_0 + m_1)} - \frac{g \cos(\alpha_0)}{\varepsilon}, \\
m_1 \in M, \quad \varepsilon \in E, \quad u_1, u_2 \in U,
\end{gathered} \tag{5.1.1}$$

где x_0 – перемещение ленты конвейера, m_0 – масса ленты конвейера, α_0 – угол подъема конвейера, α_1 – угловая скорость подъема конвейера, m_0 – масса конвейерной ленты, m_1 – общая масса грузов на конвейере, s – коэффициент момента инерции конвейера, ε – положение центра тяжести конвейера, k – коэффициент трения качения, u_1 – величина линейной силы поступательного движения конвейера, u_2 – величина крутящего момента

для управления углом подъема конвейера, U – множество векторов управления, M – множество масс на ленте конвейера, E – множество положений центров масс конвейера.

С учетом описания модели возможны два типа действий – погрузка грузов (действие 1) и разгрузка грузов (действие 2). Следует отметить, что наличие указанных действий относит модель (5.1.1) к классу переключаемых моделей [76].

Опишем сначала особенности действия 1. Действие 1 (погрузка) заключается в следующем: пусть $m = m_0 + m_1$, $\dot{x}_1(t_{i-1})$ – скорость до момента погрузки, $\dot{x}_1(t_i)$ – скорость после погрузки. Поскольку в момент погрузки справедливо равенство $m\dot{x} = \text{const}$, получим

$$\dot{x}_1(t_{i-1})m = \dot{x}_1(t_i)(m + \Delta m),\ \dot{x}_1(t_i) = \frac{\dot{x}_1(t_{i-1})m}{m + \Delta m}\ .$$

Следует отметить, что положение центра масс также изменяется согласно равенству

$$\varepsilon = f(x_1, m + \Delta m),$$

где $f : x_1, m + \Delta M \to \varepsilon.$ Таким образом, введение функции f позволяет выполнять расчет положения центра масс. Особенности действия 2 состоят в том, что в процессе разгрузки общая масса грузов уменьшается при неизменной линейной скорости.

Далее мы рассмотрим такую модель одноприводного ленточного конвейера с динамическим углом подъема, которая описывается дифференциальными уравнениями вида

$$\dot{x} = \frac{p}{m},$$

$$\dot{p} = u_p(t) - k\frac{p}{m} - (m - m_0)g\sin(\alpha_0),$$

$$\dot{\alpha}_0 = \alpha_1, \qquad (5.1.2)$$

$$\dot{\alpha}_1 = \frac{u_\alpha(t)}{mc\varepsilon^2} - \frac{g\cos(\alpha_0)}{\varepsilon},$$

$$u_p, u_\alpha \in U, \quad \varepsilon \in E, \quad m \in M,$$

где x – линейное перемещение ленты конвейера, p – импульс системы, α_0 – это угол подъема конвейера относительно нулевого положения, α_1 – скорость углового вращения конвейера, g – ускорение свободного падения, m – общая масса системы, m_0 – общая масса грузов на конвейере, $u_p(t)$ – функция управления тягой конвейера, $u_\alpha(t)$ – функция управления углом подъема ленты, ε – положение цента масс конвейера относительно нижнего ролика, c – коэффициент, определяющий момент инерции конвейера, k – коэффициент трения качения. Множества M, E, U включают в себя все возможные значения общей массы грузов, центра масс и управлений соответственно.

Изменения режимов функционирования в модели (5.1.2) соответствует выбору m_1 и ε из множеств M, E согласно заданному закону. Система (5.1.2) относится к системам с переключениями [11, 71].

Мы рассматриваем следующую постановку задачи оптимального управления для модели (5.1.2). Необходимо с помощью управлений up, uα осуществить переходный режим и стабилизировать фазовое состояние системы (5.1.2) вблизи целевой точки $E(\dot{x}_1, \alpha_{01}, \alpha_{11})$, координаты которой соответствуют «оптимальным» значениям линейной скорости, угла подъема и угловой скорости. Важным обстоятельством является то, что при решении задачи оптимального управления следует учитывать

инвариантность по отношению к начальным условиям. Мы предлагаем следующий критерий оптимальности

$$\lim_{t_n \to \infty} \frac{1}{t_n} \int_0^{t_n} \left\| E - X(t) \right\| dt \to \min, \tag{5.1.3}$$

где $t \in (0, t_n)$, $X = (\dot{x}(t), \alpha_0(t), \alpha_1(t))$. Смысл указанного критерия заключается в осуществлении максимально быстрого переходного процесса и поддержания требуемого фазового состояния системы (5.1.2). Изучению устойчивости динамических систем посвящены, например, работы [18, 51].

Далее рассмотрим построение нейросетевого регулятора и компьютерное исследование модели. Для реализации оптимального управления в системе (5.1.2) используется комбинированный регулятор, имеющий две компоненты:

i) управление со скользящим режимом для генерации u_p;

ii) нейросетевое управление для генерации u_α.

Для модели (5.1.2), как и для модели (5.1.1), реализация скользящего режима для управления углом подъема не дает ожидаемых результатов, в связи с тем, что в моделях возникает неустойчивость. Для реализации управления углом подъема конвейера для модели (5.1.1) мы предлагаем нейросеть, топология которой включает в себя три нейрона во входном слое, восемь нейронов в скрытом слое и один нейрон в выходном слое. Нейросеть предназначена для построения управляющих воздействий для u_α. Входными данными, подаваемыми на входной слой, являются угловая скорость $\dot{\alpha}$ и ошибка $\hat{\alpha}$. В скрытом и выходном слоях используется тангенциальная функция активации. Во входном и скрытом слое используются нейроны смещения.

Результатом работы нейросети на каждом шаге является значение управляющей функции. Предложенный регулятор реализует принцип управления с обратной связью с учетом ошибки и ее производной по времени. Для подбора весовых коэффициентов мы используем обучение с подкреплением (reinforcment learning). Основной принцип алгоритма обучения заключается в следующем. Необходимо минимизировать взвешенное значение n численных оценок критерия (5.1.3). Для этого на каждом этапе алгоритма оптимизации производится n численных решений уравнений (5.1.2) с последующим расчетом значения

$$H = \mathrm{mean}(\|E - X(t)\|). \qquad (5.1.4)$$

В качестве функции потерь при обучении нейросети используется значение H. Выбор обучения с подкреплением связан со спецификой задачи построения нейросетевого регулятора для стабилизации угла подъема конвейера, моделируемого системой (5.1.2). Нейросетевой регулятор с аналогичной топологией мы использовали при построении регулятора для более простой модели конвейера с мгновенной загрузкой-разгрузкой [57]. Общим вопросам нейросетевого моделирования и искусственного интеллекта посвящены работы [8, 26, 42, 50, 70]. Оптимальному управлению движением и различным аспектам оптимизации посвящены работы многих авторов (см., например, [1, 2, 4–6, 9, 10, 13, 15, 21, 22, 25, 27, 28, 39, 40, 43, 49, 56, 68, 82, 88, 89, 92, 93, 95]).

Необходимо отметить, что для модели (5.1.1) учитывается существование описываемой в элементарных функциях аддитивной части управляющей функции, с учетом которой обращаются в ноль правые части системы. Указанная аддитивная часть (уравновешивающая переменная) имеет вид

$$u_{0\alpha} = c\varepsilon g m \cos(\alpha_0).$$

При получении траекторий мы учитываем тот факт, что система находится в равновесном состоянии относительно управления. В результате траектории выходят на осциллирующий режим, либо на режим достаточно плавного изменения траекторий со временем [76]. Тем не менее, на практике расчет значения уравновешивающей переменной не всегда возможен, поскольку это требует точных значений m и ε, которые могут быть получены лишь точным измерением масс и координат грузов на конвейерной ленте. Далее, применительно к модели (5.1.2), мы рассматриваем более сложную задачу, при которой предполагается, что значения m и ε являются неизвестными.

Для проведения вычислительных экспериментов с моделью (5.1.2) разработана программа на языке Julia [55]. Мы выполняем сравнительный анализ траекторий модели (5.1.2) с траекториями модели (5.1.1) с учетом использования аналогичных условий и параметров моделей. Отметим, что компьютерная программа для расчета траекторий модели (5.1.1) разработана на языке Python3 с применением библиотек numpy, scipy, matplotlib [79, 83, 91].

На рис. 5.1.1, 5.1.2 представлены результаты вычислительного эксперимента по расчету линейной скорости для случаев, когда стабилизация осуществляется посредством регулятора со скользящим режимом. Для проведения вычислительных экспериментов использован следующий набор параметров $m = 2.5$, $m_0 = 0.1$, $k = 0.5$, $g = 9.8$, $c = 0.5$, $\varepsilon = 1$.

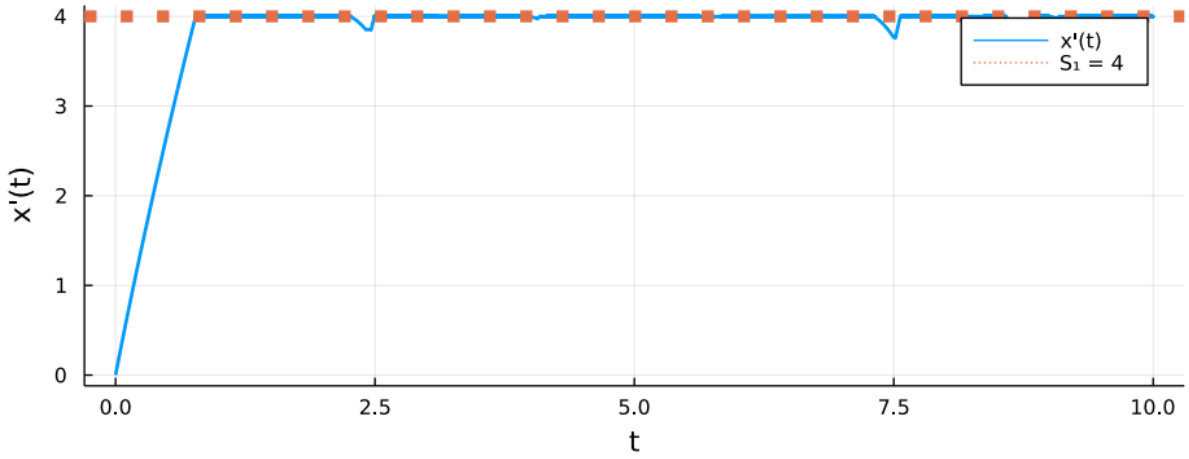


Рис. 5.1.1. График линейной скорости для модели (5.1.2)

