Руководство по Вычислительной Математике

Ольга Вячеславовна Перл

Февраль 2025
версия: 11 февраля 2025 г.

Оглавление

Часть I

Формат курса

Глава 1

Ресурсы курсы

1.1 Чат курса в Telegram

Ссылка: `https://t.me/+oRRtt6uTkiRjN2Fi`

В чате мы обсуждаем материалы и задания курса, возникающие трудности и ведём всю коммуникацию по курсу.

В связи с тем, что никнеймы у всех разные, для того, чтобы было проще вести коммуникацию, а также для некоторых тестов, Вам предлагается пройти быструю форму "регистрации" в чате курса . Это нужно для того, чтобы помощь была более адресной, а Ваши баллы за такие тесты не потерялись. Форма включает в себя только Ваш username, по которому Вас можно упоминать в чате, а также Ваш isu id. Данные из формы не будут предоставлены другим участникам курса, а также сторонним лицам.

Username преподавателя - @Heldera.

1.2 Moodle

Ссылка: `https://learning.cosm-lab.science/moodle/`

В Moodle будут представлены следующие разделы задания курса: рубежные работы (для всех), лабораторные работы, тесты по научным статьям.

По умолчанию все студенты уже зарегистрированны в Moodle и записаны на курс преподавателем. Обратите внимание на ссылку - это Moodle лаборатории.

Username - Ваш табельный номер ИТМО. Он же является паролем для первого входа. Если у Вас возникли трудности с авторизацией или курсом, пожалуйста, не стесняйтесь сразу сообщить об этом преподавателю.

Если курс не отображается у Вас на главной странице после входа, то можно выбрать в меню слева пункт "Домашняя страница"/ "Dashboard"и из списка доступных курсов выбрать "Вычислительная Математика 2025".

В Moodle, в соответствии с разделами курса, становятся доступны тесты по научным статьям, рубежные работы, а также разделы для загрузки

отчётов по лабораторным работам в формате PDF. Код отчётов по лабораторным работам выполняется и загружается на платформе Code&Test.

1.3 Диск с материалами курса

Ссылка: `https://disk.yandex.ru/d/92lTwgcTnrqteg`

На даннам диске собраны материалы полезные при освоении курса: материалы по домашним заданиям (видео и тексты), книги и статьи для разных уровней, а также дополнительные полезные материалы, например, книги по Python, книги про применение методов и другие.

1.4 Журнал

Ссылка: Google-журнал

В журнале собираем все ваши баллы и суммируем. Также в журнале можно мосмотреть лидеров с максимальным количеством баллов, таблица снижения баллов по лабораторным работам и иные результаты прохождения курса.

1.5 Code&Test

Ссылка: `http://code-n-test.cloud.sdcloud.io/ui/`

На данном ресурсе после каждого лекционного занятия Вам будет открыт доступ к следующей лабораторной работе. Он также откроется автоматически после самостоятельного выполнения предыдущей работы. Поэтому потенциально, прогресс Ваших лабораторных работ не зависит от расписания лекционных занятий.

Обратите внимание, что Code&Test лишь проверяет Ваш код лабораторной работы (на некоторые тесты и плагиат). Для того, чтобы лабораторная работа считалась выполненной, необходимо также подготовить отчёт по лабораторной работе и загрузить его в соответствующий раздел в Moodle.

Отдельно отметим, что лабораторные работы (как код, так и отчёт) необходимо выполнять самостотельно, не прибегая к помощи коллег, ИИ-ассистентов или случайных сайтов в Интернете. В таком случае у Вас появляется шанс действительно чему-то научиться. В противном случае, если система проверки плагиата обнаруживает у Вас плагиат, то доступ к следующей лабораторной работе будет закрыт, текущая лабораторная работа будет заблокирована. Для того, чтобы выполнить текущую лабораторную работу, преподавателю необходимо будет вручную заменить Вам вариант, который изначально выдаётся автоматически. Если Вы хотите быстрее получить новый вариант, напишите об этом преподавателю. В случае повторного плагиата, Вам необходимо будет реализовать код по предоставленной научной статье.

Глава 2

Способы получения баллов по курсу

2.1 Лекции

Посещение лекционных занятий и выполнение тестов на них. (16 баллов).

Во время проведения лекций будут проходить быстрые тесты, включающие себя вопросы по домашним работам и материалу текущей лекции. Максимум за занятие можно получить 2 балла. За все 8 лекций - 16 баллов.

Если за занятие получено 0 баллов (тест не выполнен или выполнен неверно), то Вам будет автоматически предоставлен доступ к тесту по научной статье по теме лекции. Предполагается, что Вы вначале подробнее разберётесь в теме лекции, затем изучите статью, а после - выполните тест. Подробнее см. про тесты по научным статьям.

2.2 Практические занятия

Посещение практических занятий и выполнение заданий на них. (44 балла)

2.3 Рубежные работы

Рубежные работы (2 штуки по 10 баллов).

В курсе предполагается 2 рубежных работы, за каждую из которых можно получить до 10 баллов. Обе рубежные работы будут проводиться на платформе Moodle. Рубежная работа 1 будет проводиться по материалам лекций 1-4 включительно, а рубежная работа 2 по материалам лекций 5-7. Основное правило для подготовки к рубежным работам – это усвоение материала лекционных и рубежных работ. Помимо теоретических заданий, в работах будут присутствовать и задания на вычисления каких-либо значений. Аналогичные задачи будут решаться и разбираться на лекционных занятиях. Решение задач на лекциях не оценивается дополнительно, однако своевременная проработка задач поможет при подготовке курса. Поэто-

му посещение лекционных занятий, а также активная работа на занятиях и выполнение соответствующих домашних заданий существенно улучшит ваши результаты при написании рубежных работ.

Рубежные работы не содержат в себе заданий на написание кода по изученным численным методам, однако владеть ими всё-таки придётся, потому что некоторые задания требуют вычисления по конкретным формулам численных методов и вычисление по другим формулам вероятнее всего приведёт вас к неправильному результату.

Полезным при подготовке к рубежным работам также будет повторение материалов лекций. Презентации по лекциям будут выложены в чат Telegram, а также на диск. Если Вам необходимы материалы на английском языке, Вы можете обратиться за ними к преподавателю.

2.4 Лабораторные работы

Максимальное количество баллов при выполнении в соответствии с требованиями и в срок - 30 баллов. Выполнение лабораторных работ является обязательным, если пропущены лабораторные занятия по соответствующим разделам курса.

Оценка по каждой лабораторной работе состоит из 2 частей: оценка за код (расчитывается на основании пройденных автоматических тестов на платформе Code&Test) и оценка за отчёт/теорию. Обе оценки изначально рассчитываются в процентах.

После этого высчитвается дата сдачи лабораторной работы как позднейшая из дат загрузки кода и загрузки отчёта. Например, лабораторная работа выполнена на Code&Test 1 марта, а отчёт загружен в Moodle 2 марта. Тогда, после проверки преподавателем, в случае успеха, датой закрытия лабораторной работы считается 2 марта.

Для того, чтобы лабораторная работа считалась сданной / закрытой, необходимо получить минимум 60% за выполнение и 60% за отчёт.

После этого на основе даты и полученных оценок в процентах рассчитывается количество баллов за лабораторную работу. Максимальное количество баллов за лабораторную работу снижается каждые 2 недели после выдачи работы (лекции по соответствующей теме).

дата	1	2	3	4	5	6	1	2	3	4	5
макс. сумма	6	6	6	6	6	20	каскад штрафов				
20.02.2025	6	6	6	6	6	20	-	-	-	-	-
27.02.2025	6	6	6	6	6	20	-	-	-	-	-
13.03.2025	5,4	6	6	6	6	20	-1	-	-	-	-
27.03.2025	4,86	5,4	6	6	6	20	-2	-1	-	-	-
10.04.2025	4,37	4,86	5,4	6	6	20	-3	-2	-1	-	-
24.04.2025	3,94	4,37	4,86	5,4	6	20	-4	-3	-2	-1	-
08.05.2025	3,54	3,94	4,37	4,86	5,4	20	-5	-4	-3	-2	-1
22.05.2025	3,19	3,54	3,94	4,37	4,86	20	-6	-5	-4	-3	-2
01.06.2025	2,87	3,19	3,54	3,94	4,37	20	-7	-6	-5	-4	-3

Подробнее о правилах выполнения лабораторных работ см. специальный раздел.

2.5 Зачёт

Зачёт представляет собой лабораторную работу №6, за которую при успешном выполнении можно получить от 12 до 20 баллов. Задание на лабораторную работу 6 будет выдано 1 мая.

Если 5 лабораторных работ сданы до 1 мая, то баллы за 6 зачётную лабораторную работу можно получить автоматов (среднее по выполнению в процентах и среднее по отчёту в процентах на основании предыдущих лабораторных работ). Если, например, количество баллов, предлагаемых автоматом, Вас не устраивает, то Вы можете выполнить зачётную лабораторную работу. Автомат при этом не сгорает - Вы сможете получить максимум из баллов.

Баллы за зачёт не снижаются через 2 недели, но для закрытия дисциплины должны быть сданы до даты зачёта по расписанию сессии.

2.6 Тесты по научным статьям

В случае, если за тесты на лекциях получено 0 баллов или тест на лекции не был написан, сразу после лекции Вам будет предоставлен доступ к тесту по научной статье. Количество баллов за тесты по научным статьям равно количеству баллов за тесты на лекциях - 16.

Если Вам выдан доступ к тесту по научной статье, это означает, что в Moodle у Вас появился тест. Статью из теста необходимо скачать и внимательно изучить, разобраться в формулах, обозначениях и смысле работы. Большинство статей будут на английском языке. Сами тесты также на английском языке.

Прежде чем браться за изучение статьи и выполнение теста, настотельно рекомендуется внимательно изучить материалы лекции. Это во многом упростит разбор материала публикации.

Если Вы уверены, что изучили статью, можете приступать к выполнению теста. Тест будет содержать 2 вопроса. На выполнение теста отводится 90 минут.

Крайний срок выполнения тестов по научным статьям - дата зачёта по расписанию сессии. Тем не менее, настоятельно не рекомендуется откладывать выполнение этих тестов на "конец полосы потому что объем работ к началу сессии обычно и без этого становится большим.

Глава 3

Стратегии получения зачёта

Исходя из способов получения баллов, описанных в предыдущей главе, можно предложить несколько способов получения зачёта по дисциплины.

3.1 Стандартная

Вы ходите на все занятия (лекции и лабораторные занятия), выполняете все домашние задания, тесты на лекциях, задания на лабораторных занятиях и 2 рубежные работы. Итого получается максимумально баллов: 10*2=20 баллов за рубежные работы, 8*2=16 баллов за тесты на лекциях, 3.14*14=44 за задания на лабораторных занятиях. Итого: 20+16+44=80. Вам также будет предоставлен доступ к зачётной лабораторной работе №6, за которую можно получить ещё 20 баллов.

Если задания на практиках выполнены не полностью, то необходимо выполнить соответствующую лабораторную работу. Например, если пропущено 4-е лабораторное занятие, то необходимо выполнить лабораторную работу №2. Дополнительное посещение лабораторного занятия 5 даст Вам дополнительные баллы в любом случае, не смотря на то, что оно тоже относится к теме 2-й лабораторной работы.

Если тесты на лекциях выполнены не успешно, Вам будет предоставлен доступ к тестам по научным статьям.

Если за рубежную работу получено менее 6 баллов, то Вам будет предложена ещё одна попытка справиться с ней.

Таким образом, если Вы посещаете занятия и выполняете задания, то нужно только написать рубежные работы, а лабораторные работы можно не выполнять. Если хочется больше баллов за курс, то можно написать ещё и зачётную лабораторную работу в конце.

3.2 Рискованная

Вы пропускаете все или выборочно посещаете лекции и лабораторные занятия. Вместо тестов на лекциях Вам предоставляются тесты по науч-

ным статьям, соответствующим тематике лекций и дающим столько же баллов, сколько и тест на лекции. Вместо заданий на лабораторных занятиях в день прочтения лекции Вам предоставляется доступ к лабораторной работе по соответствующей теме. Максимум баллов за лабораторную работу снижается каждые 2 недели. Если Вы выполняете лабораторную работу в течении 2 недель после лекции, то баллы не снижаются. Всё также обязательным является написание рубежных работ. Максимум по этой стратегии можно получить: 6*5=30 за 5 лабораторных работ по 6 баллов, 10*2=20 за рубежные работы, 2*8=16 за тесты по научным статьям. Итого: 30+20+16=66 баллов, при условии, что все лабораторные работы сданы в срок. Дополнительно можно выполнить зачётную лабораторную работу 6, чтобы получить больше баллов.

Если 5 лабораторных работ сданы до 1 мая, то баллы за 6 зачётную лабораторную работу можно получить автоматов (среднее по выполнению и среднее по отчёту на основании предыдущих лабораторных работ).

Как видно, стратегия является более рискованной в плане получения зачёта и максимальное количество баллов ниже, а задания труднее.

3.3 Прикладная

В рамках освоения курса, в течении февраля (до 1 марта) Вам будет предоставлена возможность записаться в научно-прикладной проект, где Вы сможете применять знания, получаемые на курсе. В таком случае Вам обязательны будут выполнения только 2 рубежных работ. Рекомендуется также посещать лекции, так как материал на них будет полезен для Вашей разработки. Количество мест в проекте ограничено.

При выборе этой стратегии необходимо понимать, что Вы должны будете непосредственно участвовать в проекте и работать над выделяемыми Вам задачами. В случае, если Вы записались в проект и Вам была предоставлена такая возможность, а над проектом не ведётся планомерной работы в течении всего семестра, то стратегия превращается в рискованную, а получить зачёт через участие в проекте для Вас становится не возможным.

3.4 Безумная

Если в течении семестра Вы как минимум сдали 2 рубежные работы (2 рубежная работа проводится 22 мая), то на последнем занятии курса, 27 мая, состоится мини-хакатон по трейдингу, где все участники смогут получить 3.14 балла, как за задание на лабораторном занятии, а победитель получит ещё и зачёт автоматом.

К мини-хакатону можно готовиться заранее, но нужно понимать, что в случае, если не удастся одержать победу, стратегия превратится в радикальный вариант рискованной стратегии, когда все лабораторные уже имеют большие штрафы за несоблюдение сроков и общий объем работы, которую нужно сделать для зачёта - большой.

Глава 4

Наиболее частые ситуации и способы их решения

4.1 Восстановился

Если Вы ранее Вы уже начинали освоение этого курса, то обязательно упомяните это в письме преподавателю. В любом случае обязательно свяжитесь с преподавателем, чтобы все были в курсе Вашей ситуации.

Если Вы восстановились на 3й курс, то стратегия аналогична наличию академической разницы (см. ниже). Если Вы восстановились на 2й курс, то в зависимости от того, что укажет деканат, Вы либо действуете аналогично повторному освоению, либо просто изучаете дисциплину вместе со всеми студентами курса. Разница фактически в дате, к которой Вы должны закрыть курс. О конкретных датах Вы можете узнать в деканате.

4.2 Повторное освоение

Повторное освоение предполагает, что Вы будете изучать курс заново: ходить на лекционные и лабораторные занятия, выполнять задания курса и получать баллы. Если деканат направил Вас на повторное освоение дисциплины, пожалуйста, сообщите об этом преподавателю, чтобы своевременно Вам были предоставлены все необходимые доступы и ресурсы, так как списки студентов в ИСУ часто запаздывают и у Вас будет шансы пропустить большую часть курса.

Повторное освоение дисциплины отличается от обычного изучения курса тем, что оно должно закончиться раньше, до основной сессии. О конкретных датах окончания повторного освоения дисциплины узнавайте в деканате.

4.3 Академическая разница

Если у Вас есть академическая разница при переводе с другого направления, Вам необходимо написать преподавателю на почту. Вам будут предоставлены материалы курса для самостоятельного изучения, задания на лабораторные работы, а также доступы к тестам. После освоения материалов курса, а также после выполнения лабораторных работ и тестов, Вам необходимо будет повторно связаться с преподавателем, чтобы работы были проверены. Если всё будет выполнено успешно, то сведения о закрытии Вами дисциплины преподаватель передаст в деканат.

4.4 Перевод в другую группу

Если Вы недавно перевелись в другую группу и заметили, что в журнале Вы отображаетесь не в той группе, свяжитесь, пожалуйста, с преподавателем, это упростит выставление полученных баллов.

Если Вы знаете, что кто-то из Ваших одногруппников уже перевелся в другую группу, а преподаватель продолжает розыскивать его в чате.

4.5 Смена имени/фамилии

Главное, что табельный номер ИСУ у Вас остался прежним. Если Вы по любой причине сменили имя, пожалуйста, сообщите об этом преподавателю, чтобы Ваши полученные баллы было проще правильно перенести в ИСУ.

4.6 Я не хочу посещать занятия

Все занятия курса проводятся в дистанционном формате. Посещение занятий не только входит в Ваши обязанности, как студента, но в действительно полезно, если в результате обучения в университете Вы расчитываете получить не просто "корочку"диплома, но и знания.. "на подкорочке".

Так или иначе, если по любым причинам Вы не можете посещать занятия, рассмотрите для себя рискованную стратегию получения зачёта.

4.7 Курс слишком простой

Если Вы изучили материалы курса и они кажутся Вам слишком простыми, рекомендуем в первую очередь обратить внимание на дополнительные материалы, которые будут более детально освещать изучаемую тему. Также Вы можете ознакомиться со списком дополнительных источников, чтобы углубить свои знания.

Если Вам хотелось бы получить более сложные задания и разобрать более глубокие темы дисциплины, обратитесь к преподавателю.

4.8 Курс слишком трудный

Преподаватель курса делает всё возможное, чтобы сделать его доступнее для разных уровней освоения. Однако, сделать это на больших потоковых курсах не всегда легко.

Если Вам кажется, что курс для Вас трудный, что некоторые тезисы или материалы Вам не понятны - не стесняйтесь обращаться к преподавателю. Сделать это можно как в общем чате (а вдруг другим тоже не понятно!?) так и индивидуально. Если что-то не понятно, то это нормально, для этого Вы и учитесь в Университете. Если бы Вы всё знали и всё понимали, то зачем Вам нужен Университет?

Помимо этого, Вы можете обратить внимание на книги, рекомендованные для базового освоения математики и программирования. Прежде всего книга Джейсон Уилкс - "Математика в огне"и "Чистый Python. Тонкости программирования для профи. Бейдер Д.

Не обращайте внимание на то, что вторая книга по названию не похожа на основы программирования. На самом деле, это прекрасная книга для Вас: учась на 2 курсе Университета, у Вас уже так или иначе был опыт программирования, а то, что синтаксис Python Вам может быть ещё не знаком - это не проблема. Книга как раз расскажет Вам о том, как писать чистый и понятный код на Python без всяких лишних отступлений о том, что такое переменная. Единственное, перевод этой книги на русский язык не очень хороший, если есть возможность, попробуйте читать её в оригинале, язык повествования там действительно простой.

Что касается первой книги, она также есть на диске на английском языке. Книга рассказывает об основных понятиях математики, терминах и о том, что откуда получается. Язык повествования максимально простой, чтобы не сказать детский.

4.9 Я боюсь математики

Ситуация вполне типичная. Для этого даже существует специальный термин - математическая тревожность. Для того, чтобы Вам проще было в освоении дисциплины был подготовлен курс по борьбе с математической тревожностью, который Вы можете самостоятельно изучить в Moodle.

4.10 Поздно приступил к освоению курса

В случае, если Вы поздно приступили к освоения курса, Вам первым делом необходимо постараться не пропускать оставшиеся занятия и выполнять текущие задания в срок.

За пропущенные лекционные занятия Вам необходимо будет выполнить тесты в Moodle по научным статьям, доступ к которым появится автоматически (в противном случае свяжитесь с преподавателем).

За пропущенные лабораторные занятия Вам необходимо будет выполнить лабораторные работы, код которых загрузить на Code&Test, а отчёты - в Moodle. Подробнее об этом читайте в главе про лабораторные работы.

Часть II

Содержание занятий

Глава 5

Лабораторное занятие 1. 11.02. Введение. Математическая тревожность. Чистый код.

5.1 Основные материалы лекции

Почему математика важна?

- Математика – это язык формального, унифицированного описания мира.
- Математика обладает достаточной мерой абстракции, чтобы развивать фантазию, воображение и образ мышления.

Качества и навыки программиста, развиваемые математикой

1. Умение мыслить абстрактно, выделять общее и частное;
2. Умение анализировать структуры и методы;
3. Умение формально описывать сущности и пути достижения желаемого;
4. Умение видеть и создавать красоту;
5. Умение упрощать методы и находить альтернативные пути решения;
6. Навыки стратегического мышления.

Первое – это способность мыслить абстрактно, различать общее и частное. Например, когда вы создаете несколько классов и вам следует создать иерархию, вы можете создать несколько абстрактных классов и поместить в них общую часть, а конкретные части поместить в дочерние классы.

Другой пример - когда вы работаете над каким-то требованием к продукту и вам следует указать общие и конкретные варианты использования.

Второе – это способность анализировать структуры и методы. Не уверена, что я должна это разъяснять, надеюсь, вы понимаете: когда мы обучены работать со сложными математическими структурами, способны доказывать некоторые теоремы и т. д. Это помогает вам ежедневно делать то же самое в области программирования: создавать структуры и методы, которые содержат правильные цели.

Третье – умение формально описать суть и способы достижения желаемого. Например, когда вы работаете над каким-либо требованием к продукту, вы должны разработать и понять, какие дополнительные функции вы должны охватить. Опишите поведение пользователей вашего продукта и их потребности. То же самое, что вы делаете, когда доказываете некоторые теоремы по математике.

Четвертое – это умение видеть и создавать Красоту. Это очень важный навык для программистов. Я была очень рада, что на моей первой работе в моей команде было много высококлассных специалистов со значительным опытом разработки программного обеспечения. Их код в любом случае был прекрасен, в нем было очень легко ориентироваться и понимать его. В то же время большое количество молодых программистов, в основном студентов, любят писать очень сложный код, чтобы показать, что другие, кто может его просмотреть, дураки, “если они не могут понять мой гениальный код”.

Самое важное правило разработки программного обеспечения заключается в том, что ваш код должен быть легок для понимания другими программистами. Тот же навык вы тренируете в математике: вы должны использовать самый простой из применимых методов, в другом случае вам и другим будет трудно проверить ваше решение, и это увеличит вероятность ошибок.

Пятое – способность упрощать методы и находить альтернативные решения. Я хорошо знаю, что легко сделать что-то более сложное, но чрезвычайно трудно сделать что-то более простое. Например, когда вы решаете какое-то уравнение, вы упрощаете обе его стороны (левую и правую), а затем применяете некоторые операции. Существует множество способов решения одного уравнения, но если вы хорошо подготовлены и хорошо разбираетесь в математике, то вы выберете самый простой способ решения этого конкретного уравнения. И если у вас возникнут какие-то трудности при решении, вы можете просто сменить выбранный метод на другой. И те же способности, которые вам нужны для написания кода, или для разработки каких-то функций, или для чего-то еще. В каждый конкретный момент, как и в нашей повседневной жизни, математика помогает вам упростить сложную задачу, которую вам нужно решить, и найти альтернативные способы решения этой проблемы.

