Содержание

Об авторе

Майк Икс Коэн – адъюнкт-профессор неврологии[1] в Институте Дондерса (Медицинский центр Университета Радбуда) в Нидерландах. Имеет более чем 20-летний опыт преподавания научного программирования, анализа данных, статистики и смежных тем, а также является автором нескольких онлайновых курсов[2] и учебников. У него подозрительно сухое чувство юмора, и ему нравится все фиолетовое.

[1] См. https://oreil.ly/Ee23F.

[2] См. https://oreil.ly/BurUH.

Предисловие

Условные обозначения в книге

В книге используются следующие типографические условные обозначения:

курсивный шрифт
: обозначает новые термины, URL-адреса, адреса электронной почты, имена файлов и расширения файлов.

моноширинный шрифт
: используется для листингов программ, а также внутри абзацев для ссылки на элементы программ, такие как переменные или имена функций, базы данных, типы данных, переменные среды, инструкции и ключевые слова.

Данный элемент обозначает общее замечание.

Данный элемент обозначает предупреждение или предостережение.

Использование примеров исходного кода

Дополнительные материалы (примеры исходного кода, упражнения и т. д.) доступны для скачивания по адресу https://github.com/mikexcohen/LinAlg4DataScience.

Если у вас есть технический вопрос или проблема с использованием примеров исходного кода, то, пожалуйста, отправьте электронное письмо по адресу bookquestions@oreilly.com.

Благодарности

Должен признаться, я действительно не люблю писать разделы с признаниями. И это не потому, что мне не хватает благодарности или я считаю, что мне некого благодарить, – совсем наоборот: у меня слишком много людей, которых нужно поблагодарить, и я не знаю, с чего начать, кого перечислить по имени, а кого пропустить. Должен ли я поблагодарить своих родителей за их роль в формировании меня таким человеком, который написал эту книгу? Возможно, их родителей за то, что они сформировали моих родителей? Помню, как моя учительница в четвертом классе говорила мне, что я, должно быть, стану писателем, когда вырасту. (Я не помню ее имени и не уверен, когда я вырасту, но, возможно, она оказала некоторое влияние на эту книгу.) Я написал бóльшую часть этой книги во время поездок на Канарские острова с работой на удалении; возможно, мне следует поблагодарить пилотов, кото-

рые доставили меня туда? Или электриков, которые устанавливали проводку в коворкингах? Вероятно, я должен быть благодарен Оздемиру Паше за его роль в популяризации кофе, который одновременно облегчал и отвлекал меня от писательства. И давайте не будем забывать фермеров, которые выращивали вкусную еду, которая меня поддерживала и делала счастливым.

Видите, к чему это ведет: мои пальцы печатали, но потребовалась вся история человеческой цивилизации, чтобы создать меня и среду, которая позволила мне написать эту книгу – и которая позволила вам прочитать эту книгу. Таким образом, спасибо человечеству!

Ну да ладно, я также могу посвятить один абзац более традиционному разделу благодарностей. Самое главное, я благодарен всем моим студентам на моих курсах в университете и летней школе с живым преподаванием, а также на моих онлайновых курсах Udemy за то, что они доверили мне преподавание у них и мотивировали меня продолжать совершенствовать свои объяснения прикладной математики и других технических тем. Я также благодарен Джесс Хаберману, редактору отдела закупок в O'Reilly, которая установила «первый контакт», чтобы спросить, не буду ли я заинтересован в написании этой книги. Шира Эванс (редактор разработки), Джонатан Оуэн (производственный редактор), Элизабет Оливер (выпускающий редактор), Кристен Браун (менеджер по контентным услугам) и два эксперта – технических рецензента сыграли непосредственную роль в трансформации моих нажатий клавиш в книгу, которую вы сейчас читаете. Уверен, что этот список неполон, потому что другие люди, которые помогли опубликовать эту книгу, мне неизвестны либо потому, что я забыл их из-за потери памяти в моем преклонном возрасте[1]. Благодарности всем читающим этот текст, кто чувствует, что внес хотя бы ничтожный вклад в эту книгу.

[1] ЛОЛ, когда я написал эту книгу, мне было 42 года.

Глава 1

Введение

Что такое линейная алгебра и зачем ее изучать?

Линейная алгебра имеет интересную историю в математике, восходящую к XVII веку на Западе и гораздо раньше в Китае. Матрицы – развернутые таблицы чисел, лежащие в основе линейной алгебры, – использовались для обеспечения компактной системы счисления с целью хранения наборов чисел, таких как геометрические координаты (это было первоначальное применение матриц Декартом) и систем уравнений (впервые введенных Гауссом). В XX веке матрицы и векторы использовались для многопеременной математики, включая математическое исчисление, дифференциальные уравнения, физику и экономику.

Но до недавнего времени большинству людей не нужно было заботиться о матрицах. Однако, как оказалось, компьютеры чрезвычайно эффективны в работе с матрицами. И поэтому современные вычисления породили современную линейную алгебру. Современная линейная алгебра является вычислительной, в то время как традиционная линейная алгебра является абстрактной. Современную линейную алгебру лучше всего изучать на исходном коде и приложениях в области графики, статистики, науки о данных, искусственного интеллекта и численного моделирования, в то время как традиционная линейная алгебра изучается на доказательствах и размышлениях о бесконечномерных пространствах векторов. Современная линейная алгебра предоставляет структурные элементы, которые поддерживают почти каждый алгоритм, реализованный на компьютерах, в то время как традиционная линейная алгебра зачастую является интеллектуальной пищей для продвинутых студентов математических университетов.

