Министерство Общего и Профессионального Образования
Российской Федерации
Ростовский Государственный Университет
Механико-математический Факультет
кафедра Информатики и Вычислительного Эксперимента

Квалификационная работа
бакалавра
на тему:

Обучающий комплекс по языку программирования Питон

студент 4 курса, 2 группы, Зайцев В. В.
Научный руководитель: к.т.н., доцент Литвиненко А. Н.

Ростов-на-Дону
2002

Введение

Сказать, что хороший программист может написать хорошее программное обеспечение на любом языке, — это всё равно, что сказать, что хороший пилот может управлять любым самолётом: верно, но не по существу [3].

В настоящее время такая точка зрения на выбор языка программирования, к сожалению, превалирует. Роль языка в программировании принижается по сравнению с программной методологией и инструментальными средствами [3]. Для обучения информатике выбирают один из накатанных путей, используя либо старые языки, либо выдуманные, либо чрезвычайно сложные. Нашей же целью было выбрать из множества языков (которых в настоящее время больше четырех тысяч [37]) тот, что подходил бы для обучения, был бы при этом прост и позволял заниматься реальным программированием, не слишком оторванным от жизни. Проведенный обзор языков программирования с аргументацией в пользу питона будет приведён в первой главе.

Следующим шагом для нас было практическое освоение языка питон и применение его в области математического программирования. Это позволило подтвердить существование его достоинств [31, 32, 33] и определить возможность его применения в математике [18].

Сформулированная практическая задача состояла в создании обучающего комплекса по этому языку программирования. Структура комплекса достаточно сложна и разветвлена, содержит материал для чтения лекций, презентации для их проведения, упражнения для практических занятий, сборник примеров готовых программ, тестирование и пр. Во второй главе структура разработанного сайта `http://pythonc.by.ru`, объединившего комплекс, рассмотрена подробно.

Ясно, что первой неотъемлемой частью любого обучающего комплекса должен служить собственно обучающий материал. В нашем случае он представляет собой текст шестнадцати лекций с презентациями для них. Структура курса и прочие близкие вопросы описаны в третьей главе.

Четвертая глава объясняет функционирование тестирующей программы, выполненной на языке JavaScript для возможности проведения тестирования в режиме on-line. Таким образом, тестирование гармонично вписывается в Интернет-сайт обучающего комплекса и позволяет использовать его не только при классическом, но и в дистанционном обучении.

Пятая глава предлагает некоторые темы для лабораторных работ и

задачи для практических занятий. Эти занятия являются неотъемлемой частью любого курса [35]. Мнение автора (с которым без труда соглашаются многие другие студенты) состоит в том, что знания по теории какого угодно предмета гораздо хуже усваиваются, если по этому предмету не было практических занятий. Темы сформулированы достаточно обще и открыты для возможных изменений.

Шестая глава состоит из некоторых примеров рабочих приложений, написанных в хорошем стиле на языке питон. Известно, что языки программирования существуют только для преодоления разрыва в уровне абстракции между аппаратными средствами и реальным миром [3]. Средства высоких уровней позволяют легко формулировать задачи и безопасно использовать широкий спектр программных средств, тогда как средства низких уровней более гибки и зачастую допускают более эффективную реализацию [5, 26]. Выбор средств подходящего уровня абстракции очень сложен [5, 16] и лучше всего ему учиться на примерах, предложенных в этом разделе.

В качестве задач для программирования были выбраны различные математические задачи общего характера [1, 6], задачи из теории матриц [7, 8], а также численные методы решения систем уравнений [2, 10, 12, 19]. Многие примеры реализованы с использованием элементов λ-исчисления [23, 30].

Седьмая глава полностью посвящена использованию питона для математических расчетов, она подводит итог исследованию возможностей и перспективам применения этого языка программирования в нашей предметной области [18].

Заключение подводит итоги тому, что было сделано, для чего это предназначено и насколько успешной была работа и насколько благотворным — её направление.

Глава 1

Сравнительный анализ языков программирования

1.1 Первый язык — каким он должен быть

Некоторые могут считать, что выбор первого языка для обучения программированию — не такая уж сложная задача, от исхода решения которой мало что зависит. Ведь основной целью первичного обучения является знакомство с основными конструкциями, а её можно достичь проще. Никто не запрещает объяснять следование, повторение и ветвление на блок-схемах, а иерархию, абстрагирование, инкапсуляцию, наследование и полиморфизм — на UML-схемах. Это всё так. Но чем более оторванный от реальной жизни материал используется, тем сложнее потом «обученному» таким способом новопрограммисту использовать полученные знания.

Очень важно, с какого языка начинается знакомство с программированием. Язык определяет множество доступных конструкций, вложенная в него парадигма детерминирует то, что называется философией программирования. Даже среди парадигм (которых существует не так уж много) необходимо сделать выбор, не говоря уже о том, что каждая парадигма программирования представлена подчас тысячами языков [3, 37].

По мнению авторов, к первому языку программирования предъявляются следующие требования:

- *Легкость обучения и простота использования основных конструкций*

  Это требование есть палка о двух концах. Некоторые языки программирования просты настолько, что вообще не содержат некоторых конструкций или используют их облегченные аналоги (например, это весь язык Лого, или отсутствие механизма обработки исключений и возможности модульной декомпозиции в стандартном Паскале). Простота не должна быть прокрустовым ложем, в поисках удовлетворения

стандартам укорачивающим ноги.

- *Наличие бесплатной реализации транслятора языка*

  Использование чего бы то ни было в образовательном процессе означает дополнительные траты на это. Приятно, если они содержат только издержки по переходу на новый язык и высшему учебному заведению не нужно тратиться на покупку коммерческого транслятора для каждого компьютера в парке.

- *Переносимость языка, доступность трансляторов для разных платформ*

  Если учебное заведение достаточно крупно, в парке компьютеров присутствуют представители очень разных архитектур, платформ м операционных систем. Для того, чтобы не ограничивать обучаемых только одним типом ЭВМ, язык должен предоставлять трансляторы под все используемые платформы.

- *Перспективность языка, возможность его практического использования после окончания обучения*

  Конечно, было бы чрезвычайно неприятно, если по окончании обучения студент слышал: «Ну что, обучение завершено. Можно начинать учить новый язык программирования». Несмотря на объемность многих начальных курсов по программированию, далеко не все из них позволяют решать какие-либо реальные задачи по окончании обучения.

Невыполнение любого из этих требований влечет за собой различные неприятности для обучаемого: либо сложность самого процесса обучения, либо в некотором смысле его бесполезность — невозможность или сильно ограниченная возможность применить полученные знания в проектировании собственных приложений. (Специалисты по языкам программирования знают, что задача указывает на язык, которым ее следует решать. Но верно и обратное: язык, которым владеет программист, определяет тот диапазон задач, которые он может решить).

1.2 Исхоженные тропы

В настоящее время для обучения информатике обычно выбирают один из четырех путей.

- **Искусственные образовательные языки**.

  Некоторые используют языки программирования, специально созданные в образовательных целях, а потому не имеющие реального применения. Из всем известных примеров можно привести язык Лого (автор — Сеймур Пайперт [14]), который сделал теории программирования

такой подарок как «черепашку». Другим широко известным примером является язык программирования ABC (автор — Стивен Пемберton[1] [27]).

Такие языки обычно характеризуются неестественной простотой, большой свободой приёмов, разрешаемых программисту, а также малыми возможностями при переходе от типовых упражнений к собственным приложениям.

- **Старые языки программирования**.

  Другие всё ещё строят вводные курсы на языках, не поддерживающих нововведения в области программирования (таким образом, заранее сужая область обучения). Теория программирования не стоит на месте, с шестидесятых годов прошлого тысячелетия было изобретено множество подходов, приёмов, аппаратов, конструкций, технологий, механизмов, а также немного изменился стиль программирования.

  Подобные языки не подходят сразу по нескольким параметрам. Они не содержат всех основных конструкций и они не применяются в действительности (пожалуй, единственным исключением является язык Кобол, не сильно изменившийся за последние полвека, но мало кто настолько жесток, чтобы начинать обучение с Кобола. Таким образом, это исключение лишь подтверждает правило). Одинокое достоинство этого пути разом объясняет его применение: для этих языков уже существуют готовые обучающие комплексы, составлены сборники упражнений и выверен стиль.

- **Громоздкие современные языки**.

  Немногие решаются использовать языки, изобретенные последние десять лет, в основном из-за их сложности и большого объема документации. Ява (автор — Джеймс Гослинг [26]), Перл (автор — Ларри Волл [35]), Ада95 (разработанный комитетом по модернизации, входящем в состав Департамента Защиты США) — список последних языковых изобретений так и пестрит гигантами.

1.3 Наш выбор

Каждый из этих подходов имеет свои недостатки, что заставляет желать лучшего пути. Автор предлагает использовать язык питон, который достаточно нов для содержания всех нужных для обучения конструкций, и при этом проще своих *конкурентов* и широко применяется на практике.

[1] Автор XHTML, глава комитета по работе над HTML.

Глава 2

Функционирование комплекса

Центральной и самой главной частью работы является Интернет-сайт, вобравший в себя как все материалы, созданные непосредственно автором, так и ссылки на другие ресурсы по данной тематике.

Сайт располагается по адресу http://pythonc.by.ru и содержит множество подразделов. Среди них:

- Текст лекций, написанных автором, предлагаемый для скачивания во множестве различных распространенных форматов.
- Презентации для каждой из них, которые автор рекомендует использовать при чтении лекций как альтернативу традиционным доске и мелу.
- Набор упражнений для проведения практических или лабораторных занятий, с возможностью создания многих вариантов заданий.
- Сборник примеров готовых программ, на основе которых можно научиться правильному стилю программирования на языке питон.
- On-line тестирование — для возможности использования данного обучающего комплекса не только в классическом, но и в дистанционном обучении.
- Навигационные механизмы, помогающие пользователю ориентироваться внутри сайта и быстро находить нужные ему материалы.