И шестое. Математика улучшает ваши навыки стратегического мышления. Как вы, возможно, знаете, в армии много математики используется, как бы это сказать, для реальной стратегии. Если кто-то хочет иметь

армию и эффективно управлять ею, вы должны знать математику и понимать, как она работает. Та же ситуация с экономикой или, в конце концов, с компьютерными играми. Но для нас, как для инженеров-программистов, также очень важно обладать навыками стратегического мышления. Вы должны уметь понимать цену своих решений. Например, в вашей программе есть какая-то ошибка. Что вы выберете: решите это быстро, но тогда у вас возникнут некоторые проблемы с архитектурой в будущем; или исправьте это медленно и очень тщательно, но тогда вам придется отложить дату выпуска и, возможно, дату, когда вы начнете получать деньги за свой продукт; или вы пропустите эту ошибку сейчас, потому что она чрезвычайно мала и может быть легко исправлена в будущем, но не сейчас, когда у вас будет больше информации, например, но этот путь приведет вас в технический долг, список проблем, которые вам придётся решать в будущем вместо разработки новых функций. Что вы выберете? И это становится чрезвычайно важным, если вы создаете свой собственный стартап, свой продукт, чтобы зарабатывать деньги. Низкое качество, поздняя реализация, технический долг и очень длинный список одних и тех же вещей. Вы должны выбрать, и вы решаете. В этом случае вам необходимо обладать навыками стратегического мышления, потому что это поможет вам сделать оценку, прогноз, чтобы предсказать, что поможет дальше и что вы можете сделать сейчас, чтобы улучшить это. Есть много фильмов, например, о том, как это работает и почему математика в этом так важна. Один из них –“Игра на понижение” 2015 года.

Как изучать математику?

Здесь я привожу кратко некоторые заметки о том, как программисту стоит изучать математику. Если у вас будут свои замечания по этому поводу я с удовольствием их обсужу. В дополнение рекомендую книгу "Думай как математик Барбары Оакли.

- Выделять структуры и модули в разбираемом математическом тексте.
- Изучать связные и используемые определения и термины. Рекурсивно искать термины и понятия, встретившиеся ранее.
- Референтный подход: по заданной теме изучать не один, а несколько источников. Дополнительные источники могут быть найдены отдельно или из списка литературы первого.
- Де-абстрагирование: мысленное применение абстрактных математических структур и понятий к решаемой или гипотетически решаемой задаче.

Преодоление математической тревожности

Большое количество людей страдает от математической тревожности время от времени.

Математическая тревожность часто результат математического опыта. Причины могут быть очень различные: учитель, с которым Вы не ладили, некачественные занятия, кто-то убедил Вас, что математика не для Вас и пр., и это вызвало негативную реакцию.

В результате Вы стали опасаться математики. В некоторых случаях даже появляются физические симптомы, такие как учащенное сердцебиение или дыхание, напряжение мышц, расстройство желудка или повышенное потоотделение.

Часто мы усугубляем свою проблему, приобретая привычку негативно говорить с самим собой. Эти негативные мысли заставляют терять сосредоточенность - и то, и другое необходимо для успешной учебы по математике. Помните о "Маленьком механизме, который мог бы"; если мы думаем, что можем, значит, можем.

"Математические мифы" иногда влияют на наши ожидания. Если у Вас есть предубеждения, например, что математика - это сложно и скучно, математикой могут заниматься только женщины (или только мужчины, или только люди с зелёными глазами), то это сразу может определять Ваше отношение к математике в целом, ещё до того, как Вы начали курс, решение задачи или применение на практике.

Чрезмерное использование большинства вещей приводит к тому, что мы заболеваем и теряем работоспособность. С математической тревожностью связано то же самое. Это может парализовать нас и помешать нам достичь наших целей.

С другой стороны, некоторая тревожность не дает нам успокоиться. Это держит нас в напряжении, заставляет быть более бдительными и помогает нам функционировать наилучшим образом.

Наше отношение к математике часто исходит от других людей - наших родителей, сверстников, традиций (мне особенно не нравится утверждение, что мальчики лучше разбираются в математике, чем девочки). Но на этом семинаре мы попытались показать, что отношение к математике можно изменить на позитивное.

Большинство людей преуспевают в математике и постоянно ею пользуются. Математика в школе требует непредвзятости и отношения к учебе. Иногда нам приходится отказываться от "того, как мы всегда это делали", и учиться тому, как этого хотят от нас инструктор и книга, чтобы добиться успеха. Помните, что математические способности не определяют нас; это ступенька к тому, чем мы действительно хотим заниматься.

Математика - это язык, навык и искусство, и мы не можем рассчитывать на успех в любом из них без практики и упорства.

Математика для всех. Вне зависимости от формы гениталий, цвета глаз, возраста или чего-то ещё. Более того, чем дольше Вы занимаетесь

математикой, тем дольше Ваш мозг остаётся с Вами и снижаются риски развития старческих заболеваний.

ВЫ - ключ к преодолению своей ТРЕВОЖНОСТИ по поводу МАТЕМАТИКИ!

- Признайте свою тревожность по поводу математики и возьмите на себя ответственность за нее. Затем составьте план борьбы с ней.
- Будьте готовы. Приобретите хорошие привычки в учебе.
- Научитесь расслабляться. Выберите технику, которая, по вашему мнению, будет работать на вас, и используйте ее.
- Привыкайте к новому подходу. Представьте, что у вас все хорошо с математикой. ВЫ способны к математике!

Содержание занятия

- Форма для знакомства (не оценивается) ∼10min.
- Тест 1: На остаточные знания ∼10min.
- Введение ∼20min.
- Видео: Roger Antonsen 2015 ∼17min.
- Чистый код ∼30min.
- Тест 2: Чистый код ∼5 min.

Глава 6

Лекция 1. 13.02. Погрешности.

6.1 Подготовка к занятию

Основное домашнее задание:

1. video: Р. Сапольски - Хаос и Редукционизм (диск: рус / eng / youtube: рус / eng
2. video: Р. Сапольски - Эмерджентность и Сложность (диск: рус / eng / youtube: рус / eng)
3. video: Это уравнение изменит ваш взгляд на мир (диск: рус / eng / youtube: рус / eng)
4. video: What Is The Coastline Paradox (диск: рус / eng / youtube: рус / eng)

Дополнительное рекомендованное домашнее задание:

1. Книга: Хаос. Создание новой науки. Дж. Глейк (диск: рус/eng / интернет)
2. Книга: Демидович Б.П. Основы вычислительной математики. глава 1: приближённые числа (целиком) (диск / elanbook)

6.2 Основные материалы лекции

Погрешности

Приблизительное (approximate) число a является числом, но немного отличается от точного числа A, мы пишем $a \approx A$. Если $a < A$, то это приближение по недостатку (minor), если $a > A$, то это приближение по избытку (major). Итак, мы можем вычислить значение разницы между нашим приблизительным числом a и точным числом A. Это значение обозначается как Дельта (заглавная буква греческого языка) $\Delta a = A - a$. Тогда, если

мы знаем приблизительное число и ошибку, мы можем вычислить точное число как $A = a + \Delta a$. Мы добавляем индекс a для дельты, чтобы показать, что это ошибка значения a.

Существует несколько типов погрешностей: абсолютные (absolute), предельные абсолютные (limit absolute), относительные (relative) и предельные относительные (limit relative). В английском языке погрешность именуется термином error. В русской традиции переводить error как погрешность, однако термин "ошибка" также верен.

Исходное значение дельты имеет свой собственный знак, поэтому оно может быть выше или ниже нуля. Но есть много случаев использования, когда мы не знаем знака ошибки. Тогда мы можем использовать абсолютную ошибку.

Есть два основных случая: когда известно точное число A и когда оно неизвестно. В первом случае мы можем использовать формулу абсолютной погрешности. Во втором случае мы можем использовать предельную абсолютную погрешность.

Например, есть какая-то фабрика, где мы производим печенье. Мы кладем печенье в коробки по весу, а не по точному количеству печенья. Но вес печенья может немного отличаться. Если мы подсчитаем количество печенья в каждой коробке, оно может быть разным. Затем мы можем вычислить ограниченную (предельную) абсолютную погрешность, чтобы узнать количество печенья в некотором диапазоне: не больше плюс дельта $(+\Delta)$ и не меньше минус дельта $(-\Delta)$.

Другим примером здесь является вычисление предельной абсолютной погрешности некоторой неизвестной константы, такой как π. Мы знаем, что это меньше 3,15 и больше 3,14. Тогда предельная абсолютная погрешность составляет 0,01. Если мы возьмем 3,14 и 3,142 в качестве ограничений, то дельта составит 0,002.

Под предельной абсолютной погрешностью приближённого числа понимается всякое число, не меньше абсолютной погрешности этого числа.

Тогда если Δa - предельная абсолютная погрешность, то: $\Delta = |A - a| \leq \Delta a$ Что означает: $a - \Delta a \leq A \leq a + \Delta a$. Иначе говоря, предельная абсолютная погрешность задаёт интервал вокруг точного числа A, равный абсолютной погрешности с каждой стороны. Любое число, попадающее в этот интервал, будет обладать абсолютной предельной погрешностью.

Абсолютных погрешностей недостаточно для оценки качества измерений. Пример: $a_1 = 100,8 \pm 0,1$ и $a_2 = 1,8 \pm 0,1$. Существует разница между ошибкой для a_1 и для a_2.

Тогда у нас есть относительные ошибки. Они обозначаются как дельта в нижнем регистре. Относительная погрешность δ приблизительного числа a представляет собой отношение абсолютной погрешности δ числа к модулю (абсолютному значению) соответствующего точного числа $A (A \approx 0)$.

Как и в случае абсолютных ошибок, мы введем понятие ограниченной относительной ошибки. $\delta = \Delta|A| \leq \delta a$

Теперь давайте взглянем на общую картину. Итак, у нас есть 4 различных вида ошибок, которые мы можем вычислить. Абсолютный означает,

что у нас не будет никакого знака. Относительный означает, что мы разделим дельту на абсолютное значение точного числа. Предельность для обоих случаев означает, что у нас есть диапазон ошибок, который не может быть больше или меньше некоторых значений.

Общая формула ошибок

Основная задача теории погрешности заключается в следующем: известны погрешности некоторой системы величин, требуется определить погрешность данной функции от этих величин.

Например, мы измерили две стены в комнате разными рулетками. Погрешность измерений каждой стены отличается. Теперь нам интересно с какой погрешностью мы найдем площадь комнаты.

Альтернативным примером можно привести ситуацию, когда противоположные стены в прямоугольной комнате немного не равны. Из расчёта разницы в измерениях противоположных стен вычисляется среднее и заданная погрешность для неё. Так при вычислении площади комнаты также возникает необходимость в том, чтобы узнать погрешность. Особый интерес на практике могут представлять задачи, в которых углы комнаты отличаются от 90°, так как при вычислении площади для нахождения, например, необходимого количества ламината, необходимо будет пользоваться интегралами, а при расчёте необходимого количества плитки – двойными интегралами.

Предположим, задана дифференцируемая функция $u = f(x_1, x_2, \ldots, x_n)$ и пусть $|\Delta x_i|(i = 1, 2, \ldots, n)$ - абсолютные ошибки аргументов функции. Теперь построим формулы для 4 различных видов погрешностей для функции u.

Тогда абсолютная погрешность функции — это вычитание функции, в которой у нас были все ошибки в каждом аргументе и функции без него. Это справедливо, потому что погрешности аргументов абсолютны и не имеют знака.

$$|\Delta u| = |f(x_1 + \Delta x_1, x_2 + \Delta x_2, \ldots, x_n + \Delta x_n) - f(x_1, x_2, \ldots, x_n)|$$

В практических ситуациях $|\Delta x_i|$ обычно являются малыми величинами, произведениями, квадратами и более высокими степенями которых можно пренебречь. И мы можем получить оценку погрешности функции – предельную абсолютную погрешность функции u. Здесь мы используем частную производную функции u по аргументу x_i, чтобы оценить влияние этого аргумента на изменение значения функции, после чего умножаем на значение погрешности этого аргумента. А сумма всех таких выражений для всех аргументов функции u даст нам интервал, в котором может изменяться функция так, что погрешность будет не больше абсолютной погрешности.

$$|\Delta u| \leq \sum_{i=1}^{n} |\frac{\partial u}{\partial x_i}|\Delta x_i$$

Разделив обе части неравенства на u, будем иметь оценку для относительной погрешности функции u:

$$\delta \leq \sum_{i=1}^{n} |\frac{\frac{\partial f}{\partial x_i}}{u}|\Delta x_i| = \sum_{i=1}^{n} |\frac{\partial}{\partial x_i} ln f(x_1, \ldots, x_n)||\Delta x_i|$$

Разделив обе части неравенства на u, мы получаем предельную относительную погрешность функции u:

$$\delta u = \sum_{i=1}^{n} \left| \frac{\partial}{\partial x_i} \ln u \right|)\Delta x_i$$

Обратная задача теории погрешностей

На практике важна также обратная задача: каковы должны быть абсолютные погрешности аргументов функции, чтобы абсолютная погрешность функции не превышала заданной величины? Задача математически не определена, так как заданную предельную погрешность Δu функции $u = f(x_1, x_2, \ldots, x_n)$ можно обеспечить, устанавливая по-разному предельные абсолютные погрешности Δx_i её аргументов.

В англоязычной литературе такие исследования называются discrepancy theory, а распределение значения ошибки обратно на значения аргументов называется distribute discrepancies.

Простейшее решение обратной задачи даётся принципом равных влияний (naïve). Согласно этому принципу, предполагается, что все частные дифференциалы $\frac{\partial f}{\partial x_i}\Delta x_i (i = 1, 2, \ldots, n)$ одинаково влияют на образование абсолютной погрешности Δu функции $u = f(x_1, x_2, \ldots, x_n)$. Пусть величина предельной абсолютной погрешности Δu задана. Тогда:

$$|\Delta u| \leq \sum_{i=1}^{n} |\frac{\partial u}{\partial x_i}|\Delta x_i$$

. Полагая, что все слагаемые между собой равны (т.е. аргументы имеют равное влияние на функцию), будем иметь:

$$|\frac{\partial u}{\partial x_1}|\Delta x_1 = |\frac{\partial u}{\partial x_2}|\Delta x_2 = \cdots = |\frac{\partial u}{\partial x_n}|\Delta x_n$$

отсюда

$$\Delta x_i = \frac{\partial u}{n|\frac{\partial u}{\partial x_1}|} (i = 1, 2, \ldots, n)$$

Приведем пример. Имеется формула $A = B + C$, однако числа B и C приходят к нам из одного источника, а A тоже приходит из внешнего источника. В идеальном случае формула должна остаться верной при

проверке, однако на практике такое бывает не всегда. Например, бухгалтерский расчёт или измерения. О причинах возникновения таких ситуаций мы будем говорить в разделе Источники погрешностей. Предположим, что в какой-то момент времени $B = 15$, $C = 20$, но $A = 39$. Подставив все числа в формулу, видим, что оно не верно и разница между левой и правой части равно 4: $A-(B+C) = 39-(15+20) = 4$. Тогда используя принцип равных влияний можем записать, что B и C на самом деле должны иметь другие числа: $B = 15+42 = 17$ и $C = 20+42 = 22$, тогда исходная формула снова станет верной: $A - (B + C) = 39 - (17 + 22) = 0$. По сути мы рассчитали разницу между левой и правой частью уравнения, а затем распределили значения, чтобы формула обратилась в тождество. Здесь мы распределили значения по правой части, т.к. в исходная формула – это функция из двух аргументов: $A = f(B, C) = B + C$.

Значащие цифры и верные знаки

Полагаю, что Вам известно, что, любое положительное число может быть представлено в виде конечной или бесконечной десятичной дроби, как в формуле ниже, где α – цифры числа $a(\alpha_i = 0, 1, \ldots, 9)$, начальная цифра $\alpha_m \neq 0$ и m является целым числом.

$$a = \alpha_m 10^m + \alpha_{m-1}10^{m-1} + \alpha_{m-2}10^{m-2} + \cdots + \alpha_{m-n+1}10^{m-n+1}$$

Например:

$$3141.59\cdots = 3 * 10^3 + 1 * 10^2 + 4 * 10^1 + 1 * 10^0 + 5 * 10^{-1} + 9 * 10^{-2} + \ldots$$

На практике преимущественно приходится иметь дело с приближёнными числами, представляющими собой конечные десятичные дроби: На практике преимущественно приходится иметь дело с приближёнными числами, представляющими собой конечные десятичные дроби:

$$b = \beta_m 10^m + \beta_{m-1}10^m - 1 + \beta_{m-2}10^{m-2} + \cdots + \beta_{m-n+1}10^{m-n+1}, (\beta_m \neq 0)$$

Например:

$$b = 7 * 10^{-3} + 0 * 10^{-4} + 1 * 10^{-5} + 0 * 10^{-6} = 0.007010$$

$$b = 2 * 10^9 + 0 * 10^8 + 0 * 10^7 + 3 * 10^6 + 0 * 10^5 = 2,003,000,000$$

Значащей цифрой приближённого числа является всякая цифра в его десятичном отображении, отличная от нуля, и ноль, если он содержится между значащими цифрами или является представителем сохранённого десятичного разряда. Все остальные нули, входящие в состав десятичного

числа и служащие лишь для обозначения десятичных разрядов его, не причисляются к значащим цифрам.

Рассмотрим пример: 0.002080. Здесь первые 3 нуля необходимы только для того, чтобы в десятичном виде записать остальные цифры. Цифры 2 и 8 отличны от нуля, поэтому они являются значащими цифрами. Между ними стоит 0, который тоже будет являться значащей цифрой, т.к. стоит между двумя значащими цифрами. Является ли крайний правый ноль значащей цифрой зависит от условий: если там известно, что число верно с точностью до этого разряда, то он является значащей цифрой. Если мы приписали его просто так и мы можем его отбросить, то он не является значащей цифрой. Числа 0.002080 и 0.00208 не равны, т.к. в первом случае ноль может выступать для обозначения точности, а во втором случае может быть не известно значение этого разряда. Таким образом получим следующее число: 0.002080, где цифры, выделенные цветом, являются значащими (при условии, что крайний правый ноль является представителем сохранённого десятичного разряда, т.е. показывает точность числа), а все остальные цифры являются не значащими.

Говорят, что n первых значащих цифр (десятичных знаков) приближённого числа являются верными, если абсолютная погрешность этого числа не превышает половины единицы разряда, выражаемого n-й значащей цифрой, считая слева направо. Таким образом, если для приближённого числа a, заменяющего точное число A известно, что:

$$\Delta = |A - a| \leq \frac{1}{2} * 10^{m-n+1}$$

, то по определению n первых знаков $\alpha_m, \alpha_{m-1}, \ldots, \alpha_{m-n+1}$ являются верными. Пример: точное число $A = 35.97$, приближённое число $a = 36.00$ является приближённым с тремя знаками, так:

$$|A - a| = 0.03 < \frac{1}{2} * 0.1$$

Правила округления

В школе у Вас наверняка были правила округления, однако, правила, приводимые в школе не в полной мере справедливы для инженерных и прикладных расчётов. Приведем расширенные правила округления.

Чтобы округлить число до n значащих знаков, отбрасывают все цифры справа от n-й значащей цифры или, если необходимо для сохранения разрядов, заменяют их нулями. При этом (обозначим первую отброшенную цифру как α_{n+1}:

1. Если $\alpha_{n+1} < 5$, оставляем оставшиеся цифры без изменения.
2. Если $\alpha_{n+1} > 5$, добавляем 1 к последней оставшейся цифре.
3. Если $\alpha_{n+1} = 5$ и есть ненулевая цифра среди отброшенных, добавить единицу к последней оставленной цифре.

4. Однако, если $\alpha_{n+1} = 5$ и все другие отброшенные цифры являются нулями, то единицу не добавляют, если последняя оставшаяся цифра четная и добавляют единицу, если она нечетная (правило чётной цифры).

Обычно в школе используют первые два правила, возможно правила 3 и 4 будут для Вас новыми. Рассмотрим их на примере. Возьмём число $0,0002540$ и попробуем его округлить до 4 знака после запятой. По правилу смотрим в 5й разряд после запятой: $\alpha_5 = 5$. В связи с тем, что выполнилось условие равенства числа, следующего за округляемым, 5, необходимо посмотреть на все оставшиеся справа цифры: 4 и 0. Соответственно выполнилось условие 3 и к $\alpha_4 = 2$ необходимо добавить 1. Тогда получим округлённое число: $0,0003$. Ровно также 3 правило выполнилось бы для числа $0,0002501$.

Аналогично для числа $0,0002500$ выполнится не 3 правило, а 4, потому что среди всех отбрасываемых цифр нет цифр, отличных от 0. Тогда необходимо посмотреть на чётность значения в последнем, сохраняемом разряде: $\alpha_4 = 2$. Из-за того, что 2 является чётным числом, то по 4 правилу, 1 не будет добавляться, поэтому округлённое число будет записано: $0,0002$. Однако, для числа $0,0001500$ по 4 правилу будет добавлена 1, т.к. $\alpha_4 = 1$ является нечётным числом. Поэтому округляя получим: $0,0002$.

Источники погрешностей

Ошибки, с которыми приходится сталкиваться в математических задачах, в основном можно разделить на 6 групп.

1. **Погрешности, связанные с постановкой проблемы**. Математические утверждения редко дают точную картину реальных явлений. По большей части это всего лишь идеализированные модели. При изучении явлений природы мы, как правило, ориентируемся на принятие определенных условий, упрощающих рассматриваемую проблему. Это источник ошибок (ошибок проблемы).

2. Иногда бывает так, что решить ту или иную задачу, если она точно сформулирована, либо трудно, либо даже невозможно. Если это так, то она заменяется приближенной задачей, дающей почти те же результаты. Это источник ошибки, называемой **ошибкой метода**.

3. **Остаточные погрешности** - ошибки, вытекающие из наличия бесконечных процессов в математическом анализе. Функции, включенные в математические формулы, часто задаются в виде бесконечных последовательностей или рядов. Более того, многие математические уравнения могут быть решены только с помощью желаемых решений. Поскольку, вообще говоря, бесконечный процесс не может быть завершен за конечное число шагов, мы сосредоточены на том, чтобы

остановиться на некотором члене последовательности и рассматривать его как приближение к требуемому решению. Естественно, такое у нас называется остаточной ошибкой.

4. Ошибки, вызванные числовыми параметрами (в формулах), значения которых могут быть определены только приблизительно. Таковы, например, все физические константы. Давайте договоримся называть эту ошибку **начальной ошибкой**.

5. **Погрешности округления** - ошибки, связанные с системой счисления. При изображении четных рациональных чисел в десятичной системе или какой-либо другой позиционной системе они могут представлять собой бесконечное количество цифр справа от десятичной точки (как правило, точка основания). Например, у нас может быть бесконечно повторяющаяся десятичная дробь. Очевидно, что в наших вычислениях мы можем использовать только конечное число цифр. Это является источником так называемых ошибок округления.