Добро пожаловать в современную линейную алгебру.

Стоит ли вам изучать линейную алгебру? Это зависит от того, хотите ли вы понимать алгоритмы и процедуры или же просто применять методы, разработанные другими. Я не хочу принижать последнее – в сущности, нет ничего

плохого в использовании инструментов, которые вы не понимаете (я пишу это на ноутбуке, который я могу использовать, но не смог создать его с нуля). Но, учитывая, что вы читаете книгу с таким названием в коллекции книг O'Reilly, я предполагаю, что вы (1) хотите узнать принцип работы алгоритмов либо (2) хотите разрабатывать или адаптировать вычислительные методы. Так что да, вы должны изучать линейную алгебру, и вы должны изучить ее современную версию.

Об этой книге

Предназначение этой книги – научить вас современной линейной алгебре. Но речь идет не о том, чтобы запомнить несколько ключевых уравнений и пройтись по абстрактным доказательствам; цель в том, чтобы научить вас думать о матрицах, векторах и операциях на них. Вы разовьете геометрическое понимание на уровне интуиции относительно того, почему линейная алгебра такова, какая она есть. И вы поймете, как реализовывать концепции линейной алгебры в рабочем коде Python, уделяя особое внимание приложениям в области машинного обучения и науки о данных.

Во многих традиционных учебниках по линейной алгебре избегаются числовые примеры в интересах обобщений, ожидается, что вы самостоятельно получите сложные доказательства, и преподается множество концепций, которые имеют мало отношения либо вообще не имеют отношения к применению или реализации на компьютерах. Я пишу эти строки не как критику – абстрактная линейная алгебра прекрасна и элегантна. Но если ваша цель – использовать линейную алгебру (и математику в целом) в качестве инструмента для понимания данных, статистики, глубокого обучения, обработки изображений и т. д., то традиционные учебники по линейной алгебре, возможно, покажутся досадной тратой времени, которая поставит вас в замешательство и обеспокоит, взяв под сомнение ваши потенциальные возможности в технической области.

Эта книга написана с учетом того, что учащиеся занимаются самообучением. Возможно, у вас есть степень в области математики, инженерии или физики, но вам нужно научиться реализовывать линейные алгоритмы в рабочем коде. Или же, вполне вероятно, вы не изучали математику в университете и теперь осознаете важность линейной алгебры для вашей учебы или работы. В любом случае, эта книга является самостоятельным ресурсом; она не представляет собой исключительно дополнение к лекционному курсу (хотя ее можно было бы использовать для этой цели).

Если, читая последние три абзаца, вы кивнули головой в знак согласия, то эта книга определенно для вас.

Если вы хотите погрузиться в линейную алгебру поглубже, с большим количеством доказательств и разведывательной работы, то существует несколько отличных учебников, о прочтении которых вы можете подумать,

включая мой учебник «Линейная алгебра: теория, интуиция, исходный код (Sincxpress BV)[1].

Предварительные требования

Я пытался написать эту книгу для увлеченных учеников с минимальной формальной подготовкой. И тем не менее ничего никогда не изучается по-настоящему с нуля.

Математика

Вы должны чувствовать себя комфортно в математике уровня средней школы. Просто основы алгебры и геометрии, и ничего особенного.

Для этой книги требуется абсолютно нулевой уровень в исчислении (хотя дифференциальное исчисление имеет большую важность для приложений, в которых линейная алгебра используется часто, таких как глубокое обучение и оптимизация).

Но самое главное – вам нужно чувствовать себя комфортно, думая о математике, рассматривая уравнения и графики и не боясь противостоять интеллектуальным трудностям, которые возникают при изучении математики.

Отношение

Линейная алгебра – это раздел математики, и, следовательно, данная книга посвящена математике. Изучение математики, в особенности во взрослом возрасте, требует определенного терпения, целеустремленности и напористости. Выпейте чашку кофе, сделайте глубокий вдох, уберите свой телефон в другую комнату и погрузитесь в работу.

В глубине вашей головы будет звучать голос, говорящий о том, что вы слишком стары или слишком глупы, чтобы изучать продвинутую математику. Иногда этот голос будет звучать громче, а иногда мягче, но он всегда будет присутствовать. И это не только у вас – он есть у всех. Этот голос невозможно подавить или уничтожить; даже не пытайтесь. Просто примите, что некоторая неуверенность в себе – это часть человеческого бытия. Всякий раз, когда этот голос заговаривает, вам приходится доказывать, что он неправ.

---

[1] Приношу извинения за бесстыдную саморекламу; обещаю, что в данной книге это единственный случай, когда я позволяю себе подобное послабление.

Программирование

Данная книга посвящена линейно-алгебраическим приложениям в рабочем коде. Я написал эту книгу, ориентируясь на Python, потому что Python в настоящее время является наиболее широко используемым языком в науке о данных, машинном обучении и смежных областях. Если вы предпочитаете другие языки, такие как MATLAB, R, C или Julia, то я надеюсь, что вам будет легко переложить рабочий код Python на эти языки.

Я постарался сделать код Python как можно проще, оставляя его при этом релевантным для приложений. Глава 16 содержит базовое введение в программирование на Python. Стоит ли вам просматривать эту главу? Все зависит от вашего уровня владения языком Python:

Средний/продвинутый уровень (опыт программирования > 1 года)
: Полностью пропустите главу 16 либо, по возможности, пролистайте ее, чтобы получить представление о типе рабочего кода, который появится в остальной части книги.