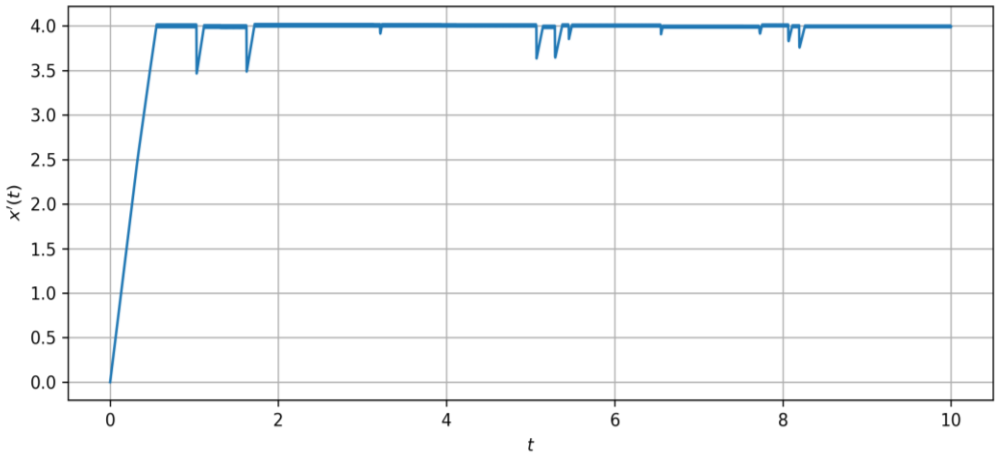


Рис. 5.1.2. График линейной скорости для модели (5.1.1)

На рис. 5.1.1, 5.1.2 сплошной линией обозначена линейная скорость (фазовая переменная $x'(t)$), пунктирной линией отмечена требуемая скорость ленты. Можно отметить, что для линейной скорости на рис. 5.1.1 достигнутое качество управления согласуется с постановкой задачи. Кратковременное уменьшение скорости в определенные моменты времени обусловлено погрузкой грузов. По сравнению с рис. 5.1.2 в графике отсутствуют резкие изменения скорости, которые связаны с мгновенной погрузкой грузов для модели (5.1.1).

На рис. 5.1.2 и рис. 5.1.3 изображены графики углового положения конвейера в модели (5.1.2) и модели (5.1.1) соответственно.

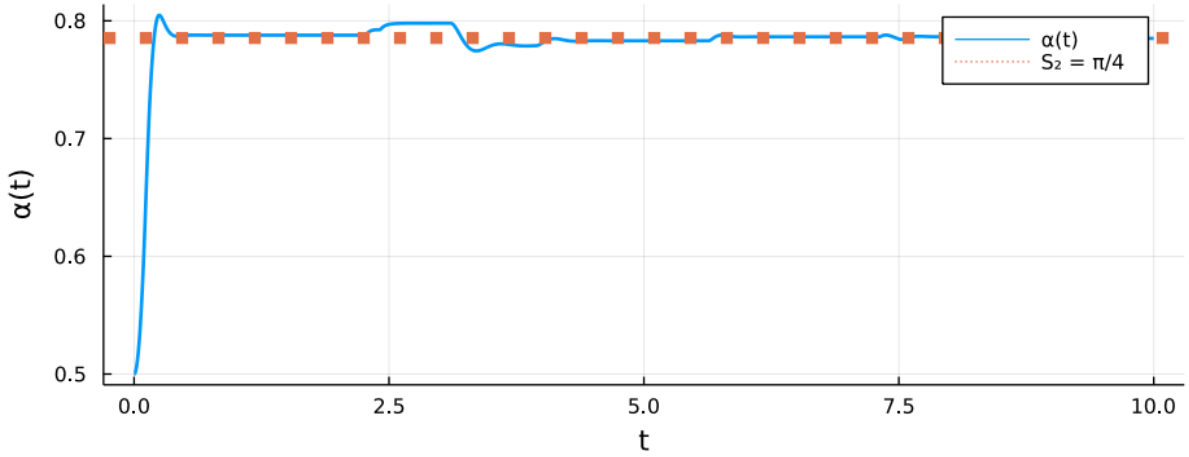


Рис. 5.1.3. График углового положения конвейера для модели (5.1.2)

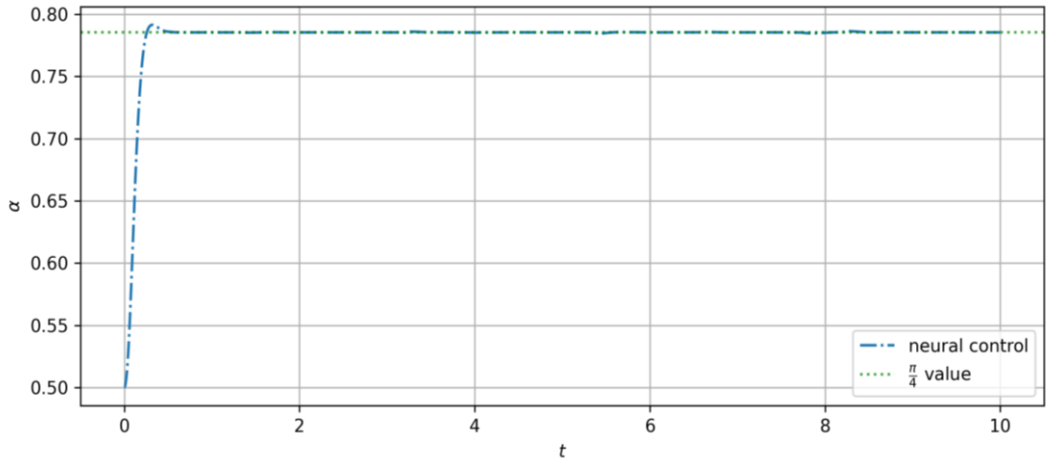


Рис. 5.1.4. График углового положения конвейера для модели (5.1.1)

На рис. 5.1.2 представлены результаты по стабилизации углового положения посредством нейросетевого регулятора с топологией 2-8-1 и с нейронами смещения.

При рассмотренном наборе параметров траектории, соответствующие угловому положению для модели (5.1.2), находятся вблизи целевого значения $S_1 = \frac{\pi}{4}$. По сравнению с графиком углового

положения для модели (5.1.1) можно отметить, что качество стабилизации углового положения модели (5.1.2) ниже. Указанный результат объясняется тем, что система с плавной погрузкой функционирует в условиях неопределенности параметров m, ε.

На рис. 5.1.5, 5.1.6 представлены графики угловой скорости конвейера.

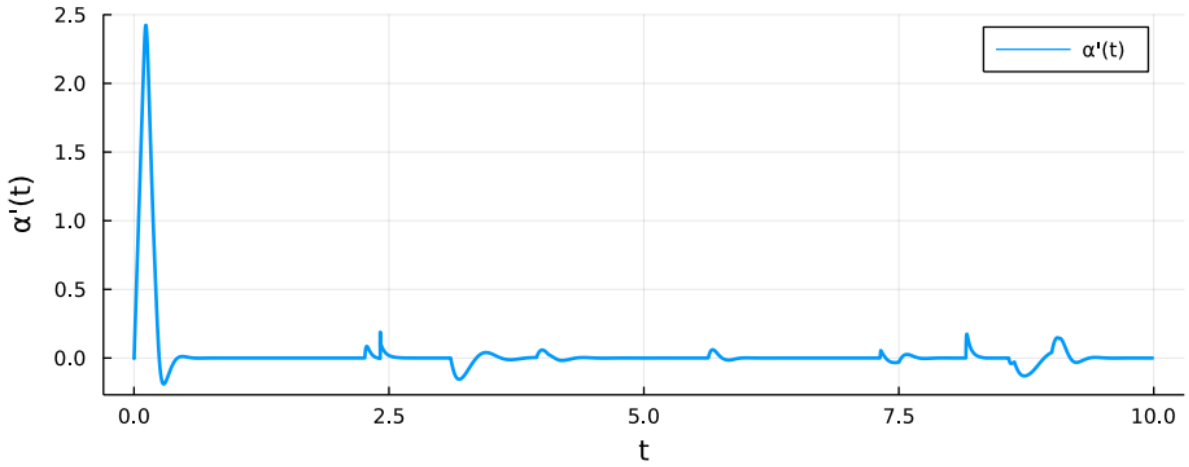


Рис. 5.1.5. График угловой скорости конвейера для модели (5.1.2)

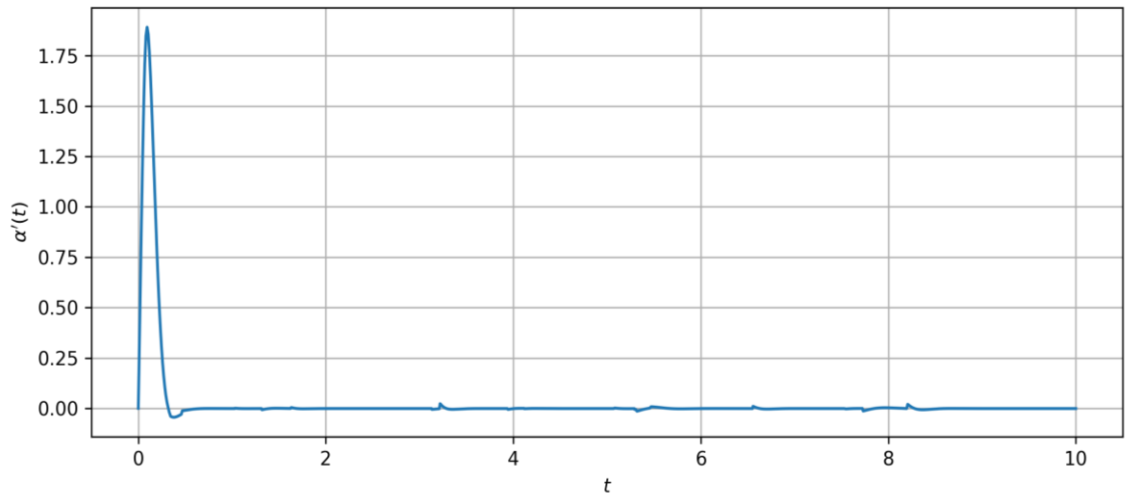


Рис. 5.1.6. График угловой скорости для модели (5.1.1)

Согласно рис. 5.1.5, угловая скорость в установившемся режиме близка к нулю, что согласуется с рассматриваемой задачей задачи. Отметим, что для модели (5.1.2) по сравнению с моделью (5.1.1)

возрастает время обучения нейросети. Таким образом, эффективность работы нейросетевого алгоритма (по отношению ко времени) для модели (5.1.2) ниже, однако информативность является более высокой.

На рис. 5.1.7, 5.1.8 изображены фазовые траектории углового положения конвейера для модели (5.1.2) и модели (5.1.1) соответственно.