6. **Погрешности действий**. Ошибки, вызванные операциями, связанными с операциями с приблизительными числами (ошибки в работе). Когда мы выполняем вычисления с приблизительными числами, мы естественным образом переносим (в некоторой степени) ошибки исходных данных в конечные данные. В этом отношении присущи ошибки эксплуатации.

Вполне естественно, что в конкретной задаче некоторые ошибки отсутствуют, а другие оказывают незначительное влияние. Но, как правило, полный анализ должен включать все типы ошибок. В дальнейшем мы в основном ограничимся вычислительными ошибками в работе и ошибками метода.

Теория хаоса и недетерминированность

Говоря о погрешностях, приближённых числах и округлениях, было бы несправедливо пропустить теорию хаоса. Подробнее о теории хаоса и о том, как она зарождалась, можно прочитать в книге Джеймса Глейка «Хаос. Создание новой науки». Эта книга также есть в списке рекомендованной литературы. Приведём здесь отрывок из этой книги. Книга начинается с описания работы Эдварда Нортона Лоренца, который был математиком и метеорологом. В 1960 году он создал компьютерную программу для прогнозирования погоды, и она работала.

«С помощью своего примитивного компьютера Лоренц свел погоду к самому простому скелету. И все же, строка за строкой, ветры и температуры в распечатках Лоренца, казалось, вели себя узнаваемым земным образом. Они соответствовали его заветной интуиции относительно погоды, его чувству, что она повторяется, демонстрируя знакомые закономерности с течением времени, повышение и понижение давления, направление воздушного потока на север и юг. Он обнаружил, что, когда линия идет от

максимума к минимуму без выпуклости, следующим будет двойной выпуклость, и он сказал: “Это такое правило, которое мог бы использовать прогнозист”. Но повторения никогда не были вполне точными. Там была закономерность, с нарушениями. Упорядоченный беспорядок.

Чтобы сделать узоры понятными, Лоренц создал примитивный вид графики. Вместо того, чтобы просто печатать обычные строки цифр, он заставил бы машину напечатать определенное количество пробелов, за которыми следует буква a. Он выбрал бы одну переменную — возможно, направление воздушного потока. Постепенно буквы "a"спускались по рулону бумаги, раскачиваясь взад и вперед волнистой линией, образуя длинную серию холмов и долин, которые представляли, как западный ветер будет дуть на север и юг по всему континенту. Упорядоченность этого, узнаваемые циклы, повторяющиеся снова и снова, но никогда дважды одним и тем же способом, обладали гипнотическим очарованием. Система, казалось, медленно раскрывала свои секреты глазу прогнозиста.

Однажды зимой 1961 года, желая более подробно изучить одну последовательность, Лоренц выбрал короткий путь. Вместо того чтобы начать весь пробег заново, он начал с середины. Чтобы задать машине начальные условия, он набрал цифры прямо из предыдущей распечатки. Затем он прошел по коридору, чтобы уйти от шума и выпить чашечку кофе. Когда он вернулся через час, то увидел нечто неожиданное, нечто такое, что посеяло семена новой науки.

Этот новый прогон должен был в точности повторить старый. Лоренц сам скопировал цифры в машину. Программа не изменилась. Тем не менее, когда он уставился на новую распечатку, Лоренц увидел, что его погода так быстро отличается от модели последнего прогона, что всего за несколько месяцев все сходство исчезло. Он посмотрел на один набор цифр, затем снова на другой. С таким же успехом он мог бы выбрать две случайные погоды из шляпы. Его первой мыслью было, что испортилась еще одна вакуумная трубка.

Внезапно он осознал правду. Никакой неисправности не было. Проблема заключалась в набранных им цифрах. В памяти компьютера хранилось шесть знаков после запятой: .506127. На распечатке, чтобы сэкономить место, появилось всего три: .506. Лоренц ввел более короткие, округленные числа, предполагая, что разница — одна часть из тысячи — несущественна.

Этот момент стал отправной точкой новой эры наук и новой науки самой по себе. Это основа теории хаоса.»

Стало известно, что существует огромный пласт проблем, когда правильное значение никогда не может быть найдено. Независимо от того, насколько точно вы его измеряете. На рисунке ниже вы можете увидеть странный аттрактор Лоренца, который показывает, что значение некоторого параметра изменяет некоторое значение в некотором диапазоне, но никогда не проходит по одной и той же линии дважды.

Справа на рисунке вы можете увидеть каскад бифуркации, который также известен как сценарий удвоения периода. Эта картина означает, что существует какой-то сценарий, но в какой-то момент поведение системы

немного изменилось, и после этого мы имеем как бы две версии системы внутри одной системы.

Как вы можете видеть, существует много проблем с точным вычислением, поэтому мы должны уметь работать с приблизительными методами и приблизительными значениями, когда мы вычисляем некоторую математику.

Глава 7

Лабораторное занятие 2. 18.02. Погрешности, неосоответствия, теория хаоса

7.1 Подготовка к занятию

7.2 Основные материалы занятия

На занятии речь будет идти про погрешности, теорию неравенства, а также немного затронем теорию хаоса и фракталы.

Глава 8

Лекция 2. 20.02. Аппроксимация и интерполяция.

8.1 Подготовка к занятию

- Видео: Dear Calculus 2 Students, This is why you're learning Taylor Series `https://youtu.be/eX1hvWxmJVE`

- Видео: The Continuity of Splines `https://youtu.be/jvPPXbo87ds`

- Giving Personality to Procedural Animations using Math `https://youtu.be/KPoeNZZ6H4s`

8.2 Основные материалы занятия

Глава 9

Лабораторное занятие 3 25.02.

9.1 Подготовка к занятию

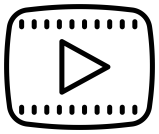
9.2 Основные материалы занятия

Глава 10

Лекция 3. 27.02. Решение СЛАУ.

10.1 Подготовка к занятию

- Видео: Плейлист видео с русскими субтитрами про линейную алгебру `https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab`



- Книга: Демидович Б.П. Основы вычислительной математики. Глава 7. Алгебра матриц (целиком)

- Jupyter Guide to Linear Algebra: `https://bvanderlei.github.io/jupyter-guide-to-linear-algebra/intro.html`

10.2 Основные материалы занятия

Системы линейных алгебраических уравнений

Двумя фундаментальными проблемами численного анализа матриц являются решения линейных систем уравнений и вычисление собственных значений и собственных векторов матрицы.

Зачем изучать решение СЛАУ? Хотя бы потому, что они используются практически везде. Например, при работе с изображениями, анимацией и расчётом расположения объектов (например, в играх), в алгоритмах на графах, в анализе данных и статистике, в теории нейронных сетей, в поисковых машинах (компания Google была основана в результате разработки алгоритма PageRank, в основе которого лежит решение СЛАУ, хотя и весьма необычным способом) и во многих других областях.

Системой n уравнений с m неизвестных $x_1, x_2, \ldots, x_m$ принадлежащими заданному числовому множеству M называется задача о нахождении упорядоченных совокупностей из m чисел $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ принадлежащих множеству M, которые при подстановке вместо соответствующих неизвестных обращают каждое из n данных уравнений в тождество.

Общий вид системы уравнений можно записать следующим образом:

$$\begin{cases} F_1(x_1, x_2, \ldots, x_m) = \Phi_1(x_1, x_2, \ldots, x_m) \\ F_2(x_1, x_2, \ldots, x_m) = \Phi_2(x_1, x_2, \ldots, x_m) \\ \ldots \\ F_n(x_1, x_2, \ldots, x_m) = \Phi_n(x_1, x_2, \ldots, x_m) \end{cases}$$

С каждой стороны уравнения одинаковое количество неизвестных. Когда мы говорим про линейные алгебраические уравнения, функции F обычно представляются в линейной форме.

Решение системы уравнений – это упорядоченная совокупность чисел $\alpha_1, \alpha_2, \ldots, \alpha_m$, каждое из которых принадлежит множеству M, в котором рассматривается система, при подстановке которых в место соответствующих неизвестных каждое уравнение системы обращается в тождество.

Соответственно, решить систему уравнений - означает найти множество всех её решений или показать, что она решений не имеет.

Решение линейного алгебраического уравнения, не имеющего решений, определяется матрицей такой системы, равной нулю. Как уже говорилось ранее, очень полезно уметь вычислять определитель.

Решение линейного алгебраического уравнения, не имеющего решений, определяется матрицей такой системы, равной нулю. Как уже говорилось ранее, очень полезно уметь вычислять определитель.

Простые методы решения СЛАУ с 2 неизвестными

Простейшим случаем систем линейных алгебраических уравнений является система, состоящая из двух уравнений и двух неизвестных. Запишем общий вид системы в современных терминах линейной алгебры:

$$\begin{cases} a_1 x + b_1 y = c_1, \\ a_2 x + b_2 y = c_2, \end{cases}$$

где x и y – неизвестные; а $a_1,\ a_2,\ b_1,\ b_2$ - коэффициенты; $c_1,\ c_2$ - столбец свободных членов (свободными членами системы).

Метод подстановки

Из общего представления мы можем исключить y и заменить его следующим результатом преобразования второго уравнения. В большинстве случаев это будет приблизительное число.

Из общего вида СЛАУ с 2 неизвестными при $b_1 \neq 0$ мы находим:

$$y = \frac{c_1 - a_1 x}{b_1}$$

Подставляем найденное значение y во второе уравнение:

$$a_2 x + b_2 \frac{c_1 - a_1 x}{b_1} = c_2$$

Откуда получаем: $(a_1 b_2 - a_2 b_1)\, x = c_1 b_2 - c_2 b_1$. Если $(a_1 b_2 - a_2 b_1) \neq 0$, то:

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1}$$

Тогда:

$$y = \frac{c_1}{b_1} - \frac{a_1}{b_1} x = \frac{c_1}{b_1} - \frac{a_1 (c_1 b_2 - c_2 b_1)}{b_1 (a_1 b_2 - a_2 b_1)} = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

Метод алгебраического сложения

Для решения системы (2), умножим обе части первого уравнения $b_2 \neq 0$, а второго – на $-b_1 \neq 0$:

$$\begin{cases} a_1 b_2 x + b_1 b_2 y = c_1 b_2, \\ -a_2 b_1 x + b_1 b_2 y = -c_2 b_1, \end{cases}$$

Полагая, что система имеет решение, складываем левые и правые части уравнений:

$$(a_1 b_2 - a_2 b_1)\, x = c_1 b_2 - c_2 b_1,$$

Откуда при $a_1 b_2 - a_2 b_1 \neq 0$ находим:

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1}$$

Аналогично поступаем, чтобы найти y. Умножаем обе части первого уравнения на $-a_2 \neq 0$, а второго – на $a_1 \neq 0$:

$$\begin{cases} -a_1 a_2 x - a_2 b_1 y = -a_2 c_1, \\ a_1 a_2 x + a_1 b_2 y = -a_1 c_2. \end{cases}$$

Складываем левые и правые части уравнений:

$$(a_1 b_2 - a_2 b_1)\, y = a_1 c_2 - a_2 c_1$$

$$y = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

Метод сравнения

Чтобы решить систему (2) предполагаем, что эта система имеет решение, и из каждого уравнения системы находим y (или x):

$$y = \frac{c_1 - a_1 x}{b_1},\ y = \frac{c_2 - a_2 x}{b_2},\ b_1 b_2 \neq 0$$

Приравнивая полученные для y выражения, получаем уравнение относительно неизвестного x. Это уравнение называется выводным.

Если $a_1 b_2 - a_2 b_1 \neq 0$, то:

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1}$$

Подставляя найденное значение x в какое-либо выражение для y (например, первое), получаем:

$$y = \frac{c_1 - a_1 \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1}}{b_1} = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

Прямые методы

Более сложные методы решения систем линейных алгебраических уравнений можно разделить на две основные категории: первая – это прямые методы, а вторая - итерационные методы. Вы должны знать, что в каждой категории существует множество методов, но в ходе этого курса мы сосредоточим наше внимание на четырех методах: по два для каждой категории. Ошибочно будет считать, что категорий методов всего 2 или что методов всего 4. Например, интересной группой метода являются проекционные методы и методы Крыловского типа. Также для решения СЛАУ может быть использован метод Холецкого, который будет рассмотрен далее.

Прямые методы означают, что вычисление корней системы является конечным алгоритмом. Итерационные методы позволяют нам получать корни системы с заданной точностью путем сходящегося бесконечного процесса. В этом источнике мы обратим наше внимание на метод исключения Гаусса и метод исключения Гаусса с поворотом, а среди итерационных методов мы увидим метод простых взаимодействий и метод Гаусса-Зейделя.

Давайте представим, что у нас есть система линейных алгебраических уравнений, подобная этой. Теперь мы можем взять известные коэффициенты из левой части и использовать их в качестве элементов матрицы A. Правая часть системы может рассматриваться как матрица B. Элементы правой части называются свободными членами системы.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2, \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3, \\ \end{cases}$$

$$A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$B = \begin{vmatrix} b_1 \\ b_2 \\ b_3 \end{vmatrix}$$

Оба прямых метода основаны на исключении во время прямого хода прямом и замене при обратном ходе. Они имеют треугольную матрицу после прямого исключения. После всех таких операций мы можем вычислить определитель исходной матрицы по треугольной матрице, поскольку использовались только элементарные преобразования. Чтобы найти определитель треугольной матрицы, можно умножить элементы лишь на главной диагонали. Подробнее об определителе матрицы и его свойствах говорилось в предыдущем разделе.

Невязка / Residual error Очень полезно вычислять остаточные ошибки (невязки) после использования прямых методов. Это помогает понять меру ошибки ваших расчетных значений неизвестного.

Невязка – это разница между левой и правой частями уравнения. Теоретически оно должно быть равно 0 (по определению уравнения, т. е. равенства). Но на практике, после большого количества операций, таких как деление, например, в случае ограниченного компьютерного представления чисел и с приблизительными числами в результате, результирующий вектор неизвестного может иметь значительную разницу.

Чтобы вычислить невязку, мы помещаем вычисленное неизвестное исходной системе в каждое уравнение, а затем вычитаем одну сторону из другой.

$$\begin{cases} a_1x + b_1y = c_1, \\ a_2x + b_2y = c_2, \end{cases} \Rightarrow \begin{cases} |a_1x + b_1y - c_1| = r_1, \\ |a_2x + b_2y - c_2| = r_2, \end{cases}$$

Таким образом, у получаем вектор остаточных ошибок (невязок) для каждого уравнения (не для каждого неизвестного!).

Метод Гаусса / Gauss Elimination Метод назван в честь Карла Фридриха Гаусса (1777-1855), хотя некоторые частные случаи этого метода - хотя и представленные без доказательств - были известны китайским математикам еще около 179 года нашей эры.

Основная идея метода исключения Гаусса заключается в нахождении треугольной матрицы. Этот метод состоит из прямого хода (исключения) и обратного хода (замены, подстановки). Необходимым и достаточным условием для этих методов является неравенство нулю ведущих элементов (элементов на главной диагонали).

Покажем на системе с 4 неизвестными при $a_{11} \neq 0$:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2, \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3, \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4. \end{cases}$$

Разделим элементы системы (3) на ведущий элемент a_{11}, тогда:

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 + \frac{a_{14}}{a_{11}}x_4 = \frac{b_1}{a_{11}}$$

Тогда можем исключить неизвестную x_1 из системы (3). Чтобы сделать это, мы должны из второго уравнения системы вычесть уравнение (4) умноженное на a_{21} из третьего уравнения вычесть уравнение (3) умноженное на a_{31} и т. д. Получим:

$$\begin{cases} a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + a_{24}^{(1)}x_4 = b_2^{(1)}, \\ a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + a_{34}^{(1)}x_4 = b_3^{(1)}, \\ a_{42}^{(1)}x_2 + a_{43}^{(1)}x_3 + a_{44}^{(1)}x_4 = b_4^{(1)}. \end{cases}$$

Где коэффициенты $a_{ij}^{(1)}(i, j{\geq}2)$ рассчитываются $a_{ij}^{(1)} = a_{ij} - a_{i1}\frac{a_{1j}}{a_{11}}$ $(i, j{\geq}2)$.

Далее: делим уравнения системы (5) на ведущий элемент $a_{22}^{(1)}$:

$$x_2 + \frac{a_{23}^{(1)}}{a_{22}^{(1)}}x_3 + \frac{a_{24}^{(1)}}{a_{22}^{(1)}}x_4 = \frac{b_2^{(1)}}{a_{22}^{(1)}}$$

Исключаем x_2 из системы таким же образом и получаем:

$$\begin{cases} a_{33}^{(2)}x_3 + a_{34}^{(2)}x_4 = b_3^{(2)} \\ a_{43}^{(2)}x_3 + a_{44}^{(2)}x_4 = b_4^{(2)} \end{cases}$$

где $a_{ij}^{(2)} = a_{ij}^{(1)} - a_{i2}^{(1)}\frac{a_{2j}^{(1)}}{a_{22}^{(1)}}$

Делим коэффициенты системы (7) на ведущий элемент $a_{33}^{(2)}$: $x_3 + \frac{a_{34}^{(2)}}{a_{33}^{(2)}}x_4 = \frac{b_3^{(2)}}{a_{33}^{(2)}}$.

Исключаем аналогично x_3: $a_{44}^{(3)}x_4 = b_4^{(3)}$. Где $a_{ij}^{(3)} = a_{ij}^{(2)} - a_{i3}^{(2)}\frac{a_{3j}^{(2)}}{a_{33}^{(2)}}$. Тогда: $x_4 = \frac{b_4^{(3)}}{a_{44}^{(3)}}$

Остальные неизвестные последовательно выводятся обратной подстановкой:

$$x_3 = \frac{b_3^{(2)}}{a_{33}^{(2)}} - \frac{a_{34}^{(2)}}{a_{33}^{(2)}}x_4$$

$$x_2 = \frac{b_2^{(1)}}{a_{22}^{(1)}} - \frac{a_{24}^{(1)}}{a_{22}^{(1)}}x_4 - \frac{a_{23}^{(1)}}{a_{22}^{(1)}}x_3$$

$$x_1 = \frac{b_1}{a_{11}} - \frac{a_{14}}{a_{11}}x_4 - \frac{a_{13}}{a_{11}}x_3 - \frac{a_{12}}{a_{11}}x_2.$$

Теперь приведем ряд достоинств и недостатков, присущих методу Гаусса. Об их справедливости и прочих достоинствах и недостатках метода предлагается порассуждать самостоятельно в ходе Лабораторной работы 2 и в ходе анализа запусков реализованного метода на различных наборах данных.

Достоинства:

- Может быть применен в любом случае, если у системы есть решение и на главной диагонали нет 0 элементов.
- Нет дополнительных подготовительных вычислений.
- Результаты могут быть получены за конечное количество операций.
- Низкая погрешность метода, если производить расчёты в последний момент (например, при вычислении вручную).

Недостатки:

- При вычислении на компьютере образуется значительная набегающая погрешность за счёт ограниченности разрядной сетки, особенно при работе с разреженными матрицами (коэффициенты близки к 0).
- Ошибки накапливаются, т. к. расчёт производится последовательно.
- Может занимать существенный объем памяти для больших матриц, так как всю матрицу нужно держать в памяти.
- Алгоритмическая сложность метода $O\left(n^3\right)$, где n – число неизвестных.

Метод Гаусса с выбором главного элемента / Gauss Elimination through pivoting Требование $a_{ii} \neq 0$ заменяется более жестким: из всех оставшихся в i-м столбце элементов нужно выбрать наибольший по модулю и переставить уравнения так, чтобы этот элемент оказался на месте элемента a_{ii};

Существует три вариации метода:

1. Искать максимальный элемент в **столбце** и переставлять **строки** так, чтобы максимальный элемент оказался ведущим элементом;
2. Искать максимальный элемент в **строке** (уравнении) и переставлять **столбцы**;
3. Искать максимальный элемент **во всей матрице** и производить перестановки и **строк, и столбцов**.

Первая и вторая вариации метода называются в англоязычной литературе partial pivoting, а третья – complete pivoting.

Выберем ненулевой, как правило, наибольший по модулю, не принадлежащий к столбцу свободных членов a_{pq}, $(q \neq n+1)$ из матрицы M, и вычислим множители:

$$m_i = \frac{-a_{iq}}{a_{pq}}$$

для всех $i \neq p$

Строка с номером p матрицы M, содержащая главный элемент, называется главной строкой.

Далее, произведем следующую операцию: к каждой не главной строке прибавим главную
строку, умноженную на соответствующий множитель m_i для этой строки. В результате мы
получим новую матрицу, у которой q-й столбец состоит из нулей. Отбрасывая этот столбец и
главную p-ю строку, получим новую матрицу $M(1)$ с меньшим на единицу числом строк и столбцов.

Это стратегия перестановки строк.

Теперь приведем ряд достоинств и недостатков, присущих методу Гаусса с выбором главного элемента. Об их справедливости и прочих достоинствах и недостатках метода предлагается порассуждать самостоятельно в ходе Лабораторной работы 2 и в ходе анализа запусков реализованного метода на различных наборах данных.

Достоинства:

- Найдёт решение, если оно существует.
- **Лучше работает с разреженными матрицами, потому что делим на максимальный элемент, а не на любой диагональный.**
- Решение найдется за конечное число операций.
- Ниже погрешность метода, если выполнять операции в последний момент, например при вычислении вручную.

Недостатки:

- При вычислении на компьютере образуется значительная набегающая погрешность за счёт ограниченности разрядной сетки, особенно при работе с разреженными матрицами (коэффициенты близки к 0). **Однако, ошибка ниже, чем у метода Гаусса.**
- Ошибки накапливаются, т. к. расчёт производится последовательно.
- Может занимать существенный объем памяти для больших матриц, так как всю матрицу нужно держать в памяти.

- Алгоритмическая сложность метода $O\left(n^3\right)$, где n – число неизвестных.

Итерационные методы

Итерационные методы основываются на концепции последовательных приближений. Для их работы необходимо задать начальные значения неизвестных, которые будут использованы на самой первой итерации для расчёта новых приближенных значений. Существуют различные способы выбора начальных приближений:

- $$0;$$
- $$b_i;$$
- некоторое предварительно рассчитанное значение (например, из прямых методов) для повышения точности результата;
- любое другое значение (например, стратегия выбора случайных значений весов для итерационных формул часто используется в нейронных сетях и при решении нелинейных уравнений и систем, как при градиентном спуске).

Однако, итерационный процесс основывается на предположении, что существует предельное значение итерационной формулы. Итерационная формула – это приближённая формула, которой замещается исходное уравнение. Иначе говоря, использование итерационной формулы не гарантирует нахождение решения системы, даже если оно существует, потому что предела итерационной формулы может для этого случая не существовать. По сути, итерационная формула – это генератор, который выдаёт одно число за другим. Если у последовательности значений такого ряда нет предела, то говорят, что итерационный процесс расходится.