Некоторые знания (опыт < 1 года)
: Рекомендую проработать эту главу на случай, если в ней есть новый материал или вам нужно его освежить. Но вы должны быть в состоянии пройти ее довольно быстро.

Полный новичок
: Подробно ознакомьтесь с этой главой. Поймите, что данная книга не является полным руководством по Python, поэтому если вы обнаружите, что вам трудно разобраться с рабочим кодом в главах книги, то, возможно, вы захотите отложить эту книгу, поработать со специальным курсом или книгой по Python, а затем вернуться к этой книге.

Математические доказательства в противовес интуитивному пониманию на основе программирования

Цель изучения математики в общем и целом состоит в том, чтобы понять математику. Как понять математику? Давайте посчитаем способы:

Строгие доказательства
: Доказательство в математике – это последовательность утверждений, демонстрирующая то, что набор допущений приводит к логическому заключению. Доказательства, несомненно, имеют большую важность в чистой математике.

Визуализации и примеры

Четко написанные объяснения, диаграммы и численные примеры помогут приобретать понимание концепций и операций в линейной алгебре на уровне интуиции. Для удобства визуализации большинство примеров выполнено в 2D либо 3D, но принципы применимы и к более высоким измерениям.

Разница между ними заключается в том, что формальные математические доказательства обеспечивают строгость, но редко понимание на уровне интуиции, в то время как визуализации и примеры обеспечивают прочное понимание на уровне интуиции благодаря практическому опыту, но рискуют приводить к неточностям, будучи основанными на конкретных примерах, которые не обобщаются.

Доказательства важных утверждений включены, но я больше концентрируюсь на упрочении интуитивного понимания посредством объяснений, визуализаций и примеров исходного кода.

И это подводит меня к пониманию математики на уровне интуиции на основе программирования (тому, что я иногда называю «мягкими доказательствами»). Вот идея: вы исходите из того, что в Python (и таких библиотеках, как NumPy и SciPy) правильно реализована низкоуровневая обработка чисел, в то время как вы концентрируетесь на принципах путем обследования многочисленных числовых примеров в исходном коде.

Краткий пример: мы «мягко докажем» принцип коммутативности умножения, который гласит, что $a \times b = b \times a$:

```
a = np.random.randn()
b = np.random.randn()
a * b - b * a
```

Приведенный выше исходный код генерирует два случайных числа и проверяет гипотезу о том, что изменение порядка умножения не влияет на результат. Если принцип коммутативности истинен, то в третьей строке будет выведено значение 0.0. Если вы выполните этот исходный код несколько раз и всегда будете получать 0.0, то, увидев один и тот же результат в многочисленных примерах с разными числами, вы приобретете понимание коммутативности на уровне интуиции.

Для ясности: понимание на уровне интуиции на основе исходного кода не заменяет строгого математического доказательства. Все дело в том, что «мягкие доказательства» позволяют понимать математические концепции, не беспокоясь о деталях абстрактного математического синтаксиса и аргументов. Это особенно выгодно для программистов, которым не хватает продвинутого математического образования.

В сухом остатке все сводится к тому, что *много математических концепций могут усваиваться при помощи небольшой порции исходного кода*.

Рабочий код в книге и предназначенный для скачивания онлайн

Данную книгу можно читать, не глядя на исходный код и не решая упражнения по программированию. Все в порядке; вы обязательно чему-нибудь научитесь. Но не расстраивайтесь, если ваши знания будут поверхностными и мимолетными. Если вы хотите действительно *понять* линейную алгебру, то вам нужно решать задачи. Вот почему эта книга снабжена демонстрациями исходного кода и упражнениями по каждой математической концепции.

Важный исходный код напечатан непосредственно в книге. Я хочу, чтобы вы читали текст и уравнения, смотрели на графики и одновременно *видели исходный код*. Это позволит вам связывать концепции и уравнения с исходным кодом.

Но листинг исходного кода в книге может занимать много места, а ручное его копирование на вашем компьютере будет утомительным. Поэтому на страницах книги печатаются только ключевые строки кода; онлайновый код содержит дополнительный исходный код, комментарии, графические украшения и т. д. Онлайновый код также содержит решения упражнений по программированию (всех упражнений, а не только выборочных задач!). Вам обязательно следует скачать исходный код и ознакомиться с ним во время работы с книгой.

Весь исходный код можно получить из репозитория на сайте GitHub https://github.com/mikexcohen/LinAlg4DataScience. Этот репозиторий можно клонировать либо просто скачать целиком в виде ZIP-файла (вам не потребуется регистрироваться, входить в систему либо платить за скачивание кода).

Я написал исходный код в среде Google Colab, используя блокнот Jupyter. Я решил использовать Jupyter, потому что это дружественная и простая в применении среда. Тем не менее призываю вас использовать любую интегрированную среду разработки на Python, которую вы предпочитаете. Для удобства онлайновый код также предоставляется в виде простых файлов .py.

Упражнения по программированию

Математика – это не зрительский вид спорта. В большинстве книг по математике есть бесчисленное множество письменных задач (и давайте будем честны: никто не решает их все). Но данная книга целиком посвящена *прикладной* линейной алгебре, и никто не применяет линейную алгебру на бумаге! Вместо этого линейная алгебра применяется в рабочем коде. Поэтому вместо ручных задач и утомительных доказательств, «оставленных читателю в качестве упражнения» (как любят писать авторы учебников по математике), в данной книге много упражнений по программированию.

