Математические методы для компьютера Зрение, робототехника и графика

Примечания к курсу CS 205A, осень 2013 г.

Джастин Соломон
Департамент компьютерных наук
Стэндфордский Университет

Содержание

Часть I

Предварительные

Глава 0

Обзор математики

В этой главе мы рассмотрим соответствующие понятия из линейной алгебры и исчисления с несколькими переменными, которые будут фигурировать в нашем обсуждении вычислительных методов. Он задуман как обзор справочного материала с уклоном в сторону идей и интерпретаций, обычно встречающихся на практике; эта глава может быть пропущена или использована в качестве справочного материала студентами с более сильным математическим образованием.

0.1 Предварительные игры: числа и наборы

Вместо того чтобы рассматривать алгебраические (а иногда и философские) рассуждения вроде «Что такое число?», мы будем полагаться на интуицию и математический здравый смысл, чтобы определить несколько

множеств: • Натуральные числа N = {1, 2, 3 , . . .}

• Целые числа Z = {. . . , 2, 1, 0, 1, 2, . . .}

• Рациональные числа Q = {a/b : a, b Z} • [1]

Действительные числа R , охватывающие Q , а также иррациональные числа, такие как π и 2 •

Комплексные числа C = {a + bi : a, b R }, где мы думаем, что i удовлетворяет i = 1.

Стоит признать, что наше определение R далеко не строгое. Построение действительных чисел может быть важной темой для практиков криптографических методов, использующих альтернативные системы счисления, но эти тонкости не имеют отношения к данному обсуждению.

Как и с любыми другими наборами, N, Z, Q, R и C можно манипулировать с помощью общих операций для создания новых наборов чисел. В частности, напомним, что мы можем определить «евклидово произведение» двух множеств A и B как

$$A \times B = \{(a, b) : a \quad A \text{ и } b \quad B\}.$$

Мы можем взять степени множеств, написав

$$A^{n} = \underbrace{A \times A \times \cdots \times A}_{n \text{ раз}}.$$

[1] Это первый из многих случаев, когда мы будем использовать обозначение {A : B}; фигурные скобки должны обозначать набор, а двоеточие можно читать как «такой, что». Например, определение Q можно прочитать как «множество дробей a/b , таких что a и b — целые числа». В качестве второго примера мы могли бы написать N = {n Z : n > 0}.

Эта конструкция дает то, что станет нашим любимым набором чисел в следующих главах:

$$p^{н} = \{(a1, a2, \ldots, an) : ai \quad R \text{ для всех } i\}$$

0.2 Векторные пространства

Вводные курсы по линейной алгебре вполне можно было бы назвать «Введение в конечномерные векторные пространства». Хотя определение векторного пространства может показаться абстрактным, мы найдем множество конкретных приложений, удовлетворяющих формальным аспектам и, таким образом, извлекающих пользу из разработанного нами механизма.

0.2.1 Определение векторных пространств

Начнем с определения векторного пространства и приведем ряд примеров:

Определение 0.1 (Векторное пространство). Векторное пространство — это множество V , замкнутое относительно скалярного умножения и сложения.

Для наших целей скаляр — это число в R, а операции сложения и умножения удовлетворяют обычным аксиомам (коммутативность, ассоциативность и т. д.). Как правило, определить векторные пространства в дикой природе несложно, включая следующие примеры:

Пример 0.1 (Rn как векторное пространство). Наиболее распространенным примером векторного пространства является Rn . Здесь сложение и скалярное умножение происходят покомпонентно:

$$(1, 2) + (\quad 3, 4) = (1 \quad 3, 2 + 4) = (\quad 2, 6)$$

$$10 \cdot (\quad 1, 1) = (10 \cdot \quad 1, 10 \cdot 1) = (\quad 10, 10)$$

Пример 0.2 (многочлены). Вторым важным примером векторного пространства является «кольцо» многочленов с вещественными входными параметрами, обозначаемое R[x]. Многочленом p R[x] называется функция p : R R, принимающая вид[2]

$$p(x) = \sum_{к} \text{к аккс}^{\cdot}$$

Сложение и скалярное умножение выполняются обычным образом, например, если $p(x) = x^2 + 2x - 1$ и $q(x) = x + 3x^3$, тогда $3p(x) + 5q(x) = 5x^3 + 2 + 6x - 3$, который является еще одним полиномом. Кроме того, обратите внимание, что функции типа $p(x) = (x \quad 1)(x + 1) + x$, хотя они явно не $^{2}{}_{(\text{икс}}{}^{3}$ for 5) все еще полиномы, даже записаны в приведенной выше форме.

Элементы v V векторного пространства V называются векторами, а взвешенная сумма вида i aivi , где ai R и vi V, известна как линейная комбинация элементов vi . В нашем втором примере «векторы» — это функции, хотя мы обычно не используем этот язык для обсуждения R[x]. Один k с последовательным способом зрения состоял бы в том, чтобы идентифицировать многочлен k akx (a0, a1, a2, · · ·); связать эти две точки помните, что многочлены имеют конечное число членов, так что последовательность в конце концов будет заканчиваться цепочкой нулей.

---

[2]Обозначение f : A B означает, что f — это функция, которая принимает на вход элемент множества A и выводит элемент множества B. Например, f : R Z принимает на вход вещественное число из R и выводит целое число Z. , как может быть в случае f(x) = x, функции «округления в меньшую сторону».

0.2.2 Размах, линейная независимость и основания

Предположим, мы начинаем с векторов v1, . . . ,vk V для векторного пространства V. По определению 0.1 у нас есть два способа начать с этих векторов и построить новые элементы V: сложение и скалярное умножение. Идея span заключается в том, что он описывает все векторы, которых вы можете достичь с помощью этих двух операций:

Определение 0.2 (Размах). Оболочкой множества S V векторов называется множество

span S {a1v1 + · · · + akvk : k 0, vi V для всех i и ai R для всех i}.

Обратите внимание, что span S является подпространством V, то есть подмножеством V , которое само по себе является векторным пространством. Мы можем приведите несколько примеров:

Пример 0.3 (Миксология). Типичный «колодец» в коктейль-баре содержит как минимум четыре ингредиента в распоряжении бармена: водка, текила, апельсиновый сок и гренадин. Предполагая, что у нас есть этот простой колодец, мы можем представить напитки в виде точек в R4 с одним слотом для каждого ингредиента. Например, типичный «текила санрайз» может быть представлен точкой (0, 1,5, 6, 0,75), представляющей количество водки, текилы, апельсинового сока и гренадина (в унциях), соответственно.

Набор напитков, которые можно приготовить из типового колодца, содержится в

диапазон {(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)},

то есть все комбинации четырех основных ингредиентов. Однако бармен, стремящийся сэкономить время, может не заметить, что во многих напитках одинаковое соотношение апельсинового сока и гренадина, и смешать бутылки. Новый упрощенный колодец может быть проще для розлива, но может приготовить меньше напитков:

диапазон {(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 6, 0,75)}

Пример 0.4 (многочлены). Определите pk(x) x ^{K}. Тогда легко увидеть, что

R[x] = диапазон {pk : k 0}.

Убедитесь, что вы достаточно хорошо понимаете нотацию, чтобы понять, почему это так.

Добавление другого элемента в набор векторов не всегда увеличивает размер его диапазона. Для Например, в R2 очевидно, что

диапазон {(1, 0), (0, 1)} = диапазон {(1, 0), (0, 1), (1, 1)}.

В этом случае говорят, что множество {(1, 0), (0, 1), (1, 1)} линейно зависимо:

Определение 0.3 (Линейная зависимость). Мы даем три эквивалентных определения. Множество S V векторов линейно зависимо, если:

1. Один из элементов S может быть записан как линейная комбинация других элементов, или S содержит нуль.

2. Существует непустая линейная комбинация элементов vk S , дающая ck = 0 для всех k. $\sum_{k=1}^{M}$ ckvk = 0, где

3. Существует v S такой, что span S = span S\{v}. То есть мы можем удалить вектор из S без влияет на его размах.

Если S не является линейно зависимым, то говорят, что оно линейно независимо.

Предоставление доказательств или неофициальных свидетельств того, что каждое определение эквивалентно своим аналогам (по принципу «если и только если»), является полезным упражнением для студентов, менее знакомых с обозначениями и абстрактной математикой.

Концепция линейной зависимости приводит к идее «избыточности» набора векторов. В этом смысле естественно спросить, насколько большой набор мы можем выбрать, прежде чем добавление другого вектора не может увеличить диапазон. В частности, предположим, что у нас есть линейно независимое множество S V, и теперь мы выбираем дополнительный вектор v V. Добавление v к S приводит к одному из двух возможных результатов:

1. Размах S {v} больше размаха S.

2. Добавление v к S не влияет на диапазон.

Размерность V — это не что иное, как максимальное количество раз, когда мы можем получить результат 1, добавить v к S и повторить.

Определение 0.4 (размерность и базис). Размерность V — это максимальный размер |S| линейно независимого множества S V такого, что span S = V. Любое множество S, удовлетворяющее этому свойству, называется базисом для V.

Пример 0.5 (Rn). Стандартным базисом для Rn является множество векторов вида

$$e_k = (0, \ldots, \underbrace{0}_{k-1 \text{ слотов}}, 1, \underbrace{0, \ldots, 0}_{n-k \text{ слотов}}).$$

То есть ek имеет все нули, кроме одного в k-м слоте. Ясно, что эти векторы линейно независимы и образуют базис; например, в R3 любой вектор (a, b, c) может быть записан как ae1 + be2 + ce3.
Таким образом, размерность Rn равна n, как и следовало ожидать.

Пример 0.6 (многочлены). Ясно, что множество $\{1, x, x^2, x^3, \ldots\}$ — линейно независимый набор полиномов, охватывающий R[x]. Обратите внимание, что это множество бесконечно велико, и поэтому размерность R[x] равна .

0.2.3 Наш фокус: Rn

Особое значение для наших целей имеет векторное пространство Rn клидово , так называемый n-мерный Eu пространство. Это не что иное, как набор координатных осей, встречающихся на уроках математики в средней школе:

- R1 R — числовая прямая
- R2 — двумерная плоскость с координатами (x, y)
- R3 — трехмерное пространство с координатами (x, y, z)

Почти все методы этого курса будут иметь дело с преобразованиями и функциями на Rn. .

Для удобства мы обычно записываем векторы в Rn в «столбцовой форме» следующим образом

$$(a_1, \ldots, a_n) \qquad \begin{matrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{matrix}$$

Эта нотация будет включать векторы как частные случаи матриц, обсуждаемые ниже.

В отличие от некоторых векторных пространств, Rn имеет не только структуру векторного пространства, но и одну дополнительную конструкцию, которая имеет все значение: скалярное произведение.

Определение 0,5 (Скалярный продукт). Скалярное произведение двух векторов a = (a1, ..., an) и b = (b1, ..., bn) в Rn определяется выражением

$$a \cdot б = \sum_{к=1}^{н} акбк .$$

Пример 0.7 (R2). Скалярное произведение (1, 2) и (2, 6) равно 1 · 2 + 2 · 6 = 2 + 12 = 10.

Скалярное произведение является примером метрики, и его существование дает Rn понятие геометрии. · Например, мы можем использовать теорему Пифагора, чтобы определить норму или длину вектора как квадратный корень

$$\sqrt{2\,2 + \cdot\,\cdot + a\,a2}_{н} = \sqrt{a \cdot a.}$$

Тогда расстояние между двумя точками a,b Rn равно просто b a2.

Скалярные произведения дают не только представления о длинах и расстояниях, но и об углах. Напомним следующее тождество из тригонометрии для a,b R3 :

$$a \cdot b = ab \cos \theta$$

где θ — угол между a и b. Однако для n 4 понятие «угол» гораздо труднее визуализировать для Rn . Мы могли бы определить угол θ между a и b как значение θ , заданное выражением

$$\theta \quad \arccos \frac{a \cdot б}{аб}.$$

Мы должны сделать нашу домашнюю работу, прежде чем давать такое определение! В частности, вспомните, что косинус выводит значения в интервале [1, 1], поэтому мы должны проверить, что входные данные арккосинуса (также обозначаемого как cos 1) находятся в этом интервале; к счастью, известное неравенство Коши-Шварца a ·b ab гарантирует именно это свойство.

Когда a = cb для некоторого c R, мы имеем θ = arccos 1 = 0, как и следовало ожидать: угол между параллельными векторами равен нулю. Что означает перпендикулярность векторов? Подставим θ = 90 . Тогда у нас есть

$$0 = \cos 90$$

$$"=" \frac{a \cdot б}{аб}$$

Умножение обеих сторон на ab мотивирует определение:

Определение 0.6 (ортогональность). Два вектора перпендикулярны или ортогональны, когда a · b = 0.

Это определение несколько неожиданно с геометрической точки зрения. В частности, нам удалось определить, что значит быть перпендикулярным, без явного использования углов. Эта конструкция облегчит решение некоторых задач, для которых нелинейность синуса и косинуса могла бы создать головную боль в более простых условиях.

В сторону 0,1. Здесь нужно обдумать множество теоретических вопросов, некоторые из которых мы рассмотрим в следующих главах, когда они станут более мотивированными:

- Все ли векторные пространства допускают скалярные произведения или аналогичные структуры?
- Все ли конечномерные векторные пространства допускают скалярное произведение?
- Каким может быть разумное скалярное произведение между элементами R[x]?

Заинтересованные студенты могут ознакомиться с текстами по реальному и функциональному анализу.

0.3 Линейность

Функция между векторными пространствами, которая сохраняет структуру, известна как линейная функция:

Определение 0.7 (Линейность). Предположим, что V и V — векторные пространства. Тогда L : V V линейна, если она удовлетворяет следующим двум критериям для всех v,v1,v2 V и c R:

- L сохраняет суммы: L[v1 +v2] = L[v1] + L[v2]
- L сохраняет скалярные произведения: L[cv] = cL[v]

Легко генерировать линейные карты между векторными пространствами, как мы можем видеть в следующих примерах:

Пример 0.8 (Линейность по Rn). Следующее отображение f : R2 R3 является линейным:

$$f(x, y) = (3x, 2x + y, \quad y)$$

Мы можем проверить линейность следующим образом:

- Сохранение суммы:

$$\begin{aligned} f(x1 + x2, y1 + y2) &= (3(x1 + x2), 2(x1 + x2) + (y1 + y2), (y1 + y2)) = \\ &(3x1, 2x1 + y1, \quad y1) + (3x2, 2x2 + y2, \quad y2) \\ &= f(x1, y1) + f(x2, y2) \end{aligned}$$

- Сохранение скалярного произведения:

$$\begin{aligned} f(cx, cy) &= (3cx, 2cx + cy, \quad cy) = c(3x, 2x + y, \quad y) \\ &= c\, f(x, y) \end{aligned}$$

Напротив, g(x, y) xy2 не является линейным. Например, g(1, 1) = 1, но g(2, 2) = 8 = 2 · g(1, 1), так что эта форма не сохраняет скалярные произведения.

Пример 0.9 (Интеграция). Следующий «функционал» L от R[x] до R является линейным:

$$L[p(x)] \qquad \int_0^1 p(x)\, dx.$$

Этот несколько более абстрактный пример отображает многочлены p(x) в действительные числа L[p(x)]. Например, мы можем написать

$$L[3x^2 + x - 1] = \int_0^1 (3x^2 1 + x - 1)\,dx = 2 \quad -.$$

Линейность исходит из следующих хорошо известных фактов из исчисления:

$$\int_0^1 c \cdot f(x)\,dx = c \int_0^1 f(x)\,dx$$

$$\int_0^1 [f(x) + g(x)]\,dx = \int_0^1 f(x)\,dx + \int_0^1 g(x)\,dx$$

Мы можем написать особенно красивую форму для линейных отображений на Rn . Напомним, что вектор a = (a1, ..., an) равен сумме k akek , где ek — k-й стандартный базисный вектор. Тогда, если L линейно, мы знаем:

$$L[a] = L \Big(\sum_k a_k e_k \text{ для стандартного базисека} \Big)$$

$$= \sum_k L[a_k e_k] \text{ по сохранению суммы}$$

$$= \sum_k a_k L[e_k] \text{ по сохранению скалярного произведения}$$

Этот вывод показывает следующий важный факт:

L полностью определяется его действием на стандартной основе vectorsek .

То есть для любого вектора a , мы можем использовать приведенную выше сумму, чтобы определить L [a] путем линейного комбинирования Rn L[e1], . . . ,L[en].

Пример 0.10 (Расширение линейной карты). Вспомним карту из примера 0.8, заданную формулой f(x, y) = (3x, 2x + y, y). Имеем f(e1) = f(1, 0) = (3, 2, 0) и f(e2) = f(0, 1) = (0, 1, 1). Таким образом, приведенная выше формула показывает:

$$f(x, y) = x\, f(e_1) + y\, f(e_2) = x \begin{matrix} 3 \\ 2 \\ 0 \end{matrix} + y \begin{matrix} 0 \\ 1 \\ 1 \end{matrix}$$

0.3.1 Матрицы

Расширение линейных карт выше предлагает один из многих контекстов, в которых полезно хранить несколько векторов в одной и той же структуре. В более общем случае, скажем, у нас есть n векторов v1, . . . ,vn Rm. Мы можем записать каждый как вектор-столбец:

$$v_1 = \begin{matrix} v_{11} \\ v_{21} \\ \vdots \\ v_{m1} \end{matrix}, v_2 = \begin{matrix} v_{12} \\ v_{22} \\ \vdots \\ v_{m2} \end{matrix}, \cdots, v_n = \begin{matrix} v_{1n} \\ v_{2n} \\ \vdots \\ v_{mn} \end{matrix}$$

Носить их по отдельности может быть громоздко, поэтому для упрощения мы просто объединяем их в единую матрицу размера m × n:

$$\begin{array}{c} | \quad | \quad\quad | \\ v1\ v2 \cdots vn \\ | \quad | \quad\quad | \end{array} \quad "=" \quad \begin{array}{cccc} v11 & v12 & \cdots & v1n \\ v21 & v22 & \cdots & v2n \\ \vdots & \vdots & \vdots & \vdots \\ вм1 & вм2 & \cdots & вмн \end{array}$$

Будем называть пространство таких матриц $R^{m\times n}$.

Пример 0.11 (Идентификационная матрица). Мы можем сохранить стандартный базис для R^n в «единичной матрице» размера $n \times n$. $I_{n\times n}$ определяется как:

$$B \times п \quad \begin{array}{c} | \quad | \quad\quad | \\ e1\ e2 \cdots en \\ | \quad | \quad\quad | \end{array} \quad "=" \quad \begin{array}{ccccc} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0\ 0\ 0 \cdots 0\ 1 \end{array}$$

Поскольку мы построили матрицы как удобные способы хранения наборов векторов, мы можем использовать умножение, чтобы выразить, как их можно комбинировать линейно. В частности, матрицу в $R^{m\times n}$ можно умножить на вектор-столбец в R^n следующим образом:

$$\begin{array}{c} | \quad | \quad\quad | \\ v1\ v2 \cdots vn \\ | \quad | \quad\quad | \end{array} \quad \begin{array}{c} c1 \\ c2 \\ \vdots \\ Сп \end{array} \quad c1v1 + c2v2 + \cdots + cnvn$$

Расширение этой суммы дает следующую явную формулу для произведений матрицы на вектор:

$$\begin{array}{cccc} v11 & v12 & \cdots & v1n \\ v21 & v22 & \cdots & v2n \\ \vdots & \vdots & \vdots & \vdots \\ вм1 & вм2 & \cdots & вмн \end{array} \quad \begin{array}{c} c1 \\ c2 \\ \vdots \\ Сп \end{array} \quad = \quad \begin{array}{l} c1v11 + c2v12 + \cdots + cnv1n\ c1v21 \\ + c2v22 + \cdots + cnv2n \\ \vdots \\ c1vm1 + c2vm2 + \cdots + cnvmn \end{array}$$

Пример 0.12 (умножение единичной матрицы). Ясно, что для любого $x \in R^n$ $x = I_{n\times n}x$, где $I_{n\times n}$ — , мы можем написать единичная матрица из примера 0.11.

Пример 0.13 (Линейная карта). Вернемся еще раз к выражению из примера 0.8, чтобы показать еще одну альтернативную форму:

$$e(x, y) = \begin{array}{cc} 3 & 0\ 2\ 1 \\ & \\ 0 & 1 \end{array} \quad \begin{array}{c} \text{Икс} \\ \\ y \end{array}$$

Аналогичным образом мы определяем произведение между матрицей в $M \in R^{m\times n}$ и другой матрицей в $R^{n\times p}$ путем объединения отдельных произведений матриц-векторов:

$$M \quad \begin{array}{c} | \quad | \quad\quad | \\ c1\ c2 \cdots сп \\ | \quad | \quad\quad | \end{array} \quad \begin{array}{c} | \quad\quad | \quad\quad | \\ Mc1\ Mc2 \cdots Mcn \\ | \quad\quad | \quad\quad | \end{array}$$

Пример 0.14 (Миксология). Продолжая пример 0.3, предположим, что мы делаем текилу восход солнца и вторую смесь с равными частями двух ликеров в нашем упрощенном колодце. Чтобы узнать, сколько базовых ингредиентов содержится в каждом заказе, мы могли бы объединить рецепты для каждого столбца и использовать матрицу. умножение:

	Скважина 1	Скважина 2	Скважина 3
Водка	1	0	0
Текила	0	1	0
ОЖ	0	0	6
Гренадин	0	0	0,75

·

	Напиток 1	Напиток 2
	0	0,75
	1,5	0,75
	1	2

"="

		Напиток 1	Напиток 2
0	0,75	1,5	6
		0,75	Текила
		12	ОЖ
0,75		1,5	Гренадин

Водка

В общем, мы будем использовать заглавные буквы для обозначения матриц, например, $A \in \mathbb{R}^{m\times n}$. Мы будем использовать обозначение $A_{ij} \in \mathbb{R}$ для обозначения элемента A в строке i и столбце j.

0.3.2 Скаляры, векторы и матрицы

Неудивительно, что мы можем записать скаляр как вектор 1×1 с $\mathbb{R}^{1\times 1}$. Аналогично, как мы уже предложенный в §0.2.3, если мы запишем векторы в $\mathbb{R}^n$ в виде столбцов, их можно рассматривать как матрицы размера $n \times 1$. $v \in \mathbb{R}^{n\times 1}$. Обратите внимание, что в этом контексте можно легко интерпретировать произведения матрицы-вектора; например, если $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, и $b \in \mathbb{R}^m$, то мы можем записать выражения вида

$$\underset{m\times n}{A}\ \underset{n\times 1}{x} = \underset{m\times 1}{b}$$

Мы введем один дополнительный оператор на матрицах, полезный в этом контексте:

Определение 0.8 (Транспонирование). Транспонированием матрицы $A \in \mathbb{R}^{m\times n}$ называется матрица $A \in \mathbb{R}^{n\times m}$ с элементами $(A^\top)_{ij} = A_{ji}$.

Пример 0.15 (Транспозиция). Транспонирование матрицы

$$A = \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix}$$

дан кем-то

$$A^\top = \begin{matrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{matrix}.$$

Геометрически транспонирование можно представить как переворачивание матрицы по диагонали.

Эта унифицированная обработка скаляров, векторов и матриц в сочетании с такими операциями, как транспозиция и умножение, может привести к гладким выводам хорошо известных тождеств. Например,

мы можем вычислить скалярное произведение векторов sa,b Rn , выполнив следующую серию шагов:

$$a \cdot b = \sum_{k=1}^{n} a_k b_k$$

$$= a_1\ a_2 \cdots a_n \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$= ab$$

Многие важные тождества из линейной алгебры могут быть получены путем объединения этих операций с помощью нескольких правил:

$$(A\ ) = A$$

$$(A + B) = A + B$$

$$(AB) = BA$$

Пример 0.16 (Остаточная норма). Предположим, у нас есть матрица A и два вектора x и b. Если мы хотим знать, насколько хорошо Ax аппроксимирует b, мы можем определить невязку r b Ax; эта невязка равна нулю ровно тогда, когда Ax = b. В противном случае мы могли бы использовать норму r2 в качестве показателя отношения между Ax и b.

Мы можем использовать приведенные выше тождества для упрощения:

$$\begin{aligned} \|r\|_2^2 &= \|b - Ax\|_2^2 \\ &= (b - Ax) \cdot (b - Ax), \text{ как объяснено в §0.2.3} \\ &= (b - Ax)(b - Ax) \text{ с помощью нашего выражения для скалярного} \\ &\text{произведения выше} = (b^T - x^T A^T)(b - Ax) \text{ по свойствам} \\ &\text{транспонирования} = b^T b - b^T Ax - x^T A^T b + x^T A^T Ax \text{ после умножения} \end{aligned}$$

Все четыре члена в правой части являются скалярами или, что эквивалентно, матрицами 1 × 1. Скаляры, рассматриваемые как матрицы, тривиально обладают одним дополнительным приятным свойством c = c, поскольку транспонировать нечего! Таким образом, мы можем написать

$$x^T A^T b = (x^T A^T b)^T = b^T Ax$$

Это позволяет нам еще больше упростить наше выражение:

$$\begin{aligned} \|r\|_2^2 &= b^T b - 2b^T Ax + x^T A^T Ax \\ &= \text{топор}\ \|_2^2 - 2b^T Ax + \|b\|_2^2 \end{aligned}$$

Мы могли бы получить это выражение, используя тождества скалярного произведения, но промежуточные шаги, описанные выше, окажутся полезными в нашем дальнейшем обсуждении.

0.3.3 Модельная задача: Ax = b

На вводном уроке алгебры студенты тратят много времени на решение линейных систем, таких как следующие для троек (x, y, z):

$$\begin{aligned} 3x + 2y + 5z &= 0 \\ 4x + 9y \quad 3z &= \quad 7\ 2x \\ \quad 3y \quad 3z &= 1 \end{aligned}$$

Наши конструкции в §0.3.1 позволяют нам кодировать такие системы более чистым способом:

$$\begin{matrix} 3 & 2 & 5 \\ 4 & 9 & 3 \\ 2 & 3 & 3 \end{matrix} \quad \begin{matrix} \text{Икс} \\ y \\ г \end{matrix} \quad \text{"="} \quad \begin{matrix} 0 \\ 7 \\ 1 \end{matrix}$$

В более общем смысле мы можем записать линейные системы уравнений в форме Ax = b, следуя той же схеме, что и выше; здесь вектор x неизвестен, а A и b известны. Такая система уравнений не всегда имеет решение. Например, если A содержит только нули, то очевидно, что ни один x не будет удовлетворять Ax = b всякий раз, когда b = 0. Мы отложим общее рассмотрение того, когда решение существует, до нашего обсуждения линейных решателей в следующих главах.

Ключевая интерпретация системы Ax =b заключается в том, что она решает задачу:

Запишите b как линейную комбинацию столбцов матрицы A.

Почему? Напомним из §0.3.1, что произведение Ax кодирует линейную комбинацию столбцов A с весами, содержащимися в элементах x. Итак, уравнение Ax =b требует, чтобы линейная комбинация Ax равнялась заданному вектору b. Учитывая эту интерпретацию, мы определяем пространство столбца A как пространство правых частей b, для которых система имеет решение:

Определение 0.9 (столбцовое пространство). Пространство столбцов матрицы A Rm×n является оболочкой столбцов матрицы A. Мы можем записать как

$$\text{col } A \quad \{ Ax : x \quad Rn \}.$$

Несколько легче рассмотреть один важный случай. Предположим, что A квадрат, поэтому мы можем написать A Rn×n . Кроме того, предположим, что система Ax = b имеет решение для всех вариантов b. Таким образом, согласно единственное условие на b состоит в том, что он является членом， нашей вышеприведенной интерпретации Ax =b, мы можем Rn , чтобы сделать вывод, что столбцы A охватывают Rn .

В этом случае, поскольку линейная система всегда разрешима, предположим, что мы подставляем стандартный базис e1, . . . ,en для получения векторов x1, . . . ,xn , удовлетворяющих условию Axk = ek для каждого k. Затем мы можем «сложить» эти выражения, чтобы показать:

$$A \quad \begin{vmatrix} | & | & & | \\ x1 & x2 & \cdots & xn \\ | & & & | \end{vmatrix} \quad \text{"="} \quad \begin{vmatrix} | & | & & | \\ Ax1 & Ax2 & \cdots & \text{Аксон} \\ | & & & | \end{vmatrix} \quad \text{"="} \quad \begin{vmatrix} | & | & & | \\ e1 & e2 & \cdots & en \\ | & & & | \end{vmatrix} = B \times n,$$

где In×n — единичная матрица из примера 0.11. Будем называть матрицу со столбцами xk обратной A 1 , который удовлетворяет

$$AA\text{-}1 = A\text{-}1A = In \times n.$$

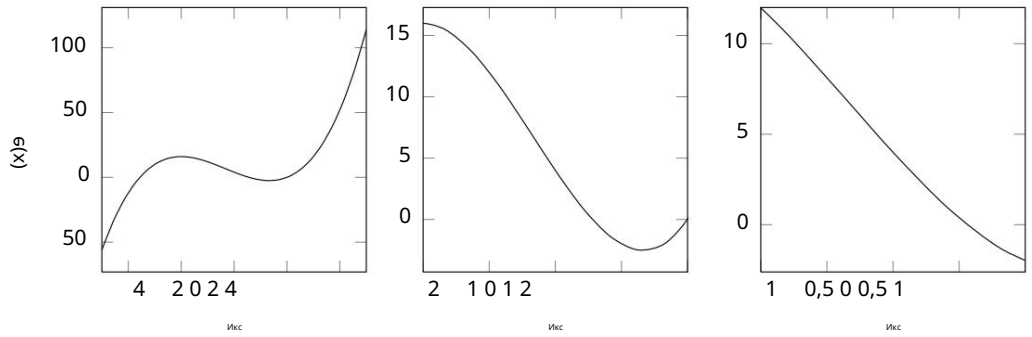


Рисунок 1: Чем ближе мы приближаемся к f(x) = x 3 2 + x 8x + 4, тем больше это похоже на линию.

Также легко проверить, что (A = A. Когда такая обратная существует, легко решить систему Ax = b. В частности, мы 1) 1
находим:

x = In×nx = (A 1A)x = A 1 (Ax) = A -1b

0.4 Нелинейность: дифференциальное исчисление

В то время как красота и применимость линейной алгебры делает ее ключевым объектом изучения, нелинейности изобилуют в природе, и нам часто приходится проектировать вычислительные системы, которые могут справиться с этим фактом жизни. В конце концов, на самом базовом уровне квадрат в знаменитом соотношении E = mc2 делает его мало поддающимся линейному анализу.

0.4.1 Дифференциация

Хотя многие функции глобально нелинейны, локально они демонстрируют линейное поведение. Эта идея «локальной линейности» является одним из основных мотивов дифференциального исчисления. Например, на рисунке 1 показано, что если вы достаточно приблизите гладкую функцию, в конечном итоге она станет похожа на линию. Производная f (x) функции f (x) : R R есть не что иное, как наклон аппроксимирующей линии, вычисляемый путем нахождения наклона линий, проходящих через все более и более близкие к x точки:

$$= \lim_{y \to x} \frac{f(y) \quad f(x)\ f(x)}{y - x}$$

Мы можем выразить локальную линейность, записав f(x + x) = f(x) + x · f (x) + O(x 2).

Если функция f принимает несколько входных данных, то ее можно записать как f(x) : Rn R для x Rn ; иными словами, каждой точке x = (x1, ..., xn) в n-мерном пространстве f ставит в соответствие единственное число f(x1, ..., xn). Наше представление о локальной линейности здесь несколько нарушается, потому что линии — это одномерные объекты. Однако фиксация всех переменных, кроме одной, сводится к случаю исчисления одной переменной. Например, мы могли бы написать g(t) = f(t, x2, ..., xn), где мы просто фиксируем константы x2, . . . , xn. Тогда g(t) — дифференцируемая функция одной переменной. Конечно, мы могли бы поместить t в любой из входных слотов для f, поэтому в общем случае мы даем следующее определение частной производной от f:

f Определение 0.10 (частная производная). k-я частная производная f , обозначенная как xk , — дается по-разному связывающая f в своей k-й входной переменной:

$$\frac{\partial}{\partial x_k} f(x_1, \ldots, x_n) \triangleq \frac{d}{dt} f(x_1, \ldots, x_{k-1}, t, x_{k+1}, \ldots, x_n) \Big|_{t=x_k}$$

Обозначение « |t=xk » следует читать как «оценивается при t = xk ».

Пример 0.17 (Относительность). Соотношение E = mc2 можно рассматривать как функцию от m и c к E.
Таким образом, мы могли бы написать E(m, c) = mc2 , что дало бы производные

$$\frac{\partial E}{\partial m} = c^2$$

$$\frac{\partial E}{\partial c} = 2mc$$

Используя исчисление с одной переменной, мы можем написать:

$$\begin{aligned} f(x + \Delta x) &= f(x_1 + \Delta x_1, x_2 + \Delta x_2, \ldots, x_n + \Delta x_n) \\ &= f(x_1, x_2 + \Delta x_2, \ldots, x_n + \Delta x_n) + \frac{\partial f}{\partial x_1} \Delta x_1 + O(\Delta x_1^2) \text{ с помощью исчисления с одной переменной} \\ &= f(x_1, \ldots, x_n) + \sum_{k=1}^{n} \frac{\partial f}{\partial x_k} \Delta x_k + O(\Delta x_k^2), \text{ повторив это n раз} \\ &= f(x) + \nabla f(x) \cdot \Delta x + O(\Delta x^2) \end{aligned}$$

где мы определяем градиент f как

$$\nabla f \triangleq \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right] \in \mathbb{R}^n$$

Из этого соотношения легко видеть, что f можно дифференцировать по любому направлению v; мы можем оценить эту производную Dv f следующим образом:

$$D_v f(x) \triangleq \frac{d}{dt} f(x + tv) \Big|_{t=0} dt = \nabla f(x) \cdot v$$

Пример 0.18 (R2). Возьмем f(x, y) = x 2y 3 . Затем,

$$\frac{\partial f}{\partial x} = 2xy^3$$

$$\frac{\partial f}{\partial x} = 3x^2y^2$$

Таким образом, мы можем написать f(x, y) = (2xy3 ,3x 2y 2). Производная f в точке (1, 2) в направлении (1, 4) определяется выражением (1, 4) · f(1, 2) = (1, 4) · (16, 12) = 32.

Пример 0.19 (Линейные функции). Очевидно, но стоит отметить, что градиент функции f(x) a · x +c = (a1x1 + c1, ..., anxn + cn) равен .

Пример 0.20 (Квадратичные формы). Возьмите любую матрицу A , и определить f(x) x Ax. Расширение Rn×n, эта функция поэлементно показывает

$$e(x) = \sum_{ij} Aijxixj ;$$

Стоит расширить f и явно проверить это отношение. Возьмем некоторое k {1, . . . , п}. Затем мы можем выделить все термины, содержащие xk :

$$f(x) = Akkx_k^2 + xk \sum_{я=к} Айкси + \sum_{j=k} Akjxj + \sum_{i,j=k} Aijxixj$$

С помощью этой факторизации легко увидеть

$$\frac{\phi}{xk} = 2Akkxk + \sum_{я=к} Айкси + \sum_{j=k} Akjxj$$

$$\overset{"="}{=} \sum_{я=1}^{н} (Айк + Аки)xi$$

Эта сумма есть не что иное, как определение умножения матрицы на вектор! Таким образом, мы можем написать

$$f(x) = Ax + A\,x.$$

Мы обобщили f : R R на f : Rn R. Чтобы достичь полной общности, мы хотели бы рассмотреть f : Rn Rm. Другими словами, f принимает n чисел и выводит m чисел. К счастью, это расширение является простым, потому что мы можем думать о f как о наборе однозначных функций f1, . . . , fm : Rn R слились в один вектор. То есть пишем:

$$e(x) = \begin{matrix} f1(x) \\ f2(x) \\ \vdots \\ фм(x) \end{matrix}$$

Каждую fk можно дифференцировать, как и раньше, поэтому в итоге мы получаем матрицу частных производных, называемую якобианом f:

Определение 0.11 (якобиан). Якобианом f : Rn Rm называется матрица D f Rm×n с элементами

$$(Df)ij \quad \frac{\phi и}{xj}.$$

Пример 0.21 (Простая функция). Предположим, что f(x, y) = (3x, xy2 , x + y). Затем,

$$D\,f(x, y) = \begin{matrix} 3 & \\ y^2 & 0 \\ 1 & 2xy\ 1 \end{matrix}.$$

Убедитесь, что вы можете произвести это вычисление вручную.

Пример 0.22 (умножение матриц). Неудивительно, что якобиан функции f(x) = Ax для матрицы A равен D f(x) = A.

Здесь мы сталкиваемся с общей точкой путаницы. Предположим, что функция имеет векторный вход и скалярный выход, то есть f : Rn R. Мы определили градиент f как вектор-столбец, поэтому, чтобы согласовать это определение с определением якобиана, мы должны написать

D ж знак равно ж .

0.4.2 Оптимизация

Напомним, что из исчисления одной переменной минимумы и максимумы f : R R должны встречаться в точках x, удовлетворяющих f (x) = 0. Конечно, это условие скорее необходимое, чем достаточное: могут существовать точки x c f (x) = 0 которые не являются максимумами или минимумами. Тем не менее, нахождение таких критических точек f может быть шагом алгоритма минимизации функции, если следующий шаг гарантирует, что результирующий x действительно является минимумом/максимумом.

Если f : Rn R минимизируется или максимизируется в точке x, мы должны гарантировать, что не существует единственного направления x от x, в котором f уменьшается или увеличивается, соответственно. Согласно обсуждению в §0.4.1, это означает, что мы должны найти точки, для которых f = 0.

Пример 0.23 (Простая функция). Предположим, что f(x, y) = x 2x + 8y. 2 + 2xy + 4y 2 и x y ___ж ж . Тогда = 2x + 2y __ "=" Таким образом, критические точки f удовлетворяют:

2x + 2y = 0 2x +

8y = 0

Ясно, что эта система решается при (x, y) = (0, 0). В самом деле, это минимум f , что можно увидеть более четко, написав f(x, y) = (x + y) 2 + 3 года 2.

Пример 0.24 (Квадратичные функции). Предположим, что f(x) = x Ax + bx + c. Тогда из примеров в предыдущем разделе мы можем написать f(x) = (A + A)x +b. Таким образом, критические точки x функции f удовлетворяют условию (A + A)x + b = 0.

В отличие от исчисления с одной переменной, когда мы выполняем исчисление на Rn , мы можем добавлять ограничения к нашей оптимизации. Наиболее общий вид такой задачи выглядит так:

минимизировать

f(x) так, чтобы g(x) =0

Пример 0.25 (прямоугольные области). Предположим, прямоугольник имеет ширину w и высоту h. Классическая задача геометрии состоит в том, чтобы максимизировать площадь с фиксированным периметром 1:

максимизировать wh

такое, что 2w + 2h - 1 = 0

Когда мы добавляем это ограничение, мы больше не можем ожидать, что критические точки удовлетворяют f(x) = 0, поскольку эти точки могут не удовлетворять g(x) = 0.

Пока предположим, что g : Rn R. Рассмотрим множество точек S0 {x : g(x) = 0}. Очевидно, для любых двух x,y S0 выполняется соотношение g(y) g(x) = 0 0 = 0. Предположим, что y = x + x для малых x. Тогда g(y) g(x) = g(x) · x + O(x 2). Другими словами, если мы начнем с x, удовлетворяющего g(x) = 0, затем, если мы сместим в направлении x, g(x) · x 0, чтобы продолжать удовлетворять этому соотношению.

Теперь вспомним, что производная от f по направлению v в точке x задается как f ·v. Если x является минимумом задачи оптимизации с ограничениями, описанной выше, то любое небольшое смещение x на x + v должно вызывать увеличение от f(x) до f(x + v). Поскольку нас интересуют только перемещения v, сохраняющие ограничение g(x + v) = c, из нашего рассуждения выше мы хотим, чтобы f · v = 0 для всех v, удовлетворяющих g(x)·v = 0. Другими словами, f и g должны быть параллельны, условие, которое мы можем записать как f = λ g для некоторого λ R.

Определять

$$\Lambda(x, \lambda) = f(x) \quad \lambda g(x).$$

Тогда критические точки Λ без ограничений удовлетворяют условию:

$$0 = \frac{\Lambda}{\lambda} = -\nabla(x)$$

$$0 = \quad x\Lambda = \quad f(x) \quad \lambda \quad g(x)$$

Другими словами, критические точки Λ удовлетворяют условиям g(x) = 0 и f(x) = λ g(x) — именно тем условиям оптимальности, которые мы получили!

Расширение до многомерных ограничений дает следующее:

Теорема 0.1 (метод множителей Лагранжа). Критические точки приведенной выше задачи оптимизации с ограничениями являются неограниченными критическими точками функции множителя Лагранжа.

$$\Lambda(x,\lambda) \quad f(x) \quad \lambda \cdot g(x),$$

как по x, так и по λ.

Пример 0.26 (Максимизация площади). Продолжая пример 0.25, определим функцию множителя Лагранжа Λ(w, h, λ) = wh λ(2w + 2h 1). Дифференцируя, находим:

$$w \quad \frac{\Lambda}{\Lambda} = h \quad 2\lambda \; 0 =$$

$$= w \quad \frac{0 =}{2\lambda}$$

$$0 = \quad \frac{h}{\lambda} \quad \Lambda = 1 \quad 2w \quad 2h$$

Итак, критические точки системы удовлетворяют

$$\begin{array}{lll} 0\,1 & 2 & ж \quad 0 \\ 1\,0 & 2 & ч \quad "=" \quad 0 \\ 2\,2\,0 & & \lambda \quad 1 \end{array}$$

Решение системы показывает w = h = 1/4 и λ = 1/8. Другими словами, при фиксированном периметре прямоугольник с максимальной площадью является квадратом.

Пример 0.27 (собственные задачи). Предположим, что A — симметричная положительно определенная матрица, что означает A = A (симметрия) и x Ax > 0 для всех x Rn\{0} (положительно определенная). Часто мы хотим минимизировать $\frac{1}{2}$ x Ax при $\frac{2}{2}$ = 1 для данной матрицы A Rn×n ; обратите внимание, что без ограничения минимум тривиально имеет место при x = 0. Определим функцию множителя Лагранжа

$$\Lambda(x, \lambda) = x\,Ax \quad \lambda(x \quad 1) = x\,Ax \quad \lambda(x\,x \quad 1).$$

Дифференцируя по x, находим

$$0 = \quad x\Lambda = 2Ax \quad 2\lambda x$$

Другими словами, x является собственным вектором матрицы A:

$$Ax = \lambda x.$$

0.5 Проблемы

Проблема 0.1. В качестве C1 (R) возьмем множество функций f : R R , допускающих первую производную f (x). Почему $^{1}C_{-}$(R) векторное пространство? Докажите, что C1 (R) имеет размерность .

Задача 0.2. Предположим, что строки A Rm×n заданы транспонированием r1, . . . ,rm Rn , а столбцы A Rm×n задаются как c1, . . . ,cn Rm. То есть,

$$A = \begin{matrix} p\,1 \\ p\,2 \\ \vdots \\ p\,м \end{matrix} \quad \begin{matrix} | & | & | \\ c1\ c2 \cdots cn \\ | & | & | \end{matrix} = .$$

Выразите элементы AA и AA через эти векторы.

Задача 0.3. Приведите линейную систему уравнений, которой удовлетворяют минимумы энергии f(x) = Ax $\frac{2}{2}$ b относительно x для x Rn и A^{b} Rm. Эта система называется «нормальными уравнениями» и будет встречаться в других местах в этих примечаниях; даже в этом случае стоит проработать и полностью понять вывод.

Задача 0.4. Предположим, что A, B Rn×n . Сформулируйте условие того, что векторы x Rn являются критическими ^{Топор} $\frac{2}{2}$ точками при Bx = $\frac{1}{2}$. Также приведите альтернативную форму для оптимальных значений Ax $^{2}_{2}$.

Задача 0.5. Зафиксируем некоторый вектор a Rn\{0} и определим f(x) =a · x. Напишите выражение для максимума f(x) при x = 1.

Глава 1

Числа и анализ ошибок

Изучая численный анализ, мы переходим от работы с целыми и длинными числами к числам с плавающей запятой и удвоениям. Этот, казалось бы, невинный переход включает в себя огромный сдвиг в том, как мы должны думать о разработке и реализации алгоритма микрофона. В отличие от основ дискретных алгоритмов, мы больше не можем ожидать наши алгоритмы дают точные решения во всех случаях. «Большое О» и подсчет операций не всегда царствовать безраздельно; вместо этого, даже в понимании самых основных методов, мы вынуждены изучать компромисс между временем, ошибкой аппроксимации и так далее.

1.1 Хранение чисел с дробными частями

Напомним, что компьютеры обычно хранят данные в двоичном формате. В частности, каждая цифра положительного целое число соответствует другой степени двойки. Например, мы можем преобразовать 463 в двоичный код. используя следующую таблицу:

1	1	1	0	0	1			1	1	1
2^8	2^7	2^6		2^5		2^4	2^3	2^2	2^1	2^0

Другими словами, эта запись кодирует тот факт, что 463 можно разложить на степени двойки. однозначно как:

$$463 = 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$
$$= 256 + 128 + 64 + 8 + 4 + 2 + 1$$

Помимо проблем с переполнением, все положительные целые числа могут быть записаны в этой форме с использованием конечного числа цифры. Отрицательные числа также могут быть представлены таким образом, либо путем введения ведущего знака немного или с помощью трюка «дополнение до двух».

Такое разложение вдохновляет на простое расширение чисел, содержащих дроби: просто включают отрицательные степени двойки. Например, разложить 463,25 так же просто, как сложить два слоты:

1	1	1	0	0	1	3	2		1	1	1.	0	0	2		1
2^8	2^7	2^6		2^5		2^4			2^2	2^1				2^{-1}		2^{-2}

Однако, как и в десятичной системе, представление дробных частей чисел таким образом не так хорошо себя ведет, как представление целых чисел. Например, запись дроби 1/3 в двоичном формате дает выражение:

$$\frac{1}{3} = 0{,}0101010101\ldots$$

Такие примеры показывают, что во всех масштабах существуют числа, которые нельзя представить с помощью конечная двоичная строка. На самом деле такие числа, как π = 11.00100100001. . .2 имеют бесконечно длинные расширения независимо от того, какую (целочисленную) базу вы используете!

По этой причине при разработке вычислительных систем, выполняющих математические операции с R вместо Z, мы вынуждены делать приближения почти для любого достаточно эффективного числового представления. Это может привести ко многим путаницам при кодировании. Например, рассмотрим следующий C++ фрагмент:

```
двойной x = 1,0;
двойной y = x/3,0;
if ( x == y *3.0) cout << " Они равны! " ;
иначе cout << " Они НЕ равны. " ;
```

Вопреки интуиции, эта программа выводит «Они НЕ равны». Почему? Определение y делает приближение к 1/3 , поскольку его нельзя записать в виде завершающей двоичной строки, округляя к ближайшему числу, которое он может представлять. Таким образом, y*3.0 больше не умножает 3 на 1/3. Один из способов исправить эта проблема ниже:

```
двойной x = 1,0;
двойной y = x/3,0;
if ( fabs (x - y *3.0) < numeric_limits < double >:: epsilon ) cout << " Они равны! " ;
else cout << " Они НЕ равны. " ;
```

Здесь мы проверяем, что x и y*3.0 находятся в пределах некоторого допуска друг друга, а не проверяем точное равенство. Это пример очень важного момента:

Редко, если вообще когда-либо, оператор == и его эквиваленты должны использоваться для дробных значений. Вместо этого следует использовать некоторый допуск, чтобы проверить, равны ли числа.

Конечно, здесь есть компромисс: размер допуска определяет грань между равенством и «близкий, но не такой же», который необходимо тщательно выбирать для данного приложения.

Ниже рассмотрим несколько вариантов представления чисел на компьютере.

1.1.1 Представления с фиксированной точкой

Самый простой вариант хранения дробей — добавить фиксированную десятичную точку. То есть как в В приведенном выше примере мы представляем значения, сохраняя коэффициенты 0/1 перед степенями двойки, которые варьироваться от 2 k до 2 для некоторого k, Z. Например, представление всех неотрицательных значений между 0 и 127,75 с шагом 1/4 так же просто, как взять k = 2 и = 7; в этой ситуации мы представляем эти значения используют 9 двоичных разрядов, два из которых идут после десятичной точки.

Основное преимущество этого представления состоит в том, что почти все арифметические операции могут быть осуществляется по тем же алгоритмам, что и с целыми числами. Например, легко видеть, что

$$a + b = (a \cdot 2^{к} + б \cdot 2^{к}) \cdot 2^{к}.$$

Умножение нашего фиксированного представления на 2k гарантирует, что результат будет целым, поэтому это наблюдение по существу показывает, что сложение может быть выполнено с использованием целочисленного сложения, по существу, «игнорируя» десятичную точку. Таким образом, вместо того, чтобы использовать специализированное оборудование, ранее существовавшее целочисленное арифметико-логическое устройство (АЛУ) быстро выполняет математические операции с фиксированной точкой.

Арифметика с фиксированной точкой может быть быстрой, но она может страдать от серьезных проблем с точностью. В частности, часто бывает так, что вывод двоичной операции, такой как умножение или деление, может потребовать битов больше, чем операндов. Например, предположим, что мы включили одну десятичную точку точности и

хотите выполнить произведение 1/2 · 1/2 = 1/4. Мы пишем 0,12 × 0,12 = 0,012, что усекается до 0. В этой системе довольно просто разумно комбинировать числа с фиксированной точкой и получить неразумный результат.

Из-за этих недостатков большинство основных языков программирования по умолчанию не включают десятичный тип данных с фиксированной запятой. Однако скорость и регулярность арифметических операций с фиксированными точками может быть значительным преимуществом для систем, которые отдают предпочтение времени, а не точности. Фактически, некоторые младшие графические процессоры (GPU) реализуют только эти операции, поскольку для многих графических приложений достаточно точности в несколько десятичных знаков.

1.1.2 Представления с плавающей запятой

Одной из многих числовых проблем при написании научных приложений является разнообразие масштабов, которые могут появиться. Только химики имеют дело со значениями где-то между 9,11 × 10–31 и 6,022 × 1023. Такая невинная операция, как смена единиц измерения, может вызвать внезапный переход между этими режимами: одно и то же наблюдение, записанное в килограммах на световой год, будет выглядеть значительно иначе в мегатоннах. в секунду. Наша работа как числовых аналитиков состоит в том, чтобы написать программное обеспечение, которое может изящно переходить между этими шкалами, не накладывая на клиента неестественные ограничения на их методы.

Несколько понятий и наблюдений из искусства научных измерений имеют отношение к такому обсуждение. Во-первых, очевидно, что одно из следующих представлений более компактно, чем другое:

$$6{,}022 \times 1023 = 602,200,000,000,000,000,000,000$$

Кроме того, в отсутствие исключительного научного оборудования разница между 6,022 × 1023 и 6,022 × 1023 + 9,11 × 10–31 незначительна. Один из способов прийти к этому заключению — сказать, что 6,022 × 1023 имеет только три цифры точности и, вероятно, представляет некоторый диапазон возможных измерений [6,022 × 1023 — ε, 6,011 × 1023 + ε] для некоторого ε 0,001 × 1023 .

Наше первое наблюдение позволило компактифицировать наше представление 6,022 × 1023 , записав его в экспоненциальной записи. Эта система счисления отделяет «интересные» цифры числа от его порядка величины, записывая его в виде a × 10b для некоторых a 1 и b Z. Мы называем этот формат формой числа с плавающей запятой, потому что в отличие от установки с фиксированной запятой в §1.1.1, здесь десятичная точка «плавает» вверх. Мы можем описать системы с плавающей запятой, используя несколько параметров (CITE):

- База β N; для научного обозначения, описанного выше, основание равно 10.
- Точность p N , представляющая количество цифр в десятичной записи.
- Диапазон показателей [L, U] , представляющих возможные значения b

Такое расширение выглядит так:

$$\pm \underbrace{(d0 + d1 \cdot \beta^{-1} + d2 \cdot \beta^{-2} + \cdots + dp_{-1} \cdot \beta^{1-p})}_{\text{мантисса}} \times \underbrace{\beta^{6}}_{\text{экспонента}}$$

знак

где каждая цифра dk лежит в диапазоне [0, β 1] и b [L, U].

Представления с плавающей запятой обладают любопытным свойством, которое может неожиданным образом повлиять на программное обеспечение: интервалы между ними неравномерны. Например, количество значений, которые могут быть представлены между β и β , хотя обычно β > β . Чтобы понять возможную точность с данной системой счисления, мы определим машинную точность εm как

наименьшее εm > 0 такое, что 1 + εm представимо. Тогда такие числа, как β + εm, невозможно выразить в системе счисления, потому что εm слишком мало!

На сегодняшний день наиболее распространенным стандартом для хранения чисел с плавающей запятой является стандарт IEEE 754. Этот стандарт определяет несколько классов чисел с плавающей запятой. Например, число с плавающей запятой двойной точности записывается с основанием β = 2 (как и большинство чисел на компьютере), с одним битом знака ± , 52 цифрами для d и диапазоном показателей степени от -1022 до 1023. Стандарт также определяет, как хранить ± и такие значения, как NaN или «не-число», зарезервированные для результатов вычислений, таких как 10/0. Дополнительную точность можно получить, записав нормализованные значения с плавающей запятой и предполагая, что старшая значащая цифра d0 равна 1, а не записывая ее.

Стандарт IEEE также включает в себя согласованные варианты работы с конечным числом значений, которые могут быть представлены заданным конечным числом битов. Например, распространенная несмещенная стратегия округления вычислений — округление до ближайшего, привязки до четности, что нарушает равноудаленные привязки путем округления до ближайшего значения с плавающей запятой с четным наименее значащим (самым правым) битом. Обратите внимание, что существует много одинаково законных стратегий округления; выбор одного из них гарантирует, что научное программное обеспечение будет одинаково работать на всех клиентских машинах, реализующих один и тот же стандарт.

1.1.3 Дополнительные экзотические опции

Двигаясь вперед, мы будем предполагать, что десятичные значения хранятся в формате с плавающей запятой, если не указано иное. Это, однако, не означает, что других систем счисления не существует, и для конкретных приложений может потребоваться альтернативный выбор. Мы признаем некоторые из этих ситуаций здесь.

Головная боль, связанная с добавлением допусков для учета ошибок округления, может оказаться неприемлемой для некоторых приложений. Такая ситуация возникает в приложениях вычислительной геометрии, например, когда различие между почти и полностью параллельными линиями может быть трудно определить. Одним из решений может быть использование арифметики произвольной точности, то есть реализация арифметики без округления или ошибок любого рода.

Арифметика произвольной точности требует специальной реализации и тщательного рассмотрения того, какие типы значений вам нужно представлять. Например, может быть так, что для данного приложения достаточно рациональных чисел Q , которые можно записать в виде отношений a/b для a, b Z.
Основные арифметические операции можно выполнять в Q без потери точности. Например, легко увидеть

$$\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd} \qquad \frac{a}{b} \div \frac{c}{d} = \frac{ad}{bc}.$$

Арифметика в рациональных числах исключает существование оператора квадратного корня, поскольку такие значения, как 2, иррациональны. Кроме того, это представление неоднозначно, поскольку, например, a/b = 5a/5b.

В других случаях может быть полезно заключить ошибку в скобки, представив значения вместе с оценками ошибки в виде пары a,ε R; мы думаем о паре (a,ε) как о диапазоне a ± ε. Затем арифметические операции также обновляют не только значение, но и оценку ошибки, как в

$$(x \pm \varepsilon 1) + (y \pm \varepsilon 2) = (x + y) \pm (\varepsilon 1 + \varepsilon 2 + \text{ошибка}(x + y)),$$

где последний член представляет собой оценку ошибки, вызванной добавлением x и y.

1.2 Понимание ошибки

За исключением систем произвольной точности, описанных в § 1.1.3, почти каждое компьютеризированное представление действительных чисел с дробными частями вынуждено использовать округление и другие схемы приближения. Эта схема представляет собой один из многих источников приближений, обычно встречающихся в числовых системах:

- Ошибка усечения возникает из-за того, что мы можем представить только конечное подмножество всего возможного набора значений в R; например, мы должны обрезать длинные или бесконечные последовательности после десятичной точки до количества битов, которое мы хотим сохранить.
- Ошибка дискретизации возникает из-за нашей компьютеризированной адаптации исчисления, физики и других аспекты непрерывной математики. Например, мы делаем аппроксимацию

$$\frac{dy}{dx} \approx \frac{y(x+\varepsilon) - y(x)}{\varepsilon}.$$

  Мы узнаем, что это приближение является законным и полезным, но в зависимости от выбора ε оно может быть не совсем правильным.
- Ошибка моделирования возникает из-за неполного или неточного описания проблем, которые мы хотим решить. Например, моделирование для прогнозирования погоды в Германии может пренебречь коллективным взмахом крыльев бабочек в Малайзии, хотя смещения воздуха этими бабочками может быть достаточно, чтобы несколько нарушить погодные условия в других местах.
- Ошибка эмпирической константы возникает из-за плохого представления физических или математических констант. Например, мы можем вычислить π , используя последовательность Тейлора, которую мы обрываем раньше, и даже ученые могут даже не знать скорость света с точностью до некоторого числа цифр.
- Ошибка ввода может возникать из-за аппроксимаций параметров данной системы, созданных пользователем (и из-за опечаток!). Моделирование и численные методы могут использоваться для ответа на вопросы типа «что, если», в которых исследовательский выбор входных настроек выбирается только для того, чтобы получить некоторое представление о том, как ведет себя система.

Пример 1.1 (Вычислительная физика). Предположим, мы разрабатываем систему для имитации вращения планет вокруг Земли. Система по существу решает уравнение Ньютона F = ma путем интегрирования сил вперед во времени. Примеры источников ошибок в этой системе могут включать:

- Ошибка усечения: использование IEEE с плавающей запятой для представления параметров и выходных данных системы и усечение при вычислении результата.
- Ошибка дискретизации: замена ускорения a разделенной разностью
- Ошибка моделирования: Пренебрежение моделированием влияния Луны на движение Земли внутри планетарной орбиты. система
- Эмпирическая ошибка: ввод массы Юпитера только до четырех цифр.
- Ошибка ввода: пользователь может захотеть оценить стоимость отправки мусора в космос, а не рисковать накоплением в стиле ВАЛЛ-И на Земле, но может только оценить количество мусора, которое правительство готово выбросить за борт таким образом.

1.2.1 Ошибка классификации

Учитывая наше предыдущее обсуждение, следующие два числа можно рассматривать как имеющие одинаковую величину потенциальной ошибки:

$$1 \pm 0{,}01$$

$$105 \pm 0{,}01$$

Хотя он имеет размер в виде диапазона [1 - 0,01, 1 + 0,01], диапазон [105 - 0,01, 105 + 0,01], по-видимому, кодирует более надежное измерение, поскольку ошибка 0,01 намного меньше по отношению к 105, чем к 1.

Различие между этими двумя классами ошибок описывается путем разграничения между ab
растворимая ошибка и относительная ошибка:

Определение 1.1 (Абсолютная ошибка). Абсолютная погрешность измерения определяется разницей между приблизительным значением и лежащим в его основе истинным значением.

Определение 1.2 (Относительная ошибка). Относительная ошибка измерения определяется делением абсолютной ошибки на истинное значение.

Один из способов различить эти два вида ошибок — использовать единицы измерения, а не проценты.

Пример 1.2 (Абсолютная и относительная погрешность). Вот два эквивалентных утверждения в противоположных формах:

Абсолют: 2 дюйма ± 0,02 дюйма

Относительный: 2 дюйма ± 1%

В большинстве приложений истинное значение неизвестно; в конце концов, если бы это было не так, использование приближения вместо истинного значения могло бы быть сомнительным предложением. Есть два популярных способа решить эту проблему. Во-первых, просто быть консервативным при проведении вычислений: на каждом шаге брать максимально возможную оценку ошибки и распространять эти оценки вперед по мере необходимости. Такие консервативные оценки хороши тем, что, когда они малы, мы можем быть уверены, что наше решение полезно.

Альтернативное разрешение связано с тем, что вы можете измерить. Например, предположим, что мы хотим решить уравнение $f(x) = 0$ для x с заданной функцией $f : R \to R$. Мы знаем, что где-то существует корень $x0$, точно удовлетворяющий $f(x0) = 0$, но если бы мы знали это root наш алгоритм не был бы необходим в первую очередь. На практике наша вычислительная система может дать некоторый $xest$, удовлетворяющий $f(xest) = \varepsilon$ для некоторого ε с $|\varepsilon| \ll 1$. Возможно, мы не сможем оценить разность $x0 - xest$, поскольку $x0$ неизвестно. С другой стороны, просто оценивая f, мы можем вычислить $f(xest) - f(x0) = f(xest)$, поскольку мы знаем, что $f(x0) = 0$ по определению. Это значение дает некоторое представление об ошибке для нашего расчета.

Этот пример иллюстрирует различие между прямой и обратной ошибкой. Прямая ошибка, создаваемая аппроксимацией, скорее всего, определяет нашу интуицию для анализа ошибок как разницу между аппроксимированным и фактическим решением, но, как мы обсуждали, ее не всегда возможно вычислить. Однако обратная ошибка отличается тем, что ее можно вычислить, но она не является нашей точной целью при решении данной проблемы. Мы можем корректировать наше определение и интерпретацию обратной ошибки по мере того, как мы подходим к различным проблемам, но одно подходящее, хотя и расплывчатое, определение выглядит следующим образом:

Определение 1.3 (обратная ошибка). Обратная ошибка определяется величиной, которую пришлось бы изменить в постановке задачи, чтобы реализовать заданное приближение к ее решению.

Это определение несколько бестолковое, поэтому мы проиллюстрируем его использование на нескольких примерах.

Пример 1.3 (Линейные системы). Предположим, мы хотим решить линейную систему размера n × n Ax = b. Назовем истинным решением x0 A 1b. На самом деле из-за ошибки усечения и других проблем наша система дает почти решение xest. Прямая ошибка этого приближения, очевидно, измеряется с помощью разности xest x0; на практике это значение невозможно вычислить, так как мы не знаем x0. На самом деле xest — это точное решение модифицированной системы Ax = лучший из лучших Axest; таким образом, мы могли бы измерить обратную ошибку с точки зрения разности b-best. В отличие от прямой ошибки, эту ошибку легко вычислить, не обращая A, и легко видеть, что xest является решением проблемы именно тогда, когда обратная (или прямая) ошибка равна нулю.

Пример 1.4 (Решение уравнений, CITE). Предположим, мы пишем функцию для нахождения квадратных корней из положительных чисел, которая выдает 2 1,4. Прямая ошибка |1,4 1,41421 · · · | 0,0142. Обратите внимание, что 1,42 = 1,96, поэтому обратная ошибка равна |1,96 2| = 0,04.

Два приведенных выше примера демонстрируют более широкую закономерность, согласно которой обратная ошибка может быть намного проще вычислена, чем прямая. Например, для оценки прямой ошибки в примере 1.3 требовалось инвертировать матрицу A, тогда как для оценки обратной ошибки требовалось только умножение на A. Аналогично, в примере 1.4 переход от прямой ошибки к обратной ошибке заменил вычисление квадратного корня умножением.

1.2.2 Кондиционирование, стабильность и точность

Почти в любой численной задаче нулевая обратная ошибка подразумевает нулевую прямую ошибку, и наоборот.
Таким образом, часть программного обеспечения, предназначенного для решения такой проблемы, безусловно, может прекратить работу, если обнаружит, что решение-кандидат имеет нулевую обратную ошибку. Но что, если обратная ошибка не равна нулю, но мала?
Обязательно ли это означает небольшую ошибку вперед? Такие вопросы мотивируют анализ большинства численных методов, целью которых является минимизация прямой ошибки, но на практике они могут измерять только обратную ошибку.

Мы хотим проанализировать изменения обратной ошибки по сравнению с прямой ошибкой, чтобы наши алгоритмы могли с уверенностью сказать, используя только обратную ошибку, что они дали приемлемые решения.
Это соотношение может быть разным для каждой проблемы, которую мы хотим решить, поэтому в итоге мы делаем следующую грубую классификацию:

- Проблема нечувствительна или хорошо обусловлена, когда небольшое количество обратных ошибок подразумевает небольшое количество прямых ошибок. Другими словами, небольшое изменение постановки хорошо обусловленной задачи дает лишь небольшое изменение истинного решения.

- Проблема чувствительна или плохо обусловлена, если это не так.

Пример 1.5 (ax = б). Предположим в качестве игрушечного примера, что мы хотим найти решение x0 b/a линейного уравнения ax = b для a, x, b R. Прямая ошибка потенциального решения x определяется как x x0 , а обратная ошибка равна определяется как b ax = a(x x0). Итак, когда |a| 1, задача хорошо обусловлена, так как малые значения обратной ошибки a(x x0) влекут за собой еще меньшие значения x x0; и наоборот, когда |a| 1 проблема плохо обусловлена, так как даже если a(x x0) мало, прямая ошибка x x0 1/a · a(x x0) может быть большой, учитывая коэффициент 1/a .