Предел числовой последовательности

Напомним условие существования предела числовой последовательности:

$$\forall \varepsilon > 0,\ \exists N\ n > N\ \ \rightarrow |x_n - L| < \varepsilon$$

Итерационный процесс не имеет смысла, если он не сходится. Невозможно получить правильных корней уравнений, если не существует предел: для любого наперед заданного ε большего 0 существует натуральное число (номер) n, такое что модуль разницы между элементом числовой последовательности с номером n и значением предела L меньше, чем ε. Иначе говоря, если предел L существует, то существует окрестность (можно изобразить как окружность) предела с радиусом в любое число ε и рано

или поздно мы сможем отыскать такой номер элемента последовательности, что в эту окрестность попадём. Ещё проще это можно объяснить так: каждое следующее значение последовательности должно быть ближе к пределу, чем предыдущее (иметь меньший модуль разницы).

Тут читателю предлагается сравнить это определение с предельной абсолютной погрешностью числа и предельной относительной погрешностью числа, чтобы убедиться, что при использовании итерационного метода предполагается на каждом шаге уменьшать имеющуюся погрешность, приближаясь к точным значениям корней. Необходимо отметить, что в отличии от прямых методов, итерационные методы не являются точными, т. е. все корни, найденные с их помощью, будут приближёнными. Однако, преимущество итерационных методов заключается в том, что величиной погрешности можно управлять, задавая различные значения ε.

Условие окончания итерационного процесса за конечное число операций:

$$\forall i\,(i = 1, 2, \ldots, n)\ \ \forall \varepsilon > 0\ \ : \left| x_i^{[j]} - x_i^{[j-1]} \right| \leq \varepsilon$$

Так как значение предела неизвестно (иначе не было бы необходимости решать уравнение), в ходе выполнения итерационного метода традиционно сравнивают с ε разницу между двумя значениями в последовательности. Если разница между ними меньше, чем ε, то итерационный процесс заканчивается, т. к. дальнейшее продолжение вычислений не изменит значение больше чем на ε и не целесообразно.

Количество итераций (вычислений новых значений по итерационной формуле с подстановкой предыдущего или начального значений) зависит от начального приближения и заданной величины ε. Чем дальше от предельного значения находится начальное приближение, тем больше потребуется итераций. Чем меньше величина ε (максимальный модуль разницы между парой чисел в последовательности), тем больше потребуется приближений (итераций).

Метод простых итераций / Stationary iterative method

Метод простой итерации заключается в том, чтобы из каждого уравнения системы выразить значение неизвестного и подставляя значения предыдущего шага или начального приближения рассчитывать новые значения неизвестных. В виде формулы это записывается следующим образом:

$$x_i^{[j+1]} = x_i^{[j]} - \frac{1}{a_{ii}} f_i^{[j]} = x_i^{[j]} - \frac{1}{a_{ii}} \left(\sum_{k=1}^{n} a_{ik} x_k^{[j]} - b_i \right)$$

В записанной формуле выше j – это номер итерации.

Для метода простой итерации итерационный процесс сходится тогда, когда выполняется условие диагонального преобладания:

$$|a_{ii}| \geq \sum_{i \neq k} |a_{ik}| \quad (i, k = 1,\ 2, \ldots, n)$$

Для случаев, когда условие диагонального преобладания не выполняется для начальной системы, возможно выполнить перестановку внутри системы так, чтобы попытаться достичь диагонального преобладания.

Достоинства:

- Может использоваться, когда вы хотите минимизировать погрешность результатов (с помощью настройки по мере необходимости).
- Нет необходимости хранить полную матрицу в памяти.
- Ошибки не накапливаются как для прямых методов.
- Вычисление каждого уравнения текущей итерации независимо друг от друга (только для предыдущей итерации), и поэтому может быть распараллелено (рассчитано одновременно).

Недостатки:

- На практике для больших матриц редко найти такую, которая бы удовлетворяла условию сходимости.
- Все неизвестные рассчитываются на основе значений, полученных на предыдущей итерации. Если вычислять уравнения последовательно, то уже известны новые значения неизвестных для частей уравнений.
- Если обозначить число итераций за l, тогда алгоритмическая сложность метода $O\left(l * n^2\right)$. Вы не знаете как много итераций потребуется (скорость сходимости) – l может быть как больше, так и меньше n.

Метод Гаусса-Зейделя / Gauss–Seidel method

Отличие метода Гаусса-Зейделя от метода простой итерации заключается в предложении использовать на текущей итерации значения неизвестных, которые были получены ранее на этой же итерации, а не на предыдущей:

$$x_i^{[j+1]} = x_i^{[j]} - \frac{1}{a_{ii}} \left(\sum_{k=1}^{i-1} a_{ik} x_k^{[j+1]} + ? \sum_{k=i+1}^{n} a_{ik} x_k^{[j]} - b_i \right)$$

Однако, такое предположение приводит к необходимости ужесточить условие сходимости итерационного процесса для метода Гаусса-Зейделя так, чтобы хотя бы в одной строке неравенство было строгим:

$$|a_{ii}| > \sum_{i \neq k} |a_{ik}| \quad (i, k = 1,\ 2, \ldots, n)$$

О причинах такой необходимости читателю предлагается порассуждать самостоятельно.

Достоинства:

- Может использоваться, когда вы хотите минимизировать погрешность результатов (с помощью настройки по мере необходимости).
- Нет необходимости хранить полную матрицу в памяти.
- Ошибки не накапливаются как для прямых методов.
- Скорость сходимости может быть выше, чем для метода простых итераций, т. к. используются свежеполученные значения неизвестных с текущей итераций.

Недостатки:

- Сложнее параллелизовать – все уравнения зависят друг от друга в пределах текущей итерации.
- Условие сходимости более строгое, чем у метода простых итераций. Сложнее найти систему, для которой можно было бы применить метод.
- Если обозначить число итераций за l, тогда алгоритмическая сложность метода $O\left(l * n^2\right)$. Вы не знаете как много итераций потребуется (скорость сходимости) – l может быть как больше, так и меньше n.

Глава 11

Лабораторное занятие 4 04.03.

11.1 Подготовка к занятию

11.2 Основные материалы занятия

Глава 12

Лабораторное занятие 5 11.03.

12.1 Подготовка к занятию

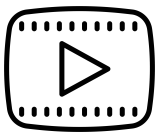
12.2 Основные материалы занятия

Глава 13

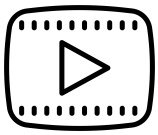
Лекция 4. Решение СНАУ. 13.03.

13.1 Подготовка к занятию

- Обязательное домашнее задание:
  1. Видео: What is Jacobian? | The right way of thinking derivatives and integrals `https://youtu.be/wCZ1VEmVjVo`

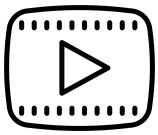


  2. Видео: Approximations. The engineering way. `https://youtu.be/kc9MinCagLQ`



  3. Книга "Алгоритмы эволюционной оптимизации"Дэн Саймон (глава 2).
- Дополнительное рекомендованное домашнее задание:
  1. Книга "Алгоритмы эволюционной оптимизации"Дэн Саймон (вся книга)
  2. Видео: Linear algebra and optimization `https://www.youtube.com/live/u_YC1haw48k?feature=share`

3. Видео: Gradient descent explained `https://youtu.be/i62czvwDlsw`



4. Конспект лекций по "Методам оптимизации"профессора М.Ю. Щеглова

13.2 Основные материалы занятия

Системы нелинейных уравнений

Стоит отметить, что решение нелинейных уравнений и систем нелинейных уравнений относится не только к нахождению корней уравнений, но и к нахождению минимальных и максимальных значений некоторых функций и решению задач оптимизации. Задачи, связанные с решением нелинейных уравнений и систем часто сопряжены наукой о данных и машинным обучением (например, нелинейная регрессия и градиентный спуск).

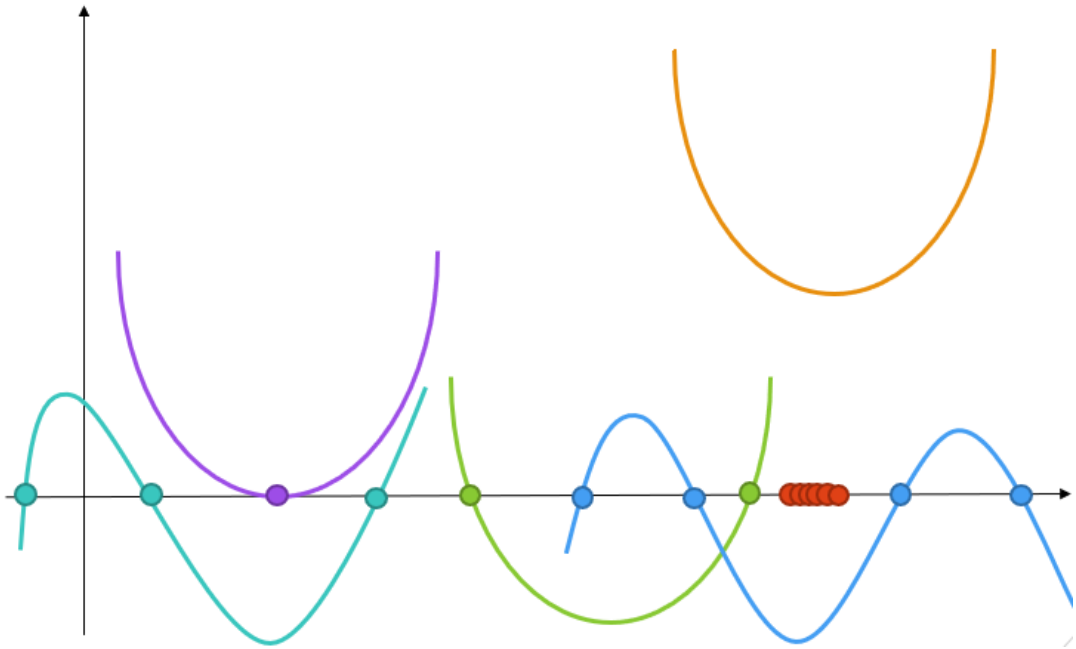
Решение нелинейных уравнений

Прежде всего, мы должны подробнее поговорить о том, что означает нелинейное уравнение?

Трансцендентное уравнение – это уравнение, которое не является алгебраическим. Обычно это уравнения, содержащие экспоненциальные, логарифмические, тригонометрические и обратные тригонометрические функции. Например, $x = \cos(x)$ и так далее.

Более строгое определение можно сформулировать так: трансцендентное уравнение – это уравнение вида $f(x) = g(x)$, где функции f и g являются аналитическими функциями, и по крайней мере одна из них не является алгебраической.

Далее нам понадобится нормальная форма уравнений. Например, у нас есть уравнение $g(V) = I$ В нормальной форме его можно выразить следующим образом: $f(V) = g(V) - I$.



Что значит решить уравнение? Прежде всего, это означает найти корни. Корни уравнения – это нули функции, точки пересечения с осью y. Есть много возможных случаев. Например, у нас может не быть корней, как у функции на графике, показанной оранжевым цветом. Если мы применим какой-либо метод к этой функции, мы можем найти некоторое минимально оптимальное значение этой функции. Мы можем применить некоторые преобразования, и тогда мы найдем корни как оптимум функции.

Следующий случай, когда есть только один корень, как у фиолетовой функции. Когда есть 2 корня, как у функции, показанной зелёным.

Когда есть нечетное количество корней (по крайне мере на рассматриваемом интервале значений аргумента x), таких как бирюзовая линия. И когда есть четное количество корней, таких как синяя линия. Они похожи, но разница обнаруживается при использовании численных методов и алгоритмов. Потому что, когда у вас есть только один корень, как, например, у фиолетовой функции, и вы должны найти его на некотором интервале от a до b, тогда это довольно простая задача, но когда функция содержит больше корней внутри этого интервала, тогда вы должны решить, как ваша программа должна обрабатывать такие случаи. Например, вы должны попытаться разделить его на несколько частей и найти корни внутри них или, например, если у вас есть какая-то функция, такая как $\sin(x)$, то вы можете найти корень этого уравнения только на какой-то части, и если вы выберете такой интервал, как a и b, который содержит больше, чем один корень вы должны решить, как с этим бороться: возможно ли найти

корни или вы не можете. Если у вас будет какое-то правило типа: “Моя программа найдет корень, только если функция имеет различные знаки в $y(a)$ и $y(b)$ (в крайних точках интервала)”: тогда такие случаи, когда у вас есть только один корень и когда у вас нечетное количество корней, будут одинаковыми (это может привести к случай, когда вы вычислили какой-то неправильный корень, когда вам нужно было найти несколько из них, или вы нашли только один и любой из них, и вы не можете сказать, какой именно номер этого корня.) Это также откажет в решении для оранжевого (правильно) и фиолетового (неверно, потому что у нас один корень).

Та же ситуация, когда у нас есть правило: “Моя программа найдет корни, когда $y'(a)$ и $y'(b)$ имеют различный знак (производная функции в концах отрезка)”. В таком случае вы попытаетесь найти корень для оранжевой линии, и вы не сможете найти корни нечетного числа корней.

Если вы решите итеративно разделить интервал от a до b, чтобы найти случай, когда $y(a)$ и $y(b)$ имеют знаки различия, когда у вас должно быть некоторое ограниченное количество таких делений, потому что вы можете найти корень на некотором интервале, который не содержит никаких корней.

Метод половинного деления / Bisection method Первый из рассматриваемых методов – это метод деления пополам. Метод деления пополам (half division / bisection method) удобно использовать для приблизительного нахождения корня (локализация корня, приближение интервала), поскольку с повышением точности объем вычислительной работы значительно возрастает.

Пусть имеем уравнение $f(x) = 0$ и функция $f(x)$ непрерывна на $[a, b]$ и $f(a) * f(b) < 0$.

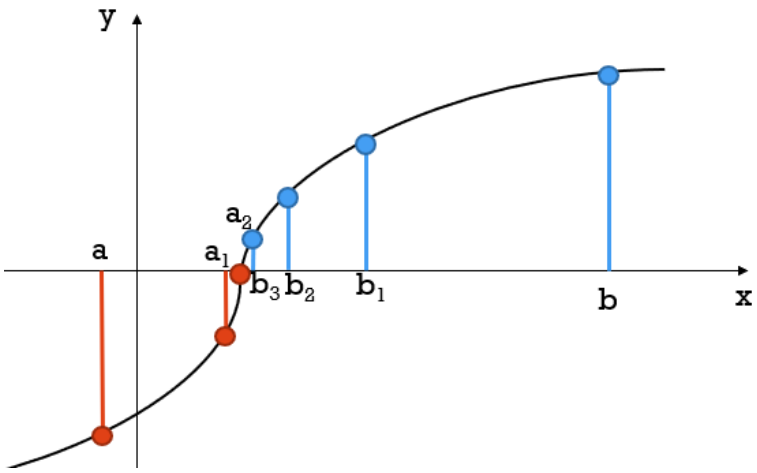
Чтобы найти корень, мы будем делить отрезок пополам. Если $f((a + b)/2) = 0$, тогда $\xi = \frac{a+b}{2}$ – это корень, иначе выбираем ту половину, которая удовлетворяет условию $f(a_1) * f(b_1) < 0$.

В терминах пределов: $\xi = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$, тогда:

$$0 \leq \xi - a_n \leq \frac{1}{2^n}(b - a)$$

Рассмотрим иллюстрацию работы метода. Имеется функция и интервал $[a; b]$. Находим середину интервала и оцениваем знак. Если знак совпадает с правой границей, то получили новое значение интервала справа b_1. Если знак совпадает с левой границей, то нашли новую левую границу a_1. Иначе говоря, мы заменяем значение левой или правой границы интервала новым найденным значением и получаем новый интервал, например: $[a; b_1]$. Работа метода основана на предположении, что если функция в двух крайних точках интервала имеет разный знак, то где-то между ними функция имеет значение 0, что и является корнем. Работа метода продолжается до тех пор, пока не будет найден корень, либо достаточно близкое к нему значение (отличие от 0 меньше заданного значения ε, см. раздел про погрешности и пределы).

На рисунке была показана последовательность, полученная для заданной функции и интервала: $b_1, a_1, b_2, a_2, b_3 \approx 0$. При делении $[a; b]$ пополам получили b_1. При делении $[a; b_1]$ пополам получили a_1. При делении $[a_1; b_1]$ пополам получили b_2. При делении $[a_1; b_2]$ пополам получили a_2. При делении $[a_2; b_2]$ пополам получили b_3, которое равно 0 или близко к нему на величину не больше ε.



Метод хорд / Secant method Следующий метод – метод секущих (метод хорд, secant method). Он также назвал метод пропорциональных частей. Скорость сходимости больше (количество итераций, за которое будет найден корень, меньше), чем в предыдущем методе.

Начальные условия те же, что и для метода деления пополам: пусть имеем уравнение $f(x) = 0$ и функция $f(x)$ непрерывна на $[a, b]$ и $f(a) * f(b) < 0$.

Положим $f(a) < 0$ и $f(b) > 0$. Тогда вместо деления пополам интервал $[a,\ b]$ более естественно делить пополам отношение $-f(a) : f(b)$. Оно возвращает приближенное значение корня $x_1 = a + h_1$, где:

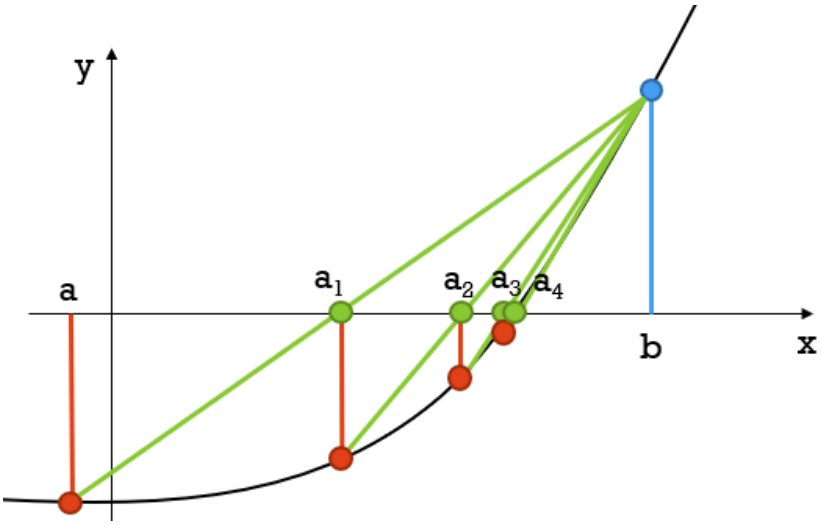
$$h_1 = \frac{-f(a)}{-f(a) + f(b)}(b - a) = \frac{-f(a)}{f(b) - f(a)}(b - a)$$

Тогда:

$$x_1 = a - \frac{f(a)}{f(b) - f(a)}(b - a)$$

Если $|x_n - x_{n-1}| < \varepsilon$, где ε заданная предельная абсолютная погрешность, тогда $|\xi - x_n| < \varepsilon$.

Иначе говоря, строим прямую, соединяющие левую и правую точки интервала и находим 0 линейной функции (как бы заменяем исходную функцию прямой). Затем выбираем это же значение аргумента x, которое дало 0 линейной функции в качестве нового значения интервала. Продолжая так достаточно долго, находим такой интервал, на котором исходная функция совпадает с прямой, тогда там прямая будет проходить через ось x и там же "рядом" (на величину не больше ε) исходная функция будет проходить через ось x. О том, что функция совпадает с прямой мы можем судить, косвенно сравнивая значения корня на текущем и предыдущем разбиении: если разница между ними не больше ε, то и дальнейшие приближения будут находиться в окрестности ε (см. раздел про погрешности и пределы).



Метод Ньютона (касательных) / Newton's (tangent) method Еще одним методом решения нелинейных уравнений является метод Ньютона, называемый также методом касательных. Он очень похож на метод хорд, но мы используем касательную вместо секущей. Метод касательных применяют к некоторому приблизительному корню, чтобы повысить точность его значения.

Геометрически, метод Ньютона эквивалентен замене небольшой дуги кривой $y = f(x)$ касательной линией, проведённой к точке на кривой. Действительно, предположим для определённости, что $f''(x) > 0$ для $a \leq x \leq b$ и $f(b) > 0$.

Выберем, например, $x_0 = b$ для которого $f(x_0) f''(x_0) > 0$. Проведём касательную к кривой $y = f(x)$ в точке $B_0 [x_0,\ f(x_0)]$.

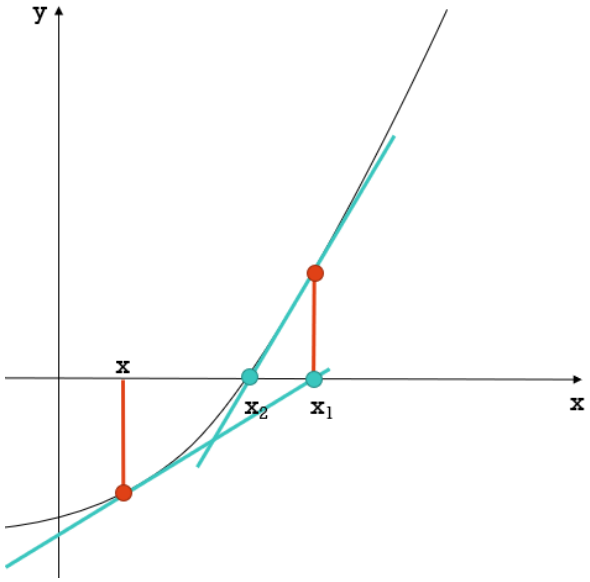
Для первого приближения x_1 корня ξ возьмём абсциссу точки пересечения этой касательной с осью X.

Полагая $y = 0, \quad x = x_{n+1}$:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Если точка x_1 лежит за пределами интервала $[a, b]$, то тогда метод Ньютона не целесообразно использовать на заданных начальных условиях. Таким образом, "хорошее" начальное приближение x_0 это такое, для которого неравенство справедливо с пределом:

$$f(x_0)\, f''(x_0) > 0$$



Метод простой итерации / Fixed-point iteration Как и метод Ньютона, метод простой итерации предполагает, что у нас есть некоторое приближённое значение корня, точность приближения которого необходимо повысить.

Аналогично работе метода простых итераций для решения систем линейных алгебраических уравнений, мы будем заменять исходную функцию эквивалентной функцией и если выбрано хорошее начальное приближение, шаг за шагом будем получать более точное значение корня.