Упражнения по программированию различаются по сложности. Если вы в Python и линейной алгебре – новичок, то некоторые упражнения, возмож-

но, покажутся вам действительно сложными. Если вы застряли, то вот вам совет: кратко взгляните на мое решение, чтобы вдохновиться, затем уберите его, чтобы не видеть мой исходный код, и продолжайте работать над своим исходным кодом.

Сравнивая свое решение с моим, имейте в виду, что существует много способов решения задач на Python. Важно найти правильный ответ; предпринимаемые вами шаги, чтобы его получить, нередко зависят от личного стиля программирования.

Как пользоваться этой книгой (для учителей и самообучающихся)

Данная книга полезна в трех средах:

Самообучение
: Я постарался сделать эту книгу доступной для читателей, которые хотят изучать линейную алгебру самостоятельно, вне формальной классной комнаты. Никаких дополнительных ресурсов или онлайновых лекций не требуется, хотя, конечно же, существует масса других книг, веб-сайтов, видеороликов YouTube и онлайновых курсов, которые студенты могут счесть полезными.

Первичный учебник на занятиях по науке о данных
: Данная книга может использоваться в качестве первичного учебника в курсе математики, лежащей в основе науки о данных, машинного обучения, искусственного интеллекта и смежных тем. Содержимое состоит из 14 глав (исключая это введение и дополнительный материал по Python), и от студентов ожидается, что они будут работать над одной–двумя главами в неделю. Поскольку учащиеся имеют доступ к решениям всех упражнений, преподаватели, возможно, пожелают дополнить упражнения книги дополнительными наборами задач.

Вторичный учебник по математикоориентированному курсу линейной алгебры
: Эта книга также может использовать в качестве дополнения к курсу математики с сильным акцентом на доказательствах. В этом случае лекции были бы сосредоточены на теории и строгих доказательствах, в то время как на данную книгу можно было бы ссылаться с целью переложения концепций в исходный код с прицелом на приложения в науке о данных и машинном обучении. Как я написал выше, преподаватели, возможно, пожелают предоставить дополнительные упражнения, поскольку решения ко всем упражнениям из книги доступны онлайн.

Глава 2

Векторы. Часть 1

Векторы обеспечивают основу, на которой построена вся линейная алгебра (и, следовательно, остальная часть этой книги).

К концу этой главы вы будете знать о векторах все: что они собой представляют, что они делают, как их интерпретировать и как создавать и работать с ними на Python. Вы поймете наиболее важные операции на векторах, включая векторную алгебру и точечное произведение. Наконец, вы узнаете о разложениях векторов, являющихся одной из главных целей линейной алгебры.

Создание и визуализация векторов в NumPy

В линейной алгебре *вектор* – это упорядоченный список чисел. (В абстрактной линейной алгебре векторы могут содержать другие математические объекты, включая функции; однако поскольку данная книга ориентирована на приложения, мы будем рассматривать только те векторы, которые содержат числа.)

Векторы обладают несколькими важными характеристиками. Первые две, с которых мы начнем, – это[1]:

размерность.
Число чисел в векторе;

ориентация.
Ориентирован ли вектор *вдоль столбца* (стоя в полный рост) либо *вдоль строки* (лежа ровно во всю ширь).

Размерность нередко указывается с помощью причудливо выглядящей записи $\mathbb{R}^N$, где $\mathbb{R}$ обозначает действительно-значные числа (ср. с символом

[1] В книге проводится четкое различие между созвучными понятиями dimensionality (математическая размерность, протяженность, объемность) и dimension (геометрическое измерение, мерность как в слове двухмерный). Первый термин переводится как размерность, а второй в зависимости от контекста – как измерение либо мерность. – *Прим. перев.*

$\mathbb{C}$ для комплексно-значных чисел), и *0* указывает на размерность. Например, считается, что вектор с двумя элементами принадлежит $\mathbb{R}^2$.

Специальный символ $\mathbb{R}$ создан с использованием кода latex, но вы также можете написать R^2, R2 или R^2.

Уравнение 2.1 показывает несколько примеров векторов; перед чтением следующего абзаца рекомендуется определить их размерность и ориентацию.

Уравнение 2.1. Примеры вектора-столбца и вектора-строки

$$\mathbf{x} = \begin{bmatrix} 1 \\ 4 \\ 5 \\ 6 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} .3 \\ -7 \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} 1 & 4 & 5 & 6 \end{bmatrix}.$$

Вот ответы: **x** – четырехмерный вектор-столбец, **y** – двумерный вектор-столбец и **z** – четырехмерный вектор-строка. Также можно написать, например, $\mathbf{x} \in \mathbb{R}^2$, где символ $\in$ означает «принадлежит множеству».

Являются ли **x** и **z** одним и тем же вектором? Технически они разные, хотя и содержат одни и те же элементы в одном и том же порядке. Более подробное изложение этого вопроса смотрите во врезке «Имеет ли значение ориентация вектора?» на стр. **9**.

Из этой книги и на протяжении всех ваших приключений, связанных с интеграцией математики и программирования, вы узнаете, что существуют различия между математикой «на меловой доске» и реализованной в рабочем коде. Некоторые расхождения незначительны и несущественны, в то время как другие вызывают путаницу и ошибки. Давайте-ка я теперь познакомлю вас с терминологическим разночтением между математикой и программированием.