Мы определяем число условий как меру чувствительности проблемы:

Определение 1.4 (Номер условия). Число обусловленности задачи — это отношение того, насколько изменится ее решение, к тому, насколько изменится ее формулировка при малых возмущениях. В качестве альтернативы, это отношение прямой ошибки к обратной для небольших изменений в постановке задачи.

Пример 1.6 (ax = b, часть вторая). Продолжая пример 1.5, мы можем точно вычислить номер условия:

$$c = \frac{\text{ошибка вперед}}{\text{ошибка назад}} \overset{"="}{=} \frac{x - x0}{a\,(x - x0)} \quad \frac{1}{a}$$

В общем, вычисление чисел условий почти так же сложно, как вычисление прямой ошибки, и поэтому их точное вычисление, вероятно, невозможно. Тем не менее, во многих случаях можно найти границы или приближения для чисел условий, чтобы помочь оценить, насколько можно доверять решению.

Пример 1.7 (Поиск корня). Предположим, что нам дана гладкая функция f : R R и мы хотим найти значения x c f(x) = 0. Обратите внимание, что f(x +) f(x) + f (x). Таким образом, аппроксимация числа условий для нахождения x может быть

$$\frac{\text{изменение прямой ошибки}}{\text{изменение обратной ошибки}} \overset{"="}{=} \frac{(x +) - x}{f(x +) - f(x)}$$

$$\overset{"="}{=} \frac{1}{\frac{f(x)}{\phi(x)}}$$

Обратите внимание, что это приближение совпадает с приближением в примере 1.6. Конечно, если мы не знаем x, мы не можем вычислить f (x), но если мы можем посмотреть на форму f и оценить | ж | вблизи x у нас есть представление о наихудшей ситуации.

Прямая и обратная ошибка являются мерой точности решения. Ради научной воспроизводимости мы также хотим вывести стабильные алгоритмы, которые дают самосогласованные решения для класса задач. Например, алгоритм, который генерирует очень точные решения только в одной пятой части времени, может не стоить реализовывать, даже если мы можем вернуться, используя описанные выше методы, чтобы проверить, является ли решение-кандидат хорошим.

1.3 Практические аспекты

Бесконечность и плотность вещественных чисел R могут вызывать опасные ошибки при реализации численных алгоритмов. Хотя теория анализа ошибок, представленная в § 1.2, в конечном счете поможет нам гарантировать качество численных методов, представленных в следующих главах, прежде чем мы продолжим, стоит отметить ряд распространенных ошибок и «подводных камней», которыми сопровождаются реализации численных методов.

В начале §1.1 мы преднамеренно представили самый большой нарушитель, который мы повторяем более крупным шрифтом для

заслуженного внимания: Редко, если вообще когда-либо, оператор == и его эквиваленты должны использоваться для дробных значений.

Поиск подходящей замены == и соответствующих условий завершения численного метода зависит от рассматриваемого метода. Пример 1.3 показывает, что метод решения Ax = b может закончиться, когда невязка b Ax равна нулю; поскольку мы не хотим явно проверять, является ли A*x==b, на практике реализации будут проверять norm(A*xb)<epsilon. Обратите внимание, что этот пример демонстрирует два метода:

- Использование обратной ошибкиb — Ax, а не прямой ошибки, чтобы определить, когда прекратить, и
- Проверка того, меньше ли обратная ошибка эпсилон, чтобы избежать запретного предиката ==0.

Параметр эпсилон зависит от того, насколько точным должно быть желаемое решение, а также от разрешения используемой системы счисления.

Программист, использующий эти типы данных и операции, должен проявлять бдительность, когда речь идет об обнаружении и предотвращении некорректных числовых операций. Например, рассмотрим следующий фрагмент кода для вычисления нормы x2 для вектора x Rn , представленного в виде одномерного массива x[]:

```
двойная нормаКвадрат = 0; for (int
i = 0; i < n; i ++) normSquared += x [i]* x [i];

вернуть sqrt (normSquared);
```

Легко видеть, что теоретически mini |xi | x2/ n maxi |xi |, т. е. норма x порядка значений элементов, содержащихся в x. Однако при вычислении x2 скрыто выражение x[i]*x[i]. Если существует такое i, что x[i] порядка DOUBLE MAX, произведение x[i]*x[i] переполнится, даже если x2 все еще находится в диапазоне удвоений. Такое переполнение легко предотвратить, разделив x на его максимальное значение, вычислив норму и умножив обратно:

```
двойной maxElement = эпсилон ; // не нужно делить на ноль! для ( int я = 0; я < п ; я ++)

                maxElement = max ( maxElement for ( int i   ,   потрясающие ( x [ я ]));
= 0; i < n ; i ++) {
                двойной масштаб = x[i]/maxElement; normSquared
                += в масштабе * в масштабе ;
}
вернуть sqrt (normSquared) * maxElement;
```

Коэффициент масштабирования устраняет проблему переполнения, гарантируя, что суммируемые элементы не превышают 1.

Этот небольшой пример показывает одно из многих обстоятельств, при котором один символ кода может привести к неочевидной числовой проблеме. Хотя нашей интуиции из непрерывной математики достаточно для создания многих численных методов, мы всегда должны перепроверять, что используемые нами операции верны с дискретной точки зрения.

1.3.1 Пример большего масштаба: суммирование

Теперь мы приведем пример числовой проблемы, вызванной арифметикой с конечной точностью, которую можно решить с помощью менее чем очевидного алгоритмического трюка.

Предположим, что мы хотим суммировать список значений с плавающей запятой, что легко требуется для систем бухгалтерского учета, машинного обучения, графики и почти любой другой области. Фрагмент кода для выполнения этой задачи, которая, без сомнения, используется в бесчисленных приложениях, выглядит следующим образом:

```
двойная сумма = 0;
для (int i = 0; i < n; i ++) sum += x [i];
```

Прежде чем мы продолжим, стоит отметить, что для подавляющего большинства приложений это совершенно стабильный и, безусловно, математически обоснованный метод.

Но что может пойти не так? Рассмотрим случай, когда n велико, а большинство значений x[i] малы и положительны. В этом случае, когда i достаточно велико, сумма переменных будет велика относительно x[i]. В конце концов сумма может быть настолько велика, что x[i] влияет только на младшие биты суммы, а в крайнем случае сумма может быть достаточно большой, чтобы добавление x[i] не имело никакого эффекта. Хотя одна такая ошибка может не иметь большого значения, накопленный эффект от повторного совершения этой ошибки может превысить сумму, которой мы вообще можем доверять.

Чтобы понять этот эффект математически, предположим, что вычисление суммы a + b может быть ошибочным на величину, равную ε > 0. Тогда явно описанный выше метод может вызвать ошибку порядка nε, которая линейно растет с ростом n. На самом деле, если большинство элементов x[i] имеют порядок ε, то этой сумме вообще нельзя доверять! Это неутешительный результат: ошибка может быть такой же большой, как и сама сумма.

К счастью, есть много способов сделать лучше. Например, добавление сначала наименьших значений может помочь объяснить их кумулятивный эффект. Парные методы, рекурсивно добавляющие пары значений из x[] и создающие сумму, также более стабильны, но их может быть сложно реализовать так же эффективно, как описанный выше цикл for. К счастью, алгоритм Кахана (CITE) предоставляет легко реализуемый метод «компенсированного суммирования», который работает почти так же быстро.

Полезное наблюдение здесь заключается в том, что мы действительно можем отслеживать приближение ошибки в сумме во время данной итерации. В частности, рассмотрим выражение

$$((a + b) - a) - b.$$

Очевидно, это выражение алгебраически равно нулю. Однако численно это может быть не так.
В частности, сумма (a + b) может округлять результат, чтобы он оставался в пределах области значений с плавающей запятой. Вычитание a и b по одному за раз дает приблизительное значение ошибки, вызванной этой операцией; обратите внимание, что операции вычитания, вероятно, лучше обусловлены, поскольку переход от больших чисел к маленьким добавляет цифры точности из-за отмены.

Таким образом, техника Кахана действует следующим образом:

```
двойная сумма = 0;
двойная компенсация = 0; // приближение ошибки

for ( int i = 0; i < n ; i ++) { // пытаемся
                прибавить и к x [ i ] и к недостающей части double nextTerm = x [ i ] +
                компенсация ;

                // вычисляем результат суммирования этой итерации double nextSum =
                sum + nextTerm ;

                // вычислить компенсацию как разницу между термином, который вы хотели // добавить, и фактическим
                результатом компенсация = nextTerm -
                ( nextSum - sum );

                сумма = следующая сумма;
}
```

Вместо того, чтобы просто поддерживать сумму, теперь мы отслеживаем сумму, а также аппроксимацию компенсации разницы между суммой и желаемым значением. Во время каждой итерации мы пытаемся добавить обратно

эту компенсацию в дополнение к текущему элементу x[], а затем мы пересчитываем компенсацию для учета последней ошибки.

Анализ алгоритма Каана требует более тщательного учета, чем анализ более простого метода нарастания. В конце этой главы вы познакомитесь с одним выводом выражения ошибки; окончательный математический результат будет заключаться в том, что ошибка улучшится с nε до O(ε + nε значительное улучшение, когда^2), а $0 < \varepsilon < 1$.

Реализовать суммирование Кэхана несложно, но оно более чем удваивает количество операций результирующей программы. Таким образом, существует неявный компромисс между скоростью и точностью, на который инженеры-программисты должны пойти при выборе наиболее подходящего метода.

В более широком смысле алгоритм Кахана является одним из нескольких методов, позволяющих избежать накопления численной ошибки в ходе вычислений, состоящих более чем из одной операции. Другие примеры включают алгоритм Брезенхема для растеризации линий (CITE), который использует только целочисленную арифметику для рисования линий, даже если они пересекают строки и столбцы пикселей в нецелочисленных местах, и быстрое преобразование Фурье (CITE), которое эффективно использует двоичное разбиение. трюк с суммированием, описанный выше.

1.4 Проблемы

Проблема 1.1. Вот проблема.

Часть II

Линейная алгебра

Глава 2

Линейные системы и LU Разложение

В главе 0 мы обсуждали различные ситуации, в которых линейные системы уравнений Ax = b появляются в математической теории и на практике. В этой главе мы решаем основную проблему и исследуем численные методы решения таких систем.

2.1 Разрешимость линейных систем

Как было введено в §0.3.3, системы линейных уравнений типа

$$\begin{aligned} 3x + 2y &= 6 \\ 4x + y &= 7 \end{aligned}$$

можно записать в матричной форме, как в

$$\begin{bmatrix} 3 & 2 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 6 \\ 7 \end{bmatrix}.$$

В более общем случае мы можем записать системы вида Ax =b для $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, и $b \in \mathbb{R}^m$.

Разрешимость системы должна попадать в один из трех случаев:

1. Система может не допускать никаких решений, например:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

   Эта система требует, чтобы $x = 1$ и $x = 1$ одновременно, очевидно, два несовместимых условия.

2. Система может допускать единственное решение; например, система в начале этого раздела решается как (x, y) = (8/11, 45/11).

3. Система может иметь бесконечно много решений, например, 0x = 0. Заметим, что если система Ax =b допускает два решения x0 и x1, то она автоматически имеет бесконечно много решений вида cx0 + (1 c)x1 при c R, так как

$$A(cx_0 + (1-c)x_1) = cAx_0 + (1-c)Ax_1 = cb + (1-c)b = b.$$

Эту линейную систему можно было бы назвать недоопределенной.

В общем случае разрешимость системы зависит как от A, так и от b. Например, если мы изменим приведенную выше неразрешимую систему, чтобы она была

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

тогда система переходит от отсутствия решений к бесконечному множеству решений в форме (1, y). Фактически, каждая матрица A допускает правую часть b такую, что Ax = b разрешима, поскольку Ax = 0 всегда может быть решена x 0 независимо от A. Напомним из §0.3.1, что умножение матрицы на вектор можно рассматривать как линейное объединение столбцов A с весами из x. Таким образом, как упоминалось в §0.3.3, мы можем ожидать, что Ax =b будет разрешимым именно тогда, когда b находится в пространстве столбцов A.

В широком смысле «форма» матрицы A Rm×n существенно влияет на разрешимость Ax =b. Напомним, что столбцы A являются m-мерными векторами. Во-первых, рассмотрим случай, когда A «широкий», то есть столбцов в нем больше, чем строк (n > m). Каждый столбец является вектором в Rm, поэтому пространство столбцов может иметь размерность не более m. Поскольку n > m, n столбцов A должны быть линейно зависимы; это означает, что существует x0 = 0 такой, что Ax0 = 0. Тогда, если мы можем решить Ax = b относительно x, то A(x + αx0) = Ax + αAx0 = b + 0 = b, показывая, что на самом деле существует бесконечно много решений x для Ax = b. Другими словами, мы показали, что ни одна широкая матричная система не допускает единственного решения.

Когда A «высокий», то есть когда в нем больше строк, чем столбцов (m > n), тогда n столбцов не могут охватывать Rm. Таким образом, существует некоторый вектор b0 Rm\col A. По определению этот вектор b0 не может удовлетворять условию Ax = b0 ни для какого x. Другими словами, любая длинная матрица A допускает неразрешимые системы Ax = b0 .

Обе описанные выше ситуации далеко не благоприятны для построения численных алгоритмов. Например, если линейная система допускает множество решений, мы должны сначала определить, какое решение требуется от пользователя: в конце концов, решение x + 1031x0 может быть не таким осмысленным, как x 0,1x0. С другой стороны, в длинном случае, даже если Ax = b разрешимо для конкретного b, любое малое возмущение Ax =b + εb0 уже не разрешимо; такая ситуация может возникнуть просто потому, что процедуры округления, обсуждавшиеся в предыдущей главе, могут, во-первых, только аппроксимировать A и b.

Учитывая эти сложности, в этой главе мы сделаем несколько упрощающих предположений: • Мы будем рассматривать только квадрат A Rn×n .

• Будем считать, что A неособо, т. е. что Ax =b разрешима для любого b.

Напомним из §0.3.3, что условие невырожденности эквивалентно требованию, чтобы столбцы A охватывает Rn и влечет существование матрицы A^{-1} такое, что $A^{-1}A = AA^{-1} = I_{n\times n}$.

Вводящее в заблуждение наблюдение состоит в том, чтобы думать, что решение Ax =b эквивалентно явному A^{-1} вычислению матрицы и последующему умножению, чтобы найти x $A^{-1}b$. Хотя эта стратегия решения, безусловно, действительна, она может представлять собой значительное количество излишеств: в конце концов, нас интересуют только значения n 2 в x, а не n. Кроме того, даже когда A ведет себя хорошо, может случиться так, что написание A^{-1} дает числовые трудности, которые можно обойти.

2.2 Стратегии специальных решений

Во вводной алгебре мы часто подходим к проблеме решения линейной системы уравнений как к искусству. Стратегия состоит в том, чтобы «изолировать» переменные, итеративно записывая альтернативные формы линейной системы до тех пор, пока каждая строка не будет иметь форму x = const.

При формулировании систематических алгоритмов решения линейных систем полезно привести пример этого процесса решения. Рассмотрим следующую систему:

$$\begin{aligned} y - z &= -1 \\ 3x - y + z &= 4 \\ x + y - 2z &= -3 \end{aligned}$$

Параллельно мы можем поддерживать матричную версию этой системы. Вместо того, чтобы явно записывать Ax = b, мы можем сэкономить немного места, написав «расширенную» матрицу ниже:

$$\left[\begin{array}{ccc|c} 0 & 1 & 1 & 1 \\ 3 & 1 & 1 & 4 \\ 1 & 1 & 2 & 3 \end{array}\right]$$

Мы всегда можем писать линейные системы таким образом, если согласны с тем, что переменные остаются в левой части уравнений, а константы — в правой.

Возможно, мы хотим сначала разобраться с переменной x. Для удобства мы можем переставить строки системы так, чтобы третье уравнение появилось первым:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3x - y + z &= 4 \end{aligned} \qquad \left[\begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & 1 & 1 \\ 3 & 1 & 1 & 4 \end{array}\right]$$

Затем мы можем подставить первое уравнение в третье, чтобы исключить член 3x. Это то же самое, что масштабировать отношение x + y - 2z = -3 на -3 и добавлять результат к третьему уравнению:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ -4y + 7z &= 13 \end{aligned} \qquad \left[\begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & 1 & 1 \\ 0 & -4 & 7 & 13 \end{array}\right]$$

Точно так же, чтобы исключить y из третьего уравнения, мы можем умножить второе уравнение на 4 и добавить результат к третьему:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3z &= 9 \end{aligned} \qquad \left[\begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 3 & 9 \end{array}\right]$$

Теперь мы изолировали z! Таким образом, мы можем масштабировать третью строку на 1/3 , чтобы получить выражение для z:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ z &= 3 \end{aligned} \qquad \left[\begin{array}{ccc|c} 1 & 1 & 2 & 3 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

Теперь мы можем подставить z = 3 в два других уравнения, чтобы удалить z из всех строк, кроме последней:

$$\begin{array}{l} x + y = 3 \\ y = 2 \\ z = 3 \end{array} \qquad \left[\begin{array}{ccc|c} 1 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

Наконец, мы делаем аналогичную замену для y, чтобы завершить решение:

$$\begin{array}{l} x = 1 \\ y = 2 \\ z = 3 \end{array} \qquad \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right]$$

Этот пример может показаться несколько педантичным, но оглядываясь назад на нашу стратегию, можно сделать несколько важных наблюдений о том, как мы можем решать линейные системы:

- Мы написали последовательные системы $A_i x = b_i$, которые можно рассматривать как упрощения исходной системы. $Ax = b$.
- Мы решили систему, даже не записывая A^{-1}.
- Мы неоднократно использовали несколько простых операций: масштабирование, добавление и перестановку строк таблицы. система.
- Те же операции были применены к A и b. Если мы масштабируем k-ю строку A, мы также масштабируем k-й ряд b. Если мы добавили строки k и из A, мы добавили строки k и из b.
- Менее очевидно, что шаги решения не зависели от b. То есть все наши решения о том, как решать, были мотивированы устранением ненулевых значений в A, а не изучением значений в b; b, которые просто попались на пути.
- Мы остановились, когда свели систему к $I_{n \times n} x = b$.

Мы будем использовать все эти общие наблюдения о решении линейных систем в наших интересах.

2.3 Кодирование операций со строками

Возвращаясь к примеру в § 2.2, мы видим, что решение линейной системы на самом деле включало в себя только три операции: перестановку, масштабирование строк и добавление масштаба одной строки к другой. На самом деле, таким образом мы можем решить любую линейную систему, поэтому стоит изучить эти операции более подробно.

2.3.1 Перестановка

Нашим первым шагом в §2.2 было поменять местами две строки в системе уравнений. В более общем случае мы могли бы индексировать строки матрицы, используя числа 1, . . . , м. Тогда перестановку этих строк можно записать в виде функции σ такой, что список σ(1), . . . , σ(m) покрывает тот же набор индексов.

Если ek — k-я стандартная базисная функция, то легко видеть, что произведение e к A дает k-ю строку матрицы A. Таким образом, мы можем «сложить» или объединить эти векторы-строки по вертикали, чтобы получить матрицу, переставляющую строки в соответствии с σ :

$$P_\sigma = \begin{matrix} e_{\sigma(1)} \\ e_{\sigma(2)} \\ \vdots \\ e_{\sigma(m)} \end{matrix}$$

То есть произведение PσA — это в точности матрица A со строками, переставленными в соответствии с σ.

Пример 2.1 (матрицы перестановок). Предположим, мы хотим переставить строки матрицы в R3×3 с σ(1) = 2, σ(2) = 3 и σ(3) = 1. Согласно нашей формуле мы бы имели

$$P_\sigma = \begin{matrix} 0\,1\,0 \\ 0\,0\,1 \\ 1\,0\,0 \end{matrix}$$

Из примера 2.1 видно, что Pσ имеет единицы в позициях вида (k, σ(k)) и нули в остальных местах. Пара (k, σ(k)) представляет утверждение: «Мы хотели бы, чтобы строка k выходной матрицы была строкой σ(k) из входной матрицы». Основываясь на этом описании матрицы перестановок, легко увидеть, что обратным Pσ является транспонированное P, поскольку это просто меняет местами роли строк и столбцов — теперь мы берем строку σ(k) входных данных и помещаем ее в строка k вывода. Другими словами, $P_\sigma^{-1} = P_\sigma^T$.

2.3.2 Масштабирование строк

Предположим, мы записываем список констант a1, . . . , am и попытаться масштабировать k-ю строку некоторой матрицы A на ak . Очевидно, это достигается путем применения масштабной матрицы Sa, определяемой следующим образом:

$$C_b = \begin{matrix} a_1 & 0 & 0 & \cdots & 0 \\ 0 & a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \cdots & & a_m \end{matrix}$$

Предполагая, что все ak удовлетворяют ak = 0, легко инвертировать Sa , «уменьшив масштаб»:

$$C_a^{-1} = S_{1/a} = \begin{matrix} 1/a_1 & 0 & 0 & \cdots & 0 \\ 0 & 1/a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \cdots & & 1/a_m \end{matrix}$$

2.3.3 Устранение

Наконец, предположим, что мы хотим масштабировать строку k с помощью константы c и добавить результат к строке . Эта операция может показаться менее естественной, чем две предыдущие, но на самом деле она весьма практична.

нужно объединить уравнения из разных строк линейной системы! Мы реализуем эту операцию, используя «матрицу исключения» M, так что произведение MA применяет эту операцию к матрице A.

Напомним, что произведение $e_k A$ выбирает k-ю строку A. Затем предварительное умножение на e дает матрица к A, который равен нулю, за исключением того, что -th строка равна k-й строке A.

Пример 2.2 (Построение матрицы исключения). Брать

$$A = \begin{matrix} 1\,2\,3 \\ 4\,5\,6 \\ 7\,8\,9 \end{matrix}$$

Предположим, мы хотим изолировать третью строку матрицы A R3×3 и переместить ее во вторую строку. Как обсуждалось выше, эта операция выполняется записью:

$$e2e\,3\ A = \begin{matrix} 0\,1 \\ 0 \end{matrix} \quad 0\,0\,1 \quad \begin{matrix} 1\,2\,3\,4 \\ 5\,6 \\ 7\,8\,9 \end{matrix}$$

$$\text{"="} \quad \begin{matrix} 0\,1 \\ 0 \end{matrix} \quad 7\,8\,9$$

$$\text{"="} \quad \begin{matrix} 0\,0\,0 \\ 7\,8\,9 \\ 0\,0\,0 \end{matrix}$$

Конечно, выше мы умножали справа налево, но так же легко могли бы сгруппировать произведение, как (e2e3)A. Структуру этого продукта легко увидеть:

$$e2e\,3 \quad \text{"="} \quad \begin{matrix} 0\,1 \\ 0 \end{matrix} \quad 0\,0\,1 = \quad \begin{matrix} 0\,0\,0\,0 \\ 0\,1 \\ 0\,0\,0 \end{matrix}$$

Нам удалось изолировать строку k и переместить ее в строку . Наша первоначальная операция исключения хотела добавить с раз строку k к строке A = k (In×n + cee к)A. Что мы теперь можем выполнить как сумму A + cee

Пример 2.3 (Решение системы). Теперь мы можем закодировать каждую из наших операций из раздела 2.2, используя матрицы, которые мы построили выше:

1. Переставьте строки, чтобы переместить третье уравнение в первую строку:

$$П = \begin{matrix} 0\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \end{matrix}$$

2. Масштабируйте первую строку на -3 и прибавьте результат к третьей строке: E1 = I3×3 3e3e.1

3. Масштабируйте вторую строку на 4 и добавьте результат к третьей строке: E2 = I3×3 + 4e3e. 2

4. Масштабируйте третью строку на 1/3: S = diag(1, 1, 1/3)

5. Масштабируйте третью строку на 2 и добавьте ее к первой строке: E3 = I3×3 + 2e1e.3

6. Добавьте третью строку ко второй: E4 = I3×3 +e2e. 3

7. Масштабируйте третью строку на -1 и прибавьте результат к первой строке: E5 = I3×3 e1e 3

Таким образом, обратный A в разделе 2.2 удовлетворяет

$$A^{1} = E5E4E3SE2E1P.$$

Убедитесь, что вы понимаете, почему эти матрицы отображаются в обратном порядке!

2.4 Исключение Гаусса

Последовательность шагов, выбранная в разделе 2.2, отнюдь не уникальна: существует множество различных путей, которые могут привести к решению Ax = b. Наши шаги, однако, следовали стратегии исключения Гаусса, известного алгоритма решения линейных систем уравнений.

В более общем смысле, скажем, наша система имеет следующую «форму»:

$$A\,б = \left|\begin{array}{c} \times\times\times\times\times \\ \times\times\times\times\times \\ \times\times\times\times\times \\ \times\times\times\times\times \end{array}\right|$$

Алгоритм работает поэтапно, как описано ниже.

2.4.1 Замена вперед

Рассмотрим верхний левый элемент нашей матрицы:

$$A\,б = \left|\begin{array}{c} \times\times\times\times\times \\ \times\times\times\times\times \\ \times\times\times\times\times \\ \times\times\times\times\times \end{array}\right|$$

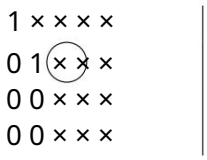
Мы назовем этот элемент нашей первой точкой опоры и будем считать, что он не равен нулю; если он равен нулю, мы можем переставить строки так, чтобы это было не так. Сначала мы применяем матрицу масштабирования, чтобы точка опоры равнялась единице:

$$\left|\begin{array}{c} 1\times\times\times\times \\ \times\times\times\times\times \\ \times\times\times\times\times \\ \times\times\times\times\times \end{array}\right|$$

Теперь мы используем строку, содержащую сводную точку, чтобы исключить все остальные значения ниже в том же столбце:

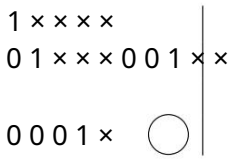
$$\left|\begin{array}{c} 1\times\times\times\times 0\times\times\times\times \\ 0\times\times\times\times 0\times\times\times\times \end{array}\right|$$

Теперь мы перемещаем нашу точку опоры на следующую строку и повторяем аналогичную серию операций:



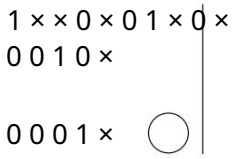
Обратите внимание, что здесь происходит приятная вещь. После того, как первая точка опоры удалена из всех остальных строк, первый столбец равен нулю под строкой 1. Это означает, что мы можем безопасно добавлять числа, кратные второй строке, к строкам под ней, не затрагивая нули в первом столбце.

Повторяем этот процесс до тех пор, пока матрица не станет верхнетреугольной:

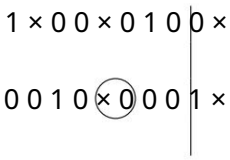


2.4.2 Обратная замена

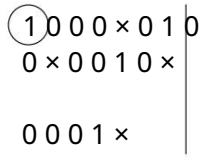
Устранение оставшихся × из системы теперь является прямым процессом. Теперь мы действуем в обратном порядке строк и удаляем назад. Например, после первой серии шагов обратной замены у нас остается следующая форма:



Точно так же вторая итерация дает:



После нашего последнего шага исключения у нас осталась желаемая форма:



Правая часть теперь является решением линейной системы Ax = b.

2.4.3 Анализ исключения Гаусса

Каждая операция со строками в методе исключения Гаусса — масштабирование, исключение и замена двух строк — очевидно, занимает O(n) времени для завершения, так как вам нужно выполнить итерацию по всем n элементам строки (или

два) из A. Как только мы выбираем точку опоры, мы должны сделать n прямых или обратных замен в строках). В общей сложности ниже или выше этой точки опоры, соответственно; это означает, что работа для одного поворота в целом составляет O (n^2 мы выбираем по одной опорной точке на строку, добавляя последний коэффициент n. Таким образом, достаточно легко видеть, что гауссовский исключение выполняется за O(n^3) время.

Одно решение, которое принимается во время исключения Гаусса и которое мы не обсуждали, — это выбор опорных точек. Напомним, что мы можем переставлять строки линейной системы по своему усмотрению перед выполнением обратной или прямой замены. Эта операция необходима, чтобы иметь возможность работать со всеми возможными матрицами A. Например, рассмотрим, что произошло бы, если бы мы не использовали поворот на следующих матрица:

$$A = \begin{matrix} (0) & 1 & 1 \\ 0 & & \end{matrix}$$

Обратите внимание, что элемент, обведенный кружком, равен нулю, поэтому мы не можем ожидать деления первой строки на любое число, чтобы заменить этот 0 на 1. Это не означает, что система неразрешима, это просто означает, что мы должны выполнить поворот, достигнутый путем замены элементов местами. первую и вторую строки, чтобы поместить в этот слот ненулевое значение.

В более общем случае предположим, что наша матрица выглядит так:

$$A = \begin{matrix} (\varepsilon) & 1 \\ 1 & 0 \end{matrix},$$

где $0 < \varepsilon$ 1. Если мы не поворачиваемся, то первая итерация исключения Гаусса дает:

$$\tilde{A} = \begin{matrix} (1) & 1/\varepsilon \\ 0 & 1/\varepsilon \end{matrix},$$

Мы преобразовали матрицу A, которая выглядит почти как матрица перестановок (на самом деле это ^{1} A , a очень простой способ решения системы!) в систему с потенциально огромными значениями 1/ε.

Этот пример показывает, что бывают случаи, когда мы можем захотеть развернуться, даже если в этом, строго говоря, нет необходимости. Поскольку мы масштабируем по обратному значению опорного значения, очевидно, что наиболее численно устойчивые варианты — это иметь большое опорное значение: маленькие опорные значения имеют большие обратные значения, масштабируя числа до больших значений в режимах, которые, вероятно, потеряют точность. Есть две хорошо известные стратегии разворота:

1. Частичная сводка просматривает текущий столбец и переставляет строки матрицы так, что наибольшее абсолютное значение появляется на диагонали.

2. Полный поворот выполняет итерацию по всей матрице и переставляет как строки, так и столбцы, чтобы получить максимально возможное значение по диагонали. Обратите внимание, что перестановка столбцов матрицы является допустимой операцией: она соответствует изменению маркировки переменных в системе или последующему умножению A на перестановку.

Пример 2.4 (Поворот). Предположим, что после первой итерации исключения Гаусса у нас осталась следующая матрица:

$$\begin{matrix} 1 & 10 & 10 \\ 0 & 0{,}1 & 9 \\ 0 & 4 & 6.2 \end{matrix}$$

Если мы реализуем частичный поворот, то мы будем смотреть только во второй столбец и поменяем местами вторую и третью строки; обратите внимание, что мы оставляем 10 в первой строке, так как алгоритм уже посещал ее:

1 10 10
0 4 6,2
0 0,1 9

Если мы реализуем полный поворот, то мы переместим 9:

1 10 10
0 9 0,1
0 6,2 4

Очевидно, что полная сводка дает наилучшие числовые значения, но цена — более дорогой поиск больших элементов в матрице.

2.5 Факторизация LU

Много раз мы хотим решить последовательность задач Ax1 =b1, Ax2 =b2, ... Как мы уже обсуждали, шаги исключения Гаусса для решения Ax =bk зависят в основном от структуры A, а не от значений в конкретном bk . Поскольку A здесь сохраняется постоянным, мы можем захотеть «запомнить» шаги, которые мы предприняли для решения системы, чтобы каждый раз, когда мы сталкиваемся с новым b, нам не приходилось начинать с нуля.

Укрепляя это подозрение, что мы можем переместить некоторые из $O(n^3)$ для исключения Гаусса во время предварительного вычисления вспомните верхнюю треугольную систему, полученную после этапа прямой подстановки:

1 × × × ×
0 1 × × × 0 0 1 × | ×
0 0 0 1 × ◯ |

На самом деле, решение этой системы обратной подстановкой ^{2}) время! Почему? Обратная замена требует только O(n, в этом случае это намного проще благодаря структуре нулей в системе. Например, в первой серии обратных подстановок мы получаем следующую матрицу:

1 × × 0 × 0 × 0 × 0 | 1
0 × 1 |
0 |
0 0 0 1 × |

Поскольку мы знаем, что значения (обведены кружком) слева от опорной точки равны нулю по построению, нам не нужно копировать их явно. Таким образом, этот шаг занял всего O(n) времени, а не $O(n^2)$, затрачиваемого на прямую замену.

Теперь наша следующая точка поворота делает аналогичную замену:

1 × 0 0 ×
0 1 0 0 ×
0 0 1 0 × 0 0 1 × |
0 |

Опять же, нули по обе стороны от 1 не нужно копировать явно.

Таким образом, мы нашли:

Наблюдение. В то время как исключение Гаусса принимает O (n^3) времени решение треугольных систем занимает O(n^2) время.

2.5.1 Построение факторизации

Напомним из §2.3, что все операции исключения Гаусса можно рассматривать как предварительное умножение Ax = b на разные матрицы M для получения более простой системы (MA)x = Mb. Как мы показали в примере 2.3, с этой точки зрения каждый шаг исключения Гаусса представляет собой систему (Mk · · · M2M1A)x = Mk · · · M2M1 b . Конечно, явное хранение этих матриц Mk как объектов размера n × n является излишним, но учет этой интерпретации с теоретической точки зрения упрощает многие наши вычисления.

После фазы прямой замены исключения Гаусса у нас остается верхняя треугольная матрица, которую мы можем назвать U Rn×n . С точки зрения умножения матриц мы можем
писать:

$$M_k \cdots M_1 A = U,$$

или, что то же самое,

$$\begin{aligned} A &= (M_k \cdots M_1)^{-1} U \\ &= (M_1^{-1} M_2^{-1} \cdots M_k^{-1}) U \\ &= LU, \text{ если сделать определение } L = M_1^{-1} M_2^{-1} \cdots M_k^{-1}. \end{aligned}$$

Мы еще ничего не знаем о структуре L, но мы знаем, что системы вида Uy = d легче решать, поскольку U является верхнетреугольным. Если L одинаково хорошо, мы могли бы решить Ax =b в два шага, записав (LU)x =b или $x = U^{-1}L^{-1}b$: 1. Решить Ly =b относительно

y, получив $y = L^{-1}b$.

2. Теперь, когда у нас есть y, решите Ux = y, что даст $x = U^{-1}y = U^{-1}(L^{-1}b) = (LU)^{-1}b = A^{-1}b$.
Мы уже знаем, что этот шаг занимает всего O(n^2) время.

Наша оставшаяся задача — убедиться, что L имеет хорошую структуру, которая облегчит решение Ly = b, чем решение Ax = b. К счастью — и это неудивительно — мы обнаружим, что L является нижнетреугольным и, следовательно, может быть решена с использованием O(n^2) прямая замена.

Чтобы убедиться в этом, предположим, что мы не реализуем поворот. Тогда каждая наша матрица Mk либо является масштабирующей матрицей, либо имеет структуру

$$M_k = B_{n\times n} + c e_i e_j^T,$$

где > k, так как мы выполнили только прямую замену. Помните, что эта матрица служит определенной цели: масштабируйте строку k по c и добавляйте результат к строке . Эту операцию, очевидно, легко отменить: масштабируйте строку k на c и вычтите результат из строки . Мы можем проверить это формально:

$$\begin{aligned} (I_{n\times n} + \text{в.п.}_k)(I_{n\times n} - \text{cee}_k) &= I_{n\times n} + (\text{cee}_k + \text{си к}) - c^2 \text{ээ э кк} \\ &= B \times n - c^2 e (e_k \, д) \, д_k \\ &= I_{n\times n}, \text{ так как } e = e_k \cdot e_k, \text{ и к} = \end{aligned}$$

Таким образом, матрица L является произведением масштабирующих матриц и матриц вида M 1 = In×n к являются + в.п. нижняя треугольная при > k. Масштабирующие матрицы диагональные, а матрица M нижняя треугольная. Вы покажете в упражнении 2.1, что произведение нижних треугольных матриц является нижним треугольным, показав, в свою очередь, что L является нижнетреугольным по мере необходимости.

Мы показали, что если возможно выполнить гауссово исключение A без использования поворота, вы можете разложить A = LU на произведение нижне- и верхнетреугольных матриц. Прямая и обратная подстановка каждая занимает O(n) времени, поэтому , если эту факторизацию можно вычислить заранее, линейное решение может 2 3) Исключение Гаусса. Ты покажешь в быть выполнено быстрее, чем полное O(n) необходимы.

2.5.2 Реализация ЛЕ

Простая реализация исключения Гаусса для решения Ax = b достаточно проста, чтобы ее можно было сформулировать. В частности, как мы обсуждали ранее, мы можем сформировать расширенную матрицу (A | b) и применять операции со строками по одной к этому n × (n + 1) блоку, пока он не будет выглядеть как (In×nA 1 |b). Этот процесс, однако, является деструктивным, то есть, в конце концов, нас интересует только последний столбец расширенной матрицы, и мы не сохранили никаких свидетельств нашего пути решения. Такое поведение явно неприемлемо для факторизации LU.

Давайте посмотрим, что происходит, когда мы умножаем две матрицы исключения:

$$(I_{n\times n} - c_{ij}e_k e_k^T)(I_{n\times n} + c_{ij}e_k e_k^T) = I_{n\times n} - c_{ij}e_k e_k^T + c_{ij}e_k e_k^T$$

Как и в нашей конструкции обратной матрицы исключения, произведение двух членов ei равно нулю, поскольку стандартный базис ортогонален. Эта формула показывает, что после масштабирования опорной точки до 1 произведение матриц исключения, используемых для прямой замены этой опорной точки, имеет вид:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \ddots & 0 \\ 0 & \times & 1 & 0 \\ 0 & \times & 0 & 1 \end{bmatrix},$$

где значения × — это значения, используемые для исключения остальной части столбца. Перемножение матриц этой формы вместе показывает, что элементы под диагональю L просто происходят из коэффициентов, используемых для выполнения подстановки.

Мы можем принять окончательное решение оставить элементы по диагонали L в факторизации LU равными 1. Это решение является законным, поскольку мы всегда можем пост-умножить L на масштабирующую матрицу S, переводя эти элементы в 1. и запишем LU = (LS)(S 1U) , не затрагивая треугольный шаблон L или U. Приняв это решение, мы можем сжать наше хранилище как L, так и U в единую матрицу n × n, верхний треугольник которой равен U и который равен L ниже диагонали; все отсутствующие диагональные элементы L равны 1.

Теперь мы готовы написать псевдокод для простейшей стратегии факторизации LU, в которой мы не переставляем строки или столбцы для получения опорных точек:

```
// Принимает на вход матрицу размера n на n A [i, j]
// Редактируем A на месте, чтобы получить описанную выше компактную факторизацию LU

для поворота от 1 до n {
        pivotValue = A [поворот, поворот]; // Плохое предположение, что это не ноль!
```

```
        for EliminationRow from ( pivot +1) to n { // Удаляем значения под сводом
                // Насколько масштабировать сводную строку, чтобы исключить значение из текущего
                // ряд ; заметьте, что мы делаем это, чтобы не масштабировать сводную строку до 1,
                // опорное значение
                масштаб = A [строка исключения, сводная точка]/поворотная величина;

                // Так как мы / вычитаем / масштабируем умноженное на опорную строку из текущей строки
                // во время исключения Гаусса // , мы /добавляем/это в обратной операции
                сохраняется в L
                A [eliminationRow, Pivot] = шкала;

                // Теперь, как и при исключении Гаусса, выполняем операцию со строками над остальными
                // строки: это станет U
                для исключенияCol от ( pivot +1) до n
                        A [ устранение Строка , EliminationCol] -= A [Pivot, EliminationCol]* масштаб;
        }
}
```

2.6 Проблемы

Проблема 2.1. Произведение нижних треугольных вещей есть нижнее треугольное; произведение сводных матриц выглядит правильно

Задача 2.2. Реализовать LU с поворотом

Задача 2.3. Неквадратный LU

Глава 3

Проектирование и анализ линейных Системы

Теперь, когда у нас есть некоторые методы решения линейных систем уравнений, мы можем использовать их для решения множества задач. В этой главе мы рассмотрим несколько таких приложений и сопутствующих аналитических методов, чтобы охарактеризовать типы решений, которые мы можем ожидать.

3.1 Решение квадратных систем

В начале предыдущей главы мы сделали несколько предположений о типах линейных систем, которые собирались решать. Хотя это ограничение было нетривиальным, на самом деле многие, если не большинство приложений линейных решателей можно представить в терминах квадратных обратимых линейных систем. Ниже мы рассмотрим несколько контрастирующих приложений.

3.1.1 Регрессия

Мы начнем с простого приложения, используемого в анализе данных, известного как регрессия. Предположим, что мы проводим научный эксперимент и хотим понять структуру наших экспериментальных результатов. Одним из способов моделирования такой связи может быть запись независимых переменных эксперимента в некотором векторе x Rn и рассмотрение зависимой переменной как функции f(x) : Rn R. Наша цель — предсказать результат f без проведения полного эксперимента.

Пример 3.1 (Биологический эксперимент). Предположим, мы хотим измерить влияние удобрений, солнечного света и воды на рост растений. Мы могли бы провести ряд экспериментов, применяя различное количество удобрений (в см3), солнечного света (в ваттах) и воды (в мл) и измеряя высоту растения через несколько дней. Мы могли бы смоделировать наши наблюдения как функцию f : R3 R , принимая три параметра, которые мы хотим проверить, и выводя высоту растения.

В параметрической регрессии мы делаем упрощающее предположение о структуре f. Например, предположим, что f линейна:

$$f(x) = a_1x_1 + a_2x_2 + \cdots + a_nx_n.$$

Тогда наша цель становится более конкретной: оценить коэффициенты ak .

Предположим, мы проводим серию экспериментов, которые показывают $f(x^{(k)})$. Подключившись к нашему форму $x \to y$ для f, мы получаем ряд утверждений:

$$\begin{aligned} y^{(1)} &= f(x^{(1)}) = a_1 x_1^{(1)} + a_2 x_2^{(1)} + \cdots + a_n x_n^{(1)} \\ y^{(2)} &= f(x^{(2)}) = a_1 x_1^{(2)} + a_2 x_2^{(2)} + \cdots + a_n x_n^{(2)} \\ &\vdots \end{aligned}$$

Обратите внимание, что вопреки нашему обозначению $Ax = b$, неизвестными здесь являются переменные a_k, а не переменные x. Если мы сделаем n наблюдений, мы можем написать:

$$\begin{bmatrix} x_1^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & & \vdots \\ x_1^{(n)} & \cdots & x_n^{(n)} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Другими словами, если мы проведем n испытаний нашего эксперимента и запишем их в столбцы матрицы $X \in \mathbb{R}^{n \times n}$, а зависимые переменные запишем в вектор $y \in \mathbb{R}^n$, то коэффициенты a можно восстановить, решив $Xa = y$.

На самом деле, мы можем обобщить наш подход на другие, более интересные нелинейные формы функции f. Здесь важно то, что f является линейной комбинацией функций. В частности, предположим, что $f(x)$ принимает следующий вид:

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_m f_m(x),$$

где $f_k : \mathbb{R}^n \to \mathbb{R}$ и мы хотим оценить параметры a_k. Тогда путем параллельного вывода (k) (k) по m наблюдениям мы можем найти параметры, решив: y формы x

$$\begin{bmatrix} f_1(x^{(1)}) & f_2(x^{(1)}) & \cdots & f_m(x^{(1)}) \\ f_1(x^{(2)}) & f_2(x^{(2)}) & \cdots & f_m(x^{(2)}) \\ \vdots & \vdots & & \vdots \\ f_1(x^{(m)}) & f_2(x^{(m)}) & \cdots & f_m(x^{(m)}) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ \text{являюсь} \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

То есть, даже если f нелинейны, мы можем узнать веса a_k, используя чисто линейные методы.

Пример 3.2 (Линейная регрессия). Система $Xa = y$ может быть восстановлена из общей формулировки, если взять $f_k(x) = x_k$.

Пример 3.3 (полиномиальная регрессия). Предположим, что мы наблюдаем функцию одной переменной $f(x)$ и хотим записать ее в виде полинома n-й степени

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + \text{тревога}^n.$$

Даны n пар $x^{(k)}$ $y^{(k)}$, мы можем решить для параметрова через систему

$$\begin{bmatrix} 1 & x_{(1)}^{(1)} & (x^{(1)})^2 & \cdots & (x^{(1)})^n \\ 1 & x_{(2)}^{(2)} & (x^{(2)})^2 & \cdots & (x^{(2)})^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{(n)}^{(n)} & (x^{(n)})^2 & \cdots & (x^{(n)})^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Другими словами, мы берем $f_k(x) = x^k$ в нашей общей форме выше. Между прочим, матрица в левой части этого соотношения известна как матрица Вандермонда, которая обладает многими особыми свойствами, характерными для ее структуры.

Пример 3.4 (Колебание). Предположим, мы хотим найти a и φ для функции $f(x) = a\cos(x + \varphi)$. Напомним из тригонометрии, что мы можем написать $\cos(x + \varphi) = \cos x \cos \varphi - \sin x \sin \varphi$. Таким образом, учитывая две точки выборки, мы можем использовать описанную выше технику, чтобы найти $f(x) = a_1 \cos x + a_2 \sin x$, и применяя это тождество, мы можем написать

$$a = \sqrt{a_1^2 + a_2^2}$$

$$\varphi = -\operatorname{arctg}\frac{a_2}{a_1}$$

Эта конструкция может быть расширена до нахождения $f(x) = \sum_k a_k \cos(x + \varphi_k)$, что дает один из способов мотивировать дискретное преобразование Фурье f.

3.1.2 Метод наименьших квадратов

Методы, описанные в §3.1.1, предоставляют ценные методы для нахождения непрерывного f, точно соответствующего набору пар данных $x_k - y_k$. У этого подхода есть два связанных недостатка:

- Возможна ошибка измерения значений x_k и y_k. В этом случае приближенное соотношение $f(x_k) - y_k$ может быть приемлемым или даже предпочтительнее точного $f(x_k) = y_k$.
- Заметьте, что если бы всего было m функций f_k, то нам понадобилось ровно m наблюдений $x_k - y_k$. Дополнительные наблюдения пришлось бы отбросить, или нам пришлось бы изменить форму f.

Обе эти проблемы связаны с более крупной проблемой переобучения: подбор функции с n степенями свободы к n точкам данных не оставляет места для ошибки измерения.

В более общем случае предположим, что мы хотим решить линейную систему $Ax = b$ относительно x. Если мы обозначим строку k матрицы A как r_k^T, когда наша система выглядит так

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} r_1^T \\ r_2^T \\ \vdots \\ r_m^T \end{bmatrix} x = \begin{bmatrix} r_1 \cdot x \\ r_2 \cdot x \\ \vdots \\ r_m \cdot x \end{bmatrix}$$

по определению умножения матриц.

Таким образом, каждой строке системы соответствует наблюдение вида $r_k \cdot x = b_k$. То есть еще один способ интерпретировать линейную систему $Ax = b$ — это n утверждений вида «Скалярное произведение x на r_k равно b_k».

С этой точки зрения, длинная система $Ax = b$ с $A \in \mathbb{R}^{m\times n}$ и $m > n$ просто кодирует более n этих наблюдений скалярного произведения. Однако когда мы делаем более n наблюдений, они могут оказаться несовместимыми; как объяснялось в §2.1, высокие системы, скорее всего, не допускают решения. В нашей «экспериментальной» установке, описанной выше, такая ситуация может соответствовать ошибкам в измерении пар $x_k - y_k$.

Когда мы не можем точно решить $Ax = b$, мы можем немного ослабить задачу, чтобы аппроксимировать $Ax \approx b$. В частности, мы можем потребовать, чтобы невязка $b - Ax$ была как можно меньше, минимизируя

норма b Ax. Заметим, что если существует точное решение линейной системы, то эта норма минимизируется в нуле, так как в этом случае мы имеем b Ax = b b = 0. Минимизация b Ax аналогична минимизации b Ax ^{2}, который мы расширили в примере 0.16 до:

$$\text{б - топор}^{\,2} = \text{x A Ax - 2б Ax + б}^{\,2}.1$$

Градиент этого выражения по x должен быть равен нулю в его минимуме, что дает следующую систему:

$$0 = 2Ax - 2Ab$$

Или, что то же самое: Ax = Ab.

Это знаменитое соотношение достойно теоремы:

Теорема 3.1 (нормальные уравнения). Минимумы невязки b Ax для A Rm×n (без ограничений на m или n) удовлетворяют условию AAx = A b.

Если хотя бы n строк матрицы A линейно независимы, то матрица AA Rn×n обратима. В этом случае минимальная невязка возникает (единственным образом) при (AA) 1A b или, что то же самое, решение задачи наименьших квадратов так же просто, как решение квадратичной линейной системы A Ax = A b из теоремы 3.1.
Таким образом, мы расширили наш набор стратегий решения до A Rm×n с m n, применяя только методы для квадратных матриц.

С недоопределенным случаем m < n справиться значительно труднее. В частности, мы теряем возможность единственного решения задачи Ax = b. В этом случае мы должны сделать дополнительное предположение относительно x, чтобы получить единственное решение, например, что оно имеет малую норму или что оно содержит много нулей. Каждое такое регуляризирующее допущение приводит к разным стратегиям решения; мы рассмотрим некоторые из них в упражнениях, сопровождающих эту главу.

3.1.3 Дополнительные примеры

Важным навыком является умение идентифицировать линейные системы «в дикой природе». Здесь мы быстро перечислим еще несколько примеров.

Выравнивание

Предположим, мы делаем две фотографии одной и той же сцены с разных позиций. Одна общая задача в компьютерном зрении и графике — сшить их вместе. Для этого пользователь (или автоматическая система) может отметить некоторое количество точек xk ,yk R2 так, что xk на первом изображении соответствует yk на втором изображении. Конечно, при сопоставлении этих точек были допущены ошибки, поэтому мы хотим найти стабильное преобразование между двумя изображениями путем передискретизации количества необходимых пар (x, y).

Предполагая, что у нашей камеры стандартный объектив, проекции камеры линейны, поэтому разумно, как предположение состоит в том, что существуют некоторый A R2×2 и вектор сдвига b R2 такие, что

$$yk \quad Axk + b.$$

[1] На этом этапе может быть полезно вернуться к предварительным этапам главы 0 для повторения.

Наши неизвестные переменные здесь — это A и b, а не xk и yk .

В этом случае мы можем найти преобразование, решив:

$$\min_{A,\,b} \sum_{k=1}^{n} \left\| (Axk + b) \quad yk \right\|^2.$$

Это выражение снова является суммой квадратов линейных выражений для наших неизвестных A и b, и путем вывода, аналогичного нашему обсуждению задачи наименьших квадратов, оно может быть решено линейно.

Деконволюция

Часто мы случайно делаем фотографии, которые немного не в фокусе. Хотя фотография, которая полностью размыта, может быть потерянной причиной, если есть локальное или мелкомасштабное размытие, мы можем восстановить более четкое изображение с помощью вычислительных методов. Одной из простых стратегий является деконволюция, описанная ниже.

Мы можем думать о фотографии как о точке в Rp. , где p — количество пикселей; конечно, если Фотография находится в цвете, нам могут понадобиться три значения (RGB) на пиксель , что приводит к . аналогичной технике в R3p . соседи на картинке. При обработке изображений эти линейные операции часто обладают другими особыми свойствами, такими как инвариантность к сдвигу, но для наших целей мы можем думать о размытии как о некотором линейном операторе x G x.

Предположим, мы взяли размытую фотографию x0 Rp .Затем мы могли бы попытаться восстановить резкое изображение x Rp путем решения задачи наименьших квадратов

$$\min_{x \in Rp} \left\| x0 - G * x \right\|^2.$$

То есть мы просим, чтобы, когда вы размываете x с помощью G, вы получали наблюдаемое фото x0. Конечно, многие четкие изображения могут давать такой же размытый результат при G, поэтому мы часто добавляем дополнительные условия к приведенной выше минимизации, прося, чтобы x0 не менялось слишком сильно.

3.2 Специальные свойства линейных систем

Наше обсуждение исключения Гаусса и факторизации LU привело к совершенно общему методу решения линейных систем уравнений. Хотя эта стратегия всегда работает, иногда мы можем получить преимущество в скорости или количественных показателях, исследуя конкретную систему, которую решаем. Здесь мы обсудим несколько распространенных примеров, когда дополнительные сведения о линейной системе могут упростить стратегии решения.

3.2.1 Положительно определенные матрицы и факторизация Холецкого

Как показано в теореме 3.1, решение задачи наименьших квадратов Ax b дает решение x, удовлетворяющее квадратичной линейной системе (A A)x = A b. Независимо от A, матрица AA обладает несколькими особыми свойствами, которые делают эту систему особенной.

Во-первых, легко видеть, что AA симметричен, так как

$$(A\,A) = A\,(A\,) = A\,A.$$

Здесь мы просто использовали тождества (AB) = BA и (A) = A. Мы можем выразить эту симметрию по индексам, записав (AA)ij = (AA)ji для всех индексов i, j. Это свойство означает, что достаточно хранить только значения AA на диагонали или над ней, так как остальные элементы могут быть получены по симметрии.

Кроме того, AA является положительно полуопределенной матрицей, как определено ниже:

Определение 3.1 (Положительное (полу)определенное). Матрица B Rn×n является положительно полуопределенной, если для всех x , Rn x Bx 0. Матрица B является положительно определенной, если x Bx > 0 при x = 0.

Легко показать, что AA положительно полуопределенно, так как:

$$x\,A\,Ax = (Ax)\,(Ax) = (Ax)\cdot(Ax) = Ax \qquad {}^{2}_{2} \quad 0.$$

На самом деле, если столбцы A линейно независимы, то AA положительно определена.

В более общем случае предположим, что мы хотим решить симметричную положительно определенную систему Cx = d. Как мы уже выяснили, мы можем LU-факторизовать матрицу C, но на самом деле мы можем сделать несколько лучше. Запишем C Rn×n как блочную матрицу:

$$C = \begin{matrix} c11 & v^{\top} \\ v & \tilde{C} \end{matrix}$$

где v Rn 1 и C˜ R(n 1)×(n 1) . Благодаря особой структуре C мы можем сделать следующее наблюдение:

$$Ce1 = 1\ 0\cdots 0\ e\ 1 \qquad \begin{matrix} c11 & v^{\top} \\ v & \tilde{C} \end{matrix} \qquad \begin{matrix} 1 \\ 0 \\ \vdots \\ 0 \end{matrix}$$

$$= 1\ 0\cdots 0 \qquad \begin{matrix} c11 \\ в \end{matrix}$$

$$= c11$$

> 0, так как C положительно определена и e1 =0.

Это показывает, что — игнорируя числовые проблемы — нам не нужно использовать поворот, чтобы гарантировать, что c11 = 0 для исключения Гаусса на первом этапе.

Продолжая исключение Гаусса, мы можем применить матрицу прямой замены E, которая в общем случае имеет вид

$$Э = \begin{matrix} 1/\sqrt{c11} & 0 \\ p & I(n\ 1)\times(n\ 1) \end{matrix} .$$

Здесь вектор r Rn 1 содержит числа, кратные строке 1, чтобы сократить остальную часть первого столбца C. Мы также масштабируем строку 1 на $1/\sqrt{\ }$ c11 по причинам, которые вскоре станут очевидными!

По замыслу, после прямой замены мы знаем продукт

$$EC = \begin{matrix} \sqrt{c11}\ в\ / & c11\sqrt{\ } \\ 0 & Д \end{matrix}$$

для некоторого D R(n 1)×(n 1) .

Вот где мы отклоняемся от исключения Гаусса: вместо того, чтобы переходить ко второй строке, мы можем пост-умножить на E, чтобы получить произведение ECE:

$$ECE = (EC)E \quad c11$$

$$\text{"="} \quad v / \overline{\quad} c11 \quad \overline{\quad} \quad 1/ \overline{c11} \quad p$$

$$0 \; Д \quad 0 \quad I(n \;\; 1)\times(n \;\; 1)$$

$$\text{"="} \quad 1 \; 0$$

$$0 \; D\tilde{}$$

То есть мы исключили первую строку и первый столбец C! Кроме того, легко проверить, что матрица $D\tilde{}$ также является положительно определенной.

Мы можем повторить этот процесс, чтобы удалить все строки и столбцы C симметрично. Уведомление что мы использовали как симметрию, так и положительную определенность для вывода факторизации, поскольку

- симметрия позволяла нам применять одну и ту же E к обеим сторонам, и
- положительная определенность гарантирует, что c11 > 0, что означает существование $\overline{\quad} c11$.

В конце концов, аналогично LU-факторизации, мы теперь получаем факторизацию C = LL для нижней треугольной матрицы L. Это известно как факторизация Холецкого C. , где D — диагональная матрица, позволяет избежать этой проблемы, и его легко вывести из приведенного выше обсуждения.

Факторизация Холецкого важна по ряду причин. Наиболее заметно то, что для хранения L требуется половина памяти, чем LU-разложение C или даже самого C, поскольку элементы над диагональю равны нулю, и, как и в LU, решение Cx = d так же просто, как прямая и обратная замена .
В упражнениях вы изучите другие свойства факторизации.

В конце концов, код для факторизации Холецкого может быть очень кратким. Чтобы получить особенно ком пакта, предположим, что мы выбираем произвольную строку k и пишем L в блочной форме, изолируя эту строку:

$$Л = \begin{matrix} L11 \; 0 \\ к \\ L31 \quad к \end{matrix} \quad \begin{matrix} 0 \; кк \; 0 \\ \\ L33 \end{matrix}$$

Здесь L11 и L33 — нижние треугольные квадратные матрицы. Тогда выполнение продукта дает:

$$ЛЛ = \begin{matrix} L11 \; 0 \\ к \\ L31 \quad к \end{matrix} \quad \begin{matrix} 0 \; кк \; 0 \\ \\ L33 \end{matrix} \quad \begin{matrix} л_{11} \quad к\, л\, кк\, (\, 31 \\ 0 \quad к) \\ 0 \; 0 \; л \; 33 \end{matrix}$$

$$\begin{matrix} \times & \times & \times \\ \text{"="} \quad L \; 11 \; кк + к & 2 \; кк & \times \\ \times & \times & \times \end{matrix}$$

Мы опускаем значения произведения, которые не нужны для нашего вывода.

В конце концов, мы знаем, что можем написать C = LL. Средний элемент продукта показывает:

$$кк = \overline{\quad ckk - к \quad}^{\,2}_{2}$$

где k Rk 1 содержит элементы k-й строки L слева от диагонали. Более того,
средний левый элемент продукта показывает

$$L11k = ck$$

где ck содержит элементы C в той же позиции ask . Поскольку L11 является нижнетреугольным, это
система может быть решена прямой заменой!

Обратите внимание, что наше обсуждение выше дает алгоритм вычисления факторизации Холецкого.
сверху вниз, так как L11 уже будет вычислена к моменту, когда мы достигнем строки k. Мы предоставляем
приведенный ниже псевдокод, адаптированный из CITE:

```
// Принимает на вход матрицу размера n на n A [i, j]
// Редактирует A на месте, чтобы получить факторизацию Холецкого в его нижнем треугольнике

для k от 1 до n {
        // Назад - подставить, чтобы найти l_k
        for i from 1 to k -1 { // элемент i из l_k
                сумма = 0;
                для j от 1 до i -1
                        сумма += A [i, j]* A [k, j];
                A [к , я ] = ( A [к , i ] - сумма )/ A [i , я];
        }

        // Применяем формулу для l_kk
        нормаКвадрат = 0
        для j от 1 до i -1
                normSquared += A [k, j]^2;
        A [к , k ] = sqrt ( A [k , k ] - нормаКвадрат );
}
```

Как и в случае факторизации LU, этот алгоритм работает за O(n ^{3}) время.

3.2.2 Разреженность

Многие линейные системы уравнений естественным образом обладают свойствами разреженности, а это означает, что большинство
элементы A в системе Ax = b ровно равны нулю. Разреженность может отражать определенную структуру в
данной проблемы, включая следующие варианты использования:

- При обработке изображений многие системы для редактирования и понимания фотографий выражают взаимосвязь между
  значениями пикселей и значениями их соседей в сетке изображения. Изображение
  может быть точкой в Rp для p пикселей, но при решении Ax =b для нового изображения размера p A Rp×p
  может иметь только O(p), а не O(p ^{2}) отличен от нуля, так как каждая строка включает только один пиксель.
  и его верхние/нижние/левые/правые соседи.

- В машинном обучении графическая модель использует структуру графа G (V, E) для выражения распределения
  вероятностей по нескольким переменным. Каждая переменная представлена с помощью узла v V
  график, а ребро e E представляет собой вероятностную зависимость. Линейные системы, возникающие в
  этот контекст часто имеет одну строку на вершину v V с ненулевыми значениями только в столбцах, содержащих
  v и его соседи.

- В вычислительной геометрии формы часто выражаются с помощью наборов связанных треугольников.
  вместе в сетку. Уравнения для сглаживания поверхности и другие задачи снова связывают положения и другие значения
  в данной вершине с теми, которые находятся в их соседях по сетке.

Пример 3.5 (Гармоническая параметризация). Предположим, мы хотим использовать изображение для текстурирования треугольной сетки. Сетку можно представить как набор вершин V R3 , соединенных ребрами E V × V в треугольники. Поскольку вершины геометрии находятся в R3 , мы должны найти способ сопоставить их с плоскостью изображения, чтобы сохранить текстуру как изображение. Таким образом, мы должны присвоить текстурные координаты t(v) R2 на плоскости изображения каждому элементу v V. См. иллюстрацию на рисунке НОМЕР.

Одна из стратегий создания этой карты включает одно линейное решение. Предположим, что сетка имеет дисковую топологию, то есть ее можно сопоставить с внутренней частью круга на плоскости. Для каждой вершины vb на границе сетки мы можем указать положение vb , поместив ее на окружность. Внутри мы можем попросить, чтобы позиция карты текстуры была средним значением соседних позиций:

$$1\ t(v) = \frac{}{|n(v)|} \quad \text{т(ш)}$$

Здесь n(v) V — множество соседей v V на сетке. Таким образом, каждому элементу v V ставится в соответствие линейное уравнение, которое либо фиксирует его на границе, либо требует, чтобы его положение равнялось среднему значению соседних положений. Это |V| × |B| система уравнений приводит к устойчивой стратегии параметризации, известной как гармоническая параметризация; матрица системы имеет только O(|V|) отличных от нуля слотов, соответствующих вершинам и их соседям.

Конечно, если A Rn×n разрежена до такой степени, что содержит значения O(n) , а не O(n) , нет причин ^{2}) ненулевой хранить A как матрицу размера n × n. Вместо этого методы хранения разреженных матриц сохраняют только O (n) отличных от нуля в более разумной структуре данных, например, в списке троек строк/столбцов/значений Выбор матричной структуры данных включает в себя рассмотрение вероятных операций, которые будут выполняться с матрицей, возможно, включая умножение, итерацию по ненулевым значениям. или перебор отдельных строк или столбцов.

К сожалению, легко видеть, что LU-факторизация разреженного A может не привести к разреженным матрицам L и U; эта потеря структуры сильно ограничивает применимость использования этих методов для решения Ax = b, когда A большое, но разреженное. К счастью, есть много прямых разреженных решателей, адаптирующих LU к разреженным матрицам, которые могут производить LU-подобную факторизацию, не вызывая большого заполнения или дополнительных ненулевых значений; обсуждение этих методов выходит за рамки этого текста. В качестве альтернативы для получения приближенных решений линейных систем использовались итерационные методы; мы отложим обсуждение этих методов до будущих глав.

Некоторые матрицы не только разрежены, но и структурированы. Например, трехдиагональная система линейные уравнения имеют следующий набор ненулевых значений:

× ×
× × ×
× × ×
× × ×
× ×

В упражнениях, следующих за этой главой, вы получите специальную версию исключения Гаусса для работы с этой простой ленточной структурой.

В других случаях матрицы не могут быть разреженными, но могут допускать разреженное представление. Для экзамена ple, рассмотрим циклическую матрицу:

abcddabccdabbcda

Очевидно, что эту матрицу можно сохранить, используя только значения a, b, c, d. Специализированные методы для этого и других классов матриц хорошо изучены и часто более эффективны, чем общее исключение Гаусса.

3.3 Анализ чувствительности

Как мы видели, важно исследовать матрицу линейной системы, чтобы выяснить, обладает ли она особыми свойствами, которые могут упростить процесс решения. Разреженность, положительная определенность, симметрия и т. д. могут дать ключи к правильному решателю для использования в конкретной задаче.

Однако даже если данная стратегия решения может работать теоретически, не менее важно понимать, насколько хорошо мы можем доверять ответу на линейную систему, данному конкретным решателем. Например, из-за округления и других дискретных эффектов может случиться так, что реализация исключения Гаусса для решения Ax = b дает решение x0 такое, что 0 < Ax0 b 1; другими словами, x0 лишь приблизительно решает систему.

Одним из способов понять вероятность этих эффектов аппроксимации является анализ чувствительности. В этом подходе мы спрашиваем, что может произойти с x, если вместо решения Ax = b в действительности мы решим возмущенную систему уравнений (A + δA)x = b + δb. Существует два способа рассмотрения выводов, сделанных в результате этого типа анализа:

1. Вероятно, мы допускаем ошибки при представлении A и b из-за округления и других эффектов. Затем этот анализ показывает наилучшую возможную точность, которую мы можем ожидать для x с учетом допущенных ошибок, представляющих проблему.