Пусть имеем уравнение $f(x) = 0$ и функция $f(x)$ непрерывна на $[a, b]$ и требуется найти вещественные корни. Заменим на эквивалентное уравнение:

$$x = \varphi(x)$$

Каким-то образом выберем начальное приблизительное значение корня x_0 и заменим его в правой части уравнения (1).

$$x_1 = \varphi(x_0)$$

Теперь вставляем x_1 в правую часть уравнения (1). Повторяя этот процесс, получаем последовательность чисел:

$$x_n = \varphi\left(x_{n-1}\right),\ (n = 1,\ 2, \dots)$$

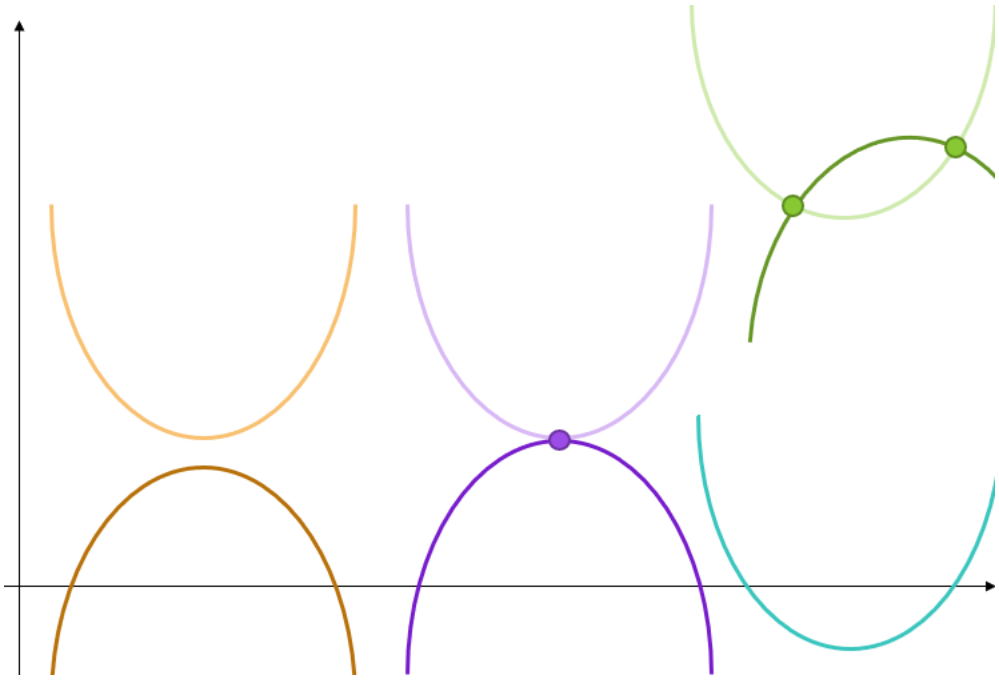
Если итерационный процесс (последовательность) сходится, то есть, если существует предел $\xi = \lim_{n\to} x_n$, то затем перейдя к пределу (2) и предполагая, что функция $\varphi\left(x\right)$ непрерывна, мы найдём:

$$\lim_{n\to\infty} x_n = \varphi(\lim_{n\to\infty} x_{n-1})$$

Решение систем нелинейных уравнений

Решить систему нелинейных уравнений – означает найти все точки пересечения функций. Так, для случая, когда система состоит из двух уравнений возможны следующие ситуации:

- – Корней нет
  - – Существует только один корень
  - – Существует два или несколько корней
  - – Бесконечное количество коней (когда графики функций совпадают)



Нелинейная система Аналогично тому, как мы записывали систему линейных уравнений, мы можем записать для системы нелинейных уравнений общий вид и векторное представление. Общий вид:

$$\begin{cases} F_1\left(x_1, x_2, \ldots, x_n\right) = 0 \\ F_2\left(x_1, x_2, \ldots, x_n\right) = 0 \\ \qquad \ldots \\ F_n\left(x_1, x_2, \ldots, x_n\right) = 0 \end{cases}$$

Векторная запись:

$$F = (F_1, F_2, \ \ldots, F_n), \ x = (x_1, x_2, \ \ldots, x_n)$$

Системы могут быть записаны как $F(x) = 0$

$\widetilde{F} = Jx + c$, где J это матрица размера $n \times n$ и c – это некоторый вектор длины n и $\lim |x_{i+1} - x_i|^2 \to 0$.

$$F(x_{i+1}) \approx F(x_i) + \nabla F(x_i)(x_{i+1} - x_i) \ .$$

Выражение ∇F – это матрица частных производных F. Компонент $(i, \ j \ )$ в ∇F есть $\frac{\partial F_i}{\partial x_j}$. Матрица ∇F – называется Якобианом (матрицей Якоби) F и обозначается J.

$$\nabla F = \begin{vmatrix} \frac{\partial F_0}{\partial x_0} & \frac{\partial F_0}{\partial x_1} \\ \frac{\partial F_1}{\partial x_0} & \frac{\partial F_1}{\partial x_1} \end{vmatrix}$$

Метод простой итерации / Fixed-point iteration

- $$F(x_i)+ \mathrm{J}(x_i)(x_{i+1} - x_i) = 0$$
- Equivalent if $\lambda \neq 0$

Метод Ньютона / Newton's method

Градиентный спуск / Gradient descent Пусть $f(x) = (f_1(x), f_2(x), \ldots, f_n(x)),\ x = (x_1, x_2, \ldots, x_n)$

Система уравнений с $n \geq 2$ эквивалентна:

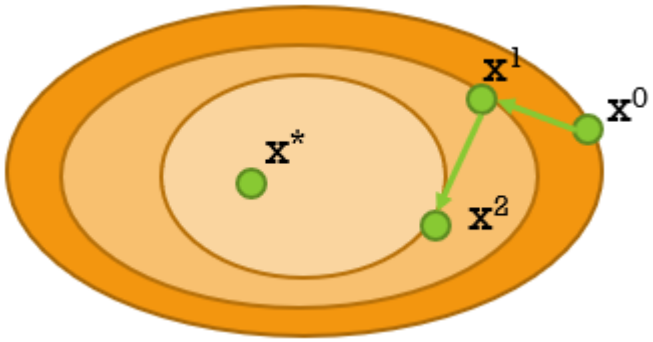
$$\Psi(x) = 0, \text{ где : } \Psi(x) = f_1^2(x) + f_2^2(x) + \ldots + f_n^2(x)$$

Тогда корни таких уравнений – это минимальные значения функции $\Psi(x) = \Psi(x?_1, x_2, \ldots, x_n)$.

Пусть функция $\Psi(x)$ это дважды дифференцируемая область, где содержится изолированное решение x^*. Если начальное значение x^0 мы можем найти минимум функции $\Psi(x^0 - \lambda\ grad\Psi(x^0))$. Таким образом мы находим минимальный неотрицательный корень $\lambda = \lambda_0$ уравнения посредством следующего метода:

$$\frac{d}{d\lambda}\Psi\left(x^0 - \lambda\ grad\Psi\left(x^0\right)\right) = 0$$

Сходимость не гарантируется, потому что мы можем получить только локальный минимум.



Глава 14

Лабораторное занятие 6 18.03.

14.1 Подготовка к занятию

14.2 Основные материалы занятия

Глава 15

Лабораторное занятие 7. 25.03. Рубежная работа 1.

15.1 Подготовка к занятию

15.2 Основные материалы занятия

Глава 16

Лекция 5 27.03. Интегрирование

16.1 Подготовка к занятию

- Обязательное домашнее задание:

    1. Видео: Analogy: Integrals as Multiplication `https://youtu.be/yIuxdrTIwjc` + статья `https://betterexplained.com/articles/a-calculus-analogy-integrals-as-multiplication/`

    2. Книга: Математика в огне. Джейсон Уилкс. (глава 1, 2, 4). Отличное описание базовых математических понятий действительно простым языком, даже если вы полный ноль в математике.

- Дополнительное рекомендованное домашнее задание:

    1. Книга: Крылов В.И. Приближённое вычисление интеграла.
    2. Книга: Infinite Powers - How Calculus Reveals the Secrets of the Universe. Бесконечная сила: как математический анализ раскрывает тайны вселенной. Стивен Строгац.
    3. Научная статья о современном способе расчёта интеграла: FAST-PT: a novel algorithm to calculate convolution integrals in cosmological perturbation theory.
    4. Можно попробовать решить задачи из типового расчёта по математике при помощи вашей лабораторной работы 3.

16.2 Основные материалы занятия

Интегрирование и дифференцирование

Студенты часто испытывают некоторые трудности с пониманием того, что такое интеграл, что такое производная операция и почему они являются взаимообратными операциями. Вам может показаться, что это сложно, но на самом деле концепция интеграла и производной очень простая.

Самое важное, что вы должны здесь понять, это то, что интегральная операция является логическим продолжением идеи умножения. При этом производная операция является логическим продолжением идеи разделения. Например, у нас есть четыре блока с равным весом, пусть 3 килограмма. И если мы хотим вычислить вес 4 объектов, мы можем просто суммировать его, но тогда мы запишем его как 3 кг + 3 кг + 3 кг + 3 кг. Но если блоков больше четырех, это будет большая сумма, и простое стремление записать ее – это умножение. Мы можем просто умножить 3 на 4 из-за двух чисел таких блоков, потому что у нас их 4, и тогда мы получим результат.

Аналогично интеграл – это стремление записать сумму изменяемых объектов. В предыдущем примере у нас будет 4 объекта, но вес их будет разный. Их общую сумму также можно записать как 2кг + 5кг + 1кг + 4кг. В таком случае в математике чаще всего пользуются значком суммы $\sum$. Однако также справедливо использовать и интеграл $\int$. Сумму удобнее при этом использовать в том случае, когда известно конечное (перечислимое) множество слагаемых. Интегралом же пользуются в тех случаях, когда имеют в виду, что ширина нашего каждого слагаемого в сумме стремится быть бесконечно малой.

Иначе говоря интеграл – это предел интегральной (изменяемых значений) суммы. Так как предел – это на практике сложно, мы можем пользоваться приближениями к этому пределу. Приближение к пределу, как вы помните из раздела о погрешностях, - это аппроксимация, нахождение какого-то приближённого числа, которое будет отличаться от точного значения предела на рассчитываемую величину погрешности. При этом, как и говорилось ранее, если точное значение предела не известно, то говорить можно только о предельной (предельной абсолютной или предельной относительной) погрешности, т. к. мы проводим сравнения между значениями, которые мы находим, приближаясь к пределу.

Когда возможно посчитать интеграл

Численные методы интегрирования

Формулы Ньютона-Котеса

Методы прямоугольников

Метод трапеций

Метод Симпсона

Глава 17

Лабораторное занятие 8 01.04.

17.1 Подготовка к занятию

17.2 Основные материалы занятия

Глава 18

Лабораторное занятие 9 08.04.

18.1 Подготовка к занятию

18.2 Основные материалы занятия

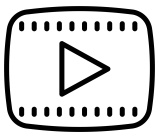
Глава 19

Лекция 6 10.04. Решение Обыкновенных Дифференциальных Уравнений и задачи Коши

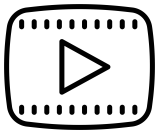
19.1 Подготовка к занятию

- Обязательное домашнее задание:

  1. Видео: Differential equations, a tourist's guide
  2. Видео: Euler's method - Differential equations
  3. Видео: Milne's method simple and good example
  4. Видео: Giving Personality to Procedural Animations using Math `https://youtu.be/KPoeNZZ6H4s`

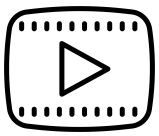
     

  5. Видео: How a Differential Equation Becomes a Robot, Part 1: Overview `https://youtu.be/apPJtJqCu44`

- Дополнительное рекомендованное домашнее задание:

1. Видео: ROB 101 Recitation: Ordinary Differential Equations (ODEs)
   `https://youtu.be/hH35ushpqlw`



19.2 Основные материалы занятия

Глава 20

Лабораторное занятие 10 15.04.

20.1 Подготовка к занятию

20.2 Основные материалы занятия

Глава 21

Лабораторное занятие 11 22.04

21.1 Подготовка к занятию

21.2 Основные материалы занятия

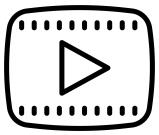
Глава 22

Лекция 7. 24.04. Собственные числа матриц. Преобразование Фурье.

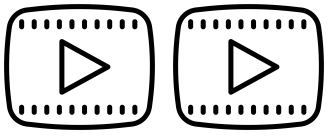
22.1 Подготовка к занятию

- Видео: But what is a partial differential equation? `https://youtu.be/ly4S0oi3Yz8`

- Видео: Что такое ряд Фурье? От теплопередачи до рисующих кругов `https://youtu.be/r6sGWTCMz2k`



- Видео: Самый важный алгоритм в истории `https://youtu.be/eQlSvfUuQNs` - The Remarkable Story Behind The Most Important Algorithm Of All Time `https://youtu.be/nmgFG7PUHfo`



22.2 Основные материалы занятия

Глава 23

Лабораторное занятие 12 29.04.

23.1 Подготовка к занятию

23.2 Основные материалы занятия

Глава 24

Лабораторное занятие 13 06.05.

24.1 Подготовка к занятию

24.2 Основные материалы занятия

Глава 25

Лабораторное занятие 14 13.05.

25.1 Подготовка к занятию

25.2 Основные материалы занятия

Глава 26

Лабораторное занятие 15 20.05.

26.1 Подготовка к занятию

26.2 Основные материалы занятия

Глава 27

Рубежная работа 2 22.05.

27.1 Подготовка к занятию

27.2 Основные материалы занятия

Содержание занятия

- Moodle: Рубежная работа №2 ∼90min.

Глава 28

Лабораторное занятие 16 27.05.

28.1 Подготовка к занятию

28.2 Основные материалы занятия

На финальном занятии курса состоится мини-хакатон по трейдингу, в котором Вам будет предложено разработать алгоритм, который сможет получить наибольшую прибыль (заработок - проигрыш) на представленных тестовых данных.

Все участники занятия смогут получить свои баллы за практическое занятие, а победитель, при условии написанных у него успешно рубежных работ получит автоматом зачёт по курсу.

Часть III

Руководство по выполнению лабораторных работ

Глава 29

Общие положения по лабораторным работам

Язык программирования: Java, C++ или Python. Язык программирования от работы к работе может отличаться.

Конечному пользователю должна быть представлена интуитивно понятная программа, без значительных дефектов (плавающие поля, бесконечно множащиеся ячейки, обработанный ввод некорректных данных и т.д.).

В программе численный метод должен быть в виде отдельной подпрограммы (метод, функция, процедура) или класса, в который исходные данные передаются в качестве параметров, выходные - тоже (либо возвращаемое значение). Ввода-вывода в классе (подпрограмме), где реализован сам численный метод, быть не должно (учимся писать код так, чтобы можно было повторно использовать). Например, код численного метода из лабораторной работы 1 вполне можно будет использовать в лабораторной работе 3, 4 и 5.

Например, если Вам необходимо реализовать метод для интегрирования чисел, то в качестве аргументов ему будут переданы подынтегральная математическая функция (как lambda-функция, объект функции или указатель на функцию или хотя бы номер функции) и границы интегрирования. Численный метод не должен знать о существовании пользователя или быть зависимым от других внешних источников. Численный метод не должен запрашивать значения у пользователя или писать в консоль результат или сообщения об ошибках.

Аналогичные ограничения распространяются и на класс, если он реализует численный метод. О любых внештатных ситуациях можно сообщить пользователю иными средствами выбранного Вами языка программирования, например, механизмом исключений. Класс, работающий со структурами данных для численного метода, может содержать метод, подготавливающий вывод объекта, например метод to_string().

Глава 30

Требования к оформлению отчётов

1. Описание метода, расчетные формулы; (Материал пишется самостоятельно по результатам изученного в ходе подготовки к лабораторной работе материала, а не копируется с различных источников, в том числе сайтов и книг)

2. Блок-схема численного метода; (см. требования к оформлению блок-схем)

3. Листинг реализованного численного метода программы; (весь листинг писать нет необходимости, только численный метод)

4. Примеры и результаты работы программы на разных данных; (3-4 примера)

5. Вывод.

Обратите внимание на порядок следования элементов отчёта и отсутствия раздела «Цель». Порядок элементов определён порядком выполнения самой работы: сначала изучается сам метод и его теоретическая основа, на основе чего строится блок-схема. Только после этого можно приступать к написанию кода на основе блок-схемы. После написания кода происходит самое важное: вы запускаете численный метод на разных данных и анализируете результат (это и есть цель лабораторной работы, а вовсе не написание кода как таковое). Как раз на основе этого вы пишете результат о том, каким является исследованный вами метод.

Раздел «Цель» отсутствует в отчёте, так как в большинстве случаев является бездумным копированием темы работы и не несет смысловой нагрузки ни для преподавателя, ни для студента.

30.1 Как составить блок-схему численного метода

Блок-схема должна строиться до написания кода, в тот момент, когда у вас ещё нет никаких переменных и программных функций. Поэтому блок-схема должна содержать в себе математические обозначения используемых понятий.

Вместо того, чтобы писать содержимое блоков в виде текста (на русском или английском языках), используйте язык формул и математических обозначений. Если частично есть слова на русском или английском языке (например, «Начало», «Конец» и пр.), то проверьте, что это везде один и тот же язык, включая стрелки на логических блоках («Да», «Нет»).

Например, если мы работаем с матрицей, как в лабораторной работе 1, нам необходимо использовать математическую запись элементов матрицы $x_i j$, а не x[i][j].

Сама блок-схема должна строиться на основе принципов построения блок-схем программ. Об основах можно прочитать подробнее: `https://ru.wikipedia.org/wiki/РҪРьР«Рэ-СҒСЖРҳРёРө`. Проверьте так же, что из блок-схемы можно понять его алгоритмическую сложность.

Для построения блок-схем допустимо использовать любые подходящие для этого программы, например:

1. Draw.io
2. Visio
3. Dia
4. Встроенные средства Word или LibreOffice Draw

30.2 Как написать вывод по лабораторной работе

Вывод должен содержать анализ выполненной работы:

- результаты запуска реализованного метода на различных данных; (не сам факт, что запускается, а что это значит и почему работает или не работает именно так)
- сравнение с другими методами;
- анализ применимости метода;
- анализ алгоритмической сложности метода;
- анализ численной ошибки самого численного метода и пр.

Вывод содержащий текст вроде "Я реализовал ... я молодец"не засчитываются как вывод к лабораторной работе.

30.3 Критерии оценивания отчётов по лабораторным работам

Оценка по отчёту измеряется по шкале от 0 до 100, что в конечном счёте принимается как процент выполнения отчёта по лабораторной работе.

Если Вы получили менее 60% за отчёт, то его необходимо исправить и загрузить повторно. Дата при этом будет обновлена.

Если Вы получили более 60%, но менее 100% за отчёт, то Вы также можете его исправить и загрузить повторно. Дата при этом также будет обновлена. Обновление даты ведёт к тому, что возможно, итоговое количество полученных баллов и не будет повышено, однако, если работа выполняется в срок и до дедлайнов, у преподавателя есть время всё проверить и дать Вам обратную связь, то это может быть полезно, чтобы получить более качественную работу и более высокий балл.

Отчёты по лабораторным работам оцениваются по следующим критериям:

- Качество описания численного метода (15 баллов)
- Корректность блок-схемы, составленной по описанному ранее численному методу (15 баллов)
- Наличие ошибок при написании кода численного метода (30 баллов)
- Примеры работы программы (10 баллов)
- Корректность сделанных выводов (25 баллов)
- Наличие всех разделов отчёта (5 баллов)

Качество описания численного метода

В разделе с описанием численного метода предполагается, что Вы самостоятельно, своими словами опишите как он работает. Нет необходимости цитировать учебник, постарайтесь изложить его **описание так, как Вы его поняли**. Это, наряду с выводами, один из самых важных и самых ценных разделов отчёта по лабораторной работе. Не забудьте упомянуть все требуемые методу формулы и почему они выглядят именно так, а не иначе.

Самостоятельность, ясность и полнота описания метода:

- Ответ должен быть написан самостоятельно, а не сгенерирован нейронной сетью.
- Ответ должен быть написан самостоятельно, а не плагиатом с сайтов.
- Ответ должен быть написан самостоятельно, а не написан одногруппниками и пр.

- Ответ должен быть написан самостоятельно в виде текста, а не картинки. Картинка часто говорит о плагиате, чтобы не было возможности его проверить. В особо наглых случаях, студенты считают, что достаточно вставить скриншот/фото из книги.

Во всех указанных выше случаев ставится 0 за этот критерий.

Если метод описан не полностью, то -5 баллов. Если метод описан не досточно ясно, то -5 баллов.

Корректность блок-схемы, составленной по описанному ранее численному методу

Подробнее см. соответствующий раздел.

- Блок-схема должна быть построена самостоятельно. Иначе, ставится 0 баллов за этот критерий.
- Блок-схема должна строиться до написания кода, а не после. Обратное легко заметно по наличию, например, вызовов собственных функций и пр. (-5 баллов)
- Блок-схема должна содержать математические обозначения переменных, а не программных. (-2 балла)
- Для построения блок-схемы должны использоваться стандартные для этого блоки. (Блоки для записи на магнитную ленту не подходят :)) (-2 балла)
- Исходя из блок-схемы численного метода должно быть легко понять его алгоритмическую сложность. (не все студенты знают что это такое и как оно рассчитывается, часто это нужно объяснять отдельно). (-3 балла)
- Допускается рисование блок-схем "от руки"с вставлением фотографии, но при этом баллы снижаются (-2 балла). Странно учиться на программиста и не уметь пользоваться подходящими инструментами построения блок-схем.

Некоторые студенты будут утверждать, что не могут использовать математические обозначения вместо программных, однако, они сами выбирают инструмент построения блок-схем, поэтому этот довод не засчитывается.

Наличие ошибок при написании кода численного метода

Код численного метода должен быть оформлен в соответствии с оговоренными практиками чистого кода особенно часто применяемыми при разработке математического кода:

1. Код численного метода должен быть оформлен в виде отдельной подпрограммы (функции, класса, процедуры или пр.). Например, код численного метода не должен быть частью функции main.

2. В численном методе не должно быть выводов в консоль / на график / отправки на сервер или ещё куда-то. Код численного метода (программная функция) выполняет свои расчёты в соответствии с получаемыми аргументами. Он не должен знать о существовании чего-то ещё, кроме как ровно той функции / расчёта, которую он выполняет.

3. Исходя из предыдущего пункта, код численного метода / функции, должен возвращать именно то значение, которое от него ожидают. Например, функция реализующая сложение 2 чисел не должна возвращать строку.

4. Названия численных методов должны отражать ровно то, что они делают. Названия функций всегда глаголы, названия переменных и названия классов, модулей - это существительные. Названия переменных должны быть тем подробнее, чем больше область их видимости. Локальные переменные не должны быть однобуквенными, за редким исключением (номер итерируемого индекса i, j). Если функция возвращает бинарное значение (true/ false), то название такой функции должно однозначно трактовать, что за результат мы получили. Например, функция, проверяющая, является ли число нечётным: is_odd.

5. Ошибки и исключения должны корректно обрабатываться, а не выводиться пользователю.

6. Методы не должны быть слишком большими и запутанными, не должно быть большого количества вложенных циклов. Слишком маленьких методов не бывает. Даже однострочные функции - это нормально, но надо помнить о правильном именовании (см. пункт 4).

За каждую из ошибок, описанную выше, балл за критерий снижается на 5.

Обратите внимание, что в отчёте нет необходимости приводить листинг всей своей программы. Вы можете ограничиться только реализованным численным методом и теми методами, которые они вызывают (например, метод перестановки строк матрицы).