Сайт выполнен с использованием современных технологий и удовлетворяет последним требованиям, предъявляемым к Интернет-страницам: валидности и юзабилити. Первое означает, что все используемые технологии были употреблены каждая на своём месте и в соответствии со стандартами. Производители программного обеспечения для работы в Интернете

(скажем, браузеров) со своей стороны также гарантируют верное отображение содержимого, подготовленного согласно стандартам. Таких технологий было использовано две: **HTML 4.0** [28] и **CSS 2** [22].

Вышеприведенные картинки (которые можно встретить на каждой странице сайта) указывают на то, что данная страница удовлетворяет стандарту **HTML 4.0**, а связанная с ней таблица стилей удовлетворяет стандарту **CSS 2**. Недоверчивый пользователь всегда имеет возможность перейти по ссылке на этой картинке и увидеть результаты синтаксического разбора (парсинга) этого документа по данному стандарту валидатором Интернет-Консорциума (W3C).

Требования к юзабилити распадаются на важность содержимого (контента) и возможность доступа (accessibility). Первое мы считаем выполненным сразу и просто стараемся предложить зашедшему на наш сайт как можно больший выбор. Второе также большей частью контролируются Интернет-Консорциумом, существует так называемая Инициатива Доступности Сети (Web Accessibility Initiative, WAI) [36], описывающая три степени доступности страницы. Каждая страница нашего сайта содержит такую картинку:

Это значит, что данная страница удовлетворяет всем требованиям доступности Интернет-содержимого (Web Content Accessibility Guidelines, WCAG) [34], имеющим приоритет 1 (WCAG Level A Conformance). С тем, что это означает, можно ознакомиться непосредственно на сайте Интернет-Консорциума [24, 25, 29]. Отметим лишь, что некоторые из этих требований, хоть и являются почти очевидными для пользователя, представляют собой существенные проблемы для проектировщика.

Прочие части сайта описаны в последующих главах. Третья глава кратко описывает собственно курс лекций и презентации для них. Четвёртая глава объясняет функционирование тестирующей программы. Пятая предлагает некоторые темы для лабораторных работ и задачи для практических занятий. Шестая состоит из некоторых примеров рабочих приложений, написанных в хорошем стиле Питона. Седьмая полностью посвящена использованию Питона для математических расчётов.

Глава 3

Лекции и презентации

3.1 Содержание лекций

Весь материал делится на несколько больших тем, каждая из которых вправе содержать какое-то ненулевое количество малых тем. Повествование идёт традиционным путём: от простого к сложному, от частного к общему. Сначала слушателя знакомят с типами обеспечения компьютеров, потом рассказывают о том, как пишут программы и какие языки для этого используются (1–2 большие темы), затем переходя к основным простейшим конструкциям программирования. Когда все простейшие конструкции объяснены (3–4 большие темы), закончена первая большая часть курса. Вторая большая часть (5–6 больших тем) раскрывает уже почти опытному пользователю секреты объектно-ориентированного программирования, проектирования и анализа [5, 16, 26], полностью раскладывая по составляющим объектную модель, поясняя каждую её часть примерами конкретных приёмов в Питоне. Большая тема номер 7 подводит итог курсу.

3.2 Представление лекций

Исходный текст данного курса лекций был написан и сформатирован на языке TEXHTML, изобретенном автором для собственного удобства. Написать две программы на питоне в тот момент было сделать легче и быстрее, чем устанавливать, например, популярную программу latex2html.

Язык разметки TEXHTML является вторичным языком разметки, изобретенным автором для удобства работы с такими распространенными форматами, как TEX(см. [9, 13, и др.]) и HTML [28]. Два питоновских скрипта легко переводят TML-файл в любой из нужных форматов (TEX, HTML). После этого требуется лишь поверхностное вмешательство и корректировка разметки гипертекста (вставка ссылок, более красивое форматирование) и обработка процессором TEXа для получения форматов DVI и PostScript.

Существующие трансляторы из TEXa в `HTML` на момент написания собственных трансляторов так или иначе не устраивали авторов.

TML файл начинается со строки ## TeXHTML1, после которой файл опознается анализатором и следующие команды также опознаются и заменяются (каждая команда начинается с диеза и принимает нуль или один параметр в квадратных скобках):

1. `#NL` — переход на новую строку, начало абзаца
2. `#Ch` — заголовок главы (лекции)
3. `#Se` — заголовок секции (большой темы)
4. `#SS` — заголовок подсекции (малой темы)
5. `#Ftt` — форматирование моноширинным шрифтом
6. `#Fit` — форматирование курсивом
7. `#Fbf` — форматирование полужирным шрифтом
8. `#Ful` — форматирование подчеркнутым шрифтом
9. `#Fqt` — взятие в кавычки
10. `#Fgr` — надпись греческими буквами
11. `#Fra` — выдача параметра без парсинга (разбора)
12. `#BFit` и `#EFit` — начало и конец курсива
13. `#BFtt` и `#EFtt` — начало и конец моноширинного шрифта
14. `#_` — символ подчеркивания
15. `#fn` — сноска
16. `#Tmt` — основной заголовок документа
17. `#Tst` — подзаголовок документа
18. `#-` — длинное тире
19. `#Ite` — вставка текста
20. `#Iim` — вставка изображения
21. `#BLnu` и `#ELnu` — начало и конец нумерованного списка
22. `#BLit` и `#ELit` — начало и конец маркированного списка
23. `#Lit` — элемент списка

24. `#LOHTML`, `#LOXML`, `#LOTeX`, `#LOCWI` — написанные фирменным шрифтом названий соответствующих программ, языков и учреждений
25. `#.` — многоточие
26. `#\` — обратная косая черта
27. `#%` — знак процента
28. `#&` — знак амперсанда
29. `##` — знак диеза
30. `#^` — знак крышки
31. `#~` — знак отрицания
32. `#LB` и `#RB` — правая и левая квадратные скобки
33. `#{` и `#}` — правая и левая фигурные скобки

Данный язык разметки не претендует на звание самого лучшего языка разметки (авторы ни в коем случае не хочет составлять конкуренцию Стивену Пембертону), но он очень помог автору при формировании всех версий лекционного курса.

Ремарка по поводу времени выполнения: широко распространено мнение, что программы на питоне медленнее своих собратьев. Это действительно так почти всегда. Но наш питоновский скрипт преобразует (для примера брались первые восемь лекций) TML в TEX за 5 секунд, сам TEX создает DVI за 3 секунды, PS ещё за 4 секунды, и наконец Acrobat Distiller делает из PS'а PDF за 17 секунд. Генерирование `HTML`-кода требует также 5 секунд работы. Таким образом, наш (возможно, не до совершенства прооптимизированный) питоновский скрипт по скорости вполне сравним с процессором самого TEXа и многократно быстрее, чем Acrobat Distiller.

3.3 Структура курса

1. Введение
   1.1. ЭВМ и её обеспечение
   1.2. Трансляторы языков программирования
   1.3. Типы языков программирования и их эволюция
2. Введение в язык питон
   2.1. Краткая история языка
   2.2. Работа с интерпретатором питона
3. Типы данных и простейшие конструкции питона

3.4 Подразделение курса на лекции

Лекция первая: Введение, компьютер и типы его обеспечения, программное обеспечение для программистов, трансляторы языков программирования, достоинства интерпретации.

Лекция вторая: Классификация и эволюция языков программирования, формальное определение языка программирования, краткая история языка питон, его распространение и использование последнее время, работа с интерпретатором питона.

Лекция третья: Простейшая программа на питоне, понятие переменной и присваивание ей значения, вывод данных, ввод данных.

Лекция четвертая: Типы данных в нетипизированном языке, целые числа, операции над ними, вещественные числа, комплексные числа, связь между числовыми типами, строковый тип, варианты заключения в кавычки.

Лекция пятая: Композитные типы данных, последовательности, кортежи, списки, их сравнение, сечения последовательностей, словари.

Лекция шестая: Булев тип, логические условия, оператор ветвления, множественное ветвление, комментарии, понятие циклов, оператор перебора, цикл с предусловием, аварийный выход из цикла.

Лекция седьмая: Понятие подпрограммы, отличие процедур от функций, определение подпрограмм в питоне, возвращение значения, область видимости имён переменных, приёмы профессиональной работы с функциями: именованные параметры, необязательные для указания параметры, параметры неизвестной длины, непредусмотренные параметры.

Лекция восьмая: Лямбда-исчисление, тезис Чёрча, элементы функционального программирования в питоне, непрямой вызов функций, отображение списков, фильтрация списков, организация цепочечных вычислений, пример использования всех приёмов в совокупности, философское понимание подпрограмм как средства поднятия уровня абстракции.

Лекция девятая: Объектная модель как новая технология создания программного обеспечения, понятия объекта и класса, объектно-ориентированное программирование, объектно-ориентированное проектирование, программирование в большом, объектно-ориентированный анализ, объекты и классы в питоне, задание простейшего класса и его экземпляра, конструкторы и деструкторы.

Лекция десятая: Составные части объектного подхода, абстрагирование, принцип наименьшего удивления, выбор абстракций, типы абстракций, контрактная модель взаимодействия объектов, исключительные ситуации, их обработка в питоне, стандартные исключения питона.

Лекция одиннадцатая: Инкапсуляция как дополнение абстракции, возможности инкапсуляции в питоне, свободные, личные и скрытые свойства и методы объектов, служебные методы, создание новых типов данных (в широком смысле) с их помощью

Лекция двенадцатая: Модуль как единица физической декомпозиции системы, разложение системы на модули, крайние варианты примене-

ния модульности, обилие стандартных модулей питона, их использование. Модулирование пространств имён.