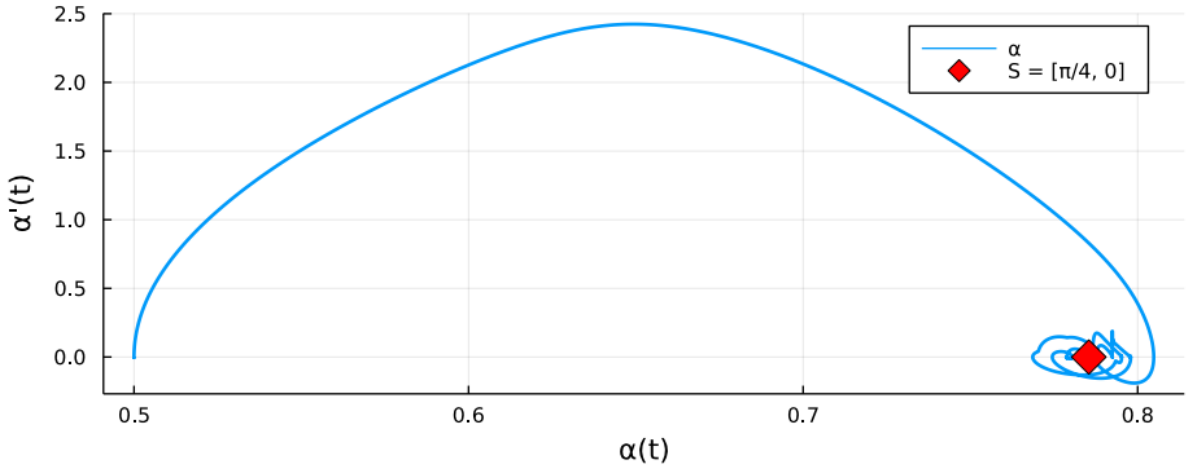


Рис. 5.1.7. Фазовая траектория углового положения системы (5.1.2)

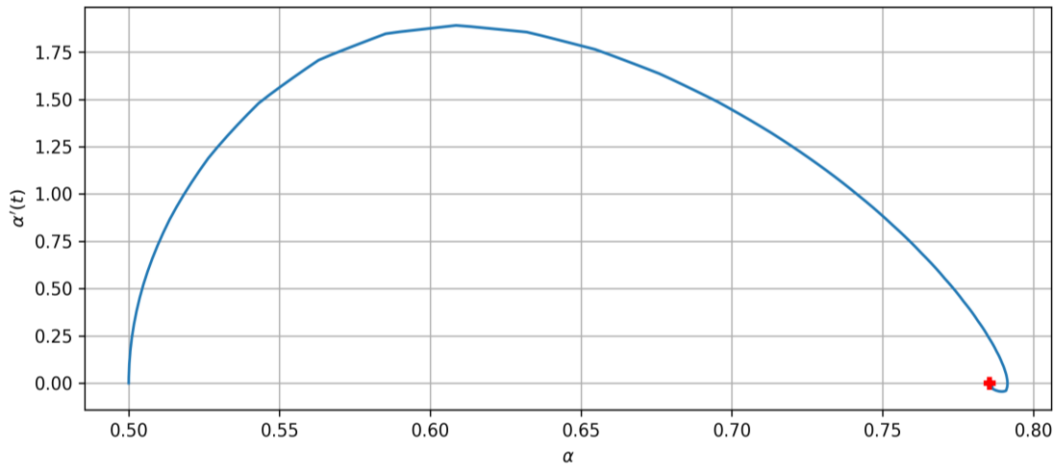


Рис. 5.1.8. Фазовая траектория углового положения системы (5.1.1)

Поведение траектории, представленной на рис. 5.1.7, соответствует множественным фокусам, лежащим в окрестности «истинного» фокуса, определенного условиями постановки задачи оптимального управления

(см. маркеры на рис. 5.1.7 и 5.1.8). Следует отметить, что траектория на рис. 5.1.7 согласуется с условием (5.1.3), задающим критерий оптимальности. Сравнение с аналогичной траекторией модели (5.1.1) (рис. 5.1.8) показывает сходный характер стабилизации с различием на конечных этапах движения.

На рис. 5.1.9 представлен график изменения импульса системы (5.1.2).

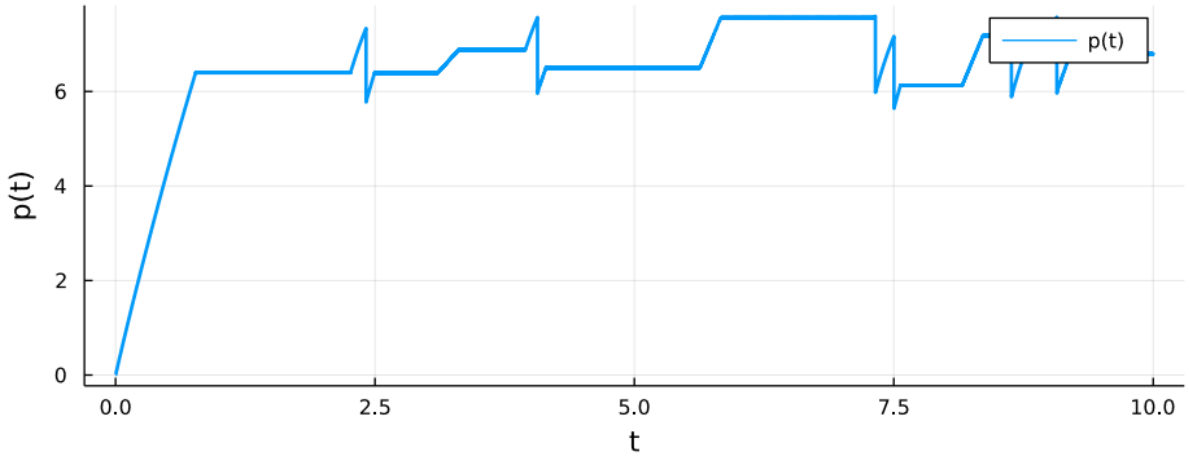


Рис. 5.1.9. График изменения импульса системы (5.1.2)

Согласно рис. 5.1.9, модель (5.1.2) с выбранными параметрами обеспечивает выход на стабилизирующий режим относительно фазовой переменной импульса системы. Представленные на рис. 5.1.9 скачкообразные переходы связаны с изменением массы грузов на конвейере, что приводит к ответному регулированию.

На рис. 5.1.10 и рис. 5.1.11 представлены график управления углом подъема конвейера для моделей (5.1.2) и (5.1.1) соответственно.

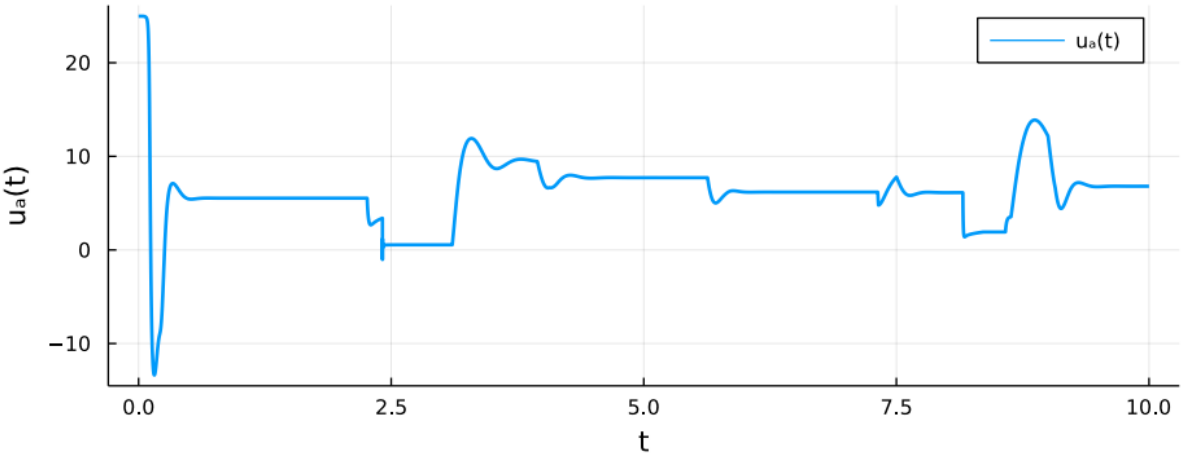


Рис. 5.1.10. График управления углом подъема конвейера для (5.1.2)

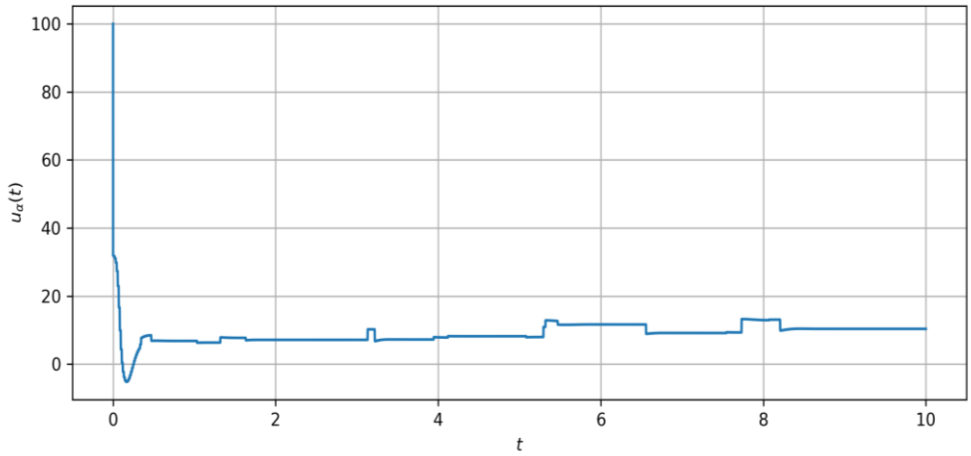


Рис. 5.1.11. График управления углом подъема конвейера для (5.1.1)

Согласно рис. 5.1.10, мы наблюдаем тренд к увеличению значения функции $u_a(t)$. При сравнении с графиком управления для модели (5.1.1) (рис. 5.1.8) получаем, что переходные процессы в модели (5.1.2) при погрузке имеют более плавный характер, а при разгрузке для обеих моделей наблюдаются скачки. Следует отметить, что осцилляции на рис. 5.1.10 связаны с ответом на возникновение незначительного перерегулирования при изменении массы. На рис. 5.1.11 указанные эффекты отсутствуют в связи с мгновенным изменением массы грузов.

При верификации модели мы получили, что скользящий режим и режим использования нейросетевого регулятора имеют сходные характеристики достижения дестабилизации системы. Отметим, что система (5.1.2) теряет устойчивость при периоде переключений управления более 0,01 секунды.

Проведенные компьютерные эксперименты позволяют выделить характерные различия в траекторной динамике между системами (5.1.1) и (5.1.2). Согласно рис. 5.1.2, 5.1.4, 5.1.6, изменение линейной и угловой скорости для модели (5.1.1) происходит скачкообразно, поскольку погрузка и разгрузка грузов происходит мгновенно. По сравнению с моделью (5.1.1), траекторная динамика модели (5.1.2) свидетельствует о наличии переходных процессов, возникающих в результате плавной погрузки (см. рис. 5.1.1, 5.1.3, 5.1.5). Изменение траекторной динамики системы (5.1.2) по сравнению с системой (5.1.1) приводит к изменению характера ответного регулирования. В частности, на рис. 5.1.10 можно отметить, что ответом на плавное изменение грузов является возникновение колебательных эффектов в сигнале управления. Согласно рис. 5.1.11, сигнал управления для модели (5.1.1) меняется скачкообразно.