Примеры работы программы

Наличие минимум 5 различных примеров работы программы на различных данных, включая граничные случаи и исключительные ситуации (например, когда метод не применим).

- Примеры работы программы должны быть различными и соответствовать исследуемому разделу курса.

- Минимум должно быть 5 различных примеров.

Если примеров менее 5, то за критерий ставится 0 баллов. При наличии 5 и более примеров, за отсутствие каждого из типов примеров (граничные сучаи и пр.), соответствующих критерию -2 балла.

Корректность сделанных выводов

Подробнее о способах написания выводов см. специальный раздел.

- Ответ должен быть написан самостоятельно, а не сгенерирован нейронной сетью. - Ответ должен быть написан самостоятельно, а не плагиатом с сайтов. - Ответ должен быть написан самостоятельно, а не написан одногруппниками и пр. - Ответ должен быть написан самостоятельно в виде текста, а не картинки. Картинка часто говорит о плагиате, чтобы не было возможности его проверить. В особо наглых случаях, студенты считают, что достаточно вставить скриншот/фото из книги.

Во всех указанных выше случаях ставится 0 за этот критерий.

Как и указано выше, вывод должен содержать в себе анализ численного метода, а не рассуждения о том, как "я программировал программу". Написание программы не является целью работы в отличии от изучения / исследования численного метода.

Отсутствие каждого из критериев -5 баллов.

Наличие всех разделов отчёта

Наличие всех разделов отчёта: описание метода, блок-схема, код численного метода, примеры работы программы, вывод.

- Наличие цели работы допускается, но не требуется, потому что как правило не содержит осмысленного текста.
- Наличие описания задания на лабораторную работу не требуется.
- Если по одному из критериев, например, описание метода, получен 0, т.е. критерий не выполнен, то критерий наличия всех разделов отчёта также 0.

Глава 31

Лабораторная работа 1: Аппроксимация и интерполяция

Обратите внимание, что в данной лабораторной работе Вам необходимо строить графики функций. На графике необходимо отобразить исходные точки и точки, полученные при помощи разработанного численного метода, согласно варианту. Для методов аппроксимации необходимо показать 2 полинома, полученные из полного и уменьшенным на 1 точку наборов данных, а также отдельно выделить точку, которая была выброшена по соответствующему методу критерию.

В лабораторной работе Вам будет предложено исследовать один из следующих вариантов:

1. Метод Ньютона
2. Метод Лагранжа
3. Метод интерполяции кубическими сплайнами
4. Аппроксимация по методу наименьших квадратов
5. Аппроксимация по методу наименьших модулей
6. Аппроксимация сигмоидами
7. Аппроксимация сплайнами
8. Метод Ньютона с полиномами Чебышёва
9. Метод Лагранжа с полиномами Чебышёва
10. Метод интерполяции кубическими сплайнами с полиномами Чебышёва

При анализе методов в лабораторной работе следует обратить внимание на сценарии их применения, основываясь на особенностях методов.

31.1 Методы интерполяции

В вариантах 1-3 Вам необходимо построить интерполяцию по заданному набору точек. Результатом будет являться полином (варианты 1-2) или набор полиномов (вариант 3). По полученному полиному необходимо уметь рассчитывать значение для новых x, не входящих в изначальный набор.

Важно, что после построения полинома, новых x может быть сколько угодно и перестраивать полином заново для каждого значения не нужно. Например, Вы построили полином Лагранжа - фактически это функция, в которую можно передать сколько угодно различных значений и она вернет соответствующее значение интерполирующего полинома. Почему-то иногда это бывает не очевидно и студенты расчитывают весь полином заново по одному и тому же исходному набору значений заново, что кратно увеличивает количество операций: вместо того, чтобы посчитать только лишь значение полинома, полином ещё раз строится. В этом нет никакой необходимости. Аналогично для методов Ньютона и метода интерполяции кубическими слайдами.

Метод Ньютона как для варианта 1, так и для варианта 8 можно реализовывать как через формулу конечных разностей, так и через формулу разделённых разностей.

31.2 Методы аппроксимации

В вариантах 4-7 Вам предлагается построить аппроксимацию по соответствующим формулам. В качестве фунции аппроксимации Вы можете сами подбирать вид аппроксимирующего полинома в тех методах, где это применимо. Например, при аппроксимации методами наименьших квадратов может использоваться как линейная, так и квадратичная или любой другой вид полинома, включая логорифмическую, тригонометрическую и пр.

Для варианта аппроксимации следует применить метод аппроксимации дважды: второй раз метод должен быть запущен на укороченном списке входных точек, при чём исключена должна быть точка, имеющая наибольший квадрат отклонения после первого запуска. Иначе говоря, алгоритм, следующий:

1. Задается произвольный (полученный ранее) набор значений пар (x;y).
2. Задается аппроксимирующая функция.
3. Рассчитываются программой коэффициенты аппроксимирующей функции.
4. Производится поиск точки с наибольшим отклонением от значения, полученного при помощи аппроксимирующего многочлена.
5. Найденная точка исключается и производится пересчёт коэффициентов аппроксимирующего многочлена (см. п.3).

6. Строится график, содержащий в себе две функции (1 - до исключения, 2 - после исключения и пересчёта) и набор заданных изначально точек (пар значений (x,y)).

7. Помимо этого, отдельно на экран выводятся полученные значения аппроксимирующих коэффициентов.

31.3 Методы интерполяции с полиномами Чебышёва

В вариантах 8-10 Вам предлагается построить интерполяцию после того, как из всего набора заданных точек останутся только те, которые будут соответствовать корням полинома Чебышёва. Отдельно отметим, что единичная сфера (от -1 до 1) может масштабироваться и за счёт коэффициентов так, чтобы получался необходимый интервал интерполяции (например, от -7 до 3).

Во всех случаях вам необходимо сгенерировать набор входных точек, на которых вы будете тестировать свою программу. Набор точек должен представлять собой некоторое количество пар $(x; y)$ полученных из какой-то существующей функции. Чтобы сгенерировать такой набор данных рекомендуется использовать подготовленную функцию, поведение которой вам уже известно. Предпочтительно иметь несколько наборов входных данных из нескольких функций. Также желательно добавить шум (отклонения значений y от математического). Имеет смысл также отобразить на графике функцию, на основе которой был получен набор данных.

Глава 32

Лабораторная работа 2: Решение систем линейных уравнений

Вам будет выдан один из следующих вариантов:

1. Метод Гаусса
2. Метод Гаусса с выбором главного элемента
3. Метод простых итераций
4. Метод Гаусса-Зейделя
5. Метод Холецкого
6. Метод прогонки

В лабораторной работе Вам предлагается реализовать один из методов, который бы позволял находить столбец неизвестных для системы линейных алгебраических уравнений. Размер системы предполагается ограниченным 20, это означает, что Вашу программу после реализации необходимо будет проверить на матрице размером 20 x 20 + 20 элементов (количество неизвестных в матрице A и столбце B). Исходя из этого условия матрицы такого размера должны иметь возможность каким-то образом попадать в Ваш численный метод. Например, ввод данных в ручном формате может быть затруднительным для 420 элементов системы с 20 неизвестными. Традиционно предлагается сделать следующие методы ввода данных в программу:

- Пользовательский ввод.
- Ввод данных из файла.
- Генерация случайных матриц.

32.1 Пользовательский ввод

Пользовательский ввод данных позволяет протестировать программу относительно просто и не требует дополнительных усилий со стороны разработчика, однако, по этой же причине вполне может быть заменён исключительно вводом из файла. С другой стороны, если такой способ ввода уже существует, то почему бы не сделать его приятным для пользователя?

Например, в случае, если пользователь вводит некорректные данные (буквы вместо чисел, две запятые или точку вместо запятой в вещественных чисел, пробел между числом и знаком минуса – всё, что не может быть распознано как число; не правильное количество данных; матрицу, определитель которой будет свидетельствовать об отсутствии решений; отрицательное значение размера матрицы и пр.) пользовательский ввод необходимо прервать с просьбой повторного ввода как можно раньше, а не продолжать запрашивать данные, чтобы сообщить об ошибке в самом конце. Особо необходимо отметить, что все подобные сценарии ошибок пользовательского ввода должны быть обработаны, при чём не только для ввода данных через терминал, но и для, например, ввода данных из файла.

32.2 Ввод данных из файла

Ввод данных из файла представляется хорошим способом проверки корректности работы численного метода, особенно когда речь идёт о матрицах большого размера. При вводе данных из терминала вполне возможно ошибиться, а при генерации случайной матрицы результат трудно проверяем. При работе с файлами данных необходимо продумать формат данных, с которыми будет проводиться работа: какие данные содержатся в самом файле, а какие вводятся отдельно пользователем. Например, для итерационных методов файл с данными будет содержать размер матрицы и саму матрицу, а точность приближения будет задавать пользователь. Так же необходимо продумать вопрос локализации: какой знак для вещественных чисел Вы будете использовать: точку или запятую. Для самопроверки численного метода в системе Moodle загружены несколько файлов с решениями. При защите лабораторной работы преподаватель может попросить Вас продемонстрировать работу Вашей программы на каком-то ином файле для того, чтобы проверить, действительно-ли Ваш реализованный численный метод считает результат достаточно точно.

32.3 Генерация случайных матриц

Генерация случайных матриц может быть реализована двумя основными способами:

1. Простая генерация матриц A, удовлетворяющей условиям применимости метода, и столбца B;

2. Генерация столбца неизвестных и генерация матрицы A, удовлетворяющей условиям применимости метода, затем расчёт правой части уравнений и «забывание» столбца неизвестных. В таком случае можно сравнить рассчитанные и «загаданные» значения неизвестных.

Обратите внимание, что в обоих случаях имеет смысл генерация только матриц, имеющих решение реализуемым Вами численном методе. Чем больше будет размер матрицы, тем меньше будет вероятность сгенерировать матрицу, которая будет подходить под условия применимости метода! Особенно это актуально для итерационных методов. Обратите внимание, что условия сходимости итерационных процессов для методов простой итерации и для метода Гаусса-Зейделя – отличаются.

В отчёт по лабораторной работе в раздел с примерами нужно будет вставить в том числе пример с матрицей некоторого размера большего, чем обычно считают, например системы с 5-6 неизвестными. Особый интерес представляют собой матрицы размера 20x20. Вы должно всё таки исследовать работу своего метода на таких матрицах особенно тщательно. Если метод будет путём перемножения миноров расчитывать определитель, чтобы сделать вывод о применимости метода, то для таких матриц результат можно ждать очень долго. Решение на Code&Test в таком случае не пройдёт проверку, так как исполнение программы будет занимать слишком много времени.

При анализе реализованного численного метода Вам необходимо запускать его на различных наборах данных, а также попытаться понять работает ли программа на этом наборе данных правильно, если не работает, то должна ли работать и если нет, то почему. Так же предлагается проанализировать скорость работы численного метода. Сделать это можно путём сравнения аналитического значения алгоритмической сложности метода (по составленной ранее блок-схеме) и путём реальных замеров скорости работы численного метода различными инструментами выбранного вами языка программирования. Например, в языке Python это можно сделать до и после вызова численного метода использовать команду $datetime.datetime.now()$, а затем вычесть полученные значения. Аналогично в Java можно использовать $System.currentTimeMillis()$.

Можно также использовать различные средства для анализа потребляемой памяти. Например, для Java можно использовать VisualVM для просмотра динамики работы JVM, либо запрашивать текущее состояние памяти через запуск в JVM команд: $Runtime.getRuntime().freeMemory()$. Аналогично можно поступать и для других языков программирования по Вашему желанию.

Для начала тестирования Вашей программы предлагаю использовать два простейших набора данных:

- Все элементы матрицы A равны 0, а значения столбца B – любые не одинаковые числа;

- Все элементы матрицы A равны 1, а значения столбца B – любые не одинаковые числа.

- Любая матрица, определитель которой для самопроверки Вы можете рассчитать при помощи метода Крамера.

Обратите также внимание, что для прямых и итерационных методов вывод в консоль должен отличаться. Для прямых методов (метод Гаусса и метод Гаусса с выбором главных элементы) должно быть реализовано:

- Вычисление определителя;

- Вывод треугольной матрицы (включая преобразованный столбец B);

- Столбец неизвестных;

- Столбец невязок.

Для итерационных методов (метод простой итерации и метод Гаусса-Зейделя) должно быть реализовано:

- Точность задается пользователем;

- Проверка диагонального преобладания для соответствующего метода (В случае, если диагональное преобладание в изначальной матрице отсутствует - предлагается сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто. В случае невозможности достижения диагонального преобладания - выводить сообщение.);

- Столбец неизвестных;

- Количество итераций, за которое было найдено решение;

- Столбец погрешностей.

Для прямых методов часто представляет собой трудность расчёт невязки. Невязка представляет собой разницу между левой и правой частью уравнения. Таким образом для каждого уравнения в системе будет рассчитываться собственное значение невязки. Если правая часть уравнения у нас уже есть – это соответствующий элемент матрицы B, то левую часть нужно рассчитать, подставив на места неизвестных x рассчитанные при помощи численного метода значения, а затем произведя операции умножения и сложения. Невязка – это то, на сколько правая часть уравнения не равна левой, в то время как в уравнении по определению обе части должны быть равны.

При расчёте определителя для прямых методов и в особенности для метода Гаусса с выбором главного элемента, необходимо помнить и учитывать свойства определителя.

В выводе помимо анализа самого метода необходимо сравнить его со вторым методом той же категории (например, для метод простой итерации сравнить с методом Гаусса-Зейделя), а также прямые методы в целом сравнить с итерационными.

Глава 33

Лабораторная работа 3: Решение систем нелинейных уравнений

В данной лабораторной работе Вам будет предложено реализовать один из следующих методов решения систем нелинейных уравнений:

1. Метод Ньютона
2. Метод простых итераций
3. Метод градиентного спуска

Нелинейные уравнения и системы нелинейных уравнений студенту предлагается выбрать самостоятельно. Тем не менее, необходимо учитывать области допустимых значений. Для примеров можно выбрать как нелинейные алгебраические, так и не алгебраические нелинейные уравнения.

Глава 34

Лабораторная работа 4: Интегрирование

Среди методов численного интегрирования вам будет предложено реализовать один из следующих методов:

1. Метод прямоугольников.
2. Метод трапеций.
3. Метод Симпсона (метод парабол).
4. Метод Люстерника-Диткина.
5. Метод кратного интегрирования методами прямоугольников.
6. Метод кратного интегрирования методом трапеций.
7. Метод кратного интегрирования методом Симпсона.
8. Метод Монте-Карло.

При этом, если Вам было задано реализовать метод прямоугольников, то должно быть реализовано все три его варианта:

- Метод левых прямоугольников;
- Метод правых прямоугольников;
- Метод средних прямоугольников;

В этом случае пользователю нужно лишь единожды задать параметры, для которых он хочет получить результат, а применено должно быть все три варианта, без уточнения каким именно вариантом метода прямоугольника ему следует рассчитать интеграл.

При расчёте интеграла необходимо учитывать ОДЗ функции и необходимые и достаточные условия существования определённого интеграла.

Если на интервале интегрирования существует устранимый разрыв первого рода, то его следует устранить одним из способов (с уточнением пользователю, каким именно способом Вы собираетесь его устранять) и выполнить расчёт интеграла. Например, может быть выполнен расчёт левой части интеграла от разрыва и правой в отдельности. Альтернативным методом устранения разрыва будет принять алгоритмическое среднее от значения от двух соседних точках функции $f(x-\epsilon), f(x+\epsilon)$, где ϵ – наперёд заданная малая постоянная, которая может быть зависима от текущего шага разбиения интеграла. В обоих случаях Вы должны продумать стратегию алгоритма устранения разрыва при попадании его на самую границу интервала.

Важным вопросом, который надо рассмотреть в данной лабораторной работе – погрешность квадратурных формул интегрирования, соответственно обозначаемых как r_i на каждом разбиении интеграла и R на глобальном интервале $[a;b]$, на котором производится интегрирование функции. На основании того факта, что интеграл по определению является пределом суммы бесконечно малых разбиений, без уточнений как именно это разбиение будет производиться, то в пределе абсолютно все методы численного интегрирования будут давать верный результат при правильном их применении. Поэтому для сравнения методов между собой следует анализировать максимальную величину шага разбиения, либо количество разбиений (особенно для методов с динамическим, а не постоянным шагом). В прямую это можно сделать при помощи сравнения полученного значения интеграла с его аналитическим значением, рассчитанным, например, по формуле Ньютона-Лейбница или табличных значений интеграла. Однако, для глобальных выводов необходимо также анализировать погрешности самих квадратурных формул R, которые также дают ответ на то, какие функции следует интегрировать при помощи тех или иных численных методов интегрирования. Ваши рассуждения, наблюдения и результаты анализа и должны составлять основу вывода в данной лабораторной работе.

Глава 35

Лабораторная работа 5: Решение дифференциальных уравнений

В лабораторной работе 5 вам будет предложено реализовать один из методов решения задачи Коши – решения дифференциального уравнения по заданным начальным условиям. В данной лабораторной работе также необходимо построить графики полученного решения, на котором также будет представлен график аналитического решения дифференциального уравнения.

Одношаговые методы:

- Метод Эйлера
- Усовершенствованный метод Эйлера (не путать с улучшенным методом Эйлера)
- Улучшенный метод Эйлера
- Метод Рунге-Кутты 4-го порядка

Многошаговые методы:

- Метод Милна
- Метод Адамса (не путать с методом Адамса-Башфорта)
- Метод Адамса-Башфорта

Методы решения дифференциального уравнения второго порядка:

- Метод пристрелки
- Метод прогонки

В лабораторной работе Вам предлагается подготовить некоторое количество дифференциальных уравнений, которые сможет решать пользователь, вводя собственные начальные условия. Помимо этого, предполагается, что пользователь сможет выбирать как далеко нужно решить дифференциальное уравнение от начального условия. При анализе и демонстрации лабораторной работы рекомендуется исследовать не только различные дифференциальные уравнения, но и различные начальные условия для одного дифференциального уравнения – график решения при этом вероятно будет меняться.

На графике необходимо показать начальное условие и полученное решение дифференциального уравнения. Полезно также отобразить аналитическое решение дифференциального уравнения, чтобы иметь возможность сравнить и проанализировать накопление или не накопление ошибки. Аппроксимацию в обоих случаях (для аналитического и для полученного решения) необходимо выполнять по методу из лабораторной работы 4 (если в лабораторной работе 4 вашим вариантом была аппроксимация, то удваивать количество графиков – до и после удаления точек, как в лабораторной работе 4 - рисовать не нужно, метод должен быть применён только один раз).

В анализе лабораторной работы следует обратить внимание на понятие устойчивости решения в терминах решения дифференцирования – будет ли ошибка накапливаться и возрастать по мере удаления от начального условия, а также на сценариях применения, основанных на особенностях методов.

Глава 36

Лабораторная работа 6: Зачёт

В зачётной лабораторной работе 6 будет требоваться реализовать один из следующих методов. Методы нахождения собственных чисел матрицы:

1. Степенной метод
2. QR-метод

Методы преобразования Фурье:

1. Быстрое Преобразование Фурье
2. Дискретное Преобразование Фурье

Требования к работе аналогины требованиям к предыдщим лабораторным работам.

36.1 Методы нахождения собственных чисел матрицы

Вам необходимо задать квадратную матрицу условно произльного размера (например, размера 20x20) и найти либо максимальное собственное число матрицы при помощи степенного метода, либо все собственные числа матрицы (QR-метод).

36.2 Методы преобразования Фурье

Вам необходимо исследовать соответствующий метод преобразования Фурье и вывести его результат. Обратите внимание, что в данной работе от Вас будет требоваться работа с комплексными числами. При этом Вам необходимо будет самостоятельно реализовать вывод данных в соответствии с описанным форматом.

Часть IV

Рекомендованная литература и иные ресурсы по курсу

Глава 37

Общие ресурсы по курсу

Для начала заметим, на всякий случай, что вычислительная математика и численные методы – это одна и та же область науки, поэтому подбор литературы не должен опираться только на одно из названий.

Список основной литературы, полезной для изучения курса и подготовки лабораторных работ:

Обратите внимание, что на многие книги приведена ссылка с доступом на `https://e.lanbook.com/` , а значит, что они бесплатно доступны вам после регистрации в библиотеке ИТМО.

Для примера оглавление второй книги из списка. Жёлтым отмечены разделы, которые непосредственно относятся к выполнению лабораторных работ, синим – полезные для прочтения главы, желательные для более глубокого понимания. Последняя глава не рассматривается в данном курсе, но для заинтересованных в решении дифференциальных уравнений в частных производных, могу порекомендовать отличный курс видео с подробными разборами от преподавателей MIT.

ОГЛАВЛЕНИЕ

1. Демидович Б. П., Марон И. А. - Основы вычислительной математики – отличная книга с подробными разъяснениями и подробными выкладками.

2. Копченова Н.В., Марон И.А. - Вычислительная математика в примерах и задачах – замечательная книга, с объяснениями, разобранными примерами и задачами (с ответами в конце книги). Прекрасно дополняет первую книгу.

3. Волков Е. А. - Численные методы – Эта книга написана более простым языком, по сравнению с предыдущими. Автор даёт подробные указания о различных методах, некоторые из них могут быть пропущены в предыдущих книгах.

4. Демидович Б.П., Марон И.А., Шувалова Э.З. - Численные методы анализа. Приближение функций, дифференциальные и интегральные уравнения – Будет полезной находкой для тех, кто стремиться лучше разобраться в решении ОДУ (лабораторная работа №5), интегрировании (лабораторная работа №3) и приближении функций (лабораторная работа №4).

5. Турчак Л.И., Плотников П.В. - Основы численных методов – Книга даёт простые и дельные объяснения большинства методов, используемых в качестве вариантов для лабораторных работ.

6. Форсайт Дж., Малькольм М., Моулер К. - Машинные методы математических вычислений – Книга предлагает альтернативный взгляд на численные методы.

7. Фаддеев М.А., Марков - Основные методы вычислительной математики – Скорее дополнительная книга, которая помогает посмотреть на изучаемую тему с разных сторон.

37.1 Базовые ресурсы по математике

1. Справочник по математике для научных работников и инженеров. Определения, теоремы, формулы : пер. с англ. / Г. А. Корн, Т. М. Корн . – Изд. 6-е, стер. – СПб. [и др.] : Лань, 2003. – 831 с. : ил. – (Учебники для вузов. Специальная литература). – Библиогр.: с. 796-800. – Предм. указ.: с. 804-831. – Отличная книга, которую я правда использую как настольный справочник и в университете и дома / на работе. Также многие коллеги из бигтеха используют её как справочник на работе.

2. Математика в огне. Джейсон Уилкс. – Отличное описание базовых математических понятий действительно простым языком, даже если вы полный ноль в математике.

3. Беклемишев, Д.В. Курс аналитической геометрии и линейной алгебры. – Есть задачки и можно свериться с тем, что выдаёт Ваша программа, например, в лабораторной работе №1.