Ранее я писал, что *размерность* вектора – это число элементов в этом векторе. Однако в Python размерность вектора или матрицы – это число геометрических измерений, используемых для распечатки числового объекта. Например, все показанные выше векторы считаются в Python «двумерными массивами» независимо от содержащегося в векторах числа элементов (то есть математической размерности). Список чисел без определенной ориентации в Python считается одномерным массивом независимо от числа элементов (этот массив будет распечатан в виде строки, но, как вы увидите позже, он обрабатывается иначе, чем вектор-строка). Математическая размерность – число элементов в векторе – в Python называется *длиной*, или *очертанием*[1], вектора.

Бывает, что эта несогласованная, а иногда и противоречивая терминология сбивает с толку. И действительно, терминология нередко становится сложной проблемой на пересечении различных дисциплин (в данном случае математики и вычислительной науки). Но не волнуйтесь, вы это освоитесь, приобретя немного опыта.

---

[1] Англ. *shape*; син. форма, контур. – *Прим. перев.*

При упоминании векторов обычно используются строчные жирные римские буквы, например **v** для обозначения «вектор v». В некоторых учебниках используется курсив (*v*) или вверху выводится стрелка в качестве диакритического знака ($\vec{v}$).

По традиции в линейной алгебре исходят из допущения, что векторы ориентированы вдоль столбца, если не указано иное. Векторы-строки записываются в виде $\mathbf{w}^T$. Надстрочная буква ^{T} указывает на *операцию транспонирования*, о которой вы узнаете подробнее чуть позже; пока же достаточно сказать, что операция транспонирования конвертирует вектор-столбец в вектор-строку.

Имеет ли значение ориентация вектора?

Действительно ли нужно беспокоиться о том, ориентированы ли векторы вдоль столбцов либо вдоль строк, или же они являются неориентированными одномерными массивами? Иногда – да, а иногда – нет. При использовании векторов для хранения данных ориентация обычно не имеет значения. Но если ориентация неправильная, то некоторые операции в Python могут давать ошибки либо неожиданные результаты. Поэтому важно понимать ориентацию векторов, поскольку, тратя 30 минут на отладку кода только для того, чтобы понять, что вектор-строка должен быть вектором-столбцом, вы гарантированно получите головную боль.

Векторы в Python могут быть представлены с использованием нескольких типов данных. Тип «список», возможно, покажется самым элементарным способом представления вектора – и это для некоторых приложений. Но многие линейно-алгебраические операции не будут работать со списками Python. Поэтому в большинстве случаев лучше всего создавать векторы в виде массивов NumPy. В следующем ниже исходном коде показаны четыре способа создания вектора:

```
asList  = [1, 2, 3]
asArray = np.array([1, 2, 3])         # одномерный массив
rowVec  = np.array([ [1, 2, 3] ])     # строка
colVec  = np.array([ [1], [2], [3] ]) # столбец
```

Переменная `asArray` – это *неориентированный* массив, и, значит, это ни вектор-строка, ни вектор-столбец, а просто одномерный список чисел в NumPy. В NumPy ориентация задается скобками: самые внешние скобки группируют все числа вместе в один объект. Затем каждый дополнительный набор скобок указывает строку: вектор-строка (переменная `rowVec`) содержит все числа в одной строке, в то время как вектор-столбец (переменная `colVec`) содержит несколько строк, причем каждая строка содержит одно число.

С этими ориентациями можно познакомиться поближе, проинспектировав очертания переменных (в программировании проверка очертаний переменных нередко бывает очень полезной):

```
print(f'asList:  {np.shape(asList)}')
print(f'asArray: {asArray.shape}')
print(f'rowVec:  {rowVec.shape}')
print(f'colVec:  {colVec.shape}')
```

Вот как выглядит результат:

```
asList:  (3,)
asArray: (3,)
rowVec:  (1, 3)
colVec:  (3, 1)
```

Результат показывает, что одномерный массив `asArray` имеет размер `(3)`, тогда как наделенные ориентацией векторы являются двумерными массивами и хранятся как размер `(1,3)` либо `(3,1)` в зависимости от ориентации. Размеры всегда указываются как (строки, столбцы).

Геометрия векторов

Упорядоченный список чисел – это алгебраическая интерпретация вектора; геометрическая интерпретация вектора представляет собой прямой отрезок с определенной длиной (также именуемой *модулем* вектора[1]) и направлением (также именуемым *углом*; он вычисляется относительно положительной оси *x*). Две точки вектора называются хвостом (где он начинается) и головой (где он заканчивается); голова часто имеет наконечник стрелки, чтобы устранить неоднозначность с хвостом.

Возможно, вы подумали, что в векторе кодируется геометрическая координата, но векторы и координаты – это на самом деле разные вещи. Однако они согласуются, когда вектор начинается в начале координат. Этот случай называется *стандартным положением* и проиллюстрирован на рис. 2.1.

Концептуализация векторов в геометрическом либо алгебраическом плане облегчает интуитивное понимание в различных приложениях, но это просто две стороны одной медали. Например, геометрическая интерпретация вектора широко применяется в физике и инженерии (например, для представления физических сил), а алгебраическая интерпретация вектора – в науке о данных (например, для хранения данных о продажах во временной динамике). Нередко линейно-алгебраические концепции усваиваются геометрически на 2D-графиках, а затем расширяются до более высоких измерений с помощью алгебры.