2. Если наш решатель генерирует приближение x0 к решению Ax = b, это точное решение системы Ax0 = b0 , если мы определяем b0 Ax0 (убедитесь, что вы понимаете, почему это предложение не является тавтологией!). Понимание того, как изменения в x0 влияют на изменения в b0, показывает, насколько чувствительна система к слегка неправильным ответам.

Обратите внимание, что наше обсуждение здесь похоже на наши определения прямой и обратной ошибки в предыдущих главах и действительно мотивировано ими.

3.3.1 Матричные и векторные нормы

Прежде чем мы сможем обсудить чувствительность линейной системы, мы должны быть несколько осторожны, чтобы определить, что означает, что изменение δx является «малым». Как правило, мы хотим измерить длину или норму вектора x. Мы уже встречались с двумя нормами вектора:

$$2 \text{ x } 2 \quad \text{x } 1 \qquad + \text{x } \tfrac{2}{2}\, 2 + \cdots + \text{x } \text{н}$$

для x Rn . Эта норма популярна благодаря своей связи с евклидовой геометрией, но она ни в коем случае не является единственной нормой на Rn . Обычно мы определяем норму следующим образом:

Определение 3.2 (Векторная норма). Векторной нормой называется функция : Rn [0,) , удовлетворяющий следующим · условия:

- x = 0 тогда и только тогда, когда x =0.

• cx = |c|x для всех скаляров c R и векторов x Rn .

• x + y x + y для всех x,y Rn .

Хотя мы используем два нижних индекса · 2 для обозначения двойной нормы вектора, если не указано иное, мы будем использовать обозначение x для обозначения двойной нормы x. Помимо этой нормы, есть много других примеров: • p-норма xp для p 1,

определяемая как:

$$xp \quad (|x1|\ p + |x2|\ p + \cdots + |xn|\ p)^{1/n}$$

Особое значение имеет 1-норма или норма «такси», заданная

$$x1 \qquad \sum_{k=1}^{n} |xk\ |.$$

Эта норма получила свое прозвище, потому что она представляет собой расстояние, которое такси проезжает между двумя точками в городе, где дороги проходят только с севера на юг и с востока на запад.

• -норма x определяется как:

$$x \qquad \max(|x1|,\ |x2|, \cdots,\ |xn|).$$

В некотором смысле многие нормы на Rn совпадают. В частности, предположим, что мы говорим, что две нормы эквивалентны, если они удовлетворяют следующему свойству:

Определение 3.3 (Эквивалентные нормы). Две нормы · и · эквивалентны, если существуют константы clow и chigh такие, что

$$clowx \quad x \quad chighx$$

для всех x Rn .

Это условие гарантирует, что с точностью до некоторых постоянных множителей все нормы согласуются в отношении того, какие векторы бывают «маленькие» и «большие». Фактически мы сформулируем без доказательства известную теорему из анализа:

Теорема 3.2 (Эквивалентность норм на Rn). Все нормы в Rn эквивалентны.

Этот несколько неожиданный результат подразумевает, что все векторные нормы ведут себя примерно одинаково, но выбор нормы для анализа или постановки конкретной проблемы может иметь огромное значение.
Например, на R3 -норма считает, что вектор (1000, 1000, 1000) имеет ту же норму, что и (1000, 0, 0) , тогда как на 2-норму, безусловно, влияют дополнительные ненулевые значения.

Так как мы возмущаем не только векторы, но и матрицы, мы также должны уметь брать норму матрицы. Конечно, основное определение нормы не меняется на Rn×m. По этой причине мы можем «развернуть» любую матрицу из Rm×n в вектор из Rnm , чтобы принять любую норму вектора к матрицам. Одной из таких норм является норма Фробениуса, заданная формулой

$$AFro \qquad \sqrt{\sum_{я, ж} a_{иж}^2}$$

Однако такая адаптация векторных норм не всегда имеет большое значение. В частности, приоритетом для понимания структуры матрицы A часто является ее действие на векторы, то есть вероятные результаты при умножении A на произвольный x. С этой мотивацией мы можем определить норму, индуцированную векторной нормой, следующим образом:

Определение 3.4 (Индуцированная норма). Норма на Rm×n , индуцированная нормой · на Rn , определяется выражением

$$A \quad \text{макс}\{Ax : x = 1\}.$$

То есть индуцированная норма — это максимальная длина изображения единичного вектора, умноженная на A.

Поскольку векторные нормы удовлетворяют cx = |c|x, легко видеть, что это определение эквивалентно требованию

$$A \quad \max_{x \in Rn \setminus \{0\}} \frac{\text{Топор}}{\text{Икс}}.$$

С этой точки зрения норма A, индуцированная · , является наибольшим достижимым отношением нормы Ax к норме входа x.

Это общее определение несколько затрудняет вычисление нормы A при заданной матрице A и выборе · . К счастью, матричные нормы, индуцированные многими популярными векторными нормами, можно упростить. Приведем некоторые такие выражения без доказательства:

- Индуцированная одна норма A – это максимальная сумма любого одного столбца A:

$$A1 = \text{макс}_{1 \quad j \quad n} \sum_{i=1}^{м} |aij|$$

- Индуцированная -норма A – это максимальная сумма любой одной строки A:

$$A \quad = \text{макс}_{1 \quad i \quad m} \sum_{j=1}^{н} |aij|$$

- Индуцированная двойная норма, или спектральная норма, A Rn×n – это квадратный корень из наибольшего собственного значения A A. То есть,

$$A \quad {}_2^2 = \max\{\lambda : \text{существует } x \quad R \quad {}^{н} \text{ где } Ax = \lambda x\}$$

По крайней мере, первые две нормы относительно легко вычислить; к третьему мы еще вернемся при обсуждении проблем с собственными значениями.

3.3.2 Номера условий

Теперь, когда у нас есть инструменты для измерения действия матрицы, мы можем определить число обусловленности линейной системы, адаптировав наше общее определение чисел обусловленности из главы 1. Мы следуем развитию, представленному в CITE.

Предположим, что мы возмущаем δA матрицы A и соответствующим возмущением δb. Для каждого ε 0, игнорируя технические аспекты обратимости, мы можем записать вектор x(ε) как решение

$$(A + \varepsilon \cdot \delta A)x(\varepsilon) = b + \varepsilon \cdot \delta b.$$

Если мы продифференцируем обе части по ε и применим правило произведения, мы получим следующий результат:

$$\delta A \cdot x + (A + \varepsilon \cdot \delta A) = \delta b \; d\varepsilon \quad \frac{dx}{}$$

В частности, при ε = 0 находим

$$\delta A \cdot x(0) + A = \delta b - \frac{dx}{d\varepsilon}\Big|_{\varepsilon=0}$$

или, что то же самое,

$$\frac{dx}{d\varepsilon}\Big|_{\varepsilon=0} = A^{-1}(\delta b - \delta A \cdot x(0)).$$

Используя разложение Тейлора, мы можем написать

$$x(\varepsilon) = x + \varepsilon x(0) + O(\varepsilon^2),$$

где мы определяем $x(0) = \frac{dx}{d\varepsilon}\Big|_{\varepsilon=0}$. Таким образом, мы можем расширить относительную ошибку, допущенную путем решения возмущенная система:

$$\frac{\|x(\varepsilon) - x(0)\|}{\|x(0)\|} \text{ "=" } \frac{\|\varepsilon x(0) + O(\varepsilon^2)\|}{\|x(0)\|} \text{ по разложению Тейлора}$$

$$\text{"=" } \frac{\|\varepsilon A^{-1}(\delta b - \delta A \cdot x(0)) + O(\varepsilon^2)\|}{\|x(0)\|} \text{ по производной, которую мы вычислили}$$

$$\frac{|\varepsilon|}{\|x(0)\|}\|(A^{-1}\delta b + A^{-1}(\delta A \cdot x(0))) + O(\varepsilon^2)\| \text{ по неравенству треугольника } \|A+B\| \le \|A\| + \|B\|$$

$$|\varepsilon|\|A^{-1}\|\frac{\|\delta b\|}{\|x(0)\|} + \|\delta A\| + O(\varepsilon^2) \text{ тождеством } \|AB\| \le \|A\|\|B\|$$

$$= |\varepsilon|\|A^{-1}\|\|A\|\left(\frac{\|\delta b\|}{\|A\|\|x(0)\|} + \frac{\|\delta A\|}{\|A\|} + O(\varepsilon^2)\right)$$

$$|\varepsilon|\|A^{-1}\|\|A\|\left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} + O(\varepsilon^2)\right), \text{ так как } \|Ax(0)\| = \|b\| \le \|A\|\|x(0)\|$$

$$= |\varepsilon|\|A^{-1}\|\|A\|\left(\frac{\|\delta b\|}{b} + \frac{\|\delta A\|}{\|A\|} + O(\varepsilon^2)\right), \text{ так как } \|Ax(0)\| = \|b\| \le \|A\|\|x(0)\|$$

поскольку по определению $\|Ax(0)\| = \|b\|$

Здесь мы применили некоторые свойства матричной нормы, вытекающие из соответствующих свойств векторов. Обратите внимание, что сумма $D := \|\delta b\|/\|b\| + \|\delta A\|/\|A\|$ кодирует относительные возмущения A и b. С этой точки зрения мы в первом порядке ограничили относительную погрешность возмущения системы по ε с помощью множителя $\kappa := \|A\|\|A^{-1}\| \ge 1$:

$$\frac{\|x(\varepsilon) - x(0)\|}{\|x(0)\|} \le \varepsilon \cdot D \cdot \kappa + O(\varepsilon^2)$$

Таким образом, величина κ ограничивает обусловленность линейных систем с участием A. По этой причине мы даем следующее определение:

Определение 3.5 (Матричное число обусловленности). Число обусловленности $A \in \mathbb{R}^{n\times n}$ для данной матричной нормы $\cdot$ есть

$$\text{конд } A = \|A\|\,\|A^{-1}\|.$$

Если A необратима, мы берем cond A .

Легко видеть, что cond $A \geq 1$ для всех A, что масштабирование A не влияет на его число обусловленности и что число обусловленности единичной матрицы равно 1. Эти свойства контрастируют с определителем, который может увеличиваться и уменьшаться. по мере масштабирования A.

Если $\cdot$ индуцируется векторной нормой и A обратим, то имеем

$$\|A^{-1}\| = \max_{x=0} \frac{\|A^{-1}x\|}{\|x\|} \quad \text{по определению}$$

$$= \max_{y=0} \frac{\|y\|}{\|Ay\|} \quad \text{подставив } y = A^{-1}x$$

$$= \min_{y=0} \frac{\|Ay\|}{\|y\|} \quad \text{принимая взаимное}$$

В этом случае номер условия A определяется как:

$$\text{конд. } A = \max_{x=0} \frac{\|Ax\|}{\|x\|} \min_{y=0} \frac{\|Ay\|}{\|y\|}$$

Другими словами, cond A измеряет максимальное или минимально возможное растяжение вектора x при A.

В более общем смысле желательное свойство устойчивости системы $Ax = b$ состоит в том, что если A или b возмущена, решение x существенно не изменится. Наша мотивация для cond A показывает, что, когда число обусловленности мало, изменение x мало по сравнению с изменением A или b, как показано на рисунке НОМЕР. В противном случае небольшое изменение параметров линейной системы может вызвать большие отклонения x; эта нестабильность может привести к тому, что линейные решатели будут делать большие ошибки в x из-за округления и других приближений в процессе решения. может быть столь же сложным, как

Норма A^{-1} вычисление полной обратной границы A, число условия A^{-1}. Один из способов снизить состоит в том, чтобы применить тождество $A^{-1}x = A^{-1}x$. Таким образом, для любого $x = 0$ $A^{-1}x/x$. Таким образом, мы можем написать A^{-1}

$$\text{конд } A = \|A\|\,\|A^{-1}\| \max_{x=0} \frac{\|A^{-1}x\|}{\|x\|}.$$

Итак, мы можем ограничить число обусловленности, решив $A^{-1}x$ для некоторых векторов x; конечно, необходимость линейного решателя для нахождения $A^{-1}x$ создает циклическую зависимость от числа обусловленности для оценки качества оценки! Когда $\cdot$ индуцируется двукратной нормой, в следующих главах мы дадим более надежные оценки.

3.4 Проблемы

Что-то вроде:

• Регрессия ядра на примере §3.1.1

• Решение по методу наименьших норм для Ax =b, в противном случае матрица наименьших квадратов обратима

• Вариационные версии тихоновской регуляризации/«гребневой» регрессии (не обычный подход к этому, но что угодно); завершение недоопределенной истории таким образом

1 • L подходы к регуляризации для контраста — нарисуйте картину, почему следует ожидать разреженности, нарисуйте единичных кругов, покажите, что норма p не является нормой при p < 1, примите предел при p 0

• Mini-Riesz – выведите матрицу для внутреннего продукта, используйте, чтобы показать, как вращать пространство

• трехдиагональное решение

• свойства номера условия

Глава 4

Пространства столбцов и QR

Один из способов интерпретации линейной задачи Ax = b для x состоит в том, что мы хотим записать b как линейную комбинацию столбцов A с весами, заданными в x. Эта перспектива не меняется, когда мы допускаем, что A Rm×n не является квадратом, но решение может не существовать или быть уникальным в зависимости от структуры пространства столбцов. По этим причинам некоторые методы факторизации матриц и анализа линейных систем ищут более простые представления пространства столбцов для устранения неоднозначности и более точного охвата, чем факторизация на основе строк, такая как LU.

4.1 Структура нормальных уравнений

Как мы показали, необходимым и достаточным условием для того, чтобы x было решением задачи наименьших квадратов Ax b, является то, что x удовлетворяет нормальным уравнениям (AA)x = A b. Эта теорема предполагает, что решение метода наименьших квадратов является достаточно простым расширением линейных методов. Такие методы, как факторизация Холецкого, также показывают, что особая структура задач наименьших квадратов может быть использована в интересах решателя.

Однако есть одна большая проблема, ограничивающая использование этого подхода. А пока предположим, что A квадрат; то мы можем написать:

$$\text{конд } AA = A\,A(A\,A)^{1}$$

$$= A\,A\,(A)^{1}\,(A)^{1} \quad \text{в зависимости от выбора } \cdot$$

$$= (\text{конд } A)^2$$

То есть число обусловленности AA примерно равно квадрату числа обусловленности A! Таким образом, в то время как общие линейные стратегии могут работать с AA, когда задача наименьших квадратов является «легкой», когда столбцы A почти линейно зависимы, эти стратегии, вероятно, будут генерировать значительную ошибку, поскольку они не имеют дело с A напрямую.

Интуитивно основная причина того, что cond A может быть большим, заключается в том, что столбцы A могут выглядеть «похожими». Думайте о каждом столбце A как о векторе в Rm. Если два столбца ai и aj удовлетворяют ai , то остаточная длина aj методом наименьших квадратов b Ax, вероятно, не сильно пострадает, если мы заменим числа, кратные ai , на числа, кратные aj , или наоборот. Этот широкий диапазон почти, но не полностью эквивалентных решений приводит к плохой обусловленности. Однако, хотя результирующий вектор x нестабилен, произведение

Топор остается практически без изменений, по замыслу нашей замены. Следовательно, если мы хотим решить Ax b просто для записи b в пространстве столбцов A, будет достаточно любого решения.

Чтобы решить такие плохо обусловленные задачи, мы будем использовать альтернативную стратегию, уделяя больше внимания пространству столбцов матрицы A, а не прибегая к операциям со строками, как при исключении Гаусса. Таким образом, мы можем явно идентифицировать такие близкие зависимости и работать с ними численно стабильным образом.

4.2 Ортогональность

Мы определили, когда задача наименьших квадратов сложна, но мы могли бы также спросить, когда она наиболее проста. Если мы сможем свести систему к простому случаю, не вызывая при этом проблем с обусловленностью, мы найдем более устойчивый способ обойти проблемы, описанные в § 4.1.

Очевидно, что проще всего решить линейную систему In×nx = b: решение просто x b! Мы вряд ли введем эту конкретную линейную систему в наш решатель явно, но мы можем сделать это случайно при решении метода наименьших квадратов. В частности, даже когда A = In×n — на самом деле, A не обязательно должна быть квадратной матрицей — мы можем в особенно удачных обстоятельствах обнаружить, что нормальная матрица AA удовлетворяет AA = In×n. Чтобы не путать с общим случаем, мы будем использовать букву Q для обозначения такой матрицы.

Простая молитва о том, что QQ = In×n , вряд ли приведет к желаемой стратегии решения, но мы можем изучить этот случай, чтобы увидеть, как он становится таким благоприятным. Запишем столбцы матрицы Q в виде векторов q1, · · · ,qn Rm. Тогда нетрудно проверить, что произведение QQ имеет следующую структуру:

$$KK = \begin{bmatrix} д\,1 \\ д\,2 \\ \vdots \\ q\,п \end{bmatrix} \begin{bmatrix} | & | & & | \\ q1 & q2 & \cdots & qn \\ | & | & & | \end{bmatrix} \quad "=" \quad \begin{bmatrix} q1 \cdot q1 & q1 \cdot q2 & \cdots & q1 \cdot qn \\ q2 \cdot q1 & q2 \cdot q2 & \cdots & q2 \cdot qn \\ \vdots & \vdots & \cdots & \vdots \\ qn \cdot q1 & qn \cdot q2 & \cdots & qn \cdot qn \end{bmatrix}$$

Установка выражения справа равным In×n дает следующее соотношение:

$$ци \cdot qj = \begin{matrix} 1 \text{ при } i = j \\ 0 \text{ при } i = j \end{matrix}$$

Другими словами, столбцы Q имеют единичную длину и ортогональны друг другу. Мы говорим, что они образуют ортонормированный базис для пространства-столбца Q:

Определение 4.1 (Ортонормированный; ортогональная матрица). Набор векторов {v1, · · · ,vk} ортонормирован, если vi = 1 для всех i и vi ·vj = 0 для всех i = j. Квадратная матрица, столбцы которой ортонормированы, называется ортогональной матрицей.

Мы мотивировали нашу дискуссию вопросом, когда мы можем ожидать, что QQ = In×n. Теперь легко видеть, что это происходит, когда столбцы матрицы Q ортонормированы. Кроме того, если Q является квадратным и обратимым с QQ = In×n, то, просто умножая обе части этого выражения на Q 1, мы находим Q 1 = Q. Таким образом, решение Qx = b в этом случае так же просто, как умножение обе стороны транспонированным Q.

Ортонормированность также имеет сильную геометрическую интерпретацию. Напомним из главы 0, что мы можем рассматривать два ортогональных вектора a и b как перпендикулярные. Итак, ортонормированный набор векторов

просто представляет собой набор перпендикулярных векторов единичной · Если Q ортогонален, то его действие длины в Rn , не влияющих на длину векторов:

$$Qx^2 = x\,Q\,Qx = x\,I_{n\times n}x = x \cdot x = x^2$$

Точно так же Q не может влиять на угол между двумя векторами, поскольку:

$$(Qx) \cdot (Qy) = x\,Q\,Qy = x\,I_{n\times n}y = x \cdot y$$ С этой точки

зрения, если Q ортогонален, то Q представляет собой изометрию Rn , т. е. сохраняет длины и углы. Другими словами, он может вращать или отражать векторы, но не может масштабировать или сдвигать их.
На высоком уровне линейная алгебра ортогональных матриц проще, потому что их действие не влияет на геометрию основного пространства каким-либо нетривиальным образом.

4.2.1 Стратегия для неортогональных матриц

За исключением особых обстоятельств, большинство наших матриц A при решении Ax = b или соответствующей задачи наименьших квадратов не будут ортогональны, поэтому механизм § 4.2 не применяется напрямую. По этой причине мы должны сделать некоторые дополнительные вычисления, чтобы связать общий случай с ортогональным.

Возьмем матрицу A $R^{m\times n}$, и обозначим его пространство столбцов как col A; напомним, что col A представляет диапазон столбцов A. Теперь предположим, что матрица B $R^{n\times n}$ обратима. Мы можем сделать простое наблюдение о пространстве столбцов AB относительно

пространства столбцов A: Лемма 4.1 (инвариантность пространства столбцов). Для любого A $R^{m\times n}$ и обратимого B $R^{n\times n}$

столбец A = столбец AB.

Доказательство. Предположим, что b col A. Тогда по определению умножения на A существует x такой, что Ax = b. Тогда (AB) · (B 1x) = Ax =b, sob col AB. Обратно, пусть c col AB, значит, существует y, для которого (AB)y = c. Тогда A · (By) = c, что показывает, что c находится в столбце A. □

Вспомним описание «матрицы исключения» метода исключения Гаусса: мы начали с матрицы A и применили матрицы операций со строками Ei так, что последовательность A, E1A, E2E1A, . . . представляли собой последовательно более простые линейные системы. Приведенная выше лемма предлагает альтернативную стратегию для ситуаций, в которых нам важно пространство столбцов: применять операции со столбцами к A путем постумножения, пока столбцы не станут ортонормированными. То есть мы получаем произведение Q = AE1E2 · · · Ek такое, что Q ортонормирован. Пока Ei обратимы, лемма показывает, что col Q = col A. Обращение этих операций дает факторизацию A = QR $_{k}^{1}{}_{k}\,{}_{1}^{1}{}_{1}\,{}_{1}^{1}\cdot E\cdots E$

для R = E. задачи квадратов Ax b могут упроститься. В частности, когда A = QR, мы можем записать решение A Ax = A b следующим образом:

$$\begin{aligned} x &= (A\,A)^{-1}A\,b \\ &= (RQ\;QR)\;\;{}^{1}R\,Q\,b, \text{ так как } A = QR \\ &= (R\,R)\;\;{}^{1}R\,Q\,b, \text{ так как } Q \text{ ортогонален} \\ &= P\;\;{}^{1}(R\,)\;\;{}^{1}R\,Q\,b, \text{ так как } (AB)^{1} = B^{-1}A^{-1} \\ &= R\;\;{}^{1}Q\,b \end{aligned}$$

Или, что то же самое, Rx = Q b

Таким образом, если мы создадим R как треугольную матрицу, то решение линейной системы Rx = Qb будет таким же простым, как обратная подстановка.

Наша задача в оставшейся части главы — разработать стратегии такой факторизации.

4.3 Ортогонализация Грама-Шмидта

Наш первый подход к нахождению QR-факторизаций является самым простым для описания и реализации, но может страдать числовыми проблемами. Мы используем его здесь в качестве первоначальной стратегии, а затем усовершенствуем его, улучшив операции.

4.3.1 Прогнозы

Предположим, у нас есть два вектора a и b. Тогда мы могли бы легко спросить: «Какое число, кратное a, ближе всего к tob?» Математически эта задача эквивалентна минимизации ca b по всем возможным c R. Если представить a и b как матрицы размера n × 1, а c — как матрицу размера 1 × 1, то это не более чем нетрадиционная задача наименьших квадратов. · c б. В этом случае нормальные уравнения показывают a · c = ab, или

$$c = \frac{a \cdot б}{a \cdot a} \text{ "=" } \frac{a \cdot б}{a^2}.$$

Обозначим эту проекцию b на a как:

$$\text{проект}_a = ca = \frac{a \cdot bb}{a \cdot a} a = \frac{a \cdot б}{a^2}$$

Очевидно, a б параллелен a. А как насчет остатка b — proj a б? Мы можем сделать простой вычисление проекта, чтобы узнать:

$$\begin{aligned} a \cdot (б \quad \text{проект}_a\ б) &= a \cdot б \quad a \cdot \frac{a \cdot б}{a^2} \\ &= a \cdot б - \frac{a \cdot б}{a^2}(a \cdot a) \\ &= a \cdot б \quad a \cdot б \\ &= 0 \end{aligned}$$

Таким образом, мы разложили b на компоненту, параллельную a, и другую компоненту, ортогональную тоа.

Предположим теперь, что aˆ1, aˆ2, · · · , aˆk ортонормированы; мы наденем шляпы на векторы единичной длины. Тогда для любого одиночного i мы можем увидеть:

$$\text{proj}\hat{a}i\ b = (\hat{a}i \cdot b)\hat{a}i$$

Член нормы не появляется, потому что aˆi = 1 по определению. Мы могли бы спроецировать b на промежуток {aˆ1, · · · , aˆk} , минимизируя следующую энергию по c1, . . . , ск € R:

$$c1a\hat{}1 + c2a\hat{}2 + \cdots + ck\,a\hat{}k \quad b \quad {}^{2} \overset{"="}{} \sum_{i=1}^{k}\sum_{j=1}^{k} cicj(a\hat{}i \cdot a\hat{}j) \quad 2b \cdot \sum_{i=1}^{k} cia\hat{}i + b \cdot b$$

применяя v $^{2} = v \cdot v$

$$\overset{"="}{} \sum_{i=1}^{k} c_{i}^{2} \quad 2ci\, b \cdot a\hat{}i + b \quad {}^{2}$$ так как aˆi ортонормированы

Обратите внимание, что второй шаг здесь допустим только из-за ортонормированности. Как минимум, производная по ci равна нулю для каждого ci , что дает:

$$ci = a\hat{}i \cdot b$$

Таким образом, мы показали, что когда aˆ1, · · · , aˆk ортонормированы, выполняется следующее соотношение:

$$\text{projspan}\{a\hat{}1,\cdots,a\hat{}k\}b = (a\hat{}1 \cdot b)a\hat{}1 + \cdots + (a\hat{}k \cdot b)a\hat{}k$$

Это просто расширение нашей проекционной формулы, и с помощью аналогичного доказательства легко увидеть, что

$$a\hat{}i \cdot (b \quad \text{projspan}\{a\hat{}1,\cdots,a\hat{}k\}b) = 0.$$

То есть мы разделили b на составляющую, параллельную размаху aˆi, и перпендикулярную невязку.

4.3.2 Ортогонализация Грама-Шмидта

Наши наблюдения выше приводят к простому алгоритму ортогонализации или нахождению ортогонального базиса {aˆ1, · · · , aˆk} , размах которого такой же, как у набора линейно независимых входных векторов {v1, · · · ,vk}:

1 комплект

$$\hat{}1 \quad \frac{v1}{v1}.$$

То есть мы принимаем aˆ1 за единичный вектор, параллельный v1.

2. Для i от 2 до k,

(а) Вычислите проекцию

$$pi \quad \text{projspan}\{a\hat{}1,\cdots,a\hat{}i \quad 1\}vi.$$

По определению aˆ1, · · · , aˆi 1 ортонормированы, поэтому применима наша вышеприведенная формула.

(б) Определить

$$я \quad \frac{ви - пи}{ви - пи}.$$

Этот метод, известный как «ортогонализация Грама-Шмидта», представляет собой прямое применение нашего обсуждения выше. Ключ к доказательству этой техники состоит в том, чтобы заметить, что span {v1, · · · ,vi} = span {aˆ1, · · · , aˆi} для каждого i {1, · · · , k}. Шаг 1 явно делает это случайным для i = 1, а для i > 1 определение aˆi на шаге 2b просто удаляет проекцию на векторы, которые мы уже видели.

Если мы начнем с матрицы A, столбцами которой являются v1, · · · ,vk , то мы можем реализовать Грэма Шмидта как серию операций со столбцами над A. Деление столбца i матрицы A на его норму эквивалентно последующему умножению A на ak × k диагональная матрица. Точно так же вычитание проекции столбца на ортонормированные столбцы слева от него, как на шаге 2, эквивалентно последующему умножению на верхнетреугольную матрицу: обязательно поймите, почему это так! Таким образом, наше обсуждение в § 4.2.1 применимо, и мы можем использовать Грамма-Шмидта, чтобы получить факторизацию A = QR.

К сожалению, алгоритм Грама-Шмидта может вносить серьезные числовые нестабильности из-за шага вычитания. Например, предположим, что мы предоставили векторы v1 = (1, 1) и v2 = (1 + ε, 1) в качестве входных данных для Грамма-Шмидта для некоторого 0 < ε 1. Обратите внимание, что очевидным основанием для span {v1,v2} является { (1, 0), (0, 1)}. Но если применить Грамма-Шмидта, то получим:

$$\hat{1} = \frac{v1}{v1} = \frac{1}{\sqrt{2}} \quad 1\ 1$$

$$p2 = 2\frac{2+e}{} \quad 1\ 1$$

$$v2 \quad p2 = \frac{1+e\ 1}{} \quad \frac{2+e}{2} \quad 1\ 1$$

$$= \frac{1}{2} \quad \varepsilon\ \varepsilon$$

Обратите внимание, что $v2 \quad p2 = (\quad 2/2) \cdot \varepsilon$, поэтому для вычисления aˆ2 потребуется деление на скаляр порядка ε. Деление на маленькие числа — нестабильная числовая операция, которой следует избегать.

4.4 Преобразования домохозяев

В §4.2.1 мы мотивировали построение QR-факторизации пост-умножением и операциями со столбцами. Эта конструкция разумна в контексте анализа пространств столбцов, но, как мы видели в нашем выводе алгоритма Грама-Шмидта, полученные в результате численные методы могут быть нестабильными.

Вместо того, чтобы начинать с A и пост-умножать на операции со столбцами, чтобы получить Q = AE1 · · · Ek , однако мы можем сохранить нашу высокоуровневую стратегию от исключения Гаусса. То есть мы можем начать с A и предварительно умножить на ортогональные матрицы Qi , чтобы получить Qk · · · Q1A = R; эти Q будут действовать как операции со строками, исключая элементы A до тех пор, пока результирующий продукт R не станет верхнетреугольным. Тогда, благодаря ортогональности Q, мы можем записать A = QR, получив коэффициент QR , так как произведение ортогональных матриц ортогонально.

Матрицы операций со строками, которые мы использовали в исключении Гаусса и LU, не будут достаточными для факторизации QR, поскольку они не ортогональны. Было предложено несколько альтернатив; мы представим одну общую стратегию, представленную в 1958 году Алстоном Скоттом Хаусхолдером.

Пространство ортогональных матриц размера n × n очень велико, поэтому мы должны найти меньшее пространство Qi , с которым легче работать. Из наших геометрических обсуждений в § 4.2 мы знаем, что ортогональные матрицы должны сохранять углы и длины, поэтому интуитивно они могут вращать и отражать только векторы.

К счастью, отражения можно легко записать в виде проекций, как показано на рисунке НОМЕР. Предположим, у нас есть вектор b, который мы хотим отразить над вектором v. Мы показали, что невязка r b proj 2proj Мы в b перпендикулярна v. Как и на рисунке НОМЕР, разность в b b отражает b над v.

можем расширить нашу формулу отражения следующим образом:

$$\text{2проект}_{\text{в}}\cdot bb \quad b = 2\frac{v}{\text{в}\cdot\text{в}}v \quad \text{b по определению проекции}$$

$$= 2\text{в}\cdot\frac{\text{вб}}{\text{вв}} \quad \text{b с использованием матричных обозначений}$$

$$\text{"="} \quad \frac{2\text{вв}}{\text{вв}} \quad \text{В×n б}$$

Hv b, где отрицательный знак вводится для согласования с другими методами лечения

Таким образом, мы можем думать об отражении b над v как о применении линейного оператора Hv к b! Конечно, Hv без отрицания по-прежнему ортогонален, поэтому с этого момента мы будем использовать его.

Предположим, мы делаем первый шаг прямой замены во время исключения Гаусса. Затем мы хотим предварительно умножить A на матрицу, которая переводит первый столбец A, который мы будем обозначать, в некоторое число, кратное первому единичному вектору e1. Другими словами, мы хотим для некоторого c R:

$$\text{ce1} = \text{Хва}$$

$$= \text{В} \times \text{п} - \frac{2\text{вв}}{\text{вв}} \quad \text{а}$$

$$= \text{а} - 2\text{в}\ \frac{\text{ва}}{\text{вв}}$$

Перемещение терминов по шоу

$$\text{v} = (\text{а} \quad \text{ce1})\cdot\frac{\text{вв}}{2\text{va}}$$

Другими словами, v должно быть параллельно разности a ce1. На самом деле масштабирование v не влияет на формулу для Hv , поэтому мы можем выбрать v = a ce1. Затем, чтобы наши отношения сохранялись, мы должны иметь

$$1 = \frac{\text{вв}}{2\text{ва}}$$

$$\text{"="} \quad \frac{\text{а}^{\ 2} \quad 2\text{ce1}\cdot\text{а} + \text{с}^{\ 2}}{2(\text{а}\cdot\text{а} \quad \text{ce1}\cdot\text{а})}$$

$$\text{Или } 0 = \text{а}^{\ 2\,2\,-\,\text{в}} \quad = \quad \text{с} = \pm\text{а}$$

При таком выборе с мы показали:

$$\text{HvA} = \begin{matrix} \text{с} \times \times \times \\ 0 \times \times \times \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ 0 \times \times \times \end{matrix}$$

Мы только что выполнили шаг, похожий на прямое исключение, используя только ортогональные матрицы!

Исходя из этого, в обозначениях CITE на k-м шаге триангуляризации имеем вектор a, который мы можем разделить на две составляющие:

$$a = \begin{matrix} a1 \\ a2 \end{matrix}$$

Здесь a1 Rk и a2 Rm k . Мы хотим найти такое v, что

$$Хва = \begin{matrix} a1 \\ 0 \\ \vdots \\ 0 \end{matrix}$$

Следуя выводу, параллельному приведенному выше, легко показать, что

$$v = \begin{matrix} 0 \\ a2 \end{matrix} \quad \text{сек}$$

совершает именно это преобразование при c = ±a2; мы обычно выбираем знак c, чтобы избежать отмены, заставляя его иметь знак, противоположный знаку k-го значения ina.

Таким образом, алгоритм Хаусхолдера QR довольно прост. Для каждого столбца матрицы A мы вычисляем v, аннулирующее нижние элементы столбца, и применяем Hv к A. Конечным результатом является верхняя треугольная матрица R = Hvn · · · Hv1 A. Ортогональная матрица Q задается произведением H, которое может быть неявно сохранен как список векторов v, который соответствует нижнему треугольнику, как v1, показанному выше.

4.5 Сокращенная QR-факторизация

Мы завершаем наше обсуждение, возвращаясь к наиболее общему случаю Ax b, когда A Rm×n не является квадратом. Обратите внимание, что оба алгоритма, которые мы обсуждали в этой главе, могут разлагать неквадратные матрицы A на произведения QR, но результат несколько отличается:

- При применении Грама-Шмидта мы выполняем операции со столбцами над A, чтобы получить Q путем ортогонализации. По этой причине размерность A равна размерности Q, что дает Q Rm×n и R Rn×n:
- При использовании отражений Хаусхолдера мы получаем Q как произведение числа m × m матрицы отражения, оставляя R Rm×n .

Предположим, что мы находимся в типичном случае для метода наименьших квадратов, для которого m n. Мы по-прежнему предпочитаем использовать метод Хаусхолдера из-за его численной стабильности, но теперь матрица Q размером m × m может оказаться слишком большой для хранения! К счастью, мы знаем, что R является верхнетреугольным. Например, рассмотрим структуру матрицы 5 × 3 R:

$$P = \begin{matrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Легко видеть, что все, что находится ниже верхнего квадрата размера n × n матрицы R, должно быть равно нулю, что дает упрощенную формулу.
катион:

$$A = QR = Q1\ Q2 \begin{matrix} P1 \\ 0 \end{matrix} = Q1R1$$

Здесь Q1 Rm×n и R1 Rn×n по-прежнему содержит верхний треугольник R. Это называется «приведенным» QR-факторизация A, поскольку столбцы Q1 содержат базис для пространства столбцов A, а не для всего Rm; он занимает гораздо меньше места. Обратите внимание, что обсуждение в § 4.2.1 все еще применимо, поэтому сокращенная факторизация QR может использоваться для метода наименьших квадратов аналогичным образом.

4.6 Проблемы

- тридиагонализация с Хаусхолдером
- Даны
- Недоопределенный QR

Глава 5

Собственные векторы

Обратимся теперь к нелинейной задаче о матрицах: нахождение их собственных значений и собственных векторов. Собственные векторы x и соответствующие им собственные значения λ квадратной матрицы A определяются уравнением Ax = λx. Есть много способов увидеть, что эта задача нелинейна.
Например, существует произведение неизвестных λ и x, и, чтобы избежать тривиального решения x = 0, мы ограничиваем x = 1; это ограничение является круговым, а не линейным. Благодаря такой структуре наши методы нахождения собственных пространств будут существенно отличаться от методов решения и анализа линейных систем уравнений.

5.1 Мотивация

Несмотря на произвольный вид уравнения Ax = λx, задача нахождения собственных векторов и собственных значений естественно возникает во многих случаях. Мы мотивируем нашу дискуссию несколькими примерами ниже.

5.1.1 Статистика

Предположим, у нас есть механизм для сбора нескольких статистических наблюдений о наборе элементов. Например, в медицинском исследовании мы можем узнать возраст, вес, артериальное давление и частоту сердечных сокращений 100 пациентов. Тогда каждый пациент i может быть представлен точкой xi в R4 , хранящей эти четыре значения.

Конечно, такая статистика может демонстрировать сильную корреляцию. Например, пациенты с более высоким кровяным давлением могут иметь более высокий вес или частоту сердечных сокращений. По этой причине, хотя мы собирали наши данные в R4 , в действительности они могут — в некоторой приблизительной степени — жить в пространстве более низкого измерения, лучше фиксируя отношения между различными переменными.

А пока предположим, что на самом деле существует одномерное пространство, аппроксимирующее наш набор данных. Затем мы ожидаем, что все точки данных будут почти параллельны некоторому вектору v, так что каждую можно записать как xi civ для различных ci R. Ранее мы знали, что наилучшее приближение xi параллельно v есть proj в хи :

$$\text{проект}_{\text{в}} = \frac{\text{xi} \cdot \text{v xi}}{\text{в} \cdot \text{в}} \text{v по определению}$$

$$= (\text{xi} \cdot \text{v}\hat{})\text{v}\hat{}\text{, поскольку v} \cdot \text{v} = \text{v}^2$$

Здесь мы определяем v^ v/v. Конечно, величина v не имеет значения для рассматриваемой задачи, поэтому разумно искать в пространстве единичных векторов v^.

Следуя схеме наименьших квадратов, у нас есть новая задача оптимизации:

$$\text{минимизировать} \quad \sum_i \left\| x_i - \text{proj}_{\hat{v}} x_i \right\|^2$$
$$\text{такой, что } \|\hat{v}\| = 1$$

Мы можем немного упростить нашу цель оптимизации:

$$\sum_i \left\| x_i - \text{proj}_{\hat{v}} x_i \right\|^2 = \sum_i \left\| x_i - (x_i \cdot \hat{v})\hat{v} \right\|^2 \quad \text{по определению проекции}$$
$$= \sum_i \|x_i\|^2 - (x_i \cdot \hat{v})^2 \quad \text{поскольку } \|\hat{v}\| = 1 \text{ и } \|w\|^2 = w \cdot w$$
$$= \text{константа} - \sum_i (x_i \cdot \hat{v})^2$$

Этот вывод показывает, что мы можем решить эквивалентную задачу оптимизации:

$$\text{максимизировать } \|X\hat{v}\|^2$$
$$\text{такой, что } \|\hat{v}\|^2 = 1,$$

где столбцы X являются векторами x_i. Обратите внимание, что $\|X\hat{v}\|^2 = \hat{v}^T X^T X \hat{v}$, поэтому в примере 0.27 вектор v соответствует собственному вектору $X^T X$ с наибольшим собственным значением. Вектор v^ известен как первый главный компонент набора данных.

5.1.2 Дифференциальные уравнения

Многие физические силы могут быть записаны как функции положения. Например, сила пружины между двумя частицами в положениях x и y в $\mathbb{R}^3$ может быть записана как k(x y) по закону Гука; такие силы пружины используются для аппроксимации сил, скрепляющих ткань во многих системах моделирования. Хотя эти силы не обязательно линейны по положению, мы часто аппроксимируем их линейным образом. В частности, в физической системе с n частицами положения всех частиц одновременно кодируются вектором $X \in \mathbb{R}^{3n}$. Тогда, если мы примем такое приближение, мы можем написать, что силы в системе приблизительно равны $F \approx AX$ для некоторой матрицы A.

Вспомните второй закон Ньютона F = ma, или сила равна массе, умноженной на ускорение. В нашем контексте мы можем написать диагональную матрицу масс $M \in \mathbb{R}^{3n \times 3n}$, содержащую массу каждой частицы в системе. Тогда мы знаем, что F = MX, где штрих означает дифференцирование по времени. Конечно, $X' = (X')'$, поэтому в итоге мы имеем систему уравнений первого порядка:

$$\frac{d}{dt}\begin{bmatrix} X \\ V \end{bmatrix} = \begin{bmatrix} 0 & I_{3n\times 3n} \\ M^{-1}A & 0 \end{bmatrix}\begin{bmatrix} X \\ V \end{bmatrix}$$

Здесь мы одновременно вычисляем как положение в $X \in \mathbb{R}^{3n}$, так и скорости $V \in \mathbb{R}^{3n}$ всех n частиц как функции времени.

В более общем смысле дифференциальные уравнения вида $x' = Ax$ появляются во многих контекстах, включая моделирование ткани, пружин, тепла, волн и других явлений. Предположим, мы знаем собственные векторы

x1, . . . ,xk группы A, такие что Axi = λixi . Если мы запишем начальное условие дифференциального уравнения через собственные векторы, как

$$x(0) = c_1x_1 + \cdots + c_kx_k ,$$

то решение уравнения можно записать в замкнутом виде:

$$x(t) = c_1e^{\lambda_1 t}\lambda_k t\, x_1 + \cdots + c_ke\, x_k ,$$

Это решение легко проверить вручную. То есть, если мы запишем начальные условия этого дифференциального уравнения в терминах собственных векторов A, то мы знаем его решение для всех моментов времени t 0 бесплатно. Конечно, эта формула не является концом истории моделирования: нахождение полного набора собственных векторов A дорого обходится, а A может меняться со временем.

5.2 Спектральное вложение

Предположим, у нас есть набор из n элементов в наборе данных и мера wij 0 того, насколько похожи каждая пара элементов i и j; будем считать wij = wji. Например, может быть, нам дана коллекция фотографий, и мы используем wij , чтобы сравнить сходство их цветовых распределений. Возможно, мы пожелаем отсортировать фотографии на основе их сходства, чтобы упростить просмотр и изучение коллекции.

Одна из моделей упорядочения коллекции может состоять в том, чтобы присвоить номер xi каждому элементу i, попросив, чтобы подобным объектам были присвоены одинаковые номера. Мы можем измерить, насколько хорошо задание группирует похожие объекты, используя энергию

$$E(x) = \sum_{ij} w_{ij}(x_i - x_j)^2 .$$

То есть E(x) требует, чтобы элементы i и j с высокими показателями подобия wij были сопоставлены с соседними значениями.

Конечно, минимизация E(x) без ограничений дает очевидный минимум: xi = const. для всех я. Добавление ограничения x = 1 не удаляет это постоянное решение! В частности, взятие xi = 1/ n для всех i дает x = 1 и E(x) = 0 неинтересным образом. Таким образом, мы должны удалить и этот случай:

$$\begin{aligned} &\text{минимизировать } E(x) \\ &\text{такой, что } x^2 = 1 \\ &1 \cdot x = 0 \end{aligned}$$

Обратите внимание, что наше второе ограничение требует, чтобы сумма x была равна нулю.

Еще раз мы можем упростить энергию:

$$\begin{aligned} E(x) &= \sum_{ij} w_{ij}(x_i - x_j)^2 \\ &= \sum_{ij} w_{ij}(x_i^2 - 2x_ix_j + x_j^2) \\ &= \sum_i x_i^2 \sum_j w_{ij} - 2\sum_{ij} w_{ij}x_ix_j + \sum_j x_j^2 \sum_i w_{ij} \\ &\quad \text{для } a_i = \sum_j w_{ij} \text{ и } b_j = \sum_i w_{ij} \\ &= x^T(A - 2W + B)x , \text{ где } \operatorname{diag}(A) = a \text{ и } \operatorname{diag}(B) = b = x^T(2A - 2W)x \text{ в силу} \\ &\quad \text{симметрии } W \end{aligned}$$

Легко проверить, что 1 является собственным вектором 2A 2W с собственным значением 0. Что еще интереснее, собственный вектор, соответствующий второму наименьшему собственному значению, соответствует решению нашей задачи минимизации выше! (TODO: добавьте доказательство KKT из лекции)

5.3 Свойства собственных векторов

Мы установили множество приложений, нуждающихся в вычислении собственного пространства. Однако прежде чем мы сможем изучить алгоритмы для этой цели, мы более подробно рассмотрим структуру проблемы собственных значений.

Мы можем начать с нескольких определений, которые, вероятно, очевидны на данный момент:

Определение 5.1 (собственное значение и собственный вектор). Собственным вектором x = 0 матрицы A Rn×n называется любой вектор, удовлетворяющий условию Ax = λx для некоторого λ R; соответствующее λ известно как собственное значение. Комплексные собственные значения и собственные векторы удовлетворяют тем же соотношениям с λ C и x Cn .

Определение 5.2 (Спектр и спектральный радиус). Спектр A — это множество собственных значений A. Спектральный радиус ρ(A) — это собственное значение λ, максимизирующее |λ|.

Масштаб собственного вектора не важен. В частности, масштабирование собственного вектора x с помощью c дает A (cx) = cAx = cλx = λ(cx), поэтому cx является собственным вектором с тем же собственным значением. Мы часто ограничиваем наш поиск, добавляя ограничение x = 1. Даже это ограничение не устраняет полностью неоднозначность, так как теперь ±x оба являются собственными векторами с одним и тем же собственным значением.

Алгебраические свойства собственных векторов и собственных значений легко могли бы заполнить целую книгу. Мы ограничим наше обсуждение несколькими важными теоремами, влияющими на разработку численных алгоритмов; мы будем следить за развитием CITE AXLER. Во-первых, мы должны проверить, что каждая матрица имеет хотя бы один собственный вектор, чтобы наши поиски не были напрасными. Наша обычная стратегия состоит в том, чтобы заметить, что если λ является собственным значением таким, что Ax = λx, то (A λIn×n)x = 0; таким образом, λ является собственным значением ровно тогда, когда матрица A λIn×n не имеет полного ранга.

Лемма 5.1 (теорема ЦИТЭ 2.1). Каждая матрица A Rn×n имеет хотя бы один (комплексный) собственный вектор.

Доказательство. Возьмем любой вектор x Rn\{0}. Множество {x, Ax, A 2x, · · · Anx } должно быть линейно зависимым, поскольку оно содержит n + 1 вектор в n измерениях. Итак, существуют константы c0, . . . , cn R , cn = 0 такие, что

$$0 = c_0 x + c_1 A x + \cdots + c_n A^n x$$

Мы можем записать многочлен

$$f(z) = c_0 + c_1 z + \cdots + c_n z^n.$$

По основной теореме алгебры существует n корней zi C таких, что f(z) = cn(z z1)(z z2) · · · (z zn).

Тогда у нас есть:

$$\begin{aligned} 0 &= c_0 x + c_1 A x + \cdots + c_n A^n x \\ &= (c_0 I_{n\times n} + c_1 A + \cdots + c_n A^n)x = \\ &c_n(A - z_1 I_{n\times n}) \cdots (A - z_n I_{n\times n})x \text{ по нашей факторизации} \end{aligned}$$

Таким образом, по крайней мере один A zi In×n имеет нулевое пространство, что показывает, что существует v с Av = ziv, что и требовалось. □

Есть еще один факт, который стоит проверить, чтобы мотивировать наше обсуждение вычисления собственных векторов. нация:

Лемма 5.2 (предложение 2.2 ЦИТЭ). Собственные векторы, соответствующие разным собственным значениям, должны быть линейно независимыми.

Доказательство. Предположим, что это не так. Тогда существуют собственные векторы x1, · · · ,xk с различными собственными значениями λ1, · · · , λk , линейно зависимые. Отсюда следует, что существуют коэффициенты c1, . . . , ck не все нули с 0 = c1x1 + · · · + ckxk . Если предварительно умножить на матрицу (A λ2 In×n) · · · (A λk In×n), найдем:

$$0 = (A - \lambda_2 I_{n\times n}) \cdots (A - \lambda_k I_{n\times n})(c_1x_1 + \cdots + c_kx_k)$$
$$= c_1(\lambda_1 - \lambda_2)\cdots(\lambda_1 - \lambda_k)x_1 \text{ , так как } Ax_i = \lambda_i x_i$$

Поскольку все λi различны, это показывает, что c1 = 0. Аналогичное доказательство показывает, что остальные ci должны быть равны нулю, что противоречит линейной зависимости. □

Эта лемма показывает, что матрица размера n × n может иметь не более n различных собственных значений, поскольку набор из n собственных значений дает n линейно независимых векторов. Максимальное количество линейно независимых собственных векторов, соответствующих одному собственному значению λ , известно как геометрическая кратность λ.

Однако неверно, что матрица должна иметь ровно n линейно независимых собственных векторов. Это имеет место для многих матриц, которые мы будем называть недефектными:

Определение 5.3 (Бездефектный). Матрица A Rn×n называется недефектной или диагонализируемой, если ее собственные векторы порождают Rn .

Мы называем такую матрицу диагонализируемой по следующей причине: если матрица диагонализируема, то она имеет n собственных векторов x1, . . . ,xn Rn с соответствующими (возможно, не единственными) собственными значениями λ1, . . . , λн. Возьмем столбцы X в качестве векторов xi и определим D как диагональную матрицу с собственными значениями λ1, . . . , λn по диагонали. Тогда по определению собственных значений имеем AX = XD; это просто «сложенная» версия Axi = λixi . Другими словами,

$$D = X^{-1}AX,$$

это означает, что A диагонализуется преобразованием подобия A X 1AX : Определение 5.4

(Подобные матрицы). Две матрицы A и B подобны, если существует такая T, что B = T 1AT.

Подобные матрицы имеют одинаковые собственные значения, так как если Bx = λx, то T 1ATx = λx. Эквивалентно, A(Tx) = λ(Tx), показывая, что Tx является собственным вектором с собственным значением λ.

5.3.1 Симметричные и положительно определенные матрицы

Учитывая наше специальное рассмотрение нормальных матриц AA, неудивительно, что симметричные и/или положительно определенные матрицы имеют особую структуру собственных векторов. Если мы сможем проверить любое из этих свойств, можно будет использовать специализированные алгоритмы для более быстрого извлечения их собственных векторов.

Во-первых, мы можем доказать свойство симметричных матриц, которое устраняет необходимость в сложных арифметика. Начнем с обобщения симметричных матриц на матрицы из Cn×n :

Определение 5.5 (комплексно сопряженное). Комплексно-сопряженным числом z a + bi C является $\bar{z}$ a bi.

Определение 5.6 (сопряженное транспонирование). Сопряженное транспонирование A Cm×n равно AH A¯.

Определение 5.7 (эрмитова матрица). Матрица A Cn×n эрмитова, если A = AH .

Обратите внимание, что симметричная матрица A Rn×n автоматически является эрмитовой, поскольку не имеет комплексной части. Имея это небольшое обобщение, мы можем доказать свойство симметрии для собственных значений.
Наше доказательство будет использовать скалярное произведение векторов в Cn , заданное выражением

$$\langle x, y \rangle = \sum_i x_i \bar{y}_i,$$

где x,y Cn . Заметим, что это определение снова совпадает с x ·y, когда x,y Rn . По большей части свойства этого скалярного произведения совпадают со свойствами скалярного произведения на Rn , за исключением того, что v, w = w ,v. , заметный

Лемма 5.3. Все собственные значения эрмитовых матриц действительны.

Доказательство. Предположим, что A Cn×n эрмитово с Ax = λx. Масштабируя, мы можем принять x 1. Тогда $\|x\|^2 = \langle x, x \rangle =$ мы имеем:

$\lambda = \lambda \langle x, x \rangle$, так как x имеет норму 1

$= \lambda \langle x, x \rangle$ по линейности $\langle \cdot, \cdot \rangle$

$= \langle Ax, x \rangle$, так как $Ax = \lambda x$

$= \langle (Ax), x \rangle$ по определению $= \langle x, (A^H x) \rangle$ $\langle \cdot, \cdot \rangle$

путем разложения произведения и использования $\overline{ab} = \bar{a}\bar{b} = \langle x, \overline{A^H x} \rangle$ по

определению A^H и $\langle \cdot, \cdot \rangle$

$= \langle x, Ax \rangle$, так как $A^H = \bar{A} = \bar{\lambda} x, x \rangle$,

поскольку $Ax = \lambda x = \bar{\lambda}$, поскольку

x имеет норму 1

Таким образом, $\lambda = \bar{\lambda}$, что может произойти, только если λ R, что и требовалось. □

Симметричные и эрмитовы матрицы также обладают особым свойством ортогональности собственных векторов.
Торы:

Лемма 5.4. Собственные векторы, соответствующие различным собственным значениям эрмитовых матриц, должны быть ортогональны.

Доказательство. Предположим, что A Cn×n эрмитово, и пусть λ = μ с Ax = λx и Ay = μy. По предыдущей лемме мы знаем λ, μ R. Тогда Ax,y = λx,y. Но поскольку A эрмитово, мы можем также написать Ax,y = x, A Hy = x, Ay = μx,y. Таким образом, λx,y = μx,y. Поскольку λ = μ, мы должны иметь x,y = 0.

□

Наконец, мы можем сформулировать без доказательства венчающий результат линейной алгебры — спектральную теорему. Эта теорема утверждает, что ни одна симметричная или эрмитова матрица не может быть дефектной, а это означает, что матрица размера n × n, удовлетворяющая этому свойству, имеет ровно n ортогональных собственных векторов.

Теорема 5.1 (спектральная теорема). Предположим, что A Cn×n эрмитово (если A Rn×n , то пусть оно симметрично). Тогда A имеет ровно n ортонормированных собственных векторов x1, · · · ,xn с (возможно, повторяющимися) собственными значениями λ1, . . . , λн . Другими словами, существуют ортонормированная матрица собственных векторов X и диагональная матрица собственных значений D такие, что D = X AX.

Из этой теоремы следует, что любой вектор y Rn можно разложить в линейную комбинацию собственных векторов эрмитовой матрицы A. В этом базисе многие вычисления проще, как показано ниже:

Пример 5.1 (Вычисление с использованием собственных векторов). Возьми x1, . . . ,xn Rn — собственные векторы единичной длины симметричной матрицы A Rn×n . Предположим, мы хотим решить Ay = b. Мы можем написать

$$b = c_1x_1 + \cdots + c_nx_n,$$

где ci =b · xi по ортонормированности. Нетрудно догадаться о следующем решении:

$$\frac{c_1}{x_1}\ \frac{c_n}{\lambda_1}\ x_1 + \cdots + y = \frac{x_n}{\lambda_n}$$

В частности, мы находим:

$$Ай = A \quad \frac{c_1}{\lambda_1}\ \frac{c_n}{\lambda_n}\ x_1 + \cdots + x_n \quad c_1\ c_n$$

$$\text{"="} \quad \frac{Ax_1}{\lambda_1} + \cdots + \frac{Ax_н}{\lambda_n}$$

$$= c_1x_1 + \cdots + c_nx_n$$

=b, по желанию.

Приведенный выше расчет является как положительным, так и отрицательным результатом. Он показывает, что при заданных собственных векторах симметричного A такие операции, как инверсия, выполняются просто. С другой стороны, это означает, что найти полный набор собственных векторов симметричной матрицы A «как минимум» так же сложно, как решить Ax = b.

Возвращаясь от нашего экскурса в комплексные числа, мы вернемся к действительным числам, чтобы доказать один последний полезный, хотя и простой факт о положительно определенных матрицах:

Лемма 5.5. Все собственные значения положительно определенных матриц неотрицательны.

Доказательство. Возьмем положительно определенное A Rn×n и предположим, что Ax = λx, где x = 1. По положительной определенности мы знаем, что x Ax 0. Но x Ax = x (λx) = λx = λ, что и требовалось. □

5.3.2 Специализированные свойства[1]

Характеристический полином

Напомним, что определитель матрицы det A удовлетворяет соотношению det A = 0 тогда и только тогда, когда A обратима. Таким образом, одним из способов нахождения собственных значений матрицы является нахождение корней характеристического многочлена

$$pA(\lambda) = \det(A \quad \lambda In\times n).$$

Мы не будем определять определители в нашем обсуждении здесь, но упрощение pA показывает, что это полином n-й степени от λ. Это дает альтернативную причину, по которой существует не более n различных собственных значений, поскольку у этой функции не более n корней.

Исходя из этой конструкции, мы можем определить алгебраическую кратность собственного значения как его кратность как корня pA. Легко видеть, что алгебраическая кратность не меньше геометрической

[1]Этот раздел можно пропустить, если у читателей недостаточно знаний, но он включен для полноты картины.

множественность. Если алгебраическая кратность равна 1, корень называется простым, поскольку он соответствует одному собственному вектору, который линейно зависит от любых других. Собственные значения, у которых алгебраическая и геометрическая кратности не равны, называются дефектными.

В численном анализе мы избегаем обсуждения определителя матрицы. Хотя это удобная теоретическая конструкция, ее практическое использование ограничено. Детерминанты трудно вычислить. На самом деле алгоритмы собственных значений не пытаются найти корни pA , поскольку для этого потребовалось бы вычисление определителя. Кроме того, определитель det A не имеет ничего общего с обусловленностью A, поэтому близкий к нулю определитель det(A λIn×n) может не показывать, что λ почти является собственным значением A.

Нормальная форма Джордана

Мы можем диагонализовать матрицу только тогда, когда она имеет полное собственное пространство. Однако все матрицы подобны матрице в жордановой нормальной форме, которая имеет следующий вид: • Ненулевые

значения находятся на диагональных элементах aii и на «наддиагональных» элементах ai(i+1) .

• Диагональные значения — это собственные значения, повторяющиеся столько раз, сколько их кратно; матрица диагональ блока вокруг этих кластеров.

• Внедиагональные значения равны 1 или 0.

Таким образом, форма выглядит примерно так:

$$\begin{bmatrix} \lambda_1 & 1 & & & & \\ & \lambda_1 & 1 & & & \\ & & \lambda_1 & & & \\ & & & \lambda_2 & 1 & \\ & & & & \lambda_2 & \\ & & & & & \lambda_3 \\ & & & & & & \ddots \end{bmatrix}$$

Нормальная форма Жордана привлекательна теоретически, потому что она всегда существует, но структура 1/0 дискретна и неустойчива при численном возмущении.

5.4 Вычисление собственных значений

Вычисление и оценка собственных значений матрицы — хорошо изученная задача, имеющая множество возможных решений. Каждое решение настраивается для разных ситуаций, и для достижения максимальной подготовки или скорости требуется экспериментировать с несколькими методами. Здесь мы рассмотрим несколько наиболее популярных и простых решений проблемы собственных значений, часто встречающихся на практике.

5.4.1 Итерация мощности

Пока предположим, что A Rn×n симметрична. Тогда по спектральной теореме мы можем записать собственные векторы x1, . . . ,xn Rn ; мы сортируем их так, чтобы их соответствующие собственные значения удовлетворяли |λ1| |λ2| · · · |λn|.

Предположим, мы взяли произвольный вектор v. Поскольку собственные векторы матрицы A порождают Rn, мы можем написать:

$$v = c_1x_1 + \cdots + c_nx_n.$$

Затем,

$$\begin{aligned} Av &= c_1Ax_1 + \cdots + c_nAx_n \\ &= c_1\lambda_1x_1 + \cdots + c_n\lambda_nx_n \text{ , так как } Ax_i = \lambda_ix_i \\ \lambda_2 \lambda_n &= \lambda_1 \\ c_1x_1 &+ c_2x_2 + \cdots + c_nx_n \\ \lambda_1 \lambda_1 & \end{aligned}$$

$$A^2v = \lambda_1^2 \left(c_1x_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^2 c_2x_2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^2 c_nx_n \right)$$

$$\vdots$$

$$A^kv = \lambda_1^k \left(c_1x_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k c_2x_2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^k c_nx_n \right)$$

Обратите внимание, что при $k \to \infty$ отношение $(\lambda_i/\lambda_1)^k \to 0$, если только $\lambda_i = \lambda_1$, поскольку λ_1 по определению имеет наибольшую величину из всех собственных значений. Таким образом, если x является проекцией вектора v на пространство собственных векторов с собственными значениями λ_1, то при $k \to \infty$ все более точно выполняется следующее приближение:

$$\lambda_1^k x \approx A^k v$$

Это наблюдение приводит к чрезвычайно простому алгоритму вычисления собственного вектора x оператора A, соответствующее наибольшему собственному значению λ_1:

1. Возьмем $v_1 \in \mathbb{R}^n$ произвольный ненулевой вектор.
2. Итерировать до сходимости для увеличения k:

$$v_k = Av_{k-1}$$

Этот алгоритм, известный как степенная итерация, будет создавать векторы v_k, все более и более параллельные желаемому x_1. Он гарантированно сходится, даже если A асимметричен, хотя доказательство этого факта сложнее, чем приведенный выше вывод. Единственный случай, когда этот метод может дать сбой, — это если мы случайно выберем v_1 так, что $c_1 = 0$, но шансы на то, что это произойдет, ничтожны.

Конечно, если $|\lambda_1| > 1$, то $v_k \to \infty$ при $k \to \infty$, нежелательное свойство для арифметики с плавающей запятой. Напомним, что нас интересует только направление собственного вектора, а не его величина, поэтому масштабирование не влияет на качество нашего решения. Таким образом, чтобы избежать этой ситуации расхождения, мы можем просто нормализовать на каждом шаге, создавая алгоритм итерации нормализованной мощности:

1. Возьмем $v_1 \in \mathbb{R}^n$ произвольный ненулевой вектор.
2. Итерировать до сходимости для увеличения k:

$$w_k = Av_{k-1}$$

$$v_k = \frac{w_k}{\|w_k\|}$$

Заметьте, что мы не украсили норму · специальным нижним индексом. Математически любая норма достаточна для предотвращения проблемы расходимости, поскольку мы показали, что все нормы на Rn эквивалентны. На практике часто используют норму бесконечности · ; в этом случае легко проверить, что w k |λ1|.

5.4.2 Обратная итерация

Теперь у нас есть стратегия нахождения собственного значения λ1 наибольшей величины . Предположим, что A обратим, так что мы можем вычислить y = A 1v , решив Ay = v с помощью методов, описанных в предыдущих главах.

Если Ax = λx, то x = λA 1x или, что то же самое,

$$A^{-1}x = \frac{1}{\lambda}x.$$

если |a| |б| Таким образом, мы показали, что 1/λ является ^{1} с собственным вектором x. Обратите внимание, что собственным ^{1} значением ^{1} оператора A, тогда |b| |a| для любых a, b R, поэтому собственное значение наименьшей величины собственный вектор величины A, ^{1} A является наибольшим . Это наблюдение дает стратегию для нахождения λn , а не λ1 называемый итерацией обратной мощности:

1. Возьмем v1 Rn произвольный ненулевой вектор.

2. Итерировать до сходимости для увеличения k:

   (а) Решите для w k : Aw k = vk 1

   (б) Нормализация: vk = $\frac{ш\ к}{ш\ к}$

Мы постоянно решаем системы уравнений, используя одну и ту же матрицу A, что является прекрасным применением методов факторизации из предыдущих глав. Например, если мы напишем A = LU, то мы могли бы сформулировать эквивалентную, но значительно более эффективную версию обратной степенной итерации:

1. Фактор A = ЛУ

2. Возьмем v1 Rn произвольный ненулевой вектор.

3. Итерировать до сходимости для увеличения k:

   (а) Решить для yk с помощью прямой подстановки: Lyk = vk 1 (b)

   Решить для w k с помощью обратной подстановки: Uw k = yk (c)

   Нормировать: vk = $\frac{ш\ к}{ш\ к}$

5.4.3 Переключение

Предположим, что λ2 является собственным значением со второй по величине величиной A. Учитывая наш первоначальный вывод степенной итерации, легко увидеть, что степенная итерация сходится быстрее всего, когда |λ2/λ1| мала, так как в этом случае мощность (λ2/λ1) быстро убывает. Напротив, если это отношение близко к 1, может потребоваться много итераций мощности, прежде чем будет выделен один собственный вектор.

Если собственные значения A равны λ1, . . . , λn, то легко видеть, что собственные значения A σIn×n равны λ1 σ, . . . , λn σ. Тогда одна из стратегий быстрой сходимости степенной итерации состоит в том, чтобы выбрать σ таким образом, чтобы:

$$\frac{\lambda 2 - \sigma}{\lambda 1 - \sigma} < \frac{\lambda 2}{\lambda 1} .$$

Конечно, угадывание такого σ может быть искусством, так как собственные значения A, очевидно, изначально неизвестны. Точно так же, если мы считаем, что σ близко к собственному значению A, то A σIn×n имеет собственное значение, близкое к 0, которое мы можем выявить с помощью обратной итерации.

Одна стратегия, в которой используется это наблюдение, известна как итерация по фактору Рэлея. Если у нас есть фиксированное предположение о собственном векторе x матрицы A, то с помощью ЧИСЛА аппроксимация методом наименьших квадратов соответствующего собственного значения σ определяется выражением

$$\sigma \quad \frac{x \text{ топор}}{\text{Икс}_{2}^{2}} .$$

Эта дробь известна как частное Рэлея. Таким образом, мы можем попытаться увеличить сходимость, итерируя следующим образом:

1. В качестве v1 Rn возьмем произвольный ненулевой вектор или начальное приближение собственного вектора.