4. Глейк Джеймс. Хаос. Создание новой науки – ***ОЧЕНЬ полезная книга***. Она не даст Вам ответов на то, как реализовать метод в лабораторной работе, но вполне **способна изменить Вашу жизнь и взгляды на мир вокруг.**

5. Любая другая книга по математике, которая Вам нравится. :)

37.2 Книги по программированию + вычислительной математике

1. Devpractice Team - Линеиная алгебра на Python. – Довольно базовая книга для начинающих в Python и кому нужно начать работать с линейной алгеброй.

2. Svein Linge, Hans Petter Langtangen - Programming for Computations: Python. – Отличная книга, которая может быть использована одновременно и как пособие для начинающих в Python и с наглядными примерами и объяснениями того, что и как реализуется в вычислительной математике.

3. Paul Orland – Math for Programmers. – Книга немного с другим фокусом, но она даёт хорошее описание того, как вообще можно работать с математикой в программировании, какие возникают задачи и как их принято решать. Примеры в основном на Python.

4. Ещё огромное количество книг про программирование и математику.

37.3 Ресурсы по программированию + математике

Часто студентам для реализации того или иного алгоритма не хватает именно навыка программирования и решения разного рода алгоритмических задач, включая математические задачи. Ниже приведён список ресурсов, на которых можно этот навык отточить и закрепить.

Иные из них часто используются и для проведения собеседований, так что потренироваться на них заранее будет очень даже полезно.

1. Project Euler — сайт представляет собой длинную череду задач, решение которых помогает освоиться как в математике, так и в программировании. Есть различные сайты с переводами задач на русский язык, например этот.

2. Hackerrank – на сайте представлены различные курсы и задачи в них, которые помогут Вам прокачаться в различных областях, будь

то DevOps, SQL или Java, Python, что угодно ещё. Решать задачи и запускать код можно на огромном количестве языков и прямо на сайте запускать. Также содержит задачки на математику.

3. binarysearch – сайт, который предоставляет отличную возможность решать задачи как самостоятельно, так и с друзьями. Есть дружелюбное русскоязычное комьюнити, а раз в неделю там проводят конкурсы.

4. codewars – сайт, похожий на Hackerrank и binarysearch, предлагающий решение задачек по различным языкам и технологиям.

5. Kaggle — отличный сайт для тех, кто хочет освоиться и попрактиковаться в Data Science! Куча практики, полезное общение и знакомство с другими участниками, которые пытаются решить те же задачи, а ещё возможность выиграть реальные и весьма немаленькие деньги.

6. codingame — вариант для тех студентов, которые пошли учиться на программистов и сутками не вылезают из компьютерных игр. На сайте программирование разных задачек является частью игры, при чём жанр игры можно выбрать самостоятельно и это вовсе не обязательно унылая графика и скучный сюжет. А ещё есть соревнования, где можно состязаться с другими участниками.

7. codebattle.hexlet.io — это, мягко выражаясь, настоящее "рубилово PvP от мира программирования. Можно соревноваться с ботом, друзьями или со случайными игроками.

Глава 38

Аппроксимация и интерполяция

Глава 39

Решение систем линейных уравнений

Глава 40

Решение нелинейных уравнений

Глава 41

Интегрирование

Глава 42

Решение дифференциальных уравнений

Глава 43

Рекомендации по чистому коду

1. D Bosuell, T Faucher - Читаемый код или программирование как искусство
2. Endi Oram, Gregori Uilson - Идеальная разработка ПО. Рецепты лучших программистов
3. Martin Fauler - Рефакторинг. Улучшение существующего кода
4. Stiv Makkonnell - Совершенный код
5. Б Керниган, Р Прайк - Практика программирования
6. Орам Э., Уилсон Г. Идеальный код (2011)
7. Р.Мартин - Чистый код
8. Vaysfeld - Объектно-ориентированное мышление
9. David S. Platt “Why software sucks... and what you can do about it”
10. Joel Spolsky “The best software writing I”
11. Brian W. Kernighan, Rob Pike “The Practice of Programming”

Глава 44

Дополнительная литература по курсу

1. Numerical Recipes: The Art of Scientific Computing by William H. Press
2. Numerical Recipes in C: The Art of Scientific Computing by William H. Press
3. Numerical Recipes: Example Book C by William T. Vetterling
4. Numerical Recipes in FORTRAN 77: The Art of Scientific Computing by William H. Press
5. Numerical Recipes in FORTRAN 90: The Art of Parallel Scientific Computing (FORTRAN Numerical Recipes, Volume 2) by William H. Press
6. Numerical Recipes: Example Book FORTRAN by William T. Vetterling
7. Numerical Methods That Work by Forman S. Acton
8. Numerical Methods by Germund Dahlquist
9. Numerical Methods and Software by David Kahaner
10. An Introduction to Numerical Methods in C++ by Brian Hilton Flowers
11. Introduction to Precise Numerical Methods [With CD] by Oliver Aberth
12. Precise Numerical Methods Using C++ [With CDROM] by Oliver Aberth
13. Introduction to Numerical Programming: A Practical Guide for Scientists and Engineers Using Python and C/C++ by Titus Adrian Beu
14. Numerical Linear Algebra by Lloyd N. Trefethen
15. Applied Numerical Linear Algebra by James W. Demmel
16. Numerical Methods for Scientists and Engineers by Richard Hamming

17. Numerical Methods and Optimization: A Consumer Guide by Eric Walter

18. Numerical Algorithms: Methods for Computer Vision, Machine Learning, and Graphics by Justin Solomon

19. Numerical Optimization by Jorge Nocedal

20. Elementary Numerical Mathematics for Programmers and Engineers (Compact Textbooks in Mathematics) by Gisbert Stoyan

21. Applied Numerical Methods in C by Shoichiro Nakamura

22. Numerical Methods in Economics by Kenneth L. Judd

23. Compact Numerical Methods for Computers by John C. Nash

24. Computer Solution of Linear Algebraic Systems by George E. Forsythe

25. Computer Methods for Mathematical Computations by George E. Forsythe

26. A First Course on Numerical Methods by Uri M. Ascher

27. Numerical Analysis with Algorithms and Programming by Santanu Saha Ray

28. Numerical Analysis for Statisticians by Kenneth Lange

29. Numerical Linear Algebra for Applications in Statistics by James E. Gentle

30. An Introduction to Numerical Methods and Analysis by James F. Epperson

31. Numerical Methods and Optimization: An Introduction by Sergiy Butenko

32. Computing for Numerical Methods Using Visual C++ by Shaharuddin Salleh

33. Numerical Linear Algebra: An Introduction by Holger Wendland

34. Introduction to Numerical Linear Algebra and Optimisation by Philippe G. Ciarlet

35. C++ and Object-Oriented Numeric Computing for Scientists and Engineers by Daoqi Yang

36. Numerical Methods and Optimization in Finance by Manfred Gilli

37. Computational Methods of Linear Algebra (Undergraduate Mathematics Books) by D.K. Faddeev

38. Computational Methods of Linear Algebra (3rd Edition) by Granville Sewell

39. Guide To Scientific Computing by Peter R. Turner

40. Guide to Scientific Computing in C++ by Joe Pitt-Francis

41. Practical Numerical Methods: Algorithms And Programs by Michael Kohn

42. Practical Numerical Algorithms for Chaotic Systems by Thomas S. Parker

43. Numerical Methods For Unconstrained Optimization And Nonlinear Equations by J.E. Dennis

44. Accuracy and Stability of Numerical Algorithms by Nicholas J. Higham

45. A First Course in Numerical Analysis by Anthony Ralston

46. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation by George E.M. Karniadakis

47. Functions of Matrices by Nicholas J. Higham

48. Scientific Computing by Michael T. Heath

49. Quadpack: A Subroutine Package For Automatic Integration by R. Piessens

50. Elementary Numerical Analysis: An Algorithmic Approach by Samuel Daniel Conte

51. A Survey of Numerical Mathematics: In Two Volumes - Volume I by David M. Young

52. A Survey of Numerical Mathematics: In Two Volumes - Volume II by David M. Young

53. Numerical Computation 1: Methods, Software, and Analysis by Christoph Uberhuber

54. Numerical Computation 2: Methods, Software, and Analysis by Christoph Uberhuber

55. Computational Integration by Arnold R. Krommer

56. Elementary Numerical Methods and C++ by Melvin R. Corley

57. Gaussian Quadrature Formulas by A.H. Stroud

58. Afternotes on Numerical Analysis by G.W. Stewart

59. Afternotes Goes to Graduate School: Lectures on Advanced Numerical Analysis by G.W. Stewart

60. Numerical Algorithms with C by Frank Uhlig

61. Numerical Algorithms With Fortran by Gisela Engeln-Mullges

62. Numerical Methods of Mathematics Implemented in Fortran by Sujit Kumar Bose

63. Numerical Optimization Of Computer Models by Hans-Paul Schwefel

64. Numerical Linear Algebra For High Performance Computers by Jack Dongarra

65. Gnu Scientific Library Reference Manual - Third Edition by Brian Gough

66. A Numerical Library in C for Scientists and Engineers by Hang Tong Lau

67. A Numerical Library In Java For Scientists And Engineers by Hang Tong Lau

Часть V

Приложения

Приложение А

Чистый и Математический код

Понятие чистого кода (Clean Code) важно для ваших навыков написания кода, и оно становится еще более важным, когда вы работаете над сложным, трудно понятным математическим кодом. Но сначала давайте дадим определение математическому коду.

А.1 Математический код

При разработке на Python одной из наиболее популярных библиотек для работы с математикой является NumPy. Быстрое преобразование Фурье – это один из фундаментальных и наиболее значимых алгоритмов в Информатике. (Подробнее про значимость алгоритма можно посмотреть видео (рус / eng): `https://youtu.be/eQlSvfUuQNs` / `https://youtu.be/nmgFG7PUHfo`). В библиотеке NumPy быстрое преобразование Фурье реализовано следующим образом (`https://github.com/numpy/numpy/blob/main/numpy/fft/_pocketfft.py#L49`) :

```
# `inv_norm` is a float by which the result of the
# transform needs to be divided. This replaces the
# original, more intuitive 'fct` parameter to avoid
# divisions by zero (or alternatively additional
# checks) in the case of zero-length axes during
# its computation.
def _raw_fft(a, n, axis, is_real, is_forward, inv_norm):
    axis = normalize_axis_index(axis, a.ndim)

    if n is None:
        n = a.shape[axis]

    fct = 1/inv_norm

    if a.shape[axis] != n:
        s = list(a.shape)
```

```
        index = [slice(None)]*len(s)
        if s[axis] > n:
            index[axis] = slice(0, n)
            a = a[tuple(index)]
        else:
            index[axis] = slice(0, s[axis])
            s[axis] = n
            z = zeros(s, a.dtype.char)
            z[tuple(index)] = a
            a = z

    if axis == a.ndim-1:
        r = pfi.execute(a, is_real, is_forward, fct)
    else:
        a = swapaxes(a, axis, -1)
        r = pfi.execute(a, is_real, is_forward, fct)
        r = swapaxes(r, axis, -1)

    return r
```

Данный код содержит как достоинства, так и недостатки. Для сравнения можно выбрать код быстрого преобразования Фурье (`https://www.kurims.kyoto-u.ac.jp/~ooura/fft.html`) написанный на C и Fortran. Выберем для примера реализацию на C одной лишь функции без используемых функций:

```
void ddst2d(int n1, int n2, int isgn, double **a, double **t,
    int *ip, double *w)
{
    void makewt(int nw, int *ip, double *w);
    void makect(int nc, int *ip, double *c);
    void bitrv2col(int n1, int n, int *ip, double **a);
    void bitrv2row(int n, int n2, int *ip, double **a);
    void cftbcol(int n1, int n, double **a, double *w);
    void cftbrow(int n, int n2, double **a, double *w);
    void cftfcol(int n1, int n, double **a, double *w);
    void cftfrow(int n, int n2, double **a, double *w);
    void rftbcol(int n1, int n, double **a, int nc, double *c);
    void rftfcol(int n1, int n, double **a, int nc, double *c);
    void dstbsub(int n1, int n2, double **a, int nc, double *c);
    void dstfsub(int n1, int n2, double **a, int nc, double *c);
    int n, nw, nc, n1h, n2h, i, ix, ic, j, jx, jc;
    double xi;

    n = n1 << 1;
    if (n < n2) {
        n = n2;
```

```
    }
    nw = ip[0];
    if (n > (nw << 2)) {
        nw = n >> 2;
        makewt(nw, ip, w);
    }
    nc = ip[1];
    if (n1 > nc || n2 > nc) {
        if (n1 > n2) {
            nc = n1;
        } else {
            nc = n2;
        }
        makect(nc, ip, w + nw);
    }
    n1h = n1 >> 1;
    n2h = n2 >> 1;
    if (isgn >= 0) {
        for (i = 0; i <= n1 - 1; i++) {
            for (j = 1; j <= n2h - 1; j++) {
                jx = j << 1;
                t[i][jx] = a[i][j];
                t[i][jx + 1] = a[i][n2 - j];
            }
        }
        t[0][0] = a[0][0];
        t[0][1] = a[0][n2h];
        t[n1h][0] = a[n1h][0];
        t[n1h][1] = a[n1h][n2h];
        for (i = 1; i <= n1h - 1; i++) {
            ic = n1 - i;
            t[i][0] = a[i][0];
            t[ic][1] = a[i][n2h];
            t[i][1] = a[ic][0];
            t[ic][0] = a[ic][n2h];
        }
        dstfsub(n1, n2, t, nc, w + nw);
        if (n1 > 2) {
            bitrv2row(n1, n2, ip + 2, t);
        }
        cftfrow(n1, n2, t, w);
        for (i = 0; i <= n1 - 1; i++) {
            t[i][1] = 0.5 * (t[i][0] - t[i][1]);
            t[i][0] -= t[i][1];
        }
        if (n2 > 4) {
```

```
            rftfcol(n1, n2, t, nc, w + nw);
            bitrv2col(n1, n2, ip + 2, t);
        }
        cftfcol(n1, n2, t, w);
        for (i = 0; i <= n1h - 1; i++) {
            ix = i << 1;
            ic = n1 - 1 - i;
            for (j = 0; j <= n2h - 1; j++) {
                jx = j << 1;
                jc = n2 - 1 - j;
                a[ix][jx] = t[i][j];
                a[ix][jx + 1] = -t[i][jc];
                a[ix + 1][jx] = -t[ic][j];
                a[ix + 1][jx + 1] = t[ic][jc];
            }
        }
    } else {
        for (i = 0; i <= n1h - 1; i++) {
            ix = i << 1;
            ic = n1 - 1 - i;
            for (j = 0; j <= n2h - 1; j++) {
                jx = j << 1;
                jc = n2 - 1 - j;
                t[i][j] = a[ix][jx];
                t[i][jc] = -a[ix][jx + 1];
                t[ic][j] = -a[ix + 1][jx];
                t[ic][jc] = a[ix + 1][jx + 1];
            }
        }
        if (n2 > 4) {
            bitrv2col(n1, n2, ip + 2, t);
        }
        cftbcol(n1, n2, t, w);
        if (n2 > 4) {
            rftbcol(n1, n2, t, nc, w + nw);
        }
        for (i = 0; i <= n1 - 1; i++) {
            xi = t[i][0] - t[i][1];
            t[i][0] += t[i][1];
            t[i][1] = xi;
        }
        if (n1 > 2) {
            bitrv2row(n1, n2, ip + 2, t);
        }
        cftbrow(n1, n2, t, w);
        dstbsub(n1, n2, t, nc, w + nw);
```

```
        for (i = 0; i <= n1 - 1; i++) {
            for (j = 1; j <= n2h - 1; j++) {
                jx = j << 1;
                a[i][j] = t[i][jx];
                a[i][n2 - j] = t[i][jx + 1];
            }
        }
        a[0][0] = t[0][0];
        a[0][n2h] = t[0][1];
        a[n1h][0] = t[n1h][0];
        a[n1h][n2h] = t[n1h][1];
        for (i = 1; i <= n1h - 1; i++) {
            ic = n1 - i;
            a[i][0] = t[i][0];
            a[i][n2h] = t[ic][1];
            a[ic][0] = t[i][1];
            a[ic][n2h] = t[ic][0];
        }
    }
}
```

Из примеров выше можно заключить, что математический код в действительности трудно читается: есть много переменных с неясным назначением и производятся манипуляции со значениями, назначение которых не очевидно для читателя кода. Однако, математический код – это всегда поиск баланса между «чистотой кода» – удобностью и скоростью его чтения другими людьми – и скоростью его работы. Среди других параметров, на которые стоит обращать внимание при написании математического кода, можно выделить, например, контроль за некорректными значениями (Вы должны решить, что будете делать, в случае деления на 0 или отрицательных чисел под знаком логарифма) и потребляемая память.

A.2 "Запахи"плохого кода

Разговора о чистом коде, как правило, начинают с “запахов” кода. Они называются “запахи”, потому что, когда вы видите такой код, вы непроизвольно морщите нос, как будто чувствуете его неприятный плохой запах. Такое представление о том, как должен выглядеть код, приходит с опытом программирования и чистого кода. “Запахи” плохого кода могут быть классифицированы по уровням.

Уровень приложения

Первый и самый общий уровень — это уровень приложения. Первая причина, почему эти запахи вынесены на уровень приложения заключается в том, что как правило такая проблема имеет массовый, а не единичный

характер. Вторая причина заключается в том, что они могут быть представлены на уровне всего приложения, а не только в одном классе или методе.

Например, первый "запах" — это дублирование кода. Это означает, что у вас есть две или более функций, классов и т. д. которые реализуют одни и те же вещи, одну и ту же функциональность. Если вы реализуете что-то дважды, это означает одно из двух: либо вы знаете, что ваша кодовая база плоха, либо есть какая-то причина, по которой вы не можете использовать свою первую функцию. В обоих случаях это означает, что что-то может быть не так, и мы обнаруживаем это как запах кода. Дублирование кода - плохая практика. В любом случае вы должны стараться избегать этого, потому что в будущем это может привести ваше приложение к более трудноразрешимым проблемам. Например, вам может потребоваться изменить некоторое поведение дублированной функции. Если у вас есть несколько копий функций, вы можете пропустить некоторые из них, потому что вам нужно помнить все места, где находятся дубликаты фрагменты кода.

Следующий пример плохого "запаха" на прикладном уровне — это надуманная сложность. Мы уже говорили об этом в разделе о математических навыках программистов. В любом случае вы должны попытаться упростить свое решение и код. Если вы напишете что-то слишком сложное, то это будет трудно понять вам, вашим коллегам, и снова вам самим позже. В таком коде будет трудно исправить дефект или даже добавить новую функциональность.

Уровень класса

Меньший по охвату уровень — это уровень "запахов" классов.

Первый "запах" этого уровня — это **большие классы**. Иногда вы открываете файл с кодом и прокручиваете его вниз снова и снова без остановки, как будто этот файл или класс бесконечен. Это проблема, потому что ваша мысленная память ограничена, поэтому трудно удержать в уме полный длинный класс. Это может привести к дублированию кода или иным проблемам. Когда Вы видите такой класс, вы должны попытаться понять причину его размера. Вероятно, когда-то давно он был маленьким, и теперь вы просто должны разделить его. Или, возможно, он содержит слишком длинные методы. И скоро мы поговорим о длинных методах.

Далее идут **жадные объекты** (объекты зависти). Это когда класс содержит методы, которые широко используют другой класс. Подумайте о переносе этого метода в класс, которому он так завидует.

Неуместная близость — это когда два класса слишком много знают друг о друге. Вы должны попытаться реализовать свои классы независимым способом, чтобы легко изменять или заменять их как блоки. Это дает вам гибкость при разработке продукта.

Далее следует **отсутствие зависимостей**. Если вы наследуете от класса, но никогда не используете какую-либо унаследованную функциональность, действительно ли вы должны использовать наследование?

Следующая проблема очень популярна особенно среди некоторых молодых и амбициозных специалистов - **ленивые классы**. Каждый дополнительный класс сам по себе увеличивает сложность проекта. Если у вас есть класс, который делает недостаточно, чтобы окупить себя, можно ли его функциональность перенести в другой класс?

Далее следует **чрезмерное использование литералов** (строковых констант). Буквальный перевод слова «литерал» — это строковая константа. Если вы помещаете одну и ту же строковую константу несколько раз в один класс, вам следует уменьшить ее до именованного константы класса. Популярным примером являются сообщения об исключениях или ошибках. Кстати, это также упрощает перевод вашего приложения на другой язык или внесение изменений в сами строковые константы, т. к. фактически у вас нет дублированных значений.

Цикломатическая сложность означает, сколько вложенных конструкций if-then-else или equal у вас есть в одном блоке кода. Таких конструкций не должно быть слишком много. Если это всё же необходимо, то рекомендуется переместить некоторые блоки в отдельные методы или даже класс. Вынесение в отдельные именованные методы и классы поможет вам и другим понять причину (при условии, что имя будет соответствовать содержимому) такой глубокой обработки.

Нисходящее приведение типов означает, когда вы вручную приводите объект родительского класса к какому-либо унаследованному. Это плохо, потому что у вас нет гарантии, что после некоторых изменений вы не получите другой унаследованный объект, а не тот, который вы ожидаете.

Наконец, **классы констант**. Это класс, который состоит только из констант, и больше никакой функциональности в нем нет. Эти классы могут быть легко заменены файлом конфигурации, файлом пакета или перечислением.

Уровень метода

И наименьшие по охвату запахи уровня метода. Первый такой запах — это когда метод имеет слишком много параметров. Это плохо, потому что вы можете допустить ошибку при вводе аргумента в функции: неправильный порядок или пропущенные значения. Чтобы решить ее, вам следует ознакомиться с этим списком. Вероятно, вы никогда не используете некоторые из них, или вы можете получить это внутри метода. Если у вас все еще слишком много параметров, вы можете создать класс для их объединения. Но не стоит забывать и о запахах классного уровня.

Следующая проблема заключается в том, что ваш метод слишком длинный. Это плохо, потому что становится трудно понять, что происходит внутри, и возникает соблазн написать комментарий. Но вы должны разделить этот метод на несколько из них с осмысленными именами вместо написания комментариев. Разделение методов гораздо более полезно для будущих изменений. С моей точки зрения, маленьких методов не суще-

ствует. Даже если у вас есть код в одну строку, можно перенести его в отдельный метод. Это намного лучше, чем наличие какого-то гигантского логического условия в исходном методе.

Слишком длинные и слишком короткие идентификаторы. Мы подробнее поговорим об именовании на одном из следующих слайдов.

Чрезмерный возврат данных. Один метод не должен возвращать слишком много разных данных, потому что один метод должен решать только одну задачу. Если он возвращает несколько переменных, то вам следует просмотреть его и подумать о том, что это такое? Вероятно, вам следует объединить их в классе, или вам следует разделить методы и просмотреть некоторые другие части кода.