---

[1] Англ. *magnitude*. – *Прим. перев.*

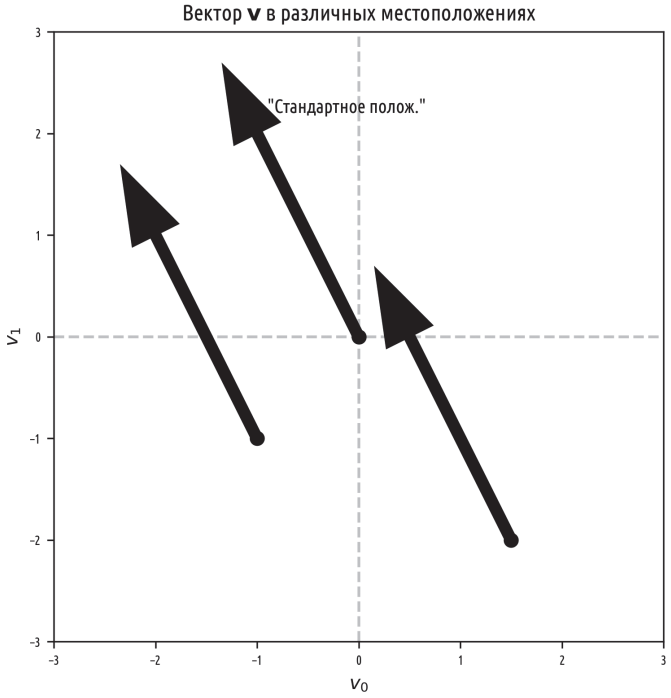

Рис. 2.1 ❖ Все стрелки выражают один и тот же вектор. Вектор в стандартном положении имеет свой хвост в начале координат и свою голову в согласованной геометрической координате

Операции на векторах

Векторы подобны существительным; они являются персонажами нашего рассказа о линейной алгебре. Веселье в линейной алгебре исходит от глаголов – действий, которые вдыхают жизнь в персонажей. Эти действия называются *операциями*.

Некоторые линейно-алгебраические операции элементарны и интуитивно понятны и работают именно так, как вы и ожидаете (например, сложение), в то время как другие более сложны и требуют объяснений в объеме целых глав (например, сингулярное разложение). Давайте начнем с элементарных операций.

Сложение двух векторов

Для того чтобы сложить два вектора, надо просто сложить каждый соответствующий элемент. Уравнение 2.2 показывает пример:

Уравнение 2.2. Сложение двух векторов

$$\begin{bmatrix}4\\5\\6\end{bmatrix}+\begin{bmatrix}10\\20\\30\end{bmatrix}=\begin{bmatrix}14\\25\\36\end{bmatrix}.$$

Как вы могли догадаться, сложение векторов определено только для двух векторов, которые имеют одинаковую размерность; например, невозможно сложить вектор в $\mathbb{R}^3$с вектором в $\mathbb{R}^5$.

Вычитание двух векторов

Вычитание векторов – это тоже то, что вы и ожидаете: вычесть второй вектор из первого поэлементно.

Уравнение 2.3 показывает пример:

Уравнение 2.3. Вычитание двух векторов

$$\begin{bmatrix}4\\5\\6\end{bmatrix}-\begin{bmatrix}10\\20\\30\end{bmatrix}=\begin{bmatrix}-6\\-15\\-24\end{bmatrix}.$$

Сложение векторов на языке Python выполняется просто:

```
v = np.array([ 4, 5, 6])
w = np.array([10,20,30])
u = np.array([ 0, 3, 6, 9])
vPlusW = v+w
uPlusW = u+w  # ошибка! измерения не совпадают!
```

Имеет ли значение ориентация вектора для сложения? Рассмотрим уравнение 2.4:

Уравнение 2.4. Можно ли сложить вектор-строку с вектором-столбцом?

$$\begin{bmatrix}4\\5\\6\end{bmatrix}+\begin{bmatrix}10 & 20 & 30\end{bmatrix}=?.$$

Возможно, вы подумаете, что между этим примером и тем, что был показан ранее, нет никакой разницы – в конце концов, оба вектора состоят из трех элементов. Давайте посмотрим, что сделает Python:

```
v = np.array([[ 4, 5, 6]])    # вектор-строка
w = np.array([[10,20,30]]).T  # вектор-столбец
v+w
```

```
>> array([[14, 15, 16],
          [24, 25, 26],
          [34, 35, 36]])
```

Возможно, результат покажется запутанным и несовместимым с приведенным ранее определением сложения векторов. На самом деле в Python реализована операция, именуемая *транслированием*. Вы узнаете о трансляции больше чуть позже в этой главе, но я призываю вас потратить немного времени на осмысление результата и подумать о том, как он возник в результате сложения вектора-строки с вектором-столбцом. Несмотря на это, данный пример показывает, что ориентация действительно важна: *два вектора можно сложить, только если они имеют одинаковую размерность и одинаковую ориентацию*.

Геометрия сложения и вычитания векторов

Для того чтобы сложить два вектора геометрически, надо расположить векторы так, чтобы хвост одного вектора находился в голове другого вектора. Суммируемый вектор проходит от хвоста первого вектора к голове второго (график A на рис. 2.2). Эту процедуру можно расширить, чтобы суммировать любое число векторов: надо просто уложить все векторы от хвоста к голове, и тогда сумма будет равна отрезку, идущему от первого хвоста к итоговой голове.