Реализация скользящего режима для управления линейной скоростью приводит к эффективным результатам по стабилизации ленточного конвейера как в случае модели с мгновенной погрузкой и разгрузкой, так и в модели с плавной погрузкой груза и с мгновенной разгрузкой. Задача стабилизации угла подъема является более сложной и нетривиальной, поскольку для ее решения приходится привлекать различные типы регуляторов. Отметим, что модель (5.1.2) учитывает достаточно широкий спектр физических эффектов, но при этом является достаточно сложной для поиска оптимальных траекторий.

5.2. Особенности компьютерной реализации

Далее рассмотрим особенности компьютерной реализации модели 5.1.2. В разделе приводятся фрагменты программного кода, иллюстрирующие ключевые аспекты, связанные с реализацией переключений в разработанной модели. Основные композитные типы данных приведены на листинге 5.2.1.

Листинг 5.2.1. Композитные типы для модели конвейера

```
using DifferentialEquations
using Plots

struct Interval
    left::Float64
    right::Float64
end

Interval(dot::Float64) = Interval(dot, dot + .2)

mutable struct mParameters
    load_times::Vector{Float64}
    x::Vector{Float64}
    m_storage::Vector{Float64}
    load_intervals::Vector{Interval}
    mass_load::Vector{Float64}
    unload_times::Vector{Float64}
    mass_unload::Vector{Float64}
end
```

```
function mParameters(a::Vector{Float64})
    load_times = a
    x = Float64[]
    m_storage = Float64[]
    load_intervals = Interval.(load_times)
    mass_load = rand(range(.075,.5,10),
                     size(load_intervals))
    unload_times = load_times .+ 3.
    mass_unload = copy(mass_load)
    mParameters(load_times, x,
                m_storage, load_intervals, mass_load,
                unload_times, mass_unload)
end

P = mParameters([1., 2.5, 3., 4., 6., 6.5, 7., 8.])
in(x::Float64, r::Interval) = r.left < x <= r.right
Base.:∈(t::Float64, a::Vector{Interval}) = any(in.(t, a))
```

На листинге 5.2.1 следует выделить композитный тип Interval и определение методов для него. Указанный тип нужен для осуществления продолжительного действия при возникновении событий. Использование указанного типа необходимо по причине ненулевого времени погрузки грузов на ленту.

Далее на листинге 5.2.2 представлен программный код для реализации уравнений модели и события зацепления груза.

Листинг 5.2.2. Реализация уравнений модели и события зацепления груза

```
function conv!(dx, x, p::Union{Tuple, Matrix, Vector}, t)
    mc, mb, k, g, c, up, ua, e = p
```

```
        m = mc + mb
        dx[1] = x[2]/m
        dx[2] = up - k*(x[2]/m) - mc*g*sin(x[3])
        dx[3] = x[4]
        dx[4] = ua/(m*c*e^2) - g*cos(x[3])/e
    end

    gripCondition(u, t, integrator) = t ∈ P.load_intervals
    function gripAffect!(integrator)
        t = integrator.t
        dt = t - integrator.tprev
        mask = in.(t, P.load_intervals)
        index = findall(x -> x == 1, mask)[1]
        t₁ = P.load_intervals[index].right
        t₀ = P.load_intervals[index].left
        m₀ = P.m_storage[end]
        m₁ = P.mass_load[index] + m₀
        a = (m₁ - m₀)/(t₁ - t₀)
        integrator.p[1] += a*dt
    end
```

Для построения траекторий системы используется программный код, представленный в листинге 5.2.3.

Листинг 5.3.2. Построение траекторий

```
    check = (dt, u, p, t)-> any(u .< -100) || any(u .> 100) ?
true : false
    prob = ODEProblem(conv!,[0,0,0.5,0],(0.0,10.0), par1)
    sol = solve(prob, Rodas4(), callback = cb,
                ,unstable_check = check,
                 tstops = [P.load_times; P.unload_times])
```

Отметим, что для увеличения скорости расчета параметров производится проверка того, что изображающая точка находится вблизи начала координат (значение каждой из фазовых переменных не должно превышать 100 по модулю). В случае, если проверка не пройдена, результаты шага алгоритма оптимизации считаются заведомо несоответствующими оптимуму и отсеиваются.

Использование языка Julia в сочетании с системой Jupyter и библиотеками Plots, DifferentialEquations, BlackBoxOptim позволило достигнуть высоких показателей производительности разработанного программного обеспечения. Применение разработанного алгоритмического и инструментального обеспечения связано с возможностями реализации технологий интеллектуального управления системами конвейерного транспорта.

§ 6. Общая характеристика языка Octave

6.1. Система типов

Язык Octave представляет собой интерпретируемый мультипарадигменный язык программирования с динамической слабой типизацией. Следует отметить, что Octave создавался как открытая альтернатива коммерческому программному обеспечению MATLAB и в значительной мере совместим с ним на уровне синтаксиса. В связи с этим система типов Octave имеет прямую аналогию с системой типов MATLAB.

Среда Octave преимущественно ориентирована на численные расчеты (см., например, [7]) и придерживается парадигмы MATLAB «думай векторно» (англ. «think vectorized»). С учетом указанной парадигмы в Octave присутствует развитая система векторно-матричных вычислений. В частности, в стандартной библиотеке предусмотрены возможности операций над векторами и матрицами, таких как сложение, умножение, транспонирование и др.

Ключевой особенностью системы типов в Octave является то, что все типы «наследуются» от массива, т.е. скалярные (scalar) типы языка представляют собой массивы размера 1x1. Иерархия типов Octave представлена на рис. 6.1.1.

Важным отличием Octave от таких языков, как Python или Julia является то, что числовым типом, инициализируемым по умолчанию, будет число с плавающей запятой двойной точности (double). Это связано с ориентированностью языка на численные расчеты, также как и в случае с языком MATLAB. Использование double по умолчанию несложно проверить, используя интерпретатор языка и специальную директиву whos, которая выводит таблицу проинициализированных переменных с

указанием их типа и размера. Пример инициализации переменной представлен в листинге 6.1.1.

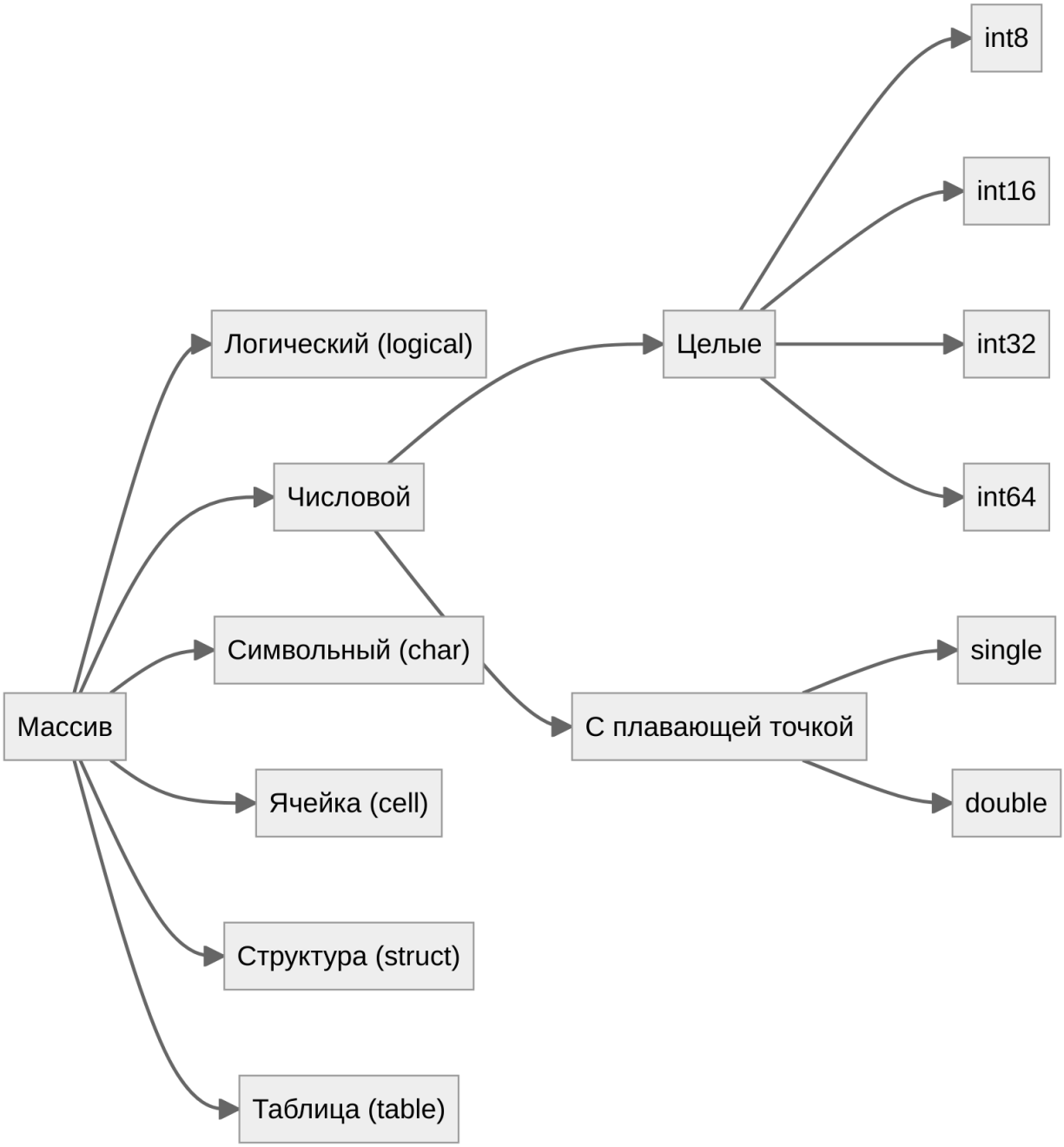


Рис. 6.1.1. Иерархия типов Octave

Листинг 6.1.1. Инициализация переменной в Octave

```
octave:13> a = 1
a = 1
octave:14> whos
Variables visible from the current scope:

variables in scope: top scope

 Attr   Name        Size                     Bytes  Class
 ====   ====        ====                     =====  =====
        a           1x1                          8  double

Total is 1 element using 8 bytes
```

Аналогично возможностям Python или Julia, при использовании Octave присутствует автоматическое приведение типов, которое производится при совершении арифметических операций. Однако алгоритм приведения типов может отличатся от алгоритмов приведения языков общего назначения. В листинге 6.1.2. представлена операция с различающимися типами.

Листинг 6.1.2. Операция с различными типами

```
octave:16> a + int32(3)
ans = 4
octave:17> whos
Variables visible from the current scope:

variables in scope: top scope

 Attr   Name        Size                     Bytes  Class
 ====   ====        ====                     =====  =====
        a           1x1                          8  double
        ans         1x1                          4  int32

Total is 2 elements using 12 bytes
```

Следует отметить, что в соответствии с листингом 6.1.2 производится не типичное для современных языков общего назначения приведение из double в int32. Кроме того, такое приведение будет осуществляется с округлением double до целого при необходимости, что следует учитывать при разработке программного обеспечения.