2. Итерировать до сходимости для увеличения k:

   (a) Запишите текущую оценку собственного значения

$$\sigma k = \frac{v\,k \quad 1Avk \quad 1}{\text{вк-1}_{2}^{2}}$$

   (b) Решите для w k : (A σk In×n)w k (c) = vk 1

   Нормируйте: vk = $\frac{\text{ш к}}{\text{ш к}}$

Эта стратегия сходится намного быстрее при хорошем начальном предположении, но матрица A σk In×n меняется на каждой итерации и не может быть предварительно факторизована с использованием LU или большинства других стратегий. Таким образом, требуется меньше итераций, но каждая итерация занимает больше времени!

5.4.4 Поиск нескольких собственных значений

До сих пор мы описывали методы нахождения одной пары собственное значение/собственный вектор: итерация по мощности для нахождения наибольшего собственного значения, обратная итерация для нахождения наименьшего и переход к промежуточным целевым значениям. Конечно, для многих приложений одного собственного значения недостаточно. К счастью, мы можем расширить наши стратегии, чтобы справиться и с этим случаем.

дефляция

Вспомните нашу стратегию степенной итерации: выберите произвольное v1 и итеративно умножайте его на A, пока не останется только самое большое собственное значение λ1 . Возьмем x1 за соответствующий собственный вектор.

Однако мы быстро отбросили маловероятную неисправность этого алгоритма, когда v1 ·x1 = 0.
В этом случае, независимо от того, сколько раз вы предварительно умножаете на A, вы никогда не восстановите вектор.

параллельно x1, так как вы не можете усилить нулевую составляющую. Вероятность выбора такой v1 ровно равна нулю, поэтому во всех случаях, кроме самых пагубных, итерация мощности остается безопасной.

Мы можем перевернуть этот недостаток с ног на голову, чтобы сформулировать стратегию нахождения более чем одного собственного значения, когда A симметрично. Предположим, что мы находим x1 и λ1 с помощью степенной итерации, как и раньше. Теперь мы перезапускаем итерацию мощности, но перед тем, как начать проект x1 из v1. Затем, поскольку собственные векторы A ортогональны, степенная итерация восстановит второе по величине собственное значение!

Из-за числовых проблем может случиться так, что применение A к вектору вводит небольшую компоненту, параллельную x1. На практике мы можем избежать этого эффекта, проецируя каждую итерацию. В итоге эта стратегия дает следующий алгоритм вычисления собственных значений в порядке убывания величины:

• Для каждого желаемого собственного значения = 1, 2, . . .

1. Возьмем v1 Rn произвольный ненулевой вектор.
2. Итерировать до сходимости для увеличения k: (a)

Проецировать собственные векторы, которые мы уже вычислили:

$$uk = vk \quad 1 \quad projspan\{x1,\ldots,x \quad 1\}\, vk \quad 1$$

(b) Умножить Auk = w k (c)

Нормировать: vk = $\frac{\text{ш к}}{\text{ш к}}$

3. Добавьте результат итерации к множеству xi

Внутренний цикл эквивалентен итерации мощности на матрице AP, где P проецирует x1, . . . ,x-1 .
Легко видеть, что AP имеет те же собственные векторы, что и A; его собственные значения равны λ , . . . , λn с обнулением остальных собственных значений.

В более общем плане стратегия дефляции включает изменение матрицы A таким образом, чтобы степенная итерация выявляла собственный вектор, который вы еще не вычислили. Например, AP — это модификация A, так что большие собственные значения, которые мы уже вычислили, обнуляются.

Наша проекционная стратегия терпит неудачу, если A асимметрична, так как в этом случае ее собственные векторы могут не быть ортогональными. В этом случае могут работать и другие менее очевидные стратегии дефляции. Например, предположим, что Ax1 = λ1x1 с x1 = 1. Возьмем H в качестве матрицы Хаусхолдера, такой что Hx1 = e1, первый стандартный базисный вектор. Преобразования подобия снова не влияют на набор собственных векторов, поэтому мы можем попробовать выполнить сопряжение с помощью H. Рассмотрим, что происходит, когда мы умножаем HAH на e1:

$$\begin{aligned} HAHe1 &= HAHe1 \text{ , так как H симметричен} \\ &= HAx1 \text{ , так как } Hx1 = e1 \text{ и } H = \lambda 1 Hx1 \text{ , } ^{2} = B \times n \\ &\text{так как } Ax1 = \lambda 1 x1 \qquad = \lambda 1 e1 \text{ по определению H} \end{aligned}$$

Таким образом, первый столбец HAH равен λ1e1, показывая, что HAH имеет следующую структуру (CITE ЗДОРОВЬЕ):

$$XA = \begin{matrix} \lambda 1 & б \\ 0 & Б \end{matrix} .$$

Матрица B R(n 1)×(n 1) имеет собственные значения λ2, . . . , λn. Таким образом, другая стратегия дефляции состоит в том, чтобы строить все меньшие и меньшие B-матрицы, где каждое собственное значение вычисляется с использованием степенной итерации.

Дефляция имеет тот недостаток, что мы должны вычислять каждый собственный вектор отдельно, что может быть медленным и может накапливать ошибку, если отдельные собственные значения неточны. Наши оставшиеся стратегии пытаются найти более одного собственного вектора за раз.

Напомним, что подобные матрицы A и B = T 1AT должны иметь одинаковые собственные значения. Таким образом, алгоритм, пытающийся найти собственные значения A, может свободно применять преобразования подобия к A. Конечно, применение T в целом может быть трудным предложением, поскольку фактически потребовало бы инвертирования T, поэтому мы ищем T-матрицы, обратные матрицы которых легко найти. применять.

Одним из наших мотивов для получения QR-факторизации было то, что матрица Q ортогональна, удовлетворяя Q-1 = Q. Таким образом, Q и Q-1 одинаково просты в применении, что делает ортогональные матрицы сильным выбором для преобразований подобия.

Но какую ортогональную матрицу Q выбрать? В идеале Q должен включать в себя структуру A, но при этом быть простым для вычислений. Неясно, как стратегически применять простые преобразования, такие как матрицы Хаусхолдера, для выявления множества собственных значений2, но мы знаем, как сгенерировать одно такое Q, просто разложив на множители A = QR. Тогда мы могли бы связать A с Q, чтобы найти: Q 1AQ = Q AQ = Q (QR)Q = (Q Q)RQ = RQ

Удивительно, но сопряжение A = QR ортогональной матрицей Q идентично записи произведения RQ!

Основываясь на этом рассуждении, в 1950-х годах несколько групп европейских математиков выдвинули гипотезу тот же элегантный итерационный алгоритм для нахождения собственных значений матрицы A:

1. Возьмем A1 = A.
2. При к = 1, 2, . . .
   (а) Фактор Ak = QkRk . (б)
   Запишите Ak+1 = RkQk .

Согласно приведенному выше выводу, все матрицы Ak имеют те же собственные значения, что и A. Кроме того, предположим, что матрицы Ak сходятся к некоторому A . Тогда мы можем разложить A = Q R , и по сходимости мы знаем, что A = Q R = R Q . По ЧИСЛУ собственные значения R — это просто значения вдоль диагонали R , а по ЧИСЛУ произведение R Q = A , в свою очередь, должно иметь те же собственные значения. Наконец, по построению A имеет те же собственные значения, что и A. Итак, мы показали, что если QR-итерация сходится, она выявляет собственные значения A прямым способом.

Конечно, приведенный выше вывод предполагает, что существует A с Ak A при k . На самом деле итерация QR — это стабильный метод, который гарантирует сходимость во многих важных ситуациях, и сходимость можно даже улучшить, изменив стратегии. Мы не будем здесь выводить точные условия, но вместо этого можем дать некоторое интуитивное представление о том, почему эта, казалось бы, произвольная стратегия должна сходиться. Ниже мы приводим некоторую интуицию для симметричного случая A = A, который легче анализировать благодаря ортогональности собственных векторов в этом случае.

Предположим, что столбцы A заданы a1, . . . ,an, и рассмотрим матрицу A ^{k} для больших к. Мы можно написать:

$$A^k = A^{k-1} \cdot A = \begin{vmatrix} & | & & | & \\ A^k a_1 & | & A^k a_2 & \cdots & A^k a_n \\ & | & & | & \end{vmatrix}$$

2Более продвинутые методы, однако, делают именно это!

Согласно нашему выводу степенной итерации, первый столбец A^k в общем случае параллелен собственному вектору x1 с наибольшей величиной $|\lambda 1|$ так как мы взяли вектор a1 и много раз умножили его на A.
Применяя нашу интуицию из дефляции, предположим, что мы проецируем a1 из второго столбца вектора A^k. Этот который должен быть почти параллелен x2, поскольку это второе по значимости собственное значение! Выполнение индуктивно, факторизация $A^k = QR$ даст набор почти собственных векторов в виде столбцов Q в порядке убывания величины собственного значения с соответствующими собственными значениями по диагонали R.

Конечно, вычисление A^k для большого k приводит число обусловленности A к k-й степени, поэтому QR на результирующей матрице, скорее всего, не удастся; это ясно видно, поскольку все столбцы A должны выглядеть как x1 для больших k. Однако мы можем сделать следующее наблюдение:

$$
\begin{aligned}
A &= Q_1R_1 \\
A^2 &= (Q_1R_1)(Q_1R_1) \\
&= Q_1(R_1Q_1)R_1 \\
&= Q_1Q_2R_2R_1 \text{, используя приведенное выше обозначение итерации QR, поскольку } A_2 = R_1Q_1 \\
&\vdots \\
A^k &= Q_1Q_2 \cdots Q_kR_kR_{k-1} \cdots R_1
\end{aligned}
$$

Группировка переменных Q_i и переменных R_i по отдельности обеспечивает QR-факторизацию A, и мы ожидаем, что столбцы $Q_1 \cdots Q_k$ будут сходиться к собственным векторам A^k. Таким образом,

Аналогичным образом можно найти

$$
\begin{aligned}
A &= Q_1R_1 \\
A_2 &= R_1Q_1 \text{ по нашей конструкции итерации QR} \\
&= Q_2R_2 \text{ по определению факторизации} \\
A_3 &= R_2Q_2 \text{ из итерации QR} \\
&= Q_2^{-1}A_2Q_2, \text{ так как } A_2 = Q_2R_2 \\
&= Q_2^{-1}R_1Q_1Q_2, \text{ так как } A_2 = R_1Q_1 \\
&= Q_2^{-1}Q_1^{-1}AQ_1Q_2, \text{ так как } A = Q_1R_1 \\
&\vdots \\
A_{k+1} &= Q_k^{-1} \cdots Q_2^{-1}Q_1^{-1}AQ_1Q_2 \cdots Q_k \text{ индуктивно} \\
&= (Q_1Q_2 \cdots Q_k)^{-1}A(Q_1Q_2 \cdots Q_k)
\end{aligned}
$$

где A_k — k-я матрица из итерации QR. Таким образом, A_{k+1} — это просто матрица A, сопряженная произведением $Q_1Q_2 \cdots Q_k$. Ранее мы утверждали, что столбцы матрицы $Q_1Q_2 \cdots Q_k$ сходятся к собственным векторам матрицы A. Таким образом, поскольку сопряжение матрицей собственных векторов дает диагональную матрицу собственных значений, мы знаем, что $A_{k+1} = Q_1Q_2 \cdots Q_k AQ_1Q_2 \cdots Q_k$ будет иметь приближенные собственные значения A вдоль его диагонали при $k \to \infty$.

Наше обоснование итерации QR включало анализ столбцов A итераций k при k как расширение мощности. В более общем случае для вектора b Rn , мы можем рассмотреть так называемую матрицу Крылова

$$K_k = \begin{bmatrix} | & | & | & & | \\ b & Ab & A^2b & \cdots & A^{k-1}b \\ | & | & | & & | \end{bmatrix}.$$

Методы анализа Kk для нахождения собственных векторов и собственных значений обычно известны как методы подпространств Крылова. Например, итерационный алгоритм Арнольди использует ортогонализацию Грама-Шмидта для поддержания ортогонального базиса {q1, . . . ,qk} для пространства столбца Kk :

1. Начнем с того, что возьмем q1 в качестве произвольного вектора единичной нормы.

2. При к = 2, 3, . . .

   (a) Takeak = Aqk 1 (b)

   Спроецируйте значения q, которые вы уже вычислили:

$$b_k = a_k \quad \text{projspan}\{q_1,\ldots,q_{k-1}\}\, a_k$$

   (c) Перенормируйте, чтобы найти следующее qk = bk/bk.

Матрица Qk , столбцами которой являются векторы, найденные выше, является ортогональной матрицей с тем же пространством столбцов, что и Kk , и оценки собственных значений могут быть восстановлены из структуры Q AQk . Использование метода Грама-Шмидта делает этот метод нестабильным, и синхронизация становится все хуже по мере увеличения k, однако для того, чтобы сделать его возможным, требуется так много расширений. Например, одна стратегия включает в себя выполнение нескольких итераций Arnoldi, использование выходных данных для получения лучшего предположения для начального q1 и перезапуск. Методы этого класса подходят для задач, требующих нескольких собственных векторов на одном из концов спектра без вычисления полного набора.

5.5 Чувствительность и кондиционирование

Как и предупреждали, мы обрисовали в общих чертах только несколько методов собственных значений из богатой и давней литературы. Для нахождения спектров были опробованы почти любые алгоритмические методы, от итерационных методов до поиска корней характеристического многочлена и методов, которые делят матрицы на блоки для параллельной обработки.

Как и в линейных решателях, мы можем оценить обусловленность проблемы собственных значений независимо от метода решения. Этот анализ может помочь понять, будет ли успешной упрощенная итерационная схема для нахождения собственных векторов данной матрицы или же необходимы более сложные методы; важно отметить, что обусловленность задачи на собственные значения — это не то же самое, что число обусловленности матрицы для решения систем, поскольку это отдельные зад

Предположим, что матрица A имеет собственный вектор x с собственным значением λ. Анализ обусловленности проблемы собственных значений включает анализ устойчивости x и λ к возмущениям в A. С этой целью мы могли бы возмущать A малой матрицей δA, тем самым изменяя набор собственных векторов. В частности, мы можем записать собственные векторы A + δA как возмущения собственных векторов A, решив задачу

$$(A + \delta A)(x + \delta x) = (\lambda + \delta\lambda)(x + \delta x).$$

Расширение обеих сторон дает:

$$Ax + A\delta x + \delta A \cdot x + \delta A \cdot \delta x = \lambda x + \lambda \delta x + \delta \lambda \cdot x + \delta \lambda \cdot \delta x$$

Предполагая δA малым, будем считать[3], что δx и δλ также малы. Произведения между этими переменными тогда незначительны, что дает следующее приближение:

$$Ax + A\delta x + \delta A \cdot x \quad \lambda x + \lambda \delta x + \delta \lambda \cdot x$$

Поскольку Ax = λx, мы можем вычесть это значение с обеих сторон, чтобы найти:

$$A\delta x + \delta A \cdot x \quad \lambda \delta x + \delta \lambda \cdot x$$

Теперь применим аналитический прием, чтобы завершить вывод. Поскольку Ax = λx, мы знаем, что (A λIn×n)x = 0, поэтому A λIn×n не имеет полного ранга. Транспонирование матрицы имеет полный ранг, только если матрица имеет полный ранг, поэтому мы знаем, что (A λIn×n) = A λIn×n также имеет нулевой пространственный вектор y. Таким образом, Ay = λy; мы можем назвать y левым собственным вектором, соответствующим x. Мы можем умножить вышеприведенную оценку возмущения слева на y :

$$y (A\delta x + \delta A \cdot x) \quad y (\lambda \delta x + \delta \lambda \cdot x)$$

Поскольку A y = λy, мы можем упростить:

$$y \, \delta A \cdot x \quad \delta \lambda y \, x$$

Перегруппировка выходов:

$$\frac{y \, (\delta A)x \, \delta \lambda}{yx}$$

Предположим, что x = 1 и y = 1. Тогда, если мы возьмем нормы с обеих сторон, мы найдем:

$$|\delta \lambda| \quad \frac{\delta A2 \, |}{y \cdot x|}$$

Таким образом, в целом условие задачи на собственные значения зависит от размера возмущения δA, как и ожидалось, и угла между левым и правым собственными векторами x и y. Мы можем использовать 1/x·y как приблизительное число обусловленности. Обратите внимание, что x = y, когда A симметричен, что дает число условия 1; это отражает тот факт, что собственные векторы симметричных матриц ортогональны и, следовательно, максимально разделены.

5.6 Проблемы

3Это предположение следует проверить более строго!

Глава 6

Разложение по сингулярным значениям

В главе 5 мы вывели ряд алгоритмов для вычисления собственных значений и собственных векторов матриц A Rn×n . Разработав этот механизм, мы завершаем наше первоначальное обсуждение числовой линейной алгебры выводом и использованием одной окончательной матричной факторизации, которая существует для любой матрицы A Rm×n : разложения по сингулярным числам (SVD).

6.1 Получение SVD

Для A Rm×n , мы можем думать о функции x Ax как об отображении, переводящем точки в Rn в точки в Rm.
С этой точки зрения мы могли бы задаться вопросом, что происходит с геометрией Rn в процессе, и, в частности, с влиянием A на длины и углы между векторами.

Применяя нашу обычную отправную точку для проблем с собственными значениями, мы можем задать вопрос о влиянии A на длины векторов путем изучения критических точек отношения

$$P(x) = \frac{\text{Топор}}{\text{Икс}}$$

при различных значениях x. Масштабирование x не имеет значения, так как

$$R(\alpha x) = \frac{A \cdot ax}{ax} \text{ "=" } \frac{|}{\alpha|} \cdot \frac{\text{Топор}}{\text{Икс}} \text{ "=" } \frac{\text{Топор}}{\text{Икс}} = P(x).$$

|α| Таким образом, мы можем ограничить наш поиск до x с x = 1. Более того, поскольку R(x) 0, вместо этого мы можем рассматривать [R(x)]2 = Ax = x A Ax. Однако, как мы показали в предыдущих главах, критические точки xAx при условии x = 1 — это в точности собственные векторы xi , удовлетворяющие условию A Axi = λixi ; заметим, что λi 0 и xi · xj = 0, когда i = j, так как AA симметрична и положительно полуопределена.

Основываясь на нашем использовании функции R, базис {xi} является разумным для изучения геометрических эффектов A. Возвращаясь к этой первоначальной цели, определим yi Axi . Мы можем сделать дополнительное наблюдение о yi , раскрывающее еще более сильную структуру собственных значений:

$$\begin{aligned}
\lambda_i y_i &= \lambda_i \cdot Ax_i \text{ по определению } y_i \\
&= A(\lambda_i x_i) \\
&= A(A\,Ax_i) \text{ , так как } x_i \text{ является собственным вектором } AA \\
&= (AA)(Ax_i) \text{ по ассоциативности} \\
&= (AA)\,y_i
\end{aligned}$$

Таким образом, имеем два случая:

1. Когда λi = 0, то yi = 0. В этом случае xi является собственным вектором AA и yi = Axi является AAxi = $\overline{\overline{\quad}}^{2}$ = x я соответствующий собственный вектор AA с yi = Axi = Axi λixi.

2. При λi = 0 yi =0.

Идентичное доказательство показывает, что если y — собственный вектор AA, то x A y — либо нуль, либо собственный вектор AA с тем же собственным значением.

Возьмем за k число строго положительных собственных значений λi > 0, обсуждавшихся выше. По нашей конструкции выше мы можем взять x1, . . . ,xk Rn — собственные векторы AA и соответствующие им собственные векторы y1, . . . ,yk Rm группы AA такие, что

$$\text{Акси} = \lambda_i x_i$$

$$AAy_i = \lambda_i y_i$$

для собственных значений λi > 0; здесь мы нормализуем так, что xi = yi = 1 для всех i. Следуя традиционным обозначениям, мы можем определить матрицы $\overline{V}$ Rn×k и $\overline{U}$ Rm×k , столбцами которых являются xi и yi соответственно.

Мы можем изучить влияние этих новых базисных матриц на A. Возьмем ei за i-й стандартный базисный вектор. Затем,

$$\begin{aligned}
\overline{U}A\overline{V}e_i &= \overline{U}Ax_i \text{ по определению } \overline{V} \\
&= \frac{1}{\overline{\lambda_i}}\overline{U}A(\lambda_i x_i) \text{ , так как мы предполагали } \lambda_i > 0 \\
&= \frac{1}{\overline{\lambda_i}}\overline{U}A(A\,Ax_i) \text{ , так как } x_i \text{ является собственным вектором } AA \\
&= \frac{1}{\overline{\lambda_i}}\overline{U}(AA)Ax_i \text{ по ассоциативности } \lambda_i\ 1 \\
&= \frac{\overline{\overline{\quad}}}{\overline{\lambda_i}}\overline{U}(AA)y_i \text{ , поскольку мы перемасштабировали так, что } y_i = \overline{\lambda_i}\overline{U}y_i \text{ , поскольку } AAy_i = \lambda_i y_i \\
&= \lambda_i e_i
\end{aligned}$$

Возьмем $\overline{\Sigma}$ = diag($\overline{\lambda_1}$, ..., $\overline{\lambda_k}$). Тогда приведенный выше вывод показывает, что $\overline{U}A\overline{V} = \overline{\Sigma}$.

Дополните столбцы $\overline{U}$ и $\overline{V}$ до U Rm×m и V Rn×n , добавив ортонормированные векторы xi и yi с Axi =0 и AAyi =0 соответственно. В этом случае легко показать UAVei =0 и/или UAV =0 . Таким образом, если мы возьмем

$$\Sigma_{ij} \quad \begin{matrix} \overline{\lambda_i} & i = j \text{ и } i \quad k \text{ в} \\ 0 & \text{противном случае} \end{matrix}$$

тогда мы можем расширить наше предыдущее соотношение, чтобы показать UAV = Σ или, что то же самое,

$$A = U\Sigma V .$$

Эта факторизация является в точности разложением по сингулярным значениям (SVD) A. Столбцы U охватывают пространство столбцов A и называются его левыми сингулярными векторами; столбцы V охватывают его пространство строк и являются правыми сингулярными векторами. Диагональные элементы σi множества Σ являются сингулярными значениями A; обычно они сортируются так, что σ1 σ2 · · · 0. И U, и V — ортогональные матрицы.

SVD обеспечивает полную геометрическую характеристику действия A. Поскольку U и V ортогональны, их можно рассматривать как матрицы вращения; как диагональная матрица Σ просто масштабирует отдельные координаты. Таким образом, все матрицы A Rm×n являются композицией поворота, масштаба и второго поворота.

6.1.1 Вычисление SVD

Напомним, что столбцы V просто являются собственными векторами AA, поэтому их можно вычислить с помощью методов, обсуждавшихся в предыдущей главе. Поскольку A = UΣ V, мы знаем, что AV = UΣ . Таким образом, столбцы U, соответствующие ненулевым сингулярным числам в Σ , просто являются нормированными столбцами AV; остальные столбцы удовлетворяют AAui = 0, что можно решить с помощью факторизации LU.

Эта стратегия ни в коем случае не является самым эффективным или стабильным подходом к вычислению SVD, но она достаточно хорошо работает для многих приложений. Мы опустим более специализированные подходы к нахождению SVD, но отметим, что многие из них являются простыми расширениями степенной итерации и других уже рассмотренных нами стратегий, которые работают без явного формирования AA или AA.

6.2 Применение СВД

Оставшуюся часть этой главы мы посвящаем знакомству со многими приложениями SVD. SVD появляется бесчисленное количество раз как в теории, так и в практике численной линейной алгебры, и ее важность трудно переоценить.

6.2.1 Решение линейных систем и псевдообратных систем

В частном случае, когда A Rn×n является квадратным и обратимым, важно отметить, что SVD можно использовать для решения линейной задачи Ax =b. В частности, имеем UΣ V x =b, или

$$x = V\Sigma \quad 1U\ б.$$

В этом случае Σ является квадратной диагональной матрицей, поэтому ^{1} просто матрица, диагональные элементы которой Σ 1/σi .

Вычисление SVD намного дороже, чем большинство методов линейного решения, которые мы представили в главе 2, поэтому это первоначальное наблюдение представляет в основном теоретический интерес. В более общем случае предположим, что мы хотим найти решение задачи Ax b методом наименьших квадратов , где A Rm×n не обязательно квадратное. Из нашего обсуждения нормальных уравнений мы знаем, что x должно удовлетворять A Ax = A b.
До сих пор мы в основном игнорировали случай, когда A является «коротким» или «недоопределенным», то есть когда A имеет больше столбцов, чем строк. В этом случае решение нормальных уравнений неединственно.

Чтобы охватить все три случая, мы можем решить задачу оптимизации следующего вида:

$$\text{минимизировать } x \quad ^{2}$$
$$\text{такое, что } A\ Ax = A\ b$$

Другими словами, эта оптимизация требует, чтобы x удовлетворял нормальным уравнениям с наименьшей возможной нормой.

Теперь давайте напишем A = UΣ V . Затем,

$$\begin{aligned} AA &= (U\Sigma V)(U\Sigma V) \\ &= V\Sigma U U\Sigma V, \text{ так как } (AB) = BA \\ &= V\Sigma \Sigma V, \text{ так как U ортогонален} \end{aligned}$$

Таким образом, спрашивать, что A Ax = A b, то же самое, что спрашивать

$$V\Sigma \Sigma V x = V\Sigma U b$$

Или, что то же самое, Σ y = d

если мы возьмем d Ub и y V x. Обратите внимание, что y = x, поскольку U ортогонален, поэтому наша оптимизация принимает вид:

$$\begin{aligned} &\text{минимизировать } \|y\|_2 \\ &\text{y так, чтобы } \Sigma y = d \end{aligned}$$

Однако, поскольку Σ является диагональным, условие Σ y = d просто утверждает σiyi = di ; поэтому всякий раз, когда σi = 0, мы должны иметь yi = di/σi . Когда σi = 0, ограничений на yi нет , поэтому, поскольку мы минимизируем y, мы могли бы также взять yi = 0. Другими словами, решением этой оптимизации является y = Σ + d, где Σ + Rn×m имеет следующий вид:

$$\Sigma^{+}_{ij} \quad \begin{cases} 1/\sigma i & i = j, \sigma i = 0 \text{ и } i \quad k \\ 0 & \text{в противном случае} \end{cases}$$

Эта форма, в свою очередь, дает x = Vy = VΣ + d = VΣ + Ub.

С этой мотивацией мы делаем следующее определение:

Определение 6.1 (псевдообратное). Псевдообратным к A = UΣ V Rm×n является A+ VΣ +U Rn×m.

Наш вывод выше показывает, что псевдоинверсия A обладает следующими свойствами:

1 • Когда A является квадратным и обратимым, A = A .

- Когда A переопределено, A +b дает решение методом наименьших квадратов для Ax b.
- Когда A недоопределенно, A +b дает решение методом наименьших квадратов для Ax b с минимальным (евклидова) норма.

Таким образом, мы, наконец, можем объединить недоопределенный, полностью определенный и переопределенный случаи Ax b.

6.2.2 Разложение на внешние произведения и аппроксимации низкого ранга

Если мы разложим произведение A = UΣ V , легко показать, что из этого соотношения следует:

$$A = \sum_{i=1} \sigma i u i v i ,$$

где min{m, n}, a ui и vi — i-е столбцы U и V соответственно. Наша сумма достигает только min{m, n}, так как мы знаем, что оставшиеся столбцы U или V будут обнулены с помощью Σ .

Это выражение показывает, что любую матрицу можно разложить как сумму внешних произведений векторы:

Определение 6.2 (Внешнее произведение). Внешнее произведение u Rm и v Rn есть матрица u v uv Rm×n .

Предположим, мы хотим написать произведение Ax. Тогда вместо этого мы могли бы написать:

$$Ax = \sum_{i=1} \sigma_i u_i v_i^T x$$

$$= \sum_{i=1} \sigma_i u_i (v_i^T x)$$

$$= \sum_{i=1} \sigma_i (v_i \cdot x) u_i, \text{ поскольку } x \cdot y = x^T y$$

Таким образом, применение A к x равносильно линейному объединению векторов ui с весами σi(vi · x). Эта стратегия вычисления Ax может обеспечить значительную экономию, когда количество ненулевых значений σi относительно невелико. Кроме того, мы можем игнорировать малые значения σi , эффективно усекая эту сумму для аппроксимации Ax с меньшими затратами.

Точно так же из §6.2.1 мы можем записать псевдоинверсию A как:

$$A^+ = \sum_{\sigma_i = 0} \frac{v_i u_i^T}{\sigma_i}.$$

Очевидно, мы можем применить тот же прием для вычисления A +x, и фактически мы можем аппроксимировать A +x, оценивая только те члены в сумме, для которых σi относительно мало. На практике мы вычисляем сингулярные значения σi как квадратные корни из собственных значений AA или AA, и такие методы, как степенная итерация, могут использоваться для выявления частичного, а не полного набора собственных значений. Таким образом, если нам нужно будет решить ряд задач наименьших квадратов Axi bi для различных bi и нас удовлетворит аппроксимация xi , может быть полезно сначала вычислить наименьшие значения σi и использовать приведенную выше аппроксимацию. Эта стратегия также избавляет от необходимости вычислять или сохранять полную матрицу A + и может быть точной, когда A имеет широкий диапазон сингулярных значений.

Возвращаясь к нашему исходному обозначению A = UΣ V,наши рассуждения выше эффективно показывают, что потенциально полезным приближением A является A˜ UΣ ˜ V, где Σ ˜ округляет малые значения Σ до нуля. Легко проверить, что пространство-столбец A˜ имеет размерность, равную количеству ненулевых значений на диагонали ˜ Σ . На самом деле, это приближение не является специальной оценкой, а скорее решает сложную задачу оптимизации с помощью следующей известной теоремы (изложена без доказательства):

Теорема 6.1 (Экарт-Янг, 1936). Предположим, что A получается из A = UΣ V путем усечения всех, кроме A A˜ Fro и A A˜ 2 , значений σi множества A, до нуля. Тогда A минимизирует оба ограничения, зависящих от k наибольших сингулярных заключающиеся в том, что пространство столбцов A имеет размерность не более k.

6.2.3 Нормы матрицы

Построение SVD также позволяет нам вернуться к нашему обсуждению матричных норм из §3.3.1. Например, вспомните, что мы определили норму Фробениуса для A как

$$A_{\text{Фро}}^2 = \sum_{ij} \text{часа}_{ij}^2.$$

Если мы напишем $A = U\Sigma V$, мы можем упростить это выражение:

$$\begin{aligned}
A_{\text{Фро}}^2 &= \sum_{j} \text{Эй}_j^2 \quad \text{так как это произведение является j-м столбцом матрицы A} \\
&= \sum_{j} U\Sigma V e_j^2, \quad \text{подставив СВД} \\
&= \sum_{j} e_j V\Sigma 2V e_j, \quad \text{так как } x^2 = xx \text{ и U ортогонален} \\
&= \Sigma V^2 \quad \text{Отсюда по той же логике} \\
&= V\Sigma^2_{\text{Фро}} \quad \text{поскольку матрица и ее транспонирование имеют одну и ту же норму Фробениуса} \\
&= \sum_{j} V\Sigma e_j^2 = \sum_{j} \sigma_j^2 \text{Дж Вей}^2 \quad \text{по диагонали } \Sigma \\
&= \sum_{j} \sigma_j^2 \quad \text{поскольку V ортогонален}
\end{aligned}$$

Таким образом, норма Фробениуса множества A Rm×n есть сумма квадратов его сингулярных значений.

Этот результат представляет теоретический интерес, но на практике основное определение нормы Фробениуса уже несложно оценить. Что еще интереснее, напомним, что индуцированная двойная норма A задается выражением

$$A_2^2 = \max\{\lambda : \text{существует } x \quad R^n \text{ с } A Ax = \lambda x\}.$$

Теперь, когда мы изучили проблемы с собственными значениями, мы понимаем, что это значение представляет собой квадратный корень из наибольшего собственного значения AA или, что то же самое,

$$A_2 = \max\{\sigma_i\}.$$

Другими словами, мы можем прочитать двойную норму A непосредственно из его собственных значений.

Точно так же напомним, что число обусловленности A определяется выражением cond $A = A_2 A^{-1}_2$. По нашему вывод A^+, сингулярные значения A^{-1} должны быть величины, обратные сингулярным числам A. В сочетании с нашим упрощением A_2 получается:

$$\text{усл } A = \frac{\sigma_{\max}}{\sigma_{\min}}.$$

Это выражение дает стратегию для оценки обусловленности A. Конечно, вычисление $\sigma_{\min}$ требует решения систем $Ax = b$, процесс, который сам по себе может страдать от плохой обусловленности A; если это проблема, то обусловливание можно ограничить и аппроксимировать, используя различные аппроксимации сингулярных значений A.

6.2.4 Проблема Прокруста и выравнивание

Многие методы компьютерного зрения включают выравнивание трехмерных фигур. Например, предположим, что у нас есть трехмерный сканер, который собирает два облака точек одного и того же жесткого объекта с разных точек зрения. Типичной задачей может быть совмещение этих двух облаков точек в единую систему координат.

Поскольку объект является жестким, мы ожидаем, что существуют некоторая матрица поворота R и перемещение t R3 , такие что вращение первого облака точек на R и последующее перемещение на байт выравнивает два набора данных. Наша работа состоит в том, чтобы оценить эти два объекта.

Если два сканирования перекрываются, пользователь или автоматизированная система могут отметить n соответствующих точек, которые соответствуют между двумя сканированиями; мы можем хранить их в двух матрицах X1, X2 R3×n . Тогда для каждого столбца x1i из X1 и x2i из X2 мы ожидаем Rx1i +t = x2i . Мы можем написать функцию энергии, измеряющую, насколько верно это соотношение:

$$E \quad \underset{я}{} \quad Rx1i + t \quad x2i \quad ^{2}.$$

Если мы зафиксируем R и минимизируем по tot, оптимизация E, очевидно, станет задачей наименьших квадратов. Теперь предположим, что мы оптимизируем для R с фиксированным значением. Это то же $^{т}_{2\,\text{Фронт}}$, самое, что минимизировать $^{т}_{2}$ - это X2, переведенные на байт, при условии, что R является матрицей вращения 3 × 3, RX1 X, где столбцы X, то есть RR = I3×3. Это известно как ортогональная проблема Прокруста.

Для решения этой задачи введем след квадратной матрицы следующим образом:

Определение 6.3 (След). След A Rn×n есть сумма его диагоналей:

$$\text{тр}(A) \quad \underset{я}{} \quad \text{аии}.$$

Несложно проверить, что A $^{2}_{\text{Фро}}$ = tr(AA). Таким образом, мы можем упростить E следующим образом:

$$\begin{aligned} RX1 - X \quad ^{т}_{2} \quad ^{2}_{\text{Фро}} &= \text{tr}((RX1 \quad X \quad 2\dot{}) (RX1 \quad X\, 2^{т})) \\ &= \text{тр}(X\, 1 \; X1 - X \quad _{1} \; RX \quad ^{т}_{2\,2} \; X^{\,т} \; RX1 + X \quad ^{т}_{2\,x\,2)} \\ &= \text{константа} \quad 2\dot{t}\text{r}(X\, 2 \end{aligned}$$

$$RX1) \text{ , поскольку } \text{tr}(A + B) = \text{tr}\, A + \text{tr}\, B \text{ и } \text{tr}(A\,) = \text{tr}(A)$$

Таким образом, мы хотим максимизировать $^{т}_{2}$ tr(X RX1) с RR = I3×3. В упражнениях вы докажете, что tr(AB) = tr(BA). Таким образом, наша цель может быть немного упрощена до tr(RC) с C X1X $^{т}_{2}$. Применяя СВД, если разложить C = UΣ V, то можно упростить еще больше:

$$\begin{aligned} \text{tr}(RC) = \text{tr}(RU\Sigma\ V\,) \text{ по определению} &= \\ \text{tr}((V\ RU)\Sigma\ ), \text{ так как } \text{tr}(AB) = \text{tr}(BA) &= \text{tr}(\tilde{R}\ \Sigma\ ) \text{ ,} \\ \text{если определить } \tilde{R} = V\ RU, \text{ что также является ортогональным} & \\ = \quad \underset{я}{} \quad \sigma i\tilde{r}\ ii \text{ , так как } \Sigma \text{ диагональна} & \end{aligned}$$

Поскольку $\tilde{R}$ ортогонален, все его столбцы имеют единичную длину. Отсюда следует, что $\tilde{r}$ ii 1, иначе норма столбца i была бы слишком велика. Поскольку σi 0 для всех i, этот аргумент показывает, что мы можем максимизировать tr(RC) , взяв $\tilde{R}$ = I3×3. Отменив наши замены, мы получим R = VRU$\tilde{}$ = VU.

В более общем виде мы показали следующее:

Теорема 6.2 (ортогональный прокруст). Ортогональная матрица R минимизирует RX Y VU, где ^{2} дан кем-то SVD применяется к фактору XY = UΣ V .

Возвращаясь к проблеме выравнивания, одной из типичных стратегий является альтернативный подход:

1. Зафиксировать R и минимизировать E относительно tot.

2. Зафиксировать полученное t и минимизировать E по R при условии, что RR = I3×3.

3. Вернитесь к шагу 1.

Энергия E уменьшается с каждым шагом и, таким образом, сходится к локальному минимуму. Поскольку мы никогда не оптимизируем и R одновременно, мы не можем гарантировать, что результатом будет минимально возможное значение E, но на практике этот метод работает хорошо.

6.2.5 Анализ основных компонентов (АПК)

Вспомните настройку из §5.1.1: мы хотим найти низкоразмерную аппроксимацию набора точек данных, которые мы можем сохранить в матрице X Rn×k для k наблюдений в n измерениях. Ранее мы показали, что если нам разрешено только одно измерение, наилучшее возможное направление задается доминирующим собственным вектором XX.

Вместо этого предположим, что нам разрешено проектировать на диапазон d векторов с d min{k, n} , и мы хотим выбрать эти векторы оптимально. Мы могли бы записать их в матрице C размера n × d; поскольку мы можем применить Грамма-Шмидта к любому набору векторов, мы можем предположить, что столбцы C ортонормированы, показывая CC = Id×d . Поскольку C имеет ортонормированные столбцы, по нормальным уравнениям проекция X на пространство столбцов C задается CCX.

В этой установке мы хотим минимизировать X CCXFro при условии CC = Id×d . Мы можем упростить наша проблема несколько:

$$\begin{aligned} \|X - CCX\|_{\text{Фро}}^2 &= \text{tr}((X - CCX)(X - CCX)) \text{ поскольку } A\|_{\text{Фро}}^2 = \text{тр}(AA) \\ &= \text{tr}(X X - 2X CCX + X CCCCX) = \text{const.} \\ &\quad \text{tr}(X CCX) \text{, поскольку } CC = Id\times d \\ &= -\|CX\|_{\text{Фро}}^2 + \text{конст.} \end{aligned}$$

Таким образом, мы можем максимизировать ^{2}Фро; для статистиков это показывает, когда строки X имеют среднее значение CX, равное нулю, что мы хотим максимизировать дисперсию проекции CX.

Теперь предположим, что мы факторизуем X = UΣ V . Тогда мы хотим максимизировать C = C˜ Σ Fro = UΣ V Fro Σ ˜ CFro по ортогональности V, если мы возьмем C˜ = CU. Если элементами C˜ являются c˜ ij, то разложение этой нормы дает

$$\|\Sigma\ \tilde{C}\|_{\text{Фро}}^2 = \sum_{i} \sigma_i^2 \sum_{j} \tilde{c}_{ij}^2.$$

По ортогональности столбцов C˜ мы знаем, что $\sum_i \tilde{c}_{ij}^2 = 1$ для всех j, и, поскольку C˜ может иметь менее n столбцов, $\sum_j \sum_i \tilde{c}_{ij}^2 \le 1$. Таким образом, коэффициент при σ_i^2 в приведенной выше сумме, поэтому, если мы тогда ясно максимум достигается изд . Отменив нашу замену , путем сортировки столбцов C˜ таким образом, что σ1 σ2 · · · bee1, . . . , координат, мы видим, что наш выбор C должен быть первым d столбцами U.

Мы показали, что SVD для X можно использовать для решения такой задачи анализа основных компонентов (PCA). На практике строки X обычно сдвигаются так, чтобы иметь среднее значение, равное нулю, перед выполнением SVD; как показано на рис. НОМЕР, это центрирует набор данных относительно начала координат, предоставляя более значимые векторы PCA ui .

6.3 Проблемы

Часть 3

Нелинейные методы

Глава 7

Нелинейные системы

Как бы мы ни старались, просто невозможно выразить все системы уравнений в линейной структуре, которую мы разработали в последних нескольких главах. Едва ли нужно мотивировать использование логарифмов, экспонент, тригонометрических функций, модулей, многочленов и т. д. в практических задачах, но, за исключением нескольких частных случаев, ни одна из этих функций не является линейной. Когда эти функции появляются, мы должны использовать более общий, хотя и менее эффективный набор механизмов.

7.1 Задачи с одной переменной

Мы начнем наше обсуждение с рассмотрения задач с одной скалярной переменной. В частности, для заданной функции f(x) : R R мы хотим разработать стратегии для нахождения точек x R , таких что f(x) = 0; мы называем x корнем f . Задачи с одной переменной в линейной алгебре, в частности, не = b/a. Однако интересный; в конце концов, мы можем решить уравнение ax b = 0 в закрытой форме, поскольку x решение cos y нелинейное уравнение, такое 2 + e 3 = 0 гораздо менее очевидно (кстати, решение как y, равно y = ± 1,30246 . . .).

7.1.1 Характеристика проблем

Мы больше не можем предполагать, что функция f линейна, но без каких-либо предположений о ее структуре мы вряд ли продвинемся вперед в решении систем с одной переменной. Например, решатель гарантированно не найдет нули f (x), заданного выражением

$$e(x) = \begin{cases} 1 & x \neq 0 \\ 1 & x > 0 \end{cases}$$

Или хуже:

$$e(x) = \begin{cases} 1 & x \neq Q \\ 1 & \text{иначе} \end{cases}$$

Эти примеры тривиальны в том смысле, что разумный клиент программы для поиска корней вряд ли будет ожидать успеха в этом случае, но гораздо менее очевидные случаи не намного сложнее.
строить.

По этой причине мы должны добавить некоторые «упорядочивающие» предположения о том, что f обеспечивает точку опоры для возможности разработки методов поиска корней. Ниже приведены типичные такие предположения, перечисленные в порядке возрастания силы:

• Непрерывность: функция f является непрерывной, если ее можно нарисовать, не поднимая пера; более формально f непрерывна, если разность f(x) f(y) обращается в нуль при x y.

• Липшиц: функция f является липшицевой, если существует константа C такая, что | e (x) - e (y) | C |x y|; Липшицевы функции не обязательно должны быть дифференцируемыми, но их скорость изменения ограничена.

• Дифференцируемость: функция f является дифференцируемой, если ее производная f существует для всех x.

производные от f существуют и непрерывны.

k раз и каждая из этих k производных непрерывна; • C : Функция C означает, что все C

По мере того как мы будем добавлять все более и более сильные предположения о f, мы сможем разработать более эффективные алгоритмы для решения f(x) = 0. Мы проиллюстрируем этот эффект, рассмотрев несколько алгоритмов ниже.

7.1.2 Непрерывность и бисекция

Предположим, что все, что мы знаем о f, это то, что оно непрерывно. В этом случае мы можем сформулировать интуитивную теорему из стандартного исчисления с одной переменной:

Теорема 7.1 (теорема о промежуточном значении). Предположим, что f : [a, b] R непрерывна. Предположим, что f(x) < u < f(y). Тогда существует z между x и y такое, что f(z) = u.

Другими словами, функция f должна достигать всех значений между f(x) и f(y).

Предположим, что нам дана на вход функция f, а также два значения и r такие, что f() · f(r) < 0; обратите внимание, что это означает, что f() и f(r) имеют противоположный знак. Тогда по теореме о промежуточном значении мы знаем, что где-то между и r есть корень f! Это обеспечивает очевидную стратегию деления пополам для нахождения x

*:

1. Вычислить c = +r/2.
2. Если f(c) = 0, вернуть x * = c.
3. Если f() · f(c) < 0, взять r c. В противном случае возьмите c.
4. Если |r | < ε, вернуть x * c.
5. Вернитесь к шагу 1

Эта стратегия просто итеративно делит интервал [,r] пополам, каждый раз сохраняя сторону, в которой, как известно, существует корень. Ясно, что по теореме о промежуточном значении он сходится безоговорочно, в том смысле, что до тех пор, пока f() · f(r) < 0, в конечном счете и r, и r гарантированно сходятся к допустимому корню x

*.

7.1.3 Анализ поиска корней

Разделение пополам — самый простой, но не обязательно самый эффективный метод поиска корней. Как и в большинстве методов собственных значений, деление пополам по своей сути является итеративным и может никогда не дать точного решения. Однако мы можем спросить, насколько близко значение ck функции c в k-й итерации к корню x *, который мы надеемся вычислить. Этот анализ обеспечит основу для сравнения с другими методами.

В общем, предположим, что мы можем установить границу ошибки Ek, такую, что оценка xk корня во время k-й итерации метода нахождения корня удовлетворяет условию |xk x | < Эк . Очевидно, что любой алгоритм с Ek 0 представляет собой сходящуюся схему; скорость сходимости, однако, может быть охарактеризована скоростью, с которой Ek приближается к 0. находятся в интервале [k ,rk], верхняя граница Например, в делении пополам, поскольку ошибка как

делим интервал пополам на каждой итерации, мы знаем, что Ek+1 ck, так и x определяется выражением Ek | рк - к |. Поскольку мы = 1/2Ek . Поскольку Ek+1 линейно в Ek , мы говорим, что деление пополам обладает линейной сходимостью.

7.1.4 Итерация с фиксированной точкой

Разделение пополам гарантированно сходится к корню для любой непрерывной функции f , но если мы узнаем больше о f, мы сможем сформулировать алгоритмы, которые сходятся быстрее.

Например, предположим, что мы хотим найти x такое, что g(x) = x ; конечно, эта установка эквивалентна задаче поиска корня, так как решение f(x) = 0 совпадает с решением f(x) + x = x.
Однако в качестве дополнительной информации мы также можем знать, что g является липшицевым с константой C < 1.

Система g(x) = x предлагает потенциальную стратегию, которую мы могли бы предположить:

1. Возьмем x0 за начальное предположение корня.

2. Повторить xk = g(xk 1).

Если эта стратегия сходится, ясно, что результатом является неподвижная точка g, удовлетворяющая вышеуказанным критериям.

К счастью, свойство Липшица гарантирует, что эта стратегия сходится к корню, если он существует.
Если взять Ek = |xk x |, то мы имеем следующее свойство:

$$
\begin{aligned}
Ek = |xk \quad x \\
&| = |g(xk \quad 1) \quad g(x \quad)| \text{ по построению итерационной схемы и определению } x \\
&C|xk \quad 1 \quad x \quad | \text{ так как g липшицева} \\
&= CEk \quad 1
\end{aligned}
$$

Индуктивное применение этого утверждения показывает, что Ek C K |E0| 0 при k . Таким образом, итерация с фиксированной точкой сходится к желаемому x !

В самом деле, если g липшицева с константой C < 1 в окрестности [x δ, x + δ], то до тех пор, пока x0 выбран в этом интервале, итерация с фиксированной точкой будет сходиться. Это верно, поскольку приведенное выше выражение для Ek показывает, что оно сжимается с каждой итерацией. и |g (x)|

Один важный случай имеет место, когда g есть C, 1 < 1. В силу непрерывности g в этом случае мы + δ] , в котором |g (x)| < 1 ε известно, что существует некоторая окрестность N = [x δ, x для для любого x N,
некоторого выбора достаточно малого ε > 0.1 . Возьмем любые x, y N. Тогда имеем

$$
\begin{aligned}
|r(x) - r(y)| = |r(\theta)| \cdot |x \quad y| \text{ по теореме о среднем значении основного исчисления для некоторого } \theta \quad [x, \\
y] < (1 \quad \varepsilon)|x \quad y|
\end{aligned}
$$

Это показывает, что g является липшицевым с константой 1 ε < 1 в N. Таким образом, когда g непрерывно дифференцируема и g (x) < 1, итерация с фиксированной точкой будет сходиться к x , когда начальное приближение x0 близко.

1Это утверждение трудно разобрать: убедитесь, что вы его понимаете!

Пока что у нас мало причин использовать итерацию с фиксированной точкой: мы показали, что она гарантированно сходится только тогда, когда g является липшицевой, а наши рассуждения о Ek показывают линейную сходимость, подобную делению пополам. Однако есть один случай, когда итерация с фиксированной точкой дает преимущество.

Предположим, что g дифференцируема с g (x) = 0. Тогда член первого порядка исчезает в ряду Тейлора для g, оставляя после себя:

$$1\ g(xk) = g(x\ ) + g\ (x\ )(xk\ \ x\ )\ 2\ \ ^{2} + O\ (xк - x\ *)\ \ ^{3}\ .$$

Таким образом, в данном случае имеем:

$$\begin{aligned} Ek &= |xk\ \ x\ \ | \\ &= |g(xk\ \ 1)\ \ g(x\ \ )|\ \text{как прежде} \\ &\text{"="}\ \ \tfrac{1}{2}\ |g\ (x\ \ )|(xk\ \ 1\ \ x\ \ )^{2} + O((xk\ \ 1\ \ x\ \ )^{3})\ \text{из аргумента Тейлора} \\ &\text{"="}\ \ \tfrac{1}{2}\ (|g\ (x\ \ )| + \varepsilon)|(xk\ \ 1\ \ x\ \ )^{2}\ \text{для некоторого } \varepsilon\text{, пока } xk\ \ 1 \text{ близко к } x^{*} \\ &\text{"="}\ \ \tfrac{1}{2}\ (|g\ (x\ \ )| + \varepsilon)E\ _{k\ \ 1}^{2} \end{aligned}$$

Таким образом, в этом случае Ek является квадратичным по Ek 1 , поэтому мы говорим, что итерация с фиксированной точкой может иметь квадратичную сходимость; обратите внимание, что это доказательство квадратичной сходимости верно только потому, что мы уже знаем Ek 0 из нашего более общего доказательства сходимости. Это означает, что Ek 0 намного быстрее, поэтому нам потребуется меньше итераций, чтобы получить разумный корень.

Пример 7.1 (Сходимость итерации по фиксированной точке).

7.1.5 Метод Ньютона

Мы еще раз сузим наш класс функций, чтобы получить метод, который имеет более последовательную квадратичную сходимость. Теперь предположим, что мы снова хотим решить f(x) = 0, но теперь мы предполагаем, что f является ^{1}, a условием C, несколько более жестким, чем условие Липшица.

В точке xk R, поскольку f теперь дифференцируема, мы можем аппроксимировать ее касательной: f(x)

$$f(xk) + f\ (xk)(x\ \ xk)$$

Решение этого приближения для f(x) 0 дает корень f(xk) f (xk)

$$xк+1 = xк - \ \ \frac{}{}\ .$$

Повторение этой формулы известно как метод Ньютона для поиска корней, и он сводится к итеративному решению линейной аппроксимации нелинейной задачи.

Обратите внимание, что если мы определим

$$(x) = x - f\ (x)\ \ \frac{f\ (x)\ г}{},$$

тогда метод Ньютона сводится к итерации с фиксированной точкой на g. Дифференцируя, находим:

$$(x) = 1 - \frac{ж\ (x)\ г^{2}\ f(x)f\ (x)\ по}{2}\ \text{фактор-правилу}\ f\ (x)\ f(x)f\ (x)$$

$$\text{"="}\ \ \frac{}{ф\ (x)^{2}}$$

итерация с * является простым корнем, что означает, что f (x *) = 0. Тогда g (x *) = 0, и из нашего вывода предположим, что x фиксированной точкой выше, мы знаем, что метод Ньютона сходится квадратично к x * для достаточно близкого первоначальная догадка. Таким образом, когда f дифференцируемо с простым корнем, метод Ньютона обеспечивает формулу итерации с фиксированной точкой, которая гарантированно сходится квадратично; однако, когда x не простое, сходимость может быть * линейной или хуже.

Вывод метода Ньютона предполагает другие методы, полученные с использованием большего количества членов в ряду Тейлора. Например, «метод Галлея» добавляет члены, включающие f, в итерации, а класс «методов Хаусхолдера» принимает произвольное число производных. Эти методы обеспечивают сходимость еще более высокого порядка за счет необходимости оценивать более сложные итерации и возможность более экзотических режимов отказа. Другие методы заменяют ряд Тейлора другими основными формами; например, линейно-дробная интерполяция использует рациональные функции для лучшей аппроксимации функций с асимптотной структурой.

7.1.6 Метод секущих

Одна проблема эффективности, которую мы еще не рассмотрели, — это стоимость оценки f и ее производных.
Если f — очень сложная функция, мы можем захотеть свести к минимуму количество вычислений f или, что еще хуже, f. Более высокие порядки сходимости помогают решить эту проблему, но мы также можем разработать численные методы, которые позволяют избежать оценки дорогостоящих производных.

Пример 7.2 (Дизайн). Предположим, мы проектируем ракету и хотим знать, сколько топлива добавить в двигатель. Для данного количества галлонов x мы можем написать функцию f (x) , определяющую максимальную высоту ракеты; наши инженеры уточнили, что мы хотим, чтобы ракета достигла высоты h, поэтому нам нужно решить f(x) = h. Вычисление f(x) включает в себя моделирование взлета ракеты и отслеживание расхода топлива, что является дорогостоящим мероприятием, и хотя мы можем подозревать, что f дифференцируема, мы не сможем оценить f за какое-то практическое время.

Одной из стратегий разработки методов с меньшим воздействием является максимально возможное повторное использование данных. Для например, мы легко могли бы аппроксимировать:

$$\frac{f(x_k) \quad f(x_k \quad 1)}{x_k - x_{k-1}} f(x_k).$$

То есть, поскольку нам нужно было вычислить f(xk 1) на предыдущей итерации, мы просто используем наклон f(xk) для аппроксимации производной. Конечно, это приближение работает хорошо, особенно когда xk близки к сходимости.

Подключив наше приближение к методу Ньютона, мы получим новую итеративную схему:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k \quad x_k \quad 1)}{f(x_k) \quad f(x_k \quad 1)}$$

Обратите внимание, что пользователь должен будет предоставить два начальных предположения x0 и x-1 , чтобы запустить эту схему, или может запустить одну итерацию Newton, чтобы запустить ее.

Анализ метода секущих несколько сложнее, чем другие методы, которые мы рассматриваем, поскольку он использует как f(xk) , так и f(xk 1); доказательство его сходимости выходит за рамки нашего обсуждения. Интересно, что анализ ошибок показывает, что ошибка уменьшается со скоростью 1+ 5/2 («золотое сечение») между линейной и квадратичной; поскольку сходимость близка к сходимости метода Ньютона без необходимости вычисления f , метод секущих может обеспечить сильную альтернативу.

7.1.7 Гибридные методы

Можно выполнить дополнительную разработку, чтобы попытаться объединить преимущества различных алгоритмов поиска корней. Например, мы могли бы сделать следующие замечания о двух рассмотренных нами методах:

- Гарантируется безусловная сходимость пополам, но только с линейной скоростью.
- Метод секущих сходится быстрее, когда достигает корня, но в некоторых случаях может не сходится. сходятся.

Предположим, мы заключили в скобки корень f(x) в интервале [k ,rk] , как при делении пополам. Мы можем сказать, Если наша текущая оценка x * определяется выражением xk = к что когда | ϕ (к) | < | f(rk)| и xk = rk в противном случае. мы отслеживаем xk и xk 1 , то мы могли бы взять xk+1 как следующую оценку корня, полученную методом секущих. Однако если xk находится вне интервала [k ,rk], мы можем заменить его на k+rk/2. Это исправление гарантирует, что xk+1 [k ,rk], и независимо от выбора мы можем обновить до допустимой скобки [k+1 ,rk+1] , как при делении пополам, проверив знак f(xk+1). Этот алгоритм известен как «метод Деккера».

Стратегия, описанная выше, пытается объединить безусловную сходимость деления пополам с более сильными корневыми оценками метода секущих. Во многих случаях он успешен, но скорость его сходимости довольно трудно анализировать; специализированные режимы отказа могут свести этот метод к линейной сходимости или к худшему — фактически в некоторых случаях деление пополам неожиданно может сходиться быстрее! Другие методы, например, «метод Брента», делают шаги пополам чаще, чтобы избежать этого случая, и могут демонстрировать гарантированное поведение за счет несколько более сложной реализации.

7.1.8 Случай с одной переменной: сводка

Теперь мы представили и проанализировали ряд методов решения f(x) = 0 в случае одной переменной. На данный момент, вероятно, очевидно, что мы коснулись только поверхности таких методов; существует много итерационных схем поиска корней, все с разными гарантиями, скоростями сходимости и предостережениями. Несмотря на это, на основе нашего опыта мы можем сделать ряд наблюдений:

- Из-за возможной общей формы f маловероятно, что мы сможем точно найти корни x , и вместо этого довольствуемся итерационными схемами.
- Мы хотим, чтобы последовательность xk корневых оценок достигла x* как можно быстрее. Если Ek является границей ошибки, то мы можем охарактеризовать ряд ситуаций сходимости, предполагая, что Ek 0 при k . Полный список условий, которые должны выполняться, когда k достаточно велико, приведен ниже:
  1. Линейная сходимость: Ek+1 CEk для некоторого C < 1 2.
  Суперлинейная сходимость: Ek+1 CEr для r > 1 (теперь мы не требуем C < 1, поскольку, если Ek достаточно к мало, степень r может сократить последствия C)
  3. Квадратичная сходимость: Ek+1 CE2 4. к
  Кубическая сходимость: Ek+1 CE3 к (и так далее)
- Метод может сходиться быстрее, но на каждой отдельной итерации требует дополнительных вычислений; по этой причине может быть предпочтительнее сделать больше итераций более простого метода, чем меньше итераций более сложного.

7.2 Многовариантные задачи

В некоторых приложениях может потребоваться решение более общей задачи f(x) =0 для функции f : Rn Rm. Мы уже видели один пример этой проблемы при решении Ax = b, что эквивалентно нахождению корней f (x) Ax b, но общий случай значительно сложнее. В частности, такие стратегии, как деление пополам, трудно расширять, поскольку теперь мы в значительной степени гарантируем, что все m различных значений равны нулю одновременно.

7.2.1 Метод Ньютона

К счастью, одна из наших стратегий расширяется очень просто. Напомним, что для f : Rn Rm мы можем записать матрицу Якоби, которая дает производную каждой компоненты f в каждом из координатных направлений:

$$(Df)_{ij} \quad dxj \ \frac{д\,фи}{}$$

Мы можем использовать якобиан f, чтобы расширить наш вывод метода Ньютона на несколько измерений.
В частности, приближение первого порядка f определяется выражением:

$$f(x) \quad f(xk) + Df(xk) \cdot (x \quad xk).$$

Замена желаемого f (x) = 0 дает следующую линейную систему для следующей итерации xk+1 :

$$D\,f(xk) \cdot (xk+1 \quad xk) = \quad f(xk)$$

Это уравнение можно решить, используя псевдообратное, когда m < n; когда m > n, можно попытаться применить метод наименьших квадратов, но существование корня и сходимость этого метода маловероятны. Однако, когда D f является квадратным, что соответствует методу f : Rn Rn : , мы получаем типичную итерацию для Ньютона

$$xk+1 = xk \quad [D\,f(xk)] \quad 1\,f(xk), \text{ где, как}$$

всегда, мы не вычисляем явно матрицу [D f(xk)] 1 , а используем ее для обозначения решения линейной системы.

Сходимость методов с фиксированной точкой, таких как метод Ньютона, который итерирует xk+1 = g(xk), требует, чтобы собственное значение максимальной величины якобиана Dg было меньше 1. После проверки этого предположения рассуждения, аналогичные одномерному случаю, показывают что метод Ньютона может , для квадратичную сходимость вблизи корней x которого D f(x) невырождена. имеют

7.2.2 Делаем Ньютона быстрее: Квазиньютон и Бройен

По мере увеличения m и n метод Ньютона становится очень дорогим. Для каждой итерации необходимо инвертировать другую матрицу D f(xk) ; поскольку он так часто меняется, предварительная факторизация D f(xk) = LkUk не помогает.

Некоторые квазиньютоновские стратегии пытаются применять разные стратегии аппроксимации для упрощения отдельных итераций. Например, один простой подход может повторно использовать D f из предыдущих итераций при повторном вычислении f(xk) в предположении, что производная не меняется очень быстро. Мы вернемся к этим стратегиям, когда будем обсуждать применение метода Ньютона к оптимизации.

Другой вариант — попытаться провести параллель с нашим выводом метода секущих. Так как метод секущих по-прежнему содержит деление, такие приближения не обязательно избавят от необходимости инвертировать матрицу, но они позволяют проводить оптимизацию без явного вычисления якобиана D f . Такие расширения не совсем очевидны, поскольку разделенные разности не дают полной приближенной матрицы Якоби.

Напомним, однако, что производная по направлению f в направлении v задается как Dv f = D f · v. Как и в случае с методом секущих, мы можем использовать это наблюдение в своих интересах, попросив, чтобы наша аппроксимация J якобиана удовлетворяла

$$J \cdot (x_k - x_{k-1}) = f(x_k) - f(x_{k-1}).$$

Метод Бройдена является одним из таких расширений метода секущих, который отслеживает не только оценку xk для x , но и матрицу Jk , оценивающую якобиан; должны быть предоставлены начальные оценки J0 и x0 . Предположим, у нас есть предыдущая оценка Jk 1 якобиана из предыдущей итерации. Теперь у нас есть новая точка данных xk , в которой мы оценили f(xk), поэтому мы хотели бы обновить Jk 1 до нового якобиана Jk с учетом этого нового наблюдения. Одна разумная модель состоит в том, чтобы попросить, чтобы новое приближение было похоже на старое приближение, за исключением направления xk xk 1 :

$$\text{минимизировать } \|J_k - J_{k-1}\|_{\text{Фро}}^2$$
$$J_k \text{ такое, что } J_k \cdot (x_k - x_{k-1}) = f(x_k) - f(x_{k-1})$$

Чтобы решить эту задачу, определим $J = J_k - J_{k-1}$ и $d = f(x_k) - f(x_{k-1}) - J_{k-1} \cdot x$. $x = x_k - x_{k-1}$ Выполнение этих замен дает следующую форму:

$$\text{минимизировать } \|J\|_{\text{Фро}}^2$$
$$J \text{ так, что } J \cdot x = d$$

Если мы возьмем λ как множитель Лагранжа, эта минимизация эквивалентна нахождению критических точек лагранжиана Λ:

$$\Lambda = \|J\|_{\text{Фро}}^2 + \lambda (J \cdot x - d)$$

Дифференцирование по (J)ij показывает:

$$0 = \frac{\partial \Lambda}{\partial} 1 = 2(J)_{ij} + \lambda_i (x)_j = J = \lambda (x) (\bar{J})_{ij} 2$$

Подстановка в J · x = d показывает λ(x) (x) = 2d или, что то же самое, λ = 2d/ x 2 . Наконец, мы можем заменить, чтобы найти:

$$J = \frac{1}{2} - \lambda (x) = x \frac{d(x)}{2}$$

Расширение нашей замены показывает:

$$J_k = J_{k-1} + J$$
$$= J_{k-1} + \frac{д(x)}{x\ 2}$$
$$= J_{k-1} + \frac{(f(x_k) - f(x_{k-1}) - J_{k-1} \cdot x)}{x_к - x_{к-1}^2} (x_k - x_{k-1})$$

Таким образом, метод Бройдена просто чередует это обновление с соответствующим ньютоновским f(xk). Дополнительную шаг xk+1 = xk J $_k^{-1}$ эффективность в некоторых случаях можно получить, отслеживая матрица J $_k^{-1}$ явно, а не матрица Jk , которую можно обновить по аналогичной формуле.

7.3 Кондиционирование

Мы уже показали в примере 1.7, что число условий нахождения корня в одной переменной:

$$\text{кондх} * \text{е знак} \frac{1}{\text{равно} \mid \text{е}(x^*) \mid}$$

Как показано на рисунке НОМЕР, этот номер условия показывает, что наилучшая возможная ситуация для нахождения корня возникает, когда f быстро меняется вблизи x, поскольку в этом случае возмущение x заставит f принимать значения, далекие от 0.

Применение идентичного аргумента, когда f многомерно, показывает число обусловленности D f (x) $^{-1}$. Обратите внимание, что когда D f необратима, число обусловленности бесконечно. Эта странность происходит из-за того, что возмущение x первого * сохраняет f(x) = 0, и такое условие действительно может быть выполнено. порядка создает сложные случаи поиска корня, подобные показанным на рисунке НОМЕР.