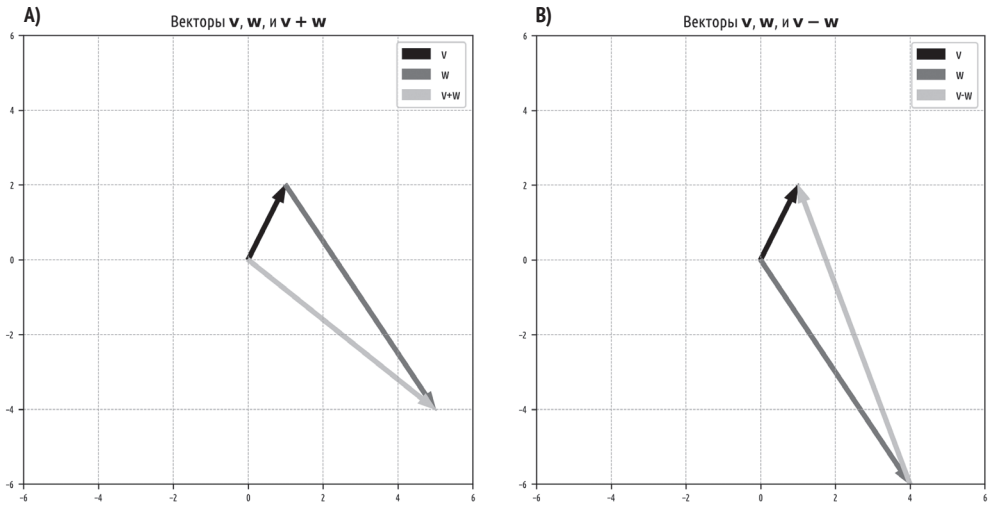

Рис. 2.2 ❖ Сумма и разность двух векторов

Геометрическое вычитание векторов немного отличается, но является одинаково элементарным: надо выровнять два вектора так, чтобы их хвосты находились в одной и той же координате (это легко достигается, если оба вектора находятся в стандартном положении); вектор разности – это отрезок,

который идет от головы «отрицательного» вектора к голове «положительного» вектора (график B на рис. 2.2).

Не стоит недооценивать важность геометрии вычитания векторов: она лежит в основе ортогонального разложения векторов, которое, в свою очередь, лежит в основе метода линейных наименьших квадратов, который является одним из наиболее важных приложений линейной алгебры в науке и технике.

Умножение вектора на скаляр

Скаляр в линейной алгебре – это число в чистом виде, не вложенное ни в вектор, ни в матрицу. Скаляры обычно обозначаются строчными греческими буквами, такими как α или λ. Поэтому умножение вектора на скаляр обозначается, например, как $\beta\mathbf{u}$.

Умножение вектора на скаляр выполняется очень просто: надо умножить каждый элемент вектора на скаляр. Для понимания будет достаточно одного численного примера (уравнение 2.5):

Уравнение 2.5. Умножение вектора на скаляр (также умножение скаляра на вектор)

$$\lambda = 4, \quad \mathbf{w} = \begin{bmatrix} 9 \\ 4 \\ 1 \end{bmatrix}, \quad \lambda\mathbf{w} = \begin{bmatrix} 36 \\ 16 \\ 4 \end{bmatrix}.$$

Вектор нулей

Вектор, состоящий из одних нулей, также именуемый *вектором нулей*, или нуль-вектором, обозначается жирным шрифтом, **0**, и в линейной алгебре является специальным вектором. Нередко использование вектора нулей для решения задачи фактически принято называть *тривиальным решением* и исключать. Линейная алгебра полна утверждений типа «найти ненулевой вектор, который может решить ...» или «найти нетривиальное решение для ...».

Ранее я писал, что тип данных хранящей вектор переменной иногда важен, а иногда и не важен. Умножение вектора на скаляр является примером случая, когда тип данных имеет значение:

```
s = 2
a = [3, 4, 5]    # в виде списка
b = np.array(a)  # в виде np-массива
print(a*s)
print(b*s)

>> [ 3, 4, 5, 3, 4, 5 ]
>> [ 6 8 10 ]
```

Приведенный выше исходный код создает скаляр (переменную `s`) и вектор в виде списка (переменную `a`), затем конвертирует их в массив NumPy (переменную `b`). В Python звездочка перегружена, то есть ее поведение зависит от типа переменной: умножение списка на скаляр повторяет список `s` раз (в данном случае два раза), что определенно *не* является линейно-алгебраической операцией умножения скаляра на вектор. Однако когда вектор хранится в виде массива NumPy, звездочка интерпретируется как поэлементное умножение. (Вот для вас небольшое упражнение: что произойдет, если задать `s = 2.0`, и почему[1]?) Обе эти операции (повторение списка и умножение вектора на скаляр) используются в реальном программировании, поэтому об указанном различии следует помнить.

Сложение скаляра с вектором

Сложение скаляра с вектором формально в линейной алгебре не определено: это два отдельных вида математических объектов, которые невозможно объединить. Однако программы числовой обработки, такие как Python, позволяют складывать скаляры с векторами, и указанная операция сравнима с умножением скаляра на вектор: скаляр прибавляется к каждому элементу вектора. Следующий ниже исходный код иллюстрирует эту идею:

```
s = 2
v = np.array([3, 6])
s+v

>> [5 8]
```

Геометрия умножения вектора на скаляр

Почему скаляры называются «скалярами»? Это вытекает из геометрической интерпретации. Скаляры шкалируют векторы, не меняя их направления. Существует четыре эффекта умножения вектора на скаляр, которые зависят от того, является ли скаляр больше 1, между 0 и 1, в точности 0 либо отрицательным. Рисунок 2.3 иллюстрирует эту идею.