Лекция тринадцатая: Математические модули Numeric и SciPy. Средства, предоставляемые математическими модулями питона.

Лекция четырнадцатая: Иерархия инкапсулированных абстракций как последнее средство упрощения системы для понимания программистом. Наследование: одиночное и множественное. Типизация, определение типа, сильная и слабая типизации. Связанные понятия: приведение типов, `__str__/__repr__`, восхождение и нисхождение, полиморфизм, однородные пространства, абстрактные типы.

Лекция пятнадцатая: Параллелизм и сохраняемость как второстепенные "киты"объектной модели, обзор элементов объектной модели опытным взором, преимущества объектной модели, широкое применение, программирование в большом. Плюсы и минусы объектно-ориентированного подхода.

Лекция шестнадцатая: Нестандартные типы данных, незнакомые парадигмы программирования. Перспективы развития программирования.

Глава 4

Тестирование

4.1 Тестирование on-line

Тестирующая программа была реализована на так называемом Dynamic HTML, то есть на HTML [28] с использованием сценарного языка JavaScript. Она доступна с сайта по гиперссылке.

Работа с тестирующей программой позволяет в диалоге с человеком выявить знания последнего, усвоенные из курса. Безусловно, этот тест не является и не может являться заменой обычному экзамену, но примерно определяет уровень знаний тестируемого. Кроме оценки (от 2 до 5 с возможной дробной частью) программа выносит некоторое суждение из неё и доводит его до пользователя. Суждение может быть как положительным (похвала за идеальные знания и уверенность в том, что тестируемый уже давно и успешно занимается питоном), так и отрицательным (предположение о том, что тестируемый ещё и не начинал заниматься по предложенному курсу или же начал, но не освоил даже простейших вещей).

Тестирующая программа не является ни неотъемлемой, ни самой важной частью обучающего комплекса, будучи написана лишь для довершения идеи использования комплекса в дистанционном обучении. Поэтому она не содержит ни ограничения по времени ответов на задаваемые вопросы, ни сколь-нибудь большую базу вопросов со случайной выборкой предоставляемых тестируемому, ни перемешивания ответов на вопросы во избежание механического запоминания местоположений ответов, ни криптографической защиты передаваемой информации. Собственно, вся информация передаётся только один раз, при загрузке страницы с тестирующей программой (ссылка Тестирование из панели навигации справа от питоновского логотипа). После передачи информации на компьютер клиента всё выполнение программы идёт независимо и локально, что позволяет ему вообще отключиться от Интернета и не находиться в он-лайне.

В будущем, возможно, данная программа будет переделана на CGI-основу (CGI — common gateway interface), каждый ответ будет передаваться

на сервер, где после обработки его CGI-сценарием и запоминания будет создаваться новый вопрос. Естественно, для реализации CGI-сценария будет использован язык питон.

4.2 Безопасная интерактивная среда

Для облегчения освоения пользователя с интерактивной средой питона автором была разработана так называемая безопасная интерактивная среда (Safe Interactive Environement) — довольно грубая эмуляция интерпретатора питона на питоне, не предоставляющая, к сожалению, возможности создания больших приложений, но гораздо более мягкая и общительная в плане сообщения пользователю его ошибок. Она просто перехватывает все возможные исключительные ситуации, могущие возникнуть при вычислении выражения по вине пользователя, его вводящего, и вербализует их причину.

Структура программы предельно проста: один класс-исключение безопасной интерактивной среды, реализующий специфические ошибки, и две вложенные конструкции попытки выполнения try/except.

Пример работы с безопасной интерактивной средой:

```
Безопасная интерактивная среда
```

```
Вводите строки питона для выполнения:
>>> два плюс два
Ошибка синтаксиса
>>> two + two
Неверное имя переменной 'two'
>>> 2 + 2
4
Вычисление проведено успешно
>>>
(c) 2002
```

Для достижения других целей можно ещё чуть улучшить безопасную интерактивную среду, введя в программу перехват и расшифровку параметров исключительных ситуаций. Как это сделать, показано на примере исключений `IOError` и `NameError`.

Исходный текст программы:

```
from string import split

class SIEError(RuntimeError):
 """
  Safe Interactive Environement
  Безопасная интерактивная среда
  Класс внутренних исключений
```

```
 """
 def __init__(self,detail):
  if detail=='':
   raise SyntaxError
  elif detail in ('s','x','go','SIEError'):
   print 'При работе с безопасной интерактивной средой невозможно обращение
          к её служебным переменным'

go=s='В процессе вычисления произошла '

print 'Безопасная интерактивная среда [redacted]
print ' [redacted]\n'

print 'Вводите строки питона для выполнения:'
while go:
 try:
  x=raw_input('>>> ')
  if x in ('','s','x','go','SIEError'):
   raise SIEError,x
  try:
   print eval(x) or 'Вычисление закончено'
  except ArithmeticError:
   print s+'арифметическая ошибка'
  except AttributeError:
   print s+'ошибка доступа к несуществующему свойству или методу'
  except EOFError:
   print s+'ошибка конца файла'
  except EnvironmentError:
   print s+'ошибка интерактивной среды'
  except FloatingPointError:
   print s+'ошибка при вычислениях с плавающей точкой'
  except ImportError:
   print 'Неверное указано имя подключаемого модуля'
  except IndentationError:
   print 'Нарушены правила отступов'
  except IndexError:
   print s+'попытка чтения несуществующего элемента последовательности'
  except IOError, (errno, strerror):
   print s+'ошибка ввода-вывода(%s): %s' % (errno, strerror)
  except KeyError:
   print s+'попытка чтения несуществующего элемента словаря'
  except MemoryError:
   print s+'ошибка выделения памяти'
  except NameError, x:
   print 'Неверное имя переменной',split(str(x))[1]
  except NotImplementedError:
```

```
   print 'Вызов абстрактного метода невозможен'
  except OSError,  SystemError:
   print 'Ошибка интерпретатора питона'
  except OverflowError:
   print s+'ошибка переполнения'
  except SyntaxError:
   print 'Ошибка синтаксиса'
  except TypeError:
   print 'Невозможно произвести приведение данных типов'
  except UnboundLocalError:
   print s+'попытка сослаться на неинициализированную локальную переменную'
  except UnicodeError:
   print s+'ошибка работы с уникодом'
  except ValueError:
   print s+'ошибка значения'
  except ZeroDivisionError:
   print 'На ноль делить нельзя'
  else:
   print 'Вычисление проведено успешно'
 except SyntaxError:
  go=0
 except SIEError:
  pass
print '(c) 2002'
```

Глава 5

Практика как дополнение теории

5.1 Описание

Практические или лабораторные занятия являются неотъемлемой частью любого курса. Мнение автора (с которым без труда соглашаются многие другие студенты) состоит в том, что знания по теории какого-либо гораздо хуже усваиваются, если по этому предмету не было практических занятий. Поэтому при проектировании собственного курса было бы, по меньшей мере, странно не учесть собственное мнение. Задачи не включают подробных указаний о требованиях, а также списков параметров задач для разных вариантов — это легко реализуемые мелочи, к тому же специфичные для каждого преподавателя, собирание и классификация которых потребовала бы ещё нескольких лет.

5.2 Содержание

Работа с интерпретатором питона.

Копирование дистрибутива языка из интернета на локальный компьютер. Инсталляция. Запуск и выполнение простейших команд в текстовом и оконном вариантах интерпретатора.

Простейшие конструкции питона.

Упражнения на перевод формул из обычной нотации в питоновскую (корни, арифметические операции, скобки, приоритеты операций). Упражнения на преобразование типов. Навыки работы в интерпретаторе, запуск внешних файлов. Во всех упражнениях программы должны вводить все числа, а не присваивать удобные значения в тексте. Упражнения на реализацию простых формул программой на языке питон (корни квадратного уравнения по коэффициентам, точки пересечения графиков функций,

вычисление объема тел). Упражнения на оператор вывода на экран (альтернативный стандартному вывод комплексного числа, требования к количеству знаков выдаваемых вещественных чисел). Для продвинутых студентов, знающих циклы, упражнения вывода на экран последовательностей (например, выдача чисел от 1 до 100 в обратном порядке по 10 в строке).

Математическая библиотека питона и комплексы.

Подключение модуля math. Упражнения на тригонометрические формулы (решение треугольников, нахождение угла между стрелками часов в определенный момент времени, разнообразные планиметрические задачи). Упражнения на использование комплексных чисел (формулы с комплекснозначными функциями, с веществennозначными функциями комплексного аргумента, функции с комплексными коэффициентами).

Строки и циклы.

Строки, варианты заключения в кавычки. Ввод строк в программу с клавиатуры оператором `input()`. Оператор `raw_input()`. Упражнения на выборку символов из строки (выдача на экран символов строки в определенном порядке, выдача только одной группы символов, перевод символов в другой регистр, печать всех различных символов, палиндромы). Ввод из файла с помощью модуля fileinput. Работа с файлами (поиск подстроки в файле, частотный анализ букв). Упражнения на обработку строк перед печатью (перевод числа в римскую запись, печать календаря или таблицы умножения, подчеркивание (печатью минусов в следующей строчке) определенных символов). Регулярные выражения. Упражнения на регулярные выражения (выщепление подстрок, замещение подстрок).

Кортежи и списки.

Ещё одно акцентирование разницы между кортежем и списком. Задачи на использование кортежей на примере работы с многочленами (нахождение значения, дифференцирование, произведение и композиция многочленов, поиск корней, поиск коэффициентов многочленов Чебышева-Эрмита). Задачи на использование списков (нахождение среднеарифметического, сортировка, проверка на отсортированность, построение цепочки чисел Фибоначчи). Задачи на одновременное использование кортежей и списков (по строке, содержащей несколько чисел, напечатать их все в обратном порядке с указанием позиции в строке (подсказка: для перевода строки в число пользуйтесь функцией eval), среди двух десятков точек пространства, координаты которых даны, найти две, расстояние между которыми максимально).