7.4 Проблемы

Множество возможностей, в том числе:

- Множество возможных схем итерации с фиксированной точкой для данной задачи поиска корня, графическая версия итерации с фиксированной точкой
- Средняя итерация поля в ML
- Метод Мюллера – комплексные корни
- Итерационные методы более высокого порядка – методы Хаусхолдера
- Интерпретация собственного материала как поиск корня
- Сходимость метода секущих
- Корни многочленов
- Метод Ньютона-Фурье (!)
- «Модифицированный метод Ньютона в случае неквадратичной сходимости»
- Конвергенция –¿ спектральный радиус для многомерного Ньютона; квадратичная сходимость
- Обновление Шермана-Моррисона для Broyden

Глава 8

Неограниченная оптимизация

В предыдущих главах мы выбрали в значительной степени вариационный подход к построению стандартных алгоритмов вычислительной линейной алгебры. То есть мы определяем целевую функцию, возможно, с ограничениями, и ставим наши алгоритмы как задачу минимизации или максимизации. Пример из нашего предыдущего обсуждения приведен ниже:

Проблема	Цель	Ограничения
Наименьших квадратов	$E(x) = Ax - б$ ^{2}	Никто
Проецироватьb наа	$E(c) = ca - b$	Никто
Собственные векторы симметричной матрицы	$E(x) = x Ax$	$x = 1$
Псевдоинверсия	$E(x) = x$ ^{2}	$Ax = A б$
Анализ основных компонентов	$E(C) = X \quad CCXFro\ CC = Id \times d$	
шаг Бройдена	$E(Jk) = Jk \quad Jk \quad 1$ $^2_{Фро}$	$Jk \cdot (xk \quad xk \quad 1) = f(xk) \quad f(xk \quad 1)$

Очевидно, что такая постановка задач является мощным и общим подходом. По этой причине важно разработать алгоритмы, которые функционируют в отсутствие специальной формы для энергии E, точно так же, как мы разработали стратегии для нахождения корней f, не зная формы априори.

8.1 Неограниченная оптимизация: мотивация

В этой главе мы будем рассматривать задачи без ограничений, то есть задачи, которые могут быть поставлены как минимизация или максимизация функции $f : Rn \quad R$ без каких-либо требований к входным данным. На практике с такими проблемами столкнуться нетрудно; мы перечисляем несколько примеров ниже.

Пример 8.1 (Нелинейный метод наименьших квадратов). Предположим, нам дано некоторое количество пар (xi , yi) таких, что $f(xi) \quad yi$, и мы хотим найти наилучшую аппроксимацию f в определенном классе. Например, мы можем ожидать, что f экспоненциальна, и в этом случае мы должны иметь возможность написать $f(x) = ceax$ для некоторого c и некоторого a; наша задача найти эти параметры. Одной из простых стратегий может быть попытка минимизировать следующую энергию:

$$E(a, c) = \sum_я (yi \quad ceaxi)^2.$$

Эта форма для E не является квадратичной по a, поэтому наши линейные методы наименьших квадратов неприменимы.

Пример 8.2 (оценка максимального правдоподобия). В машинном обучении проблема оценки параметров включает в себя изучение результатов рандомизированного эксперимента и попытку обобщить их с использованием распределения вероятностей определенной формы. Например, мы можем измерить рост каждого ученика в классе, получив список значений роста hi для каждого ученика i. Если у нас много учеников, мы можем смоделировать распределение роста учеников, используя нормальное распределение:

$$g(h; \mu, \sigma) = \sigma \quad \frac{1}{2\pi} e^{(h \quad \mu)\, 2/2\sigma\, 2},$$

где μ — среднее значение распределения, а σ — стандартное отклонение.

При таком нормальном распределении вероятность того, что мы наблюдаем рост hi учащегося i, определяется как g(hi ; μ, σ), и при (разумном) предположении, что рост учащегося i вероятностно не зависит от роста учащегося j, вероятность наблюдения всего набора наблюдаемых высот определяется произведением

$$P(\{h1, \ldots, hn\}; \mu, \sigma) = \quad \prod_{я} g(hi ; \mu, \sigma).$$

Обычный метод оценки параметров μ и σ функции g состоит в том, чтобы максимизировать P как функцию μ и σ при фиксированном {hi} ; это называется оценкой максимального правдоподобия μ и σ. На практике мы обычно оптимизируем логарифмическое правдоподобие (μ, σ) log P({h1, ..., hn}; μ, σ); эта функция имеет те же максимумы, но обладает лучшими числовыми и математическими свойствами.

Пример 8.3 (Геометрические задачи). Многие геометрические задачи, встречающиеся в графике и зрении, не сводятся к энергии наименьших квадратов. Например, предположим, что у нас есть несколько точек x1, . . . ,xk R3 . Если мы хотим сгруппировать эти точки, мы могли бы суммировать их с помощью минимизации одного x:

$$E(x) \quad \sum_{я} x \quad xi2.$$

Точка x R3 , минимизирующая E, называется геометрической медианой {x1, . . . ,xк}. Обратите внимание, что норма разности x xi в E не возведена в квадрат, поэтому энергия больше не является квадратичной по компонентам x.

Пример 8.4 (Физические равновесия, адаптировано из CITE). Предположим, мы прикрепляем объект к набору пружин; каждая пружина закреплена в точке xi R3 и имеет натуральную длину Li и постоянную ki . В отсутствие гравитации, если наш объект находится в точке p R3, сеть источников обладает потенциальной энергией

$$E(p) = 2 \quad \frac{1}{2} \sum_{я} ки (p \quad xi2 \quad Li) \quad 2$$

Равновесия этой системы задаются минимумами E и отражают точки p, в которых все силы пружины уравновешены. Такие системы уравнений используются для визуализации графов G = (V, E) путем присоединения вершин в V пружинами для каждой пары в E.

8.2 Оптимальность

Прежде чем обсуждать, как минимизировать или максимизировать функцию, мы должны четко понимать, что мы ищем; обратите внимание, что максимизация f — это то же самое, что минимизация f , поэтому для нашего рассмотрения достаточно проблемы минимизации. Для конкретного f : Rn R и x * Rn нам нужно вывести). условия оптимальности, имеет наименьшее возможное значение f(x подтверждающие, что x Конечно, в идеале мы хотели бы найти глобальные оптимумы f :

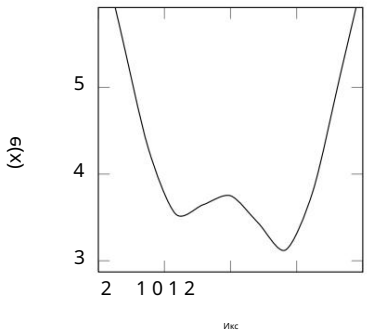


Рисунок 8.1: Функция f(x) с несколькими оптимумами.

Определение 8.1 (Глобальный минимум). Точка x f(x) f(x) * Rn является глобальным минимумом функции f : Rn R, если для всех x Rn .

Поиск глобального минимума f без какой-либо информации о структуре f фактически требует поиска в темноте. Например, предположим, что алгоритм оптимизации идентифицирует локальный минимум около x = 1 в функции на рис. 8.1. Почти невозможно понять, что существует второй, более низкий минимум около x = 1, просто угадывая значения x — насколько нам известно, может существовать третий, даже более низкий минимум f при x = 1000!

Таким образом, во многих случаях мы удовлетворяемся тем, что находим локальный минимум:

Определение 8.2 (Локальный минимум). Точка x f(x) для всех* Rn является локальным минимумом функции f : Rn R, если f(x) x Rn , удовлетворяющих x x * < ε для некоторого ε > 0.

выражением Это определение требует, * достигает наименьшего значения в некоторой окрестности, определяемой чтобы x был радиусом ε. Обратите внимание, что у алгоритмов локальной оптимизации есть серьезное ограничение: они не могут гарантировать, что они дадут наименьшее возможное значение f, как на рис. 8.1, если будет достигнут левый локальный минимум; многие стратегии, эвристические и другие, применяются для изучения ландшафта возможных значений x, чтобы помочь обрести уверенность в том, что локальный минимум имеет наилучшее возможное значение.

8.2.1 Дифференциальная оптимальность

Знакомая история из исчисления с одной и несколькими переменными состоит в том, что поиск потенциальных минимумов и максимумов функции f : Rn R более прост, когда f дифференцируема. Напомним, что вектор градиента f = (f/ x1, ..., f / xn) указывает направление, в котором f увеличивается больше всего; вектор f указывает в направлении наибольшего убывания. Один из способов убедиться в этом — вспомнить, что вблизи a f выглядит как точка линейной функции x0 Rn

$$f(x) \quad f(x0) + \quad f(x0) \cdot (x \quad x0).$$

Если мы возьмем x x0 = α f(x0), то найдем:

$$f(x0 + \alpha \quad f(x0)) \quad f(x0) + \alpha \quad f(x0)^{2}$$

Когда f(x0) > 0, знак α определяет, увеличивается или уменьшается f.

Нетрудно формализовать приведенное выше рассуждение, чтобы показать, что если x0 является локальным минимумом, то мы должны иметь f(x0) = 0. Обратите внимание, что это условие необходимо, но недостаточно: максимумы и седло

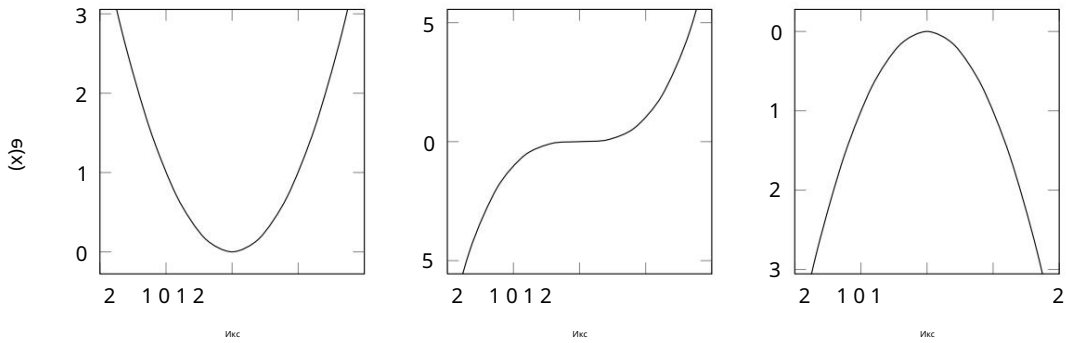


Рисунок 8.2: Критические точки могут принимать разные формы; здесь мы показываем локальный минимум, седловую точку и локальный максимум.

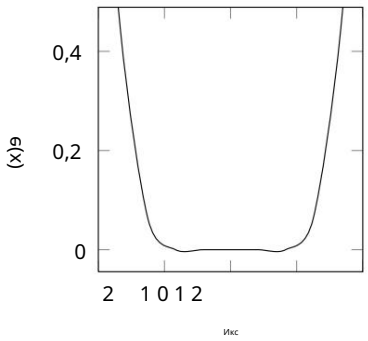


Рисунок 8.3: Функция со многими стационарными точками.

точки также имеют f(x0) = 0, как показано на рис. 8.2. Тем не менее, это наблюдение о минимумах дифференцируемых функций дает общую стратегию поиска корней:

1. Найдите точки xi, удовлетворяющие условию f(xi) =0.

2. Проверьте, какая из этих точек является локальным минимумом, а не максимумом или седловой точкой.

Учитывая их важную роль в этой стратегии, мы даем точкам, которые мы ищем, особое имя:

Определение 8.3 (Стационарная точка). Стационарной точкой функции f : Rn R называется точка x Rn , удовлетворяющая условию f(x) =0.

То есть наша стратегия минимизации может состоять в том, чтобы найти стационарные точки f и затем исключить те, которые не являются минимумами.

Важно помнить, когда мы можем ожидать успеха наших стратегий минимизации.
В большинстве случаев, таких как показанные на рис. 8.2, стационарные точки f изолированы, что означает, что мы можем записать их в дискретный список {x0,x1, . . .}. Однако вырожденный случай показан на рис. 8.3; здесь весь интервал [1/2, 1/2] состоит из стационарных точек, что делает невозможным рассмотрение их по одной. По большей части мы будем игнорировать такие вопросы, как вырожденные случаи, но вернемся к ним, когда будем рассматривать обусловленность задачи минимизации.

Предположим, мы идентифицируем точку x R как стационарную точку f и теперь хотим проверить, является ли она локальным минимумом. Если f дважды дифференцируема, одна из стратегий, которую мы можем использовать, состоит в том, чтобы записать его гессиан

матрица:

$$H\phi(x) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} & \frac{\partial^2}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} \\ \frac{\partial^2}{\partial x_2 \partial x_1} & \frac{\partial^2}{\partial x_2^2} & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} & \frac{\partial^2}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2}{\partial x_{2n}} \end{bmatrix}$$

Мы можем добавить еще один член в наше разложение Тейлора f, чтобы увидеть роль Hf :

$$f(x) \quad f(x0) + \quad f(x0) \cdot (x \quad x0) + 2 \quad \frac{1}{(x - x0)\, Hf(x - x0)}$$

Если мы подставим стационарную точку x *, то по определению мы знаем:

$$f(x) \quad f(x \quad) + (x \quad x \quad)^{T} Hf(x \quad x \quad) 2$$

Если Hf положительно определен, то это выражение показывает, что f(x) f(x), и, таким образом, x * является локальным минимумом.
В более общем случае может возникнуть одна из нескольких ситуаций:

- Если Hf положительно определена, то x * является локальным минимумом f .
- Если Hf отрицательно определен, то x * является локальным максимумом f .
- Если Hf неопределенно, то x * является седловой точкой f .
- Если Hf необратима, то могут возникнуть странности, такие как функция на рис. 8.3.

Проверить, является ли матрица положительно определенной, можно, проверив, существует ли ее факторизация Холецкого, или, что медленнее, проверив, что все ее собственные значения положительны. Таким образом, когда гессиан функции f известен, мы можем проверить стационарные точки на оптимальность, используя приведенный выше список; многие алгоритмы оптимизации, включая те, которые мы обсудим, просто игнорируют последний случай и уведомляют пользователя, поскольку это относительно маловероятно.

8.2.2 Оптимальность через функциональные свойства

Иногда, если мы знаем больше информации о f : Rn R , мы можем предоставить условия оптимальности, которые сильнее или легче проверяются, чем приведенные выше.

Одним из свойств f, которое имеет большое значение для оптимизации, является выпуклость, показанная на рис. НОМЕР:

Определение 8.4 (выпуклое). Функция f : Rn R называется выпуклой, если для всех x,y Rn и α (0, 1) выполняется следующее соотношение:

$$f((1 \quad \alpha)x + \alpha y) \quad (1 \quad \alpha)f(x) + \alpha f(y).$$

Когда неравенство строгое, функция строго выпуклая.

Выпуклость означает, что если вы соедините в Rn две точки линией, значения f вдоль линии будут меньше или равны тем, которые вы получили бы с помощью линейной интерполяции.

Выпуклые функции обладают многими сильными свойствами, самые основные из которых следующие:

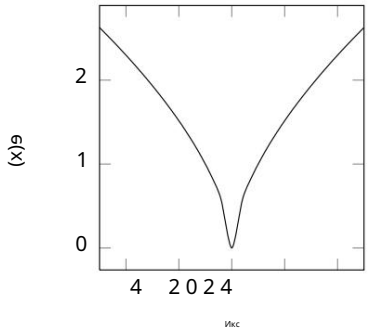


Рис. 8.4: Квазивыпуклая функция.

Предложение 8.1. Локальный минимум выпуклой функции f : Rn R обязательно является глобальным минимумом.

Доказательство. Возьмем x за такой локальный минимум и предположим, что x * = x c f(x) < f(x). существует. Тогда для α (0, 1)

$$f(x + \alpha(x \quad x)) \quad (1 \quad \alpha)f(x) + \alpha\, f(x \quad) \text{ по выпуклости}$$

$$< f(x) \text{ , так как } f(x \quad) < f(x)$$

Но переход α 0 показывает, что x не может быть локальным минимумом. □

Это предложение и связанные с ним наблюдения показывают, что можно проверить, достигли ли вы глобального минимума выпуклой функции, просто применяя оптимальность первого порядка. Таким образом, полезно проверить вручную, не окажется ли оптимизируемая функция выпуклой, что на удивление часто происходит в научных вычислениях; одно достаточное условие, которое легче проверить, когда f дважды дифференцируема, состоит в том, что Hf везде положительно определена.

Другие методы оптимизации имеют гарантии при других предположениях о f. Для экзамена ple, одной из более слабых версий выпуклости является квазивыпуклость, достигаемая, когда

$$f((1 \quad \alpha)x + \alpha y) \quad \max(f(x), f(y)).$$

Пример квазивыпуклой функции показан на рис. 8.4; хотя она не имеет характерной формы «чаши» выпуклой функции, она имеет уникальный оптимум.

8.3 Одномерные стратегии

Как и в предыдущей главе, мы начнем с одномерной оптимизации f : R R , а затем расширим наши стратегии до более общих функций f : Rn R.

8.3.1 Метод Ньютона

Наша основная стратегия минимизации дифференцируемых функций f : Rn R будет состоять в том, чтобы найти sta стационарные , удовлетворяющую f(x) = 0. Предположим, что мы можем проверить, являются ли точки условными точками x максимумов, минимумов или седловых точек в качестве пост- шаге обработки, сосредоточимся на задаче нахождения стационарных точек x *

Для этого предположим, что f : R R дифференцируемо. Тогда, как и в нашем выводе Ньютона метод нахождения корня, мы можем аппроксимировать:

$$1\ f(x) \quad f(xk) + f\,(xk)(x \quad xk) + f\,(xk)(x \quad xk)\ 2\ ^{2}.$$

Аппроксимация в правой части представляет собой параболу, вершина которой расположена в точке xk f (xk)/f (xk). Конечно, в действительности f не обязательно является параболой, поэтому метод Ньютона просто итерирует формулу

$$xk+1 = xk \quad f \quad \frac{f\,(xk)}{(xk)}.$$

Эту технику легко проанализировать, учитывая ту работу, которую мы уже проделали для понимания метода Ньютона для поиска корней в предыдущей главе. В частности, альтернативный способ вывести приведенную выше формулу исходит из нахождения корня f (x), поскольку стационарные точки удовлетворяют f (x) = 0.
Таким образом, в большинстве случаев метод оптимизации Ньютона демонстрирует квадратичную сходимость при условии, что начальное приближение x0 достаточно близко к x^{*}.

Возникает естественный вопрос: можно ли аналогичным образом применить метод секущих?
Наш вывод метода Ньютона, приведенный выше, находит корни f , поэтому метод секущих можно использовать для устранения вычисления f, но не f ; ситуации, в которых мы знаем f, но не f, относительно редки. Более подходящей параллелью является замена отрезков, используемых для аппроксимации f в методе секущих, параболами. Эта стратегия, известная как последовательная параболическая интерполяция, также минимизирует квадратичное приближение f на каждой итерации, но вместо использования f(xk), f (xk) и f (xk) для построения приближения, которое использует f(xk), f(xk 1) и f(xk 2). Вывод этого метода относительно прост, и он сходится сверхлинейно.

8.3.2 Поиск золотого сечения

Мы пропустили деление пополам в нашем параллельном методе поиска корней с одной переменной. Причин этого упущения много. Наша мотивация для деления пополам заключалась в том, что оно использовало только самое слабое предположение относительно f, необходимое для нахождения корней: непрерывность. Однако теорема о промежуточном значении не применима к минимумам каким-либо интуитивным образом, поэтому кажется, что такого прямого подхода не существует.

Однако полезно иметь по крайней мере одну доступную стратегию минимизации, которая не требует дифференцируемости f в качестве базового предположения; ведь существуют недифференцируемые функции, имеющие четкие минимумы, например f(x) |x| при x = 0. С этой целью одним из альтернативных предположений может быть то, что f унимодулярна:

Определение 8.5 (Унимодулярный). Функция f : [a, b] R называется унимодулярной, если существует x [a, b] такое, что f убывает при x [a, x] и возрастает при x [x , b].

Другими словами, унимодулярная функция некоторое время убывает, а затем начинает возрастать; локализованные минимумы не допускаются. Обратите внимание, что такие функции, как |x| не дифференцируемы, но все же унимодулярны.

Предположим, у нас есть два значения x0 и x1, такие что a < x0 < x1 < b. Мы можем сделать два обсерва ции, которые помогут нам сформулировать методику оптимизации:

- Если f(x0) f(x1), то мы знаем, что f(x) f(x1) для всех x [a, x0]. Таким образом, интервал [a, x0] можно отбросить при поиске минимума f .

• Если f(x1) f(x0), то мы знаем, что f(x) f(x0) для всех x [x1, b], и поэтому можем отбросить [x1, б].

Эта структура предлагает потенциальную стратегию минимизации, начиная с интервала [a, b] и итеративно удаляя части в соответствии с приведенными выше правилами.

Однако остается одна важная деталь. Наша гарантия сходимости для алгоритма деления пополам основывалась на том факте, что мы могли удалить половину рассматриваемого интервала на каждой итерации.
Мы могли бы действовать аналогичным образом, каждый раз удаляя треть интервала; это требует двух оценок f во время каждой итерации в новых местах x0 и x1 . Однако, если оценка f требует больших затрат, мы можем захотеть повторно использовать информацию из предыдущих итераций, чтобы избежать по крайней мере одной из этих двух оценок.

На данный момент a = 0 и b = 1; стратегии, которые мы выведем ниже, будут работать в более общем плане за счет сдвига и масштабирования. В отсутствие дополнительной информации о f мы могли бы также сделать симметричный выбор x0 = α и x1 = 1 α для некоторого α (0, 1/2). Предположим, наша итерация удаляет крайний правый интервал [x1, b]. Тогда интервал поиска становится равным [0, 1 α], и мы знаем f(α) из предыдущей итерации. Следующая итерация разделит [0, 1 α] так, что x0 = α(1 α) и x1 = (1 α) ^{2}. Если мы хотим повторно использовать f(α) из предыдущей итерации, мы могли бы установить (1 α) ^{2} = α, что дает:

$$\alpha = \frac{1}{2}(3 - \sqrt{5})$$

$$1 - a = 2 \frac{1}{-}(\sqrt{5} - 1)$$

Значение 1 α τ, указанное выше, является золотым сечением! Это позволяет повторно использовать одну из оценок функции из предыдущих итераций; симметричный аргумент показывает, что тот же выбор α работает, если мы удалили левый интервал вместо правого.

Алгоритм поиска золотого сечения использует эту конструкцию (CITE):

1. Возьмем τ $\frac{-}{1 2}($ $\sqrt{5}$ 1)., и инициализировать a и b так, чтобы f была унимодулярной на [a, b].

2. Сделать начальное подразделение x0 = a + (1 τ)(b a) и x1 = a + τ(b a).

3. Инициализировать f0 = f(x0) и f1 = f(x1).

4. Повторяйте до тех пор, пока b a не станет достаточно малым:

(a) Если f0 f1, то удалить интервал [a, x0] следующим образом:

- Переместить влево: a x0
- Повторно использовать предыдущую итерацию: x0 x1, f0 f1 • Создать новую выборку: x1 a + τ(b a), f1 f(x1)

(b) Если f1 > f0, то удалить интервал [x1, b] следующим образом:

- Переместить вправо: b x1
- Повторно использовать предыдущую итерацию: x1 x0, f1 f0 • Создать новую выборку: x0 a + (1 τ)(b a), f0 f(x0)

Очевидно, этот алгоритм сходится безусловно и линейно. Когда f не является глобально унимодальным, может быть трудно найти [a, b] такое, что f является унимодальным на этом интервале, что несколько ограничивает применение этого метода; обычно [a, b] угадывается при попытке заключить в скобки локальный минимум f .

8.4 Многовариантные стратегии

Мы продолжаем параллель с нашим обсуждением поиска корней, расширяя наше обсуждение до многовариантных проблем. Как и в случае поиска корня, задачи с несколькими переменными значительно сложнее, чем задачи с одной переменной, но на практике они возникают так часто, что заслуживают внимательного рассмотрения.

Здесь мы будем рассматривать только случай, когда f : Rn R дифференцируема. Методы оптимизации, более похожие на поиск золотого сечения для недифференцируемых функций, имеют ограниченное применение и их трудно сформулировать.

8.4.1 Градиентный спуск

Напомним из нашего предыдущего обсуждения, что f(x) указывает в направлении «самого крутого подъема» f в точке x; аналогично вектор f(x) является направлением «самого крутого спуска». По крайней мере, это определение гарантирует, что при f(x) = 0 для малых α > 0 мы должны иметь

$$f(x \quad \alpha \quad f(x)) \quad f(x).$$

Предположим, что наша текущая оценка местоположения минимума f равна xk . Затем мы можем выбрать xk+1 так, чтобы f(xk+1) < f(xk) для итеративной стратегии минимизации. Одним из способов упростить поиск xk+1 было бы использование одного из наших одномерных алгоритмов из §8.3 для решения более простой задачи. В частности, рассмотрим функцию gk(t) f(xk t f(xk)), ограничивающую f на прямую, проходящую через xk , параллельную f(xk). Благодаря нашему обсуждению градиента мы знаем, что малое t приведет к уменьшению f.

Алгоритм градиентного спуска итеративно решает эти одномерные задачи, чтобы улучшить нашу оценку xk :

1. Выбрать начальную оценку x0
2. Итерировать до сходимости xk :
   (a) Возьмем gk(t) f(xk t f(xk)) (b)
   Используем одномерный алгоритм для нахождения * минимизация gk по всем t 0 («линейный поиск»)
   t (c) Возьмем xk+1 xk t f(xk)

Каждая итерация градиентного спуска уменьшает f(xk), поэтому целевые значения сходятся. Алгоритм завершается только тогда, когда f (xk) 0, показывая, что градиентный спуск должен как минимум достигать локального минимума; Однако сходимость для большинства функций f медленная. Процесс линейного поиска можно заменить методом, который просто уменьшает цель на незначительную, если не оптимальную величину, хотя в этом случае труднее гарантировать сходимость.

8.4.2 Метод Ньютона

Параллельно с нашим выводом случая с одной переменной, мы можем написать аппроксимацию ряда Тейлора для f : Rn R , используя его гессиан Hf :

$$f(x) \quad f(xk) + \quad f(xk) \cdot (x \quad xk) + 2 \qquad \frac{1}{-}(x \quad xk) \cdot Hf(xk) \cdot (x \quad xk)$$

Дифференцирование по x и установка результата равным нулю дает следующую итеративную схему:

$$x_{k+1} = x_k \quad [Hf(x_k)] \quad 1 \quad f(x_k)$$

Легко проверить еще раз, что это выражение является обобщением выражения из § 8.3.1, и еще раз оно сходится квадратично, когда x0 близко к минимуму.

Метод Ньютона может быть более эффективным, чем градиентный спуск, в зависимости от цели оптимизации f . Напомним, что каждая итерация градиентного спуска потенциально требует множества вычислений f во время процедуры линейного поиска. С другой стороны, мы должны оценивать и инвертировать гессиан Hf во время каждой итерации метода Ньютона. Обратите внимание, что эти факторы не влияют на количество итераций, но влияют на время выполнения: это компромисс, который может быть неочевидным при традиционном анализе.

Интуитивно понятно, почему метод Ньютона быстро сходится, когда он близок к оптимуму. В частности, градиентный спуск не знает Hf ; это происходит аналогично спуску с горы, когда вы смотрите только на свои ноги. Используя Hf , метод Ньютона имеет большую картину формы f поблизости.

Однако , когда Hf не является положительно определенным, объектив локально может выглядеть как седло или пик, а не как чаша. В этом случае переход к приблизительно стационарной точке может не иметь смысла.

Таким образом, адаптивные методы могут проверять, является ли Hf положительно определенным, прежде чем применять шаг Ньютона; если он не является положительно определенным, методы могут вернуться к градиентному спуску, чтобы найти лучшее приближение к минимуму. В качестве альтернативы они могут модифицировать Hf , например, путем проецирования на ближайшую положительно определенную матрицу.

8.4.3 Оптимизация без производных: BFGS

Метод Ньютона может быть сложно применить к сложным функциям $f : \mathbb{R}^n \to \mathbb{R}$. Вторая производная от f может быть значительно более сложной, чем форма f , и Hf меняется с каждой итерацией, что затрудняет повторное использование результатов предыдущих итераций. Кроме того, Hf имеет размер $n \times n$, поэтому для хранения Hf требуется $O(n^2)$ пространство, что может быть неприемлемо.

Как и в нашем обсуждении поиска корней, методы минимизации, имитирующие метод Ньютона, но использующие приближенные производные, называются квазиньютоновскими методами. Часто они могут иметь столь же сильные свойства сходимости без необходимости явной переоценки и даже факторизации гессиана на каждой итерации. В нашем обсуждении мы будем следить за развитием (CITE NO CEDAL AND WRIGHT).

Предположим, мы хотим минимизировать $f : \mathbb{R}^n \to \mathbb{R}$, используя итеративную схему. Рядом с текущей оценкой x_k корня, мы могли бы оценить f с помощью квадратичной модели:

$$1\ f(x_k + \delta x) \quad f(x_k) + \quad f(x_k) \cdot \delta x + \quad \frac{}{2}\ (\delta x)\ B_k(\delta x).$$

Обратите внимание, что мы попросили, чтобы наше приближение согласовывалось с f до первого порядка в точке x_k ; однако, как и в методе Бройдена для нахождения корня, мы позволим нашей оценке гессиана B_k варьироваться.

Эта квадратичная модель минимизируется, если взять $\delta x = \quad B \quad {}_{k}^{1} \quad f(x_k)$. В случае, когда δx_2 велико и мы не хотим делать такой значительный шаг, мы позволим себе масштабировать эту разницу на величину шага α_k , что даст

$$x_{k+1} = x_k - \alpha_k B \quad {}_{k}^{1} \quad f(x_k).$$

Наша цель — найти разумную оценку B_{k+1} , обновив B_k , чтобы мы могли повторить этот процесс.

Гессе функции f есть не что иное, как производная от f , поэтому мы можем записать условие в стиле секущей на Bk+1 :

$$B_{k+1}(x_{k+1} - x_k) = f(x_{k+1}) - f(x_k).$$

Подставим $s_k = x_{k+1} - x_k$ и $y_k = f(x_{k+1}) - f(x_k)$, что приведет к эквивалентному условию $B_{k+1}s_k = y_k$.

Учитывая проводимую оптимизацию, мы хотим, чтобы Bk обладал двумя свойствами:

- Bk должна быть симметричной матрицей, такой как гессиан Hf .
- Bk должно быть положительно (полу)определенным, так что мы ищем минимумы, а не максимумы или седловые точки.

Условия симметрии достаточно, чтобы исключить возможность использования оценки Бройдена, развитой нами в предыдущей главе.

Ограничение положительной определенности неявно накладывает условие на связь между sk и B частности, предварительно умножая связь $B_{k+1}s_k = y_k$ на s_k $B_{k+1}s_k = s_k y_k$. Для юк . показывает с к Чтобы Bk+1 было положительно определенным, мы должны иметь $s_k \cdot y_k > 0$. Это наблюдение может направить нас при выборе α_k ; легко видеть, что оно выполняется при достаточно малых $\alpha_k > 0$.

Предположим, что sk и yk удовлетворяют нашему условию совместимости. Имея это на месте, мы можем написать оптимизация в стиле Бройдена, ведущая к возможному приближению Bk+1 :

$$\begin{aligned} &\text{минимизировать } B_{k+1} \quad \|B_{k+1} - B_k\| \\ &\text{такое, что } B_{k+1}s_k = y_k \end{aligned}$$

Для соответствующего выбора норм $\|\cdot\|$, эта оптимизация дает хорошо известный DFP (Davidon итерационная схема Флетчера-Пауэлла.

Вместо того, чтобы вдаваться в детали схемы DFP, мы перейдем к более популярному методу, известному как формула BFGS (Бройден-Флетчер-Гольдфарб-Шанно), которая используется во многих современных системах. Заметьте, что — игнорируя наш выбор α_k на данный момент — наше приближение второго порядка было минимизировано, приняв $\delta x = -B_k^{-1} f(x_k)$. Таким образом, в итоге поведение нашей итерационной схемы диктуется обратной матрицей B_k^{-1} . Запрос о том, что $B_{k+1} - B_k$ мал, все же может подразумевать относительно плохие различия между действием B_k^{-1} и что из B_{k+1}^{-1}

Учитывая это наблюдение, схема BFGS вносит небольшое изменение в приведенный выше вывод. Вместо того, чтобы вычислять Bk на каждой итерации, мы можем напрямую вычислить обратное значение $H_k = B_k^{-1}$. Теперь наше условие $B_{k+1}s_k = y_k$ меняется на $s_k = H_{k+1}y_k$; условие симметричности Bk равносильно требованию симметричности Hk . Мы решаем оптимизацию

$$\begin{aligned} &\text{минимизировать } H_{k+1} \quad \|H_{k+1} - H_k\| \\ &\text{такое, что } H_{k+1}s_k = y_k \end{aligned}$$

У этой конструкции есть приятное дополнительное преимущество, заключающееся в том, что для вычисления $\delta x = -H_k f(x_k)$ не требуется обращения матриц.

Чтобы вывести формулу для H_{k+1} , мы должны определить матричную норму $\|\cdot\|$. Как и в нашем предыдущем обсуждении, норма Фробениуса выглядит наиболее близкой к оптимизации методом наименьших квадратов, что делает более вероятным, что мы можем сгенерировать выражение в закрытой форме для H_{k+1} , а не решать приведенную выше минимизацию как подпрограмму оптимизации BFGS.

Однако у нормы Фробениуса есть один серьезный недостаток для матриц Гессе. Напомним, что матрица Гессе имеет элементы (Hf)ij = fi/ xj . Часто величины xi для разных i могут иметь разные единицы измерения; например, рассмотрите максимизацию прибыли (в долларах), полученной от продажи чизбургера радиуса r (в дюймах) и цены p (в долларах), что приводит к f : (дюймы, доллары) доллары. Возводить в квадрат эти различные количества и складывать их не имеет смысла.

Предположим, мы нашли симметричную положительно определенную матрицу W такую, что Wsk = yk ; в упражнениях проверим, что такая матрица существует. Такая матрица переводит единицы sk = xk+1 xk в единицы yk = f(xk+1) f(xk). Вдохновившись нашим выражением $A = Tr(A^2_{Фро} A)$, мы можем определить взвешенную норму Фробениуса матрицы A как

$$A \; {}^{2}_{Вт} \quad Tp(A \; BAB)$$

Несложно проверить, что это выражение имеет согласованные единицы применительно к нашей оптимизации для Hk+1 . Когда и W, и A симметричны со столбцами w i andai соответственно, расширение приведенного выше выражения показывает:

$$A \; {}^{2}_{Вт} = \sum_{ij} (w\,i \cdot aj)(w\,j \cdot ai).$$

Этот выбор нормы в сочетании с выбором W дает особенно чистую формулу для Hk+1 и yk : при заданных Hk ,sk ,

$$Hk+1 = (In\times n \quad \rho ksky\,k \quad)Hk(In\times n \quad \rho kyks\,k\,) + \rho ksks\,k\,,$$

где ρk 1/y·s. В приложении к этой главе мы покажем, как вывести эту формулу.

Алгоритм BFGS избавляет от необходимости вычислять и инвертировать матрицу Гессе для f , но по-прежнему требует O(n). Полезный вариант для $^{2}_{}$ экономии памяти, известный как L-BFGS («BFGS с ограниченной памятью»), позволяет избежать этой проблемы, сохраняя ограниченную историю векторов yk andsk и применение Hk путем рекурсивного расширения его формулы.Этот подход на самом деле может иметь лучшие числовые свойства, несмотря на его компактное использование пространства; в частности, старые векторы yk isk могут больше не иметь значения и должны быть проигнорированы.

8.5 Проблемы

Список идей:

- Вычислить Гаусс-Ньютон
- Стохастические методы, АдаГрад
- Алгоритм VSCG
- условия Вульфа для градиентного спуска; подключиться к БФГС
- Формула Шермана-Моррисона-Вудбери для Bk для BFGS
- Докажите, что BFGS сходится; показать существование матрицы W
- (Обобщенный) алгоритм уменьшенного градиента
- Номер условия для оптимизации

Приложение: Получение BFGS Update1

Наша оптимизация для Hk+1 имеет следующее выражение для множителя Лагранжа (для простоты записи мы берем Hk+1 H и Hk = H):

$$\Lambda = \sum_{ij} (w_i \cdot (h_j - h_{j*}))(w_j \cdot (h_i - h_{i*})) - \sum_{i<j} \alpha_{ij}(H_{ij} - H_{ji}) - \lambda (H y_k - s_k)$$

$$= \sum_{ij} (w_i \cdot (h_j - h_{j*}))(w_j \cdot (h_i - h_{i*})) - \sum_{ij} \alpha_{ij} H_{ij} - \lambda (H y_k - s_k) \text{ , если предположить, что } \alpha_{ij} = \alpha_{ji}$$

Взятие производных для нахождения критических точек показывает (для y yk , s sk):

$$0 = \frac{\partial \Lambda}{\partial} = 2w_i(w_j \cdot (h - h)) - \alpha_{ij} - \lambda_i y_j$$

$$H_{ij} = 2 \sum_i w_i(W(H - H))_j - \alpha_{ij} - \lambda_i y_j$$

$$= 2 \sum_i (W(H - H))_j w_i - \alpha_{ij} - \lambda_i y_j \text{ в силу симметрии } W$$

$$= 2(W(H - H)W)_{ji} - \alpha_{ij} - \lambda_i y_j$$

$$= 2(W(H - H)W)_{ij} - \alpha_{ij} - \lambda_i y_j \text{ в силу симметрии } W \text{ и } H$$

Итак, в матричной форме мы имеем следующий список фактов:

$$0 = 2W(H - H)W - A - \lambda y \text{ , где } A_{ij} = \alpha_{ij}$$

$$A = -A,\ W = W,\ H = H,\ (H^*) = H$$

$$Hy = c,\ Ws = y$$

Мы можем получить пару отношений, используя транспозицию в сочетании с симметрией H и W и асимметрией A:

$$0 = 2W(H - H)W - A - \lambda y$$

$$0 = 2W(H - H)W + A - y\lambda = 0$$

$$4W(H - H)W - \lambda y - y\lambda$$

Пост-умножение этого отношения на показывает:

$$0 = 4(y - WH - y) - \lambda(y \cdot s) - y(\lambda \cdot s)$$

Теперь возьмем скалярный продукт с:

$$0 = 4(y \cdot s) - 4(y H - y) - 2(y \cdot s)(\lambda \cdot s)$$

Это показывает:

$$\lambda \cdot s = 2\rho y\ (s - H - y), \text{ при } \rho - 1/y \cdot s$$

1Особая благодарность Тао Ду за отладку нескольких частей этого вывода.

Теперь подставим это в наше векторное равенство:

$$0 = 4(y \quad WH \quad y) \quad \lambda(y \cdot s) \quad y(\lambda \cdot s) \text{ из предыдущего}$$
$$= 4(y \quad WH \quad y) \quad \lambda(y \cdot s) \quad y[2\rho y (s \quad H \quad y)] \text{ из нашего упрощения}$$
$$= \quad \lambda = 4\rho(y \quad WH \quad y) \quad 2\rho \quad {}^{2}y (s \quad H \quad y)y$$

Пост-умножение на y показывает:

$$\lambda y = 4\rho(y \quad WH \quad y)y \quad 2\rho \quad {}^{2}y (s \quad H \quad y)yy$$

Выполняя транспонирование,

$$y\lambda = 4\rho y(y \quad yH \quad W) \quad 2\rho \quad {}^{2}y (s \quad H \quad y)yy$$

Объединение этих результатов и деление на четыре показывает:

$$\frac{1}{4}(\lambda y + y\lambda) = \rho(2yy \quad WH \quad yy \quad yy\, H \quad W) \quad \rho \quad {}^{2}y (s \quad H \quad y)yy$$

Теперь мы предварительно и после умножим на $W \quad 1$. Поскольку $Ws = y$, мы можем эквивалентно писать $= W \quad 1y$; кроме того, в силу симметрии W мы знаем, что $y\, W \quad 1 = s$. Применение этих тождеств к приведенному выше выражению показывает:

$$\frac{1}{4}W \quad 1\,(\lambda y + y\lambda)W \quad 1 = 2\rho ss \quad \rho H \quad {}^{*}ys - \rho sy\, H \quad {}^{*} \quad \rho^{2}(ys)ss + \rho \quad {}^{2}(y\, H \quad y)ss$$
$$= 2\rho ss \quad \rho H \quad {}^{*}ys - \rho sy\, H \quad {}^{*} \quad \rho ss + s\rho \quad {}^{2}(y\, H \quad y)s \text{ по определению } \rho$$
$$= \rho ss \quad \rho H \quad {}^{*}ys - \rho sy\, H \quad {}^{*} + s\rho \quad {}^{2}(y\, H \quad y)s$$

Наконец, мы можем завершить вывод шага BFGS следующим образом:

$$0 = 4W(H \quad H \quad)W \quad \lambda y \quad y\lambda \text{ из предыдущего}$$
$$= \quad H = 4 \underset{=}{\frac{1}{-}} W \quad 1\,(\lambda y + y\lambda)W \quad 1 + H \quad {}^{*}$$
$$\rho ss \quad \rho H \quad {}^{*}ys - \rho sy\, H \quad {}^{*} + s\rho \quad {}^{2}(y\, H \quad y)s + H \quad {}^{*} \text{ из последнего абзаца}$$
$$= H \quad (I \quad \rho ys) + \rho ss \quad \rho sy\, H \quad (I \quad {}^{*} + (\rho sy)H \quad (\rho ys)$$
$$= H \quad \rho ys) + \rho ss \quad \rho sy\, H \quad (I \quad \rho ys) = \rho ss + (I$$
$$\rho sy)H \quad (I \quad \rho ys)$$

Это окончательное выражение и есть шаг BFGS, представленный в этой главе.

Глава 9

Ограниченная оптимизация

Мы продолжаем рассмотрение задач оптимизации, изучая случай с ограничениями. Эти проблемы принимают следующий общий вид:

$$\begin{aligned} &\text{минимизировать} \\ &f(x) \text{ так, чтобы } g(x) = 0 \\ &h(x) \geq 0 \end{aligned}$$

Здесь $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ и $h : \mathbb{R}^n \to \mathbb{R}^p$. Очевидно, что эта форма является чрезвычайно общей, поэтому нетрудно предсказать, что алгоритмы решения таких задач при отсутствии дополнительных предположений относительно f, g или h могут быть трудно сформулировать и будут подвержены вырождениям, таким как локальные минимумы и отсутствие конвергенция. На самом деле эта оптимизация кодирует другие проблемы, которые мы уже рассмотрели; если мы возьмем $f(x) \equiv 0$, то эта оптимизация с ограничениями станет нахождением корней на g, а если мы возьмем $g(x) = h(x) \equiv 0$, то она сведется к оптимизации без ограничений на f.

Несмотря на эту несколько мрачную перспективу, оптимизация для общего случая с ограничениями может оказаться полезной, когда f, g и h не имеют полезной структуры или слишком специализированы, чтобы заслуживать специального обращения. Кроме того, когда f в любом случае является эвристическим, простое нахождение возможного x, для которого $f(x) < f(x_0)$ для начального предположения x_0, ценно. Одним из простых приложений в этой области может быть экономическая система, в которой f измеряет затраты; очевидно, мы хотим минимизировать затраты, но если x_0 представляет текущую конфигурацию, любое уменьшение f на x является ценным выходом.

9.1 Мотивация

На практике нетрудно столкнуться с задачами условной оптимизации. На самом деле, мы уже перечислили множество приложений этих задач, когда обсуждали собственные векторы и собственные значения, поскольку эту задачу можно поставить как нахождение критических точек $x^\top Ax$ при $\|x\|_2 = 1$; конечно, частный случай вычисления собственных значений допускает специальные алгоритмы, упрощающие эту задачу.

Здесь мы перечисляем другие оптимизации, которые не пользуются структурой задач на собственные значения:

Пример 9.1 (геометрическая проекция). Многие поверхности S в $\mathbb{R}^3$ можно неявно записать в виде $g(x) = 0$ для некоторого g. Например, единичная сфера получается, если взять $g(x) \equiv \|x\|_2^2 - 1$, тогда как куб может

можно построить, взяв g(x) = x1 1. Фактически, некоторые среды 3D-моделирования позволяют пользователям задавать «кляксы»-объекты, как на рисунке НОМЕР, как суммы

$$r(x) \quad c + \sum_{i} a_i e^{b_i \|x - x_i\|_2^2}.$$

Предположим, что нам дана точка y R3 , и мы хотим найти точку на S, ближайшую к y. Эта проблема решается с помощью следующей условной минимизации:

$$\begin{aligned} &\text{минимизировать } x \cdot x \\ &y2 \text{ такое, что } g(x) = 0 \end{aligned}$$

Пример 9.2 (Производство). Предположим, у вас есть m различных материалов; у вас есть si единиц каждого материала i на складе. Вы можете производить k различных продуктов; продукт j дает вам прибыль pj и использует cij материала i для его производства. Чтобы максимизировать прибыль, вы можете решить следующую оптимизацию для общего количества xj , которое вы должны произвести каждого изделия j:

$$\begin{aligned} &\text{максимекс } \sum_{j=1}^{k} p_j x_j \\ &\text{такое, что } x_j \quad 0 \quad j \quad \{1, \ldots, k\} \\ &\sum_{j=1}^{k} c_{ij} x_j \quad s_i \quad i \quad \{1, \ldots, m\} \end{aligned}$$

Первое ограничение гарантирует, что вы не сделаете отрицательные числа для любого продукта, а второе гарантирует, что вы не используете больше, чем ваш запас каждого материала.

Пример 9.3 (неотрицательный метод наименьших квадратов). Мы уже видели множество примеров задач наименьших квадратов, но иногда отрицательные значения в векторе решения могут не иметь смысла. Например, в компьютерной графике анимированная модель может быть представлена как деформирующаяся костная структура плюс сетчатая «кожа»; для каждой точки на коже можно рассчитать список весов, чтобы аппроксимировать влияние положения суставов костей на положение вершин кожи (CITE). Такие веса должны быть неотрицательными, чтобы избежать вырожденного поведения при деформации поверхности. В таком случае мы можем решить задачу «неотрицательных наименьших квадратов»:

$$\begin{aligned} &\text{минимизируем } x \; Ax \quad b2 \\ &\text{такое, что } x_i \quad 0 \quad i \end{aligned}$$

Недавние исследования включают характеристику разреженности неотрицательных решений методом наименьших квадратов, которые часто имеют несколько значений xi, точно удовлетворяющих xi = 0 (CITE).

Пример 9.4 (Корректировка связки). Предположим, что в компьютерном зрении мы делаем снимок объекта с нескольких ракурсов. Естественной задачей является воссоздание трехмерной формы объекта. Для этого мы могли бы отметить соответствующий набор точек на каждом изображении; в частности, мы можем взять xij R2 в качестве положения характерной точки j на изображении i. На самом деле у каждой характерной точки есть положение yj R3 в пространстве, которое мы хотели бы вычислить. Кроме того, мы должны найти положения самих камер, которые мы можем представить как

неизвестные проекционные матрицы Pi . К этой проблеме, известной как корректировка пакетов, можно подойти с помощью стратегии оптимизации:

$$\text{минимизация}_{j,P_i} \sum_{ij} \left\| P_i X_{ij} - x_{ij} \right\|^2$$

такое, что P_i ортогонален i

Ограничение ортогональности гарантирует разумность преобразований камеры.

9.2 Теория оптимизации с ограничениями

В нашем обсуждении мы будем предполагать, что f , g и h дифференцируемы. Существуют некоторые методы, которые делают только предположения о слабой непрерывности или липшицевости, но эти методы весьма специализированы и требуют расширенного аналитического рассмотрения.

Хотя мы еще не разработали алгоритмы общей оптимизации с ограничениями, мы неявно использовали теорию таких задач при рассмотрении методов собственных значений. В частности, вспомним метод множителей Лагранжа, введенный в теореме 0.1. В этом методе критические точки f(x) , зависящие от g(x) , характеризуются как критические точки неограниченной функции множителей Лагранжа $\Lambda(x,\lambda) \quad f(x) \quad \lambda \cdot g(x)$ относительно обоих λ и x одновременно.

Эта теорема позволила нам дать вариационную интерпретацию задач на собственные значения; в более общем смысле, он дает альтернативный (необходимый, но недостаточный) критерий того, что x является критической точкой оптимизации, ограниченной равенством.

Однако простое нахождение x, удовлетворяющего ограничениям, может стать серьезной проблемой. Мы можем разделить эти проблемы, сделав несколько определений:

Определение 9.1 (допустимая точка и допустимое множество). Допустимой точкой задачи оптимизации с ограничениями является любая точка x, удовлетворяющая условиям $g(x) = 0$ и $h(x) \quad 0$. Допустимое множество — это множество всех точек x, удовлетворяющих этим ограничениям.

Определение 9.2 (Критическая точка оптимизации с ограничениями). Критическая точка оптимизации с ограничениями — это точка, удовлетворяющая ограничениям, которая также является локальным максимумом, минимумом или седловой точкой f в допустимом множестве.

Ограниченная оптимизация сложна, потому что она одновременно решает проблемы нахождения корня (ограничение $g(x) = 0$), проблемы выполнимости (ограничение $h(x) \quad 0$) и минимизацию (функция f). Помимо этого, чтобы довести наши дифференциальные методы до полной общности, мы должны найти способ добавить ограничения неравенства в систему множителей Лагранжа. Предположим, мы нашли минимум оптимизации, обозначенный x^*. Для каждого ограничения неравенства $h_i(x^*) \quad 0$ у нас есть два варианта:

- $h_i(x^*) = 0$: такое ограничение активно, что, вероятно, указывает на то, что если ограничение будет снято, оптимум может измениться.
- $h_i(x^*) > 0$: такое ограничение неактивно, то есть в окрестности x^*, если бы мы сняли это ограничение, мы все равно достигли бы того же минимума.

Конечно, мы не знаем, какие ограничения будут активны или неактивны в точке x^*, пока она не будет вычислена.

Если бы все наши ограничения были активны, то мы могли бы изменить наше ограничение h(x) 0 на равенство, не влияя на минимум. Это может мотивировать изучение следующей системы множителей Лагранжа:

$$\Lambda(x,\lambda,\mu) \quad f(x) \quad \lambda \cdot g(x) \quad \mu \cdot h(x)$$

Однако мы больше не можем сказать, что x является критической точкой Λ, потому что неактивные ограничения удалили бы приведенные выше члены. Игнорируя пока этот (важный!) вопрос, мы могли бы действовать вслепую и запросить критические точки этого нового Λ по x, которые удовлетворяют следующему:

$$0 = \quad f(x) \quad \sum_{i} \lambda_i \quad g_i(x) \quad \sum_{j} \mu_j \quad h_j(x)$$

Здесь мы выделили отдельные компоненты g и h и обработали их как скалярные функции, чтобы избежать сложных обозначений.

Хитрый прием может распространить это условие оптимальности на системы, ограниченные неравенством. Обратите внимание, что если бы мы взяли μj = 0 всякий раз, когда hj неактивен, то это удаляет нерелевантные члены из условий оптимальности. Другими словами, мы можем добавить ограничение на множители Лагранжа:

$$\mu_j h_j(x) = 0.$$

При наличии этого ограничения мы знаем, что по крайней мере одно из значений μj и hj(x) должно быть равно нулю, и, следовательно, наше условие оптимальности первого порядка по-прежнему выполняется!

До сих пор наша конструкция не различала ограничение hj(x) 0 и ограничение hj(x) 0. Если ограничение неактивно, его можно было бы снять, не влияя локально на результат оптимизации, поэтому мы рассмотрим случай, когда ограничение активно. Интуитивно 1 в этом случае мы ожидаем, что будет способ уменьшить f, нарушив ограничение. Локализованно, направление убывания f равно $-\nabla f(x^*)$, а направление убывания hj есть $-\nabla h_j(x^*)$. Таким образом, начиная с x, мы можем еще больше уменьшить f, нарушив ограничение hj(x) 0, когда $\nabla f(x^*) \cdot \nabla h_j(x^*) \quad 0$.

Конечно, с произведениями градиентов f и hj работать сложно. Однако напомним, что при нашем условии оптимальности x^* первого порядка мы получаем:

$$\nabla f(x^*) = \sum_{i} \lambda_i^* \nabla g_i(x^*) + \sum_{j \text{ активный}} \mu_j^* \nabla h_j(x^*)$$

Неактивные значения μj равны нулю и могут быть удалены. На самом деле, мы можем снять ограничения g(x) = 0, добавив к h ограничения-неравенства g(x) 0 и g(x) 0; это математическое удобство для написания доказательства, а не численный маневр. Затем, скалярное произведение с ∇h_k для любого фиксированного k показывает:

$$\sum_{j \text{ активный}} \mu_j^* \nabla h_j(x^*) \cdot \nabla h_k(x^*) = \nabla f(x^*) \cdot \nabla h_k(x^*) \quad 0$$

Векторизация этого выражения показывает, что $Dh(x^*)Dh(x^*)^\mu$ * 0. Так как $Dh(x^*)Dh(x^*)^\mu$ положительно полуопределенно инициальна, из этого * 0. Таким образом, наблюдение $\nabla f(x^*) \cdot \nabla h_j(x^*) \quad 0$ проявляется просто как следует μ тот факт, что μj 0.

Наши наблюдения можно формализовать, чтобы доказать условие оптимальности первого порядка для оптимизаций с ограничениями неравенства:

1 Не следует считать наше обсуждение формальным доказательством, так как мы не рассматриваем многих граничных случаев.

Теорема 9.1 (условия Каруша-Куна-Таккера (KKT)). Вектор x Rn является критической точкой для минимизации f при g(x) = 0 и h(x) 0, когда существуют λ Rm и μ Rp такие, что:

• 0 = f(x) i λi gi(x) j μj hj(x) («стационарность»)

• g(x) = 0 и h(x) 0 («первичная допустимость») •

μjhj(x) = 0 для всех j («дополнительная нежесткость») • μj

0 для всех j («двойственная допустимость »)

Обратите внимание, что при удалении h эта теорема сводится к критерию множителя Лагранжа.

Пример 9.5 (Простая оптимизация[2]). Предположим, мы хотим решить

максимизировать

xy так, что x + yx, y 2 2

0

В этом случае у нас не будет λ и будет три μ. Возьмем f(x, y) = xy, h1(x, y) 2 x y и h3(x, y) = y. Условия h2(x, y) = x, KKT:

Стационарность: 0 = y + μ1 μ2 0

= x + 2μ1y μ3

Первичная выполнимость: x 2 2

+ yx, y 0

Дополнительная нежесткость: μ1(2 x y μ2x 2) = 0

= 0 μ3y = 0

Двойная выполнимость: μ1, μ2, μ3 0

Пример 9.6 (Линейное программирование). Рассмотрим оптимизацию:

минимизекс б · х

такое, что Ax c

Обратите внимание, что пример 9.2 можно записать таким образом. Условия KKT для этой задачи:

Стационарность: Aμ =b

Первичная выполнимость: Ax c

Дополнительная нежесткость: μi(ai · x ci) = 0 i, где я это строка i из A

Двойная осуществимость: μ 0

Как и в случае множителей Лагранжа, мы не можем предполагать, что любой x , удовлетворяющий условиям KKT, автоматически минимизирует f с учетом ограничений, даже локально. Один из способов проверить локальную оптимальность — исследовать гессиан функции f, ограниченный подпространством Rn , в котором x может перемещаться, не нарушая ограничений; если этот «приведенный» гессиан положительно определен, то оптимизация достигла локального минимума.

[2] Из http://www.math.ubc.ca/~israel/m340/kkt2.pdf

9.3 Алгоритмы оптимизации

Тщательное рассмотрение алгоритмов оптимизации с ограничениями выходит за рамки нашего обсуждения; к счастью, существует много стабильных реализаций этих методов, и многое можно сделать как «клиент» этого программного обеспечения, а не переписывать его с нуля. Тем не менее, полезно набросать некоторые возможные подходы, чтобы получить представление о том, как работают эти библиотеки.

9.3.1 Последовательное квадратичное программирование (SQP)

Подобно BFGS и другим методам, которые мы рассматривали при обсуждении безусловной оптимизации, одной из типичных стратегий условной оптимизации является аппроксимация f, g и h более простыми функциями, решение аппроксимированной оптимизации и итерация.

Предположим, у нас есть предположение xk решения задачи условной оптимизации. Мы могли бы применить разложение Тейлора второго порядка к f и приближение первого порядка к g и h, чтобы определить следующую итерацию следующим образом:

$$x_{k+1} \quad x_k + \text{аргумент мин.}_{d} \frac{1}{2} d H f(x_k) d + \quad f(x_k) \cdot d + f(x_k)$$

$$\text{такое, что } g_i(x_k) + \quad g_i(x_k) \cdot d = 0$$

$$h_i(x_k) + \quad h_i(x_k) \cdot d \quad 0$$

Оптимизация для нахождения d имеет квадратичную цель с линейными ограничениями, для которых оптимизация может быть значительно проще с использованием одной из многих стратегий. Она известна как квадратичная программа.

Конечно, это приближение Тейлора работает только в окрестности оптимальной точки. Когда хорошее начальное предположение x0 недоступно, эти стратегии, скорее всего, потерпят неудачу.

Ограничения равенства Когда единственными ограничениями являются равенства и h удалено, квадратичная программа для d имеет условия оптимальности множителя Лагранжа, полученные следующим образом:

$$\Lambda(d,\lambda) \quad \frac{1}{2} d H f(x_k) d + \quad f(x_k) \cdot d + f(x_k) + \lambda \left(g(x_k) + Dg(x_k) d\right)$$

$$= \quad 0 = \quad d\Lambda = Hf(x_k) d + \quad f(x_k) + [Dg(x_k)]\lambda$$

В сочетании с условием равенства получается линейная система:

$$\begin{bmatrix} Hf(x_k) & [Dg(x_k)] \\ Dg(x_k) & 0 \end{bmatrix} \begin{bmatrix} d \\ \lambda \end{bmatrix} \quad \text{"="} \quad \begin{bmatrix} f(x_k) \\ g(x_k) \end{bmatrix}$$

Таким образом, каждую итерацию последовательного квадратичного программирования при наличии только ограничений-равенств можно выполнить, решая эту линейную систему на каждой итерации, чтобы получить $x_{k+1} \quad x_k + d$. Важно отметить, что приведенная выше линейная система не является положительно определенной, поэтому в больших масштабах ее может быть трудно решить.

Расширения этой стратегии работают как BFGS, и аналогичные аппроксимации работают для оптимизации без ограничений, вводя аппроксимации гессиана Hf . Стабильность также может быть введена путем ограничения расстояния, пройденного за одну итерацию.

Ограничения неравенства Существуют специализированные алгоритмы для решения квадратичных программ, а не общих нелинейных программ, и их можно использовать для генерации шагов SQP. Одна примечательная стратегия состоит в том, чтобы поддерживать «активный набор» ограничений, которые активны как минимум по отношению к d; тогда можно применить описанные выше методы с ограничениями равенства, игнорируя неактивные ограничения. Итерации оптимизации активного набора обновляют активный набор ограничений, добавляя нарушенные ограничения к активному набору и удаляя те ограничения-неравенства hj , для которых f · hj 0, как в нашем обсуждении условий KKT.

9.3.2 Барьерные методы

Другой вариант минимизации при наличии ограничений состоит в том, чтобы изменить ограничения на энергетические условия. Например, в случае ограничения равенством мы могли бы минимизировать «расширенную» цель следующим образом:

$$f\rho(x) = f(x) + \rho g(x) \quad {}^{2}_{2}$$

Обратите внимание, что принятие ρ заставит g(x) быть как можно меньше, поэтому в конечном итоге мы достигнем g(x) 0. Таким образом, барьерный метод оптимизации с ограничениями применяет итерационные методы оптимизации без ограничений к fρ и проверяет, насколько хорошо выполняются ограничения; если они не находятся в пределах заданного допуска, ρ увеличивается, и оптимизация продолжается с использованием предыдущей итерации в качестве отправной точки.

Барьерные методы просты в реализации и использовании, но они могут демонстрировать некоторые пагубные виды отказов. В частности, с ростом ρ влияние f на целевую функцию уменьшается, и гессиан функции fρ становится все менее обусловленным.

Барьерные методы также могут применяться к ограничениям неравенства. Здесь мы должны убедиться, что hi(x) 0 для всех i; типичный выбор барьерных функций может включать 1/hi(x) («обратный барьер») или log hi(x) («логарифмический барьер»).

9.4 Выпуклое программирование

Вообще говоря, методы, подобные тем, которые мы описали для оптимизации с ограничениями, практически не дают гарантий качества выходных данных. Конечно, эти методы не могут получить глобальные минимумы без хорошего начального приближения x0, а в некоторых случаях, например, когда гессиан
рядом с x* не является положительно определенной, они могут вообще не сходиться.

Есть одно заметное исключение из этого правила, которое встречается во многих важных оптимизациях: выпуклое программирование. Идея здесь состоит в том, что когда f — выпуклая функция, а само допустимое множество выпукло, то оптимизация имеет единственный минимум. Мы уже определили выпуклую функцию, но нам нужно понять, что означает, что набор ограничений

выпуклый:

Определение 9.3 (выпуклое множество). Множество S Rn называется выпуклым, если для любых x,y S точка tx + (1 t)y также принадлежит S для любого t [0, 1].

Как показано на рисунке НОМЕР, интуитивно множество является выпуклым, если его граничная форма не может изгибаться ни внутрь, ни наружу.

Пример 9.7 (Круги). Диск {x Rn : x2 1} выпукл, а единичная окружность {x Rn : x2 = 1} — нет.

Легко видеть, что выпуклая функция имеет единственный минимум, даже если эта функция ограничена выпуклой областью. В частности, если функция имеет два локальных минимума, то линия точек между этими минимумами должна давать значения f не выше, чем в конечных точках.

Сильные гарантии сходимости доступны для выпуклых оптимизаций, которые гарантируют нахождение глобального минимума, пока f выпукло, а ограничения на g и h создают выпуклое допустимое множество. Таким образом, ценным упражнением почти для любой задачи оптимизации является проверка ее выпуклости, поскольку такое наблюдение может во много раз повысить уверенность в качестве решения и шансы на успех.

Новая область, называемая дисциплинированным выпуклым программированием, пытается объединить простые правила о выпуклости для создания выпуклых оптимизаций (CITE CVX), позволяя конечному пользователю комбинировать простые выпуклые энергетические условия и ограничения, если они удовлетворяют критериям, делающим окончательную оптимизацию выпуклой. Полезные утверждения о выпуклости в этой области включают следующее:

- Пересечение выпуклых множеств выпукло; таким образом, добавление нескольких выпуклых ограничений является допустимой операцией.
- Сумма выпуклых функций выпукла.
- Если f и g выпуклы, то h(x) max{ f(x), g(x)}.
- Если f — выпуклая функция, множество {x : f(x) c} выпукло.

Такие инструменты, как библиотека CVX, помогают отделить реализацию различных выпуклых задач от их минимизации.

Пример 9.8 (выпуклое программирование).

- Неотрицательная задача наименьших квадратов в примере 9.3 является выпуклой, поскольку Ax b2 является выпуклой функция от x, а множество x 0 выпукло.
- Задача линейного программирования в примере 9.6 является выпуклой, поскольку имеет линейную цель и линейную ограничения.
- Мы можем включить x1 в задачу выпуклой оптимизации, введя переменную y. Для этого мы добавляем ограничения yi xi и yi xi для каждого i и цели i yi . Члены этой суммы не меньше |xi | и что энергия и ограничения выпуклы. Как минимум, мы должны иметь yi = |xi | поскольку мы наложили ограничение на yi |xi | и мы хотим минимизировать энергию. «Дисциплинированные» выпуклые библиотеки могут выполнять такие операции за кулисами, не раскрывая такие подстановки конечному пользователю.

Особенно важным примером выпуклой оптимизации является линейное программирование из примера 9.6. Знаменитый симплекс-алгоритм отслеживает активные ограничения, вычисляет результирующий x с помощью линейной системы и проверяет, нужно ли обновлять активное множество; приближения Тейлора не нужны, потому что целевое и допустимое множество задаются линейным механизмом. Стратегии линейного программирования внутренних точек, такие как барьерный метод, также успешно решают эти задачи. По этой причине линейные программы можно решать в огромных масштабах — до миллионов или миллиардов переменных! — и они часто появляются в таких задачах, как планирование или ценообразование.

9.5 Проблемы

- Получить симплекс?
- Двойственность линейного программирования

Глава 10

Итерационные линейные решатели

В предыдущих двух главах мы разработали стратегии для решения нового класса задач, связанных с минимизацией функции f(x) с ограничениями на x или без них. При этом мы отказались от численной линейной алгебры и, в частности, метода исключения Гаусса, согласно которому мы должны найти точное решение системы уравнений, и вместо этого обратились к итерационным схемам, которые гарантированно приближают минимум функции все лучше и лучше по мере того, как они повторять все больше и больше. Даже если мы никогда точно не найдем минимум, мы знаем, что в конечном итоге мы найдем x0 с f (x0) 0 с произвольным качеством уровней, в зависимости от количества выполненных итераций.

Теперь у нас есть повторение нашей любимой задачи из числовой линейной алгебры, решение Ax = b относительно x, но мы применяем итеративный подход, а не ожидаем найти решение в закрытой форме. Эта стратегия открывает новый класс решателей линейных систем, которые могут находить надежные приближения x за невероятно малое число итераций. Мы уже предложили, как подойти к этому в нашем обсуждении линейной алгебры, предполагая, что решения линейных систем являются среди прочего минимумами энергии Ax b. $^{2}_{2}$,

Зачем создавать еще один класс решателей линейных систем? До сих пор большинство наших прямых подходов требовали от нас представления A в виде полной матрицы размера n × n, а такие алгоритмы, как LU, QR или факторизация Холецкого, приводили около тысячи ^{3}) время. Есть два случая, о которых следует помнить для poten причин для использования итерационных схем:

1. Когда A является разреженным, такие методы, как исключение Гаусса, имеют тенденцию вызывать заполнение, а это означает, что даже) если A содержит O(n) ненулевых значений, промежуточные шаги исключения могут ввести O(n ненулевых ^{2} значений. Это свойство может быстро привести к нехватке памяти в системах линейной алгебры. Напротив, алгоритмы в этой главе требуют только того, что вы можете применить A для векторов, что может быть выполнено за время, пропорциональное количеству ненулевых значений в матрице.