Логический тип данных, по аналогии с Python и Julia, принимает значения true или false. Присутствует автоматическое приведение типов к числовым данным, что проиллюстрировано в таблице 6.1.1.

Таблица 6.1.1. Приведение логических типов в Octave

Из логического в числовой		**Из числового в логический**	
logical	number	number	logical
i = true	i = 1	i ≠ 0	i = true
i = false	i = 0	i = 0	i = false

Символьный тип данных в Octave, который используется для хранения как отдельных символов, так и строк, по принципу работы схож с массивом строк `char *`, который используется в языке C. Отдельный символ может быть извлечен путем индексирования. Присутствуют встроенные функции для работы с массивами символов как со строками. Следует отметить, что индексирование содержимого массивов в Octave производится с помощью круглых скобок, аналогично вызову функций. Приведем пример индексирования строк:

```
octave:1> x = "abcd"
x = abcd
octave:2> x(1)
ans = a
```

Следует отметить, что в отличие от списков Python и массивов Julia массивы Octave не предназначены для хранения различающихся типов в

качестве элементов. Для того, чтобы обойти данное ограничение, существует тип данных Ячейка (cell). Указанный тип данных представляет собой контейнер, который может хранить данные любого другого типа. Массив ячеек (cell array) может быть создан с помощью следующего синтаксиса:

```
c = {"a string", rand(2, 2)};
```

Индексирование из массива ячеек также может быть осуществлено с использованием фигурных скобок, например:

```
c{1}
     ⇒ ans = a string
```

Для записи логически упорядоченных данных существуют два типа данных в Octave: структура и таблица.

Структура в Octave представляет собой массив ячеек, каждая из которых содержит информацию о конкретном элементе структуры. Например, структура может содержать следующие поля:

– имя элемента;

– значение элемента;

– тип элемента (например, число, строка, логический).

Для создания структуры в Octave используется ключевое слово struct. Например, для создания структуры с двумя полями – «имя» и «значение», можно использовать следующий код:

```
my_structure = struct("name", "Jonh", "value", 42);
```

Внутри структуры можно обращаться к полям по имени. В случае, если в переменной содержится более одной структуры, обращение по индексу выведет все поля. Например:

```
% обращение к полю по имени
printf('Значение элемента %s равно %d',
my_structure.name, my_structure.value);
```

Другим важным типом данных в Octave являются таблицы. Указанный тип данных позволяет хранить информацию в виде множества записей, каждая из которых содержит элементы различного типа. В определенном контексте таблица может выступать более удобной заменой массива структур. Тем не менее, на момент выхода Octave 8.2.0 (2023 г.) реализация функции table не завершена.

Язык Octave, также как Python, поддерживает объектно-ориентированное программирование. Рассмотрение примера с компьютерным моделированием движения частицы с применением Octave будет аналогичен варианту на Python по принципу построения (см. Раздел 2.2). Объектно-ориентированному программированию на Octave посвящены работы [3, 63, 81]. Применение Octave в конкретных научных задачах рассмотрен в [4] и в других работах.

Далее рассмотрим пример с построением компьютерной модели движения летательного аппарата в условиях вариативности граничных условий.

6.2. Анализ модели технической системы с переключениями и особенности программной реализации

Исследования различных моделей систем с переключениями проводились в [11, 17, 37, 38, 44, 45, 58, 59, 71] и в других работах. В

данном разделе представим результаты моделирования и направления программной реализации двумерной модели движения технической системы с переключениями, а именно, такой модели транспортной системы с переменной тягой, которая учитывает условие минимальности расхода топлива.

Пусть движение осуществляется в декартовых координатах x, y с начальной точкой (0,0) и делится на два этапа. На первом этапе, соответствующем интервалу $(0, t_1)$, объект перемещается на плоскости xoy под воздействием постоянной векторной тяги (p, q) до достижения высоты h. На втором этапе, соответствующем интервалу (t_1, t_2), объект перемещается с постоянной векторной тягой $(-b, s)$, достигая конечной точки $L(l, 0)$. На объект действует сила тяготения. Движение считается допустимым, если значение $x(t)$ является постоянно возрастающим от 0 до l, $y(t)$ возрастает на первом этапе до высоты h и убывает до нуля на втором этапе, т.е.

$$\forall x(t), t \in (0,t_2): \dot{x} > 0, \quad \forall x(t), t \in (0,t_1]: \dot{y} \geq 0, \quad \forall y(t) \in (t_1,t_2]: \dot{y} < 0.$$

При этом в момент приземления t_2, т.е. когда $y(t_2) = 0$, получаем $y(t_2) = l$.

Выбор значений векторной тяги производится из строго положительных интервалов

$$p(p_1, p_2),\ q(q_1, q_2),\ b(b_1, b_2),\ s(s_1, s_2). \tag{6.2.1}$$

Критерий оптимальности можно записать в интегральном виде:

$$\int_0^{t_1} (p+q)\, dt + \int_{t_1}^{t_2} (b+s)\, dt \to \min. \tag{6.2.2}$$

Физический смысл критерия заключается в минимизации затрат на создание векторной тяги и уменьшении времени достижения конечной точки. Задача состоит в нахождении p, q, b, s, t_1, t_2, удовлетворяющих критерию (6.2.2).

Предлагается следующий алгоритм решения задачи. Сначала согласно подходу, предложенному в [34], выбирается однозначная реализация, состоящая из единственных значений этих параметров. Тогда дифференциальные включения

$$\begin{matrix} m\ddot{x} \in p, \\ m\ddot{y} \in q - mg, \end{matrix} \quad 0 \le t < t_1,$$

$$\begin{matrix} m\ddot{x} \in -b, \\ m\ddot{y} \in s - mg, \end{matrix} \quad t_1 \le t \le t_2,$$

описывающие движение объекта, сводятся к системам дифференциальных уравнений второго порядка

$$\begin{matrix} m\ddot{x} = p, \\ m\ddot{y} = q - mg, \end{matrix} \quad 0 \le t < t_1,$$

$$\begin{matrix} m\ddot{x} = -b, \\ m\ddot{y} = s - mg, \end{matrix} \quad t_1 \le t \le t_2. \qquad (6.2.3)$$

Модели, описываемые уравнениями вида (6.2.3), и рассматриваемые далее обобщения могут найти применение в задачах транспортной динамики, системах доставки и логистики, робототехнике и моделировании движения тел под воздействием постоянных возмущений. С помощью замены

$$P_1 = p/m, \quad Q_1 = (q - mg)/m, \quad R_1 = b/m, \quad S_1 = (s - mg)/m,$$

на первом этапе система примет вид:

$$\begin{matrix} \ddot{x} = P_1, \\ \ddot{y} = Q_1, \end{matrix} \qquad (6.2.4)$$

с начальными условиями $x(0) = y(0) = \dot{x}(0) = \dot{y}(0) = 0$, $y(t_1) = h$. Решение системы (6.2.4) имеет вид:

$$x = \frac{P_1 t^2}{2}, \quad y = \frac{Q_1 t^2}{2}. \qquad (6.2.5)$$

На втором этапе движение описывается системой

$$\dot{x} = -R_1, \quad x(t_1) = \frac{P_1 t_1^2}{2}, \quad \dot{x}(t_1) = P_1 t_1,$$

$$\ddot{y} = -S_1, \quad y(t_1) = \frac{Q_1 t_1^2}{2} = h, \quad \dot{y}(t_1) = Q_1 t_1, \quad t_1 \le t \le t_2,$$

решение которой имеет вид

$$\begin{aligned} x(t) &= \frac{-R_1 t^2}{2} + (P_1 + R_1) t_1 t - \frac{(P_1 + R_1) t_1^2}{2}, \\ y(t) &= \frac{-S_1 t^2}{2} + (Q_1 + S_1) t_1 t - \frac{(Q_1 + S_1) t_1^2}{2}, \quad t_1 \le t \le t_2. \end{aligned} \tag{6.2.6}$$

При условиях $x(t_2) = l,\ x'(t_2) = 0,\ y(t_2) = 0,\ y(t_1) = h,\ y'(t_2) = 0$ система (6.2.6) не имеет аналитического решения.

Полагая $\tau = \frac{t_2}{t_1}$, с учетом (6.2.5), (6.2.6) получим:

$$P_1 = \frac{2l}{t_1^2}\tau, \quad R_1 = \frac{2l}{t_1^2 \tau(\tau - 1)}, \quad S_1 = \frac{2(2\tau - 1)h}{t_1^2 (\tau - 1)^2}, \quad Q_1 = \frac{2h}{t_1^2}.$$

Далее найдем τ и t_1. Воспользуемся условием оптимальности (6.2.2), которое запишем в виде

$$F = \int_0^{t_1} (p + q)\, dt + \int_{t_1}^{t_2} (b + s)\, dt \to \min.$$

где

$$F(t_1, \tau) = m t_1 (P_1 + Q_1 + g) + (R_1 + S_1 + q)(t_2 - t_1).$$

Для нахождения критических точек продифференцируем F по t_1 и по τ, полученную систему приравняем к нулю:

$$\begin{aligned} 4\frac{l}{\tau} + 6h + t_1^2 \tau g + 2\frac{h}{\tau - 1} &= 0, \\ 4\frac{l}{\tau} - t_1^2 \tau g + 2h\frac{\tau}{(\tau - 1)^2} &= 0. \end{aligned}$$

Отсюда, ввиду $\tau > 1$ имеем

$$\tau = 1 + \frac{1}{\sqrt{3}}.$$

Тогда

$$t_1 = \sqrt{\frac{4l(\tau-1)^2 + 2h\tau^2}{\tau^2(\tau-1)^2 g}}$$

и

$$t_2 = \tau\sqrt{\frac{4l(\tau-1)^2 + 2h\tau^2}{\tau^2(\tau-1)^2 g}}.$$

Исходя из (6.2.2), (6.2.6), получим

$$p = \frac{2l\tau(\tau-1)^2 gm}{4l(\tau-1)^2 + 2h\tau^2},\ q = \frac{2h\tau^2(\tau-1)^2 gm}{4l(\tau-1)^2 + 2h\tau^2} + mg,$$

$$b = \frac{2l\tau(\tau-1)gm}{4l(\tau-1) + 2h\tau^2},\ s = \frac{2h\tau^2(2\tau-1)gm}{4l(\tau-1)^2 + 2h\tau^2} + mg.$$

Проведенные вычисления показывают прямолинейную зависимость силы тяги *p, q, b, s* от *h, l* и массы объекта.

На основе полученных выражений разработана тестовая программа построения траекторий движения изучаемой системы на встроенном языке системы Octave [3]. Указанная программа подготовлена с учетом интеграции в соответствующий модуль программного комплекса, предназначенного для моделирования технических систем с переключениями [17, 45].