Ранее я писал, что скаляры не меняют направление вектора. Но на рисунке показано, что направление вектора меняется, когда скаляр отрицательный (то есть его угол поворачивается на 180°). Возможно, это покажется противоречием, но существует интерпретация векторов, в которой они указывают вдоль бесконечно длинной линии, проходящей через начало координат и уходящей в бесконечность в обоих направлениях (в следующей главе я буду называть такую интерпретацию «одномерным подпространством»). В этом смысле «повернутый» вектор по-прежнему указывает вдоль той же самой

[1] Выражение `a*s` выдаст ошибку, потому что повторять список можно только с использованием целых чисел; невозможно повторить список 2.72 раза!

бесконечной линии, и, следовательно, отрицательный скаляр не меняет направления. Указанная интерпретация важна для пространств матриц, собственных векторов и сингулярных векторов, все из которых представлены в последующих главах.

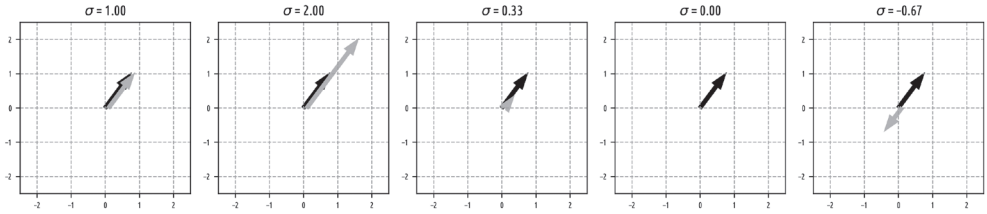

Рис. 2.3 ❖ Один и тот же вектор (черная стрелка), умноженный на разные скаляры σ (серый отрезок; для наглядности слегка сдвинут)

Умножение вектора на скаляр в сочетании со сложением векторов приводит непосредственно к *усреднению векторов*. Усреднение векторов – это то же самое, что усреднение чисел: просуммировать и поделить на количество чисел. Таким образом, для того чтобы усреднить два вектора, надо их сложить, а затем скалярно умножить на .5. В общем случае, для того чтобы усреднить N векторов, надо их просуммировать и скалярно умножить результат на $1/N$.

Транспонирование

Вы уже узнали об операции транспонирования: она конвертирует векторы-столбцы в векторы-строки и наоборот. Тут стоит дать несколько более формальное определение, которое будет обобщено на транспонирование матриц (тема в главе 5).

Матрица состоит из строк и столбцов; следовательно, каждый элемент матрицы имеет индекс в формате (строка, столбец). Операция транспонирования просто меняет местами эти индексы. Она формализуется в уравнении 2.6:

Уравнение 2.6. Операция транспонирования

$$\mathbf{m}^{\mathrm{T}}_{i,j} = \mathbf{m}_{j,i}.$$

Векторы имеют либо одну строку, либо один столбец, в зависимости от их ориентации. Например, шестимерный вектор-строка имеет $i = 1$ и индексы j от 1 до 6, тогда как шестимерный вектор-столбец имеет индексы i от 1 до 6 и $j = 1$. Таким образом, перемена мест индексов i, j меняет местами строки и столбцы.

Вот важное правило: двойное транспонирование возвращает вектор в изначальную ориентацию. Другими словами, $\mathbf{v}^{\mathrm{TT}} = \mathbf{v}$. Возможно, указанное правило покажется очевидным и тривиальным, но оно является краеугольным камнем нескольких важных доказательств в науке о данных и машин-

ном обучении, включая создание симметричных матриц ковариаций при умножении матрицы данных на ее транспонированную версию (что, в свою очередь, является причиной того, что анализ главных компонент есть ортогональный поворот пространства данных… не волнуйтесь, это предложение обретет смысл в данной книге чуть позже!).

Транслирование векторов в Python

Транслирование – это операция, которая существует только в современной компьютерной линейной алгебре; это не та процедура, которую вы бы нашли в традиционном учебнике по линейной алгебре.

Транслирование, по существу, означает многократное повторение операции между одним вектором и каждым элементом другого вектора. Рассмотрим следующую ниже серию уравнений[1]:

[1 1] + [10 20];

[2 2] + [10 20];

[3 3] + [10 20].

Обратите внимание на закономерности в векторах. Приведенный выше набор уравнений можно компактно реализовать, сжав указанные закономерности в векторы [1 2 3] и [10 20], а затем оттранслировав сложение. Вот как это выглядит на Python:

```
v = np.array([[1, 2, 3]]).T  # вектор-столбец
w = np.array([[10, 20]])     # вектор-строка
v + w  # сложение при помощи транслирования

>> array([[11, 21],
          [12, 22],
          [13, 23]])
```

Здесь вы снова заметите важность ориентации в линейно-алгебраических операциях: попробуйте выполнить приведенный выше исходный код, изменив `v` на вектор-строку и `w` – на вектор-столбец[2].

Поскольку транслирование позволяет проводить эффективные и компактные вычисления, оно встречается в числовом программировании очень часто. В данной книге вы увидите несколько примеров транслирования, в том числе в разделе, посвященном кластеризации методом *k*-средних (глава 4).

---

[1] Для ясности вот как данный термин трактуется в документации NumPy: термин «транслирование» (англ. *broadcasting*) относится к тому, как NumPy обрабатывает массивы с разным размером во время арифметических операций. С учетом определенных ограничений меньший массив «заполняется» по большему массиву, чтобы они имели совместимые очертания. – *Прим. перев.*

[2] Python по-прежнему выполнит транслирование, но в результате вместо матрицы 2×3 получится матрица 3×2.