2. Мы можем захотеть победить O(n ^{3}) время выполнения стандартных методов матричной факторизации. В частности, если итерационная схема может найти довольно точное решение Ax = b за несколько итераций, время выполнения можно значительно сократить.

Кроме того, обратите внимание, что многие из рассмотренных нами нелинейных методов оптимизации, в частности, зависящие от ньютоновского шага, требуют решения линейной системы на каждой итерации! Таким образом, разработка максимально быстрого решателя может иметь большое значение при реализации крупномасштабных методов оптимизации, требующих одного или нескольких линейных решений на итерацию. На самом деле, в этом случае неточное, но быстрое решение линейной системы может быть приемлемым, поскольку оно в любом случае используется в более крупной итерационной методике.

Обратите внимание, что большая часть нашего обсуждения связана с CITE, хотя наше развитие может быть несколько короче, учитывая развитие предыдущих глав.

10.1 Градиентный спуск

Мы сосредоточим наше обсуждение на решении Ax = b, где A имеет три свойства:

1. A Rn×n является квадратным
2. A симметрична, т. е. A = A
3. A положительно определена, т. е. для всех x = 0, x Ax > 0

Ближе к концу этой главы мы ослабим эти предположения. А пока заметьте, что мы можем заменить Ax = b нормальными уравнениями A Ax = A b, чтобы удовлетворить этим критериям, хотя, как мы обсуждали, эта замена может создать проблемы с числовыми условиями.

10.1.1 Получение итерационной схемы

В этом случае легко проверить, что решения Ax = b являются минимумами функции f(x) , заданной квадратичной формой

$$1 \quad f(x) \quad x \; Ax \quad bx + c \; 2 \text{ для}$$

любого c R. В частности, взятие производной от f показывает

$$f(x) = Ax \quad b,$$

и установка f (x) = 0 дает желаемый результат.

Вместо того, чтобы решать f(x) = 0 напрямую, как мы это делали в прошлом, предположим, что мы применяем стратегия градиентного спуска к этой минимизации. Вспомним базовый алгоритм градиентного спуска:

1. Вычислить направление поиска d k f(xk 1) =b Axk 1 .
2. Определить xk xk 1 + αkd k , где αk выбрано так, что f(xk) < f(xk 1)

Для общей функции f определение значения αk может быть сложной одномерной задачей «линейного поиска», сводящейся к минимизации f(xk 1 + αkd k) как функции одной переменной αk 0. Для наш конкретный выбор квадратичной формы f(x) = x Ax bx + c, однако мы можем выполнять линейный поиск в закрытой форме. В частности, определить

$$\begin{aligned} r(\alpha) &= f(x + \alpha d) \\ &= \tfrac{1}{2}(x + \alpha d)\, A(x + \alpha d) \quad b\,(x + \alpha d) + c \\ &= \tfrac{1}{2}(x\,Ax + 2\alpha x\,Ad + \alpha^2 d\,Ad) \quad bx \quad \alpha b\,d + c \text{ в силу симметрии } A \\ &= \tfrac{1}{2} d\,Ad + \alpha(x\,Ad \quad b\,d) + \text{const.} \end{aligned}$$

$$= \frac{dg}{d\alpha}(\alpha) = \alpha d\,Ad + d(Ax \quad b)$$

Таким образом, если мы хотим минимизировать g по α, мы просто выбираем

$$\alpha = \frac{d(b \quad Ax)\, dAd}{}$$

В частности, для градиентного спуска мы выбрали d к =b Axk , поэтому на самом деле αk принимает красивый вид:

$$\text{ак} = \frac{\text{дд кк}}{\text{г}_{\text{кк}\ \text{Объявление}}}$$

В итоге наша формула для линейного поиска дает следующую схему итеративного градиентного спуска для решения Ax =b в симметричном положительно определенном случае:

$$\text{д}_{\text{к}} = b - Axk\text{-}1$$

$$\text{ак} = \frac{\text{дд кк}}{\text{д}_{\text{к}\ \text{Объявление}_{\text{к}}}}$$

$$xk = xk \quad 1 + \alpha kd\ k$$

10.1.2 Конвергенция

По построению наша стратегия градиентного спуска уменьшает f(xk) при k . Тем не менее, мы не показали, что алгоритм достигает минимально возможного значения f, и мы не смогли описать, сколько итераций мы должны выполнить, чтобы достичь разумного уровня уверенности в том, что Axk b .

Одна простая стратегия для понимания сходимости алгоритма градиентного спуска для нашего выбора f состоит в том, чтобы исследовать изменение обратной ошибки от итерации к итерации.1 Предположим, что это решение, которое мы $^{*}x$ ищем, то есть Ax = b. Затем мы можем изучить соотношение ошибка от итерации к итерации:

$$Rk \quad \frac{f(xk) \quad f(x \quad)}{f(xk \quad 1) \quad f(x \quad)}$$

Очевидно, ограничение Rk < β < 1 для некоторого β показывает, что градиентный спуск сходится.

Для удобства мы можем разложить f(xk):

$$f(xk) = f(xk \quad 1 + \alpha kd\ k) \text{ по нашей итерационной схеме}$$

$$\text{"="} \frac{1}{2}(xk \quad 1 + \alpha kd\ k)\, A(xk \quad 1 + \alpha kd\ k) \quad b\,(xk \quad 1 + \alpha kd\ k) + c$$

$$= f(xk \quad 1) + \alpha kd\ k\, Axk \quad 1 + \alpha\, k\, 2\, 1\, (b \overset{1\ 2}{} \text{д}_{\text{к}}\ Ad\ k \quad \alpha k\, bd\ k \text{ по определению f}$$

$$= f(xk \quad 1) + \alpha kd \quad \text{к}\ d\ k) + \quad \frac{}{\text{к}\ 2}\alpha\, \text{г}\ k\ Ad\ k \quad \alpha k\, bd\ k, \text{ так как d к =b - Axk-1}$$

$$= f(xk \quad 1) \quad \alpha kd \quad \text{к} + \alpha\, \text{кк}\, 2 \overset{1}{} 2\, d\ \text{г}_{\text{кк}\ \text{Объявление}}$$

1 Этот аргумент представлен, например, в http://www-personal.umich.edu/~mepelman/teaching/IOE511/Handouts/511notes07-7.pdf.

$$= f(x_k \quad 1) \quad d \frac{\frac{\partial f}{\partial x_k}}{\frac{\partial^2 f}{\partial x_k^2}} \frac{\partial f}{\partial x_k} + \frac{1}{2} \frac{\left(\frac{\partial f}{\partial x_k}\right)^2}{\frac{\partial^2 f}{\partial x_k^2}} \quad \text{г} \quad A d_k \text{ по определению } \alpha_k d_k$$

$$= f(x_k \quad 1) \quad \frac{\left(\frac{\partial f}{\partial x_k} d_k\right)^2}{\text{г}_{x_k}} 1 + \frac{1}{2} \frac{\left(\frac{\partial f}{\partial x_k} d_k\right)^2}{\frac{\partial^2 f}{\partial x_k^2}} = f(x_k \quad 1) \quad \frac{\left(\frac{\partial f}{\partial x_k} d_k\right)^2}{2 \frac{\partial^2 f}{\partial x_k^2}}$$

Таким образом, мы можем вернуться к нашей дроби:

$$P_k = \frac{e(x_{k-1}) - \frac{\left(\frac{\partial f}{\partial x_k} d_k\right)^2}{2 \frac{\partial^2 f}{\partial x_k^2}} \quad f(x \quad)}{f(x_k \quad 1) \quad f(x \quad)} \text{ по нашей формуле для } f(x_k)$$

$$= 1 - \frac{\left(\frac{\partial f}{\partial x_k} d_k\right)^2}{2 d_k \quad A d_k (f(x_k \quad 1) \quad f(x \quad))}$$

Обратите внимание, что $A x \quad = b$, поэтому мы можем написать:

$$f(x_k \quad 1) \quad f(x \quad) = \quad \text{т} x_k \quad 1 A x_k \quad 1 \quad b x_k \quad 1 + c \quad 2 \quad \text{т} (x \quad) \text{ б} \quad \text{ь} x \quad 2^* + c$$

$$= \frac{1}{2} x_k \quad 1 A x_k \quad 1 \quad b x_k \quad 1 \quad \frac{1}{2} \text{ б } A^{-1} \text{ б } 2$$

$$= \frac{2}{2} \frac{1}{1} (A x_k \quad 1 \quad b) A 2^{\quad 1} (A x_k \quad 1 \quad b) \text{ в силу симметрии } A$$

$$= \frac{1}{2} d \quad A_k \quad {}^{1} d_k \text{ по определению } d_k$$

Таким образом,

$$R_k = 1 - 2 d_k \frac{\left(\frac{\partial f}{\partial x_k} d_k\right)^2}{A d_k (f(x_k \quad 1) \quad f(x \quad)) d}$$

$$= 1 - \text{д} \frac{\left(\frac{\partial f}{\partial x_k} k\right)^2}{A d_k . d_k \quad A \quad 1 d_k} \text{ нашим последним упрощением}$$

$$= 1 - \text{д} \frac{\text{г}_{x_k} \frac{\partial f}{\partial x_k}}{\text{г}} \cdot \frac{\text{г}_{x_k} \frac{\partial f}{\partial x_k}}{\text{г} \quad A \quad 1 d_k}$$

$$1 \quad \text{мин} \frac{1}{dAd \, d=1} \quad \text{мин} \frac{1}{\text{д}=1 \quad dA \quad 1d} \text{ так как это делает второй член меньше}$$

$$1 \quad \max_{d=1} dAd = \quad \text{макс} \quad {}^{1} \text{дА} \quad \text{г}$$

$$= 1 - \frac{\sigma_{\text{мин}}}{\sigma_{\max} 1} \text{ где } \sigma_{\min} \text{ и } \sigma_{\max} \text{ — минимальное и максимальное сингулярные значения } A$$

$$= 1 - \frac{}{\text{конд } A}$$

Потребовалось значительное количество алгебры, но мы доказали важный факт:

Сходимость градиентного спуска по f зависит от обусловленности A.

То есть, чем лучше обусловлен A, тем быстрее будет сходиться градиентный спуск. Кроме того, поскольку cond A 1, мы знаем, что наша вышеприведенная стратегия градиентного спуска безоговорочно сходится к x, хотя *, сходимость может быть медленной, когда A плохо обусловлен.

Рисунок НОМЕР иллюстрирует поведение градиентного спуска для хорошо и плохо обусловленных матриц. Как вы можете видеть, градиентный спуск может изо всех сил пытаться найти минимум нашей квадратичной функции f, когда собственные значения A имеют большой разброс.

10.2 Сопряженные градиенты

Напомним, что для решения Ax = b для A Rn×n использовалась 3) время. Пересмотр градиентного спуска стратегия O(n, описанная выше, мы видим, что каждая 2) времени, так как мы должны вычислить матрицу-вектор итерация требует O(n произведений между A, xk 1 и dk . Таким образом, если градиентный спуск занимает более n итераций, мы с тем же успехом можно было бы применить исключение Гаусса, которое восстановит точное решение за то же время.К сожалению, мы не можем показать, что градиентный спуск должен выполнять конечное число итераций, и на самом деле в плохо обусловленных случаях это может потребоваться огромное количество итераций, чтобы найти минимум.

По этой причине мы разработаем алгоритм, который гарантированно сходится не более чем за n шагов ( ) в сохраняя O(n) , что 3 наихудшем случае для решения линейных систем. По пути мы найдем этот алгоритм на самом деле демонстрирует лучшие свойства сходимости в целом, что делает его разумным выбором, даже если мы не запустим его до конца.

10.2.1 Мотивация

Наш вывод алгоритма сопряженных градиентов мотивирован довольно простым наблюдением. Предположим, мы знали решение x по-другому: * форму f = Ax Тогда мы можем записать нашу квадратичную

$$f(x) = x\,Ax - bx + c \text{ по определению } \frac{1}{2}(x - x^*)A(x - x^*) + x\,Ax^* - \frac{1}{2}(x^* - x)Ax^* - bx + c$$

$$= \frac{1}{2}(x - x^*)A(x - x^*) + x\,Ax^* - \frac{1}{2}x^*Ax^* - bx + c \text{, сложением и вычитанием одних и тех же членов}$$

$$= \frac{1}{2}(x - x^*)A(x - x^*) + xb - \frac{1}{2}x^*Ax^* - bx + c, \text{ поскольку } Ax^* = b$$

$$= \frac{1}{2}(x - x^*)A(x - x^*) + \text{const. так как условия } xb \text{ отменяются}$$

Таким образом, с точностью до постоянного сдвига f то же самое, $\frac{1}{2}(x - x^*)A(x - x^*)$. Конечно, мы делаем что и произведение, не знающее x, но это наблюдение показывает нам природу f ; это просто измерение от x к x относительно «A-нормы» v На самом деле, $\frac{2}{A}$ расстояния v Av.

поскольку A симметрична и положительно определена, даже если это может быть медленным для выполнения на практике, мы знаем, что ее можно разложить на множители с использованием стратегии Холецкого как A = ЛЛ. Имея в руках эту факторизацию, f принимает еще более приятный вид:

$$\phi(x) = \frac{1}{2} L(x - x^*) 2 + \text{конст. } 2$$

Поскольку L является обратимой матрицей, эта норма действительно является мерой расстояния между x и x * . –

Определите y L x и y * L x * Затем с этой новой точки зрения мы минимизируем 2 . Конечно, если f(y) = . – y - y 2 бы мы действительно могли добраться до этого с помощью факторизации Холецкого, оптимизируя f было бы чрезвычайно легко, но чтобы вывести схему этой минимизации без L, мы рассмотрим возможность минимизации f, используя только линейный поиск, полученный в § 10.1.1. –

Мы делаем простое наблюдение о минимизации нашей упрощенной функции f с помощью такой стратегии. egy, показанный на РИСУНКЕ НОМЕР: –

Предложение 10.1. Предположим, что {w 1, . . . , wn } ортогональны в Rn . Затем f минимизируется не более чем за n шагов путем поиска строки в направлении w 1, затем в направлении w 2 и так далее.

Доказательство. Возьмем столбцы матрицы Q Rn×n в качестве векторов w i ; Q — ортогональная матрица. Поскольку Q f(y) = y мы можем написать 2 ; другими словами, мы поворачиваем так, чтобы w 1 был первым стандартным базисным вектором, w 2 — вторым и так далее. Если мы напишем z Qy и после второй итерации * г Qy и *, то, очевидно, после первой итерации должно быть z1 = z * 1 , z2 = z2 , * так далее. После n шагов мы достигаем zn = z * n , что дает желаемый результат. □

Таким образом, оптимизацию f всегда можно выполнить с помощью n строковых поисков, если эти поиски осуществляются в ортогональных направлениях. –

Все, что мы сделали для перехода от f к f, — это повернули координаты, используя L . Такое линейное преобразование переводит прямые линии в прямые линии, поэтому выполнение поиска строки f вдоль некоторого вектора w эквивалентно равносильно поиску строки вдоль (L) 1w нашей исходной квадратичной функции f . Наоборот, если мы делаем n строковых поисков по f в направлениях vi , таких что L vi w i ортогональны, то по предложению 10.1 мы должны найти x * . Обратите внимание, что запрос w i · w = 0 аналогичен заданию j

$$0 = w_i \cdot w_j = (L v_i) (L v_j) = v_i (LL) v_j = v_i A v_j .$$

Мы только что доказали важное следствие предложения 10.1. Определим сопряженные векторы следующим образом:

Определение 10.1 (A-сопряженные векторы). Два вектора v, w являются A -сопряженными, если v Aw = 0.

Затем, основываясь на нашем обсуждении, мы показали:

Предложение 10.2. Предположим, {v1, . . . ,vn} A-сопряжены. Затем f минимизируется не более чем за n шагов путем поиска строки в направлении v1, затем в направлении v2 и так далее.

На высоком уровне алгоритм сопряженных градиентов просто применяет это предложение, генерируя и ища по A-сопряженным направлениям, а не перемещаясь по f . Обратите внимание, что этот результат может показаться несколько нелогичным: мы не обязательно движемся по направлению наискорейшего спуска, а скорее просим, чтобы наш набор направлений поиска удовлетворял глобальному критерию, чтобы убедиться, что мы не повторяем работу. Эта установка гарантирует сходимость за конечное число итераций и признает структуру f в терминах f, обсуждавшуюся выше.

Напомним, что мы мотивировали A-сопряженные направления тем, что заметили, что они ортогональны после применения L из факторизации A = LL. С этой точки зрения мы имеем дело с двумя скалярными произведениями: xi · xj и yi ·yj (L xi) · (L xj) = x i LLxj = x i Axj . Эти два продукта будут фигурировать в нашем последующем обсуждении в равных количествах, поэтому мы обозначаем «A-внутренний продукт» как

$$\langle u,v \rangle_A \equiv (L u) \cdot (L v) = u A v.$$

10.2.2 Субоптимальность градиентного спуска

На данный момент мы знаем, что если мы сможем найти n A-сопряженных направлений поиска, мы сможем решить Ax =b за n шагов с помощью линейного поиска вдоль этих направлений. Остается найти стратегию, позволяющую максимально эффективно находить эти направления. Для этого мы рассмотрим еще одно свойство алгоритма градиентного спуска, которое вдохновит на более совершенный подход.

Предположим, что мы находимся в точке xk во время метода итеративного поиска строки на f(x); мы будем называть направление наискорейшего спуска функции f в точке xk невязкой rk b Axk . Мы не можем решить выполнить линейный поиск вдоль rk, как при градиентном спуске, поскольку направления градиента не обязательно являются A-сопряженными. Итак, слегка обобщая, мы найдем xk+1 поиском по строке по еще не определенному направлению vk+1 .

Из нашего вывода градиентного спуска мы должны выбрать xk+1 = xk + αk+1vk+1 , где αk+1 определяется выражением

$$\text{ак+1} = \frac{\text{в}_{k+1}\, r_k}{\text{v}_{k+1}\text{Avk+1}}.$$

Применяя это расширение xk+1 , мы можем написать альтернативную формулу обновления для остатка:

$$\begin{aligned} r_{k+1} &= b \quad Ax_{k+1} = b \\ &\quad A(x_k + \alpha_{k+1}v_{k+1}) \text{ по определению } x_{k+1} = (b \quad Ax_k) \\ &\quad \alpha_{k+1}Av_{k+1} = r_k \quad \alpha_{k+1}Av_{k+1} \text{ по} \\ &\quad \text{определению } \text{офк} \end{aligned}$$

Эта формула верна независимо от нашего выбора vk+1 и может быть применена к любому итеративному методу поиска строк.

Однако в случае градиентного спуска мы выбрали vk+1 rk . Этот выбор дает рекуррентное соотношение:

$$r_{k+1} = r_k \quad \alpha_{k+1}Ar_k .$$

Эта простая формула приводит к поучительному утверждению:

Предложение 10.3. При выполнении градиентного спуска на f , span{r0, . . . ,rk} = span{r0, Ar0, . . . , А кр0}.

Доказательство. Это утверждение следует по индукции из нашей формулы для rk+1 выше. □

Раскрываемая нами структура начинает очень походить на методы Крылова, упомянутые в главе 5: Это не ошибка!

Градиентный спуск достигает xk , двигаясь по r0, затем по r1 и так далее по rk . Таким образом, в итоге мы знаем, что итерация xk градиентного спуска на f лежит где-то в плоскости x0 + A k 1r0} по предложению 10.3. К сожалению, что если мы запустим градиентный спуск, итерация xk это span {r0,r1, . . . ,rk 1} = x0 + span {r0, Ar0, . . . , неверно, будет оптимальной в этом подпространстве. Иными словами, в общем случае может быть так, что

$$x_0 = \min_{v \in \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}} f(x_0 + v) \arg$$

В идеале замена этого неравенства на равенство гарантирует, что генерация xk+1 из xk не «отменит» никакой работы, проделанной во время итераций с 1 по k 1.

Если мы пересмотрим наше доказательство предложения 10.1 с учетом этого факта, мы сможем сделать наблюдение, подсказывающее, как мы могли бы использовать сопряжение для улучшения градиентного спуска. В частности, когда z_i переключается на z^*, оно никогда не меняет значение в будущей итерации. Другими словами, после поворота от $z \kappa i$, x выполняется следующее предложение:

Предложение 10.4. Возьмем x_k за k-ю итерацию процесса из предложения 10.1 после поиска вдоль v_k. Затем,

$$x_k \quad x_0 = \arg\min_{v \quad \text{span}_{\{v_1,\ldots,v_k\}}} e(x_0 + v)$$

Таким образом, в лучшем из возможных миров, пытаясь превзойти градиентный спуск, мы могли бы надеяться, что A найдет A-сопряженные направления $\{v_1, \ldots, v_n\}$ такие, что охватывают $\{v_1, \ldots, v_k\} = \text{span}\{r_0, Ar_0, \ldots,$ для каждого$k \quad 1r_0\}$ k; тогда наша итерационная схема гарантированно будет работать не хуже, чем градиентный спуск на любой заданной итерации. Но мы жадно хотим сделать это без ортогонализации или хранения более чем конечного числа векторов за раз.

10.2.3 Генерация A-сопряженных направлений

Конечно, для любого набора направлений мы можем сделать их A-ортогональными, используя такой метод, как ортогонализация Грама Шмидта. К сожалению, ортогонализация $\{r_0, Ar_0, \ldots\}$ найти набор направлений поиска дорого и потребовало бы от нас ведения полного списка направлений v_k; эта конструкция, вероятно, превысила бы требования по времени и памяти даже при исключении Гаусса.
Мы покажем одно заключительное наблюдение о Граме-Шмидте, которое делает сопряженные градиенты управляемыми, создавая сопряженные направления без дорогостоящего процесса ортогонализации.

Игнорируя эти проблемы, мы могли бы написать «метод сопряженных направлений» следующим образом:

$$\text{Обновить направление поиска (неверный шаг Грама-Шмидта): } v_k = A \quad k \quad 1 \quad \sum_{я<к} \frac{A\,k \quad 1r_0, v_i A\, r_0}{v_i, v_i A} v_i$$

$$\text{Поиск строки: } \alpha_k = \frac{в_k\, r_k \quad 1}{в_k \; Авк}$$

$$\text{Обновление оценки: } x_k = x_k \quad 1 + \alpha_k v_k$$

$$\text{Обновить остаток: } r_k = r_k \quad 1 \quad \alpha_k A v_k$$

Здесь мы вычисляем k-е направление поиска v_k, просто проецируя $v_1, \ldots, v_k \quad 1$ из вектора $A\,k \quad 1r_0$. Этот алгоритм, очевидно, обладает свойством $\text{span}\{v_1, \ldots, v_k\} = \text{span}\{r_0, Ar_0, \ldots, A\,k \quad 1r_0\}$ предложено в §10.2.2, но имеет две проблемы:

1. Подобно итерации мощности для собственных векторов, мощность $A\,k \quad 1r_0$, вероятно, будет выглядеть в основном как первый собственный вектор A, делая проекцию все более и более плохо обусловленной.

2. Мы должны сохранить $v_1, \ldots, v_k \quad 1$ для вычисления v_k; таким образом, каждая итерация этого алгоритма требует больше памяти и времени, чем предыдущая.

Мы можем решить первую проблему относительно простым способом. В частности, прямо сейчас мы проецируем предыдущие направления поиска из $A\,k \quad 1r_0$, но на самом деле мы можем проецировать предыдущие направления из любого вектора w при условии, что

$$k \quad 1\, w \quad \text{span}\{r_0, Ar_0, \ldots, A \quad r_0\} \backslash \text{span}\{r_0, Ar_0, \ldots, A^{k \quad 2} r_0\},$$

то есть до тех пор, пока w имеет некоторый компонент в новой части пространства.

Альтернативным выбором w с этим свойством является невязка rk 1 . Это свойство следует из остаточного обновления rk = rk 1 αkAvk ; в этом выражении мы умножаем vk на A, вводя нужную нам новую мощность A. Этот выбор также более точно имитирует алгоритм градиентного спуска, который взял vk =rk 1 . Таким образом, мы можем немного обновить наш алгоритм:

Обновить направление поиска (плохой Грам-Шмидт на невязке): $v_k = r_{k-1} - \sum_{i<k} \frac{r_{k-1}, v_i A}{v_i, v_i A} v_i$

Поиск строки: $\alpha_k = \frac{v_k r_{k-1}}{v_k A v_k}$

Обновление оценки: $x_k = x_{k-1} + \alpha_k v_k$

Обновить остаток: $r_k = r_{k-1} - \alpha_k A v_k$

Теперь мы не выполняем арифметические действия с плохо обусловленным вектором A k 1r0 , но все еще имеем описанную выше проблему «памяти».

На самом деле удивительное наблюдение по поводу шага ортогонализации, описанного выше, состоит в том, что большинство слагаемых в сумме равны нулю! Это удивительное наблюдение позволяет выполнять каждую итерацию сопряженных градиентов без увеличения использования памяти. Мы зафиксируем этот результат в предложении:

Предложение 10.5. В приведенном выше методе «сопряженного направления» $r_k, vA = 0$ для всех $< k$.

Доказательство. Поступим индуктивно. Для базового случая $k = 1$ доказывать нечего , поэтому предположим, что $k > 1$ и что результат верен для всех $k < k$. По формуле остаточного обновления мы знаем:

$$r_k, vA = r_{k-1}, vA \quad \alpha_k Av_k, vA = r_{k-1}, vA \quad \alpha_k v_k, AvA,$$

где второе равенство следует из симметрии A.

Во-первых, предположим, что $< k$ 1. Тогда первый член указанной выше разности равен нулю по индукции. Кроме того, по построению Av span {v1, . . . ,v+1}, поэтому, поскольку мы построили наши направления поиска как A-сопряженные, мы знаем, что второй член также должен быть равен нулю.

Чтобы завершить доказательство, рассмотрим случай = k 1. Используя формулу обновления невязки, мы знаем:

$$Av_{k-1} = \frac{1}{\alpha_{k-1}} (r_{k-2} \quad r_{k-1})$$

Предварительное умножение byrk показывает:

$$r_k, v_{k-1}A = \frac{1}{\alpha_{k-1}} p_k \ (r_{k-2} \quad r_{k-1})$$

Разность rk 2 rk 1 живет в промежутке {r0, Ar0, . . . , A k 1r0} по формуле обновления невязки. Предложение 10.4 показывает, что xk оптимален в этом подпространстве. Так как rk = f(xk), отсюда следует A k 1r0}, которое мы должны иметь rk span {r0, Ar0, . . . , в поскольку в противном случае существовало бы направление, подпространстве двигаться от xk к уменьшению f . В частности, это показывает скалярное произведение выше rk ,vk 1A = 0, что и требовалось. □

Таким образом, наше доказательство выше показывает, что мы можем найти новое направление vk следующим образом:

$$v_k = r_{k-1} - \sum_{i<k} \frac{r_{k-1}^T A v_i}{v_i^T A v_i} v_i \text{ по формуле Грама-Шмидта}$$

$$= r_{k-1} - \frac{r_{k-1}^T A v_{k-1}}{v_{k-1}^T A v_{k-1}} v_{k-1}, \text{ так как остальные члены равны нулю}$$

Поскольку суммирование по i исчезает, стоимость вычисления vk не зависит от k.

10.2.4 Формулировка алгоритма сопряженных градиентов

Теперь, когда у нас есть стратегия, которая дает A-сопряженные направления поиска с относительно небольшими вычислительными затратами, мы просто применяем эту стратегию, чтобы сформулировать алгоритм сопряженных градиентов. В частности, предположим, что x0 является начальным предположением решения задачи Ax = b, и возьмем r0 b Ax0. Для удобства возьмем v0 0. Затем мы итеративно обновим xk 1 до xk , используя серию шагов для k = 1, 2, . . .:

$$\text{Обновить направление поиска: } v_k = r_{k-1} - \frac{r_{k-1}^T A v_{k-1}}{v_{k-1}^T A v_{k-1}} v_{k-1}$$

$$\text{Поиск строки: } \alpha_k = \frac{v_k^T r_{k-1}}{v_k^T A v_k}$$

$$\text{Обновление оценки: } x_k = x_{k-1} + \alpha_k v_k$$

$$\text{Обновить остаток: } r_k = r_{k-1} - \alpha_k A v_k$$

Эта итеративная схема является лишь незначительной корректировкой алгоритма градиентного спуска, но имеет много желаемых свойств по конструкции:

- f(xk) ограничено сверху значением k-й итерации градиентного спуска.
- Алгоритм сходится к x^* за n шагов
- На каждом шаге итерация xk является оптимальной в подпространстве, натянутом на первые k направлений поиска.

В интересах выжать максимальное численное качество из сопряженных градиентов, мы можем попытаться упростить численные выражения приведенных выше выражений. Например, если мы подставим обновление направления поиска в формулу для αk , по ортогональности мы можем написать:

$$\alpha_k = \frac{p_{k-1}^T r_{k-1}}{v_k^T A v_k}$$

Теперь числитель этой дроби гарантированно неотрицательный без числовой точности.
проблемы.

Точно так же мы можем определить константу βk , чтобы разделить обновление направления поиска на два этапа:

$$\beta_k = -\frac{r_{k-1}^T A v_{k-1}}{v_{k-1}^T A v_{k-1}}$$

$$v_k = r_{k-1} + \beta_k v_{k-1}$$

Мы можем упростить нашу формулу для βk :

$$\beta_k = -\frac{r_{k-1}^T v_{k-1}^T A}{v_{k-1}^T A v_{k-1}}$$

$$= -\frac{r_{k-1}^T A v_{k-1}}{v_{k-1}^T A v_{k-1}} \quad \text{по определению } v$$

$$= -\frac{r_{k-1}^T (r_{k-2} - r_{k-1})}{\alpha_{k-1} v_{k-1}^T A v_{k-1}} \quad \text{так как } r_k = r_{k-1} - \alpha_k A v_k$$

$$\overset{"="}{=} \frac{p_{k-1}^T r_{k-1}}{\alpha_{k-1} v_{k-1}^T A v_{k-1}} \quad \text{вычислением ниже}$$

$$\overset{"="}{=} \frac{p_{k-1}^T r_{k-1}}{p_{k-2}^T r_{k-2}} \quad \text{по нашей последней формуле для } \alpha_k$$

Это выражение показывает, что βk 0, свойство, которое могло бы не сохраняться после проблем с численной точностью. У нас есть одно оставшееся вычисление ниже: (rk 2 αk 1Avk 1) по

$p_{k-2}^T r_{k-1} = r_{k-2}^T$ нашей формуле остаточного обновления

$$= r_{k-2}^T - \frac{p_{k-2}^T}{v_{k-1}^T A v_{k-1}} r_{k-2}^T A v_{k-1} \quad \text{по нашей формуле для } \alpha_k$$

$$= r_{k-1}^T - \frac{p_{k-2}^T}{v_{k-1}^T A v_{k-1}} v_{k-1}^T A v_{k-1} \quad \text{обновлением для vk и A-сопряженностью vk 's k 2 rk 2 v}$$

= 0, как нужно.

С этими упрощениями у нас есть альтернативная версия сопряженных градиентов:

Обновить направление поиска: $\beta_k = \frac{p_{k-1}^T r_{k-1}}{p_{k-2}^T r_{k-2}}$

$$v_k = r_{k-1} + \beta_k v_{k-1}$$

Поиск строки: $\alpha_k = \frac{p_{k-1}^T r_{k-1}}{v_k^T A v_k}$

Обновление оценки: $x_k = x_{k-1} + \alpha_k v_k$

Обновить остаток: $r_k = r_{k-1} - \alpha_k A v_k$

По числовым причинам иногда вместо использования формулы обновления forrk рекомендуется использовать формулу остатка rk =b Axk . Эта формула требует дополнительного умножения матрицы на вектор, но исправляет числовой «дрейф», вызванный округлением с конечной точностью. Обратите внимание, что нет необходимости хранить длинный список предыдущих остатков или направлений поиска: сопряженные градиенты занимают постоянный объем памяти от итерации к итерации.

10.2.5 Условия сходимости и остановки

По построению алгоритм сопряженных градиентов (CG) гарантированно сходится не медленнее, чем градиентный спуск по f , при этом не сложнее в реализации и имеет ряд других особенностей.

положительные свойства. Подробное обсуждение сходимости компьютерной графики выходит за рамки нашего обсуждения, но в целом алгоритм лучше всего ведет себя на матрицах с равномерно распределенными собственными значениями в небольшом диапазоне. Одна грубая оценка, параллельная нашей оценке в §10.1.2, показывает, что алгоритм CG удовлетворяет:

$$\frac{f(x_k) - f(x^*)}{f(x_0) - f(x^*)} \leq 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k$$

где к cond A. В более общем смысле количество итераций, необходимых для сопряженного градиента для достижения заданного значения ошибки, обычно может быть ограничено функцией к, тогда как границы сходимости градиентного спуска пропорциональны к.

Мы знаем, что сопряженные градиенты гарантированно сходятся к x ровно за n шагов, но когда n велико, может быть предпочтительнее остановиться раньше. Фактически, формула для βk будет делить на ноль, когда невязка становится очень короткой, что может вызвать проблемы с численной точностью вблизи минимума f. Таким образом, на практике ЦГ обычно останавливается, когда отношение rk/r0 достаточно мало.

10.3 Предварительная подготовка

Теперь у нас есть две мощные итерационные схемы для нахождения решений Ax = b, когда A симметрично и положительно определено: градиентный спуск и сопряженные градиенты. Обе стратегии сходятся безоговорочно, а это означает, что независимо от начального предположения x0 при достаточном количестве итераций мы получим сколь угодно близкое к истинному решению x ровно за конечное число $*$; на самом деле, сопряженные градиенты гарантируют, что мы достигнем x^* итераций. Конечно, время, необходимое для достижения решения Ax = b для обоих этих методов, прямо пропорционально количеству итераций, необходимых для достижения x в пределах приемлемого допуска. Таким образом, имеет смысл $*$ максимально настроить стратегию, чтобы свести к минимуму количество итераций для сходимости.

С этой целью мы замечаем, что мы можем охарактеризовать скорость сходимости обоих алгоритмов и многих других связанных итерационных методов с точки зрения числа обусловленности cond A. То есть, чем меньше значение cond A, тем меньше времени это займет. решить Ax = b. Обратите внимание, что эта ситуация несколько отличается для исключения Гаусса, которое требует одинакового количества шагов независимо от A; другими словами, обусловленность A влияет не только на качество результатов итерационных методов, но и на скорость, с которой x $*$ приближается.

Конечно, для любой обратимой матрицы P решение PAx = Pb эквивалентно решению Ax = b. Хитрость, однако, заключается в том, что число обусловленности PA не обязательно должно быть таким же, как у A; в (недостижимой) крайности, конечно, если бы мы взяли $P = A^{-1}$, мы полностью устранили бы проблемы с обусловливанием! В более общем случае предположим, что P A cond A, и поэтому может быть целесообразно применить P перед решением A^{-1}. Тогда мы ожидаем cond PA линейной системы. В этом случае мы будем называть P предобуславливателем.

Хотя идея предварительной обработки кажется привлекательной, остаются два вопроса:

1. Хотя A может быть симметричным и положительно определенным, произведение PA в общем случае не будет обладать эти свойства.
2. Нужно найти P A это легче вычислить, чем A^{-1} себя.

Мы рассматриваем эти вопросы в следующих разделах.

10.3.1 CG с предварительным кондиционированием

Мы сосредоточим наше обсуждение на сопряженных градиентах, поскольку они обладают лучшими свойствами сходимости, хотя большинство наших конструкций довольно легко применимы и к градиентному спуску. С этой точки зрения, если мы посмотрим на наши конструкции в § 10.2.1, станет ясно, что наша конструкция CG сильно зависит как от симметрии, так и от положительной определенности A, поэтому запуск CG на PA обычно не будет сходиться из коробки.

Предположим, однако, что предобуславливатель P сам по себе симметричен и положительно определен. Это разумное предположение, поскольку A должен удовлетворять этим свойствам. Тогда мы снова можем написать a = EE. Делаем следующее Факторизация Холецкого обратного P Предложение ^{1} наблюдение:

10.6. Номер состояния PA такой же, как у E-1AE-.

Доказательство. Мы показываем, что PA и E 1AE имеют одинаковые сингулярные значения; число обусловленности — это отношение максимального к минимальному сингулярному значению, так что этого утверждения более чем достаточно. В частности, ясно, что E 1AE симметричен и положительно определен, поэтому его собственные векторы являются его сингулярными значениями. Итак, предположим, что E 1AE x = λx. Мы знаем P ^{1} = EE, поэтому P = E E ^{1}. Таким образом, если мы предварительно умножим обе части выражения нашего собственного вектора на E , мы найдем PAE x = λE x.
Определение y E x показывает, что PAy = λy. Таким образом, PA и E -1AE- имеют полные собственные пространства и идентичные собственные значения. □

Это предложение подразумевает, что если мы выполним CG на симметричной положительно определенной матрице E 1AE , мы получим те же преимущества кондиционирования, которые были бы у нас, если бы мы могли повторять PA. Как и при доказательстве предложения 10.6, мы могли бы выполнить наше новое решение для y = E x в два этапа:

1. Решите E 1AE y = E 1b.
2. Решите x = E y.

Нахождение E было бы неотъемлемой частью этой стратегии, но, вероятно, это сложно, но мы вскоре докажем, что в этом нет необходимости.

Игнорируя вычисление E, мы могли бы выполнить шаг 1, используя CG, следующим образом:

$$\text{Обновить направление поиска: } \beta_k = \frac{p_{k\ 1}\, r_{k\ 1}}{p_{k\ 2}\, r_{k\ 2}}$$

$$v_k = r_{k\ 1} + \beta_k v_{k\ 1}$$

$$\text{Поиск строки: } \alpha_k = \frac{p_{k\ 1}\, r_{k\ 1}}{v_{k}\ E\text{-}1AE\text{-}v_k}$$

$$\text{Обновить оценку: } y_k = y_{k\ 1} + \alpha_k v_k$$

$$\text{Обновить невязку: } r_k = r_{k\ 1} \alpha_k E\ 1AE\ v_k$$

Эта итерационная схема будет сходиться в соответствии с условиями нашей матрицы E 1AE .

Определим $\tilde{r}_k$ Erk , $\tilde{v}_k$ E vk и x_k Eyk . Если мы вспомним соотношение P = E E, ^{1}, мы можем перепишем нашу итерацию предварительно обусловленных сопряженных градиентов, используя эти новые переменные:

$$\text{Обновить направление поиска: } \beta_k = \frac{\tilde{r}_{k\ 1} P \tilde{r}_{k\ 1}}{\tilde{r}_{k\ 2} P \tilde{r}_{k\ 2}}$$

$$\tilde{v}_k = Pr\tilde{}_k \quad 1 + \beta k \tilde{v}_k \quad 1$$

$$\text{Поиск строки: } \alpha k = \frac{r_{k \quad 1 \Pi K\text{-}1}}{\tilde{v}_{\kappa} A\tilde{v}_k}$$

$$\text{Обновить оценку: } xk = xk \quad 1 + \alpha k \tilde{v}_k$$

$$\text{Обновить остаток: } \tilde{r}_k = r\tilde{}_k \quad 1 \quad \alpha k A\tilde{v}_k$$

Эта итерация не зависит от факторизации Холецкого P, выполненной с использованием ^{1} вообще, а вместо этого может быть исключительно приложений P и A. Легко видеть, что схема xk x пользуется преимуществами *, так на самом деле это предобусловливания без необходимости факторизации предобуславливателя.

В качестве примечания, еще более эффективное предварительное обусловливание можно выполнить, заменив A на PAQ для второй матрицы Q, хотя для применения этой второй матрицы потребуются дополнительные вычисления. Этот пример представляет собой распространенный компромисс: если само предварительное обусловливание требует слишком много времени для применения в одной итерации CG или другого метода, оно может не стоить уменьшенного количества итераций.

10.3.2 Общие предварительные условия

Поиск хороших предобуславливателей на практике — это не только наука, но и искусство. Поиск наилучшего зависит от аппроксимация P of A и т.д. ^{1} структуры A, конкретного приложения и
Однако даже грубые приближения могут значительно помочь сходимости, поэтому редко появляются приложения компьютерной графики, в которых не используется предобуславливатель.

Наилучшая стратегия для формулирования P часто зависит от приложения, и интересная проблема инженерной аппроксимации включает в себя разработку и тестирование различных P для наилучшего предобуславливателя.
Две общие стратегии приведены ниже:

- Диагональный (или «Якоби») преобуславливатель просто принимает P за матрицу, полученную инвертированием диагональных элементов A; то есть P — диагональная матрица с элементами 1/aii. Эта стратегия может смягчить неравномерное масштабирование от строки к строке, которое является частой причиной плохой обработки.

- Разреженный аппроксимативный обратный предобуславливатель формулируется путем решения подзадачи minP S AP IFro, где P ограничен набором S матриц, по которым оптимизация такой задачи является менее сложной. Например, общим ограничением является предписание шаблона разреженности для P, например, только ненулевые значения по диагонали или там, где A имеет ненулевые значения.

- Факторы неполного предобуславливателя Холецкого A L L * а затем приближается к A ^{1} путем решения соответствующих задач прямой и обратной замены. Например, популярная стратегия включает в себя выполнение шагов факторизации Холецкого, но сохранение результата только в позициях (i, j) , где aij = 0.

- Ненулевые значения в A можно рассматривать как граф, а удаление ребер в графе или группировка узлов может привести к разъединению различных компонентов; результирующая система является блочно-диагональной после перестановки строк и столбцов и, таким образом, может быть решена с использованием последовательности меньших решений. Такая стратегия декомпозиции области может быть эффективна для линейных систем, возникающих из дифференциальных уравнений, подобных рассмотренным в главе НОМЕР.

Некоторые предобуславливатели поставляются с ограничениями, описывающими изменения в обусловливании A после замены его на PA, но по большей части это эвристические стратегии, которые следует протестировать и уточнить.

10.4 Другие итерационные схемы

Алгоритмы, которые мы подробно разработали в этой главе, применяются для решения Ax = b, когда A является квадратным, симметричным и положительно определенным. Мы сосредоточились на этом случае, потому что он очень часто встречается на практике, но бывают случаи, когда A асимметрично, неопределенно или даже прямоугольно. Выведение итерационных алгоритмов в каждом случае выходит за рамки нашего обсуждения, поскольку многие из них требуют некоторого специализированного анализа или расширенной разработки, но мы суммируем здесь некоторые методы на высоком уровне (CITE EACH):

- Методы расщепления разлагают A = M N и отмечают, что Ax = b эквивалентно Mx = Nx +b. Если M легко инвертировать, то схему с фиксированной точкой можно вывести, написав Mxk = Nxk 1 +b (CITE); эти методы просты в реализации, но имеют сходимость в зависимости от спектра матрицы G = M 1N и, в частности, могут расходиться, когда спектральный радиус G больше единицы. Одним из популярных вариантов M является диагональ A. Такие методы, как последовательная чрезмерная релаксация (SOR), взвешивают эти два термина для лучшей сходимости.

- Метод остатка уравнения нормального сопряженного градиента (CGNR) просто применяет алгоритм CG к нормальным уравнениям A Ax = A b. Этот метод прост в реализации и гарантирует сходимость до тех пор, пока A имеет полный ранг, но сходимость может быть медленной из-за плохой обработки AA, как обсуждалось в главе NUMBER.

- Метод ошибки уравнения нормали сопряженного градиента (CGNE) аналогичным образом решает AAy =b; затем решение Ax =b есть просто A y.

- Такие методы, как MINRES и SYMMLQ, применяются к симметричным, но не обязательно положительно определенным матрицам A путем замены нашей квадратичной формы f(x) на g(x) b Ax2; эта функция g минимизируется на решениях Ax =b независимо от определенности A.

- Учитывая плохую обусловленность CGNR и CGNE, алгоритмы LSQR и LSMR также минимизируют g(x) с меньшим количеством допущений относительно A, в частности позволяя решать системы наименьших квадратов.

- Обобщенные методы, включая GMRES, QMR, BiCG, CGS и BiCGStab, решают Ax =b с единственной оговоркой, что A является квадратным и обратимым. Они оптимизируют аналогичные энергии, но часто должны хранить больше информации о предыдущих итерациях и, возможно, должны учитывать промежуточные матрицы, чтобы гарантировать сходимость с такой общностью.

- Наконец, методы Флетчера-Ривза, Полака-Рибьера и другие возвращаются к более общей задаче минимизации неквадратичной функции f, применяя шаги сопряженного градиента для поиска новых направлений поиска линии. Функции f, которые хорошо аппроксимируются квадратичными уравнениями, могут быть очень эффективно минимизированы с использованием этих стратегий, хотя они не обязательно используют гессиан; например, метод Флетчера-Ривза просто заменяет невязку в итерациях компьютерной графики отрицательным градиентом - f . Можно охарактеризовать сходимость этих методов, когда они сопровождаются достаточно эффективными стратегиями поиска линий.

Многие из этих алгоритмов почти так же легко реализовать, как компьютерную графику или градиентный спуск, и существует множество реализаций, которые просто требуют ввода A и b. Многие из перечисленных выше алгоритмов требуют применения как A, так и A, что в некоторых случаях может быть технической проблемой. Как правило, чем более обобщенным является метод, то есть чем меньше допущений метод делает в отношении структуры матрицы A, тем больше итераций может потребоваться, чтобы компенсировать это отсутствие допущений. При этом не существует жестких и быстрых правил, просто взглянув на наиболее успешную итеративную схему, хотя существует ограниченное теоретическое обсуждение, сравнивающее преимущества и недостатки каждого из этих методов (CITE).

10.5 Проблемы

- Получить CGNR и/или CGNE
- Получение МИНРЕС
- Получить Флетчер-Ривз
- Слайд 13 из http://math.ntnu.edu.tw/~min/matrix_computation/Ch4_Slide4_CG_2011.pdf

Часть IV

Функции, производные и интегралы

Глава 11

Интерполяция

До сих пор мы вывели методы для анализа функций f , например, для нахождения их минимумов и корней. Вычисление f(x) при конкретном x Rn может быть дорогостоящим, но фундаментальное допущение методов, разработанных нами в предыдущих главах, состоит в том, что мы можем получить f(x) , когда захотим, независимо от x.

Есть много контекстов, когда это предположение нереалистично. Например, если мы делаем снимок с помощью цифровой камеры, мы получаем сетку n × m значений цвета пикселей, отображающую континуум света, попадающего в объектив камеры. Мы можем думать о фотографии как о непрерывной функции от положения изображения (x, y) до цвета (r, g, b), но на самом деле мы знаем значение изображения только в точках, разделенных нм на плоскости изображения. Точно так же в машинном обучении и статистике часто нам даются образцы функции только в точках, где мы собирали данные, и мы должны интерполировать их, чтобы получить значения в другом месте; в медицинских учреждениях мы можем наблюдать за реакцией пациента на разные дозы лекарства, но можем только предсказать, что произойдет при дозировке, которую мы явно не пробовали.

В этих случаях, прежде чем мы сможем минимизировать функцию, найти ее корни или даже вычислить значения f(x) в произвольных точках x, нам нужна модель для интерполяции f(x) ко всему Rn (или некоторому его подмножеству) при заданном набор выборок f(xi). Конечно, методы, решающие эту проблему интерполяции, по своей сути являются приближенными, поскольку мы не знаем истинных значений f, поэтому вместо этого мы стремимся к тому, чтобы интерполируемая функция была гладкой и служила «разумным» предсказанием значений функции.

В этой главе мы будем предполагать, что значения f(xi) известны с полной уверенностью; в этом случае мы могли бы также думать о проблеме как о расширении f на оставшуюся часть домена, не нарушая значения ни в одном из входных местоположений. В главе НОМЕР (НАПИШИТЕ МНЕ В 2014 ГОДУ) мы рассмотрим задачу регрессии, в которой значение f(xi) известно с некоторой неопределенностью, и в этом случае мы можем полностью отказаться от сопоставления f(xi) в пользу того, чтобы сделать f более гладким. .

11.1 Интерполяция в одной переменной

Прежде чем рассматривать наиболее общий случай, мы разработаем методы интерполяции функций одной переменной f : R R. В качестве входных данных мы возьмем набор из k пар (xi , yi) с предположением f(xi) = yi ; наша задача — найти f(x) для x {x1, . . . , xк}.

Наша стратегия в этом и других разделах будет черпать вдохновение из линейной алгебры, записывая f(x) в базисе. То есть набор всех возможных функций f : R R слишком велик, чтобы с ним можно было работать, и включает в себя множество функций, которые непрактичны в вычислительной среде. Таким образом, мы упрощаем пространство поиска, заставляя f быть записанной как линейная комбинация более простого стандартного блока

Базисные функции. Эта стратегия уже знакома из базового исчисления: разложение Тейлора записывает функции в виде многочленов, тогда как ряды Фурье используют синус и косинус.

11.1.1 Полиномиальная интерполяция

Возможно, самый простой интерполянт состоит в том, чтобы предположить, что f(x) находится в R[x], множестве многочленов. Многочлены гладкие, и найти полином степени k - 1 через k точек выборки несложно .

На самом деле в примере 3.3 уже проработаны детали такого метода интерполяции. Как напомним, предположим, что мы хотим найти f(x) a0 + a1x + a2x ; здесь наши неизвестными являются значения $2 + \cdots + a_{k-1}x^{k-1}$ a0, . . . , ак 1 . Подстановка выражения yi = f(xi) для каждого i показывает, что вектор a удовлетворяет системе Вандермонда k × k:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^{k-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{k-1} \end{bmatrix}$$

Таким образом, полиномиальную интерполяцию степени k можно выполнить с помощью линейного решения ak × k, применяя наши общие стратегии из предыдущих глав, но на самом деле мы можем добиться большего.

Один из способов думать о нашей форме для f(x) состоит в том, что она написана в базисе. Так же, как базой для Rn является набор из n линейно независимых векторов v1, . . . ,vn, здесь пространство полиномов степени x k 1}. Это может в диапазоне мономов {1, x, x R[x], но наш текущий выбор $2, \ldots,$ быть наиболее очевидный базис для k 1, записанный имеет несколько свойств, которые делают его полезным для задачи интерполяции.
Один из способов увидеть эту проблему — изобразить последовательность функций 1, $2, 3x, \ldots$ для x [0, 1]; в этом выглядеть одинаково. интервале, легко видеть, что при увеличении k функции x, которые начинают

Продолжая применять нашу интуицию из линейной алгебры, мы можем решить записать наш многочлен в базисе, который больше подходит для рассматриваемой проблемы. На этот раз напомним, что нам дано k пар (x1, y1), . . . ,(хк , ук). Мы будем использовать эти (неподвижные) точки для определения интерполяционного базиса Лагранжа φ1, . . . , φk , написав:

$$\varphi_i(x) \quad \frac{\prod_{j=i}(x - x_j)}{\prod_{j=i}(x_i - x_j)}$$

Хотя это и не записано в базисе 1, x, x номиналом $2, \ldots, x^{k-1}$, легко видеть, что каждый φi по-прежнему полигон. степени k 1. Кроме того, базис Лагранжа обладает следующим желательным свойством:

$$\phi_i(x) = \begin{cases} 1, & \text{когда } = i \\ 0 & \text{в противном случае.} \end{cases}$$

Таким образом, найти единственный полином степени k 1, соответствующий нашим парам (xi , yi) , легко в базисе Лагранжа:

$$f(x) \quad \sum_i y_i \varphi_i(x)$$

В частности, если мы подставим x = xj , то получим: f(xj)

$$= \sum_i y_i \varphi_i(x_j)$$

= yj в соответствии с нашим выражением для φi(x) выше.

Таким образом, в базисе Лагранжа можно записать замкнутую формулу для f(x) , не требующую решения системы Вандермонда. Недостаток, однако, заключается в том, что каждый φi(x) требует O(k) времени для оценки с использованием приведенной выше формулы для данного x, поэтому 2) время; если мы найдем коэффициенты нахождение f(x) берет O(n ai из системы Вандермонда в явном виде, однако оценка время можно сократить до O(n).

Помимо времени вычислений, у базиса Лагранжа есть дополнительный численный недостаток. Обратите внимание, что знаменатель является произведением ряда членов. Если xi близки друг к другу, то произведение может включать много членов, близких к нулю, поэтому мы делим на потенциально небольшое число.
Как мы видели, эта операция может создавать числовые проблемы, которых мы хотим избежать.

Одной из баз для полиномов степени k - 1, которая пытается найти компромисс между числовым качеством мономов и эффективностью базиса Лагранжа, является базис Ньютона, определяемый следующим образом:

$$\psi_i(x) = (x - x_j)$$

Определим ψ1(x) 1. Заметим, что ψi(x) — многочлен степени i 1. По определению ψi ясно, что ψi(x) = 0 для всех < i. Если мы захотим написать f(x) = i ciψi(x) и выписать это наблюдение более явно, мы найдем:

$$f(x_1) = c_1\psi_1(x_1)$$
$$f(x_2) = c_1\psi_1(x_2) + c_2\psi_2(x_2)$$
$$f(x_3) = c_1\psi_1(x_3) + c_2\psi_2(x_3) + c_3\psi_3(x_3)$$
$$\vdots \quad \vdots$$

Другими словами, мы можем решить следующую систему сил нижнего треугольника:

$$\begin{bmatrix} \psi_1(x_1) & 0 & \cdots & 0 \\ \psi_1(x_2) & \psi_2(x_2) & \cdots & 0 \\ \psi_1(x_3) & \psi_2(x_3) & \psi_3(x_3) & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \psi_1(x_k) & \psi_2(x_k) & \psi_3(x_k) & \cdots \psi_k(x_k) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}$$

Эту систему можно решить за O(n, 2) время с использованием прямой замены, а не O(n 3) время необходимое для решения системы Вандермонда.

Теперь у нас есть три стратегии интерполяции k точек данных с использованием полинома степени k - 1 путем записи его в мономиальном, лагранжевом и ньютоновском базисах. Все три представляют различные компромиссы между числовым качеством и скоростью. Однако важным свойством является то, что результирующая интерполированная функция f(x) одинакова в каждом случае. Точнее говоря, существует ровно один многочлен степени k - 1, проходящий через набор из k точек, поэтому, поскольку все наши интерполянты имеют степень k - 1, они должны иметь одинаковый выход.

11.1.2 Альтернативные базы

Хотя полиномиальные функции особенно хорошо поддаются математическому анализу, нет фундаментальной причины, по которой наш интерполяционный базис не может состоять из различных типов функций. Например, венчающий результат анализа Фурье подразумевает, что большой класс функций хорошо аппроксимируется суммами тригонометрических функций cos(kx) и sin(kx) при k N. Конструкция

как система Вандермонда все еще применима в этом случае, и на самом деле алгоритм быстрого преобразования Фурье (который заслуживает более широкого обсуждения) показывает, как выполнить такую интерполяцию еще быстрее.

Меньшее расширение разработки в §11.1.1 относится к рациональным функциям вида:

$$= 2 \qquad \frac{p0 + p1x + p2x\ f(x)^{2} + \cdots + пмх^{м}}{q0 + q1x + q2x + \cdots + qnx^{н}}$$

Обратите внимание, что если нам дано k пар (xi , yi), то нам потребуется m + n + 1 = k, чтобы эта функция была корректно определена. Необходимо установить одну дополнительную степень свободы, чтобы учесть тот факт, что одна и та же рациональная функция может быть выражена несколькими способами путем одинакового масштабирования числителя и знаменателя.

Рациональные функции могут иметь асимптоты и другие закономерности, которые невозможно получить, используя только полиномы, поэтому они могут быть желательными интерполянтами для функций, которые быстро изменяются или имеют полюса. На самом деле, когда m и n фиксированы, коэффициенты pi и qi по-прежнему можно найти с помощью линейных методов, умножив обе части на знаменатель:

$$yi(q0 + q1xi + q2x_{я}^{2} + \ldots + qnx_{я}^{н}) = p0 + p1xi + p2x_{я}^{2} + \ldots + пмх_{я}^{м}$$

Опять же, неизвестными в этом выражении являются p и q.

Однако гибкость рациональных функций может вызвать некоторые проблемы. Например, рассмотрим следующий пример: Пример 11.1

(Ошибка рациональной интерполяции, Булирш-Стёр, §2.2). Предположим, мы хотим найти f(x) со следующими точками данных: (0, 1), (1, 2), (2, 2). Мы могли бы выбрать m = n = 1. Тогда наши линейные условия станут такими:

$$q0 = p0$$

$$2(q0 + q1) = p0 + p1\ 2(q0 + 2q1) = p0 + 2p1$$

Одно нетривиальное решение этой системы:

$$p0 = 0$$

$$p1 = 2$$

$$q0 = 0$$

$$q1 = 1$$

Отсюда следует следующая форма для f(x):

$$f(x) = \frac{2x}{Икс}$$

Эта функция имеет вырождение при x = 0, и на самом деле сокращение x в числителе и знаменателе не дает f (0) = 1, как хотелось бы.

Этот пример иллюстрирует более широкое явление. Наша линейная система для нахождения p и q может столкнуться с проблемами, когда результирующий знаменатель px имеет корень при любом из фиксированных xi .
Можно показать, что в этом случае не существует рациональной функции с фиксированным выбором m и n, интерполирующих заданные значения. Типичное частичное разрешение в этом случае представлено в (CITE), которое попеременно увеличивает m и n до тех пор, пока не будет найдено нетривиальное решение. Однако с практической точки зрения специализированный характер этих методов является хорошим индикатором того, что альтернативные стратегии интерполяции могут быть предпочтительнее, когда основные рациональные методы не работают.

11.1.3 Кусочная интерполяция

До сих пор мы строили наши стратегии интерполяции, комбинируя простые функции на всех R.
Однако, когда количество точек данных k становится большим, становятся очевидными многие вырождения.
Например, на рисунке NUMBER показаны примеры, в которых подгонка полиномов высокой степени к входным данным может привести к неожиданным результатам. Кроме того, на рисунке NUMBER показано, что эти стратегии нелокальны, а это означает, что изменение любого отдельного значения yi во входных данных может изменить поведение f для всех x, даже тех, которые далеки от соответствующего xi . Почему-то это свойство нереалистично: мы ожидаем, что только входные данные вблизи заданного x повлияют на значение f(x), особенно когда имеется большое облако входных точек.

По этим причинам, когда мы проектируем набор базисных функций φ1, . . . , φk , желательно не только то, что с ними легко работать, но и то, что они имеют компактный носитель:

Определение 11.1 (Компактная опора). Функция g(x) имеет компактный носитель, если существует C R такое, что g(x) = 0 для любого x с |x| > C.

То есть функции с компактным носителем имеют только конечный диапазон точек, в которых они могут принимать ненулевые значения.

Обычная стратегия построения интерполяционных базисов с компактным носителем состоит в том, чтобы делать это кусочно. В частности, большая часть литературы по компьютерной графике основана на построении кусочных полиномов, которые определяются путем разбиения R на набор интервалов и записи разных полиномов в каждом интервале. Для этого мы упорядочим наши точки данных так, чтобы x1 < x2 < · · · < xk . Затем два простых примера кусочных интерполянтов следующие:

- Кусочная постоянная (НОМЕР РИСУНКА): для заданного x найдите точку данных xi , минимизирующую |x xи | и определить f(x) = yi .

- Кусочно-линейный (НОМЕР РИСУНКА): Если x < x1, возьмите f(x) = y1, а если x > xk , возьмите f(x) = yk . В противном случае найдите интервал с x [xi , xi+1] и определите

$$f(x) = y_{i+1} \cdot \frac{x - x_i}{x_{i+1} - x_i} + y_i \cdot \frac{x_{i+1} - x}{x_{i+1} - x_i}.$$

В более общем случае мы можем записать разные многочлены в каждом интервале [xi , xi+1]. Обратите внимание на нашу схему: кусочно-постоянные полиномы разрывны, а кусочно-линейные функции непрерывны. Легко видеть, что кусочно-квадратичные могут быть C и т.д. Эта повышенная непрерывность и дифференцируемость кубических кусочных функций, даже если C , каждый yi имеет локальную поддержку; эта теория детально разработана при построении «сплайнов» или кривых, интерполирующих между точками заданных значений функций и касательных.

Однако такая повышенная преемственность имеет свои недостатки. С каждой дополнительной степенью дифференцируемости мы делаем более сильное предположение о гладкости f . Это допущение может быть нереалистичным: многие физические явления действительно зашумлены или прерывисты, и эта повышенная гладкость может негативно повлиять на интерполяционные результаты. Одна область, в которой этот эффект особенно заметен, — это когда интерполяция используется в сочетании с инструментами физического моделирования. Моделирование турбулентных потоков жидкости с помощью сверхсглаженных функций может устранить прерывистые явления, такие как ударные волны, которые желательны в качестве выходных данных.

Помимо этих проблем, кусочные полиномы все еще могут быть записаны как линейные комбинации базисных функций. Например, следующие функции служат основой для кусочно-постоянных функций:

$$\phi i(x) = \begin{cases} 1, \text{ когда } xi \quad \frac{1+xi}{2} x < \quad \frac{xи+xи+1}{2} \\ 0 \text{ иначе} \end{cases}$$

Этот базис просто помещает константу 1 рядом с xi и 0 в другом месте; кусочно-постоянная интерполяция множества точек (xi , yi) записывается как f(x) = i yiφi(x). Точно так же так называемый базис «шляпы», показанный на рисунке НОМЕР, охватывает набор кусочно-линейных функций с острыми краями в наших точках данных.
xи :

$$\psi i(x) = \begin{cases} \frac{x \quad xi \quad 1}{xi \quad xi \quad 1} & \text{когда } xi \quad 1 < x \quad xi \\ \frac{xi+1 \quad x}{xi+1 \quad xi} & \text{когда } xi < x \quad xi+1 \text{ в} \\ 0 & \text{противном случае} \end{cases}$$

Опять же, по построению кусочно-линейная интерполяция заданных точек данных есть f (x) = i yiψi(x).

11.1.4 Гауссовские процессы и кригинг

Не включено в CS 205A, осень 2013 г.

11.2 Многопараметрическая интерполяция

Существует множество расширений описанных выше стратегий для интерполяции функции с заданными точками данных (xi , yi) , где xi Rn теперь может быть многомерным. Стратегии интерполяции в этом случае, однако, не столь ясны, поскольку менее очевидно разбить Rn на небольшое число областей вокруг xi . По этой причине общепринятой схемой является интерполяция с использованием функций относительно низкого порядка, то есть предпочтение упрощенных и эффективных стратегий интерполяции тем, которые выводят функции C .

Если все, что нам дано, это набор входных и выходных данных (xi , yi), то одна кусочно-постоянная стратегия интерполяции состоит в использовании интерполяции ближайшего соседа. В этом случае f(x) просто принимает значение yi , соответствующее xi, сводящее к минимуму x xi2; простые реализации перебирают все i, чтобы найти это значение, хотя структуры данных, такие как деревья kd, могут быстрее находить ближайших соседей. Точно так же, как кусочно-постоянные интерполяции делят R на интервалы вокруг точек данных xi , стратегия ближайшего соседа делит Rn на набор ячеек Вороного:

Определение 11.2 (ячейка Вороного). Дан набор точек S = {x1,x2, . . . ,xk} Rn , соответствующее , камера Вороного конкретному xi, есть множество Vi {x : x xi2 < x xj2 для всех j = i}. То есть это множество точек ближе к xi , чем к любому другому xj в S.

На рисунке НОМЕР показан пример ячеек Вороного о наборе точек данных в ячейках R2 , обладающих . Эти многими благоприятными свойствами; например, они являются выпуклыми многоугольниками и локализованы относительно каждого xi. На самом деле связность ячеек Вороного — это хорошо изученная проблема вычислительной геометрии, которая привела к построению знаменитой триангуляции Делоне.

Существует много вариантов непрерывной интерполяции функций на Rn , каждый со своими преимуществами и недостатками. Например, если мы хотим расширить нашу стратегию ближайшего соседа, описанную выше, мы могли бы вычислить несколько ближайших соседей x и интерполировать f (x) на основе x —

xi2 для каждого ближайшего соседа xi . Определенные структуры данных «k-ближайших соседей» могут ускорить запросы, когда вы хотите найти несколько точек в наборе данных, ближайших к заданному x.

Другой стратегией, часто встречающейся в литературе по компьютерной графике, является барицентрическая интерполяция. Предположим, у нас есть ровно n + 1 выборочная точка (x1, y1), . . . ,(xn+1, yn+1), где xi Rn , и, как всегда, мы хотим интерполировать значения y на все Rn ; например, на плоскости нам дали бы три значения, связанные с вершинами треугольника. Любую точку x Rn можно однозначно записать в виде линейной комбинации x = i=1 aixi с дополнительным ограничением, что i ai = 1; другими словами, мы записываем x как средневзвешенное значение точек xi . Барицентрическая интерполяция в этом случае просто пишет f(x) = i ai(x)yi .