Для моделирования рассматривается пространство изменяющихся параметров допустимого движения. Фазовые траектории двумерной системы с учетом изменения значений *h* и *m* представлены на рис. 6.2.1.

Отмечается согласованность численных методов интегрирования и аналитических траекторий в том смысле, что траектории, полученные с использованием численных методов при заданных начальных условиях совпадают с кривыми аналитических решений.

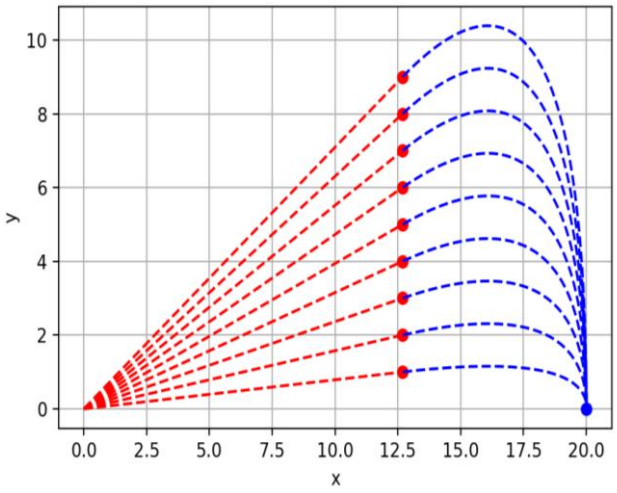


Рис. 6.2.1. Фазовые траектории с учетом точек переключения для двумерной системы (6.2.3)

Далее представлена компьютерная реализация модели (6.2.6) с использованием языка Octave.

Листинг 6.2.1. Фрагмент программы для построения траекторий трехмерной модели с учетом вариативности позиционирования конечной точки

```
%==================================
% Установка начальных параметров и условий
%==================================

Cd=struct('tu',1,'l',10,'h',5,'g',9.8,'m',1,'t1',1,'t2',1,'dt
'
↳ ,1,'p',1,'q',1,'r',1,'s',1,'P1',1,'R1',1,'S1',1,'Q1',1,'R'
↳ ,1);

Cd=SetConditions(Cd,true);
%============================
% Постройка первоначального графика
%============================
```

```
td=0:0.1:Cd.t1;
x0=(Cd.P1*td.^2)/2;
y0=(Cd.Q1*td.^2)/2;
hA=axes;
size1=size(x0);
size1=size1(2);
plot3(x0,zeros(size1,1),y0);
SetCanv(hA);
hold on;

td=Cd.t1:0.1:Cd.t2;
xt=-((Cd.R1*td.^2)/2)+((Cd.P1+Cd.R1)*Cd.t1*td)-
↳ (((Cd.P1+Cd.R1)*(Cd.t1^2))/2);
yt=((-Cd.S1*(td.^2))/2)+((Cd.Q1+Cd.S1)*Cd.t1*td)
↳ -((Cd.Q1+Cd.S1)*(Cd.t1^2))/2;
size2=size(xt);
size2=size2(2);
plot3(xt, zeros(size2,1), yt);
SetCanv(hA);
Z=[xt', yt'];
%=====================
% Численное решение ОДУ %
%=====================
% пусть x - y(1), y - y(2), x' - y(3) = dy(1),
% y'- y(4) = dy(2), x'' = dy(3), y'' = dy(4), тогда
% -> movement.m
% начальные условия - (P1*dt^2)/2, (Q1*dt^2)/2, P1*dt, Q1*dt
IPoint=InitY(Cd);
Cd.R=1;
dr=0.05;
[T,Y] = ode45(@(t,y) movement(t,y,Cd.R1,Cd.S1),
↳ [Cd.dt,Cd.t2],IPoint);
Circle(Cd,Y);
SetCanv(hA);
```

```
numfile=1;
while Cd.R>dr
    [T,Y] = ode45(@(t,y) movement(t,y,Cd.R1,Cd.S1),
       ↳Cd.dt,Cd.t2],IPoint);
    L=[Y(:,1), Y(:,2)];
    Gamma=find(L(:,1) >= (max(Z(:,1)))-Cd.R/4);
    Direct1=L(Gamma,1);
    Direct2=L(Gamma,2);
    plot3(Direct1(1),zeros(size(Direct1(1)),1),Direct2(1),
    ↳'-o');
     plot3(Direct1,zeros(size(Direct2),1),Direct2,'LineWidth',
    ↳2);
    SetCanv(hA);
    print('-dpng','-r300', strcat('Кривая',
    ↳ num2str(numfile)));
    sizeofD1=size(Direct1);
    sizeofD1=sizeofD1(1);
    Counter=1;
    Arr=rand(1,sizeofD1)*pi;
    sizeoArr=size(Arr);
    sizeoArr=sizeoArr(2);
    if Cd.R >= 0.25
      while Counter<sizeoArr
            R0=Direct1(sizeofD1)-Direct1(1);
            VectorZ=R0*cos(Arr(Counter));
            ArrayZ=linspace(0,VectorZ,sizeofD1);
            Direct1=(Direct1-Direct1(1))*cos(Arr(Counter)) +
            ↳ Direct1(1);
            plot3(Direct1,ArrayZ,Direct2,'--');
            SetCanv(hA);
            Circle(Cd,Y);
            SetCanv(hA);
            Counter=Counter+1;
      end
```

```
    end
    Z=[Y(:,1), Y(:,2)];
    Cd=SetConditions(Cd,false);
    IPoint=InitY(Cd);
    numfile=numfile+1;
end;
SetCanv(hA);
hold off;
%===============
% Система уравнений
%===============
function dy = movement(t,y,R1,S1)
     dy = zeros(4,1);
     dy(3) = -R1;
     dy(4) = -S1;
     dy(1) = y(3);
     dy(2) = y(4);
end
%======================
% Функция расчета параметров
%======================
function Cnd = SetConditions(Cd, flag)
%====================
Функция обновляет начальные условия и параметры при каждом
переключении.
Тело функции приводится из-за громоздкости описания формул.
%======================
end

%============================
% Функция расчета начальных условий
%============================

function yy = InitY(Cd)
     y(1)=(Cd.P1*Cd.dt^2)/2;
```

```
    y(2)=(Cd.Q1*Cd.dt^2)/2;
    y(3)=Cd.P1*Cd.dt;
    y(4)=Cd.Q1*Cd.dt;
    yy=y;
end

%==============================
% Функция задания параметров отрисовки
%==============================

function s = SetCanv(hA)
    set(hA, 'XLim', [-1 12]);
    set(hA, 'YLim', [-1 8]);
    set(hA, 'ZLim', [-1 8]);
    xlabel('Ox');
    ylabel('Oz');
    zlabel('Oy');
    end
%==============================
% Функция отрисовки кругов положений
%==============================
function cir = Circle(Cd, Y)
    sizeY=size(Y);
    xcirclepos = Y(sizeY(1),1);
    CircleT = 1:0.1:10;
    CircleX = Cd.R * sin(CircleT) + xcirclepos;
    CircleZ = Cd.R * cos(CircleT);
    plot3(CircleX, CircleZ,
    ↳ zeros(size(CircleT)) ,'LineWidth', 2);
end
```

В листинге 6.2.1 представлен фрагмент кода для компьютерной модели, оформленный в процедурном стиле программирования. Для хранения

параметров модели используется структура `Cd`. Решение дифференциальных уравнений осуществляется с помощью функции `ode45` (метод Рунге-Кутты 4(5) порядка), вывод результатов производится в графической форме с помощью функции `plot3` (построение трехмерной траектории средствами стандартной библиотеки языка Octave). Предусмотрена возможность сохранения промежуточных траекторий в файлы формата *.png с помощью функции `print`. Отметим, что несмотря на некоторую громоздкость процедурного подхода к компьютерному моделированию, указанный подход позволяет достичь определенной степени реиспользуемости кода.

§ 7. Задачи и упражнения

7.1. Рассмотреть обобщение модели движения частицы (2.2.1) на случай k частиц со случайно сгеренированными начальными состояниями. Разработать компьютерную программу для случая $k = 10$ на языке Python с применением листинга 2.2.1. Выполнить визуализацию траекторий.

7.2. Рассмотреть обобщение двумерной модели движения частицы (2.2.1) на трехмерный случай, соответствующий движению частицы в пространстве. Разработать компьютерную программу на языке Python с применением листинга 2.2.1. Выполнить визуализацию траекторий.

7.3. Рассмотреть обобщение двумерной модели движения частицы (2.2.1), когда сила тяготения прямо пропорциональна значению x. Разработать компьютерную программу на языке Python с применением листинга 2.2.1.

7.4. Рассмотреть обобщение двумерной модели движения частицы (2.2.1) на переключаемый случай, когда существует прямая, «отбивающая» частицы в противоположную сторону с линейным уменьшением скорости. Разработать компьютерную программу на языке Julia с применением листинга 4.2.1.

7.5. Для системы Лоренца

$$\begin{aligned} \dot{x} &= -\sigma x + \sigma y, \\ \dot{y} &= -xz + rx - y, \\ \dot{z} &= xy - bz, \end{aligned} \tag{7.1}$$

где b, r, $\sigma > 0$,

а) найти состояния равновесия с помощью компьютерной программы на языке Python с применением библиотеки SymPy;

б) составить компьютерную программу поиска траекторий на языке Python при различных наборах параметров с применением библиотеки SciPy;

в) исследовать хаотическое поведение системы, возникающее при значениях параметров $\sigma = 10$, $b = 8/3$, $r = 28$.

Материал по аналитическому и качественному исследованию системы (7.1) можно найти в [35, 48].

7.6. Построить и изучить модель оптимального управления для модели транспортной системы, задаваемой конечномерным дифференциальным уравнением вида

$$m\ddot{x} = PT + mG + D, \quad P \in \Gamma \tag{7.2}$$

где $P_{m\times n}$ – матрица коэффициентов, $T = (0, t^1, t^2, \ldots, t^m)^*$, m – масса, G – вектор потенциального поля (сила тяжести). Здесь Γ – дискретное отображение, ставящее в соответствие определенным моментам времени соответствующую матрицу коэффициентов, D – вектор внешних возмущений, который задается вектор-функцией $D(t, x)$. Для указанной функции справедливо неравенство: $\forall t, x : \| D(t,x) \| < \lambda$, где λ – ограничение на максимальное возмущение. Модель (7.2) может описывать движение летательных аппаратов, транспортных систем и других систем с переключениями режимов функционирования. Ряд частных случаев модели (7.2), а также результаты исследований некоторых моделей с переключениями приведены в [38, 45, 58].

Требуется изучить модель (7.2) для случая неоднородного поля тяготения mG с учетом

$$D = 0, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad G = \begin{pmatrix} 0 \\ -g \end{pmatrix}, \quad g = 2x_1x_2.$$

Критерий качества задать с учетом минимизации действующих в системе сил. Алгоритм переключений состоит в следующем.

Шаг 1. Задать начальные условия I для модели (7.2).

Шаг 2. Совершить одну итерацию численного алгоритма интегрирования.