Булев тип. Закрепление знаний о циклах.

Упражнения, дающие ответ да или нет (определение знака числа, високосности года, вхождения числа в список, проверка принадлежности точки геометрической фигуре, проверка матрицы на вырожденность). Более сложные упражнения с проверками (метод дихотомии). Упражнения только на циклы (приближенное нахождение значения функций разложением в ряд Тейлора). Закрепление пройденного (численное интегрирование по формулам прямоугольников и трапеций с использованием правила Рунге практической оценки погрешности, печать таблиц истинности для булевых

функций, построение СКНФ и СДНФ).

Глава 6

Примеры приложений

6.1 Важность

Восприятие языка программирования может быть существенно облегчено и улучшено рассмотрением уже готовых программ, использующих те или иные обкатанные приёмы работы с различными структурами и алгоритмами. «Освоить новый язык легко, — говорят программисты, — куда труднее освоиться с ним». И это верно. Выучить другое написание тех же конструкций и джентльменский набор встроенных подпрограмм во много крат проще, чем уяснить правильный стиль программирования на новом языке и выбор языковых средств для формулирования конкретной задачи. Для того, чтобы общаться на японском, недостаточно знать форму всех иероглифов — нужно знать их значения и все правила грамматики, показывающие возможные способы их комбинирования. Так же непросто и научиться писать действительно эффективные программы [20], и обычно даже у умудренного опытом программиста на это уходит долгое время (в течение которого он широко использует этот язык).

В качестве примеров автор использовал различные математические задачи из алгебры и вычислительных методов [1, 6, 7, 8].

Питон, в отличие от многих других монопарадигменных языков программирования поддерживает не только процедурное (прозу) и объектно-ориентированное программирование. Он содержит довольно мощный аппарат функционального программирования [17, 30], в том числе программирования в λ-функциях. Безусловно, сказанное выше отнюдь не означает мультипарадигменности питона в строгом смысле этого слова [21].

6.2 λ-исчисление

В первой половине XX века американский математик Алонзо Чёрч предложил использовать для описания частично рекурсивных функций достаточно простой формализм, названный им λ-исчислением (лямбда-

исчислением). Он же сформулировал так называемый **тезис Чёрча** (на котором базируются тезисы Тьюринга и Маркова) о том, что любая функция, вычислимая в интуитивном смысле эквивалентна некоей частично рекурсивной функции [23]. Этот тезис содержит в себе нестрогое определение, поэтому, с одной стороны, не может быть доказан, а с другой, позволяет упростить некоторые теоретические выкладки. Частично рекурсивные функции суть функции, которые могут зависеть от собственного значения при других входных данных и могут быть определены не для всех входных данных.

Впервые λ-выражения появились в языке Лисп в конце 1950-х годов. Позаимствовав термин у Чёрча, его (Лиспа) создатели, безусловно, внесли множество изменений. Позже было создано несколько языков чисто функционального типа, как на базе Лиспа (Scheme, Loops, CLOS, Miranda, Haskell), так и сильно отличающихся от него (FP, ML, Hope, Erlang). Функциональное программирование — это отдельная парадигма программирования, где программист задаёт зависимость функций друг от друга, определяя таким образом их свойства и значения. В языках, наиболее точно соответствующих этой концепции, например, нет переменных. Элементы функционального программирования, конечно, есть и в питоне.

λ-функция в питоне — это функция без имени, о которой известно только количество аргументов и формула для вычисления итогового значения, причем формула должна записываться единым выражением. Вот пример λ-функции, складывающей три числа:

```
lambda x,y,z:x+y+z
```

Проще и не придумать. Понятно, что описывать огромную функцию, вызывающуюся много раз, λ-функцией, по меньше мере неразумно, но в некоторых случаях (которые мы рассмотрим на этой лекции) λ-функции бывают полезны. Во-первых, их можно присваивать:

```
R=lambda x,y:pow(x*x+y*y,0.5)
print R(3,4)
```

На печать будет выдано `5.0`. Та же функцию, объявленная стандартным питоновским способом, будет занимать две строчки и быть длиннее, многословнее и, что более важно, менее очевидной. Когда же мы продемонстрируем применение λ-функций в подходах чисто функционального программирования, выгод будет ещё больше.

Второе применение λ-функций следует из того, что определение обычной функции не может быть сгенерировано программой «на лету», а определение λ-функции — может, и очень просто:

```
genincr=lambda n:lambda x,i=n:x+i
```

Эта функция возвратит функцию-инкрементатор, увеличивающую свой аргумент на `n`, где `n` даётся при создании функции.

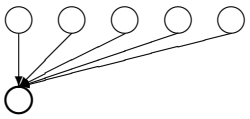
Кроме всех этих очевидных применений λ-функций, существуют ещё четыре стандартных приёма:

1. **Вызов функций — apply()**.

   Но в некоторых случаях бывает удобно выполнять функции не скоб-

ками (`func()`, где `func` — имя функции), а сначала последовательно подготовить все аргументы, и только потом вызвать функцию. Это и осуществляет функция `apply()`.

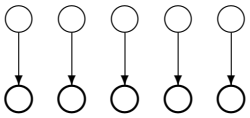
Если вычисление по алгоритму должно происходить по следующей схеме (кружки означают исходные данные, стрелки — их использование, жирный кружок — результат), значит, его нужно реализовывать через `apply()`:



2. **Отображение списков — map()**.

   Функция `map()` порождает новый список из значений функции, примененной к каждому элементу первоначального списка. Использование многопараметрических функций возможно, но в этом случае нужно давать столько списков или кортежей, сколько у неё параметров. Конечно же, они должны быть одинаковой длины. Функция `map` является как бы обобщением предыдущей функции, `apply`, последовательно применяя данную функцию к элементам последовательности.

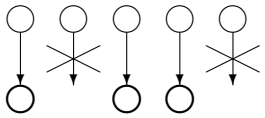
   Если вычисление по алгоритму должно происходить по следующей схеме (обозначения прежние), значит, его нужно реализовывать через `map()`:



3. **Фильтрация списков — filter()**.

   Функция `filter()` генерирует новый список из тех элементов исходного списка, для которых проверочная функция истинна. Сами значения элементов при этом не изменяются.

   Если вычисление по алгоритму должно происходить по следующей схеме, значит, его нужно реализовывать через `filter()`:

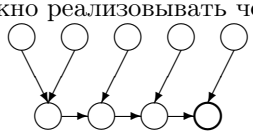


4. **Цепочечные вычисления — reduce()**.

   Функция `reduce()` производит цепочечные вычисления, многократно применяя данную функцию к каждому элементу, подставляя аккумулятор в качестве первого параметра, а сам элемент — в качестве второго. При этом она берёт от двух до трех аргументов: функцию для вычисления и последовательность как обязательные и стартовое

значение аккумулятора как необязательный. Цепочечные вычисления идут слева направо.

Если вычисление по алгоритму должно происходить по следующей схеме, значит, его нужно реализовывать через `reduce()`:



6.3 Арифметика и простейшие алгоритмы.

6.3.1 Наибольший общий делитель — алгоритм Евклида.

Алгоритм Евклида гласит: для нахождения наибольшего общего делителя двух целых положительных чисел нужно подменять большее число его остатком от деления на меньшее до тех пор, пока большее и меньшее числа не будут равны.

Реализуем это через λ-функции.

Главное правило при программировании в λ-функциях питона: не использовать никаких условий. Всё должно быть ясно еще на момент начала выполнения. Поэтому нужно подумать, за сколько шагов (под шагом понимаем взятие остатка от деления) мы гарантированно доберемся до цели.

Математическое ожидание $N \mod M$ равно $\frac{M}{2}$, т.е. на каждом шаге, начиная со второго, оба аргумента уменьшаются приблизительно в два раза. Но это получится нечто вроде оценки сверху математического ожидания числа шагов алгоритма, а не оценка сверху. Оценку сверху можно получить так: худший случай — это когда $N \mod M$ принадлежит $\left(\frac{M}{2}, \frac{2M}{3}\right)$. Почему плохо это и особенно это? Потому что если остаток будет меньше $\frac{M}{2}$, то оценка логарифмом по основанию 2 будет хорошей, а если больше $\frac{2M}{3}$, то эта же оценка станет хорошей на следующем шаге. Получаем, что число шагов ограничено логарифмом по основанию $\frac{3}{2}$ от меньшего аргумента.

Итак, организуем цепочечные вычисления, строя остатки от деления чисел, начиная с двух данных. Кроме того, следует застраховаться от деления на ноль. Последний полученный элемент и будет нашим наибольшим общим делителем.

6.3.2 Наименьшее общее кратное.

Лучший способ для математика решить задачу — это свести её к предыдущей. Применив наши знания в теории, мы легко поймем, что наименьшее общее кратное — это отношение произведения чисел к их наибольшему общему делителю. Задача решена!

6.3.3 Факториал.

Идеальная задача опять-таки для функционального программирования — можно считать, что список целых чисел, не превосходящих данное, у нас уже есть, осталось лишь применить к нему перемножающую функцию.

6.3.4 Поиск простых чисел.

Формулу для k-го простого числа мы выводить не будем, хотя бы потому, что эта задача до сих пор считается нерешенной.