На плоскости R2 барицентрическая интерполяция представляет собой прямую геометрическую интерполяцию в областях с вращающимся треугольником, как показано на рисунке НОМЕР. Кроме того, легко проверить, что полученная интерполированная функция f(x) аффинна, т. е. ее можно записать как f(x) = c + d · x для некоторых c R и d Rn .

В общем, система уравнений, которую мы хотим решить для барицентрической интерполяции при некотором x Rn :

$$\sum_i a_i x_i = x$$

$$\sum_i a_i = 1$$

При отсутствии вырождений эта система fora обратима при наличии n + 1 точек xi . Однако при наличии большего количества xi система для a становится недоопределенной. Это означает, что существует несколько способов записать данный x как средневзвешенное значение xi .

Одним из решений этой проблемы является добавление дополнительных условий на вектор усреднения весова. Результатом этой стратегии являются обобщенные барицентрические координаты, что является предметом исследований в области современной математики и инженерии. Типичные ограничения на a требуют, чтобы она была гладкой как функция на Rn и неотрицательной внутри множества xi , когда эти точки определяют многоугольник или многогранник. На рисунке NUMBER показан пример обобщенных барицентрических координат, вычисленных по точкам данных на многоугольнике с более чем n + 1 точкой.

Альтернативное решение недоопределенной проблемы для барицентрических координат связано с идеей использования кусочных функций для интерполяции; мы ограничим наше обсуждение здесь xi R2 для простоты, хотя расширения на более высокие измерения относительно очевидны.
Много раз нам давали не только множество точек xi , но и разложение интересующей нас области (в данном случае некоторое подмножество R2) на n + 1-мерные объекты, использующие эти точки в качестве вершин. Например, на рисунке НОМЕР показано такое разбиение части R2 на треугольники.
Интерполяция в этом случае проста: внутренняя часть каждого треугольника интерполируется с использованием барицентрических координат.

Пример 11.2 (Затенение). В компьютерной графике одним из наиболее распространенных представлений формы является набор треугольников в сетке. В повершинной модели затенения один цвет вычисляется для каждой вершины сетки. Затем, чтобы отобразить изображение на экране, эти значения для каждой вершины интерполируются с использованием барицентрической интерполяции к внутренней части треугольников. Подобные стратегии используются для текстурирования и других общих задач. На рисунке NUMBER показан пример этой простой модели затенения. Кроме того, одна проблема, специфичная для компьютерной графики, — это взаимодействие между перспективными преобразованиями и стратегиями интерполяции. Барицентрическая интерполяция цвета на трехмерной поверхности с последующим проецированием этого цвета на плоскость изображения отличается от проецирования треугольников на плоскость изображения с последующей интерполяцией цветов внутрь треугольника; таким образом, алгоритмы в этой области должны применять перспективную коррекцию, чтобы учесть эту ошибку.

Учитывая набор точек в R2 , проблема триангуляции далеко не тривиальна, и алгоритмы для выполнения такого рода вычислений часто плохо распространяются на Rn . Таким образом, в более высоких измерениях предпочтительными становятся стратегии ближайшего соседа или регрессии (см. НОМЕР Главы).

Барицентрическая интерполяция приводит к обобщению кусочно-линейных шляпных функций из §11.1.3, показанных на рисунке НОМЕР. Напомним, что наш результат интерполяции полностью определяется значениями yi в вершинах треугольников. На самом деле, мы можем думать о f(x) как о линейной комбинации i yiφi(x), где каждая φi(x) — это кусочно-барицентрическая функция, полученная путем помещения 1 на xi и 0 во всех остальных местах, как на рис. . Эти треугольные функции шляпы составляют основу «метода конечных элементов первого порядка», который мы рассмотрим в следующих главах; специальные конструкции, использующие полиномы более высокого порядка, известные как «элементы более высокого порядка», могут использоваться для обеспечения дифференцируемости вдоль ребер треугольника.

Альтернативное и не менее важное разложение области определения f происходит, когда точки xi встречаются на регулярной сетке в Rn . Следующие примеры иллюстрируют ситуации, когда это так:

Пример 11.3 (Обработка изображения). Как упоминалось во введении, типичная цифровая фотография представлена в виде сетки m × n интенсивностей красного, зеленого и синего цветов. Мы можем думать об этих значениях как о живущих на решетке в Z × Z. Предположим, однако, что мы хотим повернуть изображение на , угол, не кратный 90 . Затем, как показано на рисунке NUMBER, мы должны искать значения изображения в потенциально нецелочисленных позициях, требуя интерполяции цветов до R × R.

Пример 11.4 (Медицинская визуализация). Типичный вывод устройства магнитно-резонансной томографии (МРТ) представляет собой сетку значений am × n × p, представляющих плотность ткани в разных точках; теоретически типичная модель этой функции такова: f : R3 R. Мы можем извлечь внешнюю поверхность конкретного органа, показанного на рисунке НОМЕР, найдя набор уровней {x : f(x) = c} для некоторого c. Чтобы найти этот набор уровней, нам нужно распространить f на всю сетку вокселей, чтобы точно найти, где он пересекает c.

Стратегии интерполяции на основе сетки обычно применяют одномерные формулы из §11.1.3 по одному измерению за раз. Например, схемы билинейной интерполяции в R2 линейно интерполируют одно измерение за раз, чтобы получить выходное значение:

Пример 11.5 (Билинейная интерполяция). Предположим, f принимает следующие значения:

- f(0, 0) = 1
- f(0, 1) = 3
- f(1, 0) = 5
- f(1, 1) = 11

а промежуточный f получается билинейной интерполяцией. Чтобы найти $\phi(\frac{1}{4}, \frac{1}{2})$, мы сначала интерполируем по x1 , чтобы найти:

$$\phi\left(\frac{1}{4}, 0\right) = \frac{3}{4}f(0,0) + \frac{1}{4}f(1,0) = \frac{3}{4}\cdot 1 + \frac{1}{4}\cdot 5 = \frac{8}{4}$$

$$\phi\left(\frac{1}{4}, 1\right) = \frac{3}{4}f(0,1) + \frac{1}{4}f(1,1) = \frac{3}{4}\cdot 3 + \frac{1}{4}\cdot 11 = \frac{20}{4}$$

Далее интерполируем в x2:

$$\phi\left(\frac{1}{4}, \frac{1}{2}\right) = \frac{1}{2}\phi\left(\frac{1}{4}, 0\right) + \frac{1}{2}\phi\left(\frac{1}{4}, 1\right) = -2$$

Важным свойством билинейной интерполяции является то, что мы получаем один и тот же результат, интерполируя сначала по x2 , а затем по x1.

Методы более высокого порядка, такие как бикубическая интерполяция и интерполяция Ланцоша, снова используют больше полиномиальных членов, но их вычисления выполняются медленнее. В частности, в случае интерполяции изображений бикубические стратегии требуют больше точек данных, чем квадрат ближайших к точке x значений функции; эти дополнительные затраты могут замедлить работу графических инструментов, для которых каждый поиск в памяти требует дополнительного времени вычислений.

11.3 Теория интерполяции

До сих пор наше отношение к интерполяции было довольно эвристическим. Полагаясь на нашу интуицию в отношении того, что «разумная» интерполяция для набора значений функции по большей части является приемлемой стратегией, могут возникнуть тонкие проблемы с различными методами интерполяции, которые важно признать.

11.3.1 Линейная алгебра функций

Мы начали наше обсуждение с того, что представили различные стратегии интерполяции в качестве различных базисов для набора функций f : R R. Эта аналогия с векторными пространствами распространяется на полную геометрическую теорию функций и, по существу, на ранние работы в области функционального анализа. расширяет геометрию Rn до наборов функций. Здесь мы обсудим функции одной переменной, хотя многие аспекты расширения до более общих функций легко осуществить.

Точно так же, как мы можем определить понятия размаха и линейной комбинации для функций, для фиксированных a, b R мы можем определить скалярное произведение функций f(x) и g(x) следующим образом:

$$\langle f, g \rangle = \int_a^b f(x)g(x)\,dx.$$

Подобно тому, как A-внутреннее произведение векторов помогло нам вывести алгоритм сопряженных градиентов и имело много общего с скалярным произведением, функциональное скалярное произведение можно использовать для определения методов линейной алгебры для работы с пространствами функций и понимания их диапазона. Мы также определяем норму функции как $\|f\| = \sqrt{\langle f, f \rangle}$.

Пример 11.6. Внутренний продукт функции Возьмем $p_n(x) = x^n$ быть n-м мономом. Тогда при a = 0 и xb = 1, мы имеем:

$$\begin{aligned} \langle p_n, p_m \rangle &= \int_0^1 x^n \cdot x^m\,dx \\ &= \int_0^1 x^{n+m}\,dx \\ &= \frac{1}{n+m+1} \end{aligned}$$

Обратите внимание, что это показывает:

$$\frac{\text{пн}}{\text{пн}}, \frac{}{\text{вечера}} \quad \text{вечера} \quad \text{"="} \quad \frac{\text{пн, пм}}{\text{пнпм}}$$

$$\text{"="} \quad \frac{\overline{(2\text{н} + 1)(2\text{т} + 1)\ \text{п} +}}{\text{т} + 1}$$

Это значение приблизительно равно 1, когда n m, но n = m, что подтверждает наше предыдущее утверждение о том, что мономы значительно «перекрываются» на [0, 1].

Учитывая этот внутренний продукт, мы можем применить алгоритм Грама-Шмидта, чтобы найти ортонормированный базис для набора многочленов. Если мы возьмем a = 1 и b = 1, мы получим полиномы Лежандра, изображенные на рисунке НОМЕР:

$$P0(x) = 1$$

$$P1(x) = x$$

$$P2(x) = \frac{1}{2}(3x^2 \quad 1)$$

$$P3(x) = (5x^{\frac{1}{2}} \quad 1^3 \quad 3x)$$

$$P4(x) = \frac{}{8}(35x^4 \quad 30x^2 + 3)$$

$$\vdots \qquad \vdots$$

Эти полиномы обладают многими полезными свойствами благодаря своей ортогональности. Например, предположим, что мы хотим аппроксимировать f(x) суммой i aiPi(x). Если мы хотим минимизировать f i aiPi по функциональной норме, это задача наименьших квадратов! В силу ортогональности базиса Лежандра для R[x] простое расширение наших методов проецирования показывает:

$$\text{ай} = \frac{\text{ф , Пи}}{\text{Пи , Пи}}$$

Таким образом, аппроксимация f с помощью полиномов может быть достигнута простым интегрированием f по элементам базиса Лежандра; в следующей главе мы узнаем, как этот интеграл может быть выполнен приближенно.

Для положительной функции w(x) мы можем определить более общий скалярный продукт ·, ·w , написав

$$\text{f , гв} = \int_{\text{а}}^{\text{б}} w(x)f(x)g(x)\,dx.$$

Если мы возьмем $w(x) = \frac{1}{\sqrt{\quad}}$ с a = 1 и b = 1, то применение Грамма-Шмидта дает чебышевский

1 x 2 полиномы:

$$T0(x) = 1$$

$$T1(x) = x$$

$$T2(x) = 2x^2 \quad 1$$

$$T3(x) = 4x^3 \quad 3x$$

$$T4(x) = 8x^4 \quad 8x^2 + 1$$

$$\vdots \qquad \vdots$$

На самом деле для этих многочленов выполняется удивительное тождество:

$$T_k(x) = \cos(k \arccos(x)).$$

Эту формулу можно проверить, явно проверив ее для T0 и T1, а затем индуктивно применив наблюдение:

$$T_{k+1}(x) = \cos((k+1)\arccos(x)) = 2x \cos(k \arccos(x)) \cos((k-1)\arccos(x)) \text{ по тождеству } \cos((k+1)\theta) = 2\cos(k\theta)\cos(\theta) \cos((k-1)\theta) = 2xT_k(x) - T_{k-1}(x)$$

Эта формула «рекуррентности с тремя членами» также позволяет легко генерировать полиномы Чебышева.

Как показано на рисунке НОМЕР, благодаря тригонометрической формуле для полиномов Чебышева легко увидеть, что минимумы и максимумы Tk колеблются между +1 и -1.
Кроме того, эти экстремумы расположены в точках cos(iπ/k) (так называемые «точки Чебышева») для i от 0 до k; это хорошее распределение экстремумов позволяет избежать колебательных явлений, подобных показанному на рис. НОМЕР, при использовании конечного числа полиномиальных членов для аппроксимации функции. На самом деле, более технический подход к полиномиальной интерполяции рекомендует размещать xi для интерполяции рядом с точками Чебышева для получения гладкого вывода.

11.3.2 Аппроксимация кусочными полиномами

Предположим, мы хотим аппроксимировать функцию f(x) полиномом степени n на отрезке [a, b]. Определим Δx как расстояние $b - a$. Одна мера ошибки аппроксимации зависит от Δx, которая должна обращаться в нуль при $\Delta x \to 0$. Тогда, если мы аппроксимируем f кусочными полиномами, этот тип анализа говорит нам, как далеко друг от друга мы должны разнести полиномы, чтобы получить желаемый уровень приближения.

Например, предположим, что мы аппроксимируем f константой интерполяции $c = f(\frac{a+b}{2})$, как в кусочно-постоянной. Если предположить, что $|f(x)| < M$ для всех $x \in [a, b]$, мы имеем:

$$\text{Макс}_{x \in [a,b]} |e(x) - c| \le \text{макс}_{x \in [a,b]} M \text{ по теореме о среднем } \le M \Delta x$$

Таким образом, мы ожидаем ошибки $O(\Delta x)$ при использовании кусочно-постоянной интерполяции.

Предположим вместо этого, что мы аппроксимируем f, используя кусочно-линейную интерполяцию, то есть взяв

$$\tilde{f}(x) = f(a) + \frac{b - x}{b - a} f(a) + \frac{x - a}{b - a} f(b).$$

По теореме о среднем значении мы знаем, что $f(x) = f(\theta)$ для некоторого $\theta \in [a, b]$. Разложение Тейлора относительно θ показывает, что $f(x) = f(\theta) + f'(\theta)(x - \theta) + O(\Delta x^2)$ на $[a, b]$, а нашу линейку $\tilde{f}(x) = f(\theta) + f'(\theta)(x - \theta)$ можно переписать . Таким образом, вычитание этих двух выражений показывает, что). Нетрудно мере того, как ошибка аппроксимации f уменьшается до $O(\Delta x^2)$ предсказать эту ошибку аппроксимации , хотя на практике (Δx с полиномом степени n, делает $O(\Delta x^{n+1})$ кусочно- квадратичная сходимость линейных аппроксимаций достаточной для большинства приложений.

11.4 Проблемы

Идеи:

- Метод Хорнера для вычисления многочленов
- Рекурсивная стратегия для полиномиальных коэффициентов Ньютона.
- Сплайны, де Кастельжо
- Проверить интерполяцию площади треугольника барицентрической интерполяции

Глава 12

Численное интегрирование и Дифференциация

В предыдущей главе мы разработали инструменты для заполнения разумных значений функции f(x) с учетом выборки значений (xi , f(xi)) в области определения f . Очевидно, что эта интерполяционная проблема полезна сама по себе для завершения функций, о которых известно, что они непрерывны или дифференцируемы, но значения которых известны только в наборе изолированных точек, но в некоторых случаях мы затем хотим изучить свойства этих функций. В частности, если мы хотим применить инструменты исчисления к f , иметь возможность аппроксимировать его интегралы и мы должны производные.

На самом деле существует множество приложений, в которых ключевую роль в вычислениях играют численное интегрирование и дифференцирование. В самом простом случае некоторые известные функции определяются как интегралы. Например, «функция ошибки», используемая в качестве кумулятивного распределения кривой Гаусса или колоколообразной кривой, записывается:

$$\operatorname{erf}(x) \quad \frac{2}{\sqrt{\pi}} \int_0^{x_{\text{вес}}} e^{-t^2} \, dt$$

Аппроксимации erf (x) необходимы во многих статистических контекстах, и один из разумных подходов к нахождению этих значений состоит в том, чтобы выполнить приведенный выше интеграл численно.

В других случаях численные аппроксимации производных и интегралов являются частью более крупной системы. Например, методы, которые мы разработаем в будущих главах для аппроксимации решений дифференциальных уравнений, будут сильно зависеть от этих аппроксимаций. Точно так же в вычислительной электродинамике интегральные уравнения, решающие для неизвестной функции φ при заданном ядре K и выходе f, появляются в соотношении:

$$e(x) = \int_{\text{Рн}} K(x,y)\varphi(y)\, dy.$$

Эти типы уравнений должны быть решены для оценки электрических и магнитных полей, но если φ и K не являются очень особыми, мы не можем надеяться найти такой интеграл в закрытой форме, но решить это уравнение самостоятельно для неизвестной функции φ.

В этой главе мы разработаем различные методы численного интегрирования и дифференцирования на основе выборки значений функции. Эти алгоритмы обычно представляют собой довольно простые аппроксимации, поэтому для их сравнения мы также разработаем несколько стратегий, которые оценивают ожидаемую эффективность различных методов.

12.1 Мотивация

Нетрудно сформулировать простые приложения численного интегрирования и дифференцирования, учитывая, как часто инструменты исчисления появляются в основных формулах и методах физики, статистики и других областей. Здесь мы предлагаем несколько менее очевидных мест, где проявляются интеграция и дифференциация.

Пример 12.1 (выборка из дистрибутива). Предположим, что нам дано распределение вероятностей p(t) на интервале [0, 1]; то есть, если мы случайным образом выбираем значения в соответствии с этим распределением, мы ожидаем, что p(t) будет пропорциональна количеству раз, когда мы рисуем значение рядом с t. Распространенной задачей является генерация случайных чисел, распределенных как p(t).

Вместо того, чтобы разрабатывать специальный метод, чтобы делать это каждый раз, когда мы получаем новое значение p(t), можно сделать полезное наблюдение. Мы определяем кумулятивную функцию распределения p как

$$\Phi(\tau) = \int_0^\tau p(x)\,dx.$$

Тогда, если X — случайное число, равномерно распределенное в [0, 1], можно показать, что $F^{-1}(X)$ распределено как p, где F^{-1} — обратное число F. Таким образом, если мы можем аппроксимировать F или F^{-1} мы можем генерировать случайные числа в соответствии с произвольным распределением p; это приближение сводится к интегрированию p, которое, возможно, придется выполнять численно, когда интегралы не известны в точной форме.

Пример 12.2 (Оптимизация). Напомним, что большинство наших методов минимизации и нахождения корней функции f зависело от наличия не только значений $f(x)$, но и ее градиента $\nabla f(x)$ и даже гессиана Hf . Мы видели, что такие алгоритмы, как BFGS и метод Бройдена, строят грубые приближения производных f в процессе оптимизации. Однако, когда f имеет высокие частоты, может быть лучше аппроксимировать f вблизи текущей итерации x_k , а не использовать значения из потенциально далеких точек x для $< k$.

Пример 12.3 (рендеринг). Уравнение рендеринга из трассировки лучей и других алгоритмов для высококачественного рендеринга представляет собой интеграл, утверждающий, что свет, покидающий поверхность, равен интегралу света, попадающего на поверхность по всем возможным входящим направлениям после его отражения и рассеяния; по сути, в нем говорится, что световая энергия должна сохраняться до и после взаимодействия света с объектом. Алгоритмы рендеринга должны аппроксимировать этот интеграл, чтобы вычислить количество света, излучаемого поверхностью, отражающей свет в сцене.

Пример 12.4 (обработка изображения). Предположим, мы думаем об изображении как о функции двух переменных $I(x, y)$. Многие фильтры, в том числе размытие по Гауссу, можно рассматривать как свертки, заданные формулой

$$(I * g)(x, y) = \int I(u, v)g(x - u, y - v)\,du\,dv.$$

Например, чтобы размыть изображение, мы могли бы принять g за гауссиан; в этом случае $(I * g)(x, y)$ можно рассматривать как средневзвешенное значение цветов I вблизи точки (x, y). На практике изображения представляют собой дискретные сетки пикселей, поэтому этот интеграл необходимо аппроксимировать.

Пример 12.5 (правило Байеса). Предположим, что X и Y — непрерывнозначные случайные величины; мы можем использовать $P(X)$ и $P(Y)$ для выражения вероятности того, что X и Y принимают определенные значения. Иногда знание X может повлиять на наше знание Y. Например, если X — артериальное давление пациента, а Y — вес пациента,

тогда, зная, что у пациента большой вес, можно предположить, что у него также высокое кровяное давление. Таким образом, мы также можем записать условные распределения вероятностей P(X|Y) (читай «вероятность X при заданном Y»), выражающие такие отношения.

Основа современной теории вероятностей гласит, что P(X|Y) и P(Y|X) связаны следующим образом: P(Y|X)P(X)

$$P(X|Y) = \frac{}{P(Y|X)P(X)\,dY}$$

Оценка интеграла в знаменателе может быть серьезной проблемой в алгоритмах машинного обучения, где распределения вероятностей принимают сложные формы. Таким образом, приблизительные и часто рандомизированные схемы интеграции необходимы для алгоритмов выбора параметров, которые используют это значение как часть более крупного метода оптимизации.

12.2 Квадратура

Мы начнем с рассмотрения задачи численного интегрирования, или квадратуры. Эту проблему — с одной переменной — можно выразить так: «Дана выборка из n точек из некоторой функции f(x), f(x) dx». В предыдущем разделе мы найти приближение $\int_a^b$ представили несколько ситуаций, которые
уваривания именно к этой методике.

Есть несколько вариантов проблемы, которые требуют немного другого лечения или адаптации.
ция:

- Конечные точки a и b могут быть фиксированными, или мы можем захотеть найти квадратурную схему, которая может эффективно аппроксимировать интегралы для многих пар (a, b) .
- Мы можем запросить f(x) при любом x, но хотим аппроксимировать интеграл, используя относительно небольшое количество выборок, или нам может быть предоставлен список предварительно вычисленных пар (xi , f(xi)) и мы вынуждены использовать эти данные. точек в нашем приближении.

Эти соображения следует иметь в виду, когда мы разрабатываем различные алгоритмы для квадратурной задачи.

12.2.1 Интерполяционная квадратура

Многие стратегии интерполяции, разработанные в предыдущей главе, могут быть расширены до методов квадратур с использованием очень простого наблюдения. Предположим, мы запишем функцию f(x) в терминах набора базисных функций φi(x):

$$f(x) = \sum_i a_i \varphi_i(x).$$

Тогда мы можем найти интеграл от f следующим образом:

$$\int_a^b f(x)\,dx = \int_a^b \sum_i a_i \varphi_i(x)\,dx \quad \text{по определению } f$$

$$= \sum_i a_i \int_a^b \varphi_i(x)\,dx$$

$$= \sum_i c_i a_i \text{, если сделать определение } c_i \int_a^b \varphi_i(x)\,dx$$

Другими словами, интегрирование f просто включает в себя линейное комбинирование интегралов базисных функций, составляющих f.

Пример 12.6 (одночлены). Предположим, мы пишем $f(x) = \sum_k a_k x^k$. Мы знаем

$$\int_0^1 x^k \, dx = \frac{1}{x^{k+1}},$$

поэтому, применяя приведенный выше вывод, мы знаем

$$\int_0^1 f(x) \, dx = \sum_k \frac{a_k}{k+1}.$$

Другими словами, в наших обозначениях выше мы определили $c_k = \frac{1}{k+1}$.

Схемы, в которых мы интегрируем функцию путем интерполяции отсчетов и интегрирования интерполированной функции, известны как интерполяционные квадратурные правила; почти все методы, которые мы представим ниже, могут быть написаны таким образом. Конечно, мы можем столкнуться с проблемой курицы и яйца, если сам интеграл $\varphi_i(x)\,dx$ не известен в точной форме. Некоторые методы конечных элементов более высокого порядка решают эту проблему, затрачивая дополнительное вычислительное время на выполнение высококачественной численной аппроксимации интеграла одного φ_i, а затем, поскольку все φ имеют одинаковую форму, применяют формулы замены координат к напишите интегралы для остальных базисных функций. Этот канонический интеграл можно аппроксимировать в автономном режиме с помощью высокоточной схемы, а затем использовать пов

12.2.2 Правила квадратуры

Если нам дан набор пар $(x_i, f(x_i))$, наше обсуждение выше предлагает следующую форму квадратурного правила для аппроксимации интеграла от f на некотором интервале:

$$Q[f] \approx \sum_i w_i f(x_i).$$

Различные веса w_i дают разные аппроксимации интеграла, которые, как мы надеемся, становятся все более похожими по мере того, как мы собираем более плотную выборку x_i.

На самом деле, даже классическая теория интегрирования предполагает, что эта формула является разумной отправной точкой. Например, интеграл Римана, представленный во многих вводных классах исчисления, принимает форму:

$$\int_a^b f(x) = \lim_{\Delta x_k \to 0} \sum_k f(\tilde{x}_k)(x_{k+1} - x_k)$$

Здесь интервал [a, b] разбит на куски $a = x_1 < x_2 < \cdots < x_n = b$, где $\Delta x_k = x_{k+1} - x_k$, $\tilde{x}_k$ — любая точка из $[x_k, x_{k+1}]$. Для фиксированного набора x_k до предела этот интеграл явно может быть записан в форме $Q[f]$, приведенной выше.

С этой точки зрения выбор $\{x_i\}$ и $\{w_i\}$ полностью определяет стратегию квадратуры. Есть много способов определить эти значения, как мы увидим в следующем разделе и как мы уже видели для интерполяционной квадратуры.

Пример 12.7 (Метод неопределенных коэффициентов). Предположим, мы фиксируем $x_1, \ldots, x_n$ и хотят найти разумный набор сопутствующих весов w_i, чтобы $\sum_i w_i f(x_i)$ была подходящей аппроксимацией интеграла

выключенный . Альтернативой описанной выше стратегии базисной функции является использование метода неопределенных коэффициентов. В этой стратегии мы выбираем n функций f1(x), . . . , fn(x) , интегралы которых известны, и попросим, чтобы наше квадратурное правило точно восстановило интегралы этих функций:

$$\begin{aligned}
&\int_a^b f1(x)\,dx = w1\,f1(x1) + w2\,f1(x2) + \cdots + wn\,f1(xn)\\
&\int_a^b f2(x)\,dx = w1\,f2(x1) + w2\,f2(x2) + \cdots + wn\,f2(xn)\\
&\qquad\vdots \qquad\qquad\vdots\\
&\int_a^b fn(x)\,dx = w1\,fn(x1) + w2\,fn(x2) + \cdots + wn\,fn(xn)
\end{aligned}$$

Это создает линейную систему уравнений размера n × n для wi .

Один из распространенных вариантов - взять $fk(x) = x^{k-1}$, то есть убедиться, что квадратурная схема восстанавливается $(x) = x$ интегралы от полиномов низкого порядка. Мы знаем

$$\int_a^b x^k\,dx = \frac{b^{k+1} - a^{k+1}}{k+1}.$$

Таким образом, мы получаем следующую линейную систему уравнений для wi :

$$\begin{aligned}
&w1 + w2 + \cdots + wn = b - a\\
&x1w1 + x2w2 + \cdots + xnwn = \frac{b^2 - a^2}{2}\\
&x_1^2 w1 + x_2^2 w2 + \cdots + x_n^2 wn = \frac{b^3 - a^3}{3}\\
&\qquad\vdots \qquad\qquad\vdots\\
&x_1^{n-1} w1 + x_2^{n-1} w2 + \cdots + x_n^{n-1} wn = \frac{b^n - a^n}{n}
\end{aligned}$$

Эта система в точности является системой Вандермонда, обсуждавшейся в § 11.1.1.

12.2.3 Квадратура Ньютона-Котеса

Квадратура правит, когда правит x_is равномерно распределены в [a, b], известны как квадратуры Ньютона-Котеса Как показано на рис. НОМЕР, есть два разумных выбора равномерно расположенных образцов:

- Замкнутая квадратура Ньютона-Котеса помещает xi в точки a и b. В частности, при k {1, . . . , n} мы брать

$$xk = a + \frac{(k-1)(b-a)}{n-1}.$$

• Открытая квадратура Ньютона-Котеса не помещает xi в точку a или b:

$$x_k \quad a + \frac{k\,(b - a)}{n + 1}.$$

После этого выбора формулы Ньютона-Котеса просто применяют полиномиальную интерполяцию для аппроксимации интеграла от a до b; степень многочлена, очевидно, должна быть n - 1, чтобы квадратурное правило оставалось корректным.

В общем, мы будем держать n относительно небольшим. Таким образом, мы избегаем колебательных и шумовых явлений, возникающих при подгонке полиномов высокой степени к набору точек данных. Как и при кусочно-полиномиальной интерполяции, при интегрировании по большому интервалу [a, b] мы объединяем небольшие фрагменты в составные правила .

Закрытые правила. Замкнутые квадратурные стратегии Ньютона-Котеса требуют n 2, чтобы избежать деления на ноль. На практике часто встречаются две стратегии:

• Правило трапеций получается для n = 2 (так что x1 = a и x2 = b) путем линейной интерполяции от f(a) до f(b). В нем говорится, что

$$\int_a^b dx \quad (b \quad a)\,2 \qquad \frac{f(a) + f(b)}{} \; f(x).$$

• Правило Симпсона исходит из принятия n = 3, так что теперь мы имеем

$$x1 = a$$

$$x2 = \frac{a + b}{2}$$

$$x3 = b$$

Интегрирование параболы, проходящей через эти три точки, дает

$$\int_a^b f(x)\,dx \qquad \frac{b - a}{6} \quad f(a) + 4\,f \qquad \frac{a + b}{2} \quad + f(b).$$

Открытые правила. Открытые правила квадратур допускают возможность n = 1, что дает упрощенное правило средней точки:

$$\int_a^b f(x)\,dx \quad (b \quad a)f \qquad \frac{a + b}{2} \quad .$$

Большие значения n дают правила, подобные правилу Симпсона и правилу трапеций.

Композитная интеграция. Обычно нам может понадобиться интегрировать f(x) с более чем одним, двумя или тремя значениями xi . Очевидно, как построить составную линейку из средней точки или трапециевидной линейки, приведенной выше, как показано на рисунке НОМЕР; просто суммируйте значения по каждому интервалу. Например, если мы разделим [a, b] на k интервалов, то мы можем взять x и xi $\frac{b-k}{a}$ + i x. Тогда составное правило средней точки:

$$\int_a^b f(x)\,dx \quad f \qquad \frac{x_{i+1} + x_i}{2} \qquad x$$

Точно так же правило составных трапеций:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{k} \frac{f(x_i) + f(x_{i+1})}{2} \times$$

$$= \times \frac{1}{-}\,f(a) + f(x_1) + f(x_2) + \cdots + f(x_{k-1}) + f(b)\,\frac{1}{2}\,\frac{1}{-}$$

путем разделения двух усредненных значений f в первой строке и переиндексации

Альтернативный подход к составному правилу средней точки состоит в применении интерполяционной квадратурной формулы из § 12.2.1 к кусочно-линейной интерполяции; точно так же составная версия правила трапеций получается из кусочно-линейной интерполяции.

Составная версия правила Симпсона, показанная на рисунке НОМЕР, объединяет три точки одновременно, чтобы получить параболические приближения. Смежные параболы встречаются в точках xi с четными индексами и могут не иметь общих касательных. Это суммирование, которое существует только при четном n, принимает вид:

$$\int_a^b f(x)\,dx \approx \frac{x}{3}\,f(a) + 2\sum_{i=1}^{n-2-1} f(x_{2i}) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + f(b)$$

$$\overset{\text{"="}}{=} \frac{x}{3}\,[\,f(a) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(b)]$$

Точность. До сих пор мы разработали ряд квадратурных правил, которые эффективно комбинируют один и тот же набор f(xi) различными способами для получения различных приближений интеграла от f . Каждое приближение основано на другом инженерном предположении, поэтому неясно, лучше ли какое-либо из этих правил, чем какое-либо другое. Таким образом, нам необходимо разработать оценки ошибок, характеризующие их соответствующее поведение. Мы будем использовать наши интеграторы Newton-Cotes выше, чтобы показать, как можно проводить такие сравнения, как это представлено в CITE.

Во-первых, рассмотрим квадратурное правило средней точки на одном интервале [a, b]. Определите $c = \frac{1}{2}(a + b)$.
Ряд Тейлора f o c равен:

$$f(x) = f(c) + f'(c)(x - c) + \frac{f''(c)}{2}(x - c)^2 + \frac{f'''(c)}{6}(x - c)^3 + \frac{f^{(4)}(c)}{24}(x - c)^4 + \cdots$$

Таким образом, в силу симметрии относительно c нечетные члены выпадают:

$$\int_a^b f(x)\,dx = (b - a)f(c) + \frac{1}{24}f''(c)(b - a)^3 + \frac{1}{1920}f^{(4)}(c)(b - a)^5 + \cdots$$

Обратите внимание, что первый член этой суммы в точности соответствует $\int_a^b f(x)\,dx$, обеспечиваемый средней точкой оценке правила, так что это правило точно до $O(x^3)$.

Теперь, подставив a и b в наш ряд Тейлора для f o c, мы получим:

$$f(a) = f(c) + f'(c)(a - c) + \frac{1}{2}f''(c)(a - c)^2 + \frac{1}{6}f'''(c)(a - c)^3 + \cdots$$

$$f(b) = f(c) + f'(c)(b - c) + \frac{f''(c)}{2}(b - c)^2 + \frac{f'''(c)}{6}(b - c)^3 + \cdots$$

Если сложить их вместе и умножить обе части на b − a/2, получится:

$$(b - a)\left(\frac{f(a) + f(b)}{4}\right) = f(c)(b - a) + f''(c)\left(\frac{(a - c)^2}{2} + \frac{(b - c)^2}{2}\right) + \cdots$$

Член f (c) обращается в нуль по определению c. Обратите внимание, что левая часть представляет собой интегральную оценку по правилу трапеций, а правая часть согласуется с нашим рядом Тейлора для $\int_a^b f(x)\,dx$ с точностью до кубического члена. Другими словами, правило трапеций также $O(\Delta x^3)$ с точностью до одного интервала.

Здесь мы остановимся, чтобы отметить поначалу неожиданный результат: правила трапеций и средней точки имеют один и тот же порядок точности! На самом деле, изучение члена третьего порядка показывает, что правило средней точки примерно в два раза точнее, чем правило трапеций. Этот результат кажется противоречащим здравому смыслу, поскольку правило трапеций использует линейную аппроксимацию, а правило средней точки является постоянным.
Однако, как показано на рисунке НОМЕР, правило средней точки фактически восстанавливает интеграл линейных функций, что объясняет его дополнительную степень точности.

Аналогичный аргумент применим к нахождению оценки ошибки для правила Симпсона. [НАПИСАТЬ ОБЪЯСНЕНИЕ). НАРОД ЗДЕСЬ; ИСКЛЮЧИТЬ ИЗ 205A]. В конце концов мы обнаруживаем, что правило Симпсона имеет ошибку типа $O(\Delta x^5)$

К такого рода анализу относится важное предостережение. В общем, теорема Тейлора применима только тогда, когда Δx достаточно мало. Если выборки находятся далеко друг от друга, тогда проявляются недостатки полиномиальной интерполяции, а колебательные явления, как обсуждалось в разделе НОМЕР, могут привести к нестабильным результатам для схем интегрирования высокого порядка.

Таким образом, возвращаясь к случаю, когда a и b далеко друг от друга, мы теперь разделим [a, b] на интервалы ширины Δx и применим любое из наших квадратурных правил внутри этих интервалов. Обратите внимание, что наше общее количество интервалов равно $(b-a)/\Delta x$, поэтому в этом случае мы должны умножить наши оценки ошибок на $1/\Delta x$. В частности, выполняются следующие порядки точности:

- Составная средняя точка: $O(\Delta x^2)$
- Составная трапеция: $O(\Delta x^2)$
- Композитный Симпсон: $O(\Delta x^4)$

12.2.4 Квадратура Гаусса

В некоторых приложениях мы можем выбирать местоположения x_i , в которых f выбирается. В этом случае мы можем оптимизировать не только веса квадратурного правила, но и местоположения x_i , чтобы получить максимальное качество. Это наблюдение приводит к сложным, но теоретически привлекательным квадратурным правилам.

Детали этой техники выходят за рамки нашего обсуждения, но мы предлагаем один простой способ ее получения. В частности, как и в примере 12.7, предположим, что мы хотим оптимизировать $x_1, \ldots, x_n$ и $w_1, \ldots, w_n$ одновременно для повышения порядка схемы интегрирования. Теперь у нас есть 2n вместо n известных, поэтому мы можем обеспечить равенство для 2n примеров:

$$\begin{aligned}
\int_a^b f_1(x)\,dx &= w_1 f_1(x_1) + w_2 f_1(x_2) + \cdots + w_n f_1(x_n) \\
\int_a^b f_2(x)\,dx &= w_1 f_2(x_1) + w_2 f_2(x_2) + \cdots + w_n f_2(x_n) \\
&\vdots \qquad\qquad \vdots \\
\int_a^b f_{2n}(x)\,dx &= w_1 f_n(x_1) + w_2 f_n(x_2) + \cdots + w_n f_n(x_n)
\end{aligned}$$

Теперь и x_i , и w_i неизвестны, поэтому эта система уравнений больше не является линейной. Например, если мы хотим оптимизировать эти значения для многочленов на интервале $[-1, 1]$, мы должны

необходимо решить следующую систему полиномов (CITE):

$$w1 + w2 = \int_{-1}^{1} 1\,dx = 2$$

$$w1x1 + w2x2 = \int_{-1}^{1} x\,dx = 0$$

$$2\,w1x_1^{\,2} + w2x_2^{\,2} = \int_{-1}^{1} x^2\,dx = \frac{2}{3}$$

$$3\,w1x_1^{\,3} + w2x_2^{\,3} = \int_{-1}^{1} x^3\,dx = 0$$

Может случиться так, что системы, подобные этой, имеют несколько корней и другие вырождения, которые зависят не только от выбора fi (обычно многочленов), но и от интервала, на котором мы аппроксимируем интеграл. Кроме того, эти правила не являются прогрессивными в том смысле, что набор xi для n точек данных не имеет ничего общего с набором для k точек данных, когда k = n, поэтому повторно использовать данные для получения более точной оценки сложно. С другой стороны, когда они применимы, квадратура Гаусса имеет наивысшую возможную степень при фиксированном n. Квадратурные правила Кронрода пытаются избежать этой проблемы, оптимизируя квадратуру с 2n + 1 точками при повторном использовании точек Гаусса.

12.2.5 Адаптивная квадратура

Как мы уже показали, существуют некоторые функции f, интегралы которых лучше аппроксимируются данным квадратурным правилом, чем другие; например, правила средней точки и правила трапеций интегрируют линейные функции с полной точностью, в то время как при быстрых колебаниях f могут возникнуть проблемы с дискретизацией и другие проблемы.

Напомним, что квадратурное правило Гаусса предполагает, что размещение xi может влиять на качество квадратурной схемы. Однако осталась одна часть информации, которую мы не использовали: значения f(xi). Ведь именно они определяют качество нашей квадратурной схемы.

Имея это в виду, адаптивные квадратурные стратегии исследуют текущую оценку и генерируют новые xi , где подынтегральное выражение более сложное. Стратегии адаптивной интеграции часто сравнивают результаты нескольких квадратурных методов, например трапеции и средней точки, с предположением, что они согласуются там, где достаточно выборки f (см. НОМЕР РИСУНКА). Если они не согласуются с каким-либо допуском на заданном интервале, генерируется дополнительная точка выборки и обновляются интегральные оценки.

ДОБАВИТЬ ПОДРОБНЕЕ ИЛИ ПРИМЕР; ОБСУЖДЕНИЕ РЕКУРСИВНОГО АЛГОРИТМА; ГАНДЕР И ГАУЧИ

12.2.6 Множественные

переменные Много раз мы хотели бы интегрировать функции f(x) , где x Rn . Например, когда n = 2, мы можем интегрировать по прямоугольнику, вычисляя

$$\int_a^b \int_c^d f(x, y)\,dx\,dy.$$

В более общем смысле, как показано на рис. ЧИСЛО ¡, мы можем захотеть найти интеграл, где $\int_{\Omega} f(x)\,dx$, Ω — некоторое подмножество Rn .

«Проклятие размерности» экспоненциально усложняет интеграцию по мере увеличения размерности. В частности, количество выборок f, необходимых для достижения сравнимой квадратурной точности для интеграла в Rk, увеличивается как O(n). Это наблюдение может быть обескураживающим, но в некоторой степени разумным: чем больше входных измерений для f, тем больше выборок требуется, чтобы понять его поведение во всех измерениях.

Простейшей стратегией интегрирования в Rk является интегрированный интеграл. Например, если f — функция Для двух переменных предположим, что мы хотим найти f(x,y) dx dy. Для фиксированного y мы можем аппроксимировать внутренний интеграл по x, используя одномерное квадратурное правило; затем мы интегрируем эти значения по y, используя другое квадратурное правило. Очевидно, что обе схемы интегрирования вызывают некоторую ошибку, поэтому нам может потребоваться выборка xi более плотно, чем в одном измерении, для достижения желаемого качества вывода.

В качестве альтернативы, точно так же, как мы разделили [a, b] на интервалы, мы можем подразделить Ω на треугольники и прямоугольники в 2D, многогранники или прямоугольники в 3D и т. д. и использовать простые интерполяционные квадратурные правила в каждой части. Например, одним из популярных вариантов является интегрирование результатов барицентрической интерполяции внутри многогранников, поскольку этот интеграл известен в закрытой форме.

Однако, когда n велико, нецелесообразно делить домен, как предлагается. В этом случае мы можем использовать рандомизированный метод Монте-Карло. В этом случае мы просто генерируем k случайных точек xi Ω, например, с равномерной вероятностью. Усреднение значений f(xi) дает аппроксимацию f(x) dx, которая сходится как 1/ k – Ом независимо от размерности Ω! Так, в больших размерах оценка Монте-Карло предпочтительнее описанных выше детерминированных квадратурных методов.

ПОДРОБНЕЕ О КОНВЕРГЕНЦИИ МОНТЕ-КАРЛО И ВЫБОРЕ РАСПРЕДЕЛЕНИЙ БОЛЕЕ Ом

12.2.7 Кондиционирование

До сих пор мы рассматривали качество квадратурного метода с использованием значений точности O(x, к); очевидно, что по этой метрике предпочтительнее набор квадратурных весов с большими k.

Однако другая мера уравновешивает измерения точности, полученные с использованием аргументов Тейлора. В частности, напомним, что мы записали наше квадратурное правило как Q[f] i wi f(xi). Предположим, мы возмущаем f некоторым другим f . Определить f f maxx [a,b] | f(x) f(x)|. Затем,

$$\frac{|Q[f] - Q[\hat{f}]|}{\text{ж - ж}} \overset{\text{"="}}{=} \frac{|\sum_i w_i(f(x_i) - \hat{f}(x_i))| \text{ ж}}{\text{- ж}}$$

$$\leq \frac{\sum_i |w_i||f(x_i) - \hat{f}(x_i)| \text{ по}}{\text{f}}$$

неравенству треугольника f , так как | f(xi) f(xi)| f f по определению.

Таким образом, устойчивость или обусловленность квадратурного правила зависит от нормы набора весов ш .

В общем, легко проверить, что по мере увеличения порядка квадратурной точности условие w ухудшается, поскольку wi принимает большие отрицательные значения; это контрастирует со случаем всех положительных значений, где обусловленность ограничена b - a, потому что i wi = b - a для полиномиальных интерполяционных схем, а большинство методов низкого порядка имеют только положительные коэффициенты (CHECK). Этот факт является отражением той же интуиции, что мы не должны интерполировать функции, используя многочлены высокого порядка. Таким образом, на практике мы обычно предпочитаем составные квадратуры методам высокого порядка, которые могут давать более точные оценки, но могут быть неустойчивыми при численном возмущении.

12.3 Дифференциация

Численное интегрирование является относительно стабильной задачей. в том, что влияние любого отдельного значения $f(x_a^b)$ на f(x) dx уменьшается до нуля, когда a и b становятся далеко друг от друга. С другой стороны, аппроксимация производной функции f (x) таким свойством устойчивости не обладает. С точки зрения анализа Фурье можно показать, что интеграл f(x) обычно имеет более низкие частоты, чем f , в то время как дифференцирование для получения f усиливает высокие частоты f , что делает ограничения выборки, условия и стабильность особенно сложными для аппроксимации f .

Несмотря на сложные обстоятельства, аппроксимации производных обычно относительно легко вычислить, и они могут быть устойчивыми в зависимости от рассматриваемой функции. На самом деле, при разработке правила секущих, метода Бройдена и т. д. мы использовали простые аппроксимации производных и градиентов, чтобы помочь в процедурах оптимизации.

Здесь мы сосредоточимся на аппроксимации f для f : R R. Нахождение градиентов и якобианов часто достигается путем дифференцирования в одном измерении за раз, что эффективно сводится к одномерной задаче, которую мы здесь рассматриваем.

12.3.1 Дифференциация базовых функций

Простейший случай дифференцирования подходит для функций, построенных с использованием подпрограмм интерполяции. Как и в § 12.2.1, если мы можем записать f(x) = i aiφi(x) , то по линейности мы знаем

$$f(x) = (x). \text{ айф я}$$

Другими словами, мы можем думать о функциях φ i как о базисе для производных функций, записанных в базисе φi !

Пример этой процедуры показан на рисунке НОМЕР. Это явление часто связывает разные интерполяционные схемы. Например, кусочно-линейные функции имеют кусочно-постоянные производные, полиномиальные функции имеют полиномиальные производные меньшей степени и т. д.; мы вернемся к этой структуре, когда будем рассматривать дискретизацию уравнений в частных производных. Между тем, в этом случае полезно знать, что f известно с полной уверенностью, хотя, как и на рисунке ЧИСЛО, его производные могут иметь нежелательные разрывы.

12.3.2 Конечные разности

Более распространен случай, когда у нас есть функция f(x) , которую мы можем запросить, но производные которой неизвестны. Это часто происходит, когда f принимает сложную форму или когда пользователь предоставляет f(x) как подпрограмму без аналитической информации о ее структуре.

Определение производной предлагает разумный подход:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \quad f(x)$$

Как и следовало ожидать, для конечного h > 0 с малым |h| выражение в пределе обеспечивает возможное значение, приближающееся к f (x).

Чтобы подтвердить эту интуицию, мы можем использовать ряд Тейлора, чтобы написать:

$$f(x+h) = f(x) + f(x)h + f(x)h^2 \frac{1}{2} + \cdots$$

Преобразование этого выражения показывает:

$$(x) = \frac{f(x+h) - f(x)}{f} + O(ч)\ ч$$

Таким образом, следующая прямая разностная аппроксимация f имеет линейную сходимость:

$$h \quad \frac{f(x+h) \quad f(x)}{f(x)}$$

Точно так же изменение знака h показывает, что обратные различия также имеют линейную сходимость:

$$f(x) \quad \frac{f(x) \quad f(x \quad h)}{h}$$

На самом деле мы можем улучшить сходимость нашего приближения, используя хитрость. Тейлор теорему мы можем написать:

$$f(x+h) = f(x) + f(x)h + \frac{1}{2}\phi(x)ч^{2} + \frac{1}{6}\frac{1}{}\phi(x)ч \quad 3 + \cdots$$

$$f(x-h) = f(x) - f(x)h + \frac{1}{2}\phi(x)ч^{2} \quad \frac{1}{6}\phi(x)ч \quad 3 + \cdots$$

$$1 = \quad f(x+h) \quad f(x \quad h) = 2f(x)h + f(x)h\ 3 \quad 3 + \cdots$$

$$= \quad \frac{f(x+h) \quad f(x \quad h)}{} = f(x) + O(h\ 2h^{2})$$

Таким образом, эта центрированная разность дает приближение f (x) с квадратичной сходимостью; это высший порядок сходимости, которого мы можем ожидать с разделенной разностью. Однако мы можем добиться большей точности, оценив f в других точках, например, x + 2h, хотя на практике это приближение редко используется в пользу простого уменьшения h.

Построение оценок производных высших порядков может происходить по аналогичным построениям. Для Например, если мы сложим разложения Тейлора f(x + h) и f(x h), мы увидим

$$f(x+h) + f(x-h) = 2f(x) + f(x)h\ f(x^{2} + O(ч^{3})$$

$$= \quad \frac{+ h) - 2f(x) + f(x-h) = f(x)}{} + O(ч\ ч\ 2 \quad ^{2})$$

Чтобы предсказать подобные комбинации для высших производных, нужно заметить, что наша вторая формулу производной можно разложить по-разному:

$$\frac{f(x+h) - 2f(x) + f(x-h)}{h\ 2} \quad "=" \quad \frac{\frac{f(x+h) \quad f(x)}{h} \quad \frac{f(x) \quad f(x \quad h)}{h}}{час}$$

То есть наша аппроксимация второй производной представляет собой «конечную разность конечных разностей». Один из способов интерпретации этой формулы показан на рисунке НОМЕР. Когда мы вычисляем аппроксимацию прямой разности f между x и x + h, мы можем думать об этом наклоне как о живущем при x + h/2; мы аналогичным образом можем использовать обратные разности, чтобы поместить наклон в x - h/2. Нахождение наклона между этими значениями возвращает приближение к x.

Одной из стратегий, которая может улучшить сходимость приведенных выше приближений, является экстраполяция Ричардсона. В качестве примера более общего шаблона предположим, что мы хотим использовать прямые разности для аппроксимации f. Определять

$$D(h) \quad \frac{e(x+h) - e(x)}{h}.$$

Очевидно , что D(h) приближается к f (x) при h 0. Однако, более конкретно, из нашего обсуждения в § 12.3.2 мы знаем, что D(h) принимает форму:

$$D(h) = f(x) + \frac{1}{2}f(x)h + O(h^{2})$$

Предположим, что мы знаем D(h) и D(αh) для некоторого 0 < α < 1. Мы знаем:

$$D(\alpha h) = f(x) + \frac{1}{2}f(x)\alpha h + O(h^{2})$$

Мы можем записать эти два отношения в матрицу:

$$\begin{matrix} 1 & \frac{1}{2}час \\ 1 & \frac{1}{2}ax \end{matrix} \begin{matrix} ж(x) \\ ж(x) \end{matrix} \quad "=" \quad \begin{matrix} Д(ч) \\ D(ax) \end{matrix} + O(ч^{2})$$

Или, что то же самое,

$$\begin{matrix} ж(x) \\ ж(x) \end{matrix} \quad "=" \quad \begin{matrix} 1 \\ \frac{1}{1}час \\ 1\ 1\ \frac{2}{2}1ax \end{matrix} \begin{matrix} Д(ч) \\ D(ax) \end{matrix} + O(ч^{2})$$

То есть мы взяли аппроксимацию f (x) за O(h) с помощью D(h) и превратили ее в ^{2}) ок аппроксимацию за O(h) ! аппроксимация D(h).Тот же прием можно применить и ко многим другим задачам, записав аппроксимацию D(h) = a + bhn + O(h m), где m > n, где a — величина, которую мы надеемся оценить, и b - следующий член в разложении Тейлора.На самом деле, даже экстраполяцию Ричардсона можно применять рекурсивно для получения приближений все более и более высокого порядка.

12.3.3 Выбор размера шага

В отличие от квадратуры, численное дифференцирование обладает любопытным свойством. Оказывается, что любой выбранный нами метод может быть сколь угодно точным, просто выбрав достаточно малое значение h. Это наблюдение привлекательно с точки зрения того, что мы можем добиться более качественных приближений без дополнительного времени вычислений. Загвоздка, однако, в том, что мы должны делить на h и сравнивать все больше и больше похожих значений f(x) и f(x + h); в арифметике с конечной точностью сложение и/или деление на почти нулевые значения вызывает числовые проблемы и нестабильность. Таким образом, существует диапазон значений h, которые недостаточно велики, чтобы вызвать значительную ошибку дискретизации, и недостаточно малы, чтобы создавать численные проблемы; На рисунке NUMBER показан пример дифференциации простой функции в арифметике с плавающей запятой IEEE.

12.3.4 Интегрированные количества

Не включено в CS 205A, осень 2013 г.

12.4 Проблемы

- Квадратура Гаусса – всегда содержит средние точки, стратегия с использованием ортогональных полиномов
- Адаптивная квадратура
- Применение экстраполяции Ричардсона в других местах

Глава 13

Обыкновенные дифференциальные уравнения

Мы мотивировали проблему интерполяции в главе 11, перейдя от анализа к поиску функций. То есть в таких задачах, как интерполяция и регрессия, неизвестное — это функция f, а задача алгоритма — заполнить недостающие данные.

Мы продолжим это обсуждение, рассмотрев аналогичные задачи, связанные с заполнением значений функций. Здесь наше неизвестное по-прежнему является функцией f , но вместо того, чтобы просто угадывать пропущенные значения, мы хотели бы решить более сложные задачи проектирования. Например, рассмотрим следующие проблемы:

- Найти f, аппроксимирующую некоторую другую функцию f0 , но удовлетворяющую дополнительным критериям (гладкость, непрерывность, низкая частота и др.).
- Смоделируйте некоторую динамическую или физическую взаимосвязь как f(t), где t — время.
- Найдите f со значениями, аналогичными f0 , но с некоторыми общими свойствами с другой функцией. г0.

В каждом из этих случаев наша неизвестная является функцией f, но наши критерии успеха более сложны, чем «соответствие заданному набору точек данных».

Теории обыкновенных дифференциальных уравнений (ОДУ) и уравнений в частных производных (УЧП) изучают случай, когда мы хотим найти функцию f (x) на основе информации о ее производных или взаимосвязях между ними. Обратите внимание, что мы уже решили простую версию этой проблемы при обсуждении квадратуры: при заданном f (t) методы квадратуры обеспечивают способы аппроксимации f (t) с помощью интегрирования.

В этой главе мы рассмотрим случай обыкновенных дифференциальных уравнений и, в частности, начальные задачи. Здесь неизвестной является функция f(t) : R Rn , и мы дали уравнение, которому удовлетворяют f, и ее производные, а также f(0); наша цель — предсказать f(t) для t > 0. Мы приведем несколько примеров ОДУ, встречающихся в литературе по информатике, а затем перейдем к описанию общих методов решения.

В качестве примечания мы будем использовать обозначение f для обозначения производной d f/dt от f : [0,) Rn .Наш цель будет состоять в том, чтобы найти f(t) с заданными отношениями между t, f(t), f (t), f (t) и так далее.

13.1 Мотивация

ОДУ появляются практически в любой части научного примера, и нетрудно встретить практические ситуации, требующие их решения. Например, основные законы физического движения задаются ОДУ:

Пример 13.1 (второй закон Ньютона). Продолжая §5.1.2, вспомним, что Второй закон Ньютона гласит, что F = ma, то есть общая сила, действующая на объект, равна произведению его массы на его ускорение.
Если мы моделируем n частиц одновременно, то можно представить себе объединение всех их положений в вектор x R3n . Точно так же мы можем написать функцию F(t,x,x) R3n , которая принимает время, положения частиц и их скорости и возвращает общую силу, действующую на каждую частицу. Эта функция может учитывать взаимосвязь между частицами (например, гравитационные силы или пружины), внешние эффекты, такие как сопротивление ветра (которое зависит от x), внешние силы, изменяющиеся во времени t, и так далее.

Затем, чтобы найти положение всех частиц в зависимости от времени, мы хотим решить уравнение x = F(t,x,x)/m. Обычно нам даются положения и скорости всех частиц в момент времени t = 0 в качестве начального условия.

Пример 13.2 (Сворачивание белков). В меньшем масштабе уравнения, управляющие движением молекул, также являются обыкновенными дифференциальными уравнениями. Одним из особенно сложных случаев является укладка белка, в которой геометрическая структура белка предсказывается путем моделирования межмолекулярных сил с течением времени. Эти силы часто принимают нелинейные формы, которые продолжают бросать вызов исследователям в области вычислительной биологии.

Пример 13.3 (Градиентный спуск). Предположим, мы хотим минимизировать функцию энергии E(x) по всем x. В главе 8 мы узнали, что E(x) указывает в направлении, в котором E больше всего уменьшается при заданном x, поэтому мы провели линейный поиск вдоль этого направления от x, чтобы минимизировать E локально. Альтернативный вариант, популярный в некоторых теориях, состоит в том, чтобы решить ОДУ формы x = - E(x); другими словами, думайте о x как о функции времени x(t) , которая пытается уменьшить E, спускаясь с горы.

Например, предположим, что мы хотим решить Ax =b для симметричного положительно определенного A. Мы знаем из что это эквивалентно минимизации E(x) x = f(x) $\frac{1}{2}$§10.1.1 x Ax bx + c. Таким образом, мы могли бы попытаться решить ОДУ =b Ax. При t мы ожидаем, что x(t) будет все лучше и лучше удовлетворять линейной системе.

Пример 13.4 (моделирование толпы). Предположим, мы пишем программное обеспечение для видеоигр, требующее реалистичного моделирования виртуальных скоплений людей, животных, космических кораблей и т.п. Одна из стратегий создания правдоподобного движения, показанная на рисунке НОМЕР, заключается в использовании дифференциальных уравнений. Здесь скорость члена толпы определяется как функция его окружения; например, в толпе людей близость других людей, расстояние до препятствий и т. д. могут влиять на направление движения данного агента. Эти правила могут быть простыми, но в совокупности их взаимодействие сложно. В основе этого механизма лежат стабильные интеграторы для дифференциальных уравнений, поскольку мы не хотим иметь заметно нереалистичное или нефизическое поведение.

13.2 Теория ОДУ

Полное рассмотрение теории обыкновенных дифференциальных уравнений выходит за рамки нашего обсуждения, и мы отсылаем читателя к CITE за более подробной информацией. Помимо этого, мы упомянем здесь некоторые основные моменты, которые будут иметь отношение к нашей разработке в следующих разделах.

13.2.1 Основные понятия

Наиболее общая проблема начального значения ОДУ принимает следующую форму:

$$\text{Найти } f(t) : R \quad R, \quad {}^{н}$$

$$\text{удовлетворяющее } F[t, f(t), f(t), f(t), \ldots, f(k)(t)] = 0$$

$$\text{Даны } f(0), f(0), f(0), \ldots, f(k \quad 1)(0)$$

Здесь F — некоторая связь между f и всеми ее производными; мы используем f () для обозначения -th производной от f . Мы можем думать об ОДУ как об определяющих эволюцию f во времени t; мы знаем f и ее производные в нулевое время и хотим предсказать их в будущем.

ОДУ принимают разные формы даже в одной переменной. Например, обозначим y = f(t) и предположим, что y R1 . Затем примеры ОДУ включают:

- y = 1 + cos t: Это ОДУ можно решить путем интегрирования обеих частей, например, с помощью квадратурного уравнения. методы
- y = ay: это ОДУ линейно по y
- y = ay + e ^{т}: ОДУ зависит от времени и положения.
- y + 3y y = t: это ОДУ включает несколько производных от y
- y грех y = e ^{ты} : Это ОДУ нелинейно по y и t.

Очевидно, что решить самые общие ОДУ может быть непросто. Мы ограничим большую часть нашего обсуждения случаем явных ОДУ, в которых можно выделить производную высшего порядка:

Определение 13.1 (явное ОДУ). ОДУ является явным, если его можно записать в виде

$$f(k)(t) = F[t, f(t), f(t), f(t), \ldots, f(k \quad 1)(t)].$$

Например, явная форма второго закона Ньютона: $x(t) = \frac{1}{м} a(т,x(т),x(т))$.

Удивительно, но, обобщая прием из §5.1.2, на самом деле любое явное ОДУ можно преобразовать в уравнение первого порядка f (t) = F[t, f(t)], где f имеет многомерный выход. Это наблюдение подразумевает, что нам нужно не более одной производной при рассмотрении алгоритмов ОДУ. Чтобы увидеть эту связь, мы просто вспомним, что d 2y/dt2 = d/dt(dy/dt). Таким образом, мы можем определить промежуточную переменную z dy/dt и понимать d 2y/dt2 как dz/dt с ограничением z = dy/dt. В более общем случае, если мы хотим решить явную задачу

$$f(k)(t) = F[t, f(t), f(t), f(t), \ldots, f(k \quad 1)(t)],$$

где f : R Rn , то мы определяем g(t) : R Rkn с помощью ОДУ первого порядка:

$$\frac{г}{дт} \begin{matrix} г1(т) \\ г2(т) \\ \vdots \\ gk \quad 1(t) \\ gk(t) \end{matrix} = \begin{matrix} г2(т) \\ г3(т) \\ \vdots \\ гк(т) \\ F[t, g1(t), g2(t), \ldots, gk \quad 1(t)] \end{matrix}$$

Здесь мы обозначаем, что gi(t) : R Rn содержит n компонент g. Тогда g1(t) удовлетворяет исходному ОДУ. Чтобы убедиться в этом, мы просто проверим, что из приведенного выше уравнения следует, что g2(t) = g 1 (t), g3(t) = g 2 (t) = g 1 (t) и так далее. Таким образом, выполнение этих замен показывает, что последняя строка кодирует исходный ODE.

Приведенный выше прием упростит наши обозначения, но следует соблюдать некоторую осторожность, чтобы понять, что этот подход не упрощает вычисления. В частности, во многих случаях наша функция f(t) будет иметь только один выход, а ОДУ будет состоять из нескольких производных. Заменим этот случай одной производной и несколькими выходами.

Пример 13.5 (расширение ОДУ). Предположим, мы хотим решить y = 3y 2y + y, где y(t) : R R.
Это уравнение эквивалентно:

$$\frac{г}{дт}\begin{matrix} у \\ г \\ ж \end{matrix} \quad "=" \quad \begin{matrix} 0\ 1\ 0 \\ 0\ 0\ 1 \\ 3 \end{matrix} \quad 1 \quad 2 \quad \begin{matrix} у \\ г \\ ж \end{matrix}$$

Подобно тому, как наш вышеприведенный трюк позволяет нам рассматривать только ОДУ первого порядка, мы можем еще больше ограничить наши обозначения автономными ОДУ. Эти уравнения имеют вид f (t) = F[f(t)], т. е. F больше не зависит от t. Для этого мы могли бы определить

$$г\ (т) \quad \begin{matrix} f(t) \\ g^{-}(t) \end{matrix} \quad .$$

Тогда вместо этого мы можем решить следующее ОДУ для g:

$$г\ (т) = \quad \begin{matrix} f\ (т) \\ g^{-}\ (т) \end{matrix} \quad "=" \quad \begin{matrix} F[\ f(t),\ g^{-}(t)] \\ 1 \end{matrix} \quad .$$

В частности, $g^{-}(t) = t$, если принять $g^{-}(0) = 0$.

Поведение ОДУ можно визуализировать разными способами, как показано на рисунке НОМЕР. Например, если неизвестная функция f(t) является функцией одной переменной, то мы можем думать о F[f(t)] как о характеристике наклона f(t), как показано на рисунке НОМЕР. В качестве альтернативы, если f(t) имеет выход в R2 , , мы больше не можем визуализировать зависимость от времени t, но можем нарисовать фазовое пространство, которое показывает тангенс f(t) при каждом (x, y) R2

13.2.2 Существование и уникальность

Прежде чем мы сможем перейти к дискретизации задачи о начальных значениях, мы должны кратко признать, что не все задачи дифференциальных уравнений разрешимы. Кроме того, некоторые дифференциальные уравнения допускают несколько решений.

Пример 13.6 (Неразрешимое ОДУ). Рассмотрим уравнение y = 2y/t с заданным y(0) = 0; обратите внимание, что мы не делим на ноль, потому что y(0) предписывается. Переписать как

$$\frac{1}{y}\frac{dy}{dt} \quad "=" \quad \frac{2}{т}$$

и интегрирование по t с обеих сторон показывает:

$$пер\ |y| = 2\ \ln t + c \text{ Или,}$$

что то же самое, y = Ct2 для некоторого C R. Обратите внимание, что y(0) = 0 в этом выражении, что противоречит нашим начальным условиям. Таким образом, это ОДУ не имеет решения при заданных начальных условиях.

Пример 13.7 (неединственные решения). Теперь рассмотрим то же ОДУ с y(0) = 0. Рассмотрим y(t) , заданное равенством y(t) = Ct2 для любого C R. Тогда y (t) = 2Ct. Таким образом,

$$\frac{2г}{т} \text{ "=" } \frac{2Кт2}{т} = 2Ct = y\,(t),$$

показывая, что ОДУ решается этой функцией независимо от C. Таким образом, решения этой задачи неединственны.

К счастью, существует богатая теория, характеризующая поведение и устойчивость решений дифференциальных уравнений. Наше развитие в следующей главе будет иметь более сильный набор условий, необходимых для существования решения, но на самом деле при слабых условиях на f можно показать, что ОДУ f (t) = F[f (t)] имеет решение. Например, одна из таких теорем гарантирует локальное существование решения:

Теорема 13.1 (Локальное существование и единственность). Предположим, что F непрерывна и липшицева, т. е. F[y] F[x]2 Ly x2 для некоторого L. Тогда ОДУ f (t) = F[f(t)] допускает ровно одно решение для всех t 0 независимо от начальных условий.

В нашем последующем развитии мы будем предполагать, что ОДУ, которое мы пытаемся решить, удовлетворяет условиям такой теоремы; это предположение довольно реалистично в том смысле, что, по крайней мере локально, должно быть достаточно вырожденное поведение, чтобы разрушить такие слабые предположения.