Шаг 3. Проверка условия останова. Если оно выполняется, алгоритм закончен. В противном случае нужно перейти к следующему шагу.

Шаг 4. Если $t = \frac{t_2}{2}$, то перейти к следующему шагу. В противном случае вернуться к шагу 2.

Шаг 5. $t_2 := t_2 - t$; $t = 0$.

Шаг 6. Расчет новых значений *P*. Переход к шагу 2.

Составить компьютерную программу на языке Julia для реализации алгоритма переключений в системе с возрастающей частотой.

7.7. Построить и изучить модель оптимального управления для модели, задаваемой уравнением вида (7.2), для случая неоднородного поля тяготения *mG* с учетом

$$D = 0,\ x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},\quad G = \begin{pmatrix} 0 \\ -g \end{pmatrix},\quad g = \sqrt{x_1^2 + x_2^2}\ .$$

Критерий качества задать с учетом минимизации действующих в системе сил. Составить компьютерную программу на языке Python для реализации алгоритма переключений в системе с возрастающей частотой.

7.8. Построить и изучить модель оптимального управления для модели, задаваемой уравнением вида (7.2), для случая неоднородного поля тяготения *mG* с учетом

$$D = 0,\ x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},\quad G = \begin{pmatrix} 0 \\ -g \end{pmatrix},\quad g = \sin(x_1)\cos(x_2).$$

Критерий качества задать с учетом минимизации действующих в системе сил. Составить компьютерную программу на языке Python для реализации алгоритма переключений в системе с возрастающей частотой.

7.9. Рассмотреть модель поэтапного усвоения знаний, описанную в разделе 3.1 настоящего пособия:

$$\begin{aligned} \dot{Z}_1 &= k\alpha_1\left(U - Z\right) - k\alpha_2 Z_1 - \gamma_1 Z_1, \\ \dot{Z}_2 &= k\alpha_2 Z_1 - \gamma_2 Z_2, \end{aligned} \tag{7.3}$$

где $Z = Z_1 + Z_2$, k – индикатор наличия процесса обучения (0 или 1), α_1, α_2 – коэффициенты скорости усвоения знаний, γ_1, γ_2 – коэффициенты забывания для Z_1 и Z_2 соответственно (определяются как $\frac{1}{\tau}$, где τ – время снижения знаний в e раз, U – уровень требований учителя (сложность заданий).

Составить компьютерную программу на языке Julia для изучения траекторной динамики с учетом начальных условий $Z_1(0) = 0.1$, $Z_2(0) = 0.1$, а также следующих параметров:

а) $\alpha_1 = 0.6$, $\alpha_2 = 0.2$, $\gamma_1 = 1/3$, $\gamma_2 = 1/6$;

б) $\alpha_1 = 0.5$, $\alpha_2 = 0.15$, $\gamma_1 = 1/3$, $\gamma_2 = 1/6$;

в) $\alpha_1 = 0.4$, $\alpha_2 = 0.1$, $\gamma_1 = 1/3$, $\gamma_2 = 1/6$.

Выполнить сравнительный анализ результатов расчета траекторий с различными параметрами системы (7.3).

СПИСОК ЛИТЕРАТУРЫ

1. *Александров В.В., Болтянский В.Г., Лемак С.С., Парусников Н.А., Тихомиров В.М.* Оптимальное управление движением. М.: Физматлит, 2005.

2. *Алексеев В.М., Тихомиров В.М., Фомин С.В.* Оптимальное управление. М.: Наука, 1979.

3. *Алексеев Е.Р., Чеснокова О.В*. Введение в Octave для инженеров и математиков. М.: ALT Linux, 2012.

4. *Алтунин К.К., Штром Е.С.* Разработка электронного образовательного ресурса «Квантовая механика наносистем с Octave»// Наука онлайн. 2019. №4(9). С. 42–73.

5. *Афанасьев В.Н., Колмановский В.Б., Носов В.Р.* Математическая теория конструирования систем управления. М.: Высшая школа, 1998.

6. *Афанасьев В.Н., Преснова А.П.* Формирование алгоритмов оптимизации нестационарных систем управления на основе необходимых условий оптимальности // Мехатроника, автоматизация, управление. 2018. Т. 19. № 3. С. 153–159.

7. *Бабенко К.И*. Основы численного анализа. М.: Наука, 1986.

8. *Беркинблит М.Б.* Нейронные сети. М.: МИРОС и ВЗМШ РАО, 1993.

9. *Благодатских В.И.* Введение в оптимальное управление (линейная теория): Учебник / Под ред. *В.А. Садовничего.* М.: Высшая школа, 2001.

10. *Васильев С.Н.* К интеллектному управлению // Нелинейная теория управления и ее приложения. М.: Физматлит, 2000. С. 57–126.

11. *Васильев С.Н., Маликов А.И.* О некоторых результатах по устойчивости переключаемых и гибридных систем // Сборник статей «Актуальные проблемы механики сплошной среды. К 20-летию ИММ КазНЦ РАН». Казань: Фолиант, 2011. Т. 1. С. 23–81.

12. *Воронов А.А.* Введение в динамику сложных систем. М.: Наука, 1985.

13. *Газизов Т.Т.* Методы глобальной оптимизации: Учебное пособие. Томск: В-Спектр, 2017.

14. *Гостев В.И.* Нечеткие регуляторы в системах автоматического управления. Киев: Радіоматор, 2008.

15. *Громов Ю.Ю., Земской Н.А., Лагутин А.В., Иванова О.Г., Тютюнник В.М.* Специальные разделы теории управления. Оптимальное управление динамическими системами. Тамбов: Изд-во ТГТУ, 2007.

16. *Демидова А.В., Дружинина О.В., Масина О.Н., Петров А.А.* Разработка алгоритмического и программного обеспечения моделирования управляемых динамических систем с применением символьных вычислений и стохастических методов // Программирование. 2023. № 2. С. 54–68.

17. *Дружинина О.В., Корепанов Э.Р., Белоусов В.В., Масина О.Н., Петров А.А.* Развитие инструментального обеспечения отечественной вычислительной платформы «Эльбрус 801-РС» в задачах нейросетевого моделирования нелинейных динамических систем // Нелинейный мир. 2021. Т. 19. № 1. С. 15–28.

18. *Дружинина О.*В., *Масина О.Н.* Методы анализа устойчивости динамических систем интеллектного управления. М.: Изд. группа URSS, 2016.

19. *Дружинина О.В.*, *Масина О.Н.*, *Петров А.А*. Модель управления движением транспортной системы с учетом условий оптимальности, многозначности и вариативности // Транспорт: наука, техника, управление. 2017. № 4. С. 3–9.

20. *Дружинина О.В., Масина О.Н., Петров А.А*. Разработка алгоритмов поиска параметров моделей управляемых систем с учетом условия минимальности расхода топлива // Вестник Рязанского

государственного радиотехнического университета. 2018. № 2. Вып. 64. С. 46–54.

21. *Евтушенко Ю.Г., Посыпкин М.А.* Применение метода неравномерных покрытий для глобальной оптимизации частично целочисленных нелинейных задач // ЖВМ и ВМ. 2011. Т. 51. № 8. С. 1376–1389.

22. *Емельянов С.В., Коровин С.К.* Новые типы обратной связи: управление при неопределенности. М.: Физматлит, 1997.

23. *Заде Л.А.* Понятие лингвистической переменной и его применение к принятию приближенных решений. М.: Мир, 1976.

24. *Ивашкин Ю.А., Назойкин Е.А.* Мультиагентное имитационное моделирование процесса накопления знаний // Программные продукты и системы. 2011. № 1. С. 47–52.

25. *Кажаров А.А., Курейчик В.М.* Муравьиные алгоритмы для решения транспортных задач. Известия РАН. Теория и системы управления. 2010. № 1. С. 32–45.

26. *Каллан Р.* Основные концепции нейронных сетей. М.: Вильямс, 2001.

27. *Карпенко А.П.* Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой. М.: МГТУ им. Н.Э. Баумана, 2014.

28. *Каширина И.А., Демченко М.В.* Исследование и сравнительный анализ методов оптимизации, используемых при обучении нейронных сетей // Вестник ВГУ. Сер. Системный анализ и информационные технологии. 2018. № 4. С. 123–132.

29. *Кожубаев Ю.Н., Семенов И.М.* Системы управления ленточным конвейером // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. 2014. 2(195). С. 181–186.

30. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ / Под ред. И.В. Красикова. 2-е изд. М.: Вильямс, 2005.

31. *Леонтьев Л.П., Гохман О.Г.* Проблемы управления учебным процессом: Математические модели. Рига: Зинатне, 1984.

32. *Майер Р.В.* Кибернетическая педагогика: имитационное моделирование процесса обучения. Глазов: Глазовский государственный педагогический институт, 2014.

33. *Майер Р.В.* Многокомпонентная модель обучения и ее использование для исследования дидактических систем // Фундаментальные исследования. 2013. № 10. С. 2524–2528.

34. *Масина О.Н.* Вопросы управления движением транспортных систем // Транспорт: наука, техника, управление. 2006. № 12. С. 10–12.

35. *Масина О.Н., Дружинина О.В., Рапопорт Л.Б.* Элементы теории устойчивости математических моделей управляемых систем. Учебное пособие. Елец: Елецкий государственный университет им. И.А. Бунина, 2019.

36. *Масина О.Н., Петров А.А., Дружинина О.В.* Основы методологии научных исследований в области моделирования сложных управляемых систем. Учебное пособие. Елец: Елецкий государственный университет им. И.А. Бунина, 2022. 86 с.

37. *Масина О.Н., Петров А.А., Дружинина О.В., Рапопорт Л.Б.* Моделирование и стабилизация нелинейных управляемых систем. Учебное пособие. Елец: Елецкий государственный университет им. И.А. Бунина, 2020.

38. *Масина О.Н., Петров А.А., Дружинина О.В., Рапопорт Л.Б.* Моделирование управляемых систем с применением методов стабилизации и алгоритмов поиска оптимальных траекторий. Учебное

пособие. Елец: Елецкий государственный университет им. И.А. Бунина, 2021.

39. *Матренин П.В., Гриф М.Г., Секаев В.Г.* Методы стохастической оптимизации. Новосибирск: Изд-во НГТУ, 2016.

40. *Мирошник И.В., Никифоров В.О., Фрадков А.Л.* Нелинейное и адаптивное управление сложными динамическими системами. СПб.: Наука, 2000.

41. *Новиков Д.А.* Теория управления образовательными системами. М.: Народное образование, 2009.

42. *Осипов Г.С.* Лекции по искусственному интеллекту. М.: Изд. Группа URSS, 2018.

43. *Охорзин В.А., Сафонов К.В.* Теория управления: Учебник. Красноярск: Изд-во Сиб. гос. аэрокосмич. ун-та, 2011.

44. *Петров А.А.* Моделирование и построение алгоритма поиска оптимальных параметров управляемых динамических систем, описываемых дифференциальными включениями // Нелинейный мир. 2017. Т. 15. № 4. С. 47–52.