Что есть простое число? Согласно определению, простое число не делится без остатка ни на какое другое число. Понятно, что нет смысла проверять делимость на числа, превышающие исходное. Более того, простой делитель числа не может превышать его корня. Напишем сначала функцию, составляющую список всех остатков от деления данного числа на другие:

```
Z=lambda n:map(lambda a,b=n:b%a,range(2,pow(n,0.5)+1))
```

Если в этом списке есть хотя бы один ноль, это означает, что число n делится без остатка на какое-то другое число, то есть, что n не простое. Построим функцию, возвращающую 1, если в данном ей списке нет нулей и 0 в противном случае:

```
Y=lambda l:reduce(lambda c,d:c*d!=0,l,1)
```

Теперь $Y(Z(n))$ возвращает 1, если n — простое и 0 в противном случае. Половина дела уже сделана. Осталось реализовать перебор всех чисел из какого-то промежутка, то есть построить отображение (map) списка последовательных чисел в список нулей и единиц. В таком случае единицы будут стоять на местах, обозначающих номер простого числа. Будет лучше, если вместо единиц мы будем выдавать само число, тогда, удалив все нули и списка, мы получим список простых чисел без лишних усилий.

```
X=lambda m:map(lambda e:e*Y(Z(e)),range(2,m))
```

Ну и не забываем удалить нули - это для нас проще простого:

```
W=lambda k:filter(None,X(k))
```

Теперь подставим Z в Y, Y в X, а X в W (не забывая, что разные переменные из разных областей действия имён могут иметь одинаковые имена), а затем ужаснемся результату наших усилий.

6.3.5 Разложение целого числа на множители.

Аналитическое решение этой задачи также достойно будущих академиков. Мы же пока что решим её перебором — медленно, но верно. Воспользуемся функцией проверки числа на делимость другим (она была написана нами в предыдущем примере). Только не стоит забывать, что множители могут (и должны) превосходить корень исходного числа. Чтобы не менять концептуальные части, будем добавлять вместе с каждым делителем результат деления исходного числа на него. Это обнаружит маленькую проблему: если число имеет целый корень, он будет включен в список дважды. Не ограничивая себя функциональным программированием (мы же на питоне пишем,

а не на лиспе!), напишем обрамляющую подпрограмму, удаляющую повторное вхождение корня и сортирующую список по возрастанию. Готово! Как мы и задумывали в самом начале, медленно.

6.3.6 Код модуля на питоне

```
#!/usr/bin/env python
"""
 Секция 1. Работа с числами.
 В данный модуль входят следующие функции:

  primes(x) - поиск простых чисел, не превосходящих данное
  nod(a,b) - наибольший общий делитель двух чисел
  nok(a,b) - наименьшее общее кратное двух чисел
  factorial(n) - факториал целого положительного числа
  factors(N) - разложение числа на множители

"""

# Пошли лямбда-функции

# 1
primes=lambda x:filter(None,map(lambda x:x*reduce(
       lambda x,y:x*y!=0,map(lambda y,x=x:x%y,
                  range(2,1+pow(x,0.5))),1),range(2,x)))
nod=lambda x,y:reduce(lambda a,N:a+[(a[N]-1)\%a[N+1]+1],
               range(0,log(max(x,y))/log(1.5)),[x,y]).pop()
nok=lambda A,B:A*B/nod(A,B)
factorial=lambda n:reduce(lambda x,y:x*y,range(2,n+1),1L)
facts=lambda N:reduce(lambda a,n,N=N:a+[n,N/n],
      filter(None,map(lambda k,N=N:(not(N\%k))*k,
      range(1,pow(N,0.5)+1))),[])

# Лямбда-функции кончились, начались обычные (обрамляющие)

# 1
def factors(N):
 a=facts(N)
 if a[-1]!=pow(N,0.5): a+=[pow(N,0.5)]
 a.pop()
 a.sort()
 return a

if __name__ == "__main__":
 print primes(50)
```

6.4 Работа с векторами и матрицами.

6.4.1 Работа с векторами

Операции с векторами

Поэлементное сложение и вычитание векторов также легко реализовать через λ-функции — следует всего лишь применить операцию сложения (`lambda a,b:a+b`) или умножения к каждой паре элементов векторов (`map`). Умножение вектора на число и деление вектора на число реализуются *половиной* λ-функции, то есть λ-функцией, имеющей один аргумент фиксированным. Скалярное же произведение, наоборот, требует двойной λ-функции. Вспомним определение — скалярное произведение есть сумма всех произведений элементов. То есть, один `map` на произведение и один `reduce` на сложение.

Через скалярное произведение вводится и норма вектора (по формуле $||x|| = \sqrt{(x,x)}$).

Процесс ортогонализации Грама-Шмидта

Немодифицированным процессом ортогонализации Грама-Шмидта называют алгоритм построения для данной линейно независимой системы векторов $\{a_i\}_{i=1}^n$ в евклидовом или эрмитовом пространстве E ортогональной системы векторов $\{b_i\}_{i=1}^n$, порождающей то же самое подпространство в E.

$$l(a_1, a_2, \ldots, a_n) = l(b_1, b_2, \ldots, b_n),$$

где $l(\ldots)$ обозначает линейную оболочку, натянутую на вектора.

Алгоритм имеет следующий вид: полагают $b_1 = a_1$, далее построение ведется индуктивно. На каждом этапе строится перпендикуляр к линейной оболочке векторов $a_1, \ldots, a_j$ до конца вектора a_{j+1}. Таким образом, если $b_1, \ldots, b_j$ уже построены, то за b_{j+1} можно взять

$$b_{j+1} = a_{j+1} - \sum_{i=1}^{j} \frac{(a_{j+1}, b_i)}{(b_i, b_i)} b_i$$

Такой сложный алгоритм, тем более так хорошо разбитый на этапы, не стоит портить попыткой реализации функциональным программированием. Вернемся к процедурному. Дальше комментировать практически нечего, так как алгоритм весь описан выше. Единственная ремарка: для защиты нашей подпрограммы от линейной зависимости векторов следует вставить дополнительное условие (иначе будет деление на ноль).

Ортонормирование системы векторов

В этом нет ничего сложного, под ортонормированием понимается процесс ортогонализации, после которого $\forall i \quad ||a_i|| = 1$. Это легко сделать, поделив каждый вектор на его норму.

Нам нетрудно реализовать похожую функцию, только нормирующую систему векторов. Возможно, кому-нибудь это будет полезно.

6.4.2 Работа с матрицами

Простейшие операции с матрицами

Через уже существующие λ-функции легко реализовать поэлементное сложение и вычитание матриц, поэлементное умножение и деление матрицы не число.

Транспонирование матрицы подробно объяснено в лекциях, здесь же мы позволим себе воспользоваться встроенной функцией питона по имени `zip`, которая переделывает список списков в список кортежей соответствующих элементов списков (то есть если матрица представлена списком списков, то эта функция уже почти транспонирует её, остаётся лишь вернуть прежний формат).

Произведение матриц легко реализуется через скалярное произведение и транспонирование. Ведь

$$AB = C$$

$$c_{ij} = (A^{i\cdot}, B^{\cdot j}),$$

где $A^{i\cdot}$ — i-я строка матрицы A, а $B^{\cdot j}$ — j-й столбец матрицы B. Транспонирование используется здесь сугубо для удобства программирования доступа к столбцам матрицы.

Определитель

Подсчет определителя матрицы размера $n \times n$ — также одна из часто используемых операций, стоящая, тем не менее, несколько особняком от умножений и сложений. Мы реализуем подсчет определителя методом Гаусса (описание воспоследствует). При этом в качестве столбца свободных членов мы берем нулевой вектор, затем приводим матрицу к квазидиагональному виду. Собственно определитель равен произведению всех ненулевых элементов (коих остаётся n) на их сигнатуру. Сигнатура считается прямо по определению [7] весьма элегантным по-питоновски способом (см. ниже).

Проверка на свойства матрицы

Известно [7, 8], что любая матрица обладает рядом свойств, таких как вырожденность, ортогональность, симметричность, кососимметричность, эрмитовость, etc. Часть из них легко реализуема средствами питона.

Невырожденными называются матрицы, имеющие ненулевой определитель ($\det A \neq 0$). В этом определении уже содержится программа на питоне.

Ортогональными называются квадратные матрицы, для которых выполняется соотношение:

$$A\,A^{\tau} = E,$$

где E — единичная матрица, размером совпадающая с A. Для определения наличия этого свойства достаточно перемножить матрицу на неё же транспонированную, и проверить на равенство единичной.

Симметрическими матрицами называются те, для которых $a_{ij} = a_{ji}$, а кососимметрическими — для которых $a_{ij} = -a_{ji}$. Реализация идёт аналогичным путём.

Решение СЛАУ методом ортогонализации

Вышеописанный процесс ортогонализации Грама-Шмидта может быть истолкован как разложение невырожденной квадратной матрицы в произведение ортогональной (в эрмитовом случае — унитарной) и верхнетреугольной матрицы с положительными диагональными элементами.

Ортогональной матрицей называется квадратная матрица с действительными элементами, столбцы которой образуют ортонормированную систему векторов, то есть

$$a_{i1}a_{k1} + a_{i2}a_{k2} + \cdots + a_{in}a_{kn} = \delta_i^j,$$

где δ_i^j — символ Кронекера:

$$\delta_i^j = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

Если $A = \{a_{ij}\}_{i,j=1}^{n}$ — квадратная матрица n-го порядка, $x = (x_1, x_2, \ldots, x_n)^{\tau}$, $b = (b_1, b_2, \ldots, b_n)^{\tau}$, $c_i = (a_{i1}, a_{i2}, \ldots, a_{in}, -b_i)$, $i = 1, 2, \ldots, n$, $y = (x_1, x_2, \ldots, x_n, 1)$, то исходная система $Ax = b$ может быть записана в виде $(c_i, y) = 0, \quad i = 1, 2, \ldots, n$. К системе векторов $a_1, a_2, \ldots, a_n, (0, 0, \ldots, 0, 1)$ применяется процесс ортогонализации Грама-Шмидта с нормированием. Из полученной системы ортонормированных векторов $q_1, q_2, \ldots, q_{n+1}$ нам нужен только последний, элементы которого удовлетворяют соотношениям $(c_i, y) = 0, \quad i = 1, 2, \ldots, n$. Если $q_{n+1} = (z_1, z_2, \ldots, z_{n+1})$, то искомое решение системы —

$$\left(\frac{z_1}{z_{n+1}}, \frac{z_2}{z_{n+1}}, \ldots, \frac{z_n}{z_{n+1}}\right)^{\tau}$$

Этот метод равносилен так называемому LU-разложению [19], то есть представлению матрицы в виде произведения $A = LU$, где L — треугольная, а U — унитарная[1].