13.2.3 Уравнения модели

Один из способов понять поведение ОДУ состоит в том, чтобы изучить поведение решений некоторых простых модельных уравнений, которые можно решить в замкнутой форме. Эти уравнения представляют собой линеаризацию более практичных уравнений и, таким образом, локально моделируют тип поведения, который мы можем ожидать.

Начнем с ОДУ в одной переменной. Учитывая наши упрощения в § 13.2.1, простейшее уравнение, с которым мы могли бы работать, было бы y = F[y], где y(t) : R R. Использование линейного приближения дало бы уравнения типа y = ay + б. Подстановка y¯ y + b/a показывает: y¯ = y = ay + b = a(y¯ b/a) + b = ay¯. Таким образом, в наших модельных уравнениях константа b просто вызывает сдвиг, и для нашего феноменологического исследования в этом разделе мы можем принять b = 0.

Согласно приведенному выше аргументу, мы локально можем понять поведение y = F[y] , изучая линейную уравнение y = ay. На самом деле, применение стандартных рассуждений исчисления показывает, что

$$y(t) = Ceat .$$

Очевидно, что есть три случая, показанные на рис. НОМЕР:

1. a > 0: в этом случае решения становятся все больше и больше; на самом деле, если y(t) и yˆ(t) оба удовлетворяют ОДУ с немного разными начальными условиями, при t они расходятся.

2. a = 0: система в этом случае решается константами; решения с разными начальными точками остаются на одном и том же расстоянии друг от друга.

3. a < 0: Тогда все решения ОДУ стремятся к 0 при t .

Мы говорим, что случаи 2 и 3 стабильны в том смысле, что незначительное возмущение y(0) дает решения, которые становятся все ближе и ближе с течением времени; случай 1 нестабилен, так как небольшая ошибка в задании входного параметра y(0) будет усиливаться с течением времени t. Нестабильные ОДУ порождают некорректные вычислительные проблемы; без тщательного рассмотрения мы не можем ожидать, что численные методы генерируют полезные решения в этом случае, поскольку даже теоретические выходные данные очень чувствительны к возмущениям входных данных. С другой стороны, устойчивые задачи хорошо поставлены, поскольку небольшие ошибки в y(0) со временем уменьшаются.

Переходя к нескольким измерениям, мы могли бы изучить линеаризованное уравнение

$$y = Ay.$$

Как объяснялось в §5.1.2, если y1, · · · ,yk — собственные векторы оператора A с собственными значениями λ1, . . . , λk и y(0) тогда c1y1 + · · · + ckyk ,=

$$y(т) = c1e \quad \lambda 1t \lambda k\, t\, y1 + \cdots + cke \quad yk .$$

Другими словами, собственные значения A занимают место a в нашем одномерном примере. Из этого результата нетрудно интуитивно понять, что многомерная система устойчива именно тогда, когда ее спектральный радиус меньше единицы.

В действительности мы хотим решить y = F[y] для общих функций F. Предполагая, что F дифференцируема, мы можем написать F[y] F[y0] + JF(y0)(y y0), что дает приведенное выше модельное уравнение после смены. Таким образом, на коротких промежутках времени мы ожидаем поведение, подобное уравнению модели. Кроме того, условия в теореме 13.1 можно рассматривать как ограничение поведения JF, обеспечивающее связь с менее локализованными теориями ОДУ.

13.3 Схемы временного шага

Перейдем теперь к описанию нескольких методов решения нелинейного ОДУ y = F[y] для потенциально нелинейных функций F. В общем случае, при заданном «временном шаге» h наши методы будут использоваться для получения оценок y (t + h) при заданном y(t). Применение этих методов итеративно генерирует оценки y0 y(t), y1 y(t + h), y2 y(t + 2h), y3 y(t + 3h) и так далее. Обратите внимание, что, поскольку F не имеет зависимости от t, механизм генерации каждого дополнительного шага такой же, как и первый, поэтому по большей части нам нужно будет описать только один шаг этих методов. Мы называем методы построения аппроксимаций y(t) интеграторами, отражающими тот факт, что они интегрируют производные во входном уравнении.

Ключевое значение для нашего рассмотрения имеет идея стабильности. Точно так же, как ОДУ могут быть стабильными или нестабильными, так же могут быть и дискретизации. Например, если h слишком велико, некоторые схемы будут накапливать ошибки с экспоненциальной скоростью; Напротив, другие методы устойчивы в том смысле, что даже если h велико, решения останутся ограниченными. Однако стабильность может конкурировать с точностью; часто схемы, устойчивые во времени, являются плохим приближением y(t), даже если они гарантированно не ведут себя дико.

13.3.1 Форвард Эйлер

Наша первая стратегия ODE исходит из нашего построения схемы прямого дифференцирования в §12.3.2:

$$F[yk\ ] = y\ (t) = \frac{yк+1 - yк}{} + O(ч)\ ч$$

Решение этой зависимости для показов yk+1

$$y_{k+1} = y_k + hF[y_k] + O(h^2) \quad y_k + hF[y_k].$$

Таким образом, прямая схема Эйлера применяет формулу справа для оценки yk+1 . Это одна из наиболее эффективных стратегий временного шага, поскольку она просто вычисляет F и прибавляет результат к yk . По этой причине мы называем его явным методом, т. е. существует явная формула для yk+1 через yk и F.

Проверить точность этого метода довольно просто. Обратите внимание, что наше приближение), поэтому каждый yk +1 равно O(h ^{2}шаг вызывает квадратичную ошибку. Мы называем эту ошибку локализованной ошибкой усечения, потому что это ошибка, вызванная одним шагом; слово «усечение» относится к тому факту, что мы усекли ряд Тейлора, чтобы получить эту формулу. Конечно, наш итератейк уже может быть неточным из-за накопленных ошибок усечения от предыдущих итераций. Если мы проинтегрируем от t0 до t с шагами O(1/h) , то наша общая ошибка будет выглядеть как O(h); эта оценка представляет собой глобальную ошибку усечения, и поэтому мы обычно пишем, что прямая схема Эйлера является «точной первого порядка».

Стабильность этого метода требует несколько большего рассмотрения. В нашем обсуждении мы будем работать над устойчивостью методов в случае одной переменной y = ay, имея в виду, что аналогичные утверждения переносятся на многомерные уравнения путем замены a на спектральный радиус. В этом случае мы знаем

$$y_{k+1} = y_k + ahy_k = (1 + ah)y_k .$$

Другими словами, yk = (1 + ah) ky0. Таким образом, интегратор устойчив при |1 + ah| 1, так как иначе |yk | экспоненциально. Предполагая, что a < 0 (иначе задача некорректна), мы можем упростить:

$$|1 + ax| \quad 1 \qquad 1 \quad 1 + ah \quad 1 \qquad 2$$

$$ah \quad 0$$

$$0 \quad h \quad |a| \quad \frac{2}{}, \text{ поскольку } a < 0$$

Таким образом, прямой Эйлер допускает ограничение шага по времени для устойчивости, определяемое нашим окончательным условием на h. Другими словами, выход прямого метода Эйлера может резко увеличиться, даже если y = ay стабильно, если h недостаточно мало. На рисунке НОМЕР показано, что происходит, когда это условие соблюдается или нарушается. В нескольких измерениях мы можем заменить это ограничение аналогичным ограничением, используя спектральный радиус A. Для нелинейных ОДУ эта формула дает руководство по стабильности, по крайней мере, локально во времени; глобально h, возможно, придется скорректировать, если якобиан F станет хуже обусловленным.

13.3.2 Обратный Эйлер

Точно так же мы могли бы применить схему обратного дифференцирования в yk+1 для разработки интегратора ОДУ:

$$F[y_{k+1}] = y(t) = h \qquad \frac{y_{k+1} - y_k}{} + O(h)$$

Таким образом, мы решаем следующую потенциально нелинейную систему уравнений относительно yk+1 :

$$y_k = y_{k+1} \quad hF[y_{k+1}].$$

Поскольку мы должны решить это уравнение для yk+1 , обратный Эйлер является неявным интегратором.

Этот метод обладает точностью первого порядка, как и форвард Эйлер, по идентичному доказательству. Однако стабильность этого метода значительно отличается от метода Эйлера. Снова рассматривая модельное уравнение y = ay, пишем:

$$y_k \quad y_k = y_{k+1} \qquad h a y_{k+1} = \qquad y_{k+1} = \frac{}{1 \qquad h a}.$$

Параллельно с нашим предыдущим аргументом обратный Эйлер стабилен при следующем условии:

$$\frac{1}{|1 \quad h a|} \qquad 1 \qquad |1 \quad h a| \qquad 1 \quad |1$$

$$1 \quad h a \qquad 1 \text{ или } 1 \quad h a \quad 1$$

$$2 \qquad h \quad \text{или} \quad h \quad 0 \text{ при } a < 0$$

Очевидно, мы всегда берем h 0, поэтому обратный Эйлер безусловно устойчив.

Конечно, даже если обратный Эйлер устойчив, он не обязательно точен. Если h слишком велико, y_k слишком быстро приблизится к нулю. При моделировании ткани и других физических материалов, для реалистичности которых требуется большое количество высокочастотных деталей, обратный метод Эйлера может оказаться неэффективным выбором. Кроме того, мы должны инвертировать F[·] , чтобы найти y_{k+1} .

Пример 13.8 (обратный Эйлер). Предположим, мы хотим решить y = Ay для A Rn×n . Затем, чтобы найти y_{k+1}, решаем следующую систему:

$$y_k = y_{k+1} \qquad h A y_{k+1} = \qquad y_{k+1} = (I_{n\times n} \qquad h A) \qquad {}_{1\ гк}.$$

13.3.3 Трапециевидный метод

Предположим, что y_k известно в момент времени t_k , а y_{k+1} представляет значение в момент времени $t_{k+1} = t_k + h$. Предположим, что мы также знаем $y_{k+1/2}$ на полпути между этими двумя шагами. Тогда из нашего вывода центрированного дифференцирования мы знаем:

$$y_{k+1} = y_k + hF[y_{k+1/2}] + O(h^3)$$

Из нашего вывода правила трапеций:

$$\frac{F[y_{к+1}] + F[y_к]}{2} = F[y_{k+1/2}] + O(h^2)$$

Подстановка этого соотношения дает нашу первую схему интегрирования второго порядка, метод трапеций для интегрирования ОДУ:

$$y_{k+1} = y_k + h2\frac{F[y_{k+1}] + F[y_k]}{}$$

Подобно обратному Эйлеру, этот метод является неявным, поскольку мы должны решить это уравнение относительно y_{k+1} .

Снова проводя анализ устойчивости по y = ay, находим в этом случае временные шаги метода трапеций решения

$$y_{к+1} = y_к + \frac{1}{2} - га(y_{к+1} + y_к)$$

Другими словами,

$$y_к = \frac{1 + \frac{1}{2}га}{1 - \frac{1}{2}хак}\ y_0.$$

Таким образом, метод устойчив, когда

$$\frac{1 + \frac{1}{2} ha}{1 - \frac{1}{2} ha} < 1.$$

Легко видеть, что это неравенство выполняется при a < 0 и h > 0, что показывает безусловную устойчивость метода трапеций.

Однако, несмотря на более высокий порядок точности при сохранении стабильности, метод трапеций имеет некоторые недостатки, которые делают его менее популярным, чем обратный метод Эйлера. В частности, рассмотрим соотношение

$$p \quad \frac{y_{k+1}}{y_k} \text{ "=" } \frac{1 + \frac{1}{2} ha}{1 - \frac{1}{2} ha}$$

Когда a < 0, для достаточно больших h это отношение в конце концов становится отрицательным; на самом деле при h имеем R 1. Таким образом, как показано на рисунке ЧИСЛО, если временные шаги слишком велики, трапециевидный метод интегрирования имеет тенденцию демонстрировать нежелательное колебательное поведение, которое совсем не похоже на то, что мы могли бы ожидать для решений y = ay .

13.3.4 Методы Рунге-Кутты

Класс интеграторов можно вывести, сделав следующее наблюдение:

$$y_{k+1} = y_k + \int_{t_k}^{t_k + t} y(t)\,dt \text{ по основной теореме исчисления}$$

$$= y_k + \int_{t_k}^{t_k + t} F[y(t)]\,dt$$

Конечно, прямое использование этой формулы не подходит для формулировки метода временного шага, поскольку мы не знаем y(t), но тщательное применение наших квадратурных формул из предыдущей главы может привести к выработке возможных стратегий.

Например, предположим, что мы применяем метод трапеций для интегрирования. Затем находим:

$$\text{ч } y_{k+1} = y_k + \frac{}{2} (F[y_k] + F[y_{k+1}]) + O(h^3)$$

Это формула, которую мы написали для метода трапеций в § 13.3.3.

Однако , если мы не хотим вычислять yk+1 в неявном виде, мы должны найти выражение для аппроксимации). помощник F[yk+1]. Однако, используя метод Эйлера, мы знаем, что yk+1 = yk + hF[yk] + O(h) эта ^{2}Изготовление замена yk+1 не влияет на порядок аппроксимации трапецеидального временного шага, описанного выше, поэтому мы можем написать:

$$= y_k + 2 \quad \text{час} \quad (F[y_k] + F[y_k + hF[y_k]]) + O(h\, y_{k+1}^3)$$

Игнорирование O(h ^{3}) дает новую стратегию интеграции, известную как метод Хойна, которая второго порядка точного и явного.

Если мы изучим поведение устойчивости метода Хойна для y = ay для a < 0, мы знаем:

$$\text{ч } y_{k+1} = y_k + \frac{}{} (a y_k + a(y_k + a y_k))$$

$$+ 2 \qquad \frac{2}{} \text{ч } a(2 + ha)\, y_k = 1$$

$$1 = 1 + ha + \text{ч } 2 \quad - \quad \frac{2\,2\,a}{} \quad y_k$$

Таким образом, метод устойчив, когда

$$1 \quad 1 \quad 1 + ha + \text{ч } 2 \frac{}{2} \quad \frac{2\,2\,a}{} \qquad 1$$

$$4 \quad 2\,ha + \text{ч} \qquad \frac{2\,a}{} \qquad 0$$

Неравенство справа показывает, что h a < $\frac{}{2}$, а тот, что слева, всегда верен при h > 0 и |a| ,
0, поэтому условием устойчивости является $\frac{2}{|a|}$.

h Метод Хойна является примером метода Рунге-Кутты, полученного путем применения квадратурных методов к приведенному выше интегралу и подстановки шагов Эйлера в F[·]. Прямой Эйлер — это точный метод Рунге-Кутты первого порядка, а метод Хойна — метод второго порядка. Популярный метод Рунге-Кутты четвертого порядка (сокращенно «RK4») определяется следующим образом:

$$= y_k + 6 \qquad \frac{}{} \text{ч } (k1 + 2k2 + 2k3 + k4)\, y_{k+1}$$

где $k1 = F[y_k]$

$$k2 = Fy_k + \_ \quad \frac{1}{2} \text{час1}$$

$$k3 = Fy_k + \_ \quad \frac{1}{2} \text{час2}$$

$$k4 = Fy_k + hk3 \_$$

Этот метод можно получить, применив квадратурное правило Симпсона.

Методы Рунге-Кутты популярны, потому что они являются явными и поэтому их легко оценить, обеспечивая при этом высокую степень точности. Цена такой точности, однако, состоит в том, что F[·] необходимо оценивать большее количество раз. Кроме того, стратегии Рунге-Кутты могут быть расширены до неявных методов, которые могут решать жесткие уравнения.

13.3.5 Экспоненциальные интеграторы

Один класс интеграторов, который обеспечивает высокую точность, когда F[·] приблизительно линейна, заключается в явном использовании нашего решения модельного уравнения. В частности, если бы мы решали ОДУ y = Ay, используя собственные векторы A (или любой другой метод), мы могли бы найти явное решение y(t) , как объяснено в § 13.2.3. Обычно мы пишем $y_{k+1} = e^{Ah} y_k$, где e^{Ax} кодирует наше возведение в степень собственных значений (на самом деле мы можем найти матрицу e^{Ax} Теперь, если мы из этого выражения, которое решает ОДУ к времени h).

напишем

$$y = Ay + G[y],$$

где G — нелинейная, но малая функция, мы можем достичь довольно высокой точности, интегрируя часть A в явном виде, а затем отдельно аппроксимируя нелинейную часть G. Например, экспоненциальный интегратор первого порядка применяет прямой Эйлер к нелинейному члену G:

$$y_{k+1} = e^{Ax} y_k - A^{-1}(1 - e^{Ah})G[y_k]$$

Анализ, выявляющий преимущества этого метода, более сложен, чем то, что мы написали, но интуитивно ясно, что эти методы будут вести себя особенно хорошо, когда G мало.

13.4 Многозначные методы

Преобразования в §13.2.1 позволили нам значительно упростить обозначения в предыдущем разделе, приведя все явные ОДУ к форме y = F[y]. На самом деле, хотя все явные ОДУ могут быть записаны таким образом, неясно, всегда ли они должны это делать.

В частности, когда мы сводили ОДУ k-го порядка к ОДУ первого порядка, мы вводили ряд переменных, представляющих производные от первого до k - 1-го желаемого результата. Фактически, в нашем окончательном решении нас интересует только нулевая производная, то есть сама функция, поэтому порядки точности временных переменных менее важны.

С этой точки зрения рассмотрим ряд Тейлора.

$$y(t_k + h) = y(t_k) + hy(t_k) + \frac{h^2}{2}y(t_k) + O(h^3)$$

Если мы знаем только y с точностью до $O(h^2)$, это не влияет на нашу аппроксимацию, так как y умножается на h. Точно так же, если мы знаем только y до O (h), это приближение не повлияет на приведенные выше члены ряда Тейлора, потому что оно будет умножено на h 2/2. Таким образом, теперь мы рассматриваем «многозначный» meth (k) (t) = F[t,y (t),y (t), . . . ,y различных производных функции (k 1) (t)] с разным порядком точности для од, предназначенных для интегрирования y y.

Учитывая важность второго закона Ньютона F = ma, мы ограничимся случаем y = F[t,y,y]; существует множество расширений для менее распространенного случая k-го порядка. Введем вектор «скорости» v(t) = y (t) и вектор «ускорения» a. Используя наше предыдущее сокращение, мы хотим решить следующую систему первого порядка:

$$\begin{aligned} y(t) &= v(t) \\ v(t) &= a(t) \\ a(t) &= F[t,y(t),v(t)] \end{aligned}$$

Наша цель — разработать интегратор, специально адаптированный для этой системы.

13.4.1 Схемы Ньюмарка

Начнем с вывода знаменитого класса интеграторов Ньюмарка.[1] Обозначим y_k , v_k и a_k как векторы положения, скорости и ускорения в момент времени t_k ; наша цель — перейти ко времени t_{k+1} $t_k + h$.

---

[1] Мы следим за развитием в http://www.stanford.edu/group/frg/course_work/AA242B/CA-AA242B-Ch7.pdf.

Используйте y(t), v(t) и a(t) для обозначения функций времени, предполагая, что мы начинаем с tk . Тогда, очевидно мы можем написать

$$v_{k+1} = v_k + \int_{t_k}^{t_{k+1}} a(\tau)\, d\tau$$

Мы также можем записать yk+1 как интеграл, включающий a(t), выполнив несколько шагов:

$$\begin{aligned}
y_{k+1} &= y_k + \int_{t_k}^{t_{k+1}} v(t)\, dt \\
&= y_k + [tv(t)]_{t_k}^{t_{k+1}} - \int_{t_k}^{t_{k+1}} ta(t)\, dt \text{ после интегрирования по частям} \\
&= y_k + t_{k+1}v_{k+1} - t_k v_k - \int_{t_k}^{t_{k+1}} ta(t)\, dt \text{ путем разложения разностного члена} \\
&= y_k + hv_k + t_{k+1}v_{k+1} - t_{k+1}v_k - \int_{t_k}^{t_{k+1}} ta(t)\, dt \text{ путем сложения и вычитания } hv_k \\
&= y_k + hv_k + t_{k+1}(v_{k+1} - v_k) - \int_{t_k}^{t_{k+1}} ta(t)\, dt \text{ после факторинга} \\
&= y_k + hv_k + t_{k+1}\int_{t_k}^{t_{k+1}} a(\tau)\, d\tau - \int_{t_k}^{t_{k+1}} ta(t)\, dt, \text{ так как } v(t) = a(t) \\
&= y_k + hv_k + \int_{t_k}^{t_{k+1}} (t_{k+1} - t)a(t)\, dt
\end{aligned}$$

Предположим, мы выбрали $\tau \in [t_k, t_{k+1}]$. Тогда мы можем написать выражения для a_k и a_{k+1}, используя ряд Тейлора относительно τ:

$$\begin{aligned}
a_k &= a(\tau) + a'(\tau)(t_k - \tau) + O(h^2) \\
a_{k+1} &= a(\tau) + a'(\tau)(t_{k+1} - \tau) + O(h^2)
\end{aligned}$$

Для любой константы $\gamma \in \mathbb{R}$, если мы масштабируем первое уравнение на $1 - \gamma$, а второе на γ и суммируем результаты, мы находим:

$$\begin{aligned}
a(\tau) &= (1 - \gamma)a_k + \gamma a_{k+1} + a'(\tau)((\gamma - 1)(t_k - \tau) - \gamma(t_{k+1} - \tau)) + O(h^2) \\
&= (1 - \gamma)a_k + \gamma a_{k+1} + a'(\tau)(\tau - h\gamma - t_k) + O(h^2) \text{ после замены } t_{k+1} = t_k + h
\end{aligned}$$

В конце концов, мы хотим проинтегрировать от t_k до t_{k+1}, чтобы получить изменение скорости. Таким образом, мы вычисляем:

$$\begin{aligned}
\int_{t_k}^{t_{k+1}} a(\tau)\, d\tau &= (1 - \gamma)ha_k + \gamma ha_{k+1} + \int_{t_k}^{t_{k+1}} a'(\tau)(\tau - h\gamma - t_k)\, d\tau + O(h^3) \\
&= (1 - \gamma)ha_k + \gamma ha_{k+1} + O(h^2),
\end{aligned}$$

где выполняется второй шаг, поскольку подынтегральная функция равна $O(h)$, а интервал интегрирования имеет ширину h. Другими словами, теперь мы знаем:

$$v_{k+1} = v_k + (1 - \gamma)ha_k + \gamma ha_{k+1} + O(h^2)$$

Чтобы сделать аналогичное приближение для y_{k+1}, мы можем написать

$$\begin{aligned}
\int_{t_k}^{t_{k+1}} (t_{k+1} - t)a(t)\, dt &= \int_{t_k}^{t_{k+1}} (t_{k+1} - \tau)((1 - \gamma)a_k + \gamma a_{k+1} + a'(\tau)(\tau - h\gamma - t_k))\, d\tau + O(h^3) \\
&= \frac{1}{2}(1 - \gamma)h^2 a_k + \gamma h^2 a_{k+1} + O(h^2) \text{ по тому же принципу.}
\end{aligned}$$

Таким образом, мы можем использовать наши более ранние отношения, чтобы показать:

$$y_{k+1} = y_k + hv_k + \int_{t_k}^{t_{k+1}} (t_{k+1} - t)a(t)\,dt \text{ из предыдущего}$$

$$= y_k + hv_k + \left(\frac{1}{2} - \beta\right)h^2 a_k + \beta h^2 a_{k+1} + O(h^3)$$

Здесь мы используем β вместо γ (и поглощаем коэффициент два в процессе), потому что γ , который мы выбираем для аппроксимации y_{k+1}, не обязательно должен быть таким же, как тот, который мы выбираем для аппроксимации v_{k+1} .

После всего этого интегрирования мы получили класс схем Ньюмарка с двумя входными параметры γ и β, который имеет точность первого порядка по приведенному выше доказательству:

$$y_{k+1} = y_k + hv_k + \left(\frac{1}{2} - \beta\right)h^2 a_k + \beta h^2 a_{k+1}$$

$$v_{k+1} = v_k + (1 - \gamma)ha_k + \gamma h a_{k+1}$$

$$a_k = F[t_k, y_k, v_k]$$

Разный выбор β и γ приводит к разным схемам. Например, рассмотрим следующие примеры:

- $\beta = \gamma = 0$ дает интегратор постоянного ускорения:

$$y_{k+1} = y_k + hv_k + \frac{1}{2}h^2 a_k$$

$$v_{k+1} = v_k + ha_k$$

  Этот интегратор является явным и выполняется точно, когда ускорение является постоянной функцией.

- $\beta = 1/2$, $\gamma = 1$ дает постоянный неявный интегратор ускорения:

$$y_{k+1} = y_k + hv_k + \frac{1}{2}h^2 a_{k+1}$$

$$v_{k+1} = v_k + ha_{k+1}$$

  Здесь скорость неявно ступенчата с использованием обратного Эйлера, что дает точность первого порядка. Обновление y, однако, может быть записано

$$y_{k+1} = y_k + \frac{1}{2}h(v_k + v_{k+1}),$$

  показывая, что он локально обновляется по правилу средней точки; это наш первый пример схемы, в которой обновления скорости и положения имеют разные порядки точности. Но даже в этом случае можно показать, что этот метод, однако, имеет глобальную точность первого порядка по y.

- $\beta = 1/4$, $\gamma = 1/2$ после некоторой алгебры дает следующую схему трапеций второго порядка:

$$x_{k+1} = x_k + \frac{1}{2}h(v_k + v_{k+1})$$

$$v_{k+1} = v_k + \frac{1}{2}h(a_k + a_{k+1})$$

• β = 0, γ = 1/2 дает схему центральных разностей второго порядка точности. В каноническом форма, у нас есть

$$x_{k+1} = x_k + hv_k + h\,a_k\,2^{\frac{1}{2}}$$

$$в_{к+1} = в_к + 2^{\frac{1}{2}}\,ч(а_к + а_{к+1})$$

Метод получил свое название, потому что упрощение приведенных выше уравнений приводит к альтернативной форме:

$$v_{k+1} = \frac{y_{k+2} \quad y_k}{2h\ y_{k+1}}$$

$$h\ 2 \qquad \frac{2y_{k+1} + y_k\ a_{k+1} =}{}$$

Можно показать, что методы Ньюмарка безусловно устойчивы при 4β > 2γ > 1, а точность второго порядка возникает именно при γ = 1/2 (ПРОВЕРКА).

13.4.2 Ступенчатая сетка

Другой способ достижения точности второго порядка iny заключается в использовании центрированных разностей относительно времени $t_{k+1/2} \quad t_k + h/2$:

$$y_{k+1} = y_k + hv_{k+1/2}$$ Вместо

того, чтобы использовать аргументы Тейлора, чтобы попытаться переместить $v_{k+1/2}$, мы можем просто сохранить скорости v в половине точек на сетке временных шагов.

Затем мы можем использовать аналогичное обновление, чтобы увеличить скорость:

$$в_{к+3/2} = в_{к+1/2} + ха_{к+1}\ .$$

Обратите внимание, что это обновление на самом деле имеет второй порядок точности и для x, поскольку, если мы подставим наши выражения для $v_{k+1/2}$ и $v_{k+3/2}$, мы можем написать:

$$а_{к+1} = \frac{1}{ч\ 2}(у_{к+2} - 2у_{к+1} + у_к)$$

Наконец, для члена ускорения достаточно простого приближения, поскольку это член более высокого порядка:

$$a_{k+1} = F\ t_{k+1}\ ,x_{k+1}\ ,\ 2 \qquad ^{-1}(v_{k+1/2} + v_{k+3/2})$$

Это выражение можно подставить в выражение для $v_{k+3/2}$.

Когда F[·] не зависит от v, метод полностью явный:

$$y_{k+1} = y_k + hv_{k+1/2}$$

$$a_{k+1} = F[t_{k+1}\ ,y_{k+1}\ ]$$

$$v_{k+3/2} = v_{k+1/2} + ha_{k+1}$$

Это известно как метод интегрирования чехарды благодаря шахматной временной сетке и тому факту, что каждая средняя точка используется для обновления следующей скорости или положения.

В противном случае, если обновление скорости зависит от v, метод становится неявным.
Часто зависимость от скорости симметрична; например, сопротивление ветра просто меняет знак, если вы движетесь в обратном направлении. Это свойство может привести к симметричным матрицам на неявном этапе обновления скоростей, что позволяет использовать сопряженные градиенты и связанные с ними быстрые итерационные методы для решения.

13.5 Сделать

- Определить жесткое ОДУ
- Приведите таблицу методов временного шага для F[t;y]
- Используйте обозначение Y более последовательно.

13.6 Проблемы

- ТВД РК
- Многошаговые/многозначные методы а-ля Хит
- Интеграция Verlet
- Симплектические интеграторы

Глава 14

Уравнения с частными производными

Наше интуитивное представление об обыкновенных дифференциальных уравнениях обычно основано на эволюции физических систем во времени. Уравнения, подобные второму закону Ньютона, определяющему движение физических объектов во времени, доминируют в литературе по таким проблемам с начальными значениями; дополнительные примеры приходят из химических концентраций, реагирующих с течением времени, популяций хищников и жертв, взаимодействующих от сезона к сезону, и так далее. В каждом случае исходная конфигурация — например, положения и скорости частиц в системе в нулевое время — известна, и задача состоит в том, чтобы предсказать поведение с течением времени.

Однако в этой главе мы рассматриваем возможность связывания отношений между различными производными функции. Нетрудно найти примеры, когда такая связь необходима. Например, при моделировании количества дыма или газов, таких как «градиенты давления», производные от давления газа в пространстве, учитывайте, как газ движется во времени; такая структура является разумной, поскольку газ естественным образом диффундирует из областей высокого давления в области низкого давления. При обработке изображений производные связываются еще более естественно, поскольку измерения изображений, как правило, происходят одновременно в направлениях x и y.

Уравнения, связывающие вместе производные функций, известны как уравнения в частных производных. Они являются предметом богатой, но очень тонкой теории, достойной более широкого рассмотрения, поэтому наша цель здесь будет состоять в том, чтобы обобщить ключевые идеи и предоставить достаточный материал для решения проблем, обычно возникающих на практике.

14.1 Мотивация

Уравнения в частных производных (УЧП) возникают, когда неизвестной является некоторая функция $f : \mathbb{R}^n \to \mathbb{R}^m$. Нам задано одно или несколько соотношений между частными производными f, и цель состоит в том, чтобы найти f, удовлетворяющее критериям. PDE появляются практически в любой области прикладной математики, и мы перечислим лишь некоторые из них ниже.

Кроме того, прежде чем вводить конкретные УЧП, мы должны ввести некоторые обозначения. В частности, есть несколько комбинаций частных производных, которые часто встречаются в мире частных производных. Если $f : \mathbb{R}^3 \to \mathbb{R}$ и $v : \mathbb{R}^3 \to \mathbb{R}^3$, то стоит запомнить следующие операторы:

Градиент: $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)$

Дивергенция: $\cdot v + + x1 \frac{v1}{v2} \frac{v2}{x2} \frac{v3}{x3} \frac{}{v3} \frac{}{2} \frac{}{2}$

Завиток: $\times v \quad \frac{2 fff}{} \quad \frac{}{}, \frac{v1}{x3} \quad \frac{v3}{x1}, \frac{v2}{x1} \quad \frac{v1}{x2}$

Лапласиан: $2 f + + x \frac{}{2} \; x 2 \; \frac{}{x 2 1 2 3}$

Пример 14.1 (Моделирование жидкости). Поток жидкостей, таких как вода и дым, регулируется уравнениями Навье-Стокса, системой УЧП со многими переменными. В частности, предположим, что в некоторой области Ω R3 движется жидкость . Мы определяем следующие переменные, показанные на рисунке NUMBER:

- t [0,): время
- v(t) : Ω R3 : скорость жидкости
- ρ(t) : Ω R: плотность жидкости
- p(t) : Ω R: давление жидкости
- f(t) : Ω R3 : Внешние силы, такие как гравитация, действующие на жидкость.

Если жидкость имеет вязкость μ, то, если предположить, что она несжимаема, уравнения Навье-Стокса утверждают:

$$p \frac{v}{t} + v \cdot v = p + \mu 2 v + f$$

Здесь $2v = 2v1/x2 + 2v2/x2 + 2v3/x2$; мы думаем о градиенте $_3$ как о градиенте в пространстве, а не как о $f f x1, x2,$ времени, т. е. $f = \frac{}{} \frac{}{} \frac{}{f x3}$). Эта система уравнений определяет временную динамику движения жидкости и (на самом деле можно построить, применяя второй закон Ньютона к отслеживанию «частиц» жидкости. Однако его формулировка включает не только производные по времени $\frac{}{t}$ но также и производные в пространстве , что делает его УЧП.

Пример 14.2 (уравнения Максвелла). Уравнения Максвелла определяют взаимодействие электрических полей E и магнитных полей B во времени. Как и в случае с уравнениями Навье-Стокса, мы думаем о градиенте, дивергенции и завитке как о частных производных по пространству (а не по времени t). Тогда систему Максвелла (в «сильной» форме) можно записать:

Закон Гаусса для электрических полей: $\cdot E = \frac{p}{\varepsilon 0}$

Закон Гаусса для магнетизма: $\cdot B = 0$

Закон Фарадея: $\times E = t \frac{B}{}$

Закон Ампера: $\times B = \mu 0 J + \varepsilon 0 t \frac{E}{}$

Здесь ε0 и μ0 — физические константы, а J кодирует плотность электрического тока. Как и уравнения Навье-Стокса, уравнения Максвелла связывают производные физических величин по времени t с их производными в пространстве (задаваемыми терминами ротора и дивергенции).

Пример 14.3 (уравнение Лапласа). Предположим, Ω — область в R2 с границей Ω и что нам дана функция g : Ω R, показанная на рис. НОМЕР. Мы можем захотеть интерполировать g внутрь Ω. Однако, когда Ω имеет неправильную форму, наши стратегии интерполяции из главы 11 могут не сработать.

Предположим, мы определяем f(x) : Ω R как интерполирующую функцию. Затем одна стратегия, вдохновленная нашим подходом к методу наименьших квадратов, заключается в определении функционала энергии:

$$E[f] = \int_{\Omega} \phi(x)\ {}^{2}_{2}\ dx$$

То есть E[f] измеряет «полную производную» f, измеренную путем взятия нормы ее градиента и интегрирования этой величины по всему Ω. Сильно флуктуирующие функции f будут иметь высокие значения E[f] , поскольку во многих местах наклон f будет большим; гладкие и низкочастотные функции f , с другой стороны, будут иметь малое E[f] , поскольку их наклон будет мал везде.1 Тогда мы могли бы потребовать, чтобы f интерполировала g, будучи как можно более гладкой внутри Ω, используя следующая оптимизация:

$$\begin{aligned} &\text{минимизировать} \\ &f\ E[f] \text{ такое, что } f(x) = g(x) \quad x \quad \Omega \end{aligned}$$

Эта установка выглядит как оптимизация, которую мы решили в предыдущих примерах, но теперь наша неизвестная — это функция f, а не точка в Rn !

Если f минимизирует E, то E[f + h] E[f] для всех функций h(x). Это утверждение верно даже для малых возмущений E[f + εh] при ε 0. Деля на ε и переходя к пределу при ε 0, мы должны иметь $E[f + \varepsilon h]|_{\varepsilon=0} \frac{d}{d\varepsilon} 0$; это то же самое, что приравнять производные по направлению к нулю, чтобы найти ее минимумы. Мы можем упростить:

$$E[f + \varepsilon h] = \int_{\Omega} f(x) + \varepsilon \quad h(x)\ {}^{2}\ dx\ {}^{2}$$

$$\text{"="} \int_{\Omega} \left(\quad f(x)\ {}^{2}_{2} + 2\varepsilon \quad f(x) \cdot \quad h(x) + \varepsilon\ {}^{2}\ \text{ч}(x)\ {}^{2}_{2} \right) dx$$

Дифференциальные шоу:

$$\frac{d}{d\varepsilon} E[f + \varepsilon h] = \int_{\Omega} (2 \quad f(x) \cdot \quad h(x) + 2\varepsilon \quad h(x)\ {}^{2}_{2})\ dx$$

$$\frac{d}{d\varepsilon} E[f + \varepsilon h]|_{\varepsilon=0} = 2 \int_{\Omega} [\quad f(x) \cdot \quad h(x)]\ dx$$

Эта производная должна равняться нулю для всех h, так что, в частности, мы можем выбрать h(x) = 0 для всех x Ω. Тогда, применяя интегрирование по частям, имеем:

$$\frac{d}{d\varepsilon} E[f + \varepsilon h]|_{\varepsilon=0} = \quad 2 \int_{\Omega} h(x) \quad 2\ f(x)\ dx$$

Это выражение должно быть равно нулю для всех (всех!) возмущений h, поэтому мы должны иметь 2 f(x) = 0 для всех x Ω\ Ω (формальное доказательство выходит за рамки нашего обсуждения). То есть проблема интерполяции выше

1 Используемое здесь обозначение E[·] не означает «ожидание», как это могло бы быть в теории вероятностей, а скорее просто функционал «энергия»; это стандартное обозначение в областях функционального анализа.

можно решить с помощью следующего уравнения в

частных производных:

$$2 f(x) = 0 \quad f(x) = g(x) \quad x \quad \Omega$$

Это уравнение известно как уравнение Лапласа, и его можно решить, используя разреженные положительно определенные линейные методы, подобные тому, что мы рассмотрели в главе 10. Как мы видели, его можно применять к задачам интерполяции для нерегулярных областей Ω; кроме того, E[f] может быть увеличено для измерения других свойств f , например, насколько хорошо f аппроксимирует некоторую зашумленную функцию f0, чтобы получить связанные УЧП путем параллельного рассуждения выше.

Пример 14.4 (уравнение Эйконала). Предположим, что Ω Rn — некоторая замкнутая область пространства. Тогда мы могли бы взять d(x) как функцию, измеряющую расстояние от некоторой точки x0 до x полностью в пределах Ω. Когда Ω выпукло, мы можем записать d в закрытой форме:

$$d(x) = x \quad x02.$$

Однако, как показано на рисунке НОМЕР, если Ω невыпуклая или представляет собой более сложную область, такую как поверхность, вычисление расстояний усложняется. В этом случае функции расстояния d удовлетворяют локализованному условию, известному как уравнение эйконала:

$$d2 = 1.$$

Если мы сможем его вычислить, d можно будет использовать для таких задач, как планирование пути роботов путем минимизации расстояния, которое они должны пройти, с ограничением, что они могут двигаться только в Ω.

Специализированные алгоритмы, известные как методы быстрого марша, используются для нахождения оценок d при данных x0 и Ω путем интегрирования уравнения эйконала. Это уравнение нелинейно относительно производной d, поэтому методы интегрирования для этого уравнения несколько специализированы, а доказательство их эффективности сложно. Интересно, но неудивительно, что многие алгоритмы решения уравнения эйконала имеют структуру, аналогичную алгоритму Дейкстры для вычисления кратчайших путей на графах.

Пример 14.5 (Гармонический анализ). Различные объекты по-разному реагируют на вибрации, и в значительной степени эти реакции зависят от геометрии объектов. Например, виолончели и фортепиано могут играть одну и ту же ноту, но даже неопытный музыкант легко различает издаваемые ими звуки. С математической точки зрения мы можем принять Ω R3 как форму, представленную либо в виде поверхности, либо в виде объема.
Если зажать ребра фигуры, то ее частотный спектр задается решениями следующей дифференциальной задачи на собственные значения:

$$2 f = \lambda f \quad f(x)$$

$$= 0 \quad x \quad \Omega, \text{ где } 2 —$$

лапласиан области Ω, а Ω — граница области Ω. На рисунке НОМЕР показаны примеры этих функций в разных областях Ω.

Легко проверить, что sin kx решает эту задачу, когда Ω — интервал [0, 2π] для k Z. В частности, лапласиан в одном измерении есть 2/ x 2 , и таким образом мы можем проверить:

$$\frac{2}{} \quad \sin kx = k \cos kx \quad x \, 2 \quad x$$

$$2 = \quad k \sin kx$$

$$\sin k \cdot 0 = 0$$

$$\sin k \cdot 2\pi = 0$$

Таким образом, собственными функциями являются sin kx с собственными значениями k 2 .

14.2 Основные определения

Используя обозначения CITE, мы будем предполагать, что наша неизвестная есть некоторая функция f : Rn R. Для уравнений, содержащих до трех переменных, мы будем использовать индексы для обозначения частных производных:

$$\phi x \quad \frac{}{x},$$

$$f\ fy \quad \frac{}{y},$$

$$f\ fxy \quad \frac{2}{x\ y},$$

и так далее.

Частные производные обычно указываются как отношения между двумя или более производными f в следующем:как

- Линейная, однородная: fxx + fxy fy = 0
- Линейная: fxx y fyy + f = xy2
- Нелинейный: f^2_{xx} = ϕy

Как правило, мы действительно хотим найти f : Ω R для некоторого Ω Rn . Точно так же, как ОДУ были сформулированы как задачи с начальным значением, мы будем формулировать большинство УЧП как задачи с граничными значениями. То есть наша задача будет заключаться в том, чтобы заполнить f внутри Ω заданными значениями на его границе. На самом деле, мы можем думать о проблеме начального значения ОДУ следующим образом: областью является Ω = [0,) с границей Ω = {0}, где мы предоставляем входные данные. Рисунок NUMBER иллюстрирует более сложные примеры. Граничные условия для этих задач принимают разные формы:

- Условия Дирихле просто определяют значение f(x) на Ω
- Условия Неймана задают производные f(x) на Ω
- Смешанные условия или условия Робина объединяют эти два

14.3 Уравнения модели

Вспомним из предыдущей главы, что мы смогли понять многие свойства ОДУ, исследуя модельное уравнение y = ay. Мы можем попытаться использовать аналогичный подход для PDE, хотя мы обнаружим, что история становится более тонкой, когда производные связаны друг с другом.

Как и в случае уравнения модели для ОДУ, мы будем изучать линейный случай с одной переменной. Мы также ограничимся системами второго порядка, т. е. системами, содержащими не более чем вторую производную от u; модельное ОДУ было первого порядка, но здесь нам нужно по крайней мере два порядка, чтобы изучить, как производные взаимодействуют нетривиальным образом.

Линейное УЧП второго порядка имеет следующий общий вид:

$$c = 0\ \sum_{ij} a_{ij} \frac{f\ f}{x_i\ x_j} + \sum_{я} b_i + \frac{}{x_i}$$

Формально мы можем определить «оператор градиента» как:

$$\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}.$$

Вы должны проверить, что этот оператор является разумной записью в том смысле, что такие выражения, как f , · v и × v, обеспечивают правильные выражения. В этих обозначениях УЧП можно рассматривать как принимающую матричную форму:

$$(A + \cdot b + c)f = 0.$$

Эта форма имеет много общего с нашим изучением квадратичных форм в сопряженных градиентах, и на самом деле мы обычно характеризуем УЧП структурой A:

- Если A положительно или отрицательно определено, система является эллиптической.
- Если A положительно или отрицательно полуопределенно, система является параболической.
- Если A имеет только одно собственное значение отличного от остальных знака, система является гиперболической.
- Если A не удовлетворяет ни одному из критериев, система является ультрагиперболической.

Эти критерии перечислены примерно в порядке сложности решения каждого типа уравнений. Мы рассмотрим первые три случая ниже и приведем примеры соответствующего поведения; ультрагиперболические уравнения встречаются на практике реже и требуют для своего решения узкоспециализированных методов.

TODO: Приведение к канонической форме с помощью собственного материала A (не в 205A)

14.3.1 Эллиптические УЧП

Точно так же, как положительно определенные матрицы позволяют использовать специализированные алгоритмы, такие как разложение Холецкого и сопряженные градиенты, которые упрощают их обращение, эллиптические УЧП имеют особенно сильную структуру, которая приводит к эффективным методам решения.

Модельное эллиптическое УЧП представляет собой уравнение Лапласа, задаваемое формулой 2 f = g для некоторой заданной функции g в виде в примере 14.3. Например, с двумя переменными уравнение Лапласа принимает вид

$$f_{xx} + f_{yy} = г.$$

На рисунке NUMBER показаны некоторые решения уравнения Лапласа для различных вариантов выбора u и f в двумерной области.

Эллиптические уравнения обладают многими важными свойствами. Особое теоретическое и практическое значение имеет идея эллиптической регулярности, согласно которой решения эллиптических УЧП автоматически являются гладкими функциями в C (Ω). Это свойство не сразу очевидно: УЧП второго порядка в f требует только, чтобы f было дважды дифференцируемо, чтобы иметь смысл, но на самом деле при слабых условиях они автоматически бесконечно дифференцируемы. Это свойство подтверждает физическую интуицию, согласно которой эллиптические уравнения представляют собой устойчивые физические равновесия, подобные позе покоя растянутого резинового листа.

Эллиптические уравнения второго порядка в приведенной выше форме также гарантированно допускают решения, в отличие от УЧП в некоторых других формах.

Пример 14.6 (Пуассон по одной переменной). Уравнение Лапласа с g = 0 получило специальное название уравнения Пуассона. В одной переменной можно записать f (x) = 0, что тривиально решается как f (x) = αx + β. Этого уравнения достаточно, чтобы исследовать возможные граничные условия на [a, b]:

- Условия Дирихле для этого уравнения просто определяют f(a) и f(b); очевидно, существует единственная линия, проходящая через (a, f(a)) и (b, f(b)), которая обеспечивает решение уравнения.

- Условия Неймана определяют f (a) и f (b). Но f (a) = f (b) = α для f(x) = αx + β. Таким образом, граничные значения для задач Неймана могут подчиняться условиям совместимости, необходимым для принятия решения. Кроме того, выбор β не влияет на граничные условия, поэтому при их выполнении решение не единственно.

14.3.2 Параболические УЧП

Продолжая параллель со структурой линейной алгебры, с положительно-полуопределенными системами уравнений иметь дело лишь немного сложнее, чем с положительно-определенными. В частности, положительно полуопределенные матрицы допускают нулевое пространство, с которым нужно обращаться осторожно, но в остальных направлениях матрицы ведут себя так же, как и определенный случай.

Уравнение теплопроводности представляет собой модель параболического УЧП. Предположим, что f(0; x, y) — распределение тепла в некоторой области Ω R2 в момент времени t = 0. Тогда уравнение теплопроводности определяет, как тепло распространяется с течением времени t как функция f(t; x, y) :

$$\frac{f}{t} = \alpha \quad 2f,$$

где α > 0, и мы снова считаем 2 лапласианом в пространственных переменных x и y, то есть 2 = 2/ x 2 + 2/ y 2 . Это уравнение должно быть параболическим, так как перед fxx и fyy стоит один и тот же коэффициент α , но ftt в уравнении не фигурирует.

Рисунок НОМЕР иллюстрирует феноменологическую интерпретацию уравнения теплопроводности. Мы можем думать о 2 f как об измерении выпуклости f , как на рисунке ЧИСЛО (a). Таким образом, уравнение теплопроводности увеличивает u со временем, когда его значение «закрыто» вверх, и уменьшает f в противном случае. Эта отрицательная обратная связь устойчива и приводит к равновесию при t .

Для уравнения теплопроводности необходимы два граничных условия, оба из которых прямые физические интерпретации:

- Распределение тепла f(0; x, y) в момент времени t = 0 во всех точках (x, y) Ω

- Поведение f при t > 0 в точках (x, y) Ω. Эти граничные условия описывают поведение на границе области. Здесь условия Дирихле дают f(t; x, y) для всех t 0 и (x, y) Ω, что соответствует ситуации, когда посторонний агент фиксирует температуры на границе области. Эти условия могут возникнуть, если Ω представляет собой кусок фольги, расположенный рядом с источником тепла, на температуру которого фольга не оказывает существенного влияния, например, большой холодильник или духовка. Напротив, условия Неймана задают производную f в направлении, нормальном к границе Ω, как на рисунке НОМЕР; они соответствуют фиксации потока тепла из Ω, вызванного различными типами изоляции.

14.3.3 Гиперболические УЧП

Окончательное уравнение модели представляет собой волновое уравнение, соответствующее случаю неопределенной матрицы:

$$\frac{2}{t^2} f \quad c^2 \quad 2 f = 0$$

Волновое уравнение является гиперболическим, потому что вторая производная по времени имеет противоположный знак от двух пространственных производных. Это уравнение определяет движение волн через упругую среду, подобную резиновому листу; например, его можно получить, применив второй закон Ньютона к точкам на отрезке резинки, где x и y - положения на листе, a f (t; x, y) - высота отрезка резинки в момент времени t.

Рисунок НОМЕР иллюстрирует одномерное решение волнового уравнения. Поведение волны значительно отличается от диффузии тепла в том смысле, что при t энергия может не распространяться. В частности, волны могут бесконечно отражаться взад и вперед по домену. По этой причине мы увидим, что стратегии неявного интегрирования могут не подходить для интегрирования гиперболических УЧП, поскольку они имеют тенденцию гасить движение.

Граничные условия для волнового уравнения аналогичны граничным условиям для уравнения теплопроводности, но теперь мы должны указать как f(0; x, y) , так и ft(0; x, y) в нулевое время:

- Условия при t = 0 определяют положение и скорость волны в начальный момент времени.
- Граничные условия на Ω определяют, что происходит на концах материала. Условия Дирихле соответствуют фиксации сторон волны, например, защипывание струны виолончели, которая удерживается двумя концами на инструменте. Условия Неймана соответствуют тому, что концы волны остаются нетронутыми, как конец кнута.

14.4 Производные как операторы

В PDE и в других местах мы можем думать о производных как об операторах, действующих на функции так же, как матрицы действуют на векторы. Наш выбор обозначений часто отражает эту параллель: производная d f/dx выглядит как произведение оператора d/dx и функции f . На самом деле дифференцирование является линейным оператором, как и умножение матриц, поскольку для всех f , g : R R и a, b R

$$\frac{d}{dx}(a f(x) + b g(x)) = a \frac{f}{dx}(x) + b \frac{g}{dx}$$

На самом деле, когда мы дискретизируем УЧП для численного решения, мы можем провести эту аналогию полностью. Например, рассмотрим функцию f на [0, 1] , дискретизированную с использованием n + 1 равноотстоящих отсчетов, как показано на рисунке НОМЕР. Напомним, что расстояние между двумя образцами равно h = 1/n. В главе 12 мы разработали приближение для второй производной f (x) :

$$f(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$

Предположим, что наши n выборок f(x) на [0, 1] таковы: y0 f(0), y1 f(h), y2 f(2h), . . . , yn = f(nh).
Затем применение нашей формулы выше дает стратегию для аппроксимации f в каждой точке сетки:

$$y_k \approx \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2}$$

То есть вторую производную функции на сетке точек можно вычислить с помощью шаблона 1— — 2—1, показанного на рисунке НОМЕР (a).

Одна тонкость, которую мы не рассмотрели, заключается в том, что происходитпри y 0 и y n , поскольку приведенная выше формула требует y 1 и yn+1. Фактически это решение кодирует граничные условия, введенные в §14.2.
Принимая во внимание, что y0 = f(0) и yn = f(1), примеры возможных граничных условий для f включают:

• Граничные условия Дирихле: y 1 = yn+1 = 0, то есть просто зафиксировать значение y за пределами конечные точки

• Граничные условия Неймана: y 1 = y0 и yn+1 = yn, кодирующие граничное условие f (0) = f (1) = 0.

• Периодические граничные условия: y 1 = yn и yn+1 = y0, что делает отождествление f(0) = ϕ(1)

Предположим, мы складываем выборки yk в вектор y Rn+1 , а выборки yk — вектор w Rn+1 . в секунду Тогда из нашей вышеприведенной конструкции легко увидеть, что h 2w = L1y, где L1 — один из следующих выборов:

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 \end{pmatrix} \qquad \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 2 & -1 & & & -1 \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ -1 & & & -1 & 2 \end{pmatrix}$$

Дирихле Нойманн периодический

То есть матрицу L можно рассматривать как дискретизированную версию оператора y Rn+1 , а $\frac{д}{2\ дх2}$ действующий на не функции f : [0, 1] R.

Мы можем написать аналогичную аппроксимацию для 2 f, когда мы выбираем f : [0, 1] × [0, 1] R с сеткой значений, как на рисунке НОМЕР. В частности, напомним, что в этом случае 2 f = fxx + fyy, так что, в частности, мы можем просуммировать вторые производные x и y, как мы это делали в приведенном выше одномерном примере. Это приводит к двойному трафарету 1— — 2—1, как на рисунке НОМЕР. Если мы пронумеруем наши выборки как yk, f(kh, h), то наша формула для лапласиана f в этом случае будет:

$$(\ 2\ y)k, \quad \frac{1}{h^2}(y(k\ 1), + yk,(\ 1) + y(k+1), + yk,(+1) \quad 4yk,)$$

Если мы еще раз объединим наши выборки y и y в y и w, то, используя аналогичную конструкцию и выбор граничных условий, мы снова можем записать h 2w = L2y. Этот двумерный сеточный лапласиан L2 используется во многих приложениях для обработки изображений, где (k,) используется для индексации пикселей на изображении.

Естественный вопрос, который следует задать после обсуждения выше, заключается в том, почему мы перешли к лапласиану второй производной в нашем обсуждении выше, а не к дискретизации первой производной f (x). В принципе, нет никаких причин, по которым мы не могли бы сделать аналогичные матрицы D, реализующие прямую, обратную или симметричную разностную аппроксимацию f ? Однако несколько технических моментов несколько усложняют эту задачу, как подробно описано ниже.

Самое главное, мы должны решить, какое приближение первой производной использовать. Если мы напишем , 1 как прямая разница h − например, y k (yk+1 yk), то мы окажемся в неестественно асимметричном положении, когда нам понадобится граничное условие в yn , но не в y0. Напротив, мы могли бы использовать симметричную разность ($\frac{1}{2\ часа}$ yk+1 yk 1), но эта дискретизация страдает от более тонкого ограждения проблема, показанная на рисунке НОМЕР. В частности, эта версия y k игнорирует значение yk и смотрит только на своих соседей yk 1 и yk+1 , что может создавать артефакты, поскольку каждая строка D включает yk только для четного или нечетного k, но не для обоих.

Если мы используем прямые или обратные производные, чтобы избежать проблем со столбами забора, мы теряем порядок точности, а также страдаем от описанной выше асимметрии. Как и в случае интеграции чехарды

Алгоритм в §13.4.2, один из способов избежать этих проблем состоит в том, чтобы думать о производных как о живущих в половинных узлах сетки, как показано на рисунке НОМЕР. В одномерном случае это изменение соответствует (yk+1 разницы в вершинах, ребрах и $\frac{1}{\text{час}}$ yk) как yk+1/2 . Этот метод размещения различных производных для маркировки центрах ячеек сетки особенно распространен в моделировании жидкости, которое поддерживает давление, скорость жидкости и т. д. в местах, которые упрощают вычисления.

Помимо этих тонкостей, наш главный вывод из этого обсуждения состоит в том, что если мы дискретизируем функцию f(x), отслеживая выборки (xi , yi) , то наиболее разумные приближения производных f будут вычислимы как произведение Lx для некоторой матрицы L. Это наблюдение завершает аналогию: «Производные действуют на функции, как матрицы действуют на векторы». Или в стандартизированной записи экзамена: Производные : Функции :: Матрицы : Векторы

14.5 Численное решение УЧП

Многое еще предстоит сказать о теории УЧП. Вопросы существования и уникальности, а также возможность характеристики решений различных УЧП приводят к детальным обсуждениям с использованием передовых аспектов реального анализа. Хотя полное понимание этих свойств необходимо для строгого доказательства эффективности дискретизации PDE, у нас уже есть достаточно, чтобы предложить несколько методов, которые используются на практике.

14.5.1 Решение эллиптических уравнений

Мы уже сделали большую часть работы по решению эллиптических УЧП в § 14.4. В частности, предположим, что мы хотим решить линейное эллиптическое УЧП вида Lf = g. Здесь L — дифференциальный оператор; например, чтобы решить уравнение Лапласа, мы возьмем L 2 лапласиана. Затем в §14.4 мы показали, что если мы дискретизируем f, взяв набор отсчетов в векторе y с yi = f(xi), то соответствующее приближение Lf можно записать как Ly для некоторой матрицы L. Если мы также дискретизируем g используя отсчеты в векторе b, затем решение эллиптического УЧП Lf = g аппроксимируется решением линейной системы Ly =b.

Пример 14.7 (Эллиптическая дискретизация УЧП). Предположим, мы хотим аппроксимировать решения f (x) = g(x) на [0, 1] с граничными условиями f(0) = f(1) = 0. Будем аппроксимировать f(x) вектором y Rn выборка f следующим образом:

$$\begin{matrix} y1 & \phi(ч) \\ y2 & \phi(2ч) \\ \vdots & \vdots \\ п & \phi\ (нч) \end{matrix}$$

где h = 1/n+1. Мы не добавляем образцы в x = 0 или x = 1, так как граничные условия определяют значения там. Мы будем использовать b для хранения аналогичного набора значений для g(x).

Учитывая наши граничные условия, мы дискретизируем f (x) как $\frac{1}{h^2}$ Ly, где L определяется как:

$$L = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}$$

Таким образом, наше приближенное решение УЧП определяется выражением y = h 2L 1b.

Точно так же, как эллиптические УЧП являются наиболее простыми для решения, их дискретизация с использованием матриц, как в приведенном выше примере, является наиболее простой для решения. На самом деле дискретизация эллиптического оператора L обычно представляет собой положительно определенную и разреженную матрицу, идеально подходящую для методов решения, описанных в главе 10.

Пример 14.8 (Эллиптические операторы как матрицы). Рассмотрим матрицу L из примера 14.7. Мы можем показать, что L отрицательно определена (и, следовательно, положительно определенная система Ly = h 2b может быть решена с использованием сопряженных градиентов), заметив, что L = DD для матрицы D R(n+1)×n, заданной формулой:

$$D = \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ & & & & -1 \end{bmatrix}$$

Эта матрица есть не что иное, как первая производная с конечной разностью, поэтому это наблюдение соответствует тому факту, что d 2 f/dx2 = d/dx(d f/dx). Таким образом, x Lx = x DDx = Dx 2 0, что показывает, что L является отрицательно полуопределенным. Легко видеть, что Dx = 0 именно тогда, когда x = 0, что завершает доказательство того, что L на самом деле отрицательно определена.

14.5.2 Решение параболических и гиперболических уравнений

Параболические и гиперболические уравнения обычно вводят в формулировку временную переменную, которая также является дифференцированной, но потенциально более низкого порядка. Поскольку решения параболических уравнений обладают многими свойствами устойчивости, численные методы для работы с этой переменной времени часто устойчивы и хорошо обусловлены; напротив, необходимо уделять больше внимания гиперболическому поведению и предотвращать затухание движения с течением времени.

Полудискретные методы Вероятно, наиболее простым подходом к решению простых уравнений, зависящих от времени, является использование полудискретного метода. Здесь мы дискретизируем домен, но не переменную времени, что приводит к ОДУ, которое можно решить с помощью методов из главы 13.

Пример 14.9 (полудискретное уравнение теплопроводности). Рассмотрим уравнение теплопроводности с одной переменной, заданное формулой ft = fxx, где f(t; x) представляет теплоту в точке x и в момент времени t. В качестве граничных данных пользователь предоставляет

функция f0(x) такая, что f(0; x) f0(x); мы также присоединяем границу x {0, 1} к холодильнику и тем самым обеспечиваем f(t; 0) = f(t; 1) = 0.

Предположим, мы дискретизируем переменную x, определив:

$$\begin{aligned} \bar{f}1(t) &\quad f(h; t) \\ \bar{f}2(t) &\quad f(2h; t) \\ &\vdots \quad \vdots \\ \bar{f}n(t) &\quad f(nh; t), \end{aligned}$$

где, как и в примере 14.7, мы берем h = 1/n+1 и опускаем выборки при x {0, 1} , так как они обеспечиваются граничными условиями.

Комбинируя эти fi , мы можем определить f(t) : R Rn как полудискретную версию f, в которой мы выбрали пространство, но не время. По нашей конструкции приближение полудискретного УЧП представляет собой ОДУ, заданное формулой f (t) = L f (t).

В предыдущем примере показан пример очень общего шаблона для параболических уравнений. Когда мы моделируем непрерывные явления, такие как тепло, проходящее через область, или химические вещества, диффундирующие через мембрану, обычно имеется одна временная переменная, а затем несколько пространственных переменных, которые дифференцируются эллиптическим образом. Когда мы полудискретно дискретизируем эту систему, мы можем использовать стратегии интеграции ОДУ для их решения. Фактически, точно так же, как матрица, используемая для решения линейного эллиптического уравнения, как в § 14.5.1, обычно является положительно или отрицательно определенной, когда мы записываем полудискретное параболическое УЧП f = L f , матрица L обычно является отрицательно определенной. Это наблюдение означает, что f, решающая это непрерывное ОДУ, безусловно устойчива, поскольку отрицательные собственные значения со временем затухают.

Как отмечалось в предыдущей главе, у нас есть много вариантов решения ОДУ во времени, возникающих в результате пространственной дискретизации. Если временные шаги малы и ограничены, могут быть приемлемы явные методы. Неявные решатели часто применяются для решения параболических уравнений в частных производных; диффузионное поведение неявного Эйлера может привести к неточности, но с точки зрения поведения похоже на диффузию, обеспечиваемую уравнением теплопроводности, и может быть приемлемо даже при довольно больших временных шагах. Гиперболические УЧП могут потребовать неявных шагов для стабильности, но расширенные интеграторы, такие как «симплектические интеграторы», могут предотвратить чрезмерное сглаживание, вызванное этими типами шагов.

Один противоположный подход состоит в том, чтобы записывать решения полудискретных систем f = L f через собственные векторы L. Предположим, что v1, . . . ,vn — собственные векторы оператора L с собственными значениями λ1, . . . , λn и что мы знаем f(0) = c1v1 + · · · + cnvn. Затем вспомните, что решение f = L f определяется выражением:

$$f(\tau) = \sum_i c_i e^{\lambda_i t} v_i$$

В этой формуле нет ничего нового, кроме §5.1.2, который мы ввели во время обсуждения собственных векторов и собственных значений. Однако собственные векторы оператора L могут иметь физический смысл в случае полудискретного УЧП, как в примере 14.5, где показано, что собственные векторы лапласиана L соответствуют различным резонансным колебаниям области. Таким образом, этот подход на основе собственных векторов можно применять, например, для разработки «низкочастотных аппроксимаций» исходных данных путем усечения приведенной выше суммы по i, с тем преимуществом, что зависимость t известна точно без временного шага.

Пример 14.10 (собственные функции лапласиана). На рисунке NUMBER показаны собственные векторы матрицы L из примера 14.7. Собственные векторы с низкими собственными значениями соответствуют низкочастотным функциям на [0, 1] со значениями, фиксированными на концах, и могут быть хорошими приближениями f (x) , когда он относительно гладкий.

Полностью дискретные методы В качестве альтернативы мы могли бы обращаться с переменными пространства и времени более демократично и дискретизировать их обе одновременно. Эта стратегия дает систему уравнений для решения, больше похожую на §14.5.1. Этот метод легко сформулировать, распараллелив эллиптический случай, но результирующие линейные системы уравнений могут быть большими, если зависимость между временными шагами имеет глобальный охват.

Пример 14.11 (Полностью дискретная диффузия тепла). Явный, неявный, Крэнк-Николсон. Не входит в CS 205A.

Важно отметить, что, в конце концов, даже полудискретные методы можно считать полностью дискретными в том смысле, что метод ОДУ с временным шагом по-прежнему дискретизирует переменную t; разница в основном заключается в классификации того, как были получены методы. Однако одним из преимуществ полудискретных методов является то, что они могут корректировать временной шаг для t в зависимости от текущей итерации, например, если объекты движутся быстро в физическом моделировании, может иметь смысл сделать больше временных шагов и разрешить это движение. Некоторые методы даже регулируют дискретизацию области значений x в случае, если требуется большее разрешение вблизи локальных разрывов или других артефактов.

14.6 Метод конечных элементов

Не входит в 205A.

14.7 Практические примеры

Вместо тщательного рассмотрения всех часто используемых методов PDE в этом разделе мы приводим примеры того, где они применяются на практике в компьютерных науках.

14.7.1 Обработка изображений в области градиента

14.7.2 Фильтрация с сохранением границ

14.7.3 Жидкости на основе сетки

14.8 Сделать

- Подробнее о существовании/уникальности
- Условия КЛЛ
- Теорема Лакса об эквивалентности
- Последовательность, стабильность и друзья

14.9 Проблемы

- Показать, что 1d лапласиан может быть факторизован как DD для первой производной матрицы D
- Решить УЧП первого порядка

Благодарности

[Обсуждение идет здесь]

Я очень благодарен студентам Стэнфордского университета CS 205A, осенью 2013 г., за обнаружение многочисленных опечаток и ошибок при написании этой книги. Ниже приводится, несомненно, неполный список студентов, внесших свой вклад в эту работу: Тао Ду, Леннарт Янссон, Майлз Джонсон, Люк Неппер, Минджэ Ли, Ниша Машарани, Джон Рейна, Уильям Сонг, Бен-Хан Сун, Мартина Троеш, Ожан. Тургут, Патрик Уорд, Чжунъёб Ё и Ян Чжао.

Особая благодарность Яну Хейланду и Тао Ду за помощь в разъяснении вывода алгоритма BFGS.

[Подробнее обсуждение здесь]