45. *Петров А.А.* Структура программного комплекса для моделирования технических систем в условиях переключения режимов работы // Электромагнитные волны и электронные системы. 2018. Т. 23. № 4. С. 61–64.

46. *Петров А.А., Дружинина О.В., Масина О.Н.* Моделирование систем оценивания знаний в рамках гибридной интеллектуальной обучающей среды // Современные информационные технологии и ИТ-образование. 2021. Т. 17. № 1. С. 1–14.

47. *Плас Дж. В.* Python для сложных задач. Наука о данных и машинное обучение. СПб.: Питер, 2018.

48. *Поляк Б.Т., Хлебников М.В., Рапопорт Л.Б.* Математическая теория автоматического управления: Учебное пособие. М.: ЛЕНАНД / Изд. группа URSS, 2019.

49. *Сахаров М.К., Карпенко А.П.* Меметические алгоритмы для решения задач глобальной нелинейной оптимизации. Обзор // Наука и Образование. М.: МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. Вып. 12. С.119–142.

50. *Хайкин С.* Нейронные сети. М.: Издательский дом «Вильямс», 2006.

51. *Шестаков А.А.* Обобщенный прямой метод Ляпунова для систем с распределенными параметрами. 1-е изд. М.: Наука, 1990. 2-е изд., доп. М.: URSS, 2007.

52. *Aliworom C., Uzoechi L., Olubiwe M.* Design of fuzzy logic tracking controller for industrial conveyor system // International Journal of Engineering Trends and Technology. 2018. V. 61. P. 64–71.

53. *Andrejiova M., Grincova A., Marasova D.* Monitoring dynamic loading of conveyer belts by measuring local peak impact forces // Measurement. 2020. V. 158, P. 107690.

54. *Andrejiova M., Grincova A., Marasova D.* Measurement and simulation of impact wear damage to industrial conveyor belts // Wear. 2016. V. 12. P. 368–369.

55. *Bezanson J., Edelman A., Karpinski S., Shah V. B.* Julia: A fresh approach to numerical computing // SIAM Review. 2017. V. 59. No. 1: 65–98.

56. *Das A., Kempe D.* Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection // Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, 2011.

57. *Druzhinina O.V., Masina O.N., Petrov A.A.* Modeling of the belt conveyor control system using artificial intelligence methods // Journal of Physics: Conference Series. 2021. V. 2001. P. 012011.

58. *Druzhinina O.V., Masina O.N., Petrov A.A.* The synthesis of the switching systems optimal parameters search algorithms // Communications in Computer and Information Science (CCIS). Springer, 2019. V. 974. P. 306–320.

59. *Druzhinina O.V., Masina O.N., Petrov A.A., Lisovsky E.V., Lyudagovskaya M.A.* Neural network optimization algorithms for controlled switching systems // Advances in Intelligent Systems and Computing (AISC). 2020. V. 1225. P. 470–483.

60. *Farouq O., Selamat H., Noor S.* Intelligent modeling and control of a conveyor belt grain dryer using a simplified type 2 neuro-fuzzy controller // Drying Technology. 2015. V. 33. No. 10. P. 1210–1222.

61. *Fuhrer C, Solem J.E., Verdier O.* Scientific computing with Python 3. Packt Publishing, 2016.

62. *Halvorsen H.P.* Python for science and engineering. 2019. Электронный ресурс: https://www.halvorsen.blog/ documents/programming/python/resources/Python for Science and Engineering.pdf (дата обращения: 28.04.2022).

63. *Hansen J.S.* GNU Octave. Beginner's guide. Packt Publishing, 2011.

64. Julia micro-benchmarks. Электронный ресурс: https://julialang.org/benchmarks/ (дата обращения: 28.04.2022).

65. *Khalid H.* Implementation of artificial neural network to achieve speed control and power saving of a belt conveyor system // Eastern-European Journal of Enterprise Technologies. 2021. V. 2. P. 44-53.

66. *Kim H.* Intelligent control of vehicle dynamic systems by artificial neural network. Ph. D. Thesis, North Carolina State University.1995.

67. *Kumar R., Singh, V.P., Mathur A.* Intelligent algorithms for analysis and control of dynamical systems. Singapore: Springer, 2021.

68. *Lampinen J.A.* Constraint handling approach for the differential evolution algorithm // Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600). V. 2. IEEE, 2002.

69. *Lamy R.* Instant SymPy starter. Packt Publishing, 2013.

70. *Lee D., Seo H., Jung M.W.* Neural basis of reinforcement learning and decision making // Annual Review of neuroscience. 2012. V. 35. No. 1. P. 287–308

71. *Liberzon D., Morse A.S.* Basic problems in stability and design of switched systems // IEEE Control systems magazine. 1999. V. 19. № 5. P. 59–70.

72. *Listova M.A., Dmitrieva V.V., Sizin P.E.* Reliability of the belt conveyor bed when restoring failed roller supports // IOP Conference Series: Earth and Environmental Science. 21, Interdisciplinary Topics in Mining and Geology. 2021. P. 012002.

73. *Lutfy O.F., Mohd Noor S.B., Marhaban M.H.* Design of an intelligent control system for conveyor-belt grain dryers: an application of soft computing techniques in grain drying systems. LAP LAMBERT Academic Publishing, 2012.

74. *Lv Y., Liu B., Liu N., Zhao M.* Design of automatic speed control system of belt conveyor based on image recognition // IEEE 2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD) - Chengdu, China. 2020.

75. *Ma X.M., Gao X.X.* Coal belt conveyor PID controller parameter regulation with neural network // Applied Mechanics and Materials. 2013. V. 319. P. 583–589.

76. *Masina O.N., Druzhinina O.V., Igonina E.V., Petrov A.A.* Synthesis and stabilization of belt conveyor models with intelligent control // Lecture Notes in Networks and Systems. 2021. V. 228. P. 645–658.

77. *Masina O.N., Druzhinina O.V., Petrov A.A.* Computer Research and Stabilization of Dynamic Models of Conveyor Systems // Communications in Computer and Information Science (CCIS). Springer, 2023. V. 1821. P. 165–176. A. Gibadullin (Ed.): ITIDMS 2022. (Proceedings of the Second International Scientific and Practical Conference «Information Technologies and Intelligent Decision Making Systems» (ITIDMS 2022), Moscow, Russia, December 12–14, 2022).

78. *Masina O.N., Druzhinina O.V., Petrov A.A.* Controllers synthesis for computer research of dynamic conveyor belt model using intelligent algorithms // Lecture Notes in Networks and Systems (LNNS). 2022. V. 502. P. 462–473.

79. *Mckinney W.* Python for data analysis, 2e: data wrangling with Pandas, Numpy, and Ipython. Boston: O′Reilly, 2017.

80. *Meurer A.* et al. SymPy: symbolic computing in Python // PeerJ Computer Science. 2017. P. 1–27.

81. *Nakamura S. GNU* Octave primer for begginers: EZ guide to the commands and graphics. Paperback, 2015.

82. *Nguyen H., Petrova G.* Greedy strategies for convex optimization // Calcolo. 2017. V. 54. P. 207–224.

83. *Oliphant T.E.* Guide to NumPy. 2nd edition. Create Space Independent Publishing Platform, USA, 2015.

84. *Oliphant T.E.* Python for scientific computing // Computing in Science and Engineering. 2007. V. 9. No. 3. P. 10–20.

85. *Ozgur C., Colliau T., Rogers G.* MatLab vs. Python vs. R // Journal of Data Science. 2017. V. 15. № 3. P. 355–372.

86. *Petrov A.A., Druzhinina O.V., Masina O.N.* Neural network control of a belt conveyor model with a dynamic angle of elevation // Lecture Notes in Networks and Systems (LNNS). Springer, 2023. V. 724. P. 1–14.

87. *Petrov A.A., Druzhinina O.V., Masina O.N.* Application of the computational intelligence method to modeling the dynamics of multidimensional population system // Lecture Notes in Networks and Systems (LNNS). Springer, 2023. V. 597. P. 565–575.

88. *Pinter J.D.* Global optimization: software, test problems, and applications // In: Handbook of Global Optimization. Boston: Kluver, 2002. P. 515–569.

89. *Plotnikova N.P., Fedosin S.A., Teslya V.V.* Gravitation search training algorithm for asynchronous distributed multilayer perceptron model // Lecture Notes in Electrical Engineering. 2015. V. 312. P. 417–423.

90. *Quarteroni A., Saleri F., Gervasio P.* Scientific computing with MATLAB and Octave (texts in computational science and engineering). Springer, 2014.

91. SciPy 1.6.3 documentation. Электронный ресурс: https://docs.scipy.org/doc/scipy/reference/ (дата обращения: 28.04.2022).

92. *Sidorov S., Mironov S., Pleshakov M.* Dual greedy algorithm for conic optimization problem // CEUR Workshop Proceedings. 2016. V. 1623. P. 276–283.

93. *Storn R, Price K.* Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces // Journal of Global Optimization. 1997. V. 11. P. 341–359.

94. *Subba Rao D.V.* The belt conveyor: a concise basic course. London, New York: CRC Press, 2020.

95. *Vasilyev S.N., Novikov D.A., Bakhtadze N.N.* Intelligent control of industrial processes // Preprints of the 2013 IFAC Conference on Manufacturing Modelling, Management, and Control, Saint Petersburg State

University and Saint Petersburg National Research University of Information Technologies, Mechanics, and Optics, Saint Petersburg, Russia, June 19–21, 2013. P. 49–57.

96. *Wen Yu*. Recent advances in intelligent control systems. Springer, 2009.

97. *Zaitceva I., Andrievsky B.* Methods of intelligent control in mechatronics and robotic engineering: a survey // Electronics. 2022. V. 11. No. 15. P. 24–43.

98. *Zhao L., Lyn Y.* Typical failure analysis and processing of belt conveyor // Procedia Engineering. 2011. V. 26. P. 942–946.

99. *Žvirblis T., Petkevicius L., Bzinkowski D., Vaitkus D., Vaitkus P., Rucki M., Kilikevičius A.* Investigation of deep learning models on identification of minimum signal length for precise classification of conveyor rubber belt loads // Advances in Mechanical Engineering. 2022. V. 14. P. 1–13.

Учебное издание

Масина Ольга Николаевна,
Петров Алексей Алексеевич,
Дружинина Ольга Валентиновна

ИСПОЛЬЗОВАНИЕ ВЫСОКОУРОВНЕВЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ МОДЕЛИРОВАНИЯ

Учебное пособие

Техническое исполнение – В.М. Гришин
Книга печатается в авторской редакции

Формат 60 х 84 1/16. Гарнитура Times. Печать трафаретная.
Печ.л. 6,3. Уч.-изд.л. 6,2.
Тираж 500. Заказ 71

Отпечатано с готового оригинал-макета на участке оперативной полиграфии
ФГБОУ ВО «Елецкий государственный университет им. И.А. Бунина»

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Елецкий государственный университет им. И.А. Бунина»
399770, г. Елец, ул. Коммунаров, 28,1