[1] Это разложение в некоторых источниках именуется неполным разложением Холески (incomplete Choleski).

Решение СЛАУ методом Гаусса

Модифицированный метод Гаусса есть метод последовательного исключения неизвестных для нахождения решений системы линейных алгебраических уравнений. Этот метод тоже прямой, к итерационным [2, 15, 19] мы перейдем позже.

Сначала по системе $Ax = f$ ($A = \{a_{ij}\}_{i,j=1}^{n}$ — известная матрица размера $n \times n$, f — известный вектор свободных членов длины n, а x — вектор неизвестных) или

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

строится так называемая расширенная матрица:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & f_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & f_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & f_n \end{array}\right)$$

Последовательность действий, выполняемых при методе Гаусса, такова (i — номер шага):

1. В i-й строке основной части расширенной матрицы находится так называемый ведущий элемент, не равный нулю. Если такого элемента нет (все элементы строки нулевые), метод Гаусса решения не даёт. (Если при этом соответствующий элемент $f_i \neq 0$, система противоречива и решений не имеет, если $f_i = 0$, система имеет бесконечное множество решений — i-я координата не определена). Далее, не нарушая общности, будем считать, что ведущий элемент существует: a_{ij}.

2. Система *приводится* по этому ведущему элементу таким образом, чтобы в j-м столбце все элементы, кроме него, стали равны нулю. Это делается элементарными преобразованиями: из каждой строки, кроме i-й, вычитается i-я строка, умноженная на $\frac{a_{kj}}{a_{ij}}$, где k — номер данной строки. Аналогично преобразуется и столбец свободных членов.

После выполнения алгоритма для всех строк $i = 1 \ldots n$ либо придём к противоречию, либо получим в основной части расширенной матрицы квазидиагональную матрицу. И в том, и в другом случае результат очевиден: либо нужно сообщить об ошибке, либо сопоставить каждому x_i решение в виде $\frac{f'_i}{a'_{ij}}$, где f' и a' — элементы преобразованной матрицы, а j — номер столбца основной части расширенной матрицы с ненулевым элементом в i-й строке.

Решение СЛАУ по теореме Крамера

Из университетского курса алгебры и геометрии известно, что теорема Крамера о решении системы линейных алгебраических уравнений гласит:

> Система линейных неоднородных алгебраических уравнений определенна тогда и только тогда, когда определитель ее основной матрицы ненулевой. В таком случае ее решения определяются формулами:
>
> $$x_1 = \frac{\Delta_1}{\Delta},\ x_2 = \frac{\Delta_2}{\Delta},\ \dots\ x_n = \frac{\Delta_n}{\Delta},$$
>
> где $\Delta = \det A$ — определитель матрицы системы, а Δ_i — определители матриц, полученных из основной матрицы замещением i-го столбца столбцом свободных членов.

Таким образом, решение системы с вышеперечисленными свойствами на питоне сводится к вычислению определителя и замещению столбца. Последнее легко реализуется путём траспонирования и замещения строки (две строчки кода, отсутствие циклов). Отметим, что обратное транспонирование необязательно ввиду свойств определителя ($\det A^\tau = \det A$).

6.4.3 Код модуля на питоне

```
#!/usr/bin/env python

from time import clock

"""
 В данный модуль входят следующие функции:

 Секция 2. Работа с векторами.
  scalmul(x,y) - скалярное произведение векторов
  add(x,y) - поэлементное сложение векторов
  sub(x,y) - поэлементное вычитание векторов
  mulbyn(x,n) - умножение вектора на число
  divbyn(x,n) - деление вектора на число
  orthog(as) - процесс ортогонализации Грама-Шмидта
  on(as) - ортонормирование системы векторов
  zeroarr(n) - создание нового нулевого вектора длины n
  e(n,i) - создание i-го базисного n-мерного вектора
  norma(x) - норма вектора (корень скалярного квадрата)

 Секция 3. Работа с матрицами.
  addmat(A,B) - поэлементная сумма матриц
  submat(A,B) - поэлементная разность матриц
```

```
  mulmatbyn(A,n) - поэлементное умножение матрицы на число
  divmatbyn(B,n) - поэлементное деление матрицы на число
  equal(A,B) - проверка матриц на равенство с нужной точностью
  iszero(A) - проверка матрицы на равенство нулевой
  simm(A) - проверка матрицы на симметричность
  asimm(A) - проверка матрицы на кососимметричность
  isort(A) - проверка матрицы на ортогональность
  isdeg(A) - проверка матрицы на вырожденность
  trans(X) - транспонирование матрицы
  matrixmul(A,B) - произведение матриц
  GSO_sol(as,b) - решение СЛАУ методом ортогонализации
  Gauss_sol(A,b) - решение СЛАУ методом Гаусса
  Cramer_sol(C,b) - решение СЛАУ по теореме Крамера
  zeros(n,m) - создание нулевой матрицы n на m
  det(A) - определитель
  E(n) - создание единичной матрицы порядка n
  mulmatbyarr(M,A) - произведение матрицы на вектор (удобно)

 Секция 4. Визуализация.
  printv(v,t) - печать массива
  printm(M,t) - печать матрицы

"""

# Пошли лямбда-функции

# 2
scalmul=lambda x,y:reduce(lambda a,b:a+b,
                   map(lambda a,b:a*b,x,y),0.0)
add=lambda x,y:map(lambda a,b:a+b,x,y)
sub=lambda x,y:map(lambda a,b:a-b,x,y)
mulbyn=lambda x,n:map(lambda a,b=n:a*b,x)
divbyn=lambda x,n:map(lambda a,b=n:float(a)/b,x)
on=lambda as:map(lambda ai:divbyn(ai,pow(scalmul(ai,ai),0.5)),orthog(as))
normir=lambda as:map(lambda ai:divbyn(ai,pow(scalmul(ai,ai),0.5)),as)
zeroarr=lambda n:map(lambda a:0,range(0,n))
e=lambda n,i:map(lambda a,i=i:a==i,range(0,n))
norma=lambda x:pow(scalmul(x,x),0.5)

# 3
trans=lambda X:map(list,apply(zip,X))
zeros=lambda n,m:map(lambda a,m=m:zeroarr(m),range(0,n))
U=lambda n:map(lambda a,n=n:a==n,range(0,n+1))
GSO_sol=lambda as,b:apply(lambda q:map(lambda zi,zn_1=q[-1]:zi/zn_1,q),
        [on(orthog(map(lambda ai,bi:ai+[-bi],as,b)+[U(len(b))])).pop()])[:-1]
addmat=lambda x,y:map(add,x,y)
```

```
submat=lambda x,y:map(sub,x,y)
mulmatbyn=lambda x,n:map(lambda a,b=n:mulbyn(a,b),x)
divmatbyn=lambda x,n:map(lambda a,b=n:divbyn(a,b),x)
simm=lambda A:equal(A,trans(A))
asimm=lambda A:iszero(addmat(A,trans(A)))
isort=lambda T:equal(matrixmul(A,trans(A)),E(len(A)))
E=lambda n:map(lambda a,n=n:e(n,a),range(0,n))
mulmatbyarr=lambda M,A:trans(matrixmul(M,trans([A])))[0]
isdeg=lambda A:det(A)==0

# Лямбда-функции кончились, начались обычные

# 2
def list_(x):
 if type(x)==type([]):
  return x
 else:
  print x
  return list(x)

def orthog(as):
 bs=[]
 for ai in as:
  x=ai
  for bi in bs:
   bi2=scalmul(bi,bi)
   if(bi2==0):
    return None
   x=sub(x,mulbyn(bi,scalmul(ai,bi)/bi2))
  bs+=[x]
 return bs

# 3
def matrixmul(A,B):
 if (len(A[0])!=len(B)):
  return None
 Bt=trans(B)
 n,m=len(A),len(Bt)
 C=zeros(n,m)
 for i in range(0,n):
  for j in range(0,m):
   C[i][j]=scalmul(A[i],Bt[j])
 return C

def choose(x):
 i=0
```

```
 while (x[i]==0)and(i<len(x)):
  i+=1
 if i<len(x):
  return i
 else:
  return -1

def signature(i):
 s=0
 for j in range(0,len(i)):
  for k in range(j,len(i)):
   if i[k]<i[j]:
    s+=1
 return (s%2-(s+1)%2)

def det(A):
 if len(A)!=len(A[0]):
  return None
 M=A[:]
 n=len(M)
 indices=zeroarr(n)
 for i in range(0,n):
  k=choose(M[i]);
  for j in filter(lambda a,b=i:a!=b,range(0,n)):
   M[j]=sub(M[j],mulbyn(M[i],float(M[j][k])/M[i][k]))
  indices[i]=k
 su=1
 for i in range(0,n):
  su*=M[i][indices[i]]
 return su*signature(indices)

def Gauss_sol(A,b):
 if (len(A)!=len(A[0]))or(len(A)!=len(b)):
  return None
 M=A[:]
 c=b[:]
 n=len(M)
 indices=zeroarr(n)
 for i in range(0,n):
  k=choose(M[i]);
  for j in filter(lambda a,b=i:a!=b,range(0,n)):
   c[j]-=float(c[i])*M[j][k]/M[i][k]
   M[j]=sub(M[j],mulbyn(M[i],float(M[j][k])/M[i][k]))
  indices[i]=k
 x=zeroarr(n)
 for i in range(0,n):
```

```
  x[i]=c[indices[i]]
 return x

def Cramer_sol(A,b):
 if (len(A)!=len(A[0]))or(len(A)!=len(b)):
  return None
 n=len(A)
 x=zeroarr(n)
 Delta=det(A)
 for i in range(0,n):
  M=trans(A)
  M[i]=b
  x[i]=det(M)/Delta
 return x

def iszero(A,epsilon=1e-10):
 s=1
 for i in range(0,len(A)):
  for j in range(0,len(A[0])):
   if A[i][j]>epsilon:
    s=0
    break
  if not s: break
 return s

def equal(A,B,epsilon=1e-10):
 if (len(A)!=len(B))or(len(A[0])!=len(B[0])):
  return 0
 else:
  s=1
  for i in range(0,len(A)):
   for j in range(0,len(A[0])):
    if A[i][j]-B[i][j]>epsilon:
     s=0
     break
   if not s: break
  return s

# 4
def printv(v,t=10):
 f='%.'+`t`+'f'
 print '{',
 for x in v:
  print f%x,
 print '}'
```

```
def printm(M,t=10):
 f='%.'+`t`+'f'
 print '{',
 for x in M:
  print '{',
  for y in x:
   print f%y,
  if x==M[-1]:
   print '}',
  else:
   print '}'
 print '}'

if __name__ == "__main__":
 C=[[1,0],[2,1]]
 b=[1,10]
 print Cramer_sol(C,b)
```

6.5 Итерационные методы

В качестве основного источника информации по итерационным методам будем использовать фундаментальный труд Р.Барретта, М.Бери, Т.Ф.Чана, Дж.Деммела, Дж.М.Донейто, Дж.Донгарры, В.Айкаута, Р.Позо, Ч.Ромайна и Х.А.фан дер Форста «Шаблоны для решения систем линейных алгебраических уравнений: построение блоков для итерационных методов» [19, глава 2]. Он доступен в интернете по адресу http://www.netlib.org/templates/Templates.html. Кроме того, для написания материала данного раздела были использованы и другие, в большей степени устаревшие, но русскоязычные источники [2, 10, 12, 15].

Прежде всего, заметим, что даже принципиальное разных итерационных методов существуют многие десятки, а то и тысячи, и реализация их всех лежит вне компетенции данной работы. Поэтому кратно разберемся в их классификации и выделим те, что мы будем реализовывать. Итерационные методы делятся на стационарные (классические, методы Ричардсона с предобусловливанием) и нестационарные (методы Крылова) [19, 4].

Стационарным методом называется любой, где

$$x^{(k)} = Bx^{(k-1)} + c,$$

причём ни B, ни c не зависят от k. Среди стационарных методов используются метод Ричардсона (простой итерации), метод Якоби, метод Гаусса-Зейделя, метод успешной верхней релаксации (SOR) и симметричный метод успешной верхней релаксации (SSOR). Мы реализуем лишь первые два.

Нестационарные методы отличаются от стационарных тем, что используют информацию, которая изменяется при каждом шаге итерации. Как правило, константы вычисляются из невязок или других векторов, являющихся результатом итерационного метода. Среди нестационарных методов используются метод сопряженных градиентов, метод минимальных невязок, симметричный метод LQ (SYMMLQ), методы сопряженных градиентов для нормальных уравнений (CGNE и CGNR), обобщенный метод минимальных невязок (GMRES), метод бисопряженных градиентов (BiCG), метод псевдоминимальных невязок (QMR), метод сопряженных градиентов в квадрате (CGS), стабилизированный метод бисопряженных градиентов и итерация Чебышева.

6.5.1 Метод Ричардсона

Самый простой из всех итерационных методов, ещё справедливо называемый методом простой итерации, метод Ричардсона представляет собой каркас всех прочих стационарных методов. По данному начальному приближению x_0 он строит последовательность

$$x_1 = x_0 + b - Ax_0, \quad x_2 = x_1 + b - Ax_1, \quad \ldots, \quad x_{k+1} = x_k + b - Ax_k,$$

на каждом шаге прибавляя к уже имеющемуся приближению невязку. Следуя [19], мы ставим условие выхода следующим: $||r_k|| < \varepsilon$, где r_k — невязка, $r_k = b - Ax_k$. Больше ничего метод Ричардсона не содержит, поэтому мы немного углубимся в классические методы и реализуем метод Якоби.

```
def Richardson_sol(A,b,epsilon=1e-10,x0=None):
 if (len(A)!=len(A[0]))or(len(A)!=len(b)):
  return None
 k,xk=0,x0
 if (x0==None): xk=b[:]
 rk=sub(b,mulmatbyarr(A,xk))
 while (norma(rk)>epsilon):
  xk=add(xk,rk)
  k+=1
  rk=sub(b,mulmatbyarr(A,xk))
 return xk
```

6.5.2 Метод Якоби

Метод Якоби лишь чуть-чуть сложнее метода Ричардсона. В нём поправка уже не равна невязке, но легко вычисляется. По данной матрице A стоится матрица M, состоящая вне диагонали — из нулевых элементов, а на диагонали — из элементов A:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \approx \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} = M$$

Подобным же приёмом пользуются во всех остальных классических методах (Например, в методе Гаусса-Зейделя $M = D + L$, то есть от матрицы A берется отдельно диагональная часть и отдельно нижнетреугольная). Смысл его заключается в том, чтобы система $Mx = y$ легко решалась. В нашем случае M^{-1} состоит из тех же нулей, а на диагонали вместо a_{ii} стоят $\frac{1}{a_{ii}}$. Но ни в каком итерационном методе матрицы M или M^{-1} явно не вычисляются, а описывается отдельная процедура «превращения невязки в поправку», которая по данным невязке r_k и матрице A строит поправку δ_k. Соответствующая процедура на питоне занимает одну строчку — это λ-функция, поэлементно делящая свой аргумент на диагональ матрицы A. Вызов этой функции подставляется в ту строчку, которая отвечает за переход от x_{k-1} к x_k. Более ничем метод Якоби от метода Ричардсона не отличается. Подобный же подход к реализации может быть употреблен для написания любого классического итерационного метода.

```
def Jacobi_sol(A,b,epsilon=1e-10,x0=None):
 if (len(A)!=len(A[0]))or(len(A)!=len(b)):
  return None
 Mx=lambda r,A=A:map(lambda rj,j,A=A:rj/A[j][j],r,range(0,len(r)))
 k,xk=0,x0
 if (x0==None): xk=b[:]
 rk=sub(b,mulmatbyarr(A,xk))
 while (norma(rk)>epsilon):
  xk=add(xk,Mx(rk))
  k+=1
  rk=sub(b,mulmatbyarr(A,xk))
 return xk
```

6.5.3 Метод сопряженных градиентов

Данный метод эффективен для решения СЛАУ с симметричной, положительно определенной матрицей [2, 19]. Согласно методу, итерации происходят следующим образом:

$$
\begin{aligned}
&x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)},\\
&r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)},\\
&q^{(i)} = Ap^{(i)},\\
&\alpha_i = \frac{(r^{(i-1)}, r^{(i-1)})}{(p^{(i)}, q^{(i)})},\\
&p^{(i)} = r^{(i)} + \beta_{(i-1)} p^{(i-1)},\\
&\beta_i = \frac{(r^{(i)}, r^{(i)})}{(r^{(i-1)}, r^{(i-1)})},
\end{aligned}
$$

где $x^{(i)}$ — очередное приближение решения, $p^{(i)}$ — вектор направления дальнейшего поиска решения, $r^{(i)}$ — вектор невязок, α_i — коэффициент, минимизирующий ${r^{(i)}}^{\tau} A^{-1} r^{(i)}$, β_i — коэффициент для ортогонализации $p^{(i)}$ и $Ap^{(i)}$ (или, что эквивалентно, $r^{(i)}$ и $Ar^{(i)}$).

Ясно, что совсем не обязательно хранить все предыдущие приближения (достаточно одного-двух предыдущих). Таким образом, нельзя сразу запускать обычный алгоритм для $i < 2$. Алгоритм принимает вид:

$$
\begin{aligned}
&r^{(0)} = b - Ax^{(0)}\\
&\rho_0 = (r^{(0)}, r^{(0)})\\
&p^{(1)} = r^{(0)}\\
&q^{(1)} = Ap^{(1)}\\
&\alpha_1 = \frac{\rho_0}{(p^{(1)}, q^{(1)})}\\
&x^{(1)} = x^{(0)} + \alpha_1 p^{(1)}\\
&r^{(1)} = r^{(0)} + \alpha_1 q^{(1)}
\end{aligned}
$$

Далее для $i = 2, 3, \ldots$ выполняется следующее:

$$
\begin{aligned}
&\rho_{i-1} = (r^{(i-1)}, r^{(i-1)})\\
&\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}}\\
&p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}\\
&q^{(i)} = Ap^{(i)}\\
&\alpha_i = \frac{\rho_{i-1}}{(p^{(i)}, q^{(i)})}\\
&x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}\\
&r^{(i)} = r^{(i-1)} + \alpha_i q^{(i)}
\end{aligned}
$$

Всё вышесказанное легко, строчка за строчкой, переписывается в текст программы на питоне (выражается через сформулированные выше функции скалярного произведения, произведения матриц, etc):

```
def CG_sol(A,b,epsilon=1e-10,x0=None):
 if (len(A)!=len(A[0]))or(len(A)!=len(b)):
  return None
 if (x0==None):
  x=b[:]
 else:
  x=x0[:]
 r=sub(b,mulmatbyarr(A,x))
 rho1=scalmul(r,r)
```

```
p=r[:]
q=mulmatbyarr(A,p)
if (scalmul(p,q)==0):
 return x
else:
 alpha=rho1/scalmul(p,q)
x=add(x,mulbyn(p,alpha))
r=sub(r,mulbyn(q,alpha))
while(norma(r)>epsilon):
 rho2,rho1=rho1,scalmul(r,r)
 p=add(r,mulbyn(p,rho1/rho2))
 q=mulmatbyarr(A,p)
 alpha=rho1/scalmul(p,q)
 x=add(x,mulbyn(p,alpha))
 r=sub(r,mulbyn(q,alpha))
return x
```

Глава 7

Математика и питон

Как было видно из предыдущих глав, на питоне успешно реализуются многие широко используемые математические алгоритмы. Несмотря на простоту и эффективность этого подхода, существуют и более массивные вещи, разработанные для решения широкого спектра математических задач, одно из которых — Numeric Python, или NumPy [18].

NumPy представляет собой множество расширений для языка программирования питон, позволяющих программистам на питоне эффективно манипулировать большими множествами объектов, организованных наподобие сеток. Эти множества объектов в терминологии NumPy называются массивами (потому как один из основных композитных типов данных, изучаемых в первых лекциях, в питоне называется списком — и это название действительно правильно). Массивы могут иметь любое число измерений: одномерные массивы похожи на последовательности питона, а двумерные — на матрицы или тензоры второго ранга. Конечно, одномерные массивы немного отличаются от любого типа последовательностей, а двумерные массивы чуть-чуть не похожи на матрицы линейной алгебры.

Зачем же нужны эти расширения? Основная причина прозаична, она состоит в том, что манипулирование множеством миллионов чисел в стандартных типах питона (списках, кортежах или классах) слишком медленно и требует слишком много памяти. Всё, что можно сделать с помощью NumPy, доступно и из обычного питона — правда, мы можем не дожить до конца расчетов. Вторая, более тонкая причина существования этих расширений лежит глубже. Все типы операций, обычно осуществляемых программистами над массивами (которые подчас чрезвычайно сложны), могут быть разложены на малое множество элементарных операций, не принадлежащих к стандартным. Это разложение выполняется похожим образом во многих языках программирования, содержащих массив как тип данных [3]. Если взглянуть внимательно, NumPy представляет собой всего лишь приложение этого опыта к языку питон — поэтому многие операции, описанные в NumPy, работают аналогично описанным в других местах, именно так, как удобнее всего, согласно большому опыту. Документация NumPy [18]

утверждает, что они использовали семейство APL-подобных языков, язык программирования системы Матлаб, древний, но ещё живой язык вычислительных программистов Фортран, S и S+.

В зарубежных источниках (см. например http://cens.ioc.ee/projects/f2py2e) удалось обнаружить попытки использования питона в больших математических проектах. Следуя вышеприведенной ссылке, к примеру, можно найти написанный итальянцем Пеару Петерсоном транслятор из фортрана в питон — явный признак того, что математики оценили питон (с фортраном они не могут расстаться с семидесятых годов).

Заключение

Был проведен глубокий анализ существующих языков программирования на предмет выявления языка, подходящего для начального обучения информатике и программированию. Подробно точка зрения автора излагается в главе 1. В результате анализа выбор пал на язык программирования Питон.

Был разработан полный обучающий комплекс по этому языку программирования, включающий курс лекций с презентациями (см. главу 3), тестирующую программу (см. главу 4), набор практических занятий (см. главу 5), а также широкий спектр разработанных приложений, готовых к применению в математическом программировании (см. главы 6 и 7). Этот комплекс, как уже было отмечено в главе 2, готов к употреблению как в классическом, так и в дистанционном обучении.

Результаты работы в означенном направлении докладывались на студенческих конференциях и на учебно-методической конференции «Современные информационные технологии в учебном процессе», по этой же теме имеется публикация [11].

Обучающий комплекс по обучению языку программирования Питон, расположенный по адресу `http://www.pythonc.by.ru`, завершён и рекомендуется к принятию в учебный план либо к использованию в качестве дистанционного пособия по программированию.

Литература

[1] Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д. *Структуры данных и алгоритмы.* — М.: Издательский дом «Вильямс», **2000**.

[2] Бахвалов Н. С. *Численные методы (анализ, алгебра, дифференциальные уравнения)* — М.: Наука. Гл. ред. физ.-мат. лит., **1973**.

[3] Бен-Ари М. *Языки программирования. Практический сравнительный анализ:* Пер. с англ. — М.: Мир, **2000**.

[4] Бочев М. А. *GMRES, BiCGSTAB и другие методы пространства А.Н.Крылова,* доклад ка кафедре Математического Моделирования, декабрь **2000**.

[5] Буч Г. *Объектно-ориентированный анализ и проектирование с примерами приложений на C++*, 2-е изд./Пер. с англ. — М.: «Издательство Бином», СПб.: «Невский диалект», **2000**.

[6] Вирт Н. *Алгоритмы и структуры данных:* Пер. с англ. — М.: Мир, **1989**.

[7] Беллман Р. *Введение в теорию матриц,* Пер. с англ. — М.: «Наука», **1969**.

[8] Гантмахер Ф. Р. *Теория матриц* — М.:"Наука **1967**.

[9] Говорухин В. Н., Цибулин В. Г. *Компьютер в математическом исследовании.* Учебный курс. — СПб.: Питер, **2001**.

[10] Годунов С. К., Рябенький В. С. *Разностные схемы (введение в теорию)*, уч. пособие — М.: Наука. Гл. ред. физ.-мат. лит., **1977**.

[11] Зайцев В. В., Литвиненко А. Н. *Питон как первый язык для обучения информатике,* Тезисы докладов учебно-методической конференции *«Современные информационные технологии в учебном процессе»*, страницы 61–65 — Ростов-на-Дону, **2002**.

[12] Калиткин Н. Н. *Численные методы* — М.: Наука. Гл. ред. физ.-мат. лит., **1978**.

[13] Кнут Д. Е. *Всё про TEX*, Протвино, **1993**.

[14] Машбиц Е. И., Каптелинин В. Н., Маргулис Е. Д. *Введение в язык Лого*: Учеб. пособие; Под общ. ред. Стогния А. А., Ющенко Е. А., Машбица Е. И. — К.: Выща шк., **1989**.

[15] Самарский А. А. *Введение в численные методы,* 2-е изд., перераб. и доп. — М.: Наука. Гл. ред. физ.-мат. лит., **1987**.

[16] Страуструп Б. *Язык программирования C++,* 3-е изд. /Пер. с англ. — М.: «Издательство Бином», СПб.: «Невский диалект», **1999**.

[17] Хювёнен Э., Сеппянен Й. *Мир Лиспа.* В 2-х т. Т.1: *Введение в язык Лисп и функциональное программирование.* Пер. с финск. — М.: Мир, **1990**.

[18] David Ascher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin, Travis Oliphant, *Numerical Python*, **1999**.

[19] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, Henk van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, **1994**.

[20] J. L. Bentley *Writing Efficient Programs,* Prentice-Hall, Englewood Cliffs, NJ, **1982**.

[21] Timothy Budd, *Multiparadigm Programming in Leda*, Addison-Wesley, **1995**.

[22] *Cascading Style Sheets, level 2. CSS2 Specification*, http://www.w3.org/TR/1998/REC-CSS2-19980512, Editors: Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, W3C Recommendation 12 May **1998**.

[23] Alonzo Church *The Calculi of Lambda Conversion,* Annals of Math. Studies, No. 6, Princeton University Press, Princeton, NJ, **1941**.

[24] *Core Techniques for Web Content Accessibility Guidelines 1.0*, http://www.w3.org/TR/2000/NOTE-WCAG10-CORE-TECHS-20001106, Editors: Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs, W3C Note 6 November **2000**.

[25] *CSS Techniques for Web Content Accessibility Guidelines 1.0*, http://www.w3.org/TR/2000/NOTE-WCAG10-CSS-TECHS-20001106, Editors: Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs, W3C Note 6 November **2000**.

[26] Bruce Eckel, *Thinking in Java*, Prentice Hall, Upper Saddle River, New Jersey, **1998**.

[27] Leo Geurts, Lambert Meertens, Steven Pemberton, *The ABC Programmer's Handbook*, Prentice-Hall, Englewood Cliffs, New Jersey, **1989**.

[28] *HTML 4.01 Specification*, Editors: Dave Raggett, Arnaud Le Hors, Ian Jacobs, http://www.w3.org/TR/1999/REC-html401-19991224, W3C Recommendation 24 December **1999**.

[29] *HTML Techniques for Web Content Accessibility Guidelines 1.0*, http://www.w3.org/TR/2000/NOTE-WCAG10-HTML-TECHS-20001106, Editors: Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs, W3C Note 6 November **2000**.

[30] Robin Jones, Clive Maynard, Ian Stewart, *The Art of Lisp Programming*, Springer Verlag, Berlin, **1990**.

[31] Guido van Rossum, Fred L. Drake, Jr., editor, *Python Library Reference*, PythonLabs, Release 2.1, April 15, **2001**.

[32] Guido van Rossum, Fred L. Drake, Jr., editor, *Python Reference Manual*, PythonLabs, Release 2.1, April 15, **2001**.

[33] Guido van Rossum, Fred L. Drake, Jr., *Python Tutorial*, Release 2.1, April 15, **2001**.

[34] *Techniques for Web Content Accessibility Guidelines 1.0*, http://www.w3.org/TR/2000/NOTE-WCAG10-TECHS-20001106, Editors: Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs, W3C Note 6 November **2000**.

[35] Larry Wall, Tom Christiansen, Randal L. Schwartz, with Stephen Potter, *Programming Perl*, 2nd ed., O'Reilly & Associates, September **1996**.

[36] *Web Content Accessibility Guidelines 1.0*, http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505, Editors: Wendy Chisholm, Gregg Vanderheiden, Ian Jacobs, W3C Recommendation 5 May **1999**.

[37] Leslie B. Wilson, Robert G. Clark, *Comparative Programming Languages*, 3rd ed., rev. and upd. by R. G. Clark, Addison-Wesley Publishers, **2001**.

Оглавление