А.3 Комментарии к коду

Следующая тема, на которую хотелось бы обратить ваше внимание, — это комментарии к коду. В большинстве случаев вам не нужно использовать комментарии в вашем коде. Здесь вы можете увидеть длинный список таких случаев плохих комментариев. Причина в том, что они могут быть устаревшими, избыточными, сбивающими с толку и так далее. Вам также не следует комментировать какую-либо часть кода, если вы можете ее удалить. Если вам снова понадобится этот код, вы можете написать его снова или использовать систему контроля версий для его восстановления. Однако никто, кроме вас, не удалит закомментированный код. Для других программистов будут возникать вопросы “а вдруг в этом есть смысл” и “а вдруг это действительно важно”.

Но гораздо легче запомнить случаи, когда полезно иметь комментарии. Прежде всего, это **юридические комментарии**. Не забывайте об этом, особенно когда вы собираетесь поделиться с кем-то кодом или продуктом.

Хорошим примером **информативных комментариев** является формат даты, потому что иногда его трудно понять по шаблону регулярных выражений, который часто отличается в разных языках программирования.

Объяснение намерения — это когда вы помещаете в комментарий некоторое объяснение того, почему вы что-то делаете, например, почему вы возвращаете 1 из метода compareTo.

Поясняющие комментарии помогают нам увидеть, что вы делаете, если язык программирования или другая причина мешает вам показать это. Например, когда вы пишете какое-то длинное сравнение, вы можете поместить в комментарии некоторое математическое определение вашего кода.

Предупреждение о последствиях — это когда вы делаете одни и те же заметки об использовании некоторых частей кода. Например, почему кто-то не должен изменять какую-то функцию или что произойдет, если это будет сделано.

Комментарии **TODO и FIXME** очень полезны для обозначения того, что должно быть сделано или исправлено. Вы можете оставить такие комментарии, когда только начинаете реализовывать какую-то функцию, тогда вам следует удалить все такие комментарии перед мержем (слиянием) вашей ветки для разработки. Или вы можете отложить это на будущее некоторые доработки, потому что вы не можете исправить это прямо сейчас. В таких случаях я всегда рекомендую указать идентификатор задачи для ее решения. Затем у вас возникнет конкретная проблема в task tracker, чтобы исправить эту ошибку или технический долг. По этому номеру всегда можно будет отследить, были ли внесены изменения и можно ли удалить этот комментарий из кода.

Комментарии для усиления (амплификация) помогут вам обратить внимание читателя кода на некоторое важное обстоятельство, которое может выглядеть незначительно. Например, почему вы вызываете там какую-то функцию? Вероятно, нам это вообще не нужно. Комментарий может подчеркивать важность обстоятельства, которое на первый взгляд кажется несущественным.

Документация для публичных функций — это крайне полезная вещь для взаимодействия людей внутри команда и между командами выполняя роль контракта на взаимодействие. Документация публичных функций может быть использована такими системами сборки документации как Swagger или любые другие инструменты, которые создадут руководство пользователя для вашего кода или продукта. Это также очень полезно для новых членов вашей команды.

A.4 Присвоение имен

Следующий раздел посвящен рассмотрению при выборе хороших названий. Наиболее важным соображением при присвоении имени переменной является то, что имя должно полностью и точно описывает сущность, которую представляет. Эффективный способ придумать хорошее название — это сформулировать словами, что представляет переменная. Если вы не можете дать, например, переменной осмысленное имя, значит, вы не понимаете, что это значит, и, вероятно, вам не нужна эта переменная. На этом слайде вы можете увидеть несколько примеров хороших и плохих названий в зависимости от назначения переменных.

Еще одно важное правило для присвоения имени заключается в том, что имена методов всегда будут глаголами, а имена переменных и классов всегда будут существительными. Потому что они являются действующими лицами, субъектами, и они изменяют значение переменных с помощью методов, таких как предикаты.

Важным правилом в длине имени является то, что чем больше область видимости объекта именования (переменная, метод, класс и пр.), тем более подробным должно быть ее имя.

А.5 Ввод данных и дополнительная литература для него

Ввод данных очень важен, потому что это место, где ваш продукт взаимодействует с вашими пользователями. Ваши пользователи ничего не знают о вашем продукте и ваших ожиданиях. Здесь три пункта действия.

1. Проверьте значения всех данных из внешних источников. Но в то же время настоятельно требуется проверять данные из ваших собственных источников, с помощью частных методов. например.

2. Проверьте значения всех стандартных входных параметров.

3. Решите, как обрабатывать неверный ввод. Например, вам не следует выводить stack trace возникшего исключения на веб-сайт кофейни.

Ввод данных и взаимодействие с пользователем настолько важны, что я подготовила еще один список литературы, если вас это заинтересует. Опять же, это только для вас, и для нашего курса не обязательно его читать. От Вас требуется лишь общее понимание текущего раздела и применение ее в лабораторных работах.

1. David S. Platt “Why software sucks... and what you can do about it” / Дэвид С. Платт «Софт отстой и что с этим делать»

2. Joel Spolsky “The best software writing I” / Джоэл Спольски «Лучшие примеры разработки ПО»

3. Brian W. Kernighan, Rob Pike “The Practice of Programming” / Керниган Б., Пайк Р. "Практика программирования"

А.6 Тестирование математического кода

Следующая тема — тестирование математического кода или как быть уверенным, что вы написали код правильно. Существуют различные виды тестирования.

Прежде всего, это **модульные тесты**. Вы можете протестировать с его помощью свои методы и использовать их, когда практикуете разработку, основанную на тестировании. Обычно проекты содержат наибольшее количество модульных тестов.

Более сложные тесты — это **интеграционные** и **системные** тесты, они проверяют, работает ли какая-то функциональность, фича, которая содержит внутри ряд методов. Обычно они тестируют всю историю пользователя или больше.

Smoke-тестирование используется, когда у вас большое количество тестов и они выполняются слишком медленно. Набор smoke-тестов — это некоторая подборка тестов, которая помогает нам быть уверенными в том, что основная функциональность нашего продукта по-прежнему работает.

Полезно запускать smoke-тесты во время разработки некоторой функциональности, а непосредственно перед объединением для разработки ветки вы должны запустить весь объем тестов.

Нагрузочное тестирование поможет вам найти границы, в которых ваше приложение работает нормально. Например, сколько пользователей вы можете обработать. Что произойдет, если пользователей станет больше? Вернется ли ваше приложение к работе, если количество пользователей сократится? Потому что бывает, что после нагрузки не очищают какие-нибудь промежуточные буферы, кеши. Какая операция заняла слишком много времени и так далее. Очень желательно улучшить ваше приложение и знать, как оно работает.

Приемочное тестирование помогает вам быть уверенным даже не в том, что вы реализовали что-то правильно, а в том, что вы реализовали правильные вещи. Это тестирование вашего приложения в соответствии с требованиями.

A.7 Оптимизация математического кода

И последняя тема, которая на самом деле не связана с чистым кодом, но она относится к тому, как вы пишете свой код. И когда я просматриваю код студента и когда я вижу некоторые плохие методы работы с кодом, я спрашиваю студента: “Почему ты это сделал?” Очень популярный ответ заключается в том, что это для оптимизации. Но на самом деле, они не могут ответить мне, что такое оптимизация и какая выбранная функция оптимизации. Прежде всего, вы должны знать, что вы делаете и почему, и только после этого приступать к какой-либо оптимизации. Но на этом слайде я попытался собрать основные методы оптимизации. Общее правило заключается в том, что вы должны применять эти техники сверху вниз и после каждого ряда спрашивать себя, достаточно ли этого сейчас? Вероятно, вам следует прекратить его оптимизацию.

- Оптимизация логических выражений. Конечно, вы должны попытаться упростить это. Другой способ поместить некоторые условия if внутри другого оператора then.

- Циклы. Прежде всего, вы должны проверить, что у вас нет эквивалентных прогонов через массивы. Вероятно, они могут быть собраны в один цикл. Затем посмотрите, может быть, вы сможете раскатывать петли. Или используйте некоторые методы уменьшения размера карты.

- Преобразования данных. Проверьте, как с ним работать. Например, если у Вас часто изменяемые строки в Java, можно использовать StringBuffer / StringBuilder вместо String.

- Максимизация размера метода. Это плохо для чистого кода, но это четвертый уровень оптимизации кода.

- Переписывание кода на более низкоуровневый язык программирования. Это последний способ оптимизировать код. Не начинайте с этого. Едва ли Вы сможете написать более высокоэффективное приложение на C, чем на Java или Python, если никогда прежде не писали на C. Трансляторы наверняка справятся с этим лучше.

Приложение B

Матричная алгебра

Сложно переоценить влияние матриц и линейной алгебры на современные технологии. В абсолютном большинстве случаев для работы с ними используются численные методы. В данном разделе мы не будем повторять весь курс Математики, который у Вас уже был, но эти знания необходимы для того, чтобы было проще разобраться в следующих разделах. Поэтому в этой части курса рассмотрим кратко основные минимально необходимые сведения. Подробнее разобраться и освежить знания можно во время выполнения домашнего задания до и после лекции.

Множество mn чисел (вещественных или комплексных), записанных в виде прямоугольного массива из m строк и n столбцов

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

называется матрицей. Числа $a_{ij} (i = 1, 2, \dots, m; j = 1, 2, \dots, n)$, которые составляют матрицу, называются элементами матрицы.

Если $m = n$, тогда это квадратная матрица порядка n, иначе - прямоугольная.

Матрица вида

$$A = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_n \end{bmatrix}$$

Называется диагональной и короче записывается как вектор $[\alpha_1, \alpha_2, \dots, \alpha_n]$. Если $\alpha_i = 1 (i = 1, 2, \dots, n)$, матрица называется единичной и обозначается буквой E (или I - Identity).

Треугольная (супер-диагональная) если и только тогда, когда при $i > j$ $a_{ij} = 0$.

Строго треугольная если и только тогда, когда при $i \geq j$ $a_{ik} = 0$.

Диагональная если и только тогда, когда при $i \neq i$ $a_{ij} = 0$.

Мономиальная тогда и только тогда, когда каждая строка и столбец содержат один и только один элемент, отличный от нуля.

Нулевая матрица – матрица, все элементы которой равны 0. Обозначается как 0 или 0_{mn}.

B.1 Определитель матрицы и его свойства

С квадратной матрицей $A = [a_i j]_{n,n}$ связано понятие определителя.

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

Это два разных понятия: матрица — это упорядоченный набор чисел, записанный в виде прямоугольного массива; его определитель, $\det A$.

Определитель — это специальное число, которое определяется только для квадратных матриц (множественное число для матрицы). Квадратная матрица имеет одинаковое количество строк и столбцов.

Определитель используется для определения того, может ли матрица быть инвертирована или нет, он полезен при анализе и решении одновременных линейных уравнений (правило Крамера), используется в исчислении, используется для определения площади треугольников (если заданы координаты) и многое другое. Определитель матрицы A обозначается $|A|$ или $\det(A)$.

Для нахождения значения определителя может быть полезно знать неравенство Адамара:

$$|D|^2 \leq \prod_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2$$

Важность понятия определителя квадратной матрицы можно увидеть по длинному списку свойств определителя.

1. Определитель, вычисляемый в любой строке или столбце, одинаков.
2. Если все элементы строки (или столбца) равны нулям, то значение определителя равно нулю.
3. Определитель единичной матрицы $(I_n) = 1$
4. Если строки и столбцы меняются местами, то значение определителя остается прежним (значение не меняется). Следовательно, $\det A = \det A^T$, где $\det A^T$ транспонированная матрица A.
5. Если любые две строки (или два столбца) определителя меняются местами, значение определителя умножается на -1.
6. Если все элементы строки (или столбца) определителя умножить на некоторое скалярное число k, значение нового определителя равно k раз данного определителя. Следовательно, если A квадратная матрица с n рядами и K – любой скаляр, то $|KA| = K^n|A|$.

7. Если две строки (или столбцы) определителя идентичны, значение определителя равно нулю.

8. Пусть A и B две матрицы, тогда $\det(AB) = \det(A) * \det(B)$.

9. Если A матрица, то $|An| = (|A|)n$.

10. Определитель обратной матрицы может быть определен как $|A^{-1}| = \frac{1}{|A|}$.

11. Определитель диагональной матрицы, треугольная матрица (верхняя треугольная или нижняя треугольная матрица) является произведением элемента основной диагонали.

12. В определителе каждый элемент в любой строке (или столбце) состоит из суммы двух членов, тогда определитель может быть выражен как сумма двух определителей одного и того же порядка. Например:

$$\begin{vmatrix} a_1 + x_1 & b_1 & c_1 \\ a_2 + x_2 & b_2 & c_2 \\ a_3 + x_3 & b_3 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} + \begin{vmatrix} x_1 & b_1 & c_1 \\ x_2 & b_2 & c_2 \\ x_3 & b_3 & c_3 \end{vmatrix}$$

13. Если B получается путём сложения c-кратной строки A с другой строкой, то $\det B = \det A$

14. Если A - матрица, то:

$$|adj(A)| = (|A|)^{n-1}$$

$$|adj(adj(A))| = |A|^{(n-1)*(n-1)}$$

где $adj(A)$ - является сопряжённой матрицей A.

15. Если значение определителя Δ становится равным нулю путем замены $x = \alpha$, то $x - \alpha$ коэффициент Δ.

16. Обозначим c_{ij} кофактор элементов a_{ij} в Δ.

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$$

$$\Delta_1 = \begin{vmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{vmatrix} = \Delta^2$$

17. В определителе сумма произведения элементов любой строки (или столбца) с коэффициентами соответствующих элементов любой другой строки (или столбца) равна нулю. Например,

$$d = a_{i1} * Aj1 + a_{i2} * A_{j2} + a_{i3} * A_{j3} + \cdots + a_{in} * A_{jn}$$

, где $A_{j1}, A_{j2}, A_{j3}, \ldots, A_{jn}$ - кофакторы вдоль элементов j-й строки.

18. Пусть $\lambda_1, \lambda_2, \ldots, \lambda_n$ - собственные значения A (квадратная матрица порядка n). Тогда $\det(A) = \lambda_1 * \lambda_2 * \cdots * \lambda_n$ произведение собственных чисел.

В.2 Элементарные матричные преобразования

Две матрицы называются эквивалентными, если одна получается из другой с помощью конечного числа элементарных преобразований. Они не равны, но имеют одинаковый ранг. Напомним, что рангом матрицы называется максимальное число линейно независимых строк (столбцов).

К элементарным матричным преобразованиям относятся:

1. Замена двух строк или столбцов.
2. Умножение всех элементов одной строки (столбца) на одно и то же ненулевое число (скаляр).
3. Добавление кратных элементов строки (столбца) к элементам другой строки (столбца).

Приложение C

Основные постулаты теории алгоритмов

В ходе работы над реализацией численных методов в виде программ необходимо основываться на разработке алгоритмов.

C.1 Свойства алгоритма

1. **Массовость**
   может выполняться любое количество раз для разных входных данных.

2. **Определённость**
   Для одного и того же ввода должна каждый раз давать один и тот же результат.

3. **Дискретность**
   Должен состоять из отдельных шагов или команд.

4. **Корректность**
   Должен возвращать правильное решение для всех допустимых входных данных.

5. **Конечность**
   Процесс должен быть завершен после конечного числа шагов.

6. **Ясность**
   Должен включать только такие команды, которые известны исполнителю алгоритма.

C.2 Алгоритмическая сложность

1. Если алгоритм выполняет определенную последовательность шагов $f(N)$ раз для математической функции f, он выполняет $O(f(N))$ шагов.

2. Если алгоритм выполняет операцию, которая занимает $O(f(N))$ шагов, а затем выполняет вторую операцию, которая занимает $O(g(N))$ шагов для функций f и g, общая производительность алгоритма равна $O(f(N) + g(N))$.

3. Если алгоритм принимает $O(f(N) + g(N))$, а функция $f(N)$ больше, чем $g(N)$ для больших N, производительность алгоритма может быть упрощена до $O(f(N))$.

4. Если алгоритм выполняет операцию, которая занимает $O(f(N))$ шагов, и для каждого шага в этой операции он выполняет еще $O(g(N))$ шагов, общая производительность алгоритма равна $O(f(N) \times g(N))$.

5. Игнорируйте постоянные кратные числа. Если C является константой, $O(C \times f(N))$ совпадает с $O(f(N))$, а $O(f(C \times N))$ совпадает с $O(f(N))$.

Приложение D

Элементы теории оптимизации

D.1 Задача оптимизации

Оптимизация – поиск минимума/ максимума функции. Это процесс, имеющий целью направить развитие какого-либо объекта или метода к наилучшему состоянию в соответствии с целевой функцией.

Параметры оптимизации - переменные, от которых зависит функция, и которые меняются в процессе оптимизации.

- $f(x_1, x_2, x_3) -> min$
- Функция f(x) называется целевой функцией, а вектор x – независимой переменной.
- Элементы вектора x – признаки решения.
- Число элементов вектора x – размерность задачи.

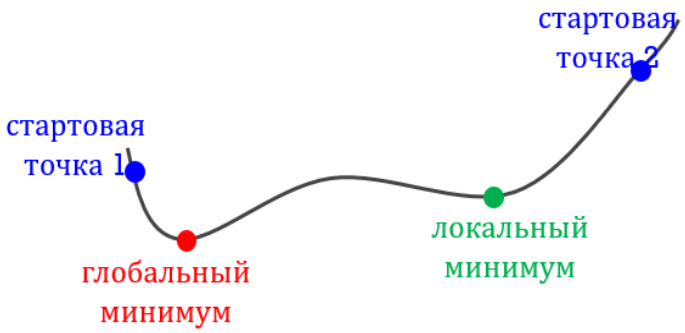
Ограничения - допустимые значения параметров.

- $x_1 > 0$ (например, толщина).
- $x_2 + x_3$ - четное число (например, в полиномах Цернике).
- **Безусловная (неограниченная) оптимизация** – нет никаких ограничений.
- **Условная (ограниченная) оптимизация** - заданы какие-то ограничения.

D.2 Локальный и глобальный минимум

- **Локальный минимум** – минимум функции в некотором диапазоне
- **Глобальные минимум** – минимум на всей области существования функции

- Стартовая точка – начальные значения всех параметров
  - От стартовой точки может зависеть скорость достижения минимума и достижение локального либо глобального максимума



D.3 Оценочная функция

- Оптимизация сводится к нахождению минимума или максимума целевой функции.
  - $\min\limits_x f(x) \Rightarrow f(x)$ называется стоимостью (cost) или целевым критерием;
  - $\max\limits_x f(x) \Rightarrow f(x)$ называется приспособленностью (fit / fitness) или целевым критерием;
- Что делать если надо найти $f \to const$, например $f \to 1,5$?
  - оценочная функция $|f - 1.5| \to min$
- Любой алгоритм для минимизации может быть использован для максимизации и наоборот.

$$\min\limits_x f(x) \Leftrightarrow \max\limits_x [-f(x)] \qquad \text{и} \qquad \max\limits_x f(x) \Leftrightarrow \min\limits_x [-f(x)]$$

D.4 Классификация методов оптимизации

- Методы **нулевого порядка**, использующие информацию только о значениях функции
- Методы **первого порядка**, использующие информацию о значениях самой функции и ее первых производных

  - методы наискорейшего градиентного спуска, дробления шага, Гаусса-Зейделя, Флетчера-Ривса

- Методы **второго порядка**, использующие, кроме того, и информацию о вторых производных функции
  - метод Ньютона и его модификации
  - **Одномерные методы** – у оптимизируемая функция зависит только от одной переменной

- **Многомерные методы** – у функция зависит от нескольких переменных
  - **Дискретная оптимизация** (дискретное программирование) – задачи, в которых переменные могут принимают только дискретные значения
  - Например, целочисленные
  - Например, марки стекол при оптимизации оптических систем

- **Многокритериальная оптимизация** – одновременно минимизируется несколько функций, иногда можно задать через однокритериальную оптимизацию:

$$(f_1 + f_2)/2 \to min$$

- **Мультимодальная оптимизация**: задача, имеющая более одного локального минимума, но найти всё ещё требуется глобальный минимум.

- **Комбинаторная оптимизация**: оптимизация для функций, представленных множеством дискретных значений. (Например, задача о коммивояжёре или рюкзаке).

- **Генетические алгоритмы (GA)** – используют методы, схожие с естественными генетическими процессами принципы для нахождения наилучших значений приспособленности. (Эволюция в природе и в алгоритмах – это нахождение особей с наилучшими значениями функции приспособленности).
  - Особь – набор значений аргументов (признаков решений).
  - Отбор – выбор особей с наибольшими функциями приспособленности на основе теории вероятностей.
  - Скрещивание – отбор двух особей и случайная комбинация их признаков.
  - Мутация – случайная замена значений признаков (не обязательно на те, что у родителей).

D.5 Более поздние эволюционные алгоритмы

- Симулирование закаливание – SA (simulated annealing)
- Оптимизация на основе муравьиной кучи - ACO (ant colony optimization)
- Оптимизация на основе роя частиц - PSO (Particle swarm optimization)
- Дифференциальная эволюция – DE (differential evolution)
- Алгоритмы оценивания вероятностных распределений – EDA (estimation-of-distribution algorithm)
- Биогеографическая оптимизация – BBO (biogeography-based optimization)
- Культурные алгоритмы – ACM (adaptive cultural model)
- Оппозиционное самообучение – OBL (opposition-based learning)
- Табуированный поиск – TS (tabu search)
- Алгоритм искусственного косяка рыб – AFSA (artificial fish swarm algorithm)
- Оптимизатор на основе группового поиска – GSO (group search algorithm)
- Алгоритм перемещения лягушачьих прыжков – SFLA (shuffled frog learning algorithm)
- Светлячковый алгоритм – GSO (glowworm / firefly search optimization)
- Оптимизация на основе бактериальной кормодобычи – BFOA (bacterial foraging optimization algorithm)
- Алгоритм искусственной пчелиной семьи – ABC (artificial bee colony)
- Алгоритм гравитационного поиска – GSA (gravity search algorithm)
- Поиск гармонии – HS (harmony search)
- Оптимизация по принципу учитель-ученик – TLBO (teaching-learning-based algorithm)

D.6 Чем я занимаюсь в кандидатской диссертации

В Лаборатории ИТМО COSM (наша группа vk) я занимаюсь изучением способов оптимизации размещения виртуальных машин на серверах датацентров.

Например, Вы — компания Amazon, у которой есть серверы в различных зонах с различным оборудованием и пр. На этом оборудовании размещаются виртуальные машины как Ваши, так и заказчиков. Все оборудование обладает параметрами объема RAM, HDD/SDD и количеством CPU. Эти же ресурсы необходимы для выделения виртуальных машин.

Перед Вами стоит задача оптимизации тех или иных параметров, например: выключить как можно больше оборудования в штуках, максимально снизить энергопотребление, снизить расходы на охлаждение и пр. Для этого существуют алгоритмы, решающие задачу, близкую к NP-сложной упаковке рюкзака.

Оптимизация может быть выполнена для существующего размещения виртуальных машин (статически) и для новых запускаемых виртуальных машин (динамически). Миграция виртуальных машин также занимает время. Виртуальные машины могут выключаться.

Если интересно поучаствовать – пишите.

Список иллюстраций

Список